รายงานวิจัยฉบับสมบูรณ์

การถอดรหัสแบบหลายสแตคโดยใช้มัลติโพรเซสเซอร์

Multiprocessor-based Multiple Stack Decoding

โดย

ผศ.ดร.วิระสิทธิ์ อิ่มถวิล

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์

มหาวิทยาลัยขอนแก่น

30 มิถุนายน 2548

สัญญาเลขที่ MRG4680124

รายงานวิจัยฉบับสมบูรณ์

การถอดรหัสแบบหลายสแตคโดยใช้มัลติโพรเซสเซอร์

Multiprocessor-based Multiple Stack Decoding

ผู้วิจัย

ผศ.ดร.วิระสิทธิ์ อิ่มถวิล

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์

มหาวิทยาลัยขอนแก่น

นักวิจัยที่ปรึกษา

ศ.ดร.วัลลภ สุระกำพลธร

ภาควิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สนับสนุนโดยสำนักงานคณะกรรมการการอุดมศึกษา

และสำนักงานกองทุนสนับสนุนการวิจัย

ACKNOWLEDGEMENT

I would like to express my gratitude to my mentor, Professor Wanlop Surakampontorn, who provided me support and encouragement throughout this research project.

I would also like to thank my present and former master degree students in the Department of Electrical Engineering, Khon Kaen University, Monkol Kupimai, Pongpaiboon Boonchai, Teera Supanon, Jariya Ponsawad, and Assawanon Noisuwan for making an active research environment.

I would like to thank the Department of Electrical Engineering, Khon Kaen University, for supporting me in carrying out this research project.

I would like to acknowledge the financial support given to me by The Thailand Research Funds and The commission on Higher Education.

Last, but not least, my most sincere thanks go to my wife, Kanokwan Imtawil, my beloved son, Wuttikorn Imtawil, and my lovely daughter, Kanika Imtawil, for their patience and support.

VIRASIT IMTAWIL

บทคัดย่อ

รหัสโครงการ : MRG4680124

ชื่อโครงการ : การถอดรหัสแบบหลายสแตคโดยใช้มัลติโพรเซสเซอร์

ชื่อนักวิจัยและสถาบัน : ผศ.ดร. วิระสิทธิ์ อิ่มถวิล ผู้วิจัย

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น

E-mail address : virasit@kku.ac.th

ระยะเวลาโครงการ : 1 กรกฎาคม 2546 ถึง 30 มิถุนายน 2548

วิธีการถอดรหัสแบบหลายสแตคเป็นวิธีการถอดรหัสแบบซีเควนเชียล เพื่อใช้สำหรับถอดรหัสสัญญาณที่มีการเข้ารหัสแบบคอนโวลูชัน หลักการของวิธีการถอดรหัสแบบหลายสแตค คือ สแตคจะถูกแบ่งออกเป็น 2 ส่วนใหญ่ๆ คือ สแตคหลักและสแตครอง โดยที่มีสแตคหลักเพียงสแตคเดียวซึ่งมีขนาดใหญ่ และมีสแตครองอีกหลายสแตคซึ่งมีขนาดเล็ก กลไกของการถอดรหัสจะเริ่มจากการทำงานบนสแตคหลักเช่นเดียวกับการถอดรหัสแบบสแตคเดียว ในสภาวะที่มีการรบกวนจากสัญญาณรบกวนมาก บางครั้งการถอดรหัสไม่สามารถสิ้นสุดได้ที่สแตคหลัก จำเป็นต้องย้ายโหนดบางส่วนมาทำงานในสแตครอง จนกระทั่งการถอดรหัสสิ้นสุดลงในสแตครอง แต่การสิ้นสุดในสแตครอง ยังไม่ถือว่าเป็นการถอดรหัสที่มีความถูกต้องแม่นยำสูง เนื่องจากเส้นทางการถอดรหัสที่ได้อาจจะไม่ใช่เส้นทางที่ถูกต้อง ดังนั้นถ้าเวลาที่ตั้งไว้สำหรับการถอดรหัสยังไม่ถึงค่าลิมิต การถอดรหัสก็จะกลับมาดำเนินการที่สแตคหลัก เมื่อได้เส้นทางการถอดรหัสที่สแตคหลักก็จะนำเส้นทางนี้มาเปรียบเทียบกับเส้นทางเดิมที่ได้ และเลือกเส้นทางที่ดีกว่าเป็นเส้นทางการถอดรหัส อย่างไรก็ตามบางครั้งพบว่า การถอดรหัสมักจะถึงเวลาที่ตั้งไว้ก่อนที่จะกลับมาดำเนินการถอดรหัสในสแตคหลัก จึงทำให้ผลของการถอดรหัสมีประสิทธิภาพที่ไม่ดีนัก

งานวิจัยนี้เป็นการประยุกต์วิธีการถอดรหัสโดยใช้มัลติโพรเซสเซอร์ทำงานไปพร้อมๆกันในวิธีการถอดรหัสแบบหลายสแตค โดยการออกแบบการทำงานร่วมกันของมัลติโพรเซสเซอร์ที่เหมาะสม สามารถที่จะเลือกจำนวนของโพรเซสเซอร์ในการถอดรหัสได้ ผลการจำลองด้วยคอมพิวเตอร์ ปรากฏว่าวิธีการถอดรหัสแบบขนานโดยใช้มัลติโพรเซสเซอร์ในวิธีการถอดรหัสแบบหลายสแตค เพื่อถอดรหัสสัญญาณที่เข้ารหัสแบบคอนโวลูชันซึ่งมีค่าความยาวคอนสเตรนท์สูง ให้ค่าอัตราความผิดพลาดบิตลดลงอย่างมาก กรณีศึกษาได้ทำการทดสอบที่จำนวนโพรเซสเซอร์เป็น 4 และ 16

คำหลัก : การถอดรหัสแบบหลายสแตค การถอดรหัสแบบขนาน การเข้ารหัสแบบคอนโวลูชัน

ABSTRACT

Project Code : MRG4680124

Project Title : Multiprocessor-based Multiple Stack Algorithm

Investigator : Assistant Professor Dr. Virasit Imtawil
Department of Electrical Engineering,
Faculty of Engineering, Khon Kaen University

E-mail Address : virasit@kku.ac.th

Project Period : 1st July 2003 to 30th June 2005

The Multiple Stack Algorithm (MSA) is an algorithm for sequential decoding for convolutional codes. The main principle of the MSA is the stacks will be separated into 2 main groups; the main or primary stack and the secondary stacks or simply called sub-stacks. The main stack is made large and the sub-stacks are made small. The decoding starts in the main stack as the conventional single stack algorithm (SSA). Occasionally, in a very noisy case, the decoding is not done in the main stack, some top nodes will be transferred into a sub-stack. If the decoding is terminated in the sub-stack, the decoded path will be stored as a tentative decision. This tentative path may not be the correct path. If the computational limit is not reached, the decoding will be back to the main stack. The decoding then happens in the main stack again. If the decoding is terminated in the main (first) stack before the computational limit is reached, the new decoded path will be compared with the tentative path. The better one will be chosen the decoded path. However, it is often found that the decoding, in the noisy case, is terminated with the computational limit without being back to the main stack again. When this situation happens, the decoding error probability is usually very high.

In order to improve the error performance particularly in a very noisy block which requires excessive tree searches, a parallel decoding scheme for a multiple stack algorithm (MSA) is proposed in this paper. In this scheme, apart from the main decoder working on the main stack, a scalable set of multiple decoders working in parallel on their multiple stacks are incorporated. All the decoders are working almost independently with the help of the control processor, where the decoding process and the termination rules are outlined. Two cases, 4-processor MSA and 16-processor MSA, are employed as examples to investigate the performance of the proposed scheme. Comparing with the conventional MSA, extensive computer simulations show that the bit error probabilities (BER) are significantly improved.

Key words: The multiple stack algorithm, parallel decoding, convolutional codes

OUTPUT จากโครงการวิจัยที่ได้รับทุนจาก สกอ. และ สกว.

1. ผลงานตีพิมพ์ในวารสารวิชาการระดับนานาชาติ

   V. **Imtawil** and W. Surakampontorn, "Parallel Decoding Scheme for Multiple Stack Algorithm" **Accepted** to be published in **IEE Proceedings: Communications** (impact factor = 0.298)

2. การนำผลงานวิจัยไปใช้ประโยชน์

   2.1 เชิงวิชาการ ได้มีการนำเอาผลงานวิจัยไปใช้ในการเรียนการสอนวิชา Digital Communications ในระดับปริญญาตรี และ Error Control Coding ในระดับบัณฑิตศึกษา

   2.2 มีการสร้างและพัฒนานักวิจัยใหม่ในสาขาวิชาไฟฟ้าสื่อสารของประเทศไทย

3. การเสนอผลงานในที่ประชุมวิชาการ

   3.1 **V. Imtawil** and W. Surakampontorn, "A scalable multiprocessor-based multiple stack algorithm" บทคัดย่อใน: การประชุม "นักวิจัยรุ่นใหม่...พบ...เมธีวิจัยอาวุโส สกว." ณ โรงแรมเฟลิกซ์ ริเวอร์แคว จ.กาญจนบุรี มกราคม 2548

   3.2 **V. Imtawil** and W. Surakampontorn, "Parallel Decoding Scheme for Multiple Stack Algorithm" โปสเตอร์นำแสดงในงาน KKU ICT'2005 ณ หอประชุมกาญจนาภิเษก มหาวิทยาลัยขอนแก่น จ.ขอนแก่น มิถุนายน 2548

สารบัญ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคนิคการเข้ารหัสเพื่อแก้ไขความผิดพลาด (forward error correction, FEC) หรือเพื่อควบคุมความผิดพลาดเป็นเทคนิคการเข้ารหัสข้อมูลดิจิตอลเพื่อเพิ่มประสิทธิภาพในการป้องกันสัญญาณรบกวนที่เกิดขึ้นในช่องสื่อสาร สามารถแบ่งออกได้เป็น 2 หมวดหมู่ใหญ่ๆ คือ การเข้ารหัสแบบบล็อก (block coding) และการเข้ารหัสแบบคอนโวลูชัน (convolutional coding) รหัสคอนโวลูชันเป็นการเข้ารหัสเพื่อแก้ไขความผิดพลาดซึ่งถูกใช้แพร่หลายในระบบสื่อสารทั้งภาคพื้นดินและผ่านดาวเทียม วิธีการถอดรหัสสัญญาณข้อมูลที่เข้ารหัสแบบคอนโวลูชันมีอยู่ 2 วิธีใหญ่ๆ คือ การถอดรหัสแบบวิเทอบิ (Viterbi decoding) และการถอดรหัสแบบซีเควนเชียล (Sequential decoding) (Forney, 1974) งานวิจัยนี้จะเกี่ยวข้องกับการถอดรหัสแบบซีเควนเชียล ซึ่งมีข้อดีเหนือการถอดรหัสแบบวิเทอบิคือสามารถใช้กับรหัสคอนโวลูชันที่มีค่าความยาวคอนสเตรนท์สูงๆได้ เนื่องจากความซับซ้อนในการคำนวณไม่ขึ้นกับค่าความยาวคอนสเตรนท์ ในขณะที่การถอดรหัสแบบวิเทอบิมีค่าความซับซ้อนในการคำนวณเพิ่มขึ้นแบบเอ็กโพเนนเชียลกับค่าความยาวคอนสเตรนท์ (Lin และ Costello, 1983) และ (Haccoun และ Begin, 1989) ด้วยเหตุนี้จึงทำให้การถอดรหัสแบบซีเควนเชียลสามารถให้อัตราความผิดพลาดบิตที่ต่ำมากๆได้โดยการเพิ่มค่าความยาวคอนสเตรนท์ของตัวเข้ารหัส อย่างไรก็ตามข้อเสียของการถอดรหัสแบบซีเควนเชียลคือค่าความแปรปรวนในการคำนวณ (Computational variability) กล่าวคือบล็อกข้อมูลที่มีสัญญาณรบกวนน้อยจะใช้เวลาในการถอดรหัสเร็ว ส่วนบล็อกข้อมูลที่มีสัญญาณรบกวนมากจะใช้เวลาในการถอดรหัสช้าและบ่อยครั้งที่การถอดรหัสไม่สามารถสิ้นสุดภายในเวลาที่กำหนด ทำให้บล็อกข้อมูลถูกลบซึ่งเรียกว่า อีเรชัว (Erasure)

ในการทบทวนเอกสารที่เกี่ยวข้องกับการถอดรหัสแบบซีเควนเชียลพบว่า มีงานวิจัยออกมามากมายเพื่อลดปัญหาของอีเรชัว และค่าความแปรปรวนในการคำนวณ เช่นการถอดรหัสแบบฟาโน (Fano algorithm, FA) (Forney และ Bower, 1971) ซึ่งเป็นอัลกอริทึมหนึ่งของการถอดรหัสแบบซีเควนเชียลพบพบว่าทำให้สามารถลดผลของอีเรชัว และค่าความแปรปรวนในการคำนวณได้ แต่ปัญหาก็คือการลดอีเรชัวโดยวิธีการถอดรหัสแบบฟาโนส่งผลให้อัตราความผิดพลาดบิตเพิ่มขึ้นโดยเฉลี่ย เนื่องจากการตั้งค่า backsearch limit อาจทำให้ทางเดินที่ถูกต้องในแผนภูมิต้นไม้แทนรหัส (code tree) ถูกลบไปก่อนที่ขบวนการถอดรหัสจะสิ้นสุดลง ต่อมาสถาปัตยกรรมของการถอดรหัสแบบหลายตัวประมวลผล (multiprocessor decoding system architecture) (Belanger et al.,1994) โดยใช้หลักการให้ตัวประมวลผลทำงานบนหลายๆทางเดินในต้นไม้แทนรหัสพร้อมๆกันด้วยโครงสร้างการทำงานที่แน่นอนของการถอดรหัสแบบสแตค The Single Stack Algorithm, SSA ผลที่ได้ในงานวิจัยนี้สามารถที่จะเปลี่ยนจำนวนของตัวประมวลผล

ตามที่ต้องการได้และการเพิ่มตัวประมวลผลก็จะได้ค่าความแปรปรวนในการคำนวณ ที่ลดลง แต่อย่างไรก็ตาม อีเรชัวก็ยังไม่หมดไป

อัลกอริทึมของการถอดรหัสแบบซีเควนเชียลวิธีการหนึ่งซึ่งเป็น Erasure-free Sequential Decoding Algorithm โดยใช้หลักการของหลายสแตคเรียกว่า The Multiple Stack Algorithm (MSA) (Chevillat และ Costello,1976) ในการถอดรหัสแบบ MSA จะอาศัยหลักการคล้ายๆกับ The Single Stack Algorithm, SSA (Jelinek, 1969) ต่างกันตรงที่จำนวนสแตคที่ใช้ใน MSA มีมากกว่าหนึ่งสแตคและการทำงานของตัวถอดรหัสจะกำหนดขนาดของสแตคออกเป็น 2 ส่วนคือ สแตคหลักหรือสแตคแรกและกลุ่มของสแตครองหรือสแตคอันดับสูงซึ่งจะมีหลายสแตค โดยที่ขนาดของสแตคหลักจะโตกว่าขนาดของสแตครองมาก ในกรณีปกติของการถอดรหัสที่มีสัญญาณรบกวนน้อย การทำงานของขบวนการถอดรหัสมักจะสิ้นสุดที่สแตคหลัก จะมีเพียงบางครั้งเท่านั้นที่สแตครองจะถูกนำมาใช้ นั่นคือในกรณีที่ช่องสื่อสารมีสัญญาณรบกวนมากซึ่งมักจะทำให้เกิดอีเรชัว ในกรณีนี้การทำงานของ MSA จะไม่สิ้นสุดจนกว่าค่าของการคำนวณที่ใช้ในการถอดรหัสจะถึงค่าๆหนึ่งซึ่งถูกตั้งไว้ก่อนหน้า โดยหลักการเช่นนี้ทำให้ MSA เป็นวิธีการถอดรหัสแบบ ซีเควนเชียลที่สามารถถอดรหัสได้สมบูรณ์ (complete decoding) โดยไม่เกิดอีเรชัว เช่นเดียวกับการถอดรหัสแบบ VA อย่างไรก็ตามข้อด้อยของ MSA ก็คือประสิทธิภาพในอัตราความผิดพลาดบิตยังมีค่าสูงกว่าการถอดรหัสแบบ VA นอกจากนั้นแล้วความต้องการของหน่วยความจำใน MSA ก็ยังมีมากในกรณีที่ต้องการประสิทธิภาพในอัตราความผิดพลาดบิตต่ำ ซึ่งต่อมาได้มีการพัฒนาวิธีการปรับปรุงประสิทธิภาพของ MSA ในการใช้หน่วยความจำที่มีประสิทธิภาพมากขึ้นโดยเฉพาะในกลุ่มของสแตครองและให้ชื่ออัลกอริทึมใหม่ว่า The modified MSA (Y. Lin และ S.H. Tu,1997) ในวิธีการนี้ใช้วิธีการจัดสแตครองในรูปแบบวงแหวนเสมือน (Ring-like structure) ซึ่งการจัดหน่วยความจำลักษณะนี้ทำให้สามารถที่จะกำหนดขนาดของสแตครองและจำนวนโนดที่ย้ายระหว่างสแตคให้มีค่าสูงได้แต่ปัญหาที่ตามมาของงานวิจัยลักษณะนี้คือการได้ทางเดินของการถอดรหัส (Decoded path) อาจจะใช้เวลานานและส่งผลถึงการเกิดโอเวอร์โฟล์ (Overflow) ที่บัฟเฟอร์ด้านอินพุทได้ ต่อมา Kaiping Li และ Samir Kallel, (1999) ได้เสนอรูปแบบใหม่ของการลดอัตราความผิดพลาดบิตใน MSA โดยใช้หลักการของการถอดรหัสสองทิศทางและให้ชื่อว่า The bi-directional MSA ซึ่งให้อัตราความผิดพลาดบิตที่น่าสนใจเมื่อเปรียบเทียบกับ VA แต่อย่างไรก็ตามก็ยังไม่พบงานวิจัยใดที่นำเอาหลักการของตัวประมวลผลแบบหลายโหนดกับ MSA และศึกษาโครงสร้างของสถาปัตยกรรมที่เหมาะสมสำหรับการถอดรหัสแบบ MSA ในรูปแบบของ multiprocessor architecture ซึ่งน่าจะให้ประสิทธิภาพในอัตราความผิดพลาดบิตที่ดีขึ้นกว่าการถอดรหัสแบบ MSA เดิม

ในงานวิจัยนี้เรานำเสนอวิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของขบวนการถอดรหัสที่ใช้ MSA ผลการทดสอบประสิทธิภาพเมื่อจำลองระบบด้วยคอมพิวเตอร์ ปรากฏว่าอัตราความผิดบิตใน MSA มีค่าลดลงอย่างมากเมื่อใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ แนวคิดวิธีการประยุกต์ใช้การถอดรหัสแบบ MSA โดยมัลติ

โพรเซสเซอร์ สำหรับถอดรหัสคอนโวลูชันเป็นอีกทางเลือกหนึ่งที่น่าสนใจโดยเฉพาะอย่างยิ่งในระบบที่ต้องการค่าอัตราความผิดพลาดบิตที่ต่ำๆ

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อพัฒนาประสิทธิภาพของการถอดรหัสแบบหลายสแตค (The Multiple Stack Algorithm)

1.2.2 เพื่อหาวิธีการประยุกต์ใช้มัลติโพรเซสเซอร์ในการถอดรหัสแบบซีเควนเชียลเพื่อลดปัญหา computational variability ซึ่งทำให้เกิด erasure ขึ้น

1.2.3 เพื่อศึกษาและออกแบบโครงสร้างของสถาปัตยกรรมมัลติโพรเซสเซอร์ที่เหมาะสมกับการถอดรหัสแบบหลายสแตค

1.2.4 เพื่อศึกษาและออกแบบการเข้ารหัสแบบอื่นๆที่เหมาะสมกับการใช้มัลติโพรเซสเซอร์ในการถอดรหัสแบบหลายสแตค

1.2.5 เพื่อเป็นการเผยแพร่วิชาการและองค์ความรู้ในเชิงการวิจัยของการถอดรหัสโดยใช้มัลติโพรเซสเซอร์

1.2.6 เพื่อพัฒนาศักยภาพในการทำงานวิจัยของผู้ทำวิจัยซึ่งเป็นอาจารย์ในสถาบันอุดมศึกษา

1.3 ขอบเขตของงานวิจัย

1.3.1 งานวิจัยนี้จะเป็นการออกแบบการประยุกต์ใช้มัลติโพรเซสเซอร์แบบต่างๆในการถอดรหัสแบบหลายสแตคเพื่อเพิ่มประสิทธิภาพการทำงานของการถอดรหัสแบบหลายสแตคโดยวิธีการจำลองในโปรแกรมซึ่งเขียนโดยใช้ภาษาซี ทำงานบนระบบปฏิบัติการณ์ยูนิกส์

1.3.2 วิเคราะห์ประสิทธิภาพการทำงานของการถอดรหัสแบบหลายสแตคเมื่อใช้มัลติโพรเซสเซอร์เพื่อเปรียบเทียบกับการถอดรหัสแบบหลายสแตคเมื่อใช้โพรเซสเซอร์เดียวในช่องสื่อสารแบบ AWGN

1.4.1 ศึกษาและออกแบบการประยุกต์ใช้การถอดรหัสแบบหลายสแตคในรหัสแบบรีเคอร์ซีพซิสเต็มเมติกคอนโวลูชัน

1.4 ผลงานวิจัยที่คาดว่าจะได้รับ

1.4.2 ได้ผลการเปรียบเทียบประสิทธิภาพการถอดรหัสแบบหลายสแตคเมื่อใช้มัลติโพรเซสเซอร์กับการถอดรหัสแบบหลายสแตคที่ใช้โพรเซสเซอร์เดียว

1.4.3 ได้ทางเลือกใหม่ในการถอดรหัสแบบซีเควนเชียล นั่นคือการถอดรหัสแบบหลายสแตคเมื่อใช้มัลติโพรเซสเซอร์ ซึ่งให้ประสิทธิภาพในอัตราความผิดพลาดบิตต่ำกว่าการถอดรหัสแบบหลายสแตคเมื่อใช้โพรเซสเซอร์ตัวเดียว

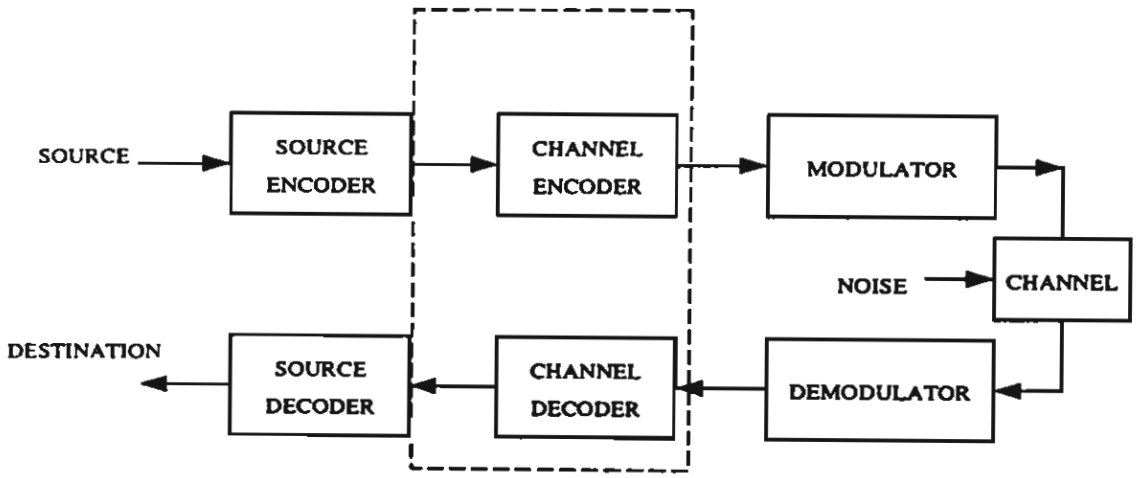
1.4.4 ได้วิธีการประยุกต์ใช้วิธีการถอดรหัสแบบหลายสเตตในรหัสแบบรีเดอร์ซีพซิสเต็มแมติกคอนโวลูชัน

1.4.5 ได้พัฒนาศักยภาพในการทำงานวิจัยของผู้วิจัย

บทที่ 2
การถอดรหัสแบบหลายสเตค

2.1 บทนำ

ในการส่งและรับข้อมูลดิจิตอลผ่านช่องสื่อสารหนึ่งๆปัญหาที่ไม่สามารถจะหลีกเลี่ยงได้คือการถูก รบกวนของสัญญาณ ซึ่งมาจากทั้งตัวอุปกรณ์ในระบบและจากภายนอก และเป็นสาเหตุที่ทำให้สัญญาณที่รับได้มีความผิดเพี้ยนจากสัญญาณเดิม หรือทำให้สัญญาณข้อมูลมีคุณภาพต่ำลง ดังนั้นเพื่อให้ได้สัญญาณที่มีคุณภาพดีที่สุดจึงได้มีการค้นคว้าหาวิธีเพื่อให้ได้มาซึ่งสัญญาณข้อมูลที่ถูกต้องมากที่สุด วิธีการหนึ่งที่จะเพิ่มประสิทธิภาพของระบบสื่อสารเพื่อให้ได้คุณภาพของสัญญาณที่เครื่องรับดีขึ้นคือการใช้เทคนิคการเข้ารหัสควบคุมการ ผิดพลาด (Error Control Coding, ECC) หรือบางครั้งก็เรียกว่าการเข้ารหัสช่องสื่อสาร (Channel Coding) ในวิธีการนี้มีหลักการเบื้องต้นก็คือการเพิ่มส่วนซ้ำซ้อน (redundancy) ให้กับข้อมูลดิจิตอลที่จะส่งผ่านช่องสื่อสารซึ่งปริมาณของส่วนซ้ำซ้อนที่ใส่เข้าไปจะสัมพันธ์กับระดับในการป้องกันสัญญาณรบกวนของสัญญาณรหัส (Coded signal)



ภาพที่ 2.1 บล็อกไดอะแกรมของระบบสื่อสารเชิงดิจิตอล

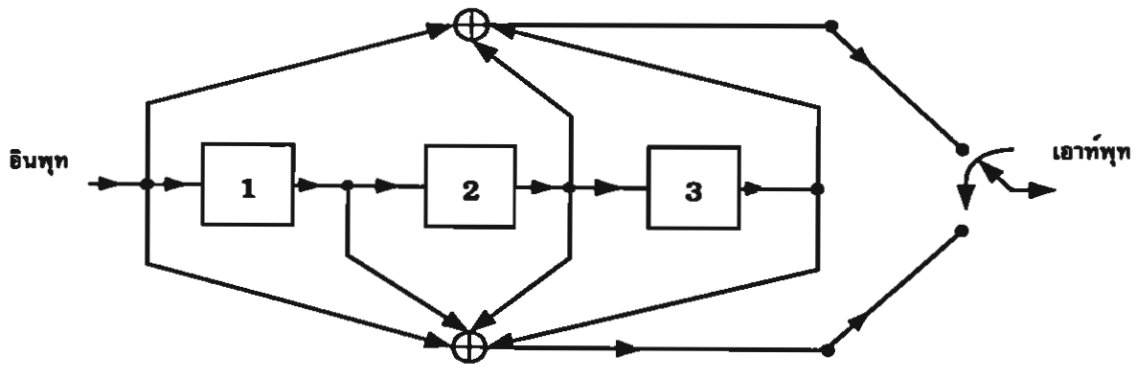
ภาพที่ 2.1 เป็นบล็อกไดอะแกรมของระบบสื่อสารเชิงดิจิตอลขั้นพื้นฐานโดยทั่วไป สำหรับงานวิจัยนี้ที่ทำการศึกษาจะเกี่ยวข้องกับตัวเข้ารหัสช่องสื่อสาร (Channel encoder) ซึ่งต่อไปจะเรียกสั้น ๆว่าตัวเข้ารหัสและตัวถอดรหัสช่องสื่อสาร (Channel decoder) ซึ่งต่อไปจะเรียกสั้น ๆว่าตัวถอดรหัส จากภาพที่ 2.1 ข้อมูลดิจิตอลที่ต้องการจะส่งถูกป้อนเข้าทางด้านอินพุทของตัวเข้ารหัสซึ่งจะทำหน้าที่เข้ารหัส

ข้อมูลโดยการเพิ่มส่วนซ้ำซ้อนให้กับข้อมูล เปรียบเสมือนกับเป็นการเพิ่มภูมิคุ้มกันสัญญาณรบกวนก่อนส่งผ่านช่องสื่อสารไปยังภาครับสัญญาณ ในส่วนของภาครับก็จะทำหน้าที่รับสัญญาณและกำจัดสัญญาณที่มีความผิดเพี้ยนเนื่องจากสัญญาณ ถูกรบกวน โดยนำสัญญาณที่รับได้มาผ่านขบวนการการถอดรหัสโดยตัวถอดรหัส (Decoder) ซึ่งจะทำหน้าที่ดึงเฉพาะสัญญาณข้อมูลดิจิตอลออกมาโดยการแยกส่วนซ้ำซ้อนออกเพื่อให้ได้ข้อมูลที่มีความเหมือนหรือใกล้เคียงกับข้อมูลที่ส่งมาซึ่งกระบวนการเข้ารหัสสัญญาณและการถอดรหัสสัญญาณจะได้กล่าวถึงในหัวข้อถัดไป

2.2 การเข้ารหัสแบบคอนโวลูชัน

เทคนิคการเข้ารหัสข้อมูลดิจิตอลสามารถแบ่งออกได้เป็น 2 หมวดหมู่ใหญ่ๆ คือ การเข้ารหัสแบบบล็อก (Block coding) และการเข้ารหัสแบบคอนโวลูชัน (Convolutional coding) ซึ่งการเข้ารหัสทั้งสองแบบ ก็จะมีขบวนการการถอดรหัสที่เหมาะสมแตกต่างกัน โดยงานวิจัยนี้จะเกี่ยวข้องกับการเข้ารหัสแบบคอนโวลูชันและการถอดรหัสแบบซีเควนเชียลโดยอัลกอริทึมแบบหลายสแตคเท่านั้น

การเข้ารหัสแบบคอนโวลูชันเป็นการเข้ารหัสที่มีการใช้หน่วยความจำสำหรับจำสถานะของบิตข้อมูลซึ่งจะแตกต่างจากการเข้ารหัสแบบบล็อก นั่นหมายความว่าสัญญาณรหัส (Code word) w บิต ณ เวลาหนึ่งๆจะ ขึ้นอยู่กับสัญญาณอินพุท (message) k บิต ณ เวลานั้นกับสัญญาณอินพุท mk บิต ก่อนหน้านั้น m สัญญาณ เมื่อ m หมายถึงอันดับของหน่วยความจำของตัวเข้ารหัส ซึ่งแตกต่างจากการเข้ารหัสแบบบล็อกคือ ในการเข้ารหัสแบบบล็อก สัญญาณรหัส m บิตจะขึ้นอยู่กับสัญญาณอินพุท k บิต ณ เวลานั้นเท่านั้น



ภาพที่ 2.2 ตัวเข้ารหัสแบบคอนโวลูชัน มี $R = 1/2$, $m = 3$

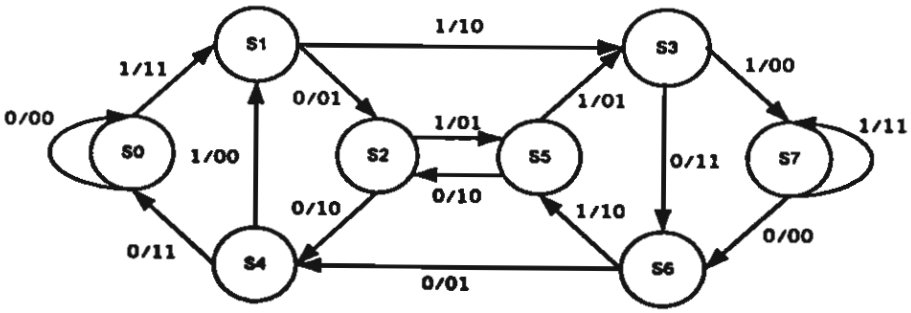
ภาพที่ 2.2 แสดงการเข้ารหัสแบบคอนโวลูชันโดยมีอัตราการเข้ารหัสเท่ากับ 1/2 และมีจำนวนหน่วยความจำเท่ากับ 3 (m = 3) อัตราการเข้ารหัส (Code rate, R) ของตัวเข้ารหัสจะถูกกำหนดด้วย

จำนวนบิตด้านอินพุท (k) หารด้วยจำนวนบิตด้านเอ้าท์พุท (w) เช่นจากภาพที่ 2.2 มีค่า $k = 1$ และ $w = 2$ ดังนั้น $R = 0.5$ เป็นต้น

การเข้ารหัสแบบคอนโวลูชันสามารถแทนด้วยแผนภูมิแบบต่าง ๆ ได้ 3 รูปแบบ คือ แผนภูมิสเตท (State diagram) แผนภูมิเทรลลิส (Trellis diagram) และ แผนภูมิต้นไม้ (Tree diagram)

แผนภูมิสเตท

จากตัวเข้ารหัสแบบคอนโวลูชันในภาพที่ 2.2 ซึ่งมีอันดับของหน่วยความจำ $m = 3$ สเตท ณ เวลาใดเวลาหนึ่งของตัวเข้ารหัสหมายถึงบิตที่ตำแหน่ง 1, 2 และ 3 ในภาพที่ 2.2 จำนวนสเตทที่เป็นไปได้ทั้งหมดของการเข้ารหัสนี้มีค่าเท่ากับ $2^m = 8$ สเตท ในกรณีนี้คือ ($s_0 = 000$), ($s_1 = 100$), ($s_2 = 010$), ($s_3 = 110$), ($s_4 = 001$), ($s_5 = 101$), ($s_6 = 011$), ($s_7 = 111$) เมื่อนำมาเขียนแผนภูมิจะได้ดังภาพ ที่ 2.3 ซึ่งการเข้ารหัสจะเริ่มต้นที่สเตทแรก ($s_0 = 000$) ตัวเลขกำกับและเส้นลูกศรที่แสดงในภาพที่ 2.3 แสดงถึงค่าที่ได้จากการเข้ารหัสและเส้นทางการเปลี่ยนสเตทเมื่อมีบิตอินพุทเข้ามา ณ สเตทนั้น ๆ เช่นขณะที่สเตทของการถอดรหัสอยู่ที่สเตทศูนย์เมื่อบิตอินพุทเข้ามา เป็น "1" สเตทจะเปลี่ยนไปที่สเตทหนึ่งและได้ผลของการเข้ารหัสคือ "11" ถ้าบิตอินพุทเข้ามาเป็น "0" สเตทจะยังคงเป็นสเตทศูนย์เช่นเดิมและได้ผลของการเข้ารหัสเป็น "00" เป็นต้น ในภาพที่ 2.3 แสดงสเตทของการเข้ารหัสเมื่อลำดับของสัญญาณอินพุทเป็น 1100 จะได้ลำดับของสเตจคือ $S_0\ S_1\ S_3\ S_6\ S_4\ S_0\ S_0$ ตามลำดับ และได้ข้อมูลรหัสคือ 11 10 11 01 11 00 00

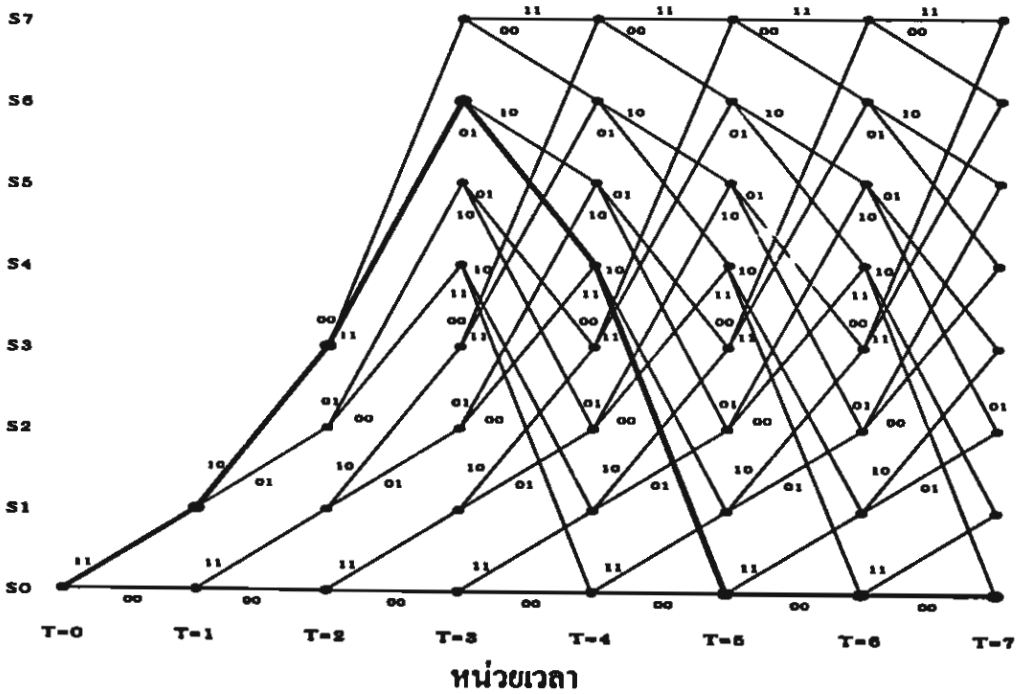


ภาพที่ 2.3 แผนภูมิสเตทของการเข้ารหัสแบบคอนโวลูชัน มี $R = 1/2$, $m = 3$

แผนภูมิเทรลลิส

แผนภูมิเทรลลิสเป็นแผนภูมิแสดงผลจากการเข้ารหัสและเส้นทางการเปลี่ยนสเตทในแต่ละช่วงเวลา ช่วงเวลาทั้งหมดที่ใช้ในการเข้ารหัสซึ่งจะเริ่มตั้งแต่สเตทศูนย์และกลับมาที่สเตทศูนย์อีกครั้งจะเท่ากับ $L + m$ ช่วงเวลา โดย L คือจำนวนของบิตข้อมูลในแต่ละบล็อก ในแผนภูมิเทรลลิสจะมีตัวเลขกำกับ

บนเส้นทางการเปลี่ยนสเตทในแต่ละช่วงเวลาเพื่อบอกผลของการเข้ารหัสเมื่อบิตอินพุทเข้ามา ณ สเตทใด ๆ ในช่วงเวลานั้น โดยเส้นกิ่งบนที่ออกจากสเตทแสดงผลจากการเข้ารหัสและเส้นทางการเปลี่ยนสเตทเมื่อบิตอินพุทที่เข้ามาสู่สเตท ณ เวลานั้นเป็น "1" เส้นกิ่งล่างที่ออกจากสเตทแสดงแสดงผลจากการเข้ารหัสและเส้นทางการเปลี่ยน สเตทเมื่อบิตอินพุทที่เข้ามาสู่สเตท ณ เวลานั้นเป็น "0" โดยทั่วไปแล้วจำนวนกิ่งที่ออกจากสเตทแต่ละสเตทในช่วงเวลาหนึ่งจะเท่ากับ 2^k กิ่ง และจะมีเส้นทางการเข้ารหัสจากสเตทเริ่มต้นจนถึงสเตทสุดท้ายทั้งหมดเท่ากับ 2^{kL} เส้นทาง โดยมีความยาวของรหัสสัญญาณที่ได้จากการเข้ารหัสเท่ากับ $n = w(L+m)$ บิต ตัวอย่าง เช่น ถ้ามีลำดับข้อมูล 1100 ถูกเข้ารหัสแบบคอนโวลูชันในภาพที่ 2.2 ซึ่งมีจำนวนหน่วยความจำเท่ากับ 3 ดังนั้นจะมีจำนวนครั้งของการเปลี่ยนสเตจเท่ากับ 7 ครั้ง และได้ข้อมูลรหัสคือ 11 10 11 01 11 00 00 โดยมีจำนวนบิตทั้งหมดเท่ากับ 14 บิต ซึ่งมีเส้นทางการเข้ารหัส(เส้นทึบ)แสดงในแผนภูมิเทรลลิสดังภาพที่ 2.4



ภาพที่ 2.4 แผนภูมิเทรลลิสของการเข้ารหัสแบบคอนโวลูชัน (2,13) และ $L=4$,

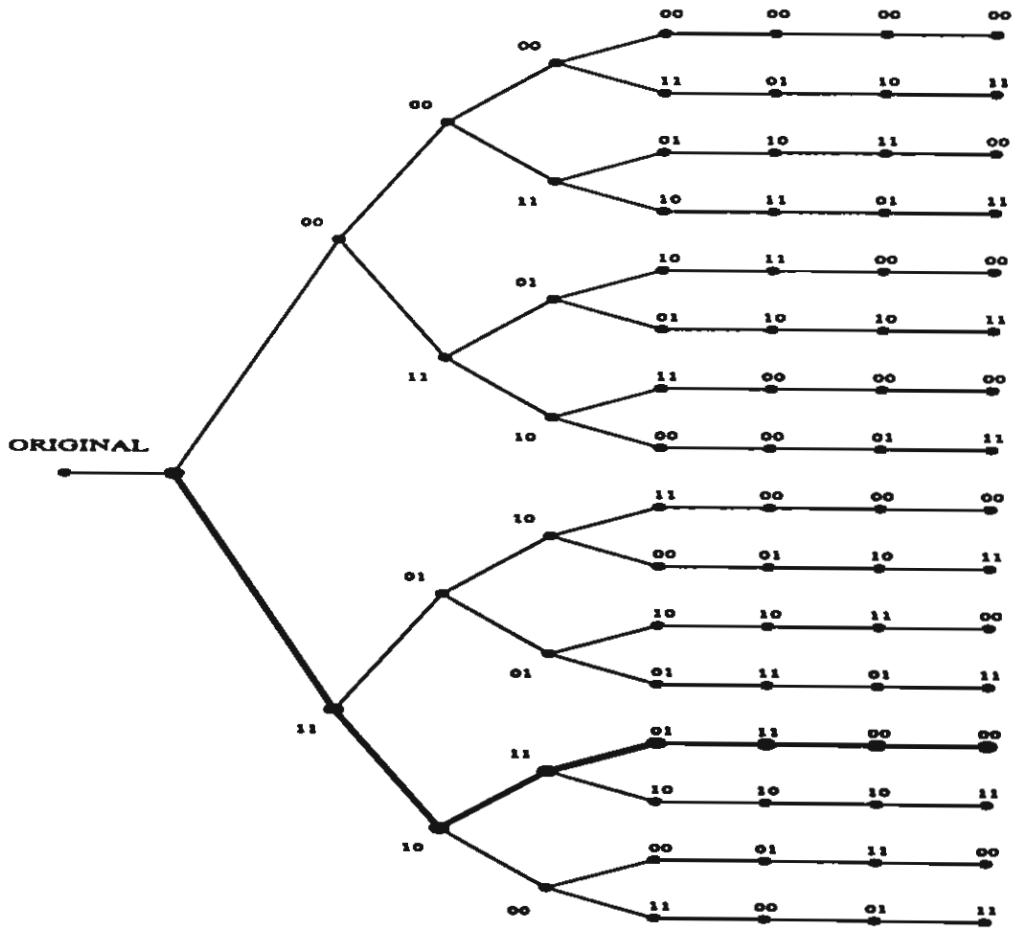
แผนภูมิต้นไม้

แผนภูมิต้นไม้แทนรหัสใช้สำหรับแสดงเส้นทาง (Path) ของการเข้ารหัสแบบคอนโวลูชั่นซึ่งจะมีเส้นทางที่เป็นไปได้จากจุดเริ่มต้นจนถึงจุดสุดท้ายทั้งหมดจะเท่ากับ 2^{kL} เส้นทาง โดยที่ L คือความยาวของลำดับข้อมูลอินพุทในแต่ละบล็อก โดยที่แต่ละโนดบนเส้นทางเดินจะมีทางออกจากโนดสองเส้นทางในภาพที่ 2.5 แสดงแผนภูมิต้นไม้แทนรหัสใช้สำหรับการเข้ารหัสแบบคอนโวลูชันในภาพที่ 2.2 โดยมี $R=1/2$, $m=3$ และมีลำดับของข้อมูลอินพุทคือ 1100 โดยกำหนดให้เส้นที่ชี้ขึ้นที่ออกจากแต่ละโนด

หมายถึงมีอินพุท เข้ามาเป็น "1" และให้เส้นที่ชี้ลงหมายถึงมีอินพุท เป็น "0" ดังนั้นได้ข้อมูลที่ถูกเข้ารหัส คือ 11 10 11 01 11 00 00 (เส้นทึบ)

2.3 การถอดรหัสสัญญาณที่เข้ารหัสแบบคอนโวลูชัน

การถอดรหัสสัญญาณข้อมูลเป็นวิธีการนำสัญญาณข้อมูลที่ได้รับทางด้านรับมาผ่านขบวนการถอดรหัสเพื่อให้ได้สัญญาณข้อมูลที่ถูกส่งมาอย่างถูกต้องมากที่สุดกระบวนการถอดรหัสสัญญาณที่เข้ารหัสแบบ คอนโวลูชัน สามารถที่จะถูกถอดรหัสได้ 2 วิธีใหญ่ๆ คือ การถอดรหัสแบบวิเทอบิ (Viterbi algorithm, VA) และการถอดรหัสแบบซีเควนเชียล (sequential decoding, SD) โดยจะกล่าวถึงรายละเอียดในลำดับต่อไป



ภาพที่ 2.5 แผนภูมิต้นไม้ ของการเข้ารหัสแบบคอนโวลูชัน มี $R = 1/2$, $m = 3$

2.3.1 การถอดรหัสแบบวิเทอบิ (Viterbi algorithm, VA)

การถอดรหัสแบบวิเทอบิจะใช้หลักการของความคล้ายกันที่สุดของสัญญาณโดยคำนวณจากระยะแฮมมิ่งหรือระยะยูคลิเดียนทางสัญญาณที่รับได้กับสัญญาณรหัสซึ่งจะคำนวณบนแผนภูมิเทรลลิส ในการส่งสัญญาณผ่านช่องสื่อสารแบบ Binary Symmetric Channel, BSC ถ้าให้สัญญาณที่รับได้ r และให้สัญญาณรหัสที่ถูกส่ง v บนเส้นทางในแต่ละช่วงบนแผนภูมิเทรลลิส โดยเรียกค่าเมตริกซ์ที่คำนวณในแต่ละช่วงเวลาบนแผนภูมิเทรลลิสว่า Bit Metrics, $M(r_i^{(J)}/v_i^{(J)})$ และมีสมการการคำนวณค่าเมตริกซ์ดังสมการที่ 2.1 โดยที่ $i=0,1,2...,(L+m-1)$ และ $j=0,1,2...,(n-1)$ และเรียกค่าเมตริกซ์ที่คำนวณได้ทั้งหมดตลอดเส้นทางจากช่วงเวลาเริ่มต้นจนถึงช่วงเวลาสุดท้ายบนเส้นทางบนแผนภูมิเทรลลิสว่า Path Metric, $M(r/v)$ มีสมการการคำนวณดังสมการที่ 2.2 Stephen B. Wicker. (1995)

$$M(r_i^{(J)}/v_i^{(J)}) = \frac{1}{[\log_2(1-p)-\log_2(p)]}[\log_2 P(r_i^{(J)}/v_i^{(J)}) - \log_2(1-p)] \qquad (2.1)$$

$$M(r/v) = \sum_{i=0}^{L+m-1}\left(\sum_{j=0}^{n-1} M(r_i^{(J)}/v)_i^{(J)}\right) \qquad (2.2)$$

โดย p (crossover probability) เป็นความน่าจะเป็นของความผิดพลาดของสัญญาณระหว่างสัญญาณรหัสที่ถูกส่งและสัญญาณที่รับได้ในช่องสื่อสาร

การถอดแบบวิเทอบิที่มีการส่งสัญญาณผ่านช่องสื่อสารแบบ BSC การคำนวณค่าบิตเมตริกซ์สามารถเขียนได้ในรูปแบบตารางโดยใช้ระยะแฮมมิ่งระหว่างระหว่างสัญญาณรหัสที่ถูกส่งและสัญญาณที่รับได้ดังแสดงในตารางที่ 2.1 ตัวอย่างการถอดรหัสแบบวิเทอบิ โดยกำหนดสัญญาณที่รับได้เป็น 11 1$\underline{1}$ 11 01 11 00 $\underline{1}$0 โดยใช้แผนภูมิเทรลลิสในภาพที่ 2.4 และใช้ค่าเมตริกซ์จากตารางที่ 2.1 เมื่อคำนวณค่าเมตริกซ์ของทุกเส้นทางบนแผนภูมิเทรลลิสแล้วจะเลือกเส้นทางที่มีค่าเมตริกซ์สูงที่สุดเป็นเส้นทางการถอดรหัสและได้ข้อมูลของการถอดรหัสคือ 11 10 11 01 11 00 00 ดังแสดงในภาพที่2.6 (ล่างสุด) ซึ่งก็คือสัญญาณอินพุท 1100

ตารางที่ 2.1

ค่าเมตริกซ์ของแต่ละบิตสำหรับการถอดรหัสแบบวิเทอบิ

$M(r_i^{(J)}/v_i^{(J)})$	$r_i^{(J)}=0$	$r_i^{(J)}=1$
$v_i^{(J)}=0$	1	0
$v_i^{(J)}=1$	0	1

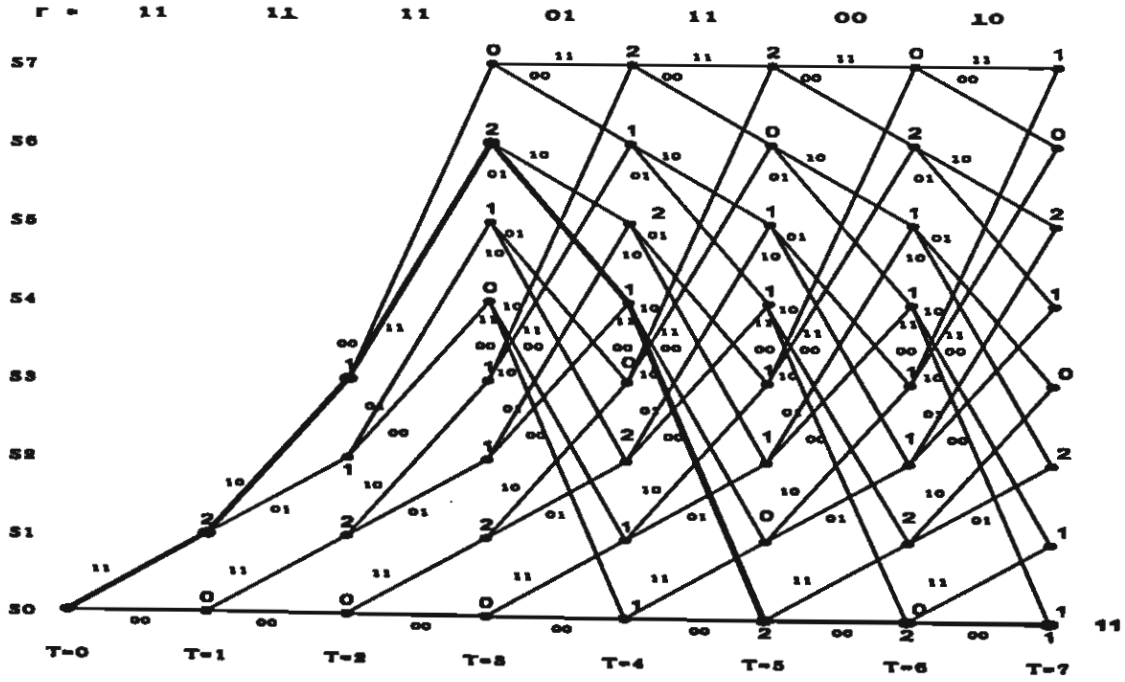
2.3.2 การถอดรหัสแบบซีเควนเชียล (Sequential decoding, SD)

การถอดรหัสแบบซีเควนเชียลเป็นวิธีการถอดรหัสสัญญาณที่เข้ารหัสแบบคอนโวลูชันที่มีลักษณะคล้ายกันกับการถอดรหัสแบบวิเทอบิคือมีการคำนวณค่าบิตเมตริกซ์ของสัญญาณที่รับได้กับสัญญาณรหัสบนแผนภูมิของรหัสแบบคอนโวลูชัน แต่การถอดรหัสแบบซีเควนเชียลจะคำนวณค่าบิตเมตริกซ์ของเส้นทางเดินบนแผนภูมิต้นไม้ โดยคำนวณค่าบิตเมตริกซ์จากสมการการคำนวณค่าบิตเมตริกซ์ที่เรียกว่า ฟาโนเมตริกซ์ (Fano metrics) Fano (1963) แสดงในสมการที่ 2.3 สำหรับการคำนวณค่าบิตเมตริกซ์ของสัญญาณที่ส่งสัญญาณผ่านช่องสื่อสารแบบ BSC ที่มีความน่าจะเป็นของความผิดพลาดของสัญญาณ p สมการเมตริกซ์ของฟาโนจะสามารถเขียนได้ใหม่ดังสมการที่ 2.4

$$M(r_i^{(j)} / v_i^{(j)}) = \log_2\left[\frac{P(r_i^{(j)} / v_i^{(j)})}{P(r_i^{(j)})}\right] - R \qquad (2.3)$$

$$M(r_i^{(j)} / v_i^{(j)}) = \begin{cases} \log_2 2p - R & \text{if} \quad r_i^{(j)} \neq v_i^{(j)} \\ \log_2 2(1-p) - R & \text{if} \quad r_i^{(j)} = v_i^{(j)} \end{cases} \qquad (2.4)$$

โดย R คืออัตราการเข้ารหัสของสัญญาณ



ภาพที่ 2.6 ตัวอย่างแผนภูมิเทรลลิสของการถอดรหัสสแบบวิเทอบิ

ในงานวิจัยนี้ได้ศึกษาวิธีการถอดรหัสแบบซีเควนเชียลที่มีการถอดรหัสโดยใช้สแตค โดยแบ่งเป็นสอง อัลกอริทึมตามจำนวนสแตคที่ใช้ในการถอดรหัสคือ การถอดรหัสโดยใช้สแตคเดียวหรือที่เรียกโดยทั่วไปว่า สแตคอัลกอริทึม (Single Stack Algorithm, SSA) และการถอดรหัสโดยใช้หลายสแตคที่เรียกโดยทั่วไปว่าการถอดรหัสแบบหลายสแตค (Multiple Stack Algorithm, MSA) ซึ่งการถอดรหัสทั้งสองแบบจะได้กล่าวในรายละเอียดต่อไป

2.3.2.1 การถอดรหัสแบบสแตคเดียว

การถอดรหัสแบบสแตคเดียวเป็นการถอดรหัสแบบซีเควนเชียล โดยมีหลักการคือ สัญญาณที่รับได้จากช่องสื่อสารจะถูกนำมาคำนวณค่าเมตริกซ์ซึ่งมีสัญญาณรหัสบนแผนภูมิต้นไม้เป็นตัวอ้างอิง ขบวนการถอดรหัสแบบสแตคเดียวมีลำดับการทำงานดังแสดงในภาพที่ 2.8 ซึ่งสามารถอธิบายได้ดังนี้คือ ขั้นตอนการถอดรหัสเริ่มต้นที่ $P = root$ ซึ่งเป็นโนดเริ่มต้นสำหรับการถอดรหัสแบบสแตคเดียวบนแผนภูมิต้นไม้ ขั้นตอนการถอดรหัสต่อไปจะคำนวณหาค่าเมตริกซ์ของเส้นทางที่อยู่ถัดไปจาก P ไป 1 ระดับ (Compute Metrics of successors of top path) ทั้งหมด 2^k เส้นทาง(ในงานวิจัยนี้ 2 เส้นทาง) แล้วเรียงเส้นทางที่มีค่าเมตริกซ์มากให้อยู่ด้านบน ขั้นตอนต่อไปให้ตัดเส้นทางที่มีค่าเมตริกซ์มากที่สุดออก (Delete top path from stack) และให้ P เท่ากับเส้นทางที่ถูกตัดออกไปสำหรับคำนวณค่าเมตริกซ์ในเส้นทางใหม่แล้วเก็บค่าเส้นทางที่เหลือไว้ใน สแต็ค (Reorder stack according to metric values) ในขั้นตอนต่อไปให้ตรวจสอบว่าเส้นทางที่มีตำแหน่งสูงที่สุดอยู่ที่จุดสิ้นสุดของแผนภูมิต้นไม้แทนรหัสหรือไม่ (Top path at end of tree?) ถ้ายังไม่ถึงให้กลับไปทำขั้นตอนของการหาค่าเมตริกซ์ของทางเดินที่อยู่ถัดจาก P ไป 1 ระดับใหม่และทำไปเรื่อยๆจนกว่า P จะเป็นโนดสิ้นสุดของแผนภูมิต้นไม้แทนรหัสซึ่งจะเห็นว่าสแตคจะใช้หน่วยความจำอย่างน้อยที่สุดเมื่อไม่มีสัญญาณรบกวนเลยเท่ากับ $(2^k - 1)L$ เพื่อให้เกิดความเข้าใจมากขึ้น จะแสดงตัวอย่างของการถอดรหัสแบบ สแตคเดียว

ตัวอย่างที่ 1 การถอดรหัสโดยใช้สแตคอัลกอริทึม โดยใช้แผนภูมิต้นไม้แทนรหัสดังภาพที่ 2.5 และให้สัญญาณที่รับได้คือ 11 11 11 01 11 00 10 โดยใช้ค่าเมตริกซ์จากตารางที่ 2 (กำหนด $p = 0.125$) จะมีขั้นตอนการถอดรหัสดังภาพที่ 2.8 หลังจากทำการถอดรหัส 8 ขั้นตอนแสดงดังภาพที่ 2.7 แล้วจะได้ค่าของการถอดรหัสคือ 1100000 ซึ่งจะได้สัญญาณอินพุทจากถอดรหัสคือ 1100

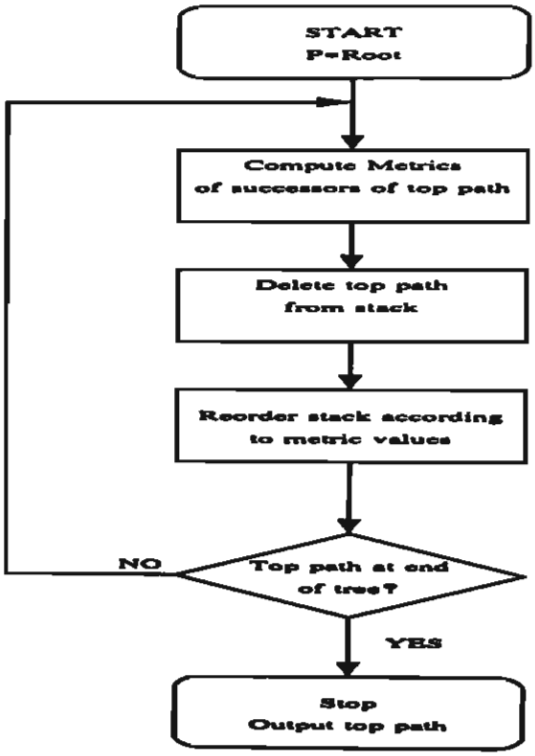
ตารางที่ 2.2

ค่าเมตริกซ์ของแต่ละบิตสำหรับการถอดรหัสแบบสแตคเดียว

$M(r_i / v_i)$	$r_i = 0$	$r_i = 1$
$v_i = 0$	1	-8
$v_i = 1$	-8	1

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8
1 (+2)	10 (-5)	11 (-5)	110 (-3)	1100 (-1)	11000 (+1)	110000 (+3)	1100000 (4)
0 (-16)	11 (-5)	100 (-12)	100 (-12)	100 (-12)	100 (-12)	100 (-12)	100 (-12)
	0 (-16)	101 (-12)	101 (-12)	101 (-12)	101 (-12)	101 (-12)	101 (-12)
		0 (-16)	0 (-16)	0 (-16)	0 (-16)	0 (-16)	0 (-16)
			111 (-21)	1101 (-19)	1101 (-19)	1101 (-19)	1101 (-19)
				111 (-21)	111 (-21)	111 (-21)	111 (-21)
							1100000 stop

ภาพที่ 2.7 ขั้นตอนการถอดรหัสแบบสแตคเดียวของตัวอย่างที่ 1



ภาพที่ 2.8 แผนภาพขั้นตอนการถอดรหัสของสแต็คอัลกอริทึม

2.3.2.2 มัลติเปิลสแตคอัลกอริทึม

งานวิจัยนี้จะเกี่ยวข้องกับการถอดรหัสแบบมัลติเปิลสแตค (Multiple stack algorithm, MSA) ซึ่งเป็นการถอดรหัสที่ถูกพัฒนาขึ้นเพื่อแก้ไขการเกิดอิเรชัว (Erasure) ใน SSA กล่าวคือ บล็อกข้อมูลที่มีการรบกวนจากสัญญาณรบกวนมาก อาจจะใช้เวลาในการค้นหาทางเดินของการถอดรหัสที่ถูกต้องบนแผนภูมิต้นไม้เกินกว่าค่าเวลาที่กำหนดไว้ทำให้จำเป็นต้องหยุดขบวนการถอดรหัสของบล็อกสัญญาณนั้นนั่นคือบล็อกนั้นจะถูกลบทิ้งไปและเรียกข้อมูลที่ถูกลบไปว่าอิเรชัว ต่อมา Chevillat และ Costello (1976) ได้พัฒนาการถอดรหัสแบบ MSA เพื่อแก้ปัญหาอิเรชัวหรือความไม่สมบูรณ์ในการถอดรหัส SSA โดยมีหลักการทำงานเริ่มต้นขั้นตอนการถอดรหัสกับ SSA แต่แตกต่างกันที่จำนวนสแตคที่ใช้ โดยใน MSA จำนวนสแตคจะมีมากกว่าหนึ่งสแตค และกำหนดขนาดของสแตคออกเป็น 2 ส่วนคือ สแตคหลักหรือสแตคแรกและกลุ่มของ สแตครองหรือสแตคอันดับสูงซึ่งจะมีจำนวนหลายสแตค โดยที่ขนาดของสแตคหลักจะโตกว่าขนาดของ สแตครองมาก ในกรณีปกติของการถอดรหัสที่มีสัญญาณรบกวนน้อย การทำงานของขบวนการถอดรหัสมักจะสิ้นสุดที่สแตคหลัก จะมีเพียงบางครั้งเท่านั้นที่สแตครองจะถูกนำมาใช้ นั่นคือในกรณีที่ช่องสื่อสารมีสัญญาณรบกวนมาก ซึ่งกรณีนี้การถอดรหัสแบบ MSA จะสิ้นสุดเมื่อค่าคำนวณของการถอดรหัสถึงช่วงเวลาที่กำหนดไว้ก่อนหน้า โดยทั่วไปแล้วอัตราความผิดพลาดบิตของการถอดรหัสแบบหลาย MSA จะขึ้นกับปัจจัยต่าง ๆ คือ ขนาดของสแตคหลัก ขนาดของสแตครอง จำนวนโนดข้อมูลที่ย้ายระหว่างสแตค ค่าลิมิตของการคำนวณ (Computational limit) C_{lim}

การถอดรหัสแบบ MSA เมื่อสแตคหลักเต็มจำนวนโนดข้อมูลที่มีค่าเมตริกซ์สูงสุดจำนวนหนึ่งจะถูกย้ายไปยังสแตครอง จากนั้นกระบวนการถอดรหัสจะดำเนินต่อไปในสแตครองโดยใช้จำนวนโนดที่ย้ายมา ถ้าการถอดรหัสถึงโนดสุดท้ายของแผนภูมิต้นไม้แทนรหัสก่อนที่สแตครองเต็ม เส้นทางเดินจากโนดเริ่มต้นจนถึงโนดสุดท้ายจะถูกเก็บไว้ในรีจิสเตอร์พิเศษ(Special register) และเรียกทางเดินที่เก็บไว้ว่า *การตัดสินใจเบื้องต้น* (Tentative decision, TD) หลังจากนั้นตัวถอดรหัสจะลบโนดที่เหลือในสแตครองและกลับมาเริ่มกระบวนการถอดรหัสอีกครั้งในสแตคหลักโดยเริ่มจากโนดที่อยู่บนสุด (ดีที่สุด) ในตอนนี้สแตคหลักจะมีเนื้อที่ว่างเท่ากับจำนวนโนดที่ถูกย้ายในตอนแรก ถ้าการถอดรหัสดำเนินมาถึงโนดสุดท้ายของแผนภูมิต้นไม้แทนรหัสก่อนที่ สแตคหลักจะเต็มอีกครั้งตัวถอดรหัสจะเปรียบเทียบค่าเมตริกซ์ของเส้นทางเดินใหม่นี้กับเส้นทางเดิน TD และเลือกทางเดินที่มีค่าเมตริกซ์ที่ดีกว่าเป็นเส้นทางการถอดรหัส ซึ่งวิธีการในลักษณะนี้สามารถให้ผลของการถอดรหัสดีเสมือนกับการถอดรหัสแบบ SSA เมื่อกำหนดขนาดสแตคให้มีค่าโตมากๆ อย่างไรก็ตามถ้าสแตคหลักเต็มอีกครั้งสแตครองก็จะถูกสร้างขึ้นมาใหม่และมีการย้ายโนดข้อมูลในลักษณะเดิม การสร้างสแตครองก็จะเกิดขึ้นในลักษณะเช่นนี้ต่อไปเรื่อยๆ จนกระทั่งตัวถอดรหัสถึงโนดสุดท้ายของเส้นทางเดินในแผนภูมิต้นไม้แทนรหัส และทุกครั้งที่มีการถึงโนดสุดท้ายตัวถอดรหัสก็จะเปรียบเทียบค่าเมตริกซ์ของเส้นทางใหม่กับค่า TD ที่เก็บไว้และเลือกเส้นทางเดินที่มีค่าเมตริกซ์ที่ดีที่สุดเสมอ กระบวนการถอดรหัสจะสิ้นสุดเมื่อการทำงานของตัวถอดรหัสถึงโนดสุดท้ายในส

แตคหลักหรือจำนวนรอบของการคำนวณถึงค่าลิมิตของการคำนวณ (C_{lim}) และเลือกเส้นทางเดินที่ดีที่สุดเมื่อเปรียบเทียบกับ TD ให้เป็นเส้นทางการถอดรหัสสุดท้าย (Final decoded path)

เพื่อให้การถอดรหัสแบบ MSA เป็นการถอดรหัสที่ไม่มีอีเรชัว สำหรับกรณีที่กำหนดจำนวนโนดที่ย้ายระหว่างสแตคเท่ากับหนึ่ง ($T=1$) และอัตราการเข้ารหัสคอนโวลูชันเป็น $1/w$ เมื่อ w คือจำนวนเอาท์พุตของตัวเข้ารหัส และกำหนดค่าความยาวคอนสเตรนท์เป็น v ค่าลิมิตของการคำนวณ $C_{\lim}$ จะต้องมีค่ามากกว่าค่าขอบเขตวิกฤตต่ำสุดของการคำนวน (C_{crit}) ซึ่งกำหนดโดย (Chevillat และ Costello,1976)

$$C_{crit} = \sum_{i=1}^{k-1}(Z_i - 1) + 2(v-1) \qquad (2.5)$$

โดยที่ $Z_i, i = 1, 2, \cdots, k-1$ เป็นขนาดของสแตคที่ i และ k คือ จำนวนบิตของสัญญาณข้อมูลต่อบล็อก การถอดรหัสที่สิ้นสุดด้วยค่า $C_{\lim}$ ใน MSA ส่วนใหญ่จะเกิดขึ้นกับบล็อกที่มีสัญญาณรบกวนมาก ๆ ซึ่งกรณีนี้ ค่าอัตราความผิดพลาดบิตมักจะมีค่าสูงเนื่องจากบ่อยครั้งที่ TD เป็นเส้นทางเดินที่ไม่ถูกต้อง ถึงแม้ว่าประสิทธิภาพของการถอดรหัสแบบ MSA จะสามารถทำให้ดีขึ้นได้โดยการเพิ่มค่าของ $C_{\lim}$ แต่ก็จะทำให้การถอดรหัสดำเนินไปได้ช้าและใช้หน่วยความจำในการถอดรหัสมากขึ้นตามไปด้วย

ปัจจัยที่มีผลต่อประสิทธิภาพของการถอดรหัสแบบหลายสแตค

ปัจจัยที่มีผลต่ออัตราความผิดพลาดบิตของอัลกอริทึมแบบ MSA ที่จะสามารถถอดรหัสสัญญาณได้โดยที่ไม่มีอิเรชัวนั้น (Chevillat และ Costello ,1976) ได้นำเสนอไว้ดังนี้

1. ขนาดของสแตคหลัก (Z_1)

ในการถอดรหัสแบบ MSA อัตราความน่าจะเป็นที่ที่คาดว่าจะเกิดอิเรชัวในขณะที่สแตคแรกเต็มถูกกำหนดโดย

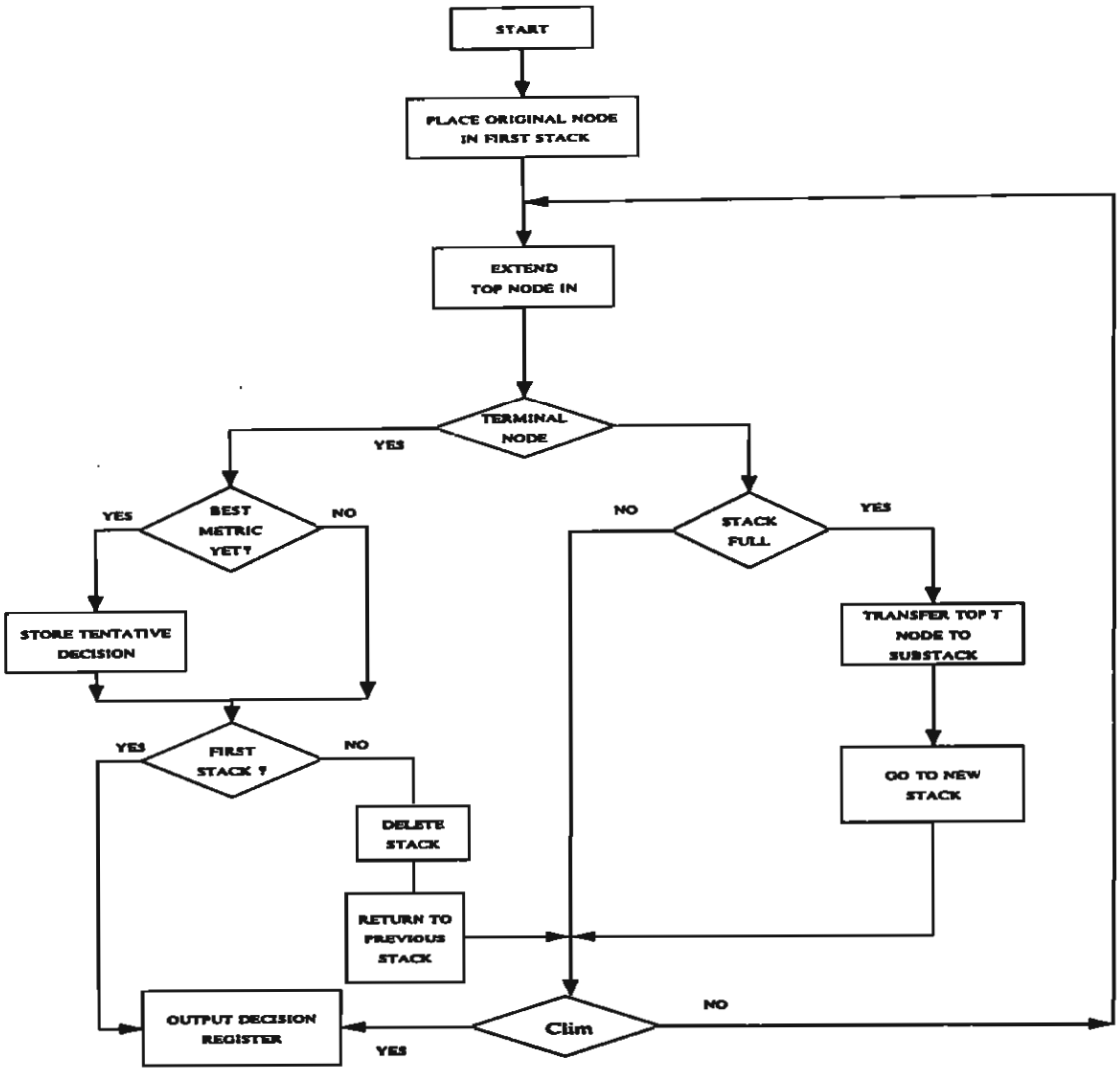
$$P_{es} = P(\hat{r} \neq \hat{v}/u > 1) \qquad (2.6)$$

โดยที่ $\hat{r}$ คือจำนวนสัญญาณที่รับได้ $\hat{v}$ คือจำนวนรหัสสัญญาณรหัสอ้างอิง และ u คือจำนวนสแตค ถ้ากำหนด $P_{E(SSA)}$ คือความน่าจะเป็นของอัตราความผิดพลาดบิตของการถอดรหัสแบบ SSA ซึ่ง มีขนาดของสแตคไม่จำกัด ดังนั้นความน่าจะเป็นของอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ที่จะไม่มีอิเรชัวจะถูกกำหนดดังสมการ ที่ 2.8

$$P_{E(MSA)} \leq P_{E(SSA)} + P_{es} \bullet P_1 \qquad (2.7)$$

โดยที่

$$P_1 = P(C > Z_1 - 1) = c_{SSA} \bullet (Z_1 - 1)^{-\gamma} \qquad (2.8)$$



ภาพที่ 2.9 แผนภาพขั้นตอนการถอดรหัสแบบหลายสแตค (MSA)

โดยที่ A คือ ความน่าจะเป็นที่สแตคแรกจะเต็ม ซึ่งมีลักษณะของการกระจายตามหลักการของ Pareto (Pareto Distributed) และ ψ คือ ค่าฟังก์ชันของช่องสัญญาณและอัตราการเข้ารหัส c_{SSA} เป็นค่าคงที่ของการถอดรหัสแบบซีเควนเชียล (Forney, G. D. ,1974)

ถ้ากำหนดให้ขนาดของสแตคหลักมีขนาดไม่จำกัด ค่า $P_1 = 0$ $\left(\lim_{Z_1 \to \infty} P_1 = 0\right)$ ความน่าจะเป็นของอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA จะเท่ากับความน่าจะเป็นของอัตราความผิดพลาดบิตของการถอดรหัสแบบ SSA

$$\lim_{Z_1 \to \infty} P_{E(MSA)} = P_{E(SSA)} \qquad (2.9)$$

ถ้ากำหนดให้ขนาดของสแตคหลักมีขนาดจำกัด ค่า $P_1 > 0$ จะได้

$$P_{E(MSA)} \leq P_{E(SSA)} + P_1 \qquad (2.10)$$

จากสมการที่ 2.10 จะเห็นว่าความน่าจะเป็นที่อัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA จะลดลง เมื่อเพิ่มขนาดของสแตคแรกตามฟังก์ชันของ P_1

2. ค่าลิมิตของการคำนวณ (Computational limit, $C_{\lim}$)

จากหัวข้อที่แล้วถ้ากำหนดขนาดของสแตคหลักของการถอดรหัสแบบ MSA ให้มีขนาดไม่จำกัดแล้วประสิทธิภาพที่ได้จากถอดรหัสจะเท่ากับการถอดรหัสแบบ SSA ซึ่งหมายความว่าค่ารอบของการคำนวณก็จะมีค่าเพิ่มตามขนาดของสแตคหลักด้วย ดังนั้นถ้ากำหนดขนาดของสแตคหลักให้คงแต่เปลี่ยนจำนวน $C_{\lim}$ แล้วค่าประสิทธิภาพในอัตราความผิดพลาดบิตที่ได้จะดีขึ้นในขณะที่มีการเพิ่มค่า $C_{\lim}$ คือ

$$\lim_{C_{\lim} \to \infty} P_{E(MSA)} \leq P_{E(SSA)} \qquad (2.11)$$

แต่อย่างไรก็ตามในการถอดรหัสแบบ MSA เพื่อไม่ให้เกิดอิเรชัวนั้น ค่าลิมิตของการคำนวณจะต้อง มีค่ามากกว่าค่าขอบเขตวิกฤตต่ำสุดของการคำนวน C_{crit} ดังสมการที่ 2.5

3. จำนวนโนดที่ย้ายระหว่างสแตค (T) และขนาดของสแตครอง (Z)

ในการถอดรหัสแบบ MSA นอกจากผลของขนาดสแตคหลัก และ ค่า $C_{\lim}$ ที่มีผลต่อประสิทธิภาพในอัตราความผิดพลาดบิตแล้ว ในขั้นตอนการถอดรหัสถ้าการถอดรหัสไม่สามารถสิ้นสุดในสแตคหลักได้ จะมีการย้ายโนดข้อมูลมายังสแตครองซึ่งก็เป็นอีกปัจจัยที่มีผลต่อการถอดรหัสเช่นกัน จากผลการทดลองของ Chevillat และ Costello (1976) ได้แสดงให้เห็นว่าถ้ากำหนดจำนวนโนดที่ย้ายระหว่างสแตคทั้งสองให้มีจำนวนมากขึ้นโอกาสที่เส้นทางเดินที่ถูกต้องของการถอดรหัสจะถูกย้ายไปดำเนินการถอดรหัสต่อในสแตครองก็จะเพิ่มมากขึ้นด้วยและจะได้ผลของการถอดรหัสที่ดีขึ้นเช่นกัน แต่ไม่มีความแตกต่างในอัตราความผิดพลาดบิตที่เป็นนัยสำคัญมากนัก และการเพิ่มจำนวนโนดที่ย้ายมากขึ้นขนาดของสแตครอง (Z) ก็จะต้องมีขนาดใหญ่มากตามด้วย อย่างไรก็ตามจุดประสงค์หลักที่สำคัญของการกำหนดให้มีสแตครองของการถอดรหัสแบบ MSA ก็เพื่อให้ การถอดรหัสไม่เกิดอิเรชัว ซึ่งการถอดรหัสจะต้องดำเนินการเพื่อให้ได้ค่า *การตัดสินใจเบื้องต้นอันดับแรก* (First Tentative decision,TD) เร็วที่สุด ดังนั้นจะต้องกำหนดให้จำนวนโนดข้อมูลที่ย้ายมีจำนวนน้อย และขนาดของสแตครองต้องมีขนาดเล็กสามารถสรุปได้ว่าประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA จะเพิ่มขึ้นได้โดยการเพิ่มขนาดของสแตคหลัก หรือการเพิ่มค่าลิมิตของการคำนวณ เป็นปัจจัยสำคัญ แต่อย่างไรก็ตาม การเพิ่มขนาดส

แคดหรือหน่วยความจำนั้นอาจจะไม่สามารถทำได้ และในกรณีที่มีสัญญาณรบกวนมาก ๆ ในช่องสัญญาณ เมื่อการถอดรหัสดำเนินในสแตครองแล้วโอกาสที่การถอดรหัสจะสามารถกลับมาสิ้นสุดที่สแตคหลักนั้นจะมีน้อยมาก ซึ่งจะทำให้การถอดรหัสจะสิ้นสุดโดยค่าของ C_{lim} ซึ่งเป็นผลให้อัตราความผิดพลาดบิตมีค่าสูง และการเพิ่มค่า C_{lim} ยังทำให้การถอดรหัสสิ้นสุดช้า ในงานวิจัยนี้ได้ทำการประยุกต์การถอดรหัสแบบ MSA โดยใช้มัลติโพรเซสเซอร์ เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของ MSA เดิมให้ดีขึ้น ที่ค่า C_{lim} และปริมาณหน่วยความจำที่เท่ากัน ซึ่งจะกล่าวถึงในบทต่อไป

บทที่ 3
การถอดรหัสแบบหลายสแตคโดยมัลติโพรเซสเซอร์

3.1 บทนำ

ในบทนี้จะกล่าวถึงวิธีการประยุกต์ใช้จำนวนหน่วยความจำรวมของระบบการถอดรหัสแบบ MSA และโครงสร้างของการถอดรหัสแบบขนานใน MSA โดยใช้ตัวประมวลผลหลายตัว เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA เดิมให้ดีขึ้นที่ค่า $C_{\lim}$ และปริมาณหน่วยความจำที่เท่ากันโดยได้แบ่งขั้นตอนในการศึกษาออกเป็น 3 ส่วนคือ การจัดสรรหน่วยความจำ การออกแบบโครงสร้างการถอดรหัสโดยใช้ตัวประมวลผลหลายตัว และการทดสอบผลการออกแบบเบื้องต้น

3.2 รูปแบบ MSA โดยมัลติโพรเซสเซอร์

การออกแบบเพื่อประยุกต์ใช้จำนวนหน่วยความจำรวมของระบบการถอดรหัสแบบ MSA และนำหน่วยความจำมาใช้ให้เป็นประโยชน์อย่างเต็มที่ ในการศึกษาเบื้องต้นนั้นได้ศึกษารูปแบบการใช้หน่วยความจำของระบบการถอดรหัสแบบ MSA และศึกษาว่าควรจะใช้จำนวนโนดที่ย้ายระหว่างตัวประมวลผลในลักษณะใด ดังนี้คือ

การจัดสรรหน่วยความจำ

การจัดสรรหน่วยความจำรวมของการถอดรหัสแบบMSA เดิม สำหรับตัวประมวลผลแต่ละตัวในระบบการถอดรหัสใหม่สามารถแบ่งได้เป็น 2 รูปแบบคือ

1. ตัวประมวลผลมีจำนวนหน่วยความจำรวมไม่เท่ากัน

การจัดหน่วยความจำในลักษณะนี้จะนำหน่วยความจำทั้งหมดของสแตครองของการถอดรหัสแบบMSA เดิมซึ่งเท่ากับ $n_z(Z)$ โดยที่ Z คือขนาดของสแตครอง, n_Z คือ จำนวนของสแตครองมาจัดสรรให้ตัวประมวลผล n ตัว และกำหนดให้ตัวประมวลผลตัวแรก (DEC #1) เท่านั้นที่มีสแตคหลัก Z_1 ซึ่งกำหนดให้มีขนาดเท่ากับสแตคหลักเดิม และมีหน่วยความจำเท่ากับ $Z_1 + n_{zq}(Z)$ สำหรับตัวประมวลผลอื่น ๆ กำหนดให้มีขนาดสแตคเท่ากับสแตครอง และมีหน่วยความจำรวมเท่ากับ $n_{zq}(Z)$ โดยที่ $n_{zq} = \frac{\Gamma - Z_1}{nZ}$, $\Gamma = Z_1 + n_z(Z)$ คือหน่วยความจำทั้งหมดของการถอดรหัสแบบ MSA เดิม

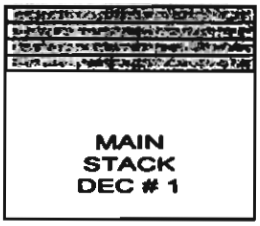
2. ตัวประมวลผลทุกตัวมีจำนวนหน่วยความจำรวมเท่ากัน

การจัดสรรหน่วยความจำในลักษณะนี้จะนำหน่วยความจำทั้งหมดของการถอดรหัสแบบ MSA เดิม ซึ่งเท่ากับ $\Gamma = Z_1 + n_z(Z)$ โดยที่ Z_1 คือขนาดของสแตคหลัก, Z คือขนาดของสแตครอง , n_Z คือ จำนวนของสแตครอง ดังนั้นถ้าระบบการถอดรหัสใหม่มีจำนวนตัวประมวลผล n ตัว จะสามารถแบ่งหน่วยความจำให้ตัวประมวลผลได้เท่ากันหมดคือเท่ากับ $\Gamma' = Z_1 + n_{zp}(Z)$ โดยที่ $n_{zp} = \frac{n_z - (n-1)(Z_1 / Z)}{n}$

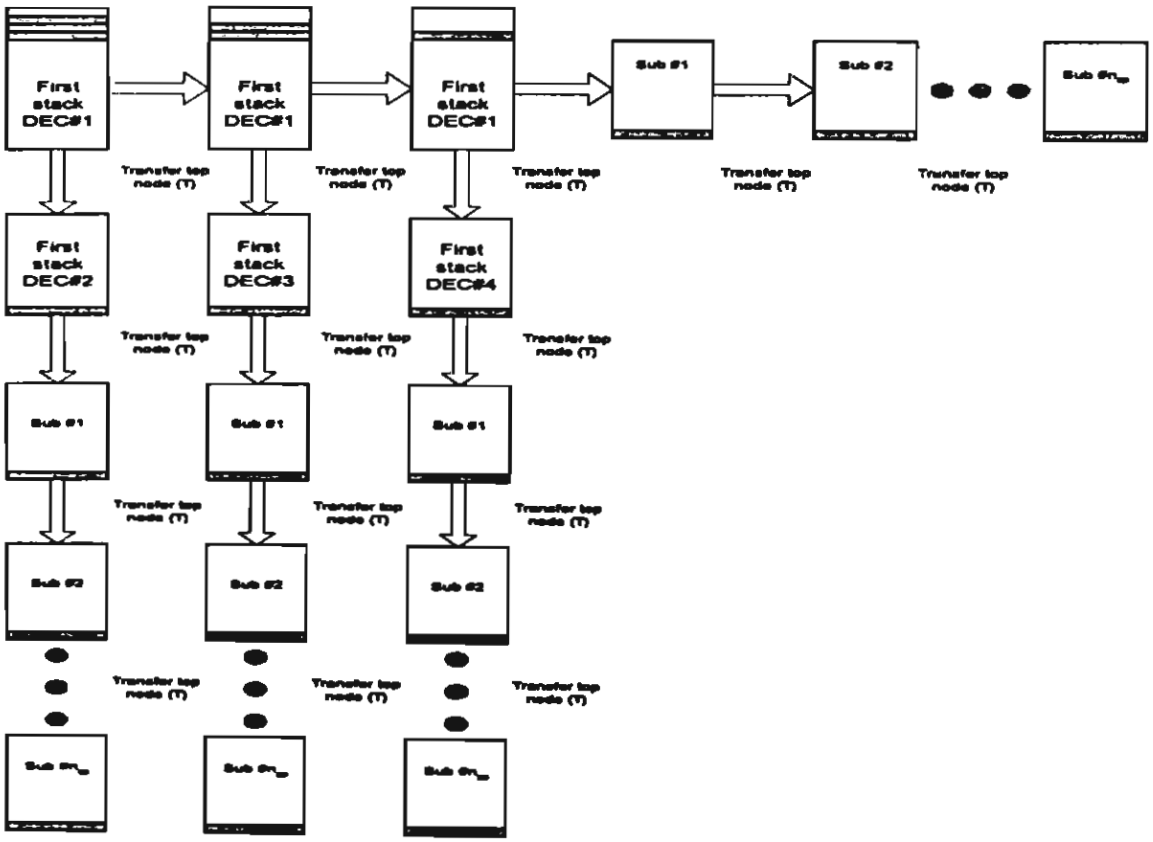
การกำหนดขนาดโนดข้อมูลที่ย้ายระหว่างตัวประมวลผล

การย้ายโนดข้อมูลระหว่างสแตคจะมีผลต่อประสิทธิภาพของการถอดรหัสอย่างไรนั้นผู้วิจัยได้ศึกษาและได้ทดลองจำลองการถอดรหัส โดยแบ่งลักษณะการย้ายจำนวนโนดข้อมูลและจำนวนโนดข้อมูลที่ถูกย้ายให้มีความสัมพันธ์กับรูปแบบการจัดสรรหน่วยความจำคือ

1. กำหนดให้จำนวนโนดที่ย้ายระหว่างตัวประมวลผลเท่ากับจำนวนโนดที่ย้ายระหว่างสแตคของแต่ละตัวประมวลผล และใช้กับการจัดสรรหน่วยความจำรูปแบบที่ 1 โดยมีลักษณะการย้ายโนดข้อมูลตามโครงสร้างดังภาพที่ 3.1 ซึ่งมีลักษณะของขั้นตอนการถอดรหัสและย้ายโนดข้อมูลได้ดังนี้คือ



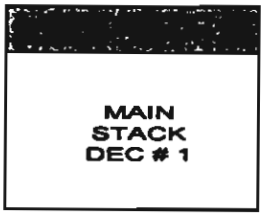
a) รูปแบบการย้ายจำนวนโนดเมื่อสแตคหลักของตัวประมวลผลตัวแรกเต็ม



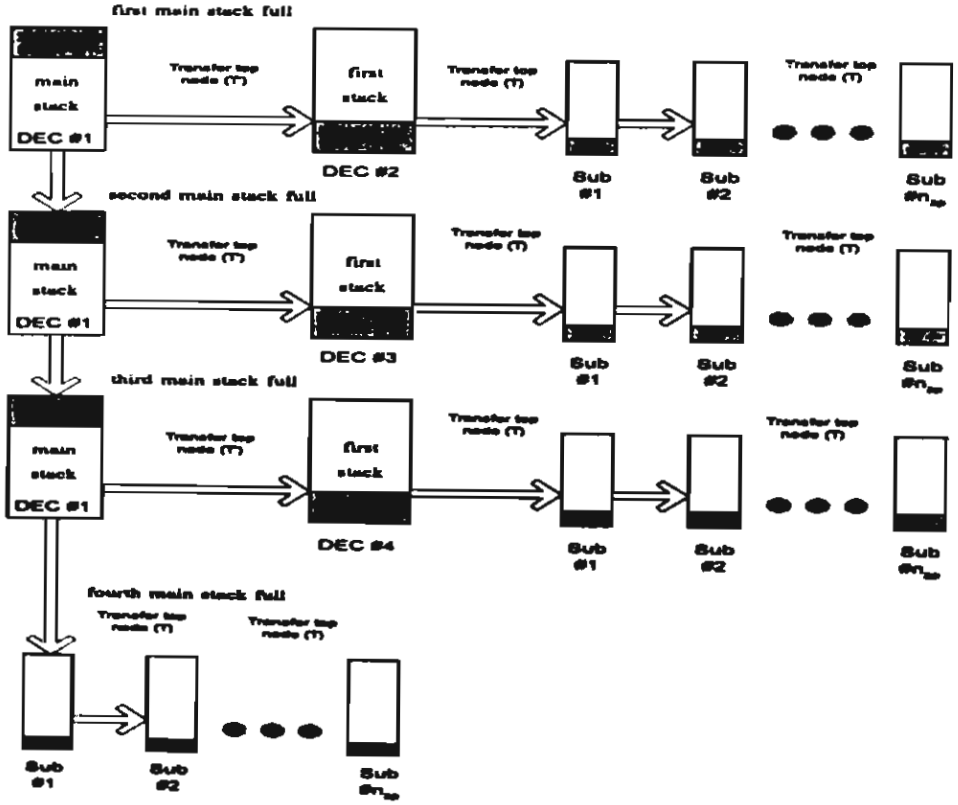
b) รูปแบบการย้ายจำนวนโนดระหว่างตัวประมวลผล

ภาพที่3.1 โครงสร้างการย้ายโนดข้อมูลของการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์รูปแบบที่ 1

เมื่อเนื้อที่หน่วยความจำของสแตคแรกของตัวประมวลผลตัวที่ 1(DEC #1) เต็มในขณะที่การถอดรหัสยังไม่ถึงโนดสุดท้ายของข้อมูลบนทางเดินของแผนภูมิต้นไม้แทนรหัส ขั้นตอนการถอดรหัสจะย้ายโนดข้อมูล T โนดจากสแตคแรกของ DEC #1 มายังสแตคแรกของตัวประมวลผลตัวที่ 2 (DEC #2) จากนั้นย้าย T โนดอันดับถัดไปมายังตัวประมวลผลตัวที่ 3 (DEC #3) และย้ายจำนวน T โนดอันดับถัดไปมายังตัวประมวลผลตัวที่ 4 (DEC #4) ตามลำดับ (ในที่นี้สมมุติใช้ตัวประมวลผล 4 ตัว) เมื่อย้ายจำนวนโนดข้อมูลมายังตัวประมวลผลครบทุกตัวแล้ว ทุกตัวประมวลผลจะเริ่มถอดรหัสต่อไปภายในสแตคแรกและดำเนินการถอดรหัสเหมือนการถอดรหัสแบบ MSA อย่างอิสระจนกระทั่งการถอดรหัสดำเนินถึงค่าลิมิตของการคำนวณ $C_{\lim}$ เมื่อการถอดรหัสสิ้นสุด จะเปรียบเทียบค่า *การตัดสินใจเบื้องต้น* (Tentative decision, TD) ของแต่ละตัวประมวลผลและเลือกค่า TD ที่ดีที่สุดเป็นเส้นทางการถอดรหัสสุดท้าย



ก) รูปแบบการย้ายจำนวนโนดเมื่อสแตคหลักของตัวประมวลผลตัวแรกเต็ม



ข) รูปแบบการย้ายจำนวนโนดระหว่างตัวประมวลผล

ภาพที่ 3.2 โครงสร้างการย้ายโนดข้อมูลของการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ รูปแบบที่ 2

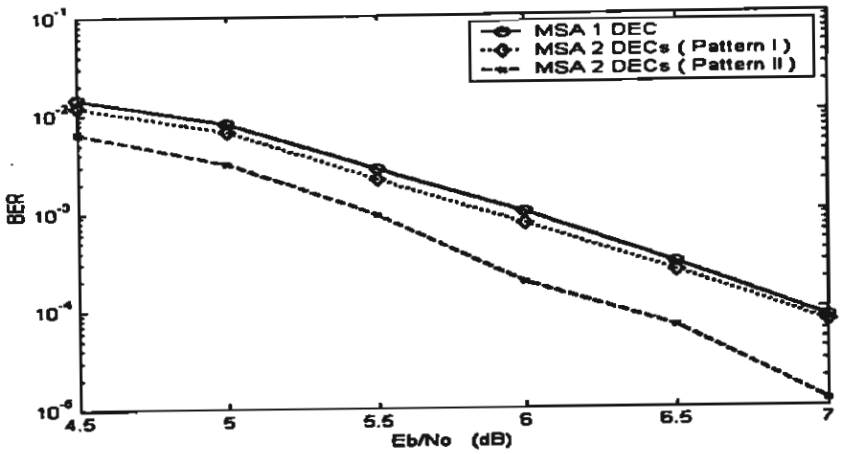
2. กำหนดให้จำนวนโนดที่ย้ายระหว่างตัวประมวลผลมีค่ามากกว่าจำนวนโนดที่ย้ายระหว่าง สแตคของแต่ละตัวประมวลผล และใช้กับการจัดสรรหน่วยความจำตามรูปแบบที่ 2 โดยมีลักษณะการย้ายโนดข้อมูลตามโครงสร้าง ดังภาพที่ 3.2 ซึ่งมีลักษณะของขั้นตอนการถอดรหัสและย้ายโนดข้อมูลได้ดังนี้คือ

เมื่อเนื้อที่หน่วยความจำของสแตคแรกของตัวประมวลผลตัวที่ 1(DEC #1) เต็มในขณะที่การถอดรหัสยังไม่ถึงโนดสุดท้ายของข้อมูลบนทางเดินของแผนภูมิต้นไม้แทนรหัส การถอดรหัสจะย้ายโนด T' โนดจากสแตคแรกของ DEC #1 มายังสแตคแรกของตัวประมวลผลตัวที่ 2 (DEC #2) จากนั้นถอดรหัสแบบ MSA พร้อมกันทั้งสองตัวประมวลผล เมื่อสแตคแรกของตัวประมวลผล DEC #1 เต็มอีกครั้งจะย้ายโนด T' โนดมายังตัวประมวลผลตัวที่ 3 (DEC #3) และดำเนินการถอดรหัสต่อเหมือนเดิมในขณะที่ตัวประมวลผล DEC #2 ก็ยังดำเนินการถอดรหัสอยู่เช่นกัน และเมื่อสแตคแรกของตัวประมวลผล DEC #1 เต็มอีกครั้งจะย้ายโนด T' โนดมายังตัวประมวลผลตัวที่ 4 (DEC #4) (ในที่นี้สมมุติใช้ตัวประมวลผล 4 ตัว) แล้วทุกตัวประมวลผลทำการถอดรหัสแบบ MSA จนกระทั่งสิ้นสุดการถอดรหัส ในโครงสร้างนี้ตัวประมวลผลทุกตัวดำเนินการถอดรหัสเหมือนการถอดรหัสแบบ MSA อย่างอิสระเช่นเดียวกันกับโครงสร้างรูปแบบที่ 1คือถ้าสแตคแรกของแต่ละตัวประมวลผลเต็มจะย้ายโนดข้อมูลจำนวน T โนดไปยังสแตคอันดับถัดไป แล้วดำเนินการถอดรหัสต่อไปจนกระทั่งการถอดรหัสดำเนินถึงลิมิตของการคำนวณ $C_{\lim}$ เมื่อการถอดรหัสสิ้นสุดจะเปรียบเทียบค่า TD ของแต่ละตัวประมวลผลและเลือกค่า TD ที่ดีที่สุดเป็นเส้นทางการถอดรหัสสุดท้าย

3.3 ผลการทดสอบโครงสร้างรูปแบบ

การทดสอบโครงสร้างที่ได้ออกแบบข้างต้นเพื่อเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตและเปรียบเทียบกับการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว ผู้วิจัยได้จำลองระบบโดยการเขียนโปรแกรมคอมพิวเตอร์ภาษาซี บนระบบปฏิบัติการลีนุกซ์ พารามิเตอร์ที่ใช้ในการจำลองที่สำคัญมีดังต่อไปนี้คือ ช่องสื่อสารเป็นแบบ AWGN (Additive White Gaussian noise) อัตราการเข้ารหัสคอนโวลูชันเป็น 1/2 ค่าความยาวคอนสเตรนท์ของตัวเข้ารหัส $m = 5$, Generator matrices $G_1 = 110101$, $G_2 = 101111$ จำนวนบิตต่อเฟรมเป็น 60 จำนวนบล็อกทั้งหมด 20,000 บล็อก (จำนวนบิตทดสอบรวม 1,200,000 บิต) $C_{\lim} = 2500$ จำนวนโนดที่ย้ายระหว่างสแตคหลักและสแตครองภายในแต่ละตัวประมวลผล T เป็น 2 ขนาดของสแตคหลัก Z_1 เป็น 150 ขนาดของสแตครอง Z เป็น 12 จำนวนสแตครองของระบบตัวประมวลเดียว n_z เป็น 300 ดังนั้นจำนวนหน่วยความจำรวมของระบบตัวประมวลเดียวเท่ากับ 3750 เมื่อใช้โครงสร้างที่ 1 (กำหนดใช้ตัวประมวลผล 2 ตัว) ดังนั้นตัวประมวลผลตัวที่ 1 (DEC #1) มีขนาดหน่วยความจำรวม 1950 ตัวประมวลผลตัวที่ 2 (DEC #2) มีขนาดหน่วยความจำรวม 1800 สำหรับโครงสร้างรูปแบบที่ 2 DEC #1 มีขนาดหน่วยความจำรวม 1725 , DEC #2 มีขนาดหน่วยความจำรวม 1725 และใช้ $T' = 50$ ภาพที่ 3.3 แสดงค่าอัตราความผิดพลาดบิตของทั้งสองโครงสร้าง ที่ค่า E_b / N_o คือ 4.5 -7.0 dB

จากผลการทดสอบแสดงให้เห็นว่าการจัดสรรจำนวนหน่วยความจำและรูปแบบการย้ายโนดข้อมูล ตามโครงสร้างรูปแบบที่ 2 ให้ประสิทธิภาพในอัตราความผิดพลาดบิตที่ดีกว่า โครงสร้างรูปแบบที่ 1 อย่างเห็นได้ชัด ดังนั้นผู้วิจัยได้ตัดสินใจเลือกรูปแบบที่ 2 นี้เป็นแนวทางในการทดสอบการถอดรหัสแบบขนานใน MSA โดยใช้ตัวประมวลผลหลายตัว ซึ่งจะกล่าวถึงรายละเอียดของขั้นตอนการถอดรหัสในหัวข้อถัดไป



ภาพที่ 3.3 ค่า BER ของการถอดรหัสแบบ MSA และ การถอดรหัสแบบ MSA 2 ตัวประมวลผลตามโครงสร้างรูปแบบ ที่ 1 และ โครงสร้างรูปแบบที่ 2 ที่ค่า E_b / N_o ต่างๆกัน

3.4 การถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์

ในงานวิจัยนี้มีเป้าหมายหลักคือการปรับปรุงประสิทธิภาพของการถอดรหัสแบบ MSA โดยใช้ตัวประมวลผลหลายตัว และได้ออกแบบการใช้ตัวประมวลผลที่มีคุณลักษณะเหมือนกันทุกประการจำนวน n ตัว สำหรับถอดรหัสโดยใช้อัลกอริทึม MSA ไปพร้อมๆกัน(ตัวประมวลผลทุกตัวจะทำหน้าที่เป็นตัวถอดรหัสอย่างอิสระ) ด้วยแนวความคิดของวิธีการถอดรหัสนี้ จะทำให้ค่า TD มีค่าที่ดีขึ้นเนื่องจากการถอดรหัสที่ใช้ตัวประมวลผลหลายตัวสามารถถึงโนดสุดท้ายได้จำนวนครั้งมากขึ้นในเวลาที่กำหนด ดังนั้นโอกาสที่จะได้ทางเดินที่ถูกต้องก็จะมากขึ้นตามไปด้วย ส่งผลให้ค่าอัตราความผิดพลาดบิตมีค่าลดลงเมื่อเปรียบเทียบทั้งสองระบบที่กำหนดให้ค่า $C_{\lim}$ และขนาดของหน่วยความจำรวมเท่ากัน

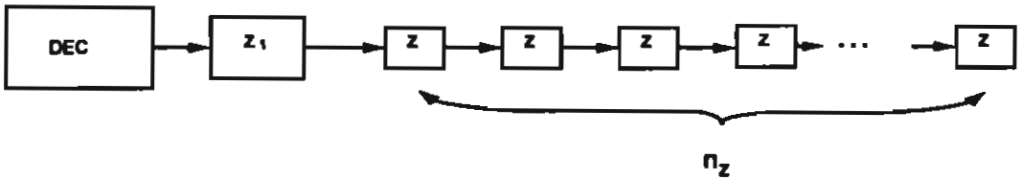
ภาพที่ 3.4 แสดงโครงสร้างของระบบการถอดรหัสแบบ MSA โดยใช้ตัวประมวลผลหลายตัวที่ใช้สำหรับงานวิจัยนี้ ซึ่งสามารถอธิบายได้ดังนี้ กำหนดขนาดหน่วยความจำรวมของสแตคทั้งหมดในระบบสำหรับใช้ในการถอดรหัสเท่ากันกับการถอดรหัสแบบ MSA เมื่อใช้ตัวประมวลผลตัวเดียว และมีขนาดของสแตคทุก สแตคเท่ากัน ข้อแตกต่างของทั้งสองระบบคือ จำนวนสแตครองในระบบ MSA เดิมจะมีจำนวนมากกว่าจำนวน สแตครองของระบบใช้ตัวประมวลผลหลายตัว ถ้ากำหนดให้จำนวนของสแตครองของการถอดรหัสแบบ MSA เดิมเท่ากับ n_s และ จำนวนของสแตครองของการถอดรหัสแบบ MSA โดยใช้ตัวประมวลผลหลายตัวเท่ากับ n_{sp} และ Γ คือขนาดของหน่วยความจำรวมของการถอดรหัสแบบ MSA

เดิมนั่นคือ $\Gamma = Z_1 + n_z Z$ เมื่อ Z_1 คือ ขนาดของสแตคหลัก และ Z คือ ขนาดของสแตครอง หน่วยความจำของตัวประมวลผลแต่ละตัวของระบบการถอดรหัสแบบ MSA โดยใช้ตัวประมวลผลหลายตัว สามารถเขียนเป็นสมการได้ดังนี้ คือ

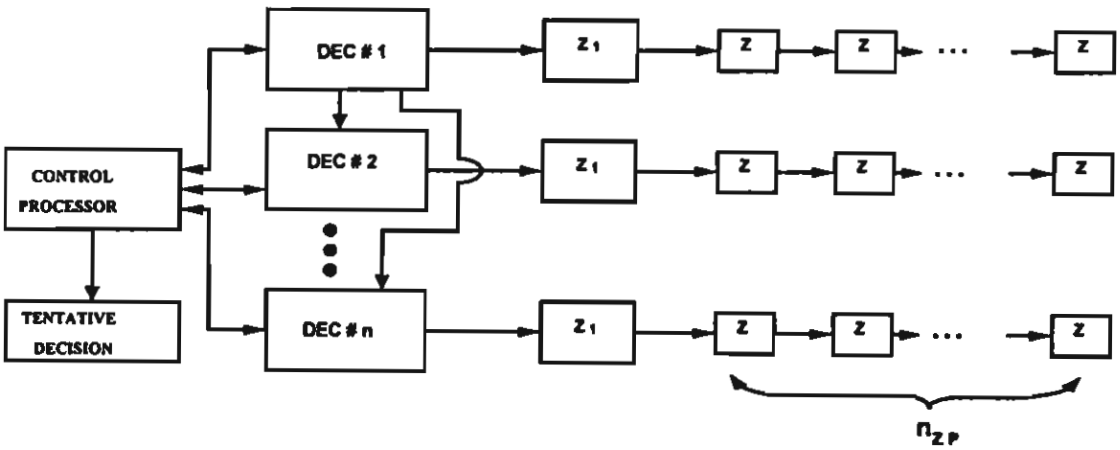
$$\Gamma' = Z_1 + n_{zp} Z, \qquad n_{zp} = \frac{n_z - (n-1)(Z_1 / Z)}{n} \tag{3.1}$$

เมื่อ Γ' คือ หน่วยความจำรวมของตัวประมวลผลแต่ละตัวของระบบการถอดรหัสแบบขนาน โดยที่

$$\Gamma = n\Gamma' \tag{3.2}$$



a. MSA Single Processor



b. MSA Multi Processor

ภาพที่ 3.4 โครงสร้างการถอดรหัสสำหรับ MSA (a) MSA ตัวประมวลผลเดียว
(b) MSA หลายตัวประมวล

3.4.1 ขั้นตอนการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์

ขั้นตอนที่ 1 ตั้งค่า $j = 1$ โหลดค่าโนดเริ่มต้นในสแตคหลักของตัวประมวลผลตัวแรก (DEC #1)

ขั้นตอนที่ 2 DEC #1 เริ่มทำการถอดรหัสแบบ MSA ในสแตคหลักเท่านั้น

ขั้นตอนที่ 3 ตรวจสอบว่าสแตคหลักของ DEC #1 เต็มหรือไม่ ถ้าใช่, ตั้งค่า $j = j+1$ จากนั้นย้ายโนดข้อมูลจำนวน T' โนดในสแตคหลักของ DEC #1 มายังสแตคหลักของตัวประมวลผลตัวที่ j (DEC # j)

ขั้นตอนที่ 4 ตรวจสอบว่า $j = n$ หรือไม่, ถ้าใช่ ไปขั้นตอนที่ 5 ถ้าไม่ใช่ ให้กลับไปขั้นที่ 2 ในขณะเดียวกัน DEC #2 ถึง DEC# j จะเริ่มถอดรหัสสัญญาณแบบ MSA อย่างอิสระพร้อมๆกันในขั้นตอนนี้

ขั้นตอนที่ 5 DEC #1 ถึง DEC # j ดำเนินการถอดรหัสโดยวิธีการถอดรหัสแบบ MSA ไปพร้อมๆกัน

3.4.2 กฎเกณฑ์การสิ้นสุดของขบวนการถอดรหัส

กระบวนการถอดรหัสจะสิ้นสุดเมื่อตัวประมวลผลควบคุม (Control processor, CP) ตรวจสอบพบเงื่อนไขในข้อใดข้อหนึ่งต่อไปนี้ คือ

1. การถอดรหัสดำเนินถึงโนดสุดท้ายของเส้นทางเดินในแผนภูมิต้นไม้แทนรหัสในสแตคหลัก (แรก) ของ DEC #1 ซึ่งกรณีนี้ค่าเมตริกซ์ของเส้นทางเดินจากโนดเริ่มต้นถึงโนดสุดท้ายจะถูกนำมาเปรียบเทียบกับค่าเมตริกซ์ของทางเดิน TD จากนั้นเลือกทางเดินที่ดีกว่า (ค่าเมตริกซ์ที่สูงกว่า) เป็นเส้นทางเดินของการถอดรหัสสุดท้าย

2. จำนวนรอบของการถอดรหัสมีค่าเท่ากับ $C_{\lim}$ ในกรณีนี้ทางเดิน TD ที่ถูกบันทึกไว้ในรีจิสเตอร์พิเศษจะเป็นเส้นทางการถอดรหัสสุดท้าย

3.4.3 หน้าที่ของตัวประมวลผลควบคุม CP

1. เปรียบเทียบและบันทึกค่าทางเดินที่ดีที่สุด TD ในรีจิสเตอร์พิเศษเมื่อตัวประมวลผลแต่ละตัวถึงโนดสุดท้าย (ในกรณีที่การถอดรหัสถึงโนดสุดท้ายในสแตคอื่นที่ไม่ใช่สแตคหลักของ DEC #1)

2. สั่งให้ตัวประมวลผลทุกตัวหยุดทำงานเมื่อกระบวนการถอดรหัสดำเนินมาถึงข้อใดข้อหนึ่งของกฎเกณฑ์การสิ้นสุดของขบวนการถอดรหัส ข้างต้น

หลักการเบื้องต้นของการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์คือการใช้ประโยชน์จากสแตครองให้มากที่สุดโดยการช่วยกันค้นหาทางเดินที่ดีที่สุดในแผนภูมิต้นไม้แทนรหัส โดยตัวประมวลผลหลายตัว ในบทต่อไปจะแสดงผลการจำลองการทดสอบและเปรียบเทียบผลการทดสอบกับการถอดรหัสแบบ MSA เดิม

3.5 การประยุกต์การถอดรหัสแบบ MSA ในรหัสแบบรีเคอร์ซีพซิสเต็มเมติกคอนโวลูชัน

หัวข้อนี้เป็นการนำเสนออัลกอริทึมการถอดรหัสแบบหลายสแตคชนิดดัดแปลง (A modified multiple stack algorithm, M-MSA) ในรหัสแบบรีเคอร์ซีพซิสเต็มเมติกคอนโวลูชัน (Recursive systematic convolutional codes, RSC) ซึ่งเป็นการนำเอาการถอดรหัสแบบหลายสแตคเดิม (Multiple Stack Algorithm, MSA) มาประยุกต์ใช้กับการถอดรหัสแบบทวินสแตค (Twin stack algorithm, TS) โดยนำเอาข้อดีของการถอดรหัสแบบสมบูรณ์ใน MSA โดยไม่มีการลบข้อมูลเมื่อสแตคเต็ม แต่แบ่งกลุ่มของสแตคออกเป็น 2 กลุ่มหลัก รายละเอียดอยู่ในหัวข้อที่ 2 จากผลการทดสอบเบื้องต้นพบว่า ผลลัพธ์แบบฮาร์ดในอัลกอริทึมมัลติเพิลสแตคแบบดัดแปลง ให้ประสิทธิภาพในอัตราความผิดพลาดบิตใกล้เคียงกับการถอดรหัสแบบทวินสแตคอัลกอริทึม แต่ใช้เวลาในการคำนวณเพื่อถอดรหัสน้อยกว่าที่ค่าอัตราส่วนของสัญญาณต่อสัญญาณรบกวนมีค่าต่ำๆและทดสอบในสภาวะของการคำนวณเดียวกัน และในงานวิจัยนี้อยู่ในระหว่างการดำเนินการประยุกต์ผลลัพธ์แบบฮาร์ดที่ได้ ไปสร้างเป็นผลลัพธ์แบบซอฟท์ เพื่อนำไปใช้ในการถอดรหัสแบบวนซ้ำในรหัสเทอร์โบ

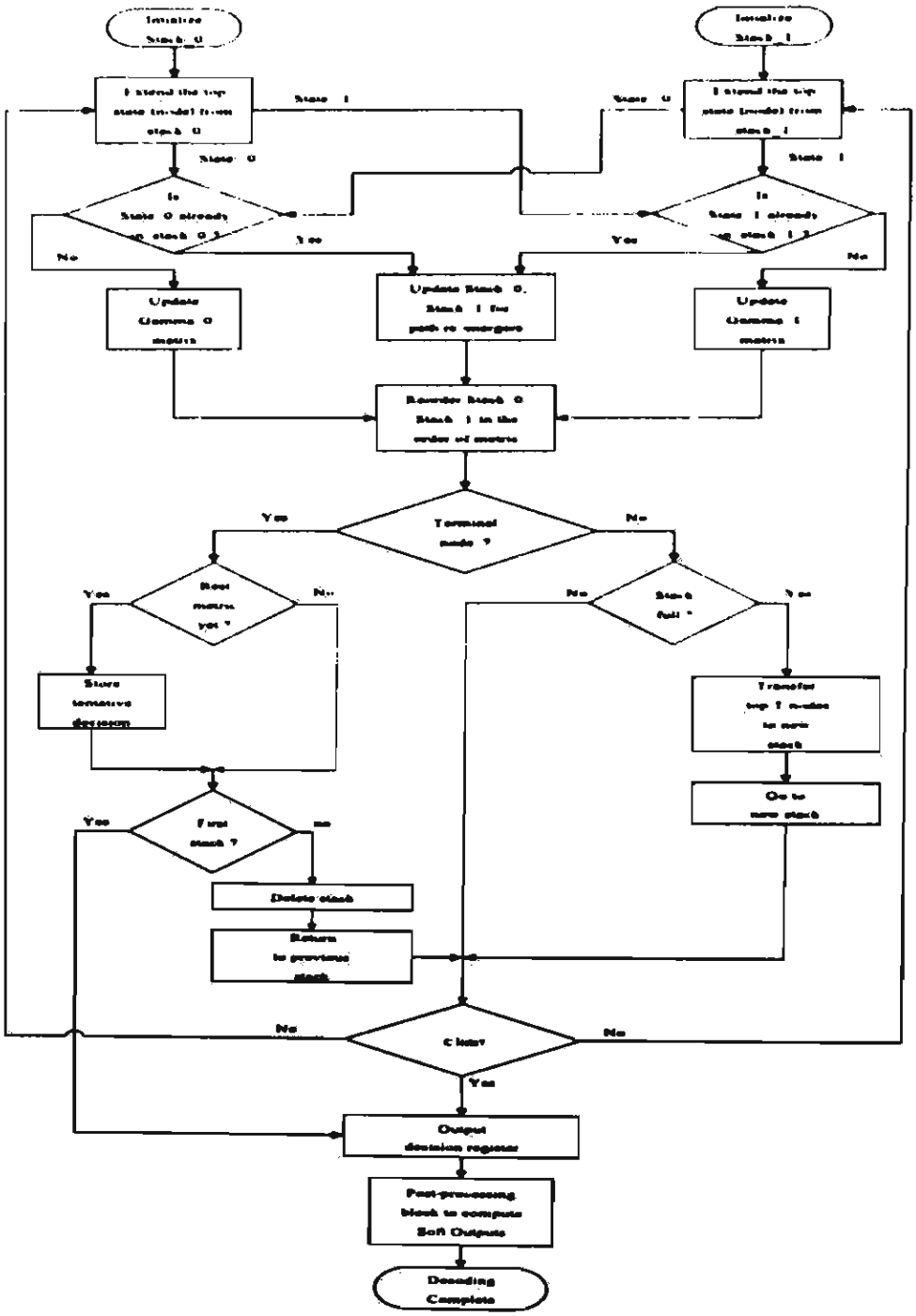
ขั้นตอนการถอดรหัสโดยใช้อัลกอริทึม M-MSAเป็นไปตามแผนภูมิในภาพที่ 3.5 ซึ่งมีรายละเอียดดังนี้ คือ จำนวนสแตคทั้งหมดในการถอดรหัสจะแบ่งออกเป็น 2 กลุ่มใหญ่ และในแต่ละกลุ่มใหญ่จะกำหนดขนาดของสแตคออกเป็น 2 ขนาดย่อย คือ ขนาดสแตคหลักหรือสแตคแรกและขนาดของสแตครองหรือสแตคอันดับสูงซึ่งจะมีจำนวนหลายสแตค โดยที่ขนาดของสแตคหลักจะโตกว่าขนาดของสแตครองมาก ในกรณีปกติของการถอดรหัสที่มีสัญญาณรบกวนน้อย การทำงานของขบวนการถอดรหัสมักจะสิ้นสุดที่สแตคหลัก จะมีเพียงบางครั้งเท่านั้นที่สแตครองจะถูกนำมาใช้ นั่นคือในกรณีที่ช่องสื่อสารมีสัญญาณรบกวนมาก จะใช้ สแตคเริ่มต้นซึ่งเป็นสแตคหลัก สองสแตคคือ สแตค$_0$ และ สแตค$_1$ ซึ่งแสตคทั้งสองนี้จะมีคุณสมบัติเหมือนกับตัวถอดรหัสแบบ SSA เพียงแต่ว่า สแตค$_0$ ใช้เก็บข้อมูลของทุกสเตท (State) ที่มีเส้นทาง (Path) ที่ลงท้ายด้วยศูนย์ และ สแตค$_1$ ใช้เก็บข้อมูลของทุกสเตทที่มีเส้นทางที่ลงท้ายด้วยหนึ่ง เนื่องจากตัวเข้ารหัสที่ใช้เป็นแบบ RSC จึงทำให้สเตทที่เกิดขึ้นลงท้ายด้วย 0 หรือ 1 ซึ่งทำให้ สเตทนั้นอาจอยู่ในทั้ง สแตค$_0$ และ สแตค$_1$ พร้อมกันได้ที่ระดับความลึกเดียวกัน แต่ถ้าหากสเตทเกิดซ้ำกันในสแตคเดียวกัน ตัวถอดรหัสจะเลือกเก็บสเตทที่มีค่าเมตริกซ์ (Metric) มากกว่าไว้ และลบเสตทที่มีค่าเมตริกซ์น้อยกว่า ซึ่งเมตริกซ์ที่ใช้ในการถอดรหัสนี้ใช้เฟโนเมตริกซ์ (Fano metric) สำหรับอัตราการเข้ารหัส $R = b/c$ และสัญญาณที่รับได้ $y_k = \{y_k^{(j)}\}$, $j = 1,\ldots,c$ ดังแสดงในสมการที่ (3.3)

$$\log[\mu_{APP}(K)] = \sum_{j=1}^{c} \log\left(\frac{\Pr(y_k^{(j)} / v_k^{(j)})}{\Pr(y_k^{(j)})}\right) + \sum_{j=1}^{b} \log \Pr(d_k^j) \qquad (3.3)$$

เมื่อ $P(y_k^{(j)} / v_k^{(j)})$ หมายถึงความน่าจะเป็นที่รับสัญญาณ y_k ตัวที่ j ได้เมื่อส่งสัญญาณ v_k ตัวที่ j และในเทอมที่สองของสมการคือ ค่า APP (A posteriori probability) ของบิตข้อมูลและจะมีการเก็บค่าแกม

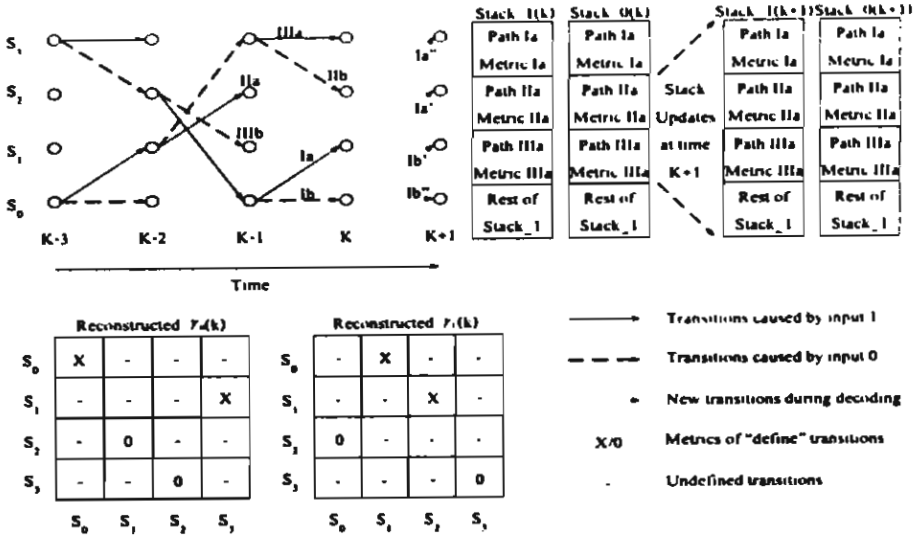
ม่าของบิตที่ k (γ_k) ตามการอัพเดทเส้นทางในสแตคไปด้วย ดังภาพที่ 3.6 ซึ่งค่าแกมม่านี้ใช้ในกระบวนการสร้างผลลัพธ์แบบซอฟท์ไปด้วย ดังสมการที่ (3.4)

$$\gamma_k(m', m) = \sum_{i} P_k(m \mid m') \cdot Q_k(X \mid m'; m) \cdot R_k(Y_k \mid X) \qquad (3.4)$$



ภาพที่ 3.5 แผนภาพขั้นตอนการทำงานของตัวถอดรหัสแบบ M-MSA

เมื่อ $P_t(m|m')$ คือ ความน่าจะเป็นที่สเตท m' ใดๆ จะเปลี่ยนเป็นสเตท m, $Q_t(X|m':m)$ คือ ความน่าจะเป็นที่จะเกิดผลลัพธ์ X เมื่อสเตท m' ใดๆ จะเปลี่ยนเป็นสเตท m และ $R_t(Y_i|X)$ หมายถึงความน่าจะเป็นที่รับสัญญาณ Y_i ได้เมื่อส่งสัญญาณ X



ภาพที่ 3.6 กระบวนการทำงานของ M-MSA และกระบวนสร้างแกมม่าที่เวลา k

ในกรณีที่สแตคหลักเต็ม จำนวนโนดข้อมูลที่มีค่าเมตริกซ์สูงสุดจำนวนหนึ่งจะถูกย้ายไปยังสแตครอง จากนั้นกระบวนการถอดรหัสจะดำเนินต่อไปในสแตครองโดยใช้จำนวนโนดที่ย้ายมา ถ้าการถอดรหัสถึงโนดสุดท้ายของแผนภูมิต้นไม้แทนรหัสก่อนที่สแตครองเต็ม เส้นทางเดินจากโนดเริ่มต้นจนถึงโนดสุดท้ายจะถูกเก็บไว้ในรีจิสเตอร์พิเศษ (Special register) และเรียกทางเดินที่เก็บไว้ว่า การตัดสินใจเบื้องต้น (Tentative decision, TD) หลังจากนั้นตัวถอดรหัสจะลบโนดที่เหลือในสแตครองและกลับมาเริ่มกระบวนการถอดรหัสอีกครั้งในสแตคหลักโดยเริ่มจากโนดที่อยู่บนสุด (ดีที่สุด) ในตอนนี้สแตคหลักจะมีเนื้อที่ว่างเท่ากับจำนวนโนดที่ถูกย้ายในตอนแรก ถ้าการถอดรหัสดำเนินมาถึงโนดสุดท้ายของแผนภูมิต้นไม้แทนรหัส ก่อนที่สแตคหลักจะเต็มอีกครั้งตัวถอดรหัสจะเปรียบเทียบค่าเมตริกซ์ของเส้นทางเดินใหม่นี้กับเส้นทางเดิน TD และเลือกทางเดินที่มีค่าเมตริกซ์ที่ดีกว่าเป็นเส้นทางการถอดรหัส อย่างไรก็ตามถ้าสแตคหลักเต็มอีกครั้งสแตครองก็จะถูกสร้างขึ้นมาใหม่และมีการย้ายโนดข้อมูลในลักษณะเดิม การสร้างสแตครองก็จะเกิดขึ้นในลักษณะเช่นนี้ต่อไปเรื่อยๆ จนกระทั่งถึงเงื่อนไขของการสิ้นสุดการถอดรหัส ในกรณีของเมื่อค่าคำนวณของการถอดรหัสถึงค่าลิมิตของการคำนวณ TD ก็จะถูกเลือกให้เป็นเส้นทางการถอดรหัสสุดท้าย

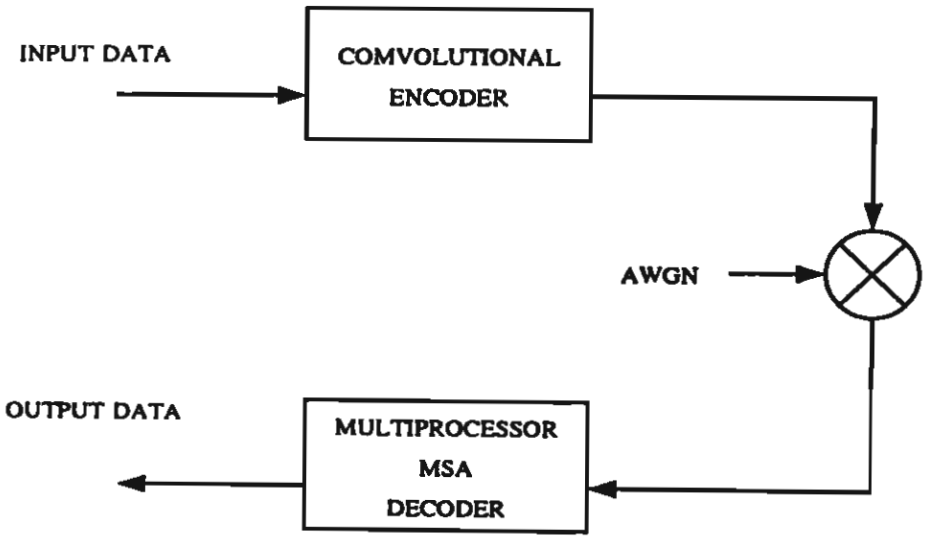
บทที่ 4
การทดสอบและผลการทดสอบ

4.1 บทนำ

บทนี้เป็นการนำเสนอผลการจำลองระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ ตามแบบโครงสร้างที่ได้ศึกษาในบทที่ 3 เนื้อหาภายในบทนี้เริ่มจากการวิเคราะห์พารามิเตอร์ที่เหมาะสม เริ่มจากการหาจำนวนโนดที่เหมาะสมที่ย้ายระหว่างตัวประมวลผล (T') ซึ่ง T' เป็นพารามิเตอร์ตัวใหม่ที่แตกต่างจากระบบการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว และได้ทดสอบพารามิเตอร์อื่น ๆ ที่มีผลต่อการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ เพื่อเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว

4.2 การจำลองระบบสื่อสารและสมมุติฐานที่ใช้ในการทดสอบ

ในการทดสอบจะใช้แบบจำลองระบบตามภาพที่ 4.1 ที่มีการเข้ารหัสสัญญาณแบบคอนโวลูชัน อัตราการเข้ารหัส 1/2 ผ่านช่องสื่อสารเป็นแบบ AWGN (Additive White Gaussian noise) และนำสัญญาณมาถอดรหัสที่ภาครับสัญญาณด้วยการถอดรหัสแบบซีเควนเชียลโดยใช้อัลกอริทึมแบบ MSA ที่ใช้ตัวประมวลผลเดียวและ อัลกอริทึมแบบ MSA โดยมัลติโพรเซสเซอร์



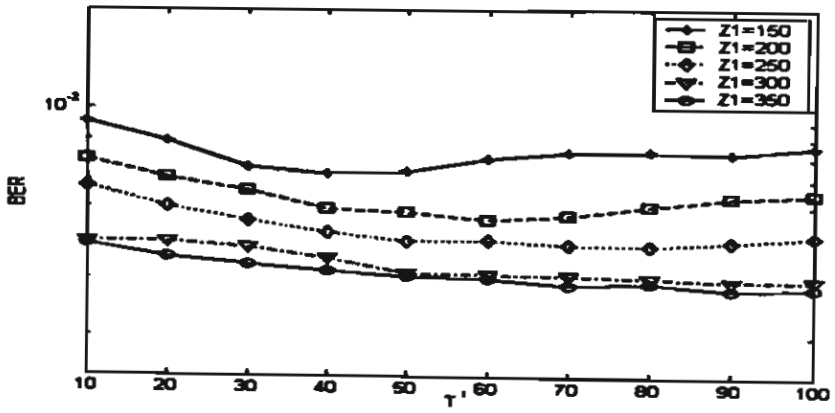
ภาพที่ 4.1 บล็อกไดอะแกรมการเข้ารหัสและถอดรหัสสำหรับการทดสอบ

เพื่อทดสอบประสิทธิภาพของหลักการที่ได้นำเสนอในงานวิจัยนี้ ผู้วิจัยได้จำลองระบบโดยการเขียนโปรแกรมคอมพิวเตอร์ภาษาซี บนระบบปฏิบัติการลินุกซ์ พารามิเตอร์ที่ใช้ในการจำลองที่สำคัญมีดังต่อไปนี้คือ ช่องสื่อสารเป็นแบบ AWGN (Additive White Gaussian noise) อัตราการเข้ารหัสคอนโวลูชันเป็น 1/2 ค่าของความยาวคอนสเตรนท์ของตัวเข้ารหัสมี 2 ค่า คือ $m = 5$ โดยมีเมตริกซ์ของตัวกำเนิด (Generator matrices) $G_1 = 110101$, $G_2 = 101111$ และ $m = 7$ ซึ่งใช้ Generator matrices $G_1 = 10111101$, $G_2 = 11001011$ จำนวนบิตต่อเฟรม K เป็น 60 บิต 120 บิตและ 180 บิตซึ่งมีจำนวนบิตทดสอบรวมเป็น 1,200,000 บิตเท่ากันในทุกกรณี $C_{\lim} = 2500$ จำนวนโนดที่ย้ายระหว่างสแตคหลักและสแตครองภายในแต่ละตัวประมวลผล T เป็น 2 ขนาดของสแตครอง Z เป็น 12 จำนวนสแตครองของระบบตัวประมวลเดียว n_z เป็น 300 และของตัวประมวลผลหลายตัว n_{ZP} เป็นไปตามสมการที่ (3.1)

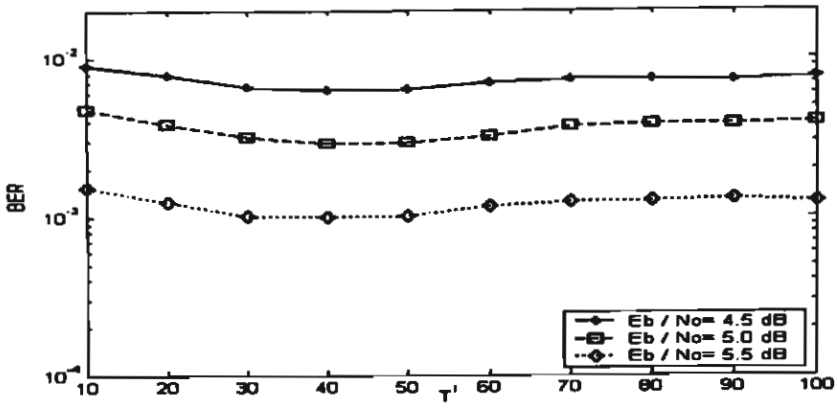
4.3 ผลการทดสอบประสิทธิภาพกับค่าพารามิเตอร์ต่าง ๆ

4.3.1 ผลจำนวนโนดที่ย้ายระหว่างตัวประมวลผล (T')

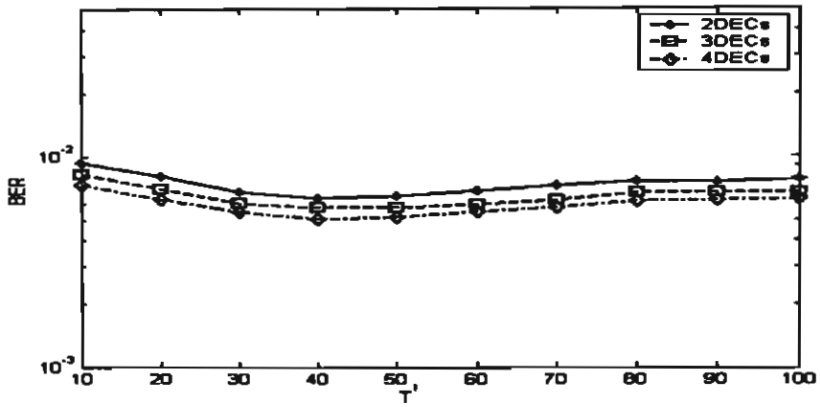
การทดสอบหาจำนวนโนดที่เหมาะสมที่ย้ายระหว่างตัวประมวลผลสำหรับงานวิจัย ผลการทดสอบ ภาพที่ 4.2 แสดงค่าอัตราความผิดพลาดบิต (Bit error rate, BER) เทียบกับจำนวนโนดที่ย้ายจาก สแตคหลักของ DEC #1 ไปยังสแตคหลักของ DEC # j ($j = 2,\ldots,n$) เมื่อกำหนดค่า $E_b / N_o = 4.5$ dB จะเห็นว่าค่า T' ที่ให้ค่า BER ต่ำที่สุดจะมากขึ้นตามค่าของ Z_1 (สังเกตจากกราฟแต่ละเส้นที่ให้ค่า BER ต่ำสุด) และเมื่อกำหนดให้ขนาดของสแตคหลักคงที่ที่ค่า $Z_1 = 150$ โดยปรับค่า E_b / N_o จาก 4.5-5.5 dB ให้ค่า BER เมื่อใช้ตัวประมวลผล 2 ตัวดังแสดงในภาพที่ 4.3 สำหรับภาพที่ 4.4 เป็นผลการจำลองเพื่อดูค่า BER เมื่อใช้ตัวประมวลผล 2 ตัว 3 ตัว และ 4 ตัว โดยกำหนดให้ $Z_1 = 150$ คงที่เช่นเดิม ภาพที่ 4.5 แสดงผลอัตราความผิดพลาดบิตที่ค่า m และ K ต่างๆกันจะเห็นว่าเมื่อใช้จำนวน m และ K มากขึ้นค่า T' ที่ทำให้ BER ต่ำที่สุดมีค่าลดลง จากผลการทดสอบข้างต้นจะเห็นว่าที่ค่า $Z_1 = 150$ ค่า T' ที่เหมาะสมคือค่าตั้งแต่ 40 ถึง 50 ดังนั้นจึงได้เลือกค่า $Z_1 = 150$ สำหรับ $m = 5$ และ $T' = 50$ เพื่อใช้ทดสอบประสิทธิภาพในบทความนี้



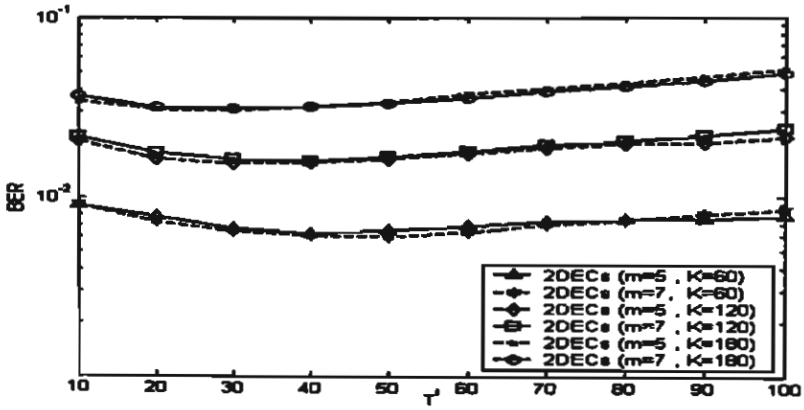
ภาพที่ 4.2 ค่า BER กับ T' เมื่อเปลี่ยนจำนวน Z_1



ภาพที่ 4.3 ค่า BER กับ T' ที่ค่า E_b / N_o ต่างๆกัน



ภาพที่ 4.4 ค่า BER กับ T' เมื่อใช้จำนวนตัวประมวลผลต่างกัน



ภาพที่ 4.5 ค่า BER กับ T' เมื่อใช้ ค่าของ m และ K ต่างกัน

ตารางที่ 4.1

จำนวนบล็อกที่ถอดรหัสเสร็จในสแตคหลักของ DEC# 1

จำนวนตัวประมวลผล	จำนวนบล็อกที่ถอดรหัสเสร็จในสแตคหลักของ DEC #1					
	$T = 20$	$T = 30$	$T = 40$	$T = 50$	$T = 60$	$T = 70$
1 DEC	18,507*					
2 DECs	18,548	18,578	18,582	18,639	18,682	18,745
3 DECs	18,569	18,635	18,692	18,763	18,832	18,903
4 DECs	18,594	18,691	18,790	18,895	18,915	19,045

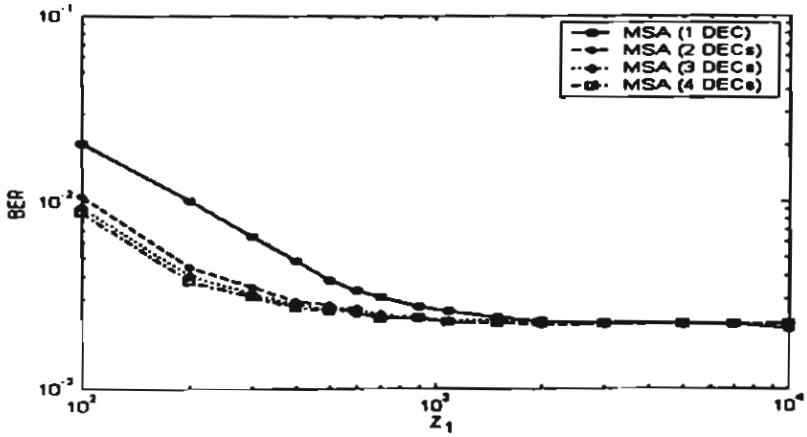
* ตัวประมวลผลตัวเดียวไม่มีการย้ายโนดข้อมูลระหว่างตัวถอดรหัส

ตารางที่ 4.1 แสดงผลจำนวนบล็อกของการถอดรหัสซึ่งสิ้นสุดภายในสแตคหลักของตัวประมวลผลตัวแรก การเพิ่ม T จะทำให้ได้จำนวนบล็อกของการถอดรหัสที่สิ้นสุดในสแตคหลักของ DEC#1 เพิ่มมากขึ้นด้วย แต่เส้นทางเดินของการถอดรหัสที่การถอดรหัสที่สิ้นสุดภายในสแตคหลักของ DEC#1 (หลังการย้าย T) นั้นอาจจะไม่ได้เป็นเส้นทางเดินของการถอดรหัสที่มีค่าแมตริกซ์ที่ดีที่สุด

4.3.2 การทดสอบผลของขนาดสแตคหลัก (Z_1) ต่อประสิทธิภาพ

โดยหลักการแล้วประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ระบบตัวประมวลผลเดียวจะสามารถทำให้ดีขึ้นได้โดยการเพิ่มขนาดของสแตคหลัก ในงานวิจัยนี้ได้ทดสอบผลของขนาดของ สแตคหลักสำหรับการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ และเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับค่าที่ได้จากการถอดรหัสแบบ MSA ระบบตัวประมวลผลเดียว จากผลการทดลองการเพิ่มขนาดของ Z_1 มีผลต่ออัตราความผิดพลาดบิตเช่นเดียวกันทั้งสองระบบ ภาพที่ 4.6 เป็นผลการทดสอบประสิทธิภาพ BER เมื่อเปลี่ยน Z_1 ค่าต่าง ๆ ที่ค่า E_b / N_o เป็น 4.5 dB เมื่อเพิ่มขนาดของสแตคหลักขึ้นเรื่อย ๆค่าอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบจะลดลงเรื่อย ๆตามลำดับเช่นกัน ด้วยเหตุผลเป็นไปตามสมการที่ (2.11) จนกระทั่งการเพิ่มขนาดสแตคหลักจะไม่มีผลต่ออัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบ เมื่อเปรียบเทียบทั้งสองระบบจะเห็นว่าค่า BER ของการถอดรหัสระบบ MSA ซึ่งใช้ตัวประมวลผลตัวเดียวจะเริ่มคงที่เมื่อขนาด Z_1 เป็น 2000 และได้ค่า BER เท่ากับ $2.28 \cdot 10^{-3}$ ในขณะที่การถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ จะมีค่าคงที่เมื่อ Z_1 เป็น 1100 และได้ค่า BER เท่ากับ $2.24 \cdot 10^{-3}$ แสดงให้เห็นว่าการใช้จำนวนตัวประมวลผลมากขึ้นสามารถให้ค่าอัตราความผิดพลาดบิตที่เท่ากับหรือใกล้เคียงกับค่าอัตราความผิดพลาดบิตที่ได้จากการถอดรหัสแบบMSA ใช้ตัวประมวลผลตัวเดียว โดยใช้ขนาดของสแตคหลักเล็กกว่า แต่อย่างไรก็ตามการเพิ่มจำนวนตัวประมวลผลในระบบการถอดรหัส MSA โดยมัลติโพรเซสเซอร์ จะ

มีผลต่ออัตราความผิดพลาดบิตน้อยลงเมื่อขนาดของ Z_1 มากกว่า 700 และการเพิ่มขนาดสแตคหลักจะไม่มีผลต่ออัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA ทั้งสองระบบ เมื่อค่า Z_1 มากว่า 2000 ซึ่งมีค่า BER ประมาณ $2.20*10^{-3}$ เนื่องจากการถอดรหัสแบบ MSA โดยระบบมัลติโพรเซสเซอร์สามารถถอดรหัสจำนวนบล็อกข้อมูลเสร็จภายในสแตคหลักของตัวประมวลผลตัวแรก DEC #1 ได้เพิ่มขึ้น ทำให้จำนวนบล็อกที่ถูกถอดรหัสภายใน สแตครองมีจำนวนน้อยลงและจะเป็นบล็อกที่มีสัญญาณรบกวนมาก ๆ เท่านั้น



ภาพที่ 4.6 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน Z_1

4.3.3 การทดสอบผลค่าลิมิตของการคำนวณ (C_{lim}) ต่อประสิทธิภาพ

การถอดรหัสแบบซีเควนเชียลจะมีความแปรปรวนในการคำนวณ (Computational variability) คือค่าของการคำนวณของการถอดรหัสจะขึ้นอยู่กับปริมาณของสัญญาณรบกวน สำหรับการถอดรหัสแบบ MSA นั้นการกำหนดค่าลิมิตของการคำนวณเพื่อให้การถอดรหัสไม่มีอิเรชัว จะต้องกำหนดให้ C_{lim} มีค่ามากกว่าค่าวิกฤตต่ำสุดของการคำนวณ C_{crit} ตามสมการที่ 2.6 การทดสอบหาต่ำสุดของการคำนวณ C_{min} ของระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์เพื่อให้การถอดรหัสไม่มีอิเรชัวนั้น แสดงตัวอย่างในตารางที่ 4.2 ผลการทดสอบค่า C_{min} ของการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียวเท่ากับ 569 ในขณะที่ค่า C_{crit} ที่ได้จากการคำนวณตามสมการที่ 2.6 เท่ากับ 897 ซึ่งการที่ค่า C_{min} ต่ำกว่าค่า C_{crit} เนื่องจากการกำหนดให้ขนาดของสแตครองมีขนาดเล็กกว่าขนาดของสแตคหลักมาก ๆ (Chevillat และ Costello ,1976) และการกำหนดค่า T เป็น 2 ผลการทดสอบเมื่อใช้ตัวประมวลผลหลายตัว ค่า C_{min} ของการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ มีค่าเท่ากับ 640, 655 และ 655 เมื่อใช้ตัวประมวลผล 2 ,3 และ 4 ตัวตามลำดับ การที่ค่า C_{min} ของระบบโดยมัลติโพรเซสเซอร์มีค่ามากกว่าค่า C_{min} ของการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียว เนื่องจากการย้ายจำนวนโนดระหว่างตัวประมวลผล T' ทำให้สแตคหลักของตัวประมวลผลตัวแรกมีเนื้อที่ในการคำนวณเพิ่มขึ้นเท่ากับ T' แต่ค่า C_{min} จะไม่เปลี่ยนแปลงมากนักเมื่อเพิ่มจำนวนตัวประมวลผลเนื่องจากการถอดรหัสสามารถถึงค่า *การตัดสินใจเบื้อง*

ต้นอันดับแรก (First Tentative decision,TD) ภายในสแตคแรกของตัวประมวลผลอันดับก่อนหน้า ภาพที่ 4.7 แสดงค่าอัตราความผิดพลาดบิตเนื่องจากผลการเพิ่มค่า C_{lim} ขึ้นเรื่อย ๆ เมื่อเพิ่มค่า C_{lim} ค่าอัตราความผิดพลาดบิตจะมีค่าลดลง เนื่องจากการเพิ่มค่า C_{lim} จะทำให้โอกาสที่การถอดรหัสจะถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัสมากขึ้น และที่ค่า C_{lim} เท่ากัน และการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้ค่าอัตราความผิดพลาดบิตที่ต่ำกว่า ในภาพที่ 4.8 แสดงจำนวนครั้งเฉลี่ยที่กระบวนการถอดรหัสถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัสเทียบกับค่า C_{lim} จะเห็นว่าการประยุกต์ใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้จำนวนครั้งที่ถึงโนดสุดท้ายได้มากกว่า MSA ที่ใช้ตัวประมวลผลเดียว และนั่นหมายถึงโอกาสที่จะได้ทางเดินของการถอดรหัสที่ดีกว่าเมื่อใช้ตัวประมวลผลหลายตัว อย่างไรก็ตามจำนวนครั้งของการถึงโนดสุดท้ายในภาพที่ 4.8 ไม่ได้หมายถึงจำนวนครั้งของการบันทึกค่าใหม่ของ TD เสมอไป ภาพที่ 4.9 ได้แสดงให้เห็นถึงจำนวนครั้งเฉลี่ยที่มีการบันทึกค่าทางเดินใหม่ใน TD ซึ่งผลที่ได้สอดคล้องกับภาพที่ 4.8 นั่นคือระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ให้ค่าเฉลี่ยของจำนวนครั้งในการบันทึกค่าใหม่ที่สูงขึ้นและจะสูงกว่าเดิมเมื่อเพิ่มจำนวนตัวประมวลผลในการถอดรหัสมากขึ้น

ตารางที่ 4.2

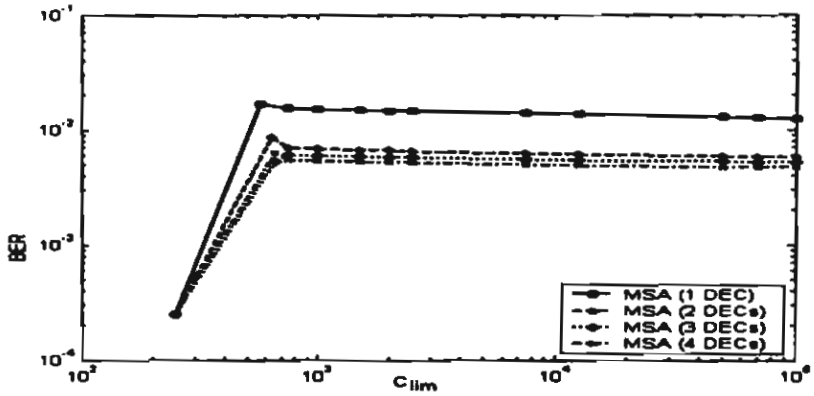
ค่าจำนวนการคำนวณต่ำสุดที่ทำให้การถอดรหัสสมบูรณ์ (erasure free)

จำนวนตัวประมวลผล	$Z_1 = 150$	$Z_1 = 300$
	C_{min}	C_{min}
1 DEC	569	855
2 DECs	640	942
3 DECs	655	956
4 DECs	655	954

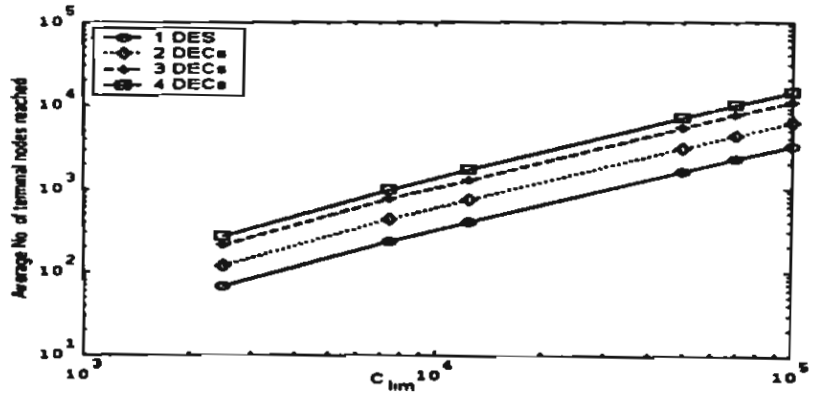
4.3.4 การทดสอบผลของขนาดของสแตครอง (Z) ต่อประสิทธิภาพ

ขนาดของสแตครองเป็นปัจจัยหนึ่งที่มีผลต่อการถอดรหัสแบบ MSA ซึ่งการกำหนดขนาดของ สแตครองจะมีผลต่ออิเรชัวได้คือถ้ากำหนดขนาดของสแตครองให้มีขนาดเล็กกว่าขนาดของสแตคหลักมาก ๆ จะทำให้การถอดรหัสสามารถถึงโนดสุดท้ายของเส้นทางตามแผนภูมิต้นไม้รหัสเร็วขึ้น และมีค่าของการคำนวณที่ต่ำลงด้วย ภาพที่ 4.10 แสดงค่าอัตราความผิดพลาดบิตเมื่อใช้ระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์เมื่อทำการเปลี่ยนขนาดของสแตครองผลการทดสอบแสดงให้เห็นว่ามีอัตราความผิดพลาดบิตที่ต่ำกว่าระบบการถอดรหัสแบบ MSA เดิม แต่การเปลี่ยนขนาดของสแตครองมีผล

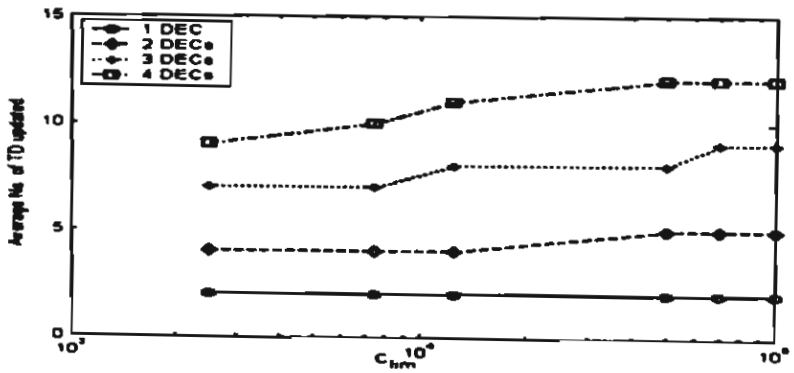
ต่ออัตราความผิดพลาดบิตที่น้อยมากทั้งสองระบบ และเมื่อเปรียบเทียบกับระบบ MSA เดิม ที่ขนาดของสแตคหลักเป็นสองเท่า (300) ค่าอัตราความผิดพลาดบิตที่ได้จากระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ก็ยังมีอัตราความผิดพลาดบิตที่ต่ำกว่าซึ่งแสดงว่าการเพิ่มจำนวนตัวประมวลผลให้ประสิทธิภาพที่ดีกว่าการเพิ่มขนาดของสแตคหลักในขณะที่ปัจจัยอื่น ๆเหมือนเดิม



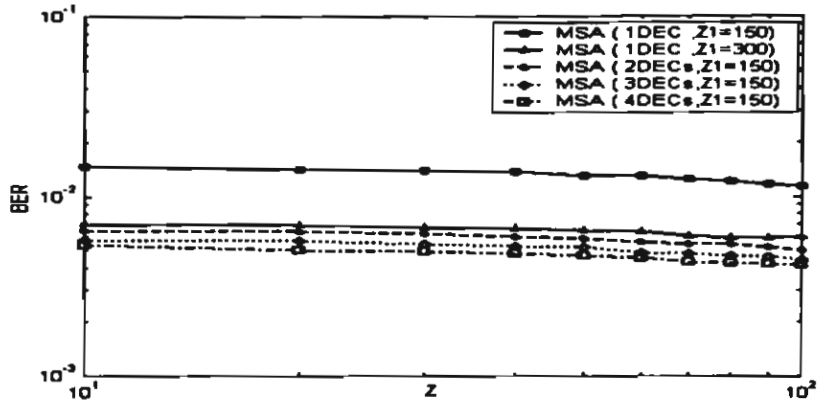
ภาพที่ 4.7 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน C_{lim}



ภาพที่ 4.8 จำนวนครั้งเฉลี่ยที่การถอดรหัสถึงโนดสุดท้ายในแผนภูมิต้นไม้แทนรหัส



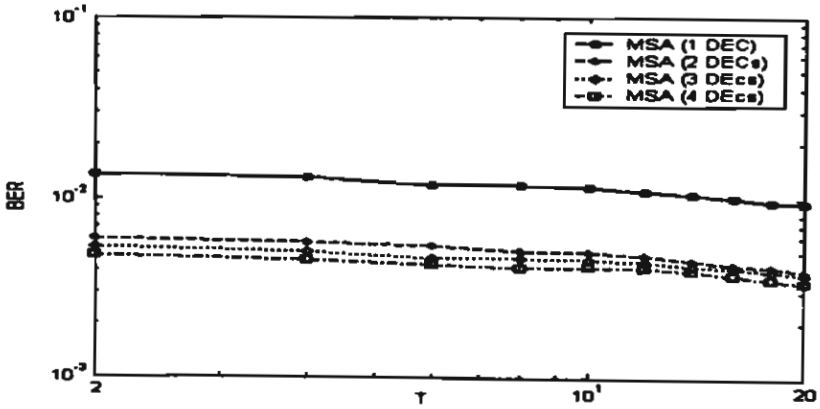
ภาพที่ 4.9 จำนวนครั้งเฉลี่ยที่มีการบันทึกค่า TD ใหม่



ภาพที่ 4.10 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน Z

4.3.5 การทดสอบผลจำนวนโนด (T) ต่อประสิทธิภาพ

จำนวนโนดที่ย้ายระหว่างสแตคเป็นอีกปัจจัยหนึ่งที่มีผลต่อการถอดรหัสแบบ MSA ซึ่งการกำหนดจำนวนโนดจะมีผลต่อจำนวนสแตครองและมีผลต่ออิเรชัวคือถ้ากำหนดจำนวนโนดข้อมูลที่ถูกย้ายมากจำนวนสแตครองที่ใช้ในการถอดรหัสก็เพิ่มมากขึ้น(ในขณะที่ขนาดของสแตครองคงที่) และทำให้การถอดรหัสสามารถถึงโนดสุดท้ายของเส้นทางตามแผนภูมิต้นไม้รหัสช้าและมีค่าของการคำนวณเพิ่มขึ้นด้วย ภาพที่ 4.11 แสดงค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวนโนดข้อมูลที่ถูกย้าย ในขณะที่ T เท่ากันระบบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์จะมีค่าอัตราความผิดพลาดบิตที่ต่ำกว่าระบบการถอดรหัสแบบ MSA เดิม และการเพิ่มจำนวนโนดข้อมูลที่ย้ายระหว่างสแตคของตัวประมวลผลแต่ละตัวจะสามารถลดค่าอัตราความผิดพลาดบิตด้วย



ภาพที่ 4.11 ค่าอัตราความผิดพลาดบิตเมื่อเปลี่ยนจำนวน T

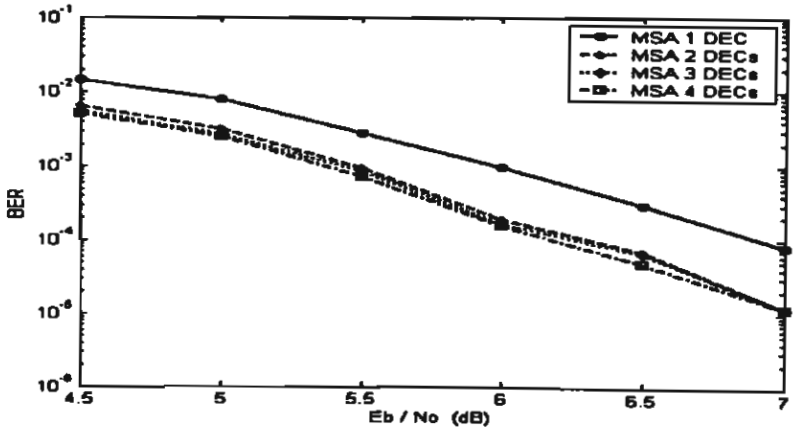
4.3.6 ค่าอัตราความผิดพลาดบิตที่ค่าสัญญาณรบกวนต่าง ๆ

ผลการทดสอบการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ที่ค่าสัญญาณรบกวนค่าต่าง ๆเพื่อเปรียบเทียบประสิทธิภาพในอัตราความผิดพลาดบิตกับการถอดรหัสแบบ MSA ที่ใช้ตัวประมวลผลเดียวภาพที่ 4.12 เป็นการทดสอบประสิทธิภาพของ BER เมื่อเปลี่ยนค่า E_b / N_o ในช่วง $4.5-7.0\,dB$ (ค่าความยาวคอน สเตรนท์ของตัวเข้ารหัส $m=5$, Generator matrices $G_1=110101$, $G_2=101111$) จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์มีค่าต่ำกว่า BER ของระบบ MSA เดิมซึ่งใช้ตัวประมวลผลตัวเดียวและค่า BER จะมีค่าลดลงต่อไปอีกเมื่อเพิ่มตัวประมวลผล

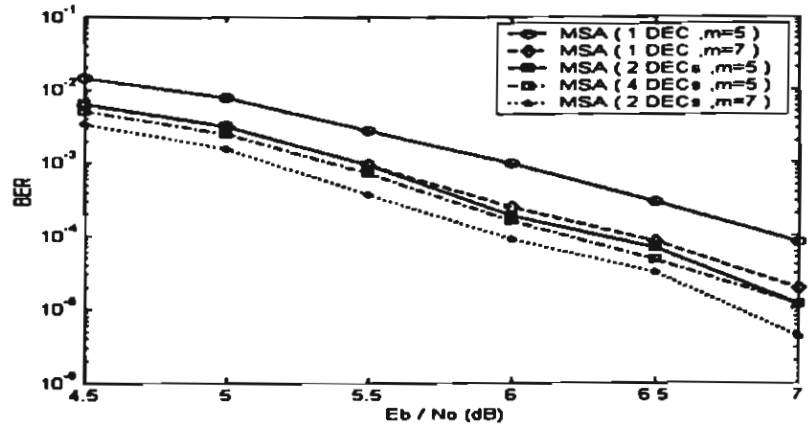
ภาพที่ 4.12 เป็นการทดสอบประสิทธิภาพของ BER เมื่อเปลี่ยนค่า E_b / N_o ในช่วง 4.5-7.0 dB จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบขนานสำหรับ MSA มีค่าต่ำกว่า BER ของระบบ MSA เดิมซึ่งใช้ตัวประมวลผลตัวเดียว อย่างไรก็ตามเมื่อเพิ่มจำนวนตัวประมวลผลเป็น 3 และ 4 ตัว ไม่ช่วยให้ประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสดีขึ้นจากเดิมมากนักในขณะที่ขนาดของ Z_1 ใหญ่มาก

ภาพที่ 4.13 เป็นการทดสอบประสิทธิภาพของ BER เมื่อค่าความยาวคอนสเตรนท์ของตัวเข้ารหัสเป็น $m=7$ Generator matrices $G_1=10111101$, $G_2=11001011$ จะเห็นว่าค่า BER ในการถอดรหัสเมื่อใช้วิธีการถอดรหัสแบบ MSA โดยมัลติโพรเซสเซอร์ ที่ค่า $m=5$ สามารถให้ค่า BER ต่ำกว่าค่า BER ที่ได้จากการถอดรหัสระบบ MSA เดิมซึ่งใช้ค่าความยาวคอนสเตรนท์มากกว่า($m=7$)

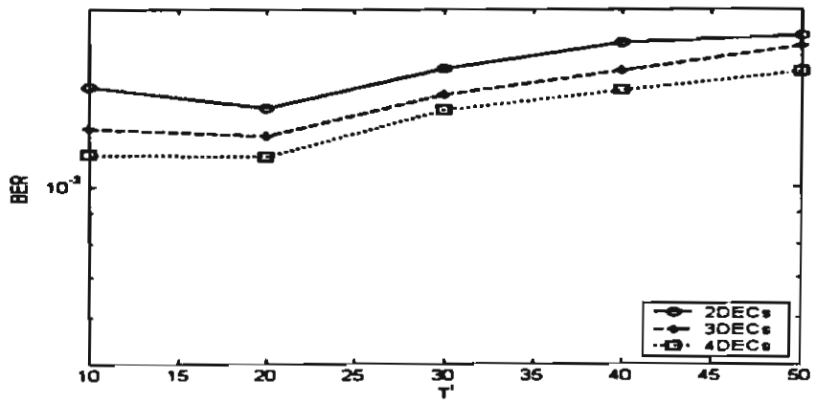
ภาพที่ 4.14 เป็นการทดสอบประสิทธิภาพของ BER ที่ค่า T' ต่าง ๆ เมื่อทดสอบในขณะที่ขนาดของสแตคหลัก $Z_1=75$ และใช้ $m=7, K=120$ จะเห็นว่าค่า T' ที่เหมาะสมคือ $T'=20$ ซึ่งให้ค่า BER ต่ำที่สุดในทุกกรณีของการถอดรหัสแบบขนานเมื่อใช้จำนวนตัวประมวลผลต่างกัน และเมื่อทดสอบที่ค่า E_b / N_o ต่าง ๆกัน ดังภาพที่ 4.15 ผลที่ได้แสดงให้เห็นว่าเมื่อเพิ่มค่า m และ K ในขณะที่ลดขนาดของ Z_1 ลงจาก 150 เป็น 75 ตัวประมวลผลที่เพิ่มขึ้นสามารถปรับปรุงประสิทธิภาพของการถอดรหัสในอัตราความผิดพลาดบิตได้เพิ่มขึ้นเล็กน้อยเมื่อเทียบกับรภาพที่ 4.12



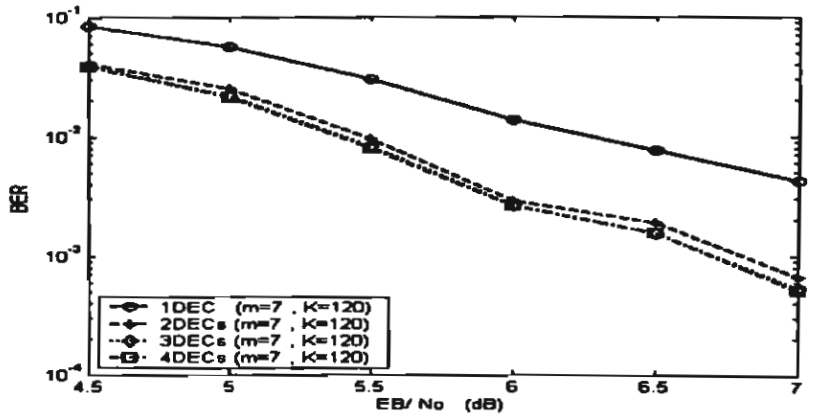
ภาพที่ 4.12 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m=5, K=60, Z_1=150, T'=50$)



ภาพที่ 4.13 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m = 5$, $m = 7$ $K = 60$, $Z_1 = 150$, $T' = 50$)



ภาพที่ 4.14 ค่า BER กับ T' เมื่อใช้ ค่าของ $m = 7$ และ $K = 120$ $Z_1 = 75$



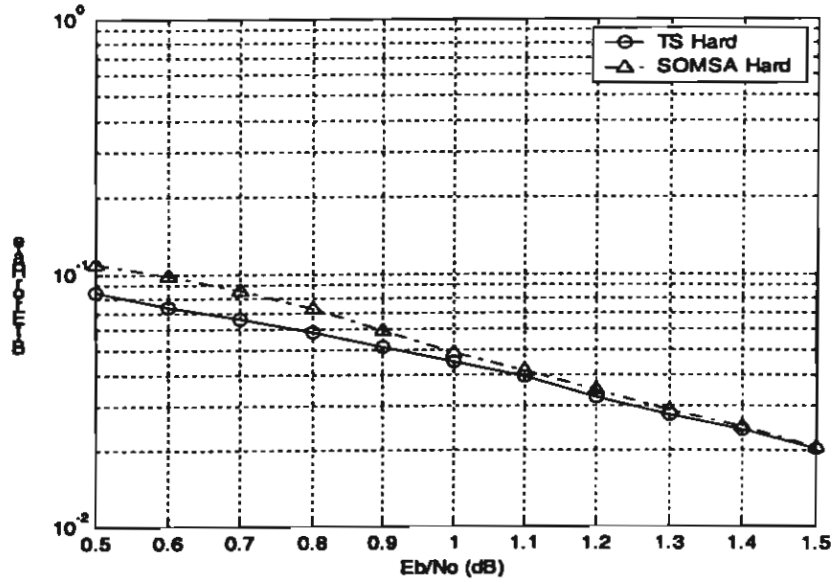
ภาพที่ 4.15 ค่า BER ที่ค่า E_b / N_o ต่างๆ ($m = 7$, $K = 120$, $Z_1 = 75$, $T' = 20$)

4.4 ผลการทดสอบประสิทธิภาพการถอดรหัสแบบ MSA ในรหัสแบบรีเคอร์ซีพซิสเต็มเมติกคอนโวลูชัน

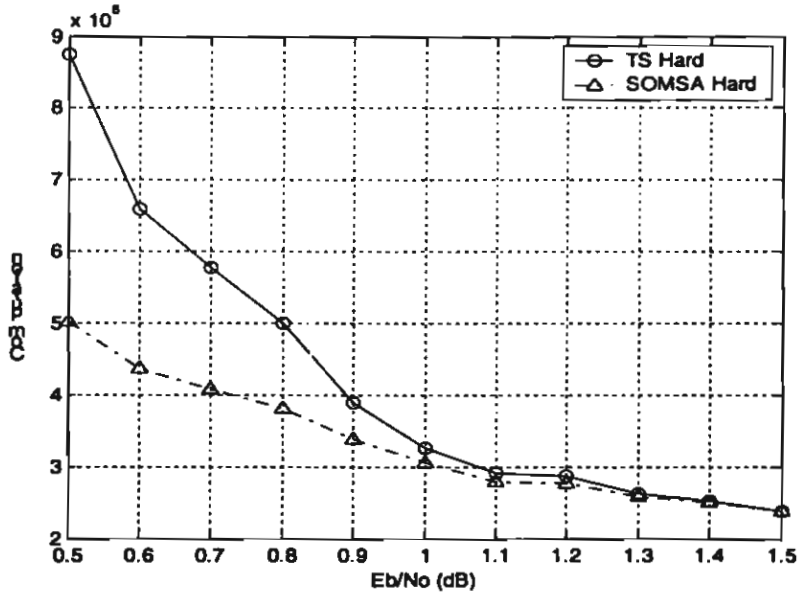
หัวข้อนี้เป็นการแสดงผลการทดสอบประสิทธิภาพของการถอดรหัสแบบ M-MSA เปรียบเทียบกับการถอดรหัสแบบ TS ซึ่งในการทดสอบจะใช้แบบจำลองระบบที่มีการเข้ารหัสสัญญาณแบบ RSC ผ่านช่องสื่อสารแบบ AWGN โดยใช้โปรแกรมคอมพิวเตอร์ภาษาซีเพื่อทำการจำลองตัวเข้ารหัส ตัวถอดรหัส และช่องสื่อสารในระบบปฏิบัติการลินุกซ์ พารามิเตอร์ที่สำคัญซึ่งใช้ในการทดสอบมีดังต่อไปนี้

- จำนวนข้อมูลที่ส่ง คือ 1,000,000 บิต โดยแบ่งออกเป็นบล็อก บล็อกละ 100 บิต
- เข้ารหัสแบบ RSC โดยมีเมตริกของตัวสร้างรหัส $G=\left[1 \quad \frac{67}{51}\right]$ และค่าความยาวคอนสเตรนท์เท่ากับ 5
- ใช้การตัดสินใจแบบซอฟท์ 8 ระดับ
- E_s / N_o จาก 0.5 ถึง 1.5 dB

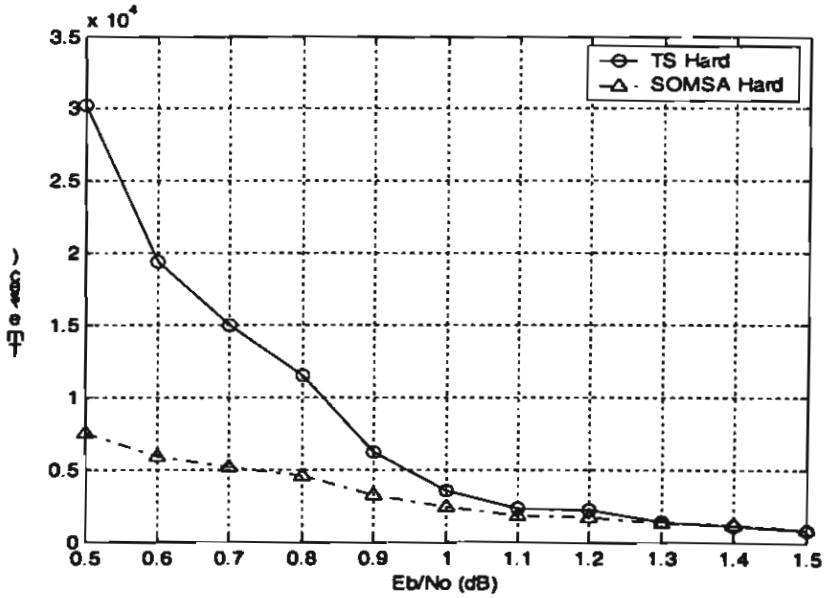
ภาพที่ 4.16 แสดงการเปรียบเทียบอัตราความผิดพลาดบิตของการถอดรหัสแบบ M-MSA และ TS จากรูปจะเห็นว่าการถอดรหัสแบบTS ให้อัตราความผิดพลาดบิตต่ำกว่าการถอดรหัสแบบ M-MSA เล็กน้อย ในช่วงที่ค่า E_s / N_o ต่ำๆ และจะค่อยๆ ใกล้เคียงกันเมื่อ E_s / N_o มีค่ามากขึ้น แต่เมื่อพิจารณาทดสอบเปรียบเทียบในจำนวนรอบของการคำนวณในการถอดรหัส และเวลาในการถอดรหัส ดังแสดงในภาพที่ 4.17 และ 4.18 พบว่าการถอดรหัสแบบ M-MSA มีจำนวนรอบของการคำนวณและเวลาในการถอดที่น้อยกว่าการถอดรหัสแบบ TS ในช่วงที่ค่า E_s / N_o ต่ำๆ และจะค่อยๆ ใกล้เคียงกันเมื่อ E_s / N_o มีค่ามากขึ้น



ภาพที่ 4.16 การเปรียบเทียบอัตราผิดพลาดบิตของการถอดรหัส



ภาพที่ 4.17 การเปรียบเทียบการคำนวณของการถอดรหัส



ภาพที่ 4.18 การเปรียบเทียบเวลาในการถอดรหัส

บทที่ 5
สรุปงานวิจัย

งานวิจัยนี้ได้นำเสนอการประยุกต์ใช้วิธีการถอดรหัสแบบขนานโดยใช้ตัวประมวลผลหลายตัวกับอัล กอริทึมการถอดรหัสแบบ MSA สำหรับรหัสคอนโวลูชัน เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของการถอดรหัสแบบ MSA เดิมให้ดีขึ้น ซึ่งปัญหาในการถอดรหัสแบบ MSA เดิมนั้น คือ บล็อกที่มีปริมาณของสัญญาณรบกวนมากไม่สามารถสิ้นสุดกระบวนการถอดรหัสในสแตคหลักได้และเมื่อทำการย้ายโนดบางส่วนในสแตคหลักมาทำการถอดรหัสต่อในสแตครอง โอกาสที่การถอดรหัสจะสามารถกลับมาสิ้นสุดที่แสตคหลักนั้นมีค่าน้อยมาก ดังนั้นการถอดรหัสมักจะสิ้นสุดโดยค่าของ C_{lim} ซึ่งให้ค่าอัตราความผิดพลาดบิตสูง ในงานวิจัยนี้ได้ทำการประยุกต์การถอดรหัสแบบขนานกับ MSA เพื่อปรับปรุงประสิทธิภาพในอัตราความผิดพลาดบิตของ MSA เดิมให้ดีขึ้น ที่ค่า C_{lim} และปริมาณหน่วยความจำที่เท่ากัน ผลการจำลองด้วยคอมพิวเตอร์ ปรากฏว่าการถอดรหัสแบบขนานโดยใช้ตัวประมวลผลหลายตัวสามารถทำให้ค่าอัตราความผิดพลาดบิตใน MSA ลดลงได้อย่างมาก

ต่อมาได้ทำการประยุกต์วิธีการการถอดรหัสแบบหลายสแตคชนิดดัดแปลงเพื่อใช้ถอดรหัสสัญญาณแบบรีเดอร์ซีพซิสเต็มมาติกคอนโวลูชัน โดยการประยุกต์วิธีการถอดรหัสแบบทวินสแตคมาใช้เพื่อสร้างผลลัพธ์แบบฮาร์ดและผลลัพธ์แบบซอฟท์ ผลการทดสอบประสิทธิภาพในอัลกอริทึมการถอดรหัสแบบหลายสแตคชนิดดัดแปลง พบว่า สามารถที่จะให้อัตราความผิดพลาดบิตที่ใกล้เคียงกับกรถอดรหัสแบบทวินสแตค แต่มีข้อที่น่าสนใจคือ เวลาที่ใช้ในการถอดรหัสและจำนวนรอบในการคำนวณในกระบวนการถอดรหัส โดยเฉลี่ยมีค่าลดลงอย่างมากสำหรับค่า E_b/N_o ต่ำๆ ปัจจุบันงานวิจัยนี้กำลังอยู่ในระหว่างการดำเนินการสร้างผลลัพธ์แบบซอฟท์โดยใช้ผลลัพธ์แบบฮาร์ดที่ได้ เพื่อที่จะนำไปประยุกต์ใช้ในวิธีการถอดรหัสแบบวนซ้ำสำหรับถอดรหัสแบบเทอร์โบต่อไป

เอกสารอ้างอิง

Forney, G.D. 1974. Convolutional Codes III: Sequential Decoding. Inf. Control, Vol. 25, pp. 267-297

Lin, S., and Costello, D. J. Jr. 1983. Error Control Coding: Fundamentals and Applications. Prentice-Hall

Haccoun, D., and Begin, G. 1989. High-rate punctured convolutional codes for Viterbi and sequential decoding. IEEE Trans. Commun. Vol. 37 No. 11, pp. 1113-1125

Forney, Jr., G.D., and Bower, E.K. 1971. A High-Speed Sequential Decoder Prototype Design and Test. IEEE Transaction on Communication Technology, COM-19(5), 821-835.

Belanger, N., Haccoun, D., andSavaria, Y. 1994.A Multiprocessor Architecture for Multiple Path Sequential Decoders. IEEE Transactions on Communications, 42(2/3/4), 951-957.

Chivillat, P.R., and Costello, Jr., D.J. 1976. A Multiple Stack Algorithm for Erasurefree Decoding of Convolutional Codes, IEEE Transactions on Commun., COM-25(12), 1460-1470

Jelinek, F. 1969 A fast sequential decoding using a stack, IBM J. Res. Dev., Vol.13, pp. 675-685

Lin, Y., and Tu, S.H. 1997. Modified multiple stack algorithm for decoding convolutional codes. IEE Pro.-Commun, Vol. 44, No. 4, August, 221-228.

Kaiping, L., and Samir, K. 1999. A Bidirectional Multiple Stack Algorithm. IEEE Transactions on Communications, Vol. 47, No. 1, January, 6-9.

ภาคผนวก

Parallel Decoding Scheme for Multiple Stack Algorithm

Journal:	*IEE Proc. Communications*
Manuscript ID:	COM-2005-0036.R1
Manuscript Type:	Research Paper
Date Submitted by the Author:	25-May-2005
Keyword:	CHANNEL CODING, CONVOLUTIONAL CODES, MULTIPLE STACK ALGORITHM

Parallel Decoding Scheme for Multiple Stack Algorithm

Virasit Imtawil*, Ph.D. and Wanlop Surakampontorn**, Ph.D.

*Department of Electrical Engineering, Faculty of Engineering,

Khon Kaen University, Khonkaen, THAILAND 40002

Corresponding author email address: virasit@kku.ac.th

Department of Electronics, Faculty of Engineering,

King Mongkut's Institute of Technology Ladkrabang, Bangkok, THAILAND 10520

Abstract In order to improve the error performance particularly in a very noisy block which requires excessive tree searches, a parallel decoding scheme for a multiple stack algorithm (MSA) is proposed in this paper. In this scheme, apart from the main decoder working on the main stack, a scalable set of multiple decoders working in parallel on their multiple stacks are incorporated. All the decoders are working almost independently with the help of the control processor, where the decoding process and the termination rules are outlined. Two cases, 4-processor MSA and 16-processor MSA, are employed as examples to investigate the performance of the proposed scheme. Comparing with the conventional MSA, extensive computer simulations show that the bit error probabilities (BER) are significantly improved.

Indexing terms: Multiple stack algorithm; Parallel decoding; Convolutional codes

1. Introduction

Convolutional codes are widely used for error correction in both satellite and terrestrial communications. There are two main techniques for decoding convolutional codes namely the Viterbi algorithm (VA) and the sequential decoding [1]. For the sequential decoding, although it is not optimal, arbitrarily low error probabilities can be achieved since its decoding effort does not depend on the code constraint length [2-3]. In addition, increasing the code constraint length means to improve the error correcting capability of the code. However, the most undesirable attribute of the sequential decoding is that noisy blocks may require a large number of

computations and decoding times can occasionally exceed the computational limit (C_{lim}), causing the information to be lost or erased (marked as an erasure). Chevillat and Costello [4] proposed an algorithm for erasurefree sequential decoding called the multi stack algorithm (MSA). The MSA is developed from the single stack algorithm (SSA) [5] by splitting the stack into many stacks, where the first stack is usually large and the second and higher order stacks with typically all the same sizes are relatively small. The basic idea of the MSA is to make use of higher order stacks for blocks that require an excessive number of computations to be decoded. The MSA decoding process will be terminated when either a terminal node is found in the main stack or the computation limit C_{lim} is reached. However, decoding termination with C_{lim} usually gives very high error rate. Later Y.Lin and S.H. Tu [6] proposed a modified multiple stack algorithm that rearranged the memory used to implement stacks as a ring-like structure. The main advantage of their work is the flexibility in defining the size of the top transferred nodes between stacks (T) and the size of higher order stacks (Z) to be as large as desired in order to improve the bit error performance. However, any increase in T or Z requires more computation and thereby a long delay may be occurred. Kallel and Li proposed a bidirectional multiple stack algorithm (BMSA) in 1999 [7]. Their technique was to apply the bi-directional tree search to the MSA. It has been shown that the performance of the conventional MSA is significantly improved with the use of a bidirectional decoding scheme.

Recently, in order to achieve high-throughput rate or to reduce decoding delay, parallel decoding has gained some interest from many researchers in the coding community [8-13]. In this study, a parallel decoding scheme for the MSA is proposed. The main idea is to exploit scalable multiple decoders working in parallel to search the correct path in the code tree. The scheme comprises of a main decoder extending paths on the main stack and many scalable sub-decoders extending paths on the sub-stacks. All the decoders work independently in parallel with the help of the control processor. It is found by extensive computer simulations that the error performance of our proposed scheme is better than that of the conventional MSA. The paper is organized as follows. Background on the MSA decoding is briefly discussed in Section 2. In Section 3, the proposed parallel decoding scheme for the MSA is described, where description of the scheme,

decoding process and the termination rules are given. In Section 4, performance of the proposed scheme is presented by computer simulation results in comparison with the conventional MSA. Finally, Section 5 is the conclusion of this paper.

2. Background on the MSA

In the MSA, when the first stack fills up, the top T nodes with best metrics are transferred to a second stack. Decoding then continues in the second stack using only these T transferred nodes. If a path in the second stack reaches a terminal node before the stack fills up, the path is stored in a special register called the *tentative decision* (TD). The decoder then deletes the remaining nodes in the second stack and returns to the first stack where decoding continues. Since T nodes were removed there are now exactly T free places in the first stack. If the decoder reaches a terminal node before the first stack fills up again, the metric of the new path is compared to that of the TD. The path with the better metric is retained and becomes the final decoding decision. It can be shown that a decision made in this way is as good as that made by the SSA with an arbitrarily large stack. However, if the first stack fills up again before the end of the tree is reached, a new second stack is formed by transferring from the top T nodes of the first stack. Additional stacks are formed in a similar manner until a path to a terminal node is found. The decoder compares this path with the TD and retains the path with the best metric. The rest of the nodes in the current stack are then deleted and decoding proceeds in the previous stack. Occasionally, the decoder may not reach a terminal node in the first stack after the number of node extensions (computations) exceeds the computational limit C_{lim}. In this case, the TD becomes the final decoding decision. To guarantee erasurefree decoding C_{lim} must be carefully chosen. For the case $T = 1$ and a rate of $1/w$ convolutional code (where w is the codeword length in bits) with the constraint length ν the value of C_{lim} must be at least equal to the critical value C_{crit} given by

$$C_{crit} = \sum_{i=1}^{k-1} (Z_i - 1) + 2(\nu - 1) \qquad (1)$$

where Z_i is the size of stack i, and k the number of tree branches without tail (the information frame size). The decoding termination with C_{lim} in the MSA usually happens with very noisy blocks that require excessive tree searches. The decoded bit error rate in this case is very high. Although, the error performance of the MSA can be improved by increasing the value of C_{lim} or increasing the size of the first stack, this choice may not be possible in some applications where decoding delays are not acceptable or the memory storage is limited with a single-processor decoder. In such situations we propose the use of a Parallel decoding process.

3. A Parallel Decoding Scheme for the MSA

Our main goal is to apply a scalable set of multiple decoders concurrently searching the correct path in the code tree. Therefore, the probability of finding the correct path is raised up with our proposed scheme.

3.1 Description of the proposed scheme

The proposed parallel decoding scheme for the MSA is shown in Fig. 1. The arrangement and the processing of the scheme are as follows.

(1) It consists of the following components: the main (first) decoder (DEC#1), scalable multiple sub-decoders (DEC#2, DEC#3,..., DEC#n), the control processor (CP), the main (first) stack (Z_1), $n-1$ primary sub-stacks (Z_2), which equal to the number of sub-decoders and several secondary sub-stacks (Z).

(2) The main decoder extends paths of the main stack only based on the SSA.

(3) Each sub-decoder is identical. One sub-decoder extends paths of its primary sub-stack and secondary sub-stacks based on the conventional MSA.

(4) The CP is used for the following tasks: (a) Stop the decoding when a decoding termination rule is found (b) notify the main decoder whenever there is a sub-decoder available, compare and update the best path with best metric as the TD in a special register whenever a decoder finds a terminal node

(5) The main stack Z_1 is usually large. The primary sub-stacks Z_2 and the secondary sub-stacks Z are very small and all have equal size.

3.2 Decoding of a data block

The decoding begins with placing the root node in the main stack where the DEC#1 extends paths is based on the SSA. When the first stack is full, the CP is informed by the DEC#1. The CP then checks if there is one sub-decoder available. If it finds one, the top T nodes of the main stack are transferred to the corresponding primary sub-stack where the available sub-decoder will extend paths based on the conventional MSA. After that the main decoder and the sub-decoder will simultaneously extend the top paths of their own stacks. This operation is different from the conventional MSA in which the node extensions in the first stack cannot be continued when the first stack is full. With this scheme, it is the task of the sub-decoder to extend those transferred nodes of its own sub-stacks. Then the first decoder can continue the node extension in the first stack straight away without having to wait for the sub-stacks to be deleted. The node extension in the first stack cannot continue only when there is no sub-decoder available. The probability of the first stack to be suspended is decreased with the number of sub-decoders increases. The advantage of this scheme is that whenever the first stack is full, some top nodes are transferred to a sub-stack as long as there is at least one sub-decoder available. All the decoders are working in parallel on their own stacks to search for the most likely path in the code tree.

For a sub-decoder, a new secondary sub-stack will be created whenever the primary sub-stack is full. Then the top T nodes will be transferred to the current secondary sub-stack as in the conventional MSA. The decoding will proceed this way until a decoding termination rule is found.

The following example shows the contents of some stacks during the decoding in progress. Assuming that two sub-decoders ($n = 3$) are employed, Fig. 2(a) shows the stacks contents at the beginning stage at which only the main decoder is busy with the node extensions in the main stack while the two sub-decoders are standstill. When the first stack is full (see Fig. 2(b)), the top T nodes are transferred to the primary sub-stack of DEC#2 (see Fig. 3(a)). Then DEC#1 and DEC#2 will extend the top nodes of their stacks simultaneously while DEC#3 is still doing nothing (see

Fig. 3(a)). Fig. 3(b) shows the situation when the first stack is full for the second time. At this point the top T nodes are transferred from the first stack to the primary sub-stack of DEC#3 (see Fig. 4(a)), after that all the decoders are working in parallel.

Fig. 4(b) shows the situation when the first stack and the primary sub-stack of DEC#2 are full. At this point the first stack is in suspension stage as there is no sub-decoder available, DEC#2 creates a secondary sub-stack and then transfers the top T nodes from the primary sub-stack to the current secondary sub-stack and DEC#3 continues extending paths of its primary sub-stack (see Fig. 5(a)).

Fig. 5(b) shows the situation when a terminal node is found in the primary sub-stack of DEC#3. This terminal node will be kept in a special register as the TD. Then the primary sub-stack of DEC#3 is deleted and DEC#3 returns to standstill (see Fig. 6(a)). After that DEC#1 is informed of the availability of DEC#3 by the CP and then the top T nodes are transferred to the primary sub-stack of DEC#3 for the second time. All the decoders are now working in parallel again (see Fig. 6(b)). The decoding process continues in this way until one of the termination rules is found.

3.3 Termination rules

The decoding process will be terminated when one of the following rules is found:

(1) The computational limit, C_{lim}, is reached (same as the conventional MSA).

(2) If a terminal node is found in the first stack **and** all the sub-decoders return to be standstill.

This decoding process is different from the conventional MSA in which the decoding is terminated when a terminal is found in the first stack. The reason of not stopping the decoding straight away when a terminal node is found in the first stack is that the correct path may be hidden in one of the sub-stacks where a sub-decoder is working on.

3.4 Discussion

Note that in our process the CP allows all the decoders to work at their full speeds without having to exchange information with each other and the usual communications bottleneck is thereby kept

at minimum. The advantage of having several sub-decoders working on the sub-stacks can be summarized as follows.

(1) The probability of finding the correct path is raised as the availability of sub-decoders helps the main decoder extend the nodes of sub-stacks allowing the main decoder to continue the node extensions in the first stack straight away without having to wait for the deletion of the primary sub-stacks as happened in the conventional MSA

(2) More terminal nodes can be found as several paths are extended simultaneously. Therefore better TD is made with the same value of C_{lim} compared to the conventional MSA.

4. Performance comparison with the MSA

By computer simulation, the performance comparison of our proposed scheme with the MSA is investigated by comparing bit error rate (BER), where 160 000 frames of length $L = 250$ were transmitted through an additive white Gaussian noise (AWGN) channel. The simulator was written in the C language and was run on a LINUX operating system. The cases of 4 and 16 were mostly investigated in this paper. The 4-processor scheme comprises of one main decoder and the other three sub-decoders. Similarly, the case of 16-processor scheme comprises of one main decoder and fifteen sub-decoders. For convenience, we name 4-processor and 16-processor cases as 4MSA and 16MSA, respectively. The code used is a nonsystematic rate $R = 1/2$ code with the code constraint length $\nu = 15$ and generators used are in octal 65231 and 43677 (optimum d_{free} code [2]). In order to make a fair comparison between the conventional MSA and our proposed scheme, the amounts of memory storage capacity of both systems are kept the same.

Fig 7 shows the BER curves as a function of the computational limit C_{lim} at the signal to noise ratio of $E_b / N_0 = 5.5\,\text{dB}$ with $Z_1 = 800, Z = 11, T = 3$ for all the three algorithms. As it can be seen from the Figure, the BER performance of 16MSA is lowest among the three. For example, at $C_{\text{lim}} = 20000$ the MSA and 4MSA reach BER values of 5.9×10^{-5} and 5.6×10^{-5} respectively, while the 16MSA reaches a BER of value 4.1×10^{-5}. Also as expected, the value of C_{erf} (the value which guarantees erasurefree decoding) is lowest for the 16MSA. This is due to the multiple path extensions by the main decoder and sub-decoders. From the Figure, the value of C_{erf} for the

16MSA is approximately 2200, while the values of C_{crit} for the cases of 4MSA and MSA are approximately 2800 and 3000, respectively. It should be noted that the value of C_{crit} is approximately determined from the value of C_{lim} at which the BER reaches the peak. Keeping the parameters the same as those used for Fig 7 except changing Z_1 to 1100, the similar results are shown in Fig 8. The humps of the curves for the 16MSA in Fig 7 and Fig 8 can be explained as follows. When increasing C_{lim} up to C_{crit} the decoding process begins to perform erasure-free decoding. After the value of C_{lim} is further increased the decoding process is done in parallel by the main decoder and the sub-decoders. The probability of finding the correct path is then increased, resulting in a further improvement in the BER performance when compared to the BER at the value of C_{crit}.

Fig. 9 shows the BER curves with $Z=11, T=3$, the computational limit $C_{lim}=10000$ as a function of Z_1 at $E_b/N_0=5$ dB. From the Figure we can see that the 4MSA has lower BER than the MSA. The improvement in the decoded BER is more pronounced when increasing the number of sub-decoders from 3 (4MSA) to 15 (16MSA). When the first stack size increases, the decoded BER of the three algorithms tend to be closer. This is because more frames finished the decoding in the first stack before it fills up for the first time when Z_1 is increased. Fig. 10 shows BER curves as a function of Z_1 at $E_b/N_0=5.5$ dB with the same parameters used for Fig. 9. The results are similar for both Figures.

Next we investigated the influence of size of Z on the decoded BER at $E_b/N_0=5.5$ dB. Fig. 11 shows the BER curves with $Z_1=700, T=3, C_{lim}=10000$ for various values of Z. It is seen from the Figure that no significant influence of Z on the BER for all the MSA, the 4MSA and the 16MSA. Similar results are shown in Fig. 12 for the BER curves as a function of Z when increasing Z_1 from 700 to 800.

Fig 13 displays the BER curves as a function of T at $E_b/N_0=5.5$ dB with $Z_1=800, Z=11, C_{lim}=10000$. As shown, an increase in T has no noticeable improvement in the BER for the MSA and 4MSA while the BER performance of the 16MSA is somewhat improved when increasing T. This is due to larger value of T and more sub-decoders allow the main stack

to continue the node extension more frequently after it fills up. For example, the BER reduces from 4.3×10^{-5} to 3.7×10^{-5} when changing T from 2 to 5. Similar results are shown in Fig 14 for the BER curves as a function of T when changing Z from 11 to 30. As expected, the BER performance of the 16MSA is the best among the three.

Next we considered the influences of the number of the top nodes transferred from Z_1 to Z_2 and the size of Z_2. Let T' be the number of the nodes transferred from Z_1 to Z_2 when Z_1 fills up. Fig 15 displays the BER performance of the 16MSA as a function of T' for various values of Z_2. The parameters used for the Figure are $Z_1 = 800, Z = 20$, $T = 3$, and $C_{lim} = 10000$ at $E_b / N_0 = 5.5$ dB. We can see that any increase in Z_2 obtains the improvement in the BER performance. This is because the larger value of Z_2 allows more node extensions in the primary sub-stack thereby raising the probability of finding a terminal node in the primary sub-stack. When a terminal is found in a primary sub-stack, this sub-stack will be deleted and then the top nodes from the main stack can be transferred. This allows the node extension in the main stack to be continued therefore the probability of finding the correct path in the main stack is raised. The Figure also shows that any increase in T' tends to improve the BER for a given value of Z_2

Figure 16 shows the comparison of the BER performance for various values of signal to noise ratio (E_b / N_0) for the MSA, the 4MSA, the 8MSA, the 16MSA, and the 32MSA. The parameters used are $Z_1 = 1000, Z = 20, T = 8$ and $C_{lim} = 20000$ for all the curves and $Z_2 = 300, T' = 50$ for the 4MSA, the 8MSA, the 16MSA, and the 32MSA. As expected, the BER performance is improved with our proposed parallel decoding scheme. For example, at signal to noise ratio of 5 dB, the BER of the MSA is 2.5×10^{-4}, the BER of the 4-MSA is 1.8×10^{-4}, the BER of the 8-MSA is 1.5×10^{-4}, the BER of the 16-MSA is 8×10^{-5} and the BER of the 32-MSA is 6×10^{-5}. It is clearly shown that when increasing the number of processors the BER is significantly decreased.

From the simulation results presented above, a guideline for parameter selection in parallel decoding scheme can be given as follows.

1. The value of C_{lim} should be large enough to guarantee erasure-free decoding.

2. The size of Z_1 should be made as large as possible. This depends on the available memory storage.
3. The size of Z can be independently selected. Therefore small size of Z is recommended.
4. The sizes of T and T' should be as large as possible.
5. The size of Z_2 should also be chosen as large as possible according to the results on Figure 15.

5. Conclusions

A parallel decoding scheme for the MSA is proposed and investigated in this paper. The idea of parallelism in the MSA mainly aims at the improvement of the bit error probability of the MSA for some blocks, which requires excessive searches. In the conventional MSA when the number of tree searches is limited with the value of C_{lim} but a terminal node is not yet found in the main stack, the decoder will usually give very poor error performance. In this paper we investigated the application of parallel decoding for the MSA in which multiple paths in the code tree are simultaneously extended by a set of scalable decoders. Simulation results show that the average bit error rate of the MSA is significantly improved with our proposed parallel decoding scheme. In addition, the BER performance is further improved when the number of processors is increased. Parallel decoding for MSA appears to be an interesting scheme for decoding convolution codes especially in applications where low probabilities are required.

6. Acknowledgement

The authors would like to thank Professor P.G. Farrell of the University of Lancaster for his very useful comments. Many thanks also to the reviewers for their patience and their many helpful suggestions. Lastly, the authors would like to acknowledge the Thailand Research Funds Agency for its support under the grant MRG4680124.

7. References

1. FORNEY, G.D.: 'Convolutional codes III: sequential decoding', *Inf. Control*, 1974, **25**, pp. 267-297

2. LIN, S., and COSTELLO, D. J. Jr.: 'Error control coding: fundamentals and applications' (Prentice-Hall, New Jersy, 1983)

3. HACCOUN, D., and BEGIN, G.: 'High-rate punctured convolutional codes for Viterbi and sequential decoding', *IEEE Trans. Commun.*, 1989, **37**, (11), pp. 1113-1125

4. CHEVILLAT, P.R., and COSTELLO, D.J. Jr.: 'A multiple stack algorithm for erasurefree decoding of convolutional codes', *IEEE Trans. on Commun.*, 1976, **25**, (12), pp. 1460-1470

5. JELINEK, F.: 'A fast sequential decoding using a stack', *IBM J. Res. Dev.*, 1969, **13**, pp. 675-685

6. LIN, Y., and TU, S.H.: 'Modified multiple stack algorithm for decoding convolutional codes', *IEE Pro. Commun.*, 1997, **44**, (4), pp. 221-228

7. KALLEL, S., and LI, K.: 'A bidirectional multiple stack algorithm', *IEEE Trans. on Commun.*, 1999, **47**, (1), pp. 6-9

8. FAIRHURST, G., PANG, S.L., and WAN, P.S.: 'Smart codec: an adaptive packet data link', *IEE Pro. Commun.*, 1998, **145**, (3), pp. 180-185

9. IMTAWIL, V.: 'Bidirectional sequential decoding: a multiprocessor approach', Ph.D. thesis, the University of Manchester, U.K., 1999

10. KWAK, J., and LEE, K.: 'Design of dividable interleaver for parallel decoding in turbo codes' Electronics Letters, 2002, **38**, (22), pp. 1362 – 1364

11. ZHANG, X., ZHAO, M., ZHOU, S., and WANG, J.: 'Parallel decoding of turbo product codes for high data rate communication', IEEE Vehicular Technology Conference, VTC 2003, pp. 2372 – 2375

12. UMANESAN, G., and FUJIWARA, E.: 'Parallel decoding cyclic burst error correcting codes' Proceedings of IEEE International Symposium on Information Theory, ISIT 2003, pp. 420 – 420

13. NIKARA, J., VASSILIADIS, S., TAKALA, J. and LIUHA, P.: 'Multiple-symbol parallel decoding for variable length codes' *IEEE Trans. on VLSI*, 2004, **12**, (7), pp. 676 – 685

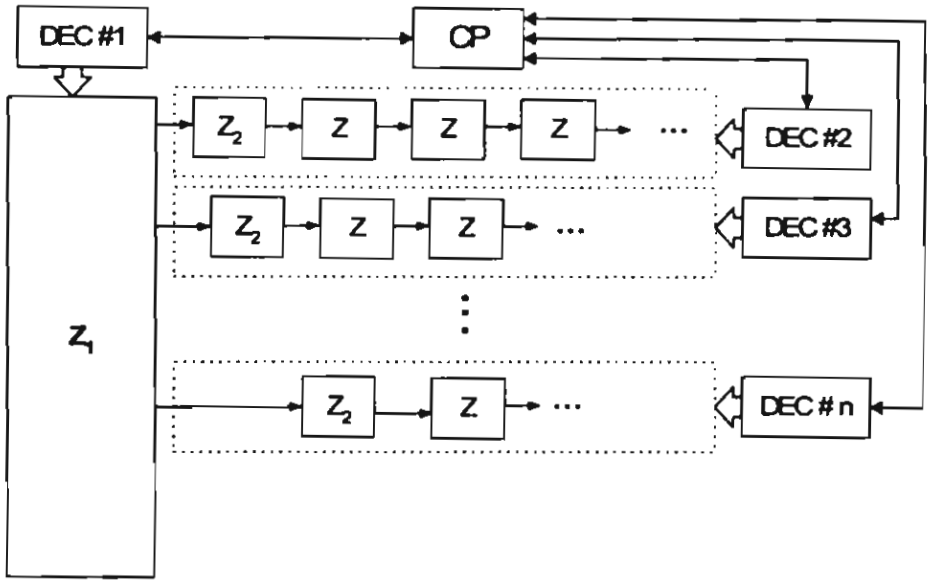


Fig. 1 *A parallel decoding scheme for the MSA*

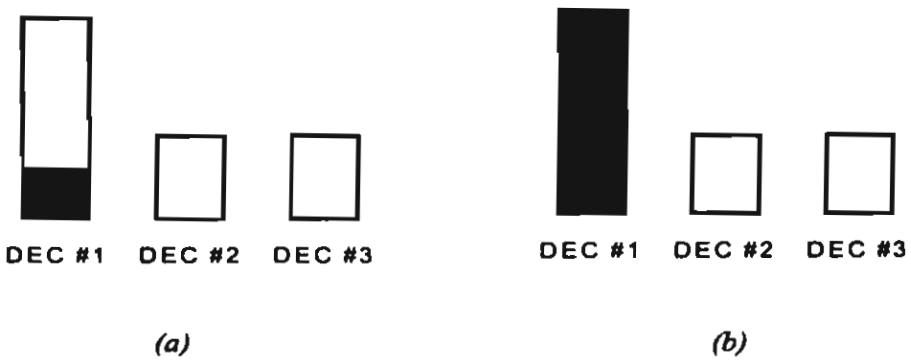


Fig. 2 *Stacks contents at the beginning stage until the first stack is full*

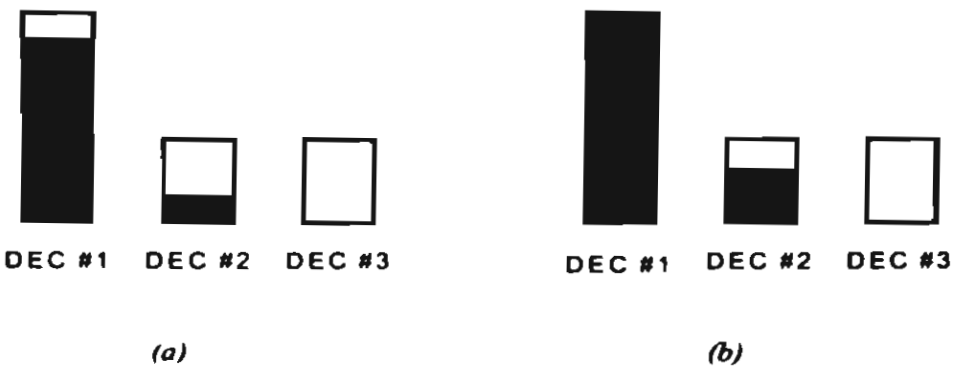


Fig. 3 *Top T nodes in the first stack are transferred to the primary sub-stack of DEC#2*

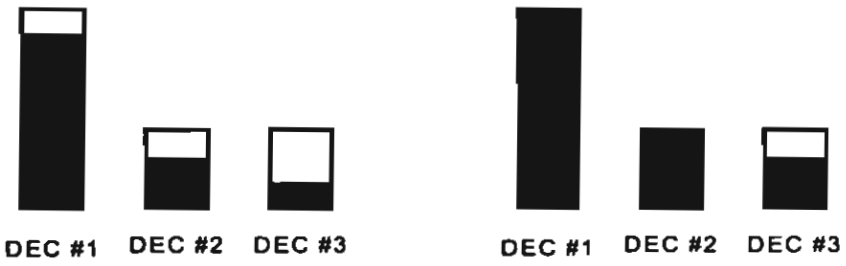


Fig. 4 *(a) All decoders are working in parallel*

(b) First stack and primary sub-stack of DEC#2 are full

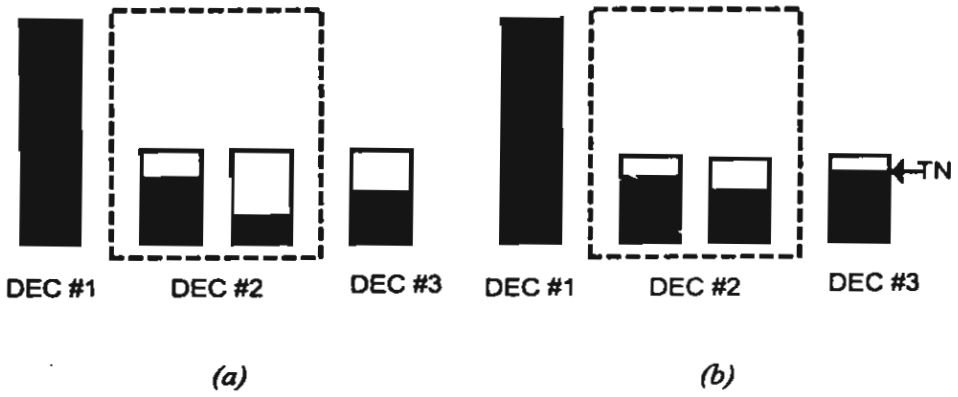


Fig. 5 *(a) First stack is suspended (b) A terminal node is found by DEC#3*

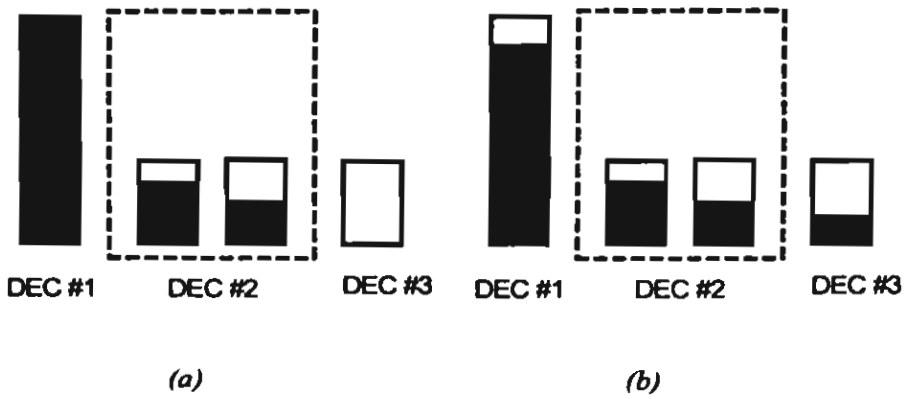


Fig. 6 *(a) The first sub-stack of DEC#3 is deleted.*

(b) All the decoders are working in parallel again

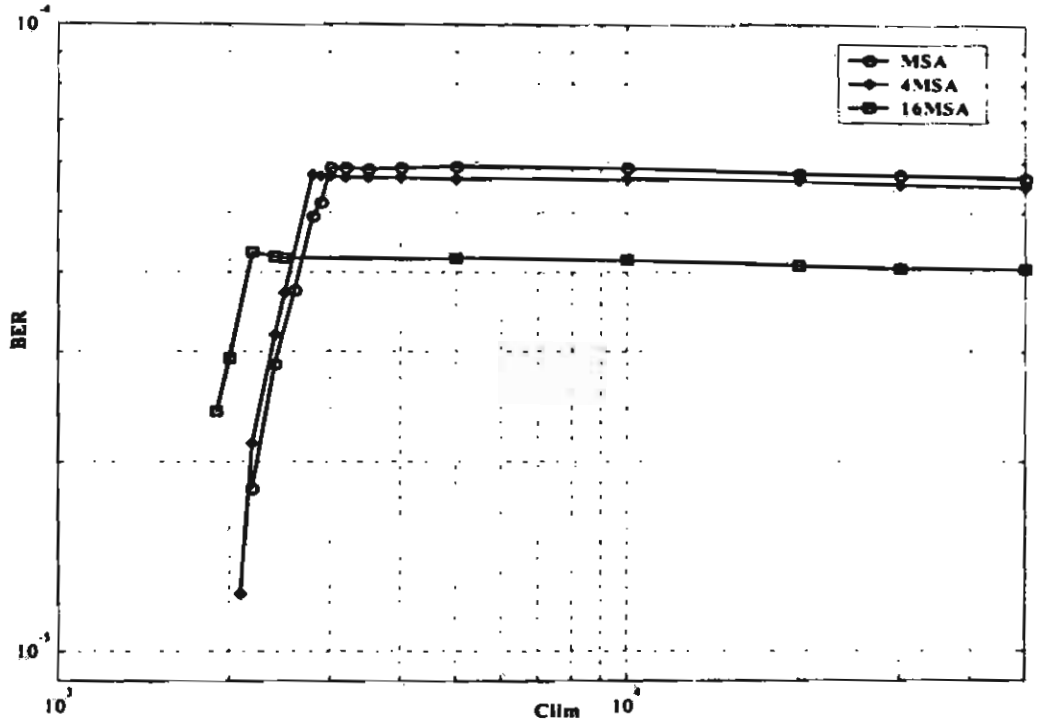


Fig. 7 *BER as a function of* C_{lim} *for* $Z_1 = 800$

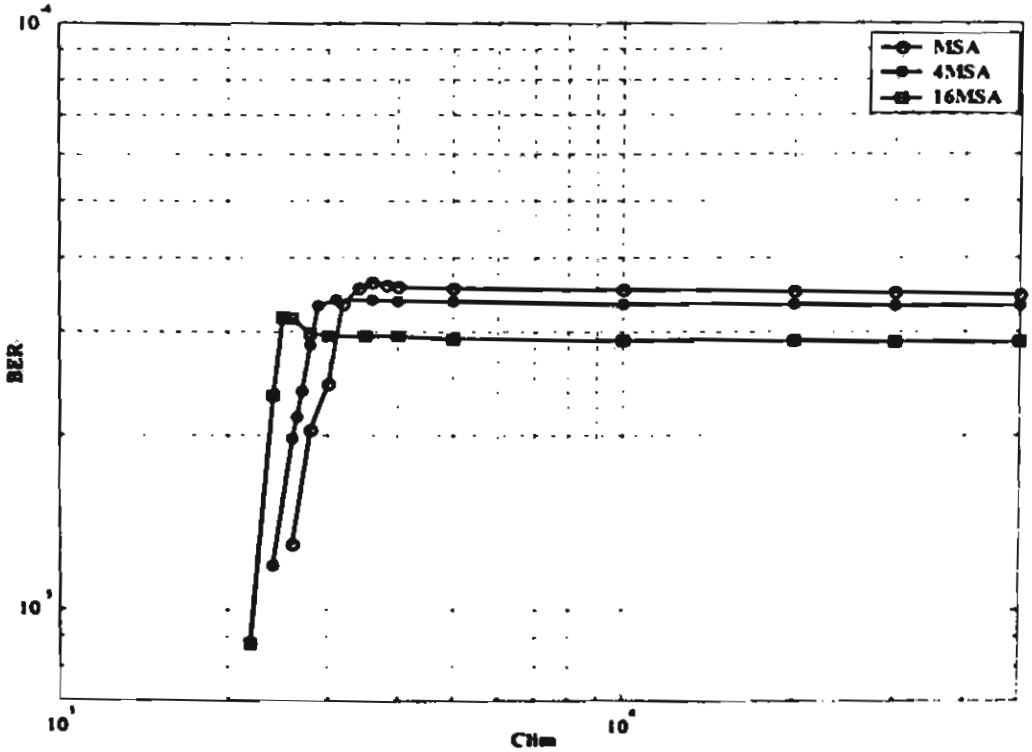


Fig. 8 *BER as a function of* C_{lim} *for* $Z_1 = 1100$

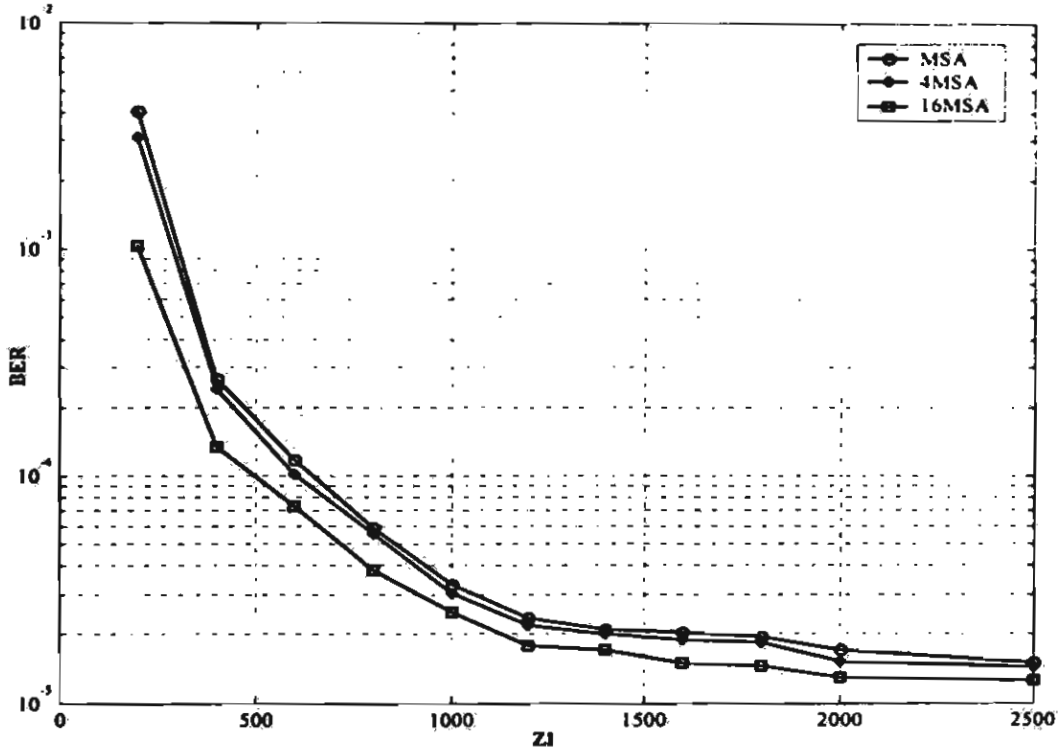


Fig. 9 *BER as a function of* Z_1 *at* $E_b / N_0 = 5$ *dB*

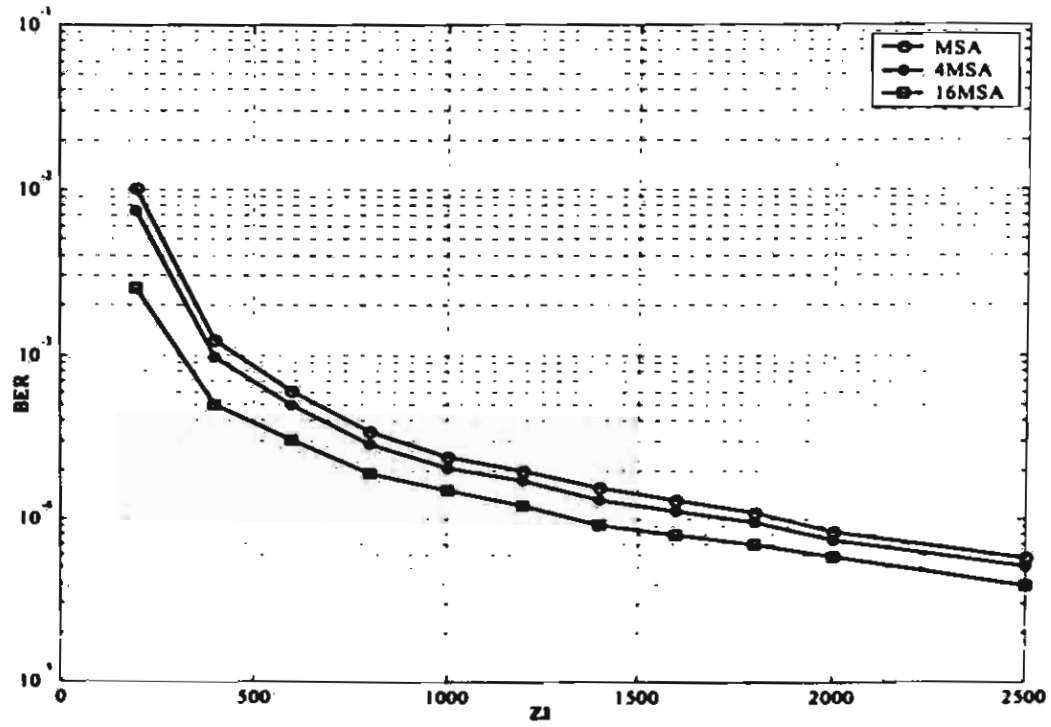


Fig. 10 *BER as a function of* Z_1 *at* $E_b / N_0 = 5.5$ *dB*

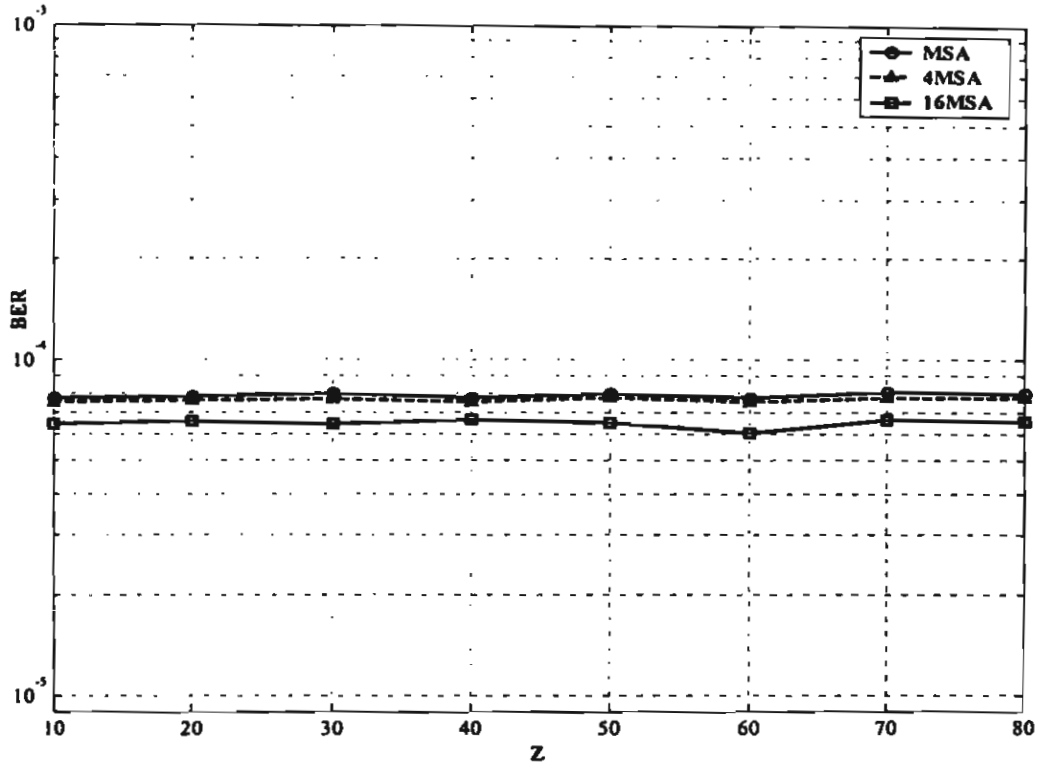


Fig. 11 *BER as a function of Z with* $Z_1 = 700$

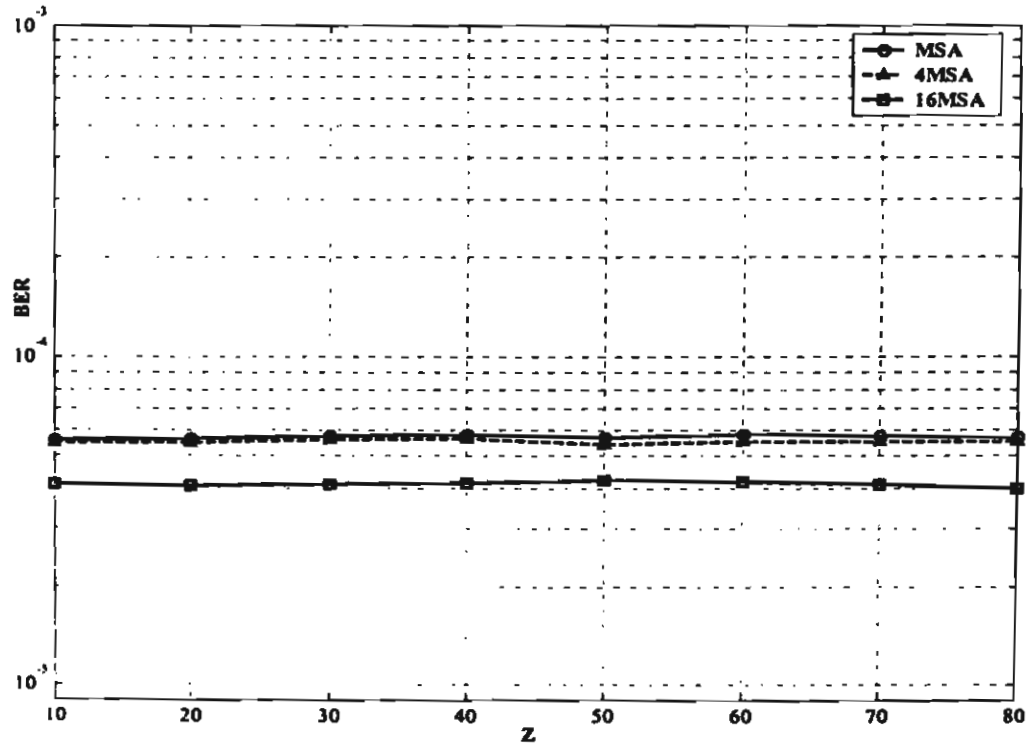


Fig. 12 *BER as a function of Z with* $Z_1 = 800$

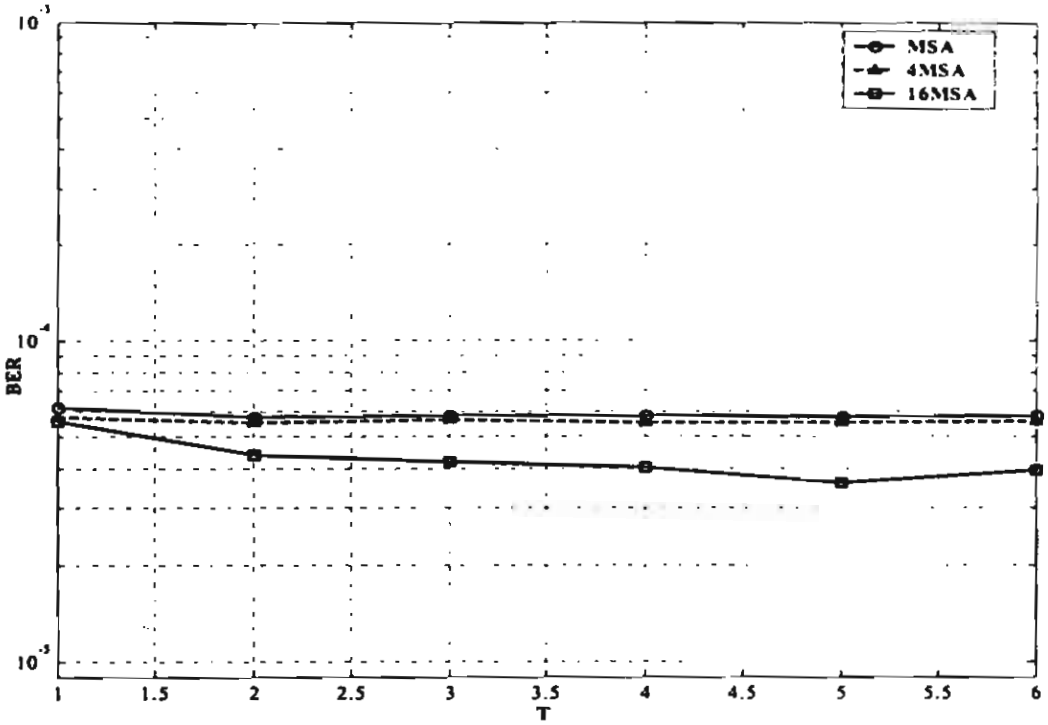


Fig. 13 *BER as a function of* T *for* $Z = 11, C_{im} = 10000$

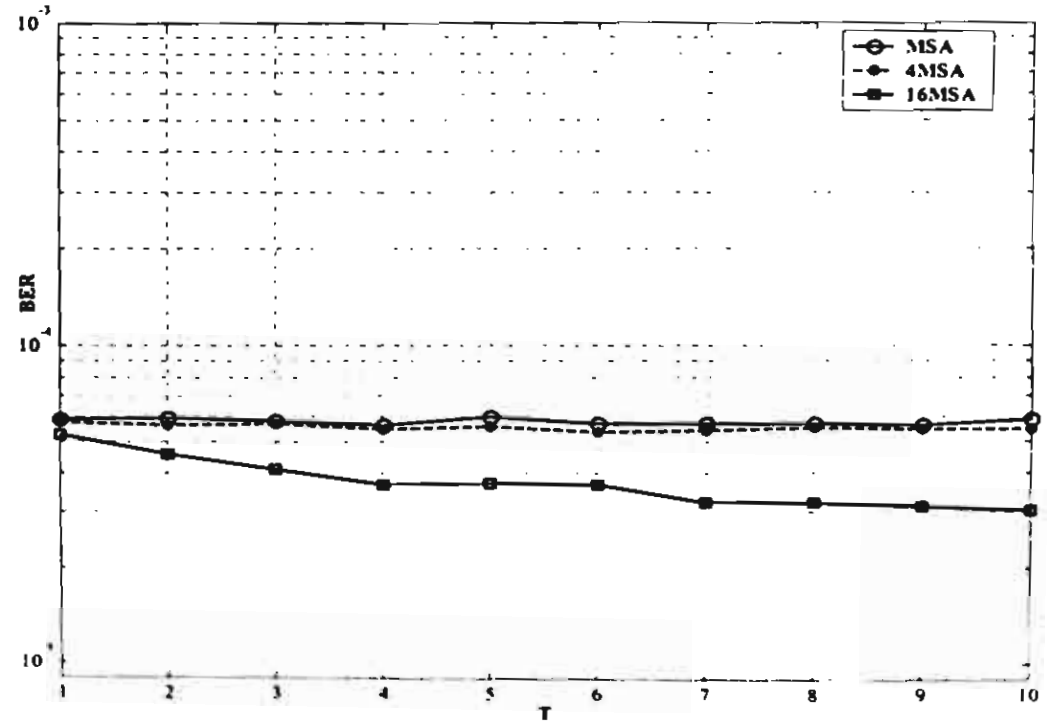


Fig. 14 *BER as a function of* T *for* $Z = 30, C_{im} = 10000$

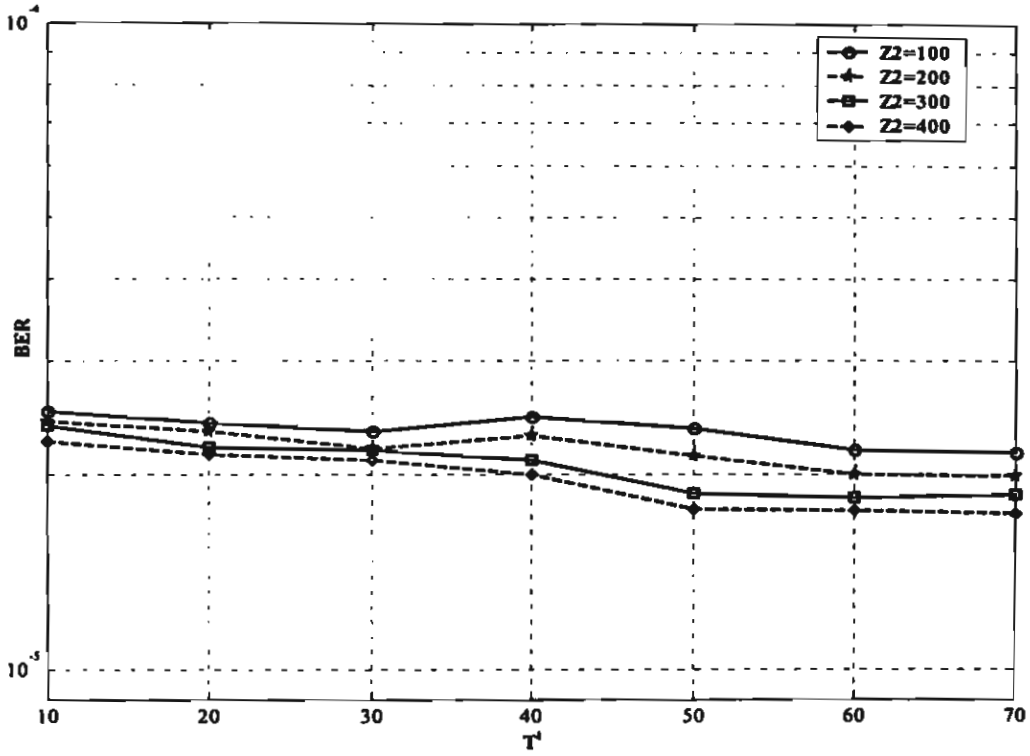


Fig. 15 *BER of the 16MSA as a function of T' for various values of* Z_2

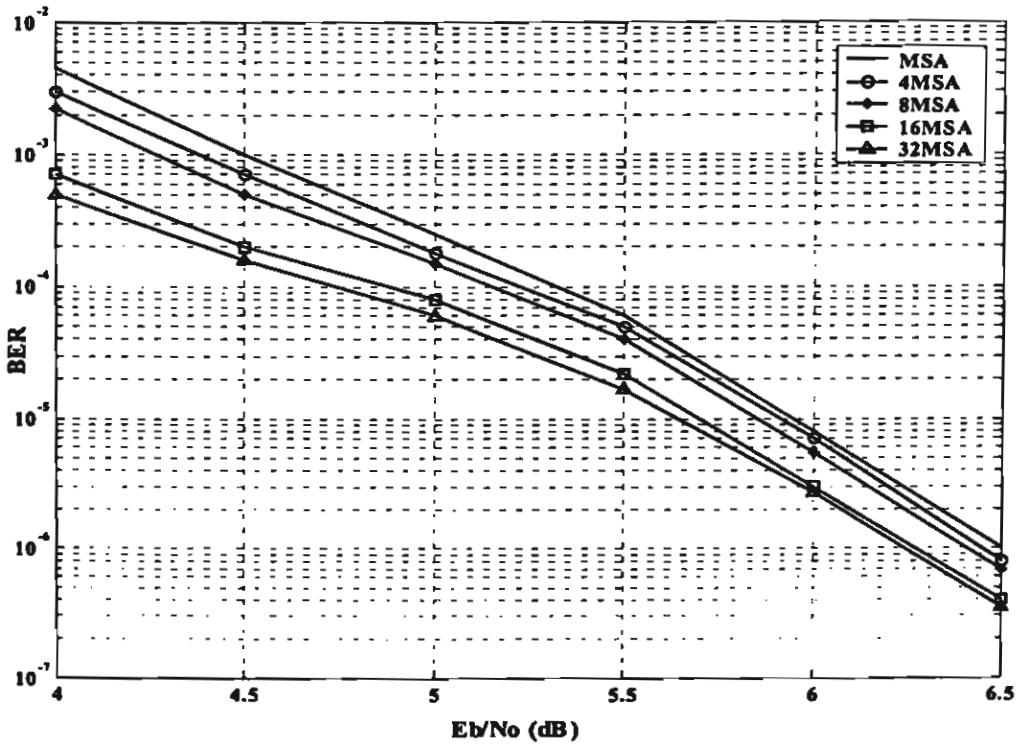


Fig. 16 Comparison on BER with the MSA

A Scalable multiprocessor-based multiple stack algorithm

Virasit Imtawil*, Ph.D. and Wanlop Surakampontorn, Ph.D.

Department of Electrical Engineering, Faculty of Engineering,

Khon Kaen University, Khonkaen, THAILAND 40002

Abstract— A parallel decoding scheme for the multiple stack algorithm (MSA) is proposed in this paper. The proposed scheme is composed of a main decoder working on the main stack and a scalable set of multiple decoders working on their multiple stacks. All the decoders are working almost independently with the help of the control processor. Two cases, 4-processor MSA and 16-processor MSA, are employed to investigate the performance of the proposed scheme. Extensive computer simulations show that the bit error probability of the MSA is improved. We also demonstrate that parallel decoding for the MSA appears to be an interesting scheme for decoding convolutional codes especially in applications where low error probabilities are required.

Keywords— Multiple stack algorithm; Parallel decoding; Convolutional codes

Output

1. V. Imtawil and W. Surakampontorn, submitted to *IEE Proceedings – Communications*

* Corresponding author Tel. 043 202353, 06 715 8082, email: virasit@kku.ac.th