ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)

ปริญญา

วิศวกรรมไฟฟ้า	วิศวกรรมไฟฟ้า
สาขา	**ภาควิชา**

เรื่อง ชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีชุดข้อมูลขาเข้า 2 ตัวเลือก บนบอร์ดทดลอง FPGA

Prototype of Vector Symbol Decoder for a (3, 2, 2) Convolutional Code with Two Alternative Choices on an FPGA Board

นามผู้วิจัย นายอรรถภัทร์ สืบเนื่อง

ได้พิจารณาเห็นชอบโดย

ประธานกรรมการ .....

(ผู้ช่วยศาสตราจารย์อุศนา ตัณฑุลเวศม์, Ph.D.)

กรรมการ .....

(ผู้ช่วยศาสตราจารย์วชิระ จงบุรี, Ph.D.)

กรรมการ .....

(รองศาสตราจารย์มงคล รักษาพัชรวงศ์, Ph.D.)

หัวหน้าภาควิชา .....

(รองศาสตราจารย์มงคล รักษาพัชรวงศ์, Ph.D.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

.....

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ ..... **เดือน** ..... **พ.ศ.** .....

วิทยานิพนธ์

เรื่อง

ชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีชุดข้อมูลขาเข้า 2 ตัวเลือก บนบอร์ดทดลอง FPGA

Prototype of Vector Symbol Decoder for a (3, 2, 2) Convolutional Code with Two Alternative Choices on an FPGA Board

โดย

นายอรรถภัทร์ สืบเนื่อง

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)
พ.ศ. 2552

อรรถภัทร์ สืบเนื่อง 2552: ชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีชุดข้อมูลขาเข้า 2 ตัวเลือก บนบอร์ดทดลอง FPGA ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า) สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า ประธานกรรมการที่ปรึกษา: ผู้ช่วยศาสตราจารย์อุศนา ตัณฑุลเวศม์, Ph.D. 140 หน้า

วิธีถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน ที่ใช้สัญลักษณ์นอนไบนารีขนาดใหญ่สามารถแก้ไขความผิดพลาดแบบแถบได้ดี เนื่องจากการใช้สัญลักษณ์นอนไบนารีขนาดใหญ่ มีข้อดีคือ เมื่อทำการแก้ไขแต่ละสัญลักษณ์จะเท่ากับเป็นการแก้ไขความผิดพลาดทั้งหมดที่เกิดขึ้นในสัญลักษณ์นั้นๆ ทำให้การถอดรหัสแบบนี้เหมาะกับช่องสัญญาณที่อาจมีคุณภาพต่ำในบางช่วงเวลา เช่น ช่องสัญญาณไร้สาย โดยไม่ต้องทำการอินเทอลีฟเหมือนกรณีรหัสที่ใช้สัญลักษณ์ไบนารี

งานวิจัยนี้ มีวัตถุประสงค์เพื่อออกแบบและสร้างชิ้นงานถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) รหัสหนึ่ง ที่ใช้สัญลักษณ์ขนาด 32 บิต และมีชุดข้อมูลขาเข้า 2 ตัวเลือก บนบอร์ด FPGA เพื่อเป็นการแสดงว่า ทฤษฎีของการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิงสำหรับรหัสแบบคอนโวลูชันสามารถนำมาประยุกต์เป็นฮาร์ดแวร์ได้จริง โดยได้ออกแบบการทำงานของตัวถอดรหัสเป็น 3 กลุ่มตามลำดับขั้นในการพัฒนาชิ้นงาน ได้แก่ การถอดรหัสด้วยซินโดรมเดียว การถอดรหัสด้วยวิธีแก้ไขได้ด้วยตัวเลือกอันดับสอง และการถอดรหัสด้วยวิธีการรวมทางพีชคณิตได้ศูนย์ ผลการทดสอบการถอดรหัสด้วยชิ้นงานต้นแบบพบว่า การทำงานของชิ้นงานต้นแบบเป็นไปตามที่ได้ออกแบบการทำงานไว้ คือสามารถแก้ไขความผิดพลาดได้ตามทฤษฎีของการถอดรหัสนี้ทุกประการ

_____ _____ ___ / ___ / ___

ลายมือชื่อนิสิต ลายมือชื่อประธานกรรมการ

Auttaphud Seubnaung 2009: Prototype of Vector Symbol Decoder for a (3, 2, 2) Convolutional Code with Two Alternative Choices on an FPGA Board. Master of Engineering (Electrical Engineering), Major Field: Electrical Engineering, Department of Electrical Engineering. Thesis Advisor: Assistant Professor Usana Tuntoolavest, Ph.D. 140 pages.

Vector symbol decoding (VSD) for convolutional codes that use large nonbinary symbols can correct burst errors well. The benefit of using large nonbinary symbols is that correcting each error symbol means correcting all error bits in that symbol. This makes the decoding technique suitable for channels with low quality at times such as wireless channels without the need to use interleaving as in the case of binary codes.

The purpose of this research was to design and implement the prototype of channel decoder called Vector symbol decoding that can decode a (3, 2, 2) convolutional code that uses 32-bit symbols with 2 alternative choices. The implementation was done on an FPGA board to show that the theory of Vector symbol decoding for convolutional codes can be applied to hardware. The development of this decoder was done in 3 steps: Decode with one syndrome, the Correct with second choice and the Correct with null combination. The result shows that the prototype can work exactly as designed. It can correct all error patterns indicated in the theory of VSD.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

ผู้วิจัยขอกราบขอบพระคุณ ผศ.ดร.อุศนา ตัณฑุลเวศม์ ประธานกรรมการที่ปรึกษาวิทยานิพนธ์ ผศ.ดร.วชิระ จงบุรี กรรมที่ปรึกษาสาขาวิชาเอก และ รศ.ดร.มงคล รักษาพัชรวงศ์ กรรมการที่ปรึกษาสาขาวิชารอง ที่ให้คำปรึกษาในการเรียน การค้นคว้าวิจัย ตลอดจนการตรวจแก้ไขวิทยานิพนธ์จนกระทั่งเสร็จสมบูรณ์ และกราบขอบพระคุณ ผศ.ดร.เขมะฑัต วิภาตะวนิช ผู้แทนบัณฑิตวิทยาลัย ที่ได้ให้ความกรุณาตรวจแก้ไขวิทยานิพนธ์ให้สมบูรณ์ยิ่งขึ้น

ขอกราบขอบพระคุณอาจารย์ภาควิชาวิศวกรรมไฟฟ้าทุกท่าน ที่ได้อบรมสั่งสอนและมอบความรู้อันเป็นประโยชน์อย่างยิ่งในการนำไปใช้ประโยชน์ต่อไป และขอขอบคุณ คุณรัชนนท์ อินทรสกุล รุ่นพี่ผู้ซึ่งเป็นครูและเพื่อนร่วมงาน ที่ได้ให้ความช่วยเหลือและให้คำแนะนำต่างๆ รวมทั้ง บริษัท AddCom Technologies จำกัด ที่ได้มอบหัวเชื่อมต่อ QSE มาให้ใช้ในงานวิจัย

ด้วยความดีหรือประโยชน์อันใดเนื่องจากวิทยานิพนธ์เล่มนี้ ขอมอบแด่คุณพ่อ คุณแม่ ที่ได้อบรมและให้กำลังใจผู้วิจัยมาตลอดในทุกเรื่อง และสุดท้ายขอขอบคุณ คุณฑิมพิกา เกษมสุขไพศาล ที่ได้ให้กำลังใจ

อรรถภัทร์ สืบเนื่อง

กุมภาพันธ์ 2552

สารบัญ

หน้า

สารบัญตาราง

ตารางผนวกที่

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

คำอธิบายสัญลักษณ์และคำย่อ

ARQ	=	Automatic repeat request
ASIC	=	Application-specific integrated circuit
CLBs	=	Configurable logic blocks
ELV	=	Error locating vector
FPGA	=	Field-programmable gate array
FSM	=	Finite state machine
HDL	=	Hardware description language
LUT	=	Lookup table
NC	=	Null combination
NI	=	Null indicator
PCB	=	Printed circuit board
USB	=	Universal serial bus
VHDL	=	Very-high-speed integrated circuits HDL
VSD	=	Vector symbol decoding

ชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีชุดข้อมูลขาเข้า 2 ตัวเลือก บนบอร์ดทดลอง FPGA

Prototype of Vector Symbol Decoder for a (3, 2, 2) Convolutional Code with Two Alternative Choices on an FPGA Board

คำนำ

การสื่อสารแบบดิจิตอลนั้น วิธีทำให้การสื่อสารที่น่าเชื่อถือมีความสำคัญเป็นอย่างยิ่ง เนื่องจากการส่งข้อมูลจากต้นทางไปยังปลายทางต้องผ่านตัวกลาง อาจทำให้ข้อมูลที่ได้รับมีความผิดพลาดได้ การเข้ารหัสช่องสัญญาณจึงเป็นวิธีการเพิ่มความน่าเชื่อถือของข้อมูลวิธีหนึ่ง โดยจะทำการเข้ารหัสช่องสัญญาณที่ต้นทาง แล้วถอดรหัสช่องสัญญาณที่ปลายทาง ซึ่งสามารถตรวจสอบความผิดพลาดของข้อมูล และสามารถแก้ไขข้อมูลที่เกิดความผิดพลาดให้เป็นข้อมูลที่ถูกต้องได้ ในปัจจุบันนี้การสื่อสารมีปริมาณของการใช้งานมากขึ้น จึงควรพัฒนาวิธีการเข้ารหัสและถอดรหัสช่องสัญญาณให้เหมาะสมกับปริมาณข้อมูลที่อาจเพิ่มขึ้นในอนาคต โดยวิธีการถอดรหัสแบบเวกเตอร์ซิมโบลดีโคดดิง (Vector symbol decoding) เป็นวิธีการถอดรหัสที่มีประสิทธิภาพวิธีหนึ่ง

วิธีการถอดรหัสแบบเวกเตอร์ซิมโบลดีโคดดิง มีความยืดหยุ่นสามารถนำมาประยุกต์ใช้งานได้หลายรูปแบบ และในปัจจุบันวิธีการถอดรหัสแบบเวกเตอร์ซิมโบลดีโคดดิงสำหรับรหัสแบบคอนโวลูชัน ถูกสร้างให้สามารถถอดรหัสได้บางส่วน (รัชนนท์, 2551) แต่เป็นการทดสอบเครื่องถอดรหัส และยังไม่สามารถถอดรหัสส่วนใหญ่ได้ จึงเป็นการดีที่จะค้นคว้าและพัฒนาให้เป็นเครื่องถอดรหัสที่สามารถถอดรหัสได้ถูกต้องตามทฤษฎี โดยได้เลือกสร้างต้นแบบบนบอร์ดทดลอง FPGA เนื่องจากมีจำนวนโลจิกเกทมากพอ และรองรับการออกแบบวงจรรวมขนาดใหญ่มากได้

โครงการวิทยานิพนธ์นี้ นอกจากจะมุ่งหมายที่จะสร้างและพัฒนาเครื่องถอดรหัสเวกเตอร์ซิมโบลดีโคดดิง แล้ว ยังให้ความสนใจเกี่ยวกับการศึกษาวิธีการออกแบบวงจรดิจิตอล การศึกษามาตรฐานของขั้นตอนในการออกแบบวงจรรวมตลอดจนมาตรฐานที่เกี่ยวข้อง การศึกษาวิธีใช้งานและข้อจำกัดของฮาร์ดแวร์ที่เกี่ยวข้อง การศึกษาข้อจำกัดและประสิทธิภาพของวิธีเวกเตอร์ซิมโบลดีโคดดิง เมื่อเป็นฮาร์ดแวร์ และการศึกษาภาษาบรรยายฮาร์ดแวร์อีกด้วย

วัตถุประสงค์

1. สร้างชิ้นงานต้นแบบเบื้องต้นของการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง บนบอร์ดทดลอง FPGA

2. เครื่องถอดรหัสวิธีเวกเตอร์ซิมโบลดีโคดดิง สามารถถอดรหัสแบบคอนโวลูชัน ที่มีพารามิเตอร์ (3, 2, 2) รหัสหนึ่งได้

3. เครื่องถอดรหัสวิธีเวกเตอร์ซิมโบลดีโคดดิง มีการนำชุดข้อมูลตัวเลือกอันดับสอง มาช่วยในการถอดรหัส

4. เครื่องถอดรหัสวิธีเวกเตอร์ซิมโบลดีโคดดิง สามารถถอดรหัสอย่างง่ายที่กำหนดให้ได้

การตรวจเอกสาร

1. บทนำ

ในระบบสื่อสารแบบดิจิตอล ช่องสัญญาณเป็นตัวกลางที่เชื่อมโยงระหว่างภาคส่งและภาครับ ทำหน้าที่ส่งผ่านสัญญาณข้อมูล ช่องสัญญาณนั้นสามารถจำแนกออกได้ในหลายรูปแบบ เช่น เป็นแบบเชิงเส้นหรือไม่เชิงเส้น เป็นแบบใช้สายหรือไร้สาย หรืออาจจะแยกตามชนิดของตัวกลาง เช่น ใยแก้วนำแสง สายตีเกลียวคู่ หรืออากาศ เป็นต้น สิ่งหนึ่งที่เกิดขึ้นเสมอในช่องสัญญาณคือ สัญญาณรบกวน (Noise) สิ่งนี้ทำให้สัญญาณมีรูปร่างผิดเพี้ยน หรือสัญญาณเกิดการแทรกสอด ทำให้ข้อมูลที่ภาครับได้มา เป็นข้อมูลที่มีความผิดพลาด (Error) ปะปนอยู่

สัญญาณรบกวนเป็นสิ่งที่หลีกเลี่ยงไม่ได้ ดังนั้นจึงต้องหาวิธีเพิ่มความน่าเชื่อถือของข้อมูลโดยใช้การเข้ารหัสช่องสัญญาณ (Channel Coding) ซึ่งจะอาศัยวิธีการเพิ่มข้อมูลส่วนเติมเต็ม (Redundancy) ลงไปในข้อมูลต้นฉบับ (Source) โดยจะเรียกการรวมกันระหว่างข้อมูลต้นฉบับกับข้อมูลส่วนเติมเต็มนี้ว่า คำรหัส (Codeword) ในขั้นตอนการสร้างคำรหัสนี้ จะอยู่ในส่วนของการเข้ารหัสช่องสัญญาณ (Channel encoding) และเมื่อนำคำรหัสส่งผ่านช่องสัญญาณไปยังภาครับ ซึ่งภาครับจะสามารถนำคำรหัสไปทำการตรวจสอบความผิดพลาดของข้อมูล (Error-detecting) และสามารถแก้ไขข้อมูลที่เกิดความผิดพลาด (Error-correcting) ให้เป็นข้อมูลที่ถูกต้อง ซึ่งในขั้นตอนการตรวจสอบและแก้ไขข้อมูลนี้เรียกว่า การถอดรหัสช่องสัญญาณ (Channel decoding)

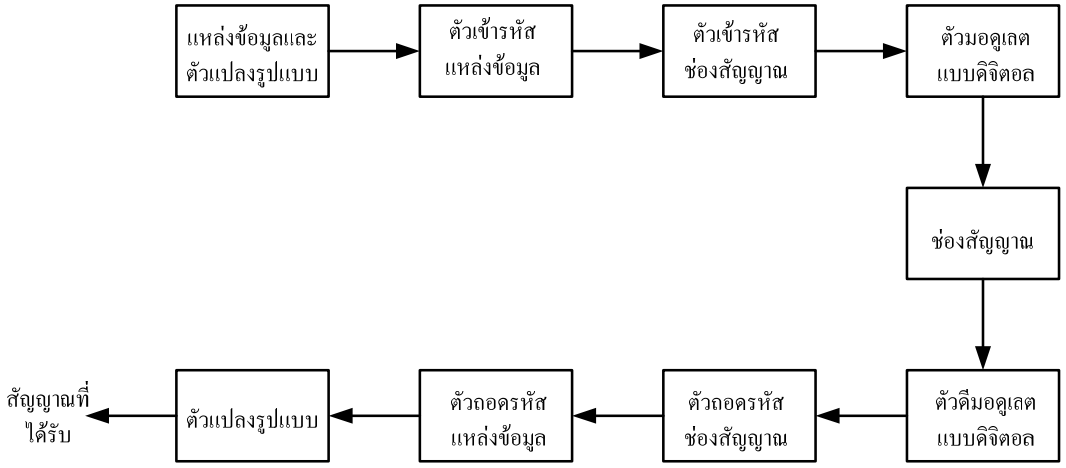
การถอดรหัสช่องสัญญาณมีหลายรูปแบบ รูปแบบหนึ่งที่น่าสนใจ ได้แก่ การถอดรหัสช่องสัญญาณของสัญลักษณ์ที่มีค่าได้หลายระดับขั้นแบบคอนโวลูชัน (Nonbinary convulotional code) ซึ่งโดยส่วนใหญ่วิธีการถอดรหัสนำมาใช้ถอดรหัสช่องสัญญาณโดยทั่วไปจะใช้ได้กับสัญลักษณ์ที่มี 2 ระดับขั้น (Binary symbol) เท่านั้น เนื่องจากวิธีการที่จะถอดรหัสช่องสัญญาณของสัญลักษณ์ที่มีค่าได้หลายระดับขั้นได้นั้น ค่อนข้างซับซ้อนจึงทำให้เครื่องมือที่จะนำมาใช้ถอดรหัสแบบนี้มีไม่มากนัก ทั้งที่มีจุดเด่นที่สามารถแก้ไขความผิดพลาดแบบแถบ (Burst error) ได้ดี เนื่องจากสัญลักษณ์ที่มีค่าได้หลายระดับขั้น มีสัญลักษณ์ขนาดใหญ่ สามารถรวบความผิดพลาดที่เกิดเป็นแถบได้เป็นสัญลักษณ์เพียงไม่กี่ตัว เช่น ค่าความผิดพลาดที่เกิด 8 บิตติดกัน ถ้าใช้วิธีการถอดรหัสที่ใช้กับสัญลักษณ์ที่มี 2 ระดับขั้น ก็หมายถึงจะต้องแก้ไขค่าความผิดพลาดถึง 8 สัญลักษณ์

ในครั้งเดียว แต่ถ้าใช้การถอดรหัสที่สัญลักษณ์มีค่าได้หลายระดับขั้นที่มีขนาด 8 บิตต่อสัญลักษณ์ ก็จะรวบค่าความผิดพลาดนั้นเหลือเพียง 1 สัญลักษณ์แล้วจึงถอดรหัส เป็นต้น

ในทางทฤษฎีแล้วการที่จะถอดรหัสช่องสัญญาณ ของสัญลักษณ์ที่มีค่าได้หลายระดับขั้นแบบคอนโวลูชันได้นั้น สามารถทำได้หลายวิธีด้วยกัน แต่โครงการวิทยานิพนธ์นี้ให้ความสนใจกับวิธีการถอดรหัสแบบเวกเตอร์ซิมโบลดีโคดดิง (Vector symbol decoding) เนื่องจากมีความยืดหยุ่นสามารถประยุกต์ใช้งานได้หลายรูปแบบ แต่ด้วยการทำงานที่ซับซ้อนจึงต้องเลือกใช้ฮาร์ดแวร์ที่สามารถรองรับได้ จึงได้เลือกใช้ชิป Field-Programmable Gate Array (FPGA) มาใช้งาน ชิปตัวนี้นิยมใช้อย่างแพร่หลายในการสร้างวงจรรวมสำหรับงานเฉพาะอย่าง (Application-Specific Integrated Circuit หรือ ASIC) โดยชิป FPGA นี้มีจำนวนโลจิกเกท ได้ตั้งแต่หนึ่งหมื่นโลจิกเกท ถึงสิบล้านโลจิกเกท ซึ่งพอเพียงที่จะใช้สร้างเครื่องถอดรหัสเวกเตอร์ซิมโบลดีโคดดิง การออกแบบด้วยชิป FPGA มีขั้นตอนในการออกแบบและมีภาษาที่ใช้งานเป็นมาตรฐานสากล ซึ่งได้ถูกกำหนดขึ้นโดยสถาบันวิศวกรรมไฟฟ้าและอิเล็กโทรนิกส์ หรือ IEEE

2. ระบบสื่อสาร

การสื่อสาร คือการส่งสารจากผู้ส่งสารไปสู่ผู้รับสารได้อย่างถูกต้อง ดังนั้นระบบสื่อสารจึงประกอบด้วย ผู้ส่งสาร สื่อกลาง และผู้รับสาร แต่ด้วยเทคโนโลยีในปัจจุบันมักกล่าวถึงระบบสื่อสารแบบดิจิตอลในรูปที่ 1 ประกอบด้วยการเข้าและถอดรหัสแหล่งข้อมูล (Source coding) ทำหน้าที่ลดขนาดและเข้ารหัสรักษาความปลอดภัยให้ข้อมูล การเข้าและถอดรหัสช่องสัญญาณ (Channel coding) ทำหน้าที่เพิ่มความน่าเชื่อถือให้ข้อมูล โดยการตรวจสอบและแก้ไขข้อมูลที่ผิดพลาด เนื่องจากสัญญาณรบกวน การมอดูเลตแบบดิจิตอล (Digital modulation) ทำหน้าที่แปลงข้อมูลดิจิตอลให้อยู่ในรูปของคลื่นสัญญาณก่อนที่จะส่งเข้าไปยังช่องสัญญาณ



ภาพที่ 1 แผนผังการสื่อสารแบบดิจิตอล

ที่มา : Proakis (2001)

3. การเข้ารหัสช่องสัญญาณ

การเข้ารหัสช่องสัญญาณ เป็นส่วนประกอบหนึ่งในระบบสื่อสารดิจิทัล ที่ภาคส่งทำหน้าที่เข้ารหัสข้อมูลให้เป็นคำรหัส ที่ภาครับทำหน้าทำการตรวจสอบความผิดพลาดของข้อมูล และสามารถแก้ไขข้อมูลที่เกิดความผิดพลาดให้เป็นข้อมูลที่ถูกต้องได้ โดยการเข้ารหัสจะจำแนกรหัสออกได้เป็นสองประเภท (Lin and Costello, 2004) คือ

3.1 รหัสแบบบล็อก (Block codes)

ประกอบด้วยข้อมูลต้นฉบับจำนวน k สัญลักษณ์ และคำรหัสจำนวน n สัญลักษณ์ อาจเรียกเป็น (n, k) block codes หรือ k/n block codes มีคุณสมบัติไม่มีความจำ (Memoryless property) กล่าวคือรหัสที่ได้จะไม่มีความสัมพันธ์กันระหว่างข้อมูลชุดปัจจุบันกับชุดก่อนหน้า จึงสามารถสร้างฮาร์ดแวร์ให้เป็นแบบวงจร Combinational logic รหัสชนิดนี้ที่รู้จักโดยทั่วไป ได้แก่ Cyclic codes (Prange, 1957), BCH codes (Hocquenghem, 1959; Bose and Ray-Chaudhuri, 1960), Reed-Solomon Code (Reed and Solomon , 1960) เป็นต้น

3.2 รหัสแบบคอนโวลูชัน (Convolutional codes)

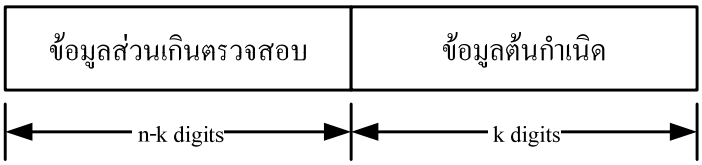
รหัสประเภทนี้ มีคุณสมบัติมีความจำ (Memory property) จำนวน m สัญลักษณ์ โดยจะเรียกได้เป็น (n, k, m) convolutional codes หรือเรียกเป็น k/n convolutional codes ตัวอย่างเช่นในงานวิจัยนี้ใช้รหัสเป็น (3, 2, 2) convolutional code ซึ่งหมายถึง ในขั้นตอนเข้ารหัสในแต่ละคาบนั้นมีข้อมูลต้นฉบับ 2 สัญลักษณ์ นำมาสนธิกับความจำที่เก็บมาจากชุดข้อมูลก่อนหน้านี้ไม่เกิน 2 คาบ เมื่อเข้ารหัสเสร็จแล้วได้เป็น 3 สัญลักษณ์ของคำรหัส ส่วนรหัสที่รู้จักกันโดยทั่วไป เช่น 2/3 convolutional code, Turbo code (Berrou *et al*, 1993) เป็นต้น

นอกจากจะจำแนกตามคุณสมบัติมีความจำหรือไม่มีความจำแล้ว เราอาจจำแนกรหัสช่องสัญญาณตามคุณสมบัติอื่นๆได้อีก เช่น

รหัสที่มีสัญลักษณ์ที่มีค่าสองระดับขั้น (Binary codes) หมายถึง รหัสที่สัญลักษณ์แต่ละตัวจะเป็นสมาชิกของ GF(2) กล่าวคือ สัญลักษณ์จะมีค่าอยู่ในเซตของ 0 กับ 1 ซึ่งทำให้มีการแทนค่าสัญลักษณ์ของ n, k และ m เป็นจำนวนหนึ่งบิต เช่น (7, 4) binary code หมายถึง รหัสที่มีข้อมูลต้นฉบับ 4 บิต และคำรหัส 7 บิต เป็นต้น โดยรหัสที่มีใช้เกือบทั้งหมดเป็นรหัสประเภทนี้

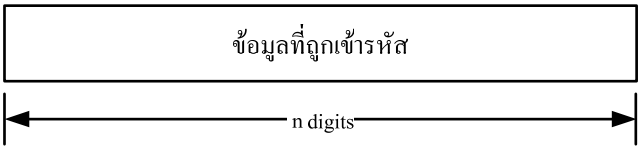
รหัสที่มีสัญลักษณ์ที่มีค่าได้หลายระดับขั้น (Nonbinary codes) หมายถึง รหัสที่สัญลักษณ์แต่ละตัวจะเป็นสมาชิกของ $GF(2^m)$ โดยที่ $m > 1$ ทำให้มีการแทนค่าสัญลักษณ์ของ n, k และ m เป็นจำนวนมากกว่าหนึ่งบิต ตัวอย่างเช่น Reed-Solomon Code และ Vector symbol decoding เป็นต้น

รหัสที่เป็นระบบ (Systematic codes) คำรหัสที่เกิดจากรหัสชนิดนี้ จะประกอบไปด้วยสองส่วนคือ ส่วนของข้อมูลต้นฉบับจำนวน k ตัว และส่วนของข้อมูลส่วนเติมเต็มตรวจสอบ (Redundant checking) จำนวน n-k ตัว (Lin and Costello, 2004)



ภาพที่ 2 รูปแบบของคำรหัสแบบเป็นระบบ

รหัสที่ไม่เป็นระบบ (Nonsystematic codes) คำรหัสของรหัสชนิดนี้จะถูกเข้ารหัสไว้ทั้งหมด ดังนั้นการถอดรหัสจะทำได้ยากกว่ารหัสที่เป็นระบบ แต่จะมีสามารถแก้ไขข้อมูลที่ผิดพลาดได้หลายรูปแบบมากกว่า



ภาพที่ 3 รูปแบบของคำรหัสแบบไม่เป็นระบบ

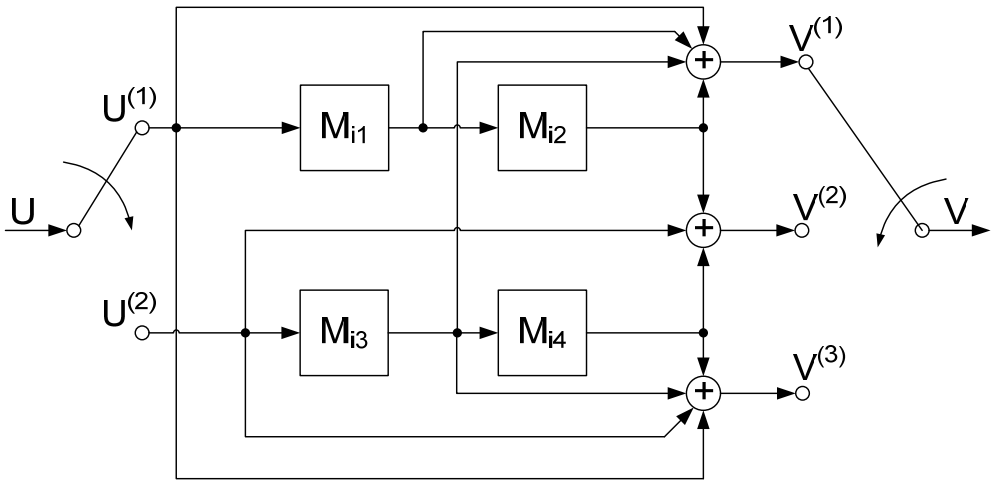
4. การเข้ารหัสคอนโวลูชันสัญลักษณ์ที่มีค่าได้หลายระดับขั้น

วิธีการเข้ารหัสคอนโวลูชันสัญลักษณ์ที่มีค่าได้หลายระดับขั้น ได้มีการสร้างเป็นฮาร์ดแวร์แล้ว โดย Intharasakul และ Tuntoolavest ในปี ค.ศ.2006 ซึ่งมีความคล้ายคลึงกับการเข้ารหัสคอนโวลูชันทั่วไปที่สัญลักษณ์มีค่าสองระดับขั้น ต่างที่การแทนค่าของสัญลักษณ์ โดยในงานวิจัยนี้ใช้รหัสคอนโวลูชัน (3, 2, 2) ซึ่งมีเมทริกซ์ก่อกำเนิด (Generator matrix) ที่มี ฟังก์ชันถ่ายโอน (Transfer function) เท่ากับ G(D) ซึ่งอยู่ในฟิลล์ (Field) ของ GF(2) ดังนี้

$$G(D)=\begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{bmatrix} \quad (1)$$

จากสมการที่ (1) เนื่องจาก G(D) อยู่ในฟิลล์ของ GF(2) ดังนั้น “การดำเนินการทางคณิตศาสตร์ของ G(D) กับค่าอื่นๆที่นำมาดำเนินการทางคณิตศาสตร์ รวมถึงค่าที่สืบทอดต่อไปด้วยนั้น จะเป็นแบบมอดูโล-2 (Modulo-2 operation) ทั้งสิ้น”

สามารถเขียน G(D) เป็นแผนภาพวงจรการเข้ารหัสได้ดังนี้



ภาพที่ 4 แผนภาพวงจรเข้ารหัสคอนโวลูชัน (3, 2, 2)

เนื่องจากรหัสนี้เป็นรหัสเชิงเส้น (Linear code) จึงสามารถหาคำรหัส V(D) จากข้อมูลต้นฉบับ U(D) และเมทริกซ์ก่อกำเนิดที่มีฟังก์ชันถ่ายโอน G(D) ได้เป็น

$$V(D) = U(D) \cdot G(D) \tag{2}$$

แทนค่า G(D) ลงในสมการที่ (2)

$$V(D) = \begin{bmatrix} U^{(1)}(D) & U^{(2)}(D) \end{bmatrix} \begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{bmatrix}$$

$$V(D) = \begin{bmatrix} V^{(1)}(D) \\ V^{(2)}(D) \\ V^{(3)}(D) \end{bmatrix}^T = \begin{bmatrix} U^{(1)}(D) + U^{(1)}(D) \cdot D + U^{(1)}(D) \cdot D^2 + U^{(2)}(D) \cdot D \\ U^{(2)}(D) + U^{(1)}(D) \cdot D^2 + U^{(2)}(D) \cdot D^2 \\ U^{(1)}(D) + U^{(2)}(D) + U^{(2)}(D) \cdot D + U^{(2)}(D) \cdot D^2 \end{bmatrix}^T$$

$$
\begin{aligned}
V^{(1)}(D) &= U^{(1)}(D) + U^{(1)}(D)\cdot D + U^{(1)}(D)\cdot D^2 + U^{(2)}(D)\cdot D \\
V^{(2)}(D) &= U^{(2)}(D) + U^{(1)}(D)\cdot D^2 + U^{(2)}(D)\cdot D^2 \\
V^{(3)}(D) &= U^{(1)}(D) + U^{(2)}(D) + U^{(2)}(D)\cdot D + U^{(2)}(D)\cdot D^2
\end{aligned}
\tag{3}
$$

กำหนดให้ M_{i1}, M_{i2}, M_{i3}, M_{i4} มีค่าเป็น

$$
\begin{aligned}
M_{i1} &= U^{(1)}(D)\cdot D \\
M_{i2} &= U^{(1)}(D)\cdot D^2 \\
M_{i3} &= U^{(2)}(D)\cdot D \\
M_{i4} &= U^{(3)}(D)\cdot D^2
\end{aligned}
\tag{4}
$$

สามารถนำสมการที่ (3) และ (4) มาเขียนให้สัมพันธ์กับแผนภาพวงจรการเข้ารหัส จะได้ว่า

$$
\begin{aligned}
V^{(1)} &= U^{(1)} + M_{i1} + M_{i2} + M_{i3} \\
V^{(2)} &= U^{(2)} + M_{i2} + M_{i4} \\
V^{(3)} &= U^{(1)} + U^{(2)} + M_{i3} + M_{i4}
\end{aligned}
\tag{5}
$$

กำหนดให้ลำดับของข้อมูลที่ป้อนเข้าและออกจากวงจรนี้เป็นดังนี้

$$
\begin{aligned}
U &= \left(A_1B_1, A_2B_2, A_3B_3, ..., A_kB_k, \vec{0}\vec{0}, \vec{0}\vec{0}\right) \\
V &= \left(V_1V_2V_3, V_4V_5V_6, V_7V_8V_9, ..., V_{3(k+2)-2}V_{3(k+2)-1}V_{3(k+2)}\right)
\end{aligned}
\tag{6}
$$

จากสมการที่ (6) ค่า A, B และ V แต่ละตัวเป็นเวกเตอร์ของสัญลักษณ์ขนาด q บิต ซึ่งในงานวิจัยนี้ใช้ q = 32 บิต ส่วนในสองลำดับสุดท้ายของ U เป็นการป้อนข้อมูลที่เป็นศูนย์เข้าไปปิดท้ายข้อมูล รวมทั้งเป็นการล้างค่าที่ค้างอยู่ในหน่วยความจำของวงจรเข้ารหัสด้วย โดยจะสามารถแจกแจงลำดับของแต่ละปมของวงจรได้เป็น

$$
\begin{aligned}
U^{(1)} &= \left(A_1, A_2, A_3, ..., A_k, \vec{0}, \vec{0}\right) \\
U^{(2)} &= \left(B_1, B_2, B_3, ..., B_k, \vec{0}, \vec{0}\right) \\
V^{(1)} &= \left(V_1, V_4, V_7, ..., V_{3(k+2)-2}\right) \\
V^{(2)} &= \left(V_2, V_5, V_8, ..., V_{3(k+2)-1}\right) \\
V^{(3)} &= \left(V_3, V_6, V_9, ..., V_{3(k+2)}\right)
\end{aligned} \tag{7}
$$

เมื่อป้อนลำดับเหล่านี้ลงในวงจรการเข้ารหัส สามารถเขียนลำดับเหล่านี้ให้อยู่ในรูปสมการที่ (5) โดยที่กำหนดให้ในคาบที่ 1 มีค่าเริ่มต้นของ M_{i1}, M_{i2}, M_{i3} และ M_{i4} เป็นศูนย์ จะได้ค่าของคำรหัสเป็นดังนี้

$$
\begin{aligned}
V_1 &= A_1 + M_{i1} + M_{i2} + M_{i3} = A_1 \\
V_2 &= B_1 + M_{i2} + M_{i4} = B_1 \\
V_3 &= A_1 + B_1 + M_{i3} + M_{i4} = A_1 + B_1
\end{aligned} \tag{8}
$$

คาบที่ 2 มี $M_{i1} = A_1$, $M_{i2} = 0$, $M_{i3} = B_1$ และ $M_{i4} = 0$ ได้ค่าคำรหัสเป็น

$$
\begin{aligned}
V_4 &= A_2 + A_1 + M_{i2} + B_1 = A_2 + A_1 + B_1 \\
V_5 &= B_2 + M_{i2} + M_{i4} = B_2 \\
V_6 &= A_2 + B_2 + B_1 + M_{i4} = A_2 + B_2 + B_1
\end{aligned} \tag{9}
$$

คาบที่ 3 มี $M_{i1} = A_2$, $M_{i2} = A_1$, $M_{i3} = B_2$ และ $M_{i4} = B_1$ ได้ค่าคำรหัสเป็น

$$
\begin{aligned}
V_7 &= A_3 + A_2 + A_1 + B_2 \\
V_8 &= B_3 + A_1 + B_1 \\
V_9 &= A_3 + B_3 + B_2 + B_1
\end{aligned} \tag{10}
$$

ด้วยวิธีเดียวกันนี้จะได้ค่าคำรหัสอื่นๆ ดังนี้

$$
\begin{aligned}
&V_{10} = A_4 + A_3 + A_2 + B_3 \\
&V_{11} = B_4 + A_2 + B_2 \\
&V_{12} = A_4 + B_4 + B_3 + B_2 \\
&V_{13} = A_5 + A_4 + A_3 + B_4 \\
&V_{14} = B_5 + A_3 + B_3 \\
&V_{15} = A_5 + B_5 + B_4 + B_3 \\
&V_{16} = A_6 + A_5 + A_4 + B_5 \\
&V_{17} = B_6 + A_4 + B_4 \\
&V_{18} = A_6 + B_6 + B_5 + B_4 \\
&\cdots \\
&V_{3k-2} = A_k + A_{k-1} + A_{k-2} + B_{k-1} \\
&V_{3k-1} = B_k + A_{k-2} + B_{k-2} \\
&V_{3k} = A_k + B_k + B_{k-1} + B_{k-2} \\
&V_{3(k+1)-2} = \vec{0} + A_k + A_{k-1} + B_k = A_k + A_{k-1} + B_k \\
&V_{3(k+1)-1} = \vec{0} + A_{k-1} + B_{k-1} = A_{k-1} + B_{k-1} \\
&V_{3(k+1)} = \vec{0} + \vec{0} + B_k + B_{k-1} = B_k + B_{k-1} \\
&V_{3(k+2)-2} = \vec{0} + \vec{0} + A_k + \vec{0} = A_k \\
&V_{3(k+2)-1} = \vec{0} + A_k + B_k = A_k + B_k \\
&V_{3(k+2)} = \vec{0} + \vec{0} + \vec{0} + B_k = B_k
\end{aligned}
\tag{11}
$$

5. การดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ

การดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ เป็นการเรียงข้อมูลที่เข้ารหัสไว้ด้วยรหัสคอนโวลูชันแบบไม่เป็นระบบ เมื่อรหัสไม่เป็นระบบแล้วนั้น คำรหัสจะไม่มีการแบ่งส่วนของข้อมูลต้นฉบับ กับข้อมูลส่วนเติมเต็มตรวจสอบออกจากกัน การแปลงข้อมูลกลับจากคำรหัสไปเป็นข้อมูลต้นฉบับ จึงไม่สามารถแยกข้อมูลออกมาได้โดยตรง ต้องนำคำรหัสมาผ่านวงจรดึงรหัสคอนโวลูชันแบบกลับคืนข้อมูลต้นฉบับ จึงจะแยกข้อมูลต้นฉบับออกมาได้ แต่ทฤษฎีที่รองรับสำหรับการสร้างวงจรชนิดนี้ มีเฉพาะสำหรับสัญลักษณ์สองระดับขั้นเท่านั้น แต่ในงานวิจัยนี้ใช้การเข้ารหัสแบบคอนโวลูชันแบบไม่เป็นระบบและใช้กับสัญลักษณ์หลายระดับขั้น จึงได้คิดหาวิธีสร้างวงจรขึ้นมาเอง ดังนี้

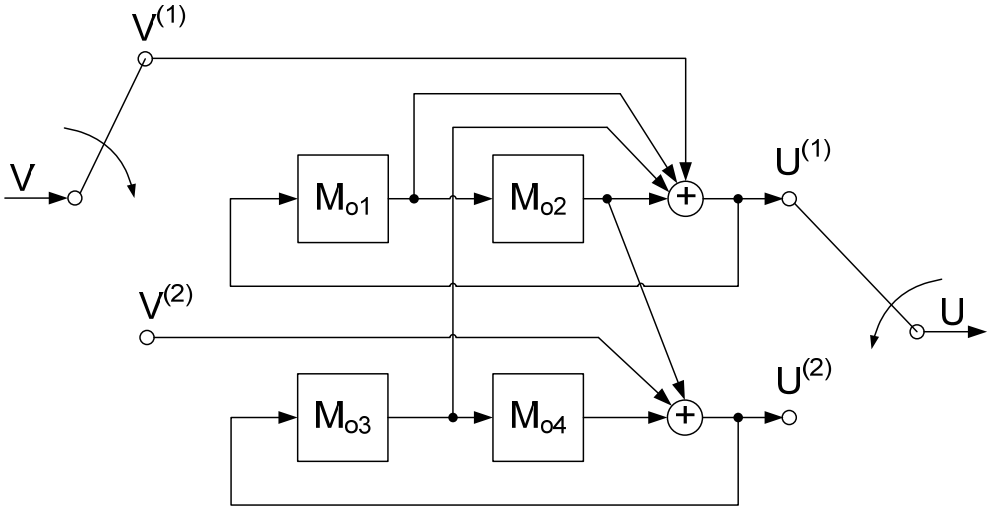
จากสมการที่ (3) ทำการย้ายพจน์ $U^{(i)}(D)$ ทุกตัว สลับข้างกับพจน์ $V^{(i)}(D)$ ทุกตัว โดยที่ i เท่ากับจำนวนเต็มใดๆ จะได้เป็น

$$U^{(1)}\left(D\right) = V^{(1)}(D) + U^{(1)}(D)\cdot D + U^{(1)}(D)\cdot D^2 + U^{(2)}(D)\cdot D \tag{12}$$

$$U^{(2)}\left(D\right) = V^{(2)}(D) + U^{(1)}(D)\cdot D^2 + U^{(2)}(D)\cdot D^2 \tag{13}$$

$$U^{(1)}(D) + U^{(2)}(D) = V^{(3)}\left(D\right) + U^{(2)}(D)\cdot D + U^{(2)}(D)\cdot D^2 \tag{14}$$

สังเกตได้ว่า มีเพียงสมการที่ (12) กับ (13) เท่านั้นก็สามารถหาค่าของ U(D) ได้ครบทั้งสองตัวแล้ว จึงไม่จำเป็นต้องใช้สมการ (14) ในการหาค่าของ U(D) แต่อาจนำไปใช้ในการตรวจทานความถูกต้องของ U(D) ได้ แต่ในที่นี้ต้องการหาค่า U(D) เท่านั้นจึงนำสมการที่ (12) กับ (13) ไปเขียนเป็นวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ได้ ซึ่งจะสังเกตได้ว่าเป็นวงจรลักษณ์ย้อนกลับ ได้ดังนี้



ภาพที่ 5 แผนภาพวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบ (3, 2, 2) กลับคืนข้อมูลต้นฉบับ

เขียนสมการที่ (12) กับ (13) ใหม่ให้สัมพันธ์กับแผนภาพวงจร ดังนี้

$$\begin{aligned} U^{(1)} &= V^{(1)} + M_{o1} + M_{o2} + M_{o3} \\ U^{(2)} &= V^{(2)} + M_{o2} + M_{o4} \end{aligned} \tag{15}$$

โดยที่

$$\begin{aligned} M_{o1} &= U^{(1)}(D) \cdot D \\ M_{o2} &= U^{(1)}(D) \cdot D^2 \\ M_{o3} &= U^{(2)}(D) \cdot D \\ M_{o4} &= U^{(3)}(D) \cdot D^2 \end{aligned} \tag{16}$$

ป้อนค่าสมการที่ (7) ลงในวงจร โดยเขียนให้อยู่ในรูปสมการที่ (15) และในคาบที่ 1 กำหนดให้ค่าเริ่มต้นของ M_{o1}, M_{o2}, M_{o3} และ M_{o4} เป็นศูนย์ จะได้ค่าของข้อมูลต้นฉบับดังนี้

$$\begin{aligned} U^{(1)} &= A_1 = V_1 + M_{o1} + M_{o2} + M_{o3} = V_1 \\ U^{(2)} &= B_1 = V_2 + M_{o2} + M_{o4} = V_2 \end{aligned} \tag{17}$$

ในคาบที่สอง $M_{o1} = V_1$, $M_{o2} = 0$, $M_{o3} = V_2$ และ $M_{o4} = 0$

$$\begin{aligned} U^{(1)} &= A_2 = V_4 + V_1 + M_{o2} + V_2 = V_4 + V_1 + V_2 \\ U^{(2)} &= B_2 = V_5 + M_{o2} + M_{o4} = V_5 \end{aligned} \tag{18}$$

ในคาบที่สาม $M_{o1} = V_4 + V_1 + V_2$, $M_{o2} = V_1$, $M_{o3} = V_5$ และ $M_{o4} = V_2$

$$\begin{aligned} U^{(1)} &= A_3 = V_7 + (V_4 + V_1 + V_2) + V_1 + V_5 = V_7 + V_4 + V_2 + V_5 \\ U^{(2)} &= B_3 = V_8 + V_1 + V_5 \end{aligned} \tag{19}$$

ใช้วิธีเดียวกันนี้ในคาบถัดไปจะได้ค่าของข้อมูลต้นฉบับทุกตัวออกมา

การตรวจสอบความถูกต้องของวงจร ทำได้โดยป้อนคำรหัสจากสมการที่ (8) ถึง (11) ลงไปยังสมการที่ (17) ถึง (19) ยกตัวอย่างเช่น ต้องการทราบว่าค่า A_3 และ B_3 ที่หามาจากสมการที่ (19) มีความถูกต้องหรือไม่ ให้นำค่าคำรหัสจากสมการที่ (8) ถึง (10) มาแทนค่าลงสมการที่ (19) ดังนี้

$$A_3 = V_7 + V_4 + V_2 + V_5$$
$$= (A_3 + A_2 + A_1 + B_2) + (A_2 + A_1 + B_1) + B_1 + B_2 = A_3$$

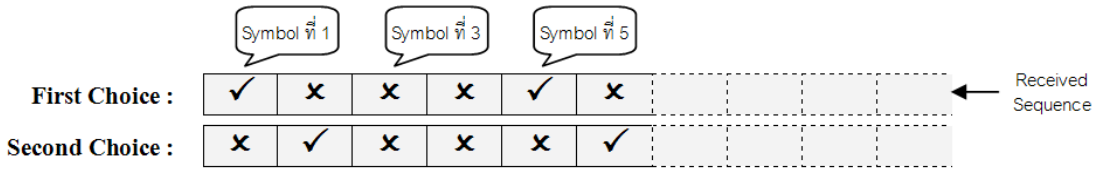
$$B_3 = V_8 + V_1 + V_5 = (B_3 + A_1 + B_1) + A_1 + B_2 = B_3$$

ทั้งสองสมการเท่ากันทั้งสองฝั่ง จึงสรุปได้ว่าค่าของ $U^{(1)}$ และ $U^{(2)}$ ที่คำนวณได้จากสมการที่ (19) มีความถูกต้อง

วิธีการสร้างวงจรนี้ สามารถใช้ได้กับรหัสคอนโวลูชันแบบไม่เป็นระบบ ที่มีเมทริกซ์ก่อกำเนิดที่มีฟังก์ชันถ่ายโอน G(D) ต่างจากตัวอย่างนี้ได้อีกด้วย โดยวงจรที่ได้จะสามารถใช้ได้กับสัญลักษณ์ที่มีค่าหลายระดับขั้น และสัญลักษณ์ที่มีค่าสองระดับขั้น

6. ชุดข้อมูลตัวเลือก

ชุดข้อมูลตัวเลือกเป็นวิธีการนำข้อมูลมาช่วยในการถอดรหัส เพื่อเพิ่มประสิทธิภาพในการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง ชุดข้อมูลตัวเลือกในงานวิจัยนี้ประกอบด้วยตัวเลือกอันดับหนึ่ง (First choice) และตัวเลือกอันดับสอง (Second choice) เพื่อให้เข้าใจถึงที่มาของตัวเลือกทั้งสอง จะอธิบายถึงแบบจำลองสมมติว่า มีสายอากาศรับสัญญาณแบบแถวลำดับ (Array antenna) ต่ออยู่กับเครื่องรับที่มีความสามารถในการเปรียบระดับสัญญาณของสายอากาศแต่ละต้น ข้อมูลที่รับมาจากสายอากาศต้นที่มีระดับสัญญาณดีที่สุดจะถูกกำหนดเป็นตัวเลือกอันดับหนึ่ง และข้อมูลที่ได้จากสายอากาศต้นที่มีระดับสัญญาณดีรองมาเป็นอันดับสองกำหนดให้เป็นตัวเลือกอันดับสอง โดยแบบจำลองนี้มีข้อกำหนดสองประการคือ ประการแรกตัวเลือกอันดับหนึ่งจะรับข้อมูลได้เสมอ ไม่ว่าข้อมูลที่ได้รับจะถูกหรือผิดมากเท่าใดก็ตาม ประการที่สองถ้าสัญลักษณ์ในตัวเลือกอันดับหนึ่งตำแหน่งที่ n ถูกต้อง สัญลักษณ์ของตัวเลือกอันดับสองตำแหน่งที่ n ต้องไม่เป็นสัญลักษณ์ที่ถูกต้อง



ภาพที่ 6 รูปแสดงตัวอย่างชุดข้อมูลขาเข้าสองตัวเลือก

เมื่อ ✓ หมายถึง สัญลักษณ์ที่ถูกต้อง

✗ หมายถึง สัญลักษณ์ที่มีความผิดพลาด

อย่างไรก็ตามการนำชุดข้อมูลตัวเลือกมาใช้นี้ เป็นเพียงส่วนเสริมที่จะนำมาช่วยในการถอดรหัสเท่านั้น ซึ่งตัวถอดรหัสจะนำชุดข้อมูลตัวเลือกมาใช้หรือไม่ก็ได้ เพราะการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง มีวิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination) เป็นวิธีหลักที่ใช้ในการถอดรหัส

7. การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง

การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง (Vector symbol decoding) เป็นรหัสวิธีการถอดรหัสที่สามารถถอดรหัสของรหัสที่มีสัญลักษณ์ที่มีค่าได้หลายระดับขั้น (Nonbinary code) ที่มีขนาดใหญ่มากถึง 32 หรือ 64 บิตได้ ซึ่งจะทำให้สามารถถอดรหัสของข้อมูลที่มีความผิดพลาดแบบแถบ (Burst error) ได้ดี ซึ่งถูกโดยคิดค้นเมื่อปี ค.ศ. 1990 โดย Metzner และ Kapurowski ถัดมาในปี ค.ศ. 1991 Seo ได้นำวิธีเวกเตอร์ซิมโบลดีโคดดิงไปประยุกต์ใช้กับรหัสแบบคอนโวลูชันนอลแบบเป็นระบบ (Systematic convolutional code) ต่อมาในปี ค.ศ. 1999 Haslach และ Vinck ได้นำมาใช้กับรหัสแบบบล็อก (Block code) สำหรับการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง ที่สามารถใช้ได้กับรหัสแบบคอนโวลูชันนอลแบบไม่เป็นระบบ (Nonsystematic convolutional code) ที่มีชุดข้อมูล 2 ชุด ซึ่งได้เริ่มพัฒนาตั้งแต่ปี ค.ศ. 2000 โดย Tuntoolavest และ Metzner (Metzner, 2000, 2002, 2003; Tuntoolavest and Metzner, 2001, 2002; Tuntoolavest, 2004) ได้ถูกคิดค้นและพัฒนาเรื่อยมา ให้สามารถใช้งานได้กับการสื่อสารในรูปแบบต่างๆ อาทินำไปประยุกต์ใช้กับระบบการสื่อสารไร้สาย เป็นต้น

7.1 ขั้นตอนการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง

วิธีการทำงานจะเป็นแบบลำดับขั้น (Sequential method) จึงสามารถสรุปขั้นตอนการทำงานสำหรับรหัสคอนโวลูชัน (n, k, m) ได้โดยลำดับดังนี้

7.1.1 ตัวถอดรหัสจะรับข้อมูลเข้ามาแบบเป็นลำดับ เมื่อรับข้อมูลตัวเลือกอันดับหนึ่ง (First choice) เข้ามาครบ n สัญลักษณ์แล้ว จะนำข้อมูล n สัญลักษณ์นี้มาคำนวณหาค่าซินโดรม (Syndrome)

7.1.2 หากค่าซินโดรมเป็นศูนย์ แสดงว่า n สัญลักษณ์นี้ตรวจไม่พบข้อมูลที่ผิดพลาด จึงรับข้อมูล n สัญลักษณ์ถัดไปมาถอดรหัส

7.1.3 ถ้าค่าซินโดรมไม่เป็นศูนย์ แสดงว่า n สัญลักษณ์นี้ตรวจพบข้อมูลที่ผิดพลาด ตัวถอดรหัสจะพยายามแก้ไขข้อมูลที่ผิดพลาด ด้วยวิธีถอดรหัสด้วยซินโดรมเดียว (Decode with one syndrome) โดยนำข้อมูลตัวเลือกอันดับสอง (Second choice) เข้ามา n สัญลักษณ์ที่สัมพันธ์กับข้อมูลตัวเลือกอันดับหนึ่งนี้ มาช่วยถอดรหัส

7.1.4 หากวิธีถอดรหัสด้วยซินโดรมเดียว สามารถแก้ไขข้อมูลที่ผิดพลาดได้ จะไปรับข้อมูล n สัญลักษณ์ถัดไปมาถอดรหัส

7.1.5 ถ้าไม่สามารถถอดรหัสด้วยซินโดรมเดียวได้ จะพยายามทำการแก้ไขข้อมูลที่ผิดพลาดต่อไป ด้วยวิธีถอดรหัสด้วยซินโดรมหลายตัว (Decode with many syndrome) โดยจะรับข้อมูลตัวเลือกอันดับหนึ่งและชุดที่สองของ n สัญลักษณ์ถัดไปมาช่วยถอดรหัส ในวิธีนี้จะแบ่งเป็น 2 ขั้นตอนย่อย คือ ขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง (Correct with second choice) และขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination)

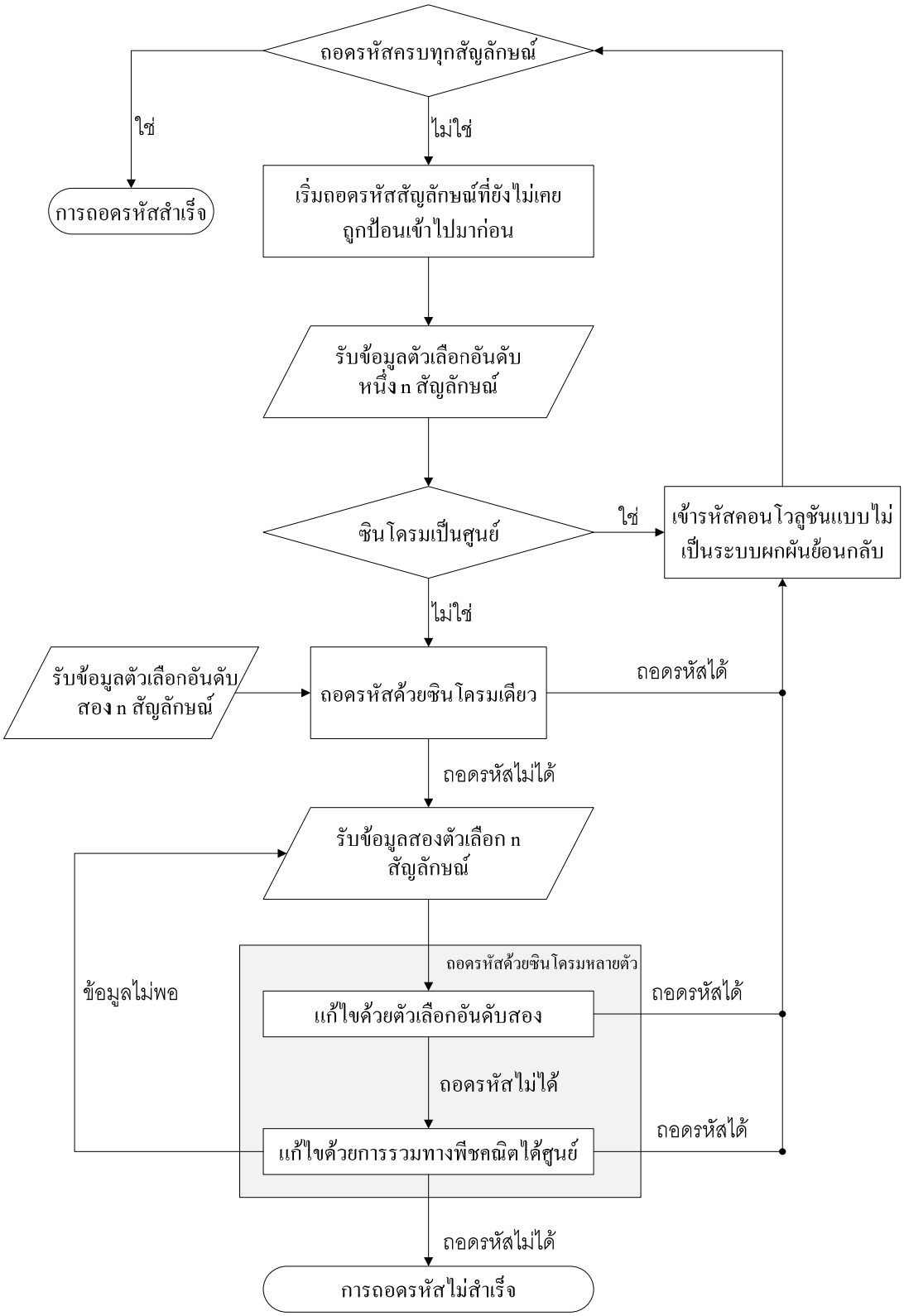
7.1.6 ขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง จะนำข้อมูลตัวเลือกอันดับหนึ่งและชุดที่สองของทุก n สัญลักษณ์ถัดไปที่รับเข้ามาช่วยถอดรหัส มาใช้ในกระบวนการถอดรหัส หากสามารถถอดรหัสได้จะไปเริ่มถอดรหัสในข้อแรกโดยรับข้อมูล n สัญลักษณ์ถัดไปที่ยังไม่ได้ถอดรหัส

7.1.7 หากถอดรหัสด้วยขั้นตอนแก้ไขด้วยตัวเลือกอันดับสองไม่ได้ จะใช้ขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ โดยนำข้อมูลตัวเลือกอันดับหนึ่งของทุก n สัญลักษณ์ถัดไปที่รับเข้ามาช่วยถอดรหัสทั้งหมด มาถอดรหัสด้วยกระบวนการของพีชคณิต หากสามารถถอดรหัสได้จะไปเริ่มถอดรหัสในข้อแรกโดยรับข้อมูล n สัญลักษณ์ถัดไปที่ยังไม่ได้ถอดรหัส

7.1.8 ถ้าถอดรหัสด้วยขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ไม่ได้ จะต้องรับข้อมูลเพิ่มอีก n สัญลักษณ์ถัดๆไปเพื่อมาช่วยถอดรหัส โดยจะไปเริ่มถอดรหัสที่ขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง แต่ถ้าหากวิธีการรับข้อมูลเพิ่มไม่สามารถช่วยถอดรหัสช่วยถอดรหัสได้แล้ว ตัวถอดรหัสสามารถหยุดกระบวนการถอดรหัสได้ โดยจะถือว่าการถอดรหัสล้มเหลว (Decoding failure) จะต้องแจ้งให้ภาคส่งทวนข้อมูลกลับมาใหม่อีกครั้ง

7.1.9 สำหรับข้อมูลที่ผ่านขั้นตอนแก้ไขข้อมูลที่ผิดพลาดแล้ว จะต้องนำมาเรียงข้อมูลกลับด้วยกระบวนการดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับก่อน จึงเสร็จสิ้นกระบวนการถอดรหัส

จากรูปที่ 7 เป็นแผนภาพแสดงขั้นตอนการถอดรหัสสำหรับ 1 คำรหัส ในขั้นแรกตัวถอดรหัสจะตรวจสอบว่า ยังมีสัญลักษณ์ใดในคำรหัสที่จะต้องนำไปถอดรหัส แล้วจึงนำสัญลักษณ์นั้น n ตัวเข้าไปหาซินโดรม ถ้าซินโดรมเป็นศูนย์ แสดงว่าตัวถอดรหัสไม่พบค่าความผิดพลาด แล้วจึงส่งสัญลักษณ์นั้นออกไปวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ เพื่อหาข้อมูลต้นฉบับแล้วเริ่มถอดรหัสต่อหรือหยุดถอดรหัสเมื่อสัญลักษณ์หมด แต่ถ้าซินโดรมไม่เป็นศูนย์ จะถอดรหัสขั้นแรกด้วยถอดรหัสด้วยซินโดรมเดียว โดยจะรับข้อมูลตัวเลือกอันดับสองมา n สัญลักษณ์ มาเข้าการถอดรหัส ถ้าถอดรหัสได้ก็ส่งออกไปหาข้อมูลต้นฉบับ แต่ถ้าถอดรหัสไม่ได้จะถอดรหัสขั้นที่สองด้วยถอดรหัสด้วยซินโดรมหลายตัว โดยจะต้องรับข้อมูลทั้งสองตัวเลือกมาอย่างละ n สัญลักษณ์ แล้วเริ่มถอดรหัสด้วยวิธีแก้ไขด้วยตัวเลือกอันดับสอง ถ้าถอดรหัสได้จะออกไปหาข้อมูลต้นฉบับ แต่ถ้ายังไม่ได้จะถอดรหัสด้วยวิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ ถ้าถอดรหัสได้จะออกไปหาข้อมูลต้นฉบับ ถ้าถอดรหัสได้แต่ยังมีค่าความผิดพลาดหลงเหลือจะถือว่าการถอดรหัสสำหรับคำรหัสนี้ทำการถอดรหัสล้มเหลว ถ้าตัวถอดรหัสตรวจสอบแล้วว่า จำนวนของค่าความผิดพลาดที่ตรวจเจอในขั้นตอนของการรวมทางพีชคณิตได้ศูนย์มีมากเกินไป จะถือว่ามีข้อมูลไม่พอที่จะใช้ถอดรหัส จึงทำการขอข้อมูลทั้งสองตัวเลือกมาเพิ่มอย่างละ n สัญลักษณ์ แล้วเริ่มถอดรหัสด้วยวิธีแก้ไขด้วยตัวเลือกอันดับสอง



ภาพที่ 7 แผนภาพขั้นตอนการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับ 1 คำรหัส

7.2 การคำนวณหาค่าซินโดรม

ซินโดรม เป็นค่าที่ใช้สำหรับตรวจหาความผิดพลาดของข้อมูล ปกติแล้วค่าซินโดรมจะเป็นศูนย์เมื่อข้อมูลที่นำมาตรวจสอบนั้นเป็นคำรหัส และจะไม่เท่ากับศูนย์เมื่อข้อมูลนั้นไม่ใช่คำรหัส สำหรับรหัสเชิงเส้นมีหลักการที่เกี่ยวข้องในการหาค่าซินโดรมดังนี้ พิจารณารหัสเชิงเส้น (n, k) ที่มีเมทริกซ์ก่อกำเนิด (Generator matrix) G และเมทริกซ์ตรวจสอบภาวะคู่หรือคี่ (Parity-check matrix) H เขียนความสัมพันธ์ระหว่างเมทริกซ์สองตัวนี้ได้เป็น

$$G \cdot H^T = 0 \tag{20}$$

มีคำรหัส (Code word) **v** ที่เกิดมาจากข้อมูลต้นฉบับ (Data) **u** กับเมทริกซ์ก่อกำเนิด G

$$\mathbf{v} = \mathbf{u} \cdot G \tag{21}$$

เมื่อนำคำรหัส **v** ส่งไปในช่องสัญญาณที่มีค่าความผิดพลาด (Error) เป็น **e** จะได้ข้อมูลที่รับได้ (Received sequence) เป็น **r**

$$\mathbf{r} = \mathbf{v} + \mathbf{e} \tag{22}$$

นำข้อมูลที่รับได้นี้ไปคำนวณหาซินโดรม (Syndrome) ดังสมการ

$$S = \mathbf{r} \cdot H^T \tag{23}$$

แทนค่าสมการที่ (22) ลงในสมการที่ (23) และแทนค่าสมการที่ (21) ลงในสมการที่ (22)

$$\begin{aligned} S &= (\mathbf{v}+\mathbf{e}) \cdot H^T = \mathbf{v} \cdot H^T + \mathbf{e} \cdot H^T \\ &= (\mathbf{u} \cdot G) \cdot H^T + \mathbf{e} \cdot H^T \\ &= \mathbf{u} \cdot (0) + \mathbf{e} \cdot H^T \\ S &= \mathbf{e} \cdot H^T \end{aligned} \tag{24}$$

จากสมการที่ (24) พบว่า ค่าซินโดรมจะเป็นศูนย์ เมื่อค่าความผิดพลาดเป็นศูนย์ หรือสรุปได้เป็น ค่าซินโดรมจะเป็นศูนย์ก็ต่อเมื่อไม่มีค่าความผิดพลาดในข้อมูลที่รับได้

จากหลักการข้างต้น สามารถหลักการนี้มาประยุกต์ในการหาค่าซินโดรมของงานวิจัยนี้ได้ แต่รหัสที่ใช้เป็นคอนโวลูชันทำให้ขนาดของซินโดรมสามารถปรับเปลี่ยนได้ตามความเหมาะสม เนื่องจากเมทริกซ์ตรวจสอบภาวะคู่หรือคี่มีลักษณะเป็นเมทริกซ์กึ่งอนันต์ (Semi-infinite matrix) ซึ่งสามารถขยายออกหรือย่อลงได้ตามความต้องการ โดยงานวิจัยนี้ได้ใช้เมทริกซ์ตรวจสอบภาวะคู่หรือคี่ (Tuntoolavest and Metzner, 2002) ดังนี้

$$H = \begin{bmatrix}
\boxed{\begin{matrix}1&1&1\\0&1&1\\0&1&0\\1&0&0\\1&1&1\end{matrix}} & \boxed{\begin{matrix}0&0&0\\1&1&1\\0&1&1\\0&1&0\\1&0&0\end{matrix}} & \boxed{\begin{matrix}0&0&0\\0&0&0\\1&1&1\\0&1&1\\0&1&0\\1&0&0\end{matrix}} & & & \\
\begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \cdots & \cdots & \cdots \\
\begin{matrix}0&0&0\\0&0&0\\1&1&1\\0&1&1\\0&1&0\\1&0&0\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \boxed{\begin{matrix}0&0&0\\0&0&0\\1&1&1\\0&1&1\\0&1&0\\1&0&0\end{matrix}} & & & \\
\begin{matrix}0&0&0\\0&0&0\\0&0&0\\1&1&1\\0&1&0\\1&0&0\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \cdots & \cdots & \cdots \\
\vdots & \vdots & \vdots & & & \\
\begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \begin{matrix}0&0&0\\0&0&0\\0&0&0\\0&0&0\\0&0&0\\1&1&1\end{matrix} & \cdots & \cdots & \boxed{\begin{matrix}1&1&1\\0&1&1\\0&1&0\\1&0&0\\1&1&1\end{matrix}}
\end{bmatrix} \qquad (25)$$

จากสมการที่ (6) และสมการที่ (22) กำหนดให้ข้อมูลที่รับได้ มีลำดับของข้อมูลดังนี้

$$R = \left(R_1R_2R_3, R_4R_5R_6, R_7R_8R_9, ..., R_{3(k+2)-2}R_{3(k+2)-1}R_{3(k+2)}\right) \qquad (26)$$

โดยที่ R แต่ละตัวแทนค่าเวกเตอร์ของสัญลักษณ์ขนาด q บิต ซึ่งงานวิจัยนี้ ใช้ q = 32 ในการหาค่าซินโดรมจะใช้สมการที่ (23) ในการคำนวณ แต่จะแบ่งคำนวณซินโดรมออกเป็นครั้งละแถว โดยใช้แถวของเมทริกซ์ในสมการที่ (25) เป็นตัวแบ่ง ตัวอย่างเช่น ในการหาค่าซินโดรมแถวที่หนึ่ง จะใช้แถวที่หนึ่งของเมทริกซ์ในสมการที่ (25) มาคำนวณ

$$s_1 = r \cdot H_1^T = \begin{bmatrix} R_1 & R_2 & R_3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T = R_1 + R_2 + R_3$$

$$s_1 = R_1 + R_2 + R_3 \qquad (27)$$

หรือในการหาค่าซินโดรมแถวที่สอง จะใช้แถวที่สองของเมทริกซ์ในสมการที่ (25) มาคำนวณ

$$s_2 = r \cdot H_2^T = \begin{bmatrix} R_1 & R_2 & R_3 & R_4 & R_5 & R_6 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

$$s_2 = R_2 + R_3 + R_4 + R_5 + R_6 \tag{28}$$

หรือในการหาค่าซินโดรมแถวที่หก จะใช้แถวที่หกของเมทริกซ์ในสมการที่ (25) มาคำนวณ

$$s_6 = r \cdot H_6^T$$

$$= \begin{bmatrix} R_1 & R_2 & R_3 & \cdots & R_{18} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$s_6 = R_4 + R_5 + R_6 + R_7 + R_{11} + R_{14} + R_{15} + R_{16} + R_{17} + R_{18} \tag{29}$$

ด้วยวิธีเดียวกันนี้สามารถหาซินโดรมในแถวที่เจ็ดได้เป็น

$$s_7 = R_7 + R_8 + R_9 + R_{10} + R_{14} + R_{17} + R_{18} + R_{19} + R_{20} + R_{21} \tag{30}$$

ในกรณีที่ทราบค่าคำรหัสเป็นบางส่วนแล้ว และต้องการหาค่าซินโดรม สามารถนำคำรหัสที่ทราบค่านั้นมาลดรูปสมการที่ใช้หาค่าซินโดรมได้ เช่น ในการถอดรหัส ได้ทำการถอดรหัสคำรหัสไปแล้ว 15 สัญลักษณ์คือ $\left(V_1V_2V_3, V_4V_5V_6, V_7V_8V_9, V_{10}V_{11}V_{12}, V_{13}V_{14}V_{15}\right)$ แล้วต้องการหาค่าซินโดรมในแถวที่หก สามารถลดรูปสมการที่ (29) ได้โดยใช้สมการที่ (8) ถึง (11) ดังนี้

$$\begin{aligned} s_6 &= R_4 + R_5 + R_6 + R_7 + R_{11} + R_{14} + R_{15} + R_{16} + R_{17} + R_{18} \\ &= V_4 + V_5 + V_6 + V_7 + V_{11} + V_{14} + V_{15} + R_{16} + R_{17} + R_{18} \\ &= (A_2 + A_1 + B_1) + B_2 + (A_2 + B_2 + B_1) + (A_3 + A_2 + A_1 + B_2) \\ &\quad + (B_4 + A_2 + B_2) + (B_5 + A_3 + B_3) + (A_5 + B_5 + B_4 + B_3) \\ &\quad + R_{16} + R_{17} + R_{18} \\ s_6 &= R_{16} + R_{17} + R_{18} + A_5 \end{aligned} \tag{31}$$

ในวิธีเดียวกันนี้สามารถหาค่าซินโดรมในแถวที่เจ็ดได้เป็น

$$s_7 = R_{17} + R_{18} + R_{19} + R_{20} + R_{21} + A_4 + B_5 \tag{32}$$

เมื่อนำสมการที่ (27) เทียบกับสมการที่ (31) และสมการที่ (28) เทียบกับสมการที่ (32) พบว่ามีรูปแบบที่คล้ายกัน เพียงแต่มีพจน์ของ A หรือ B เพิ่มขึ้นมา โดยค่าของ A และ B นี้สามารถคำนวณได้จากการนำคำรหัสที่ถอดรหัสแล้ว ไปผ่านวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ซึ่งค่า A และ B ก็คือค่าของ M_0 ในวงจรนั่นเอง ในทางปฏิบัติแล้ววงจรส่วนที่ใช้คำนวณค่าซินโดรม จะทำการสัมพันธ์กันกับวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ

สามารถสรุปขั้นตอนการหาค่าซินโดรม ในกรณีที่ทราบค่าคำรหัสเป็นบางส่วน โดยใช้วิธีลดรูปสมการได้ดังนี้

7.2.1 พิจารณาสัญลักษณ์ของคำรหัสที่ถูกถอดรหัสแล้ว ในที่นี้สมมติว่ามี w สัญลักษณ์ แต่สัญลักษณ์ของคำรหัสจะถูกนำมาถอดรหัสครั้งละ 3 สัญลักษณ์ จึงกำหนดให้ p = w/3 จะได้ว่า ข้อมูลที่รับได้ R มีทั้งหมด 1 ถึง p ตัว ที่ถูกถอดรหัสไปแล้ว ทำให้ข้อมูลที่รับได้ลำดับที่ p + 1 จะเป็นลำดับถัดเป็นที่จะถูกถอดรหัส

7.2.2 การหาค่าซินโดรมอันดับหนึ่ง (First syndrome) S_1 จากข้อมูลที่รับได้ลำดับที่ p + 1 จากสมการต่อไปนี้

$$S_1 = \mathbf{s}_{p+1} = R_{3(p+1)-2} + R_{3(p+1)-1} + R_{3(p+1)} + M_{o1} \tag{33}$$

ซึ่งได้จากการนำสมการที่ (31) มาเขียนใหม่ให้อยู่ในรูปตัวแปร และแทนค่า A ด้วยค่า M_o ในสมการที่ (16)

7.2.3 การหาค่าซินโดรมอันดับสอง (Second syndrome) S_2 จากข้อมูลที่รับได้ลำดับที่ p + 1 และ p+2 จากสมการต่อไปนี้

$$S_2 = \begin{bmatrix} \mathbf{s}_{p+1} \\ \mathbf{s}_{p+2} \end{bmatrix} \tag{34}$$

โดยค่า s_{p+1} ได้จากสมการที่ (33) ส่วนค่าของ $\mathbf{s}_{p+2}$ ได้จากการนำสมการที่ (32) มาเขียนใหม่ให้อยู่ในรูปตัวแปร และแทนค่า A และ B ด้วยค่า M_o ในสมการที่ (16) มีค่าดังนี้

$$\begin{aligned} \mathbf{s}_{p+2} = {} & R_{3(p+1)-1} + R_{3(p+1)} + R_{3(p+2)-2} + R_{3(p+2)-1} \\ & + R_{3(p+2)} + M_{o2} + M_{o3} \end{aligned} \tag{35}$$

7.2.4 การหาค่าซินโดรมอันดับสาม (Third syndrome) S_3 จากข้อมูลที่รับได้ลำดับที่ p + 1 ถึง p+3 จากสมการต่อไปนี้

$$S_3 = \begin{bmatrix} \mathbf{s}_{p+1} \\ \mathbf{s}_{p+2} \\ \mathbf{s}_{p+3} \end{bmatrix} \tag{36}$$

โดยค่า $\mathbf{s}_{p+1}$ และ $\mathbf{s}_{p+2}$ ได้จากสมการที่ (33) และ (35) ส่วนค่าของ $\mathbf{s}_{p+3}$ มาด้วยวิธีการเดียวกับสมการที่ (31) แต่มาเขียนใหม่ให้อยู่ในรูปตัวแปร และแทนค่า A และ B ด้วยค่า M_o ในสมการที่ (16) ซึ่งจะได้ค่าดังนี้

$$\begin{aligned}\mathbf{s}_{p+3} &= R_{3(p+1)-1} + R_{3(p+2)-1} + R_{3(p+2)} + R_{3(p+3)-2} \\ &\quad + R_{3(p+3)-1} + R_{3(p+3)} + M_{o1} + M_{o2} + M_{o4}\end{aligned} \tag{37}$$

7.2.5 การหาค่าซินโดรมอันดับสี่ (Forth syndrome) S_4 จากข้อมูลที่รับได้ลำดับที่ p + 1 ถึง p+4 จากสมการต่อไปนี้

$$S_4 = \begin{bmatrix} \mathbf{s}_{p+1} \\ \mathbf{s}_{p+2} \\ \mathbf{s}_{p+3} \\ \mathbf{s}_{p+4} \end{bmatrix} \tag{38}$$

โดยค่า $\mathbf{s}_{p+1}$, $\mathbf{s}_{p+2}$ และ $\mathbf{s}_{p+3}$ ได้จากสมการที่ (33), (35) และ (37) ส่วนค่าของ $\mathbf{s}_{p+4}$ มาด้วยวิธีการเดียวกับสมการที่ (31) แต่มาเขียนใหม่ให้อยู่ในรูปตัวแปร และแทนค่า A ด้วยค่า M_o ในสมการที่ (16) ซึ่งจะได้ค่าดังนี้

$$\begin{aligned}\mathbf{s}_{p+4} &= R_{3(p+1)-2} + R_{3(p+2)-1} + R_{3(p+3)-1} + R_{3(p+3)} \\ &\quad + R_{3(p+4)-2} + R_{3(p+4)-1} + R_{3(p+4)} + M_{o2}\end{aligned} \tag{39}$$

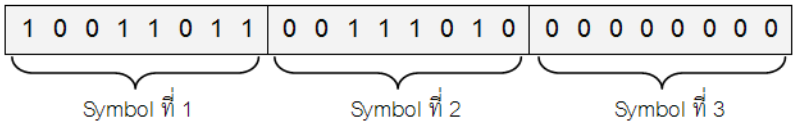
7.2.6 การหาค่าซินโดรมอันดับห้าเป็นต้นไป จะมีหลักการที่คล้ายกันคือ สมาชิกตัวที่ $\mathbf{s}_{p+1}$ ถึง $\mathbf{s}_{p+4}$ จะหาได้จากสมการที่ (33), (35), (37) และ (39) ส่วน $\mathbf{s}_{p+5}$ เป็นต้นไป จะหาได้จากสมการต่อไปนี้

$$\begin{aligned}\mathbf{s}_{p+q} &= R_{3(p+q-4)-2} + R_{3(p+q-4)-1} + R_{3(p+q-4)} + R_{3(p+q-3)-2} \\ &\quad + R_{3(p+q-2)-1} + R_{3(p+q-1)-1} + R_{3(p+q-1)} + R_{3(p+q)-2} \\ &\quad + R_{3(p+q)-1} + R_{3(p+q)}\end{aligned} \tag{40}$$

ซึ่งการหาค่าซินโดรมดังสมการที่ (40) นี้เป็นการหาค่าซินโดรมปกติที่ไม่มีการลดรูปสมการ เนื่องจากค่าซินโดรมอันดับห้าเป็นต้นไปนั้น ไม่ต้องใช้คำรหัสที่ถอดรหัสแล้วมาใช้ในการหาค่า

วิธีการลดรูปสมการดังสมการที่ (31) ถึง (39) นั้น มีประโยชน์ในการลดปริมาณการใช้หน่วยความจำของวงจรถอดรหัส แทนที่จะเลือกเก็บคำรหัสทั้งหมดที่ถอดรหัสไปแล้ว ซึ่งไม่มีประโยชน์ในการถอดรหัสแล้ว การลดรูปสมการนี้จะช่วยในการเลือกเก็บส่วนของคำรหัสที่จำเป็นต้องใช้งาน

ตัวอย่างที่ 1 การหาค่าซินโดรมอันดับหนึ่ง เมื่อกำหนดให้ข้อมูลที่รับได้เป็นข้อมูลชุดแรกที่ถูกถอดรหัส ซึ่งมีค่าดังรูป



ภาพที่ 8 ข้อมูลที่รับได้หนึ่งชุดข้อมูล ประกอบด้วย 3 สัญลักษณ์ สัญลักษณ์ละ 8 บิต

จากรูปเขียนให้อยู่ในรูปเมทริกซ์ ได้เป็น

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

ข้อมูลที่รับได้เป็นข้อมูลชุดแรก คือ $p = 0$ และ $M_{o1} = 0$ จึงสามารถหาค่าซินโดรมอันดับหนึ่งได้จากสมการที่ (33)

$$\begin{aligned} S_1 = \mathbf{s}_{p+1} &= R_{3(p+1)-2} + R_{3(p+1)-1} + R_{3(p+1)} + M_{o1} = R_1 + R_2 + R_3 + \mathbf{0} \\ &= (1\ 0\ 0\ 1\ 1\ 0\ 1\ 1) + (0\ 0\ 1\ 1\ 1\ 0\ 1\ 0) \\ &\quad + (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) + (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\ S_1 &= [\,1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\,] \end{aligned}$$

7.3 วิธีถอดรหัสด้วยซินโดรมเดียว (Decode with one syndrome)

การถอดรหัสวิธีนี้เป็นขั้นตอนหนึ่งของขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง ที่ความซับซ้อนน้อย มีความสามารถในการถอดรหัสโดยใช้ซินโดรมอันดับหนึ่งเพียงตัวเดียวเท่านั้น ซึ่งเป็นกรณีที่มีโอกาสเกิดได้มากที่สุด ซึ่งมากกว่ากรณีที่ใช้ซินโดรมตั้งแต่อันดับสองเป็นต้นไป ดังนั้นจึงแยกวิธีการแก้ไขด้วยตัวเลือกอันดับสองออกเป็นสองขั้นตอน เพื่อให้แต่ละขั้นตอนทำงานได้อย่างมีประสิทธิภาพ สำหรับการถอดรหัสที่ใช้ซินโดรมตั้งแต่อันดับสองเป็นต้นไป จะใช้วิธีถอดรหัสด้วยซินโดรมหลายตัว (Decode with many syndrome)

การที่จะสามารถถอดรหัสด้วยซินโดรมเดียวได้นี้ จะต้องมีสัญลักษณ์ที่ผิดพลาดไม่เกินหนึ่งสัญลักษณ์ต่อซินโดรมนั้น และข้อมูลในตัวเลือกอันดับสองที่สัมพันธ์กับสัญลักษณ์ที่ผิดพลาดจะต้องเป็นข้อมูลที่ถูกต้องด้วย ถ้าสามารถถอดรหัสด้วยวิธีนี้ได้ จะรายงานค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาด (Error index) ไปยังขั้นตอนการแก้ไขให้ข้อมูลถูกต้อง (Made correction) เพื่อแก้ไขสัญลักษณ์ให้ถูกต้อง หากไม่สามารถถอดรหัสด้วยวิธีนี้ได้ จะเพิ่มอันดับซินโดรมหนึ่งขั้นเป็นซินโดรมอันดับสอง แล้วทำการถอดรหัสด้วยขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง แบบถอดรหัสด้วยซินโดรมหลายตัว ในขั้นต่อไป

ขั้นตอนการถอดรหัสด้วยซินโดรมเดียวของรหัสคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่รับได้ความยาว n มีตัวเลือกอันดับหนึ่งเป็น $(Y_1Y_2Y_3)$ และตัวเลือกอันดับสองเป็น $(Z_1Z_2Z_3)$ มีขั้นตอนดังนี้

7.3.1 หาค่าซินโดรมอันดับหนึ่งด้วยสมการที่ (33) จะได้ค่า S_1

7.3.2 หาค่าผลต่างของตัวเลือกอันดับหนึ่งและสอง ได้เป็น

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} - \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = \begin{bmatrix} Y_1 - Z_1 \\ Y_2 - Z_2 \\ Y_3 - Z_3 \end{bmatrix} \tag{41}$$

7.3.3 นำค่าทั้งสองที่ได้ข้างต้นมาเรียงใหม่เป็นเมทริกซ์ให้ $\mathbf{S}_1$ อยู่ด้านบนสุด บรรทัดถัดไปเรียงด้วย D_1 ถึง D_3 ตามลำดับ เรียกเมทริกซ์ใหม่นี้ว่า ซินโดรมดัดแปรอันดับหนึ่ง (First modified syndrome) ดังนี้

$$S_1' = \begin{bmatrix} S_1 \\ D_1 \\ D_2 \\ D_3 \end{bmatrix} \tag{42}$$

7.3.4 นำซินโดรมดัดแปรอันดับหนึ่งมาหาว่า แถวที่สองถึงสี่ มีแถวใดบ้างที่อยู่ในแถวของปริภูมิ (Space) เดียวกันกับแถวที่หนึ่ง ซึ่งในที่สามารถหาได้โดยง่ายด้วยวิธีเปรียบเทียบว่าแถวใดเหมือนกับแถวที่หนึ่งบ้าง ถ้าไม่มีแถวใดเลยเหมือนกับแถวแรก จะไม่สามารถถอดรหัสด้วยซินโดรมเดียวได้ ถ้ามีแถวที่เหมือนกับแถวแรก แสดงว่าพบข้อมูลที่ผิดพลาดในแถวนั้น ให้รายงานค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดออกมา ได้แก่ แถวที่ 2 ของโดรมปรับปรุงอันดับหนึ่งมีค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดเป็น 1 ส่วนแถวที่ 3 ของโดรมปรับปรุงอันดับหนึ่งมีค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดเป็น 2 และแถวที่ 4 ของโดรมปรับปรุงอันดับหนึ่งมีค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดเป็น 3 จากนั้นให้ทำการแก้ไขความผิดพลาด โดยนำแถวที่พบค่าความผิดพลาดไปบวกกับตัวเลือกอันดับหนึ่งที่ตัวเดียวกับค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดที่ได้

<u>ตัวอย่างที่ 2</u> การถอดรหัสด้วยวิธีถอดรหัสด้วยซินโดรมเดียว

สมมุติว่าภาคส่งเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์มีขนาด 8 บิต ทำการส่งบิต 0 มาทั้งหมด กำหนดให้ตัวเลือกอันดับหนึ่งและสองมีค่าเป็นดังเมทริกซ์ข้างล่าง

1st Choice

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

2nd Choice

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

หาค่าซินโดรมอันดับหนึ่งจากสมการที่ (33)

$$\begin{aligned} S_1 = \mathbf{s}_{p+1} &= R_{3(p+1)-2} + R_{3(p+1)-1} + R_{3(p+1)} + M_{o1} \\ &= \mathbf{0} + \mathbf{0} + [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0] + \mathbf{0} \\ S_1 &= [\,0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\,] \end{aligned}$$

หาค่าผลต่างของตัวเลือกอันดับหนึ่งและสองจากสมการที่ (41)

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

เรียงสมาชิกใหม่ตามสมการที่ (42) ได้ซินโดรมดัดแปรอันดับหนึ่ง เป็น

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

แถวที่ 1 ชี้ไปที่แถวแรก แถวที่ 4 ชี้ไปที่แถวสุดท้าย

จะเห็นได้ว่า แถวที่ 4 ของเมทริกซ์ มีค่าเหมือนกับซินโดรม (ตัวที่อยู่เหนือเส้นประ) ดังนั้นจะได้ว่าค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาด คือ ตำแหน่งที่ 4-1 = 3

ทำการแก้ไขความผิดพลาดนี้ได้โดยนำเอาค่าในแถวที่ 4 ของซินโดรมดัดแปรอันดับหนึ่งไปบวกแบบมอดูโล-2 เพิ่มเข้าไปในแถวที่ 3 ของตัวเลือกอันดับหนึ่ง ซึ่งตำแหน่งของแถวก็คือตำแหน่งที่ได้จากค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาด

จะได้ว่า ตัวเลือกอันดับหนึ่งหลังจากมีการแก้ค่าความผิดพลาดแล้วจะมีค่าเป็น

$$1^{st}\ \text{Choice} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

สังเกตได้ว่าข้อมูลของเราถูกแก้ไขจนได้ข้อมูลที่ถูกต้องแล้ว เพราะสมมุติให้ค่าที่ถูกส่งมาเป็นบิต 0 ทั้งหมด

7.4 วิธีถอดรหัสด้วยซินโดรมหลายตัว (Decode with many syndrome)

วิธีนี้จะใช้เมื่อไม่สามารถถอดรหัสด้วยวิธีถอดรหัสด้วยซินโดรมเดียวได้ โดยจะต้องใช้ข้อมูลของข้อมูลตัวเลือกอันดับหนึ่งมากกว่า n สัญลักษณ์ในการถอดรหัส โดยจะแบ่งเป็นสองขั้นตอน คือ

7.4.1 ขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง (Correct with second choice)

การถอดรหัสวิธีนี้จะถูกนำมาใช้ก็ต่อเมื่อการถอดรหัสแบบถอดรหัสด้วยซินโดรมอันดับหนึ่งตัวเดียวไม่สามารถทำการถอดรหัสได้ โดยจะสามารถตรวจสอบ และแก้ไขสัญลักษณ์ที่ผิดพลาดได้บางส่วนหรือทั้งหมด ซึ่งวิธีนี้จะเริ่มต้นทำการถอดรหัสที่ใช้ซินโดรมตั้งแต่อันดับสองเป็นต้นไป อาจสามารถถอดรหัสได้เมื่อมีสัญลักษณ์ที่ผิดพลาดมากกว่าหรือเท่ากับหนึ่งสัญลักษณ์ต่อซินโดรม และมีข้อมูลในตัวเลือกอันดับสองบางส่วนที่เป็นข้อมูลที่ถูกต้อง ขั้นตอนการถอดรหัสนั้นคล้ายกับการถอดรหัสด้วยซินโดรมเดียว แต่มีข้อแตกต่างที่มีขนาดของซินโดรมดัดแปรขนาดใหญ่กว่า ทำให้การหาว่าแถวของผลต่าง แถวใดที่อยู่ในแถวของปริภูมิ (Space) เดียวกันกับแถวของซินโดรมบ้างนั้น จะทำโดยการเปรียบเทียบค่าว่าเหมือนกันหรือไม่ได้ แต่จะสามารถหาข้อมูลในตัวเลือกอันดับสองที่เป็นข้อมูลที่ถูกต้องได้ง่ายด้วยการใช้ Gauss-Jordan reduction หลังจากถอดรหัสด้วยวิธีนี้เสร็จแล้ว ผลลัพธ์จากการถอดรหัสอาจมีได้ดังนี้

ก. ถอดรหัสได้โดยสมบูรณ์ เมื่อค่าซินโดรมที่คำนวณหลังจากการถอดรหัสแล้ว มีค่าเป็นศูนย์ แสดงว่าตัวถอดรหัสไม่พบสัญลักษณ์ที่ผิดพลาดแล้ว จะรายงานค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดทั้งหมดที่พบ ไปยังขั้นตอนการแก้ไขให้ข้อมูลถูกต้อง เพื่อแก้ไขสัญลักษณ์ให้ถูกต้อง

ข. ถอดรหัสได้บางส่วน เมื่อค่าซินโดรมที่คำนวณหลังจากการถอดรหัสแล้ว มีค่าไม่เป็นศูนย์ แต่สามารถหาสัญลักษณ์ที่ผิดพลาดได้ แสดงว่าตัวถอดรหัสสามารถแก้ไขสัญลักษณ์ที่ผิดพลาดได้บางส่วน โดยยังคงมีสัญลักษณ์ที่ผิดพลาดเหลืออยู่ โดยจะรายงานค่าตำแหน่งของสัญลักษณ์ที่ผิดพลาดทั้งหมดที่พบ ไปยังขั้นตอนการแก้ไขให้ข้อมูลถูกต้อง ส่วนสัญลักษณ์ที่ผิดพลาดที่คงเหลืออยู่จะทำการถอดรหัสด้วยวิธีการถอดรหัสโดยใช้ขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ต่อไป

ค. ถอดรหัสไม่ได้ เมื่อค่าซินโดรมที่คำนวณหลังจากการถอดรหัสแล้ว มีค่าไม่เป็นศูนย์ และไม่สามารถหาสัญลักษณ์ที่ผิดพลาดได้ แสดงว่าตัวถอดรหัสไม่สามารถแก้ไขสัญลักษณ์ที่ผิดพลาดได้เลย จะต้องทำการทำการถอดรหัสด้วยขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ต่อไป

7.4.2 ขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination)

สำหรับขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์นี้ เป็นวิธีถอดรหัสที่มีขั้นตอนที่ซับซ้อนและมีประสิทธิภาพสูง สามารถแก้ไขชุดของสัญลักษณ์ที่มีรูปแบบของความผิดพลาดที่ซับซ้อนบางรูปแบบได้ โดยไม่ต้องอาศัยจากข้อมูลของตัวเลือกอันดับสองมาช่วยถอดรหัส เพื่อให้การถอดรหัสด้วยวิธีนี้มีประสิทธิภาพสูงสุด ควรแก้ไขสัญลักษณ์ที่ผิดพลาดให้ได้มากที่สุดก่อน ดังนั้นการรหัสวิธีนี้จึงถูกจัดให้เป็นขั้นตอนถอดรหัสสุดท้าย การถอดรหัสด้วยวิธีนี้ประกอบไปด้วย 3 ขั้นตอน ดังนี้

ก. การหาตัวบ่งชี้ศูนย์ (Null indicator) และผลรวมทางพีชคณิตได้ศูนย์ (Null combination) (Tuntoolavest and Metzner, 2002) โดยนิยามให้ ตัวบ่งชี้ศูนย์ คือ เวกเตอร์ขนาด p บิต โดยแต่ละบิตอาจมีค่าเป็น “1” เมื่อตำแหน่งของบิตนั้น สัมพันธ์กันกับผลรวมทางพีชคณิตได้ศูนย์ และนิยามให้ ผลรวมทางพีชคณิตได้ศูนย์ คือ สมาชิกแถวของปริภูมิ (Space) ของ H ที่สัมพันธ์กับค่าปริภูมิที่เป็นศูนย์ของ E โดยสามารถสรุปเป็นขั้นตอนได้ดังนี้

1) หาเซตของแถวซินโดรมตั้งแต่ 1 ตัวขึ้นไปที่บวกกันแบบมอดูโล-2 แล้วได้เวกเตอร์ศูนย์

<u>ตัวอย่าง</u> สมมติให้ แถวที่ i ของเมทริกซ์ซินโดรม = S_i

$$S_1 + S_3 + S_4 = \bar{0}$$

เซตของแถวซินโดรม ที่บวกแบบมอดูโล-2 แล้วได้ศูนย์ = $\{S_1, S_3, S_4\}$

2) หาตัวบ่งชี้ศูนย์ โดยทำการหาเวกเตอร์ขนาด p บิต ที่ระบุตำแหน่งของแถวซินโดรม ที่บวกกันแบบมอดูโล-2 แล้วได้ศูนย์ โดยบิตที่ i จะเป็น 1 ก็ต่อเมื่อมีแถวที่ i ของซินโดรมอยู่ในเซตของข้อที่หนึ่ง และเป็น 0 เมื่อแถวที่ i ของซินโดรมไม่อยู่ในเซตของข้อที่หนึ่ง

<u>ตัวอย่าง</u> สมมติให้ NI_j เป็นตัวบ่งชี้ศูนย์ตัวที่ j ขนาด 7 บิต
จากตัวอย่างข้างต้น จะได้

$$NI_1 = [1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$$

3) หาผลรวมทางพีชคณิตได้ศูนย์จากการบวกกันแบบมอดูโล-2 ระหว่างแถวของเมทริกซ์ตรวจสอบภาวะคู่หรือคี่ H ทั้งหมดที่สัมพันธ์กับตัวบ่งชี้ศูนย์

<u>ตัวอย่าง</u> สมมติให้ NC_j เป็นผลรวมทางพีชคณิตได้ศูนย์ตัวที่ j

$$NI_1 = [1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$$

จะได้ $NC_1 = \text{row}(H_1) + \text{row}(H_3) + \text{row}(H_4)$

4) หาตัวบ่งชี้ศูนย์ และผลรวมทางพีชคณิตได้ศูนย์ทั้งหมดทุกตัว โดยใช้วิธีเดียวกันกับสามขั้นตอนแรก

ข. การหาเวกเตอร์ระบุตำแหน่งความผิดพลาด (Error locating vector หรือ ELV) สามารถสรุปเป็นขั้นตอนได้ดังนี้

1) การหาเวกเตอร์ระบุตำแหน่งความผิดพลาดสามารถทำได้โดยการ “OR” ผลรวมทางพีชคณิตได้ศูนย์ทุกตัวเข้าด้วยกัน และด้วยเวกเตอร์ระบุตำแหน่งความผิดพลาดนี้ ทำให้ทราบว่า สัญลักษณ์ที่รับได้ที่ฝั่งรับตัวใดบ้าง ที่เป็นสัญลักษณ์ที่เกิดความผิดพลาด โดยตำแหน่งของเวกเตอร์ระบุตำแหน่งความผิดพลาดที่เป็นศูนย์ เป็นตำแหน่งเดียวกับสัญลักษณ์ที่ผิดพลาดของสัญลักษณ์ที่รับได้ที่ฝั่งรับ

<u>ตัวอย่าง</u> สมมติให้ ELV เป็นเวกเตอร์ระบุตำแหน่งความผิดพลาดของเมทริกซ์ของซินโดรม S โดยที่ NC_1 และNC_2 เป็นสมาชิกในเซตของผลรวมทางพีชคณิตได้ศูนย์ทั้งหมดที่สามารถหามาได้จากเมทริกซ์ของซินโดรม S และให้

$$NC_1 = [100 \quad 111 \quad 000 \quad 000]$$
$$NC_2 = [011 \quad 010 \quad 011 \quad 111]$$

จะได้ $ELV = "NC_1" \text{ OR } "NC_2" = [111 \quad 111 \quad 011 \quad 111]$

2) นับจำนวนสัญลักษณ์ที่ผิดพลาดที่ถูกตรวจพบ ได้จากจำนวนสมาชิกที่เป็นศูนย์ของเวกเตอร์ระบุตำแหน่งความผิดพลาด

3) ตรวจสอบว่าจำนวนสัญลักษณ์ที่ผิดพลาด กับ จำนวนแถวของซินโดรมที่เป็นอิสระเชิงเส้นต่อกันนั้นเท่ากันหรือไม่ ซึ่งจะแบ่งผลการตรวจสอบออกได้เป็น

ก) จำนวนสัญลักษณ์ที่ผิดพลาด เท่ากับ จำนวนแถวของซินโดรมที่เป็นอิสระเชิงเส้นต่อกัน จะสามารถหาค่าความผิดพลาด (Error value) ได้

ข) จำนวนสัญลักษณ์ที่ผิดพลาด มากกว่า จำนวนแถวของซินโดรมที่เป็นอิสระเชิงเส้นต่อกัน จะยังไม่สามารถหาค่าความผิดพลาดได้ ต้องทำการเพิ่มจำนวนของซินโดรมแล้วจึงทำการถอดรหัสอีกครั้ง

ค) จำนวนสัญลักษณ์ที่ผิดพลาด น้อยกว่า จำนวนแถวของซินโดรมที่เป็นอิสระต่อกัน จะไม่สามารถหาค่าความผิดพลาดได้ และการถอดรหัสล้มเหลว (Decode failure) ข้อมูลสัญลักษณ์ชุดนี้ได้ เนื่องจากมีความไม่อิสระเชิงเส้นต่อกันของสัญลักษณ์ที่ผิดพลาด (Linearly dependent error symbol)

ค. การหาค่าความผิดพลาดสามารถหาได้จากสมการ

$$E = H^{-1} \cdot S \tag{43}$$

แต่ในทางปฏิบัติแล้ว มีวิธีที่สามารถหาค่าความผิดพลาดได้ง่ายกว่า โดยทำการลดขนาดของ H และ S ให้เล็กลง ดึงเฉพาะส่วนที่ต้องใช้ในการหาค่าความผิดพลาดเท่านั้น ซึ่งสรุปเป็นขั้นตอนได้ดังนี้

1) หาเมทริกซ์ตรวจสอบภาวะคู่หรือคี่ที่สัมพันธ์กับสัญลักษณ์ที่ผิดพลาดที่เกิด H_{sub} ได้จาก

ก) เลือกแถวของเมทริกซ์ตรวจสอบภาวะคู่หรือคี่จากแถวของซินโดรมที่อิสระเชิงเส้นต่อกัน

ข) เลือกหลักของเมทริกซ์ตรวจสอบภาวะคู่หรือคี่จากตำแหน่งเวกเตอร์ระบุตำแหน่งความผิดพลาดที่เป็นศูนย์

2) หาซินโดรมที่สัมพันธ์กับสัญลักษณ์ที่ผิดพลาดที่เกิด S_{sub} โดยดึงมาเฉพาะแถวของซินโดรมที่อิสระเชิงเส้นต่อกัน

3) หาสัญลักษณ์ที่ผิดพลาดจากการคูณ H_{sub}^{-1} ด้วยซินโดรมที่อิสระต่อกัน ดังสมการ (Usana, 2002)

$$E_{sub} = H_{sub}^{-1} \cdot S_{sub} \tag{44}$$

4) นำสัญลักษณ์ที่ผิดพลาด ที่หาได้ไปใช้แก้ไข โดยเทียบจากตำแหน่งของเวกเตอร์ระบุตำแหน่งความผิดพลาดที่เป็นศูนย์

<u>ตัวอย่างที่ 3</u> ขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination)

สมมุติว่าภาคส่งเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์มีขนาด 8 บิต ทำการส่งบิต 0 มาทั้งหมด กำหนดให้ตัวเลือกอันดับสองไม่มีค่า ส่วนตัวเลือกอันดับหนึ่งมีค่าเป็นดังเมทริกซ์ต่อไปนี้

$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & \vdots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{18\times 1}$$

เนื่องจากตัวเลือกอันดับหนึ่งมีทั้งหมด 18 สัญลักษณ์ หรือคิดเป็น 6 คำรหัส จากสมการที่ (25) ทำให้ได้ เมทริกซ์ตรวจสอบภาวะคู่หรือคี่เป็นดังนี้

$$H = \begin{bmatrix} 111 & 000 & 000 & 000 & 000 & 000 \\ 011 & 111 & 000 & 000 & 000 & 000 \\ 010 & 011 & 111 & 000 & 000 & 000 \\ 100 & 010 & 011 & 111 & 000 & 000 \\ 111 & 100 & 010 & 011 & 111 & 000 \\ 000 & 111 & 100 & 010 & 011 & 111 \end{bmatrix}$$

จากสมการที่ (40) สามารถหาซินโดรมอันดับหกได้เป็น

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

จะพบว่า ซินโดรมแถวที่ 1 และ 3 นั้น เป็นอิสระเชิงเส้นกัน จะได้

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

หาตัวบ่งชี้ศูนย์ (Null indicator) จากซินโดรมได้ผลดังนี้

$NI_1 = \{S_1, S_2\}$

Null $\text{indicator}_2 = \{S_4\}$

Null $\text{indicator}_3 = \{S_1, S_5\}$

Null $\text{indicator}_4 = \{S_6\}$

หาผลรวมทางพีชคณิตได้ศูนย์ (Null combination) จากตัวบ่งชี้ศูนย์ได้เป็น

$NC_1 = \text{row}(H_1) + \text{row}(H_2) = [100\ 111\ 000\ 000\ 000\ 000]$

$NC_2 = \text{row}(H_4) = [100\ 010\ 011\ 111\ 000\ 000]$

$NC_3 = \text{row}(H_1) + \text{row}(H_5) = [000\ 100\ 010\ 011\ 111\ 000]$

$NC_4 = \text{row}(H_6) = [000\ 111\ 100\ 010\ 011\ 111]$

ELV = "NC_1" OR "NC_2" OR "NC_3" OR "NC_4" = [1**00** 111 111 111 111 111]

จากเวกเตอร์ระบุตำแหน่งความผิดพลาด จะได้ว่ามีสัญลักษณ์ที่ผิดพลาด 2 สัญลักษณ์ และเกิดกับสัญลักษณ์ตัวที่ 2 และตัวที่ 3 ของสัญลักษณ์ในตัวเลือกอันดับหนึ่ง

หา H_{sub} ได้จากการเลือกแถวที่ 1 กับ 3 ของ H และเลือกหลักที่ 2 และ 3 ของ H

$$H = \begin{bmatrix} 111 & 000 & 000 & 000 & 000 & 000 \\ 011 & 111 & 000 & 000 & 000 & 000 \\ 010 & 011 & 111 & 000 & 000 & 000 \\ 100 & 010 & 011 & 111 & 000 & 000 \\ 111 & 100 & 010 & 011 & 111 & 000 \\ 000 & 111 & 100 & 010 & 011 & 111 \end{bmatrix}$$

จึงได้ H_{sub} เป็น

$$H_{sub} = \begin{bmatrix} 11 \\ 10 \end{bmatrix}$$

$$H_{sub}^{-1} = \begin{bmatrix} 01 \\ 11 \end{bmatrix}$$

หาสัญลักษณ์ที่ผิดพลาดทั้งสองได้จากสมการที่ (44)

$$E_{sub} = H_{sub}^{-1} \cdot S_{sub}$$

$$\begin{bmatrix} \mathbf{e}_2 \\ \mathbf{e}_3 \end{bmatrix} = \begin{bmatrix} 01 \\ 11 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{e}_2 \\ \mathbf{e}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

แก้สัญลักษณ์ที่ผิดพลาดนี้โดยนำ $\mathbf{e}_2$ ไปบวกเข้ากับ Y แถวที่สอง และนำ $\mathbf{e}_3$ ไปบวกเข้ากับ Y แถวที่สาม จะได้ค่าที่ถูกต้องตัวเลือกอันดับหนึ่งเป็น

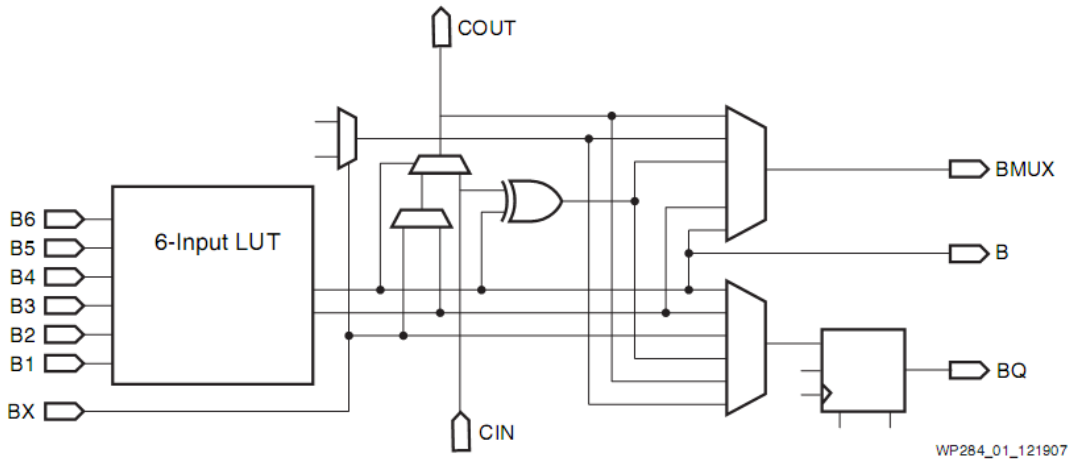
$$Y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{18\times 1}$$

จะเห็นได้ว่าค่าของตัวเลือกอันดับหนึ่งได้ทำการแก้ไขข้อมูลได้ถูกต้องแล้ว เพราะว่าค่าได้กำหนดค่าที่ส่งมาให้เป็นบิต 0 ทั้งหมด

ข้อสังเกต : ขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์นี้จะสามารถใช้ได้เมื่อเมทริกซ์ซินโดรม มีจำนวนซินโดรมตั้งแต่ 2 ตัว ขึ้นไป หรือต้องใช้ซินโดรมอันดับสองเป็นต้นไปนั่นเอง

8. การออกแบบวงจรรวมขนาดใหญ่

Field-programmable gate array (FPGA) เป็นอุปกรณ์ประเภทสารกึ่งตัวนำที่สามารถทำการโปรแกรมได้ มีความจุอยู่ที่หนึ่งหมื่นถึงหลายล้านโลจิกเกท โครงสร้างภายในประกอบด้วย Configurable logic blocks (CLBs) ต่อประสานภายในกันเป็นเมทริกซ์ โดยแต่ละ CLBs ภายในจะประกอบด้วย Lookup table (LUT) ต่ออยู่กับ Flip-flops ซึ่งจำนวนของ LUT และ Flip-flops นั้นขึ้นอยู่กับผู้ผลิต FPGA ว่าจะออกแบบมาให้ FGPA แต่ละตัวซับซ้อนเพียงใด



ภาพที่ 9 โครงสร้างภายในบางส่วนของ CLBs ที่มี 6-Input LUT

ที่มา : Future Technology Devices International Ltd. (2002)

การออกแบบวงจรรวมโดยใช้ FPGA นั้น สามารถทำหลายวิธี ตัวอย่างเช่น โปรแกรมโดยใช้ภาษาบรรยายฮาร์ดแวร์ (Hardware description language หรือ HDL) จะเป็นการเขียนโปรแกรมด้วยภาษาระดับล่าง ที่นิยมใช้มีสองภาษาคือ Verilog และ VHDL การออกแบบโดยใช้แผนผัง (Schematic design) สามารถกำหนดการทำงานของ FPGA โดยใช้วิธีการวาดแผนผังวงจรไฟฟ้า ซึ่งวิธีนี้จะเหมาะกับการออกแบบวงจรขนาดเล็กมากกว่า

ภาษา VHDL (Very-high-speed integrated circuits Hardware Description Language) ถูกกำหนดให้เป็นมาตรฐาน IEEE 1076 โดยองค์การวิศวกรรมไฟฟ้าและอิเลคโทรนิค ว่าด้วยมาตรฐานในการเขียนภาษา VHDL นอกจากนี้เครื่องมือที่ใช้ในการเขียนโปรแกรมด้วยภาษา VHDL นั้น มีผู้ผลิตบางรายอนุญาตให้ใช้ได้โดยไม่เสียค่าใช้จ่ายด้านลิขสิทธิ์

ก่อนที่จะทำการเขียนภาษา VHDL ได้นั้น ในขั้นแรกต้องทำการออกแบบ (Finite state machine หรือ FSM) ให้กับวงจรรวมที่จะออกแบบเสียก่อน ซึ่งจะใช้ FSM แบบใดก็ได้ระหว่างแบบ Mealy Machine และแบบ Moore Machine หรือจะผสมทั้งสองแบบก็ได้ ขั้นตอนที่สองจึงทำการเขียนโปรแกรมด้วยภาษา VHDL พร้อมทั้งกำหนดขาของ FPGA แต่ละขาว่าจะให้ต่อกับอุปกรณ์ใด และแต่ละขานั้นจะมีคุณสมบัติเช่นใด ขั้นตอนที่สามเป็นการนำโปรแกรมต้นฉบับที่เขียนเสร็จแล้ว นำมาทำการทดสอบการทำงานด้วย การจำลองการทำงาน (Simulation) ซึ่งจะเป็น

การใช้โปรแกรมมาจำลองการทำงานบนแผนภาพเวลากับรูปคลื่นสัญญาณ ขั้นตอนที่สี่นำโปรแกรมต้นฉบับที่ทดสอบการจำลองการทำงานเสร็จแล้ว มาทำการสังเคราะห์วงจร (Synthesize) ซึ่งจะเป็นการตรวจสอบความถูกต้องของภาษาที่เขียน แล้วแปลงเป็นภาษาเครื่องที่เรียกกันว่า Netlist เมื่อเสร็จขั้นตอนนี้จะสามารถดูได้ว่า โปรแกรมที่เขียนจะใช้ LUT และ Flip-flops ประมาณเท่าใด ขั้นตอนที่ห้าเป็นการนำแฟ้ม Netlist ที่ได้จากขั้นตอนที่สี่ มาทำการ Fit หรืออาจเรียกเป็น Implementation ซึ่งจะเป็นการกำหนดการทำงานของ CLBs แต่ละตัว รวมถึงขาสัญญาณภายในและภายนอก FPGA ขั้นตอนที่หกเป็นขั้นตอนการสร้างแฟ้มไว้สำหรับบันทึกลง FPGA ขั้นตอนที่เจ็ดนำแฟ้มที่ได้จากขั้นตอนที่หกมาทำการดาวน์โหลดโปรแกรมลง FPGA แล้วทดสอบการทำงานบนบอร์ดทดลอง เป็นอันเสร็จสิ้นกระบวนการออกแบบวงจรรวมโดยใช้ FPGA

อุปกรณ์และวิธีการ

อุปกรณ์

1. เครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer)
2. บอร์ดทดลอง FPGA Xilinx ตระกูล Virtex5 ชิปรุ่น XC5VLX110 -1 FF676 C
3. เครื่องดาวน์โหลดโปรแกรม Xilinx Platform Cable USB
4. USB-module รุ่น Ezy USB-M01
5. หัวต่อเชื่อม SAMTEC รุ่น QSE-060-01-F-D-A
6. เครื่องมือวัดดิจิตัลมัลติมิเตอร์ (Digital multi-meter)
7. สายแพ (Ribbon Cable)

วิธีการ

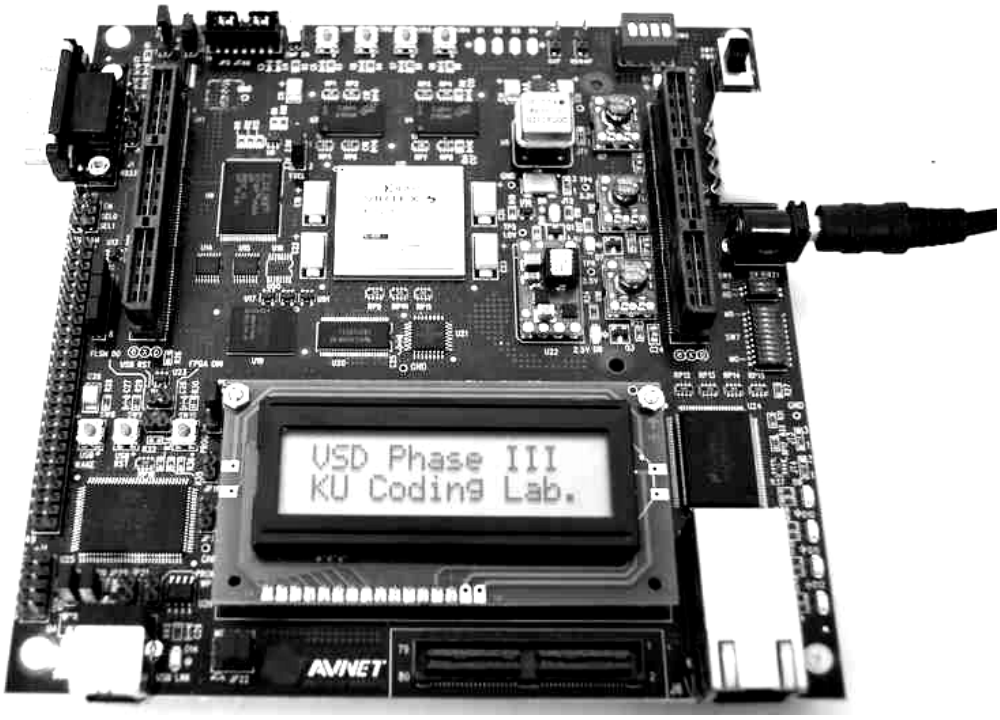
1. การเตรียมอุปกรณ์

1.1 รายละเอียดอุปกรณ์

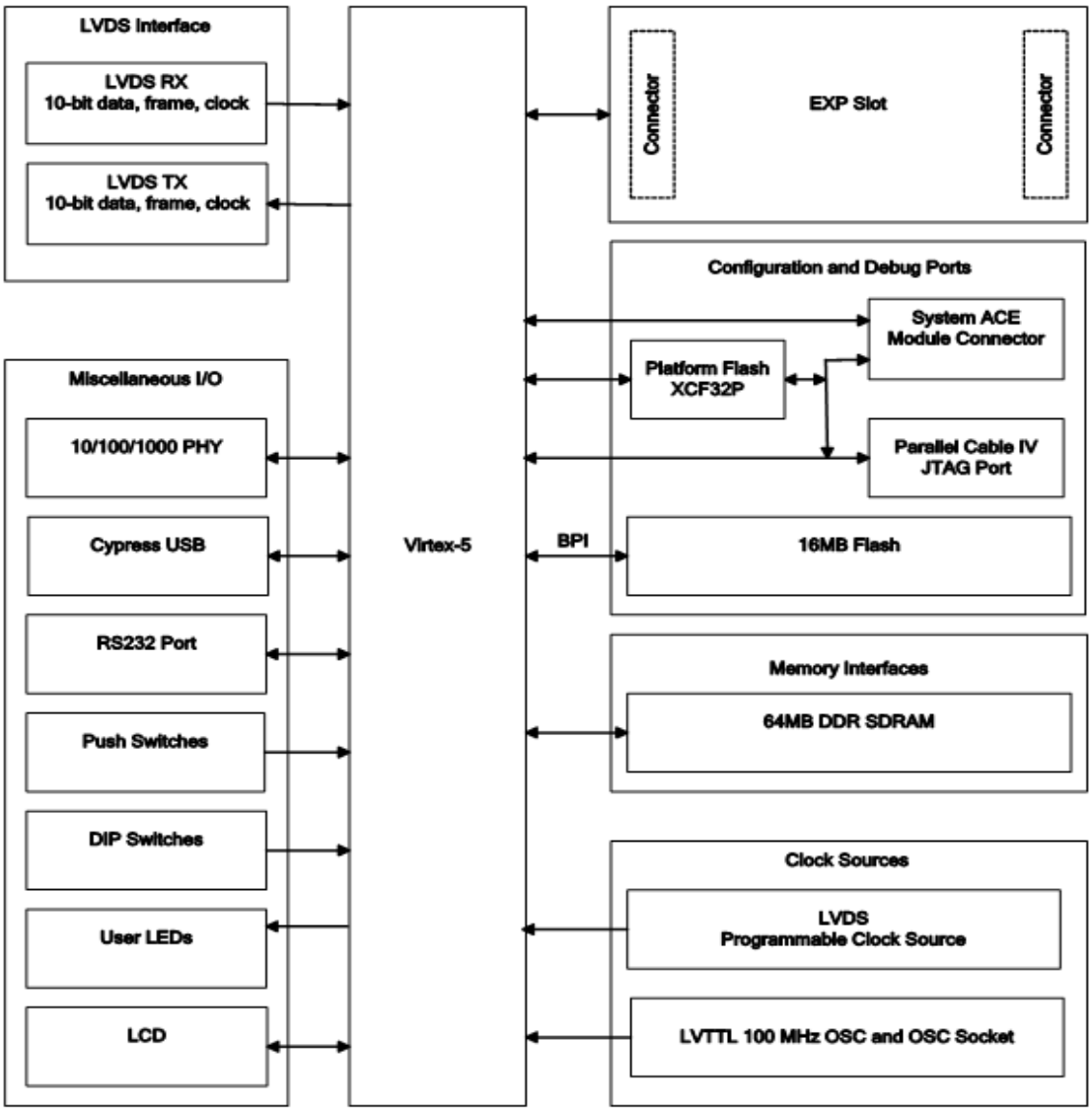
1.1.1 บอร์ดทดลอง FPGA Xilinx ตระกูล Virtex5 ชิปรุ่น XC5VLX110 -1 FF676 C

บอร์ดที่ใช้ยี่ห้อ Xilinx ตระกูล Virtex5 ดังรูปที่ 10 โดยบอร์ดนี้มี FPGAชิปรุ่น XC5VLX110 -1 FF676 C ที่มีขนาดของ Configurable Logic Blocks (CLBs) เท่ากับ 160 x 54 = 8640 เซลล์ หนึ่งเซลล์ประกอบด้วย Flip-flop 8 ตัว และ 6-input LUT (Look up table) 8 ตัว ดังนั้น ชิปตัวนี้จึงประกอบไปด้วย Flip-flops 69120 ตัว และ 6-input LUT 69120 ชุด หรือคิดเป็น 69120 x 2^6 = 4423680 บิต ความถี่ของสัญญาณนาฬิกาที่ใช้ในการประมวลผลประมาณ 550 MHz มีขาสัญญาณไว้ต่อกับอุปกรณ์ภายนอก 440 ขา ซึ่งได้ต่อกับอุปกรณ์ต่างๆ ดังรูปที่ 11 เช่น ช่องต่อเชื่อมสัญญาณ EXP 2 ช่อง ช่องต่อเชื่อมสัญญาณมาตรฐานแบบ RS-232 1 ช่อง ช่องต่อเชื่อม

สัญญาณมาตรฐานแบบ USB 2.0 1 ช่อง ช่องต่อเชื่อมสัญญาณมาตรฐานแบบ Ethernet 10/100/1000 1 ช่อง ช่องต่อสำหรับการดาวน์โหลดโปรแกรม 1 ช่อง จอแสดงผล LCD แบบขาวดำ 1 จอ ปุ่มกดและปุ่มโยก 8 ปุ่ม ไฟแสดงผล LED 4 ดวง สัญญาณนาฬิกา 10 ตัว เป็นต้น นอกจากนี้ยังมีหน่วยความจำให้ใช้ได้อีก 3 ชุด คือ DDR2 SDRAM 64 ล้านไบท์ Flash memory 16 ล้านไบท์ และ EEPROM 4 ล้านไบท์ สำหรับดาวน์โหลดโปรแกรมเก็บไว้ ใช้ในกรณีที่ต้องปิด Power



ภาพที่ 10 บอร์ด FPGA Xilinx ตระกูล Virtex5 รุ่น XC5VLX110 -1 FF676 C



ภาพที่ 11 แผนภาพแสดงส่วนประกอบของบอร์ด FPGA Virtex5 รุ่น XC5VLX110

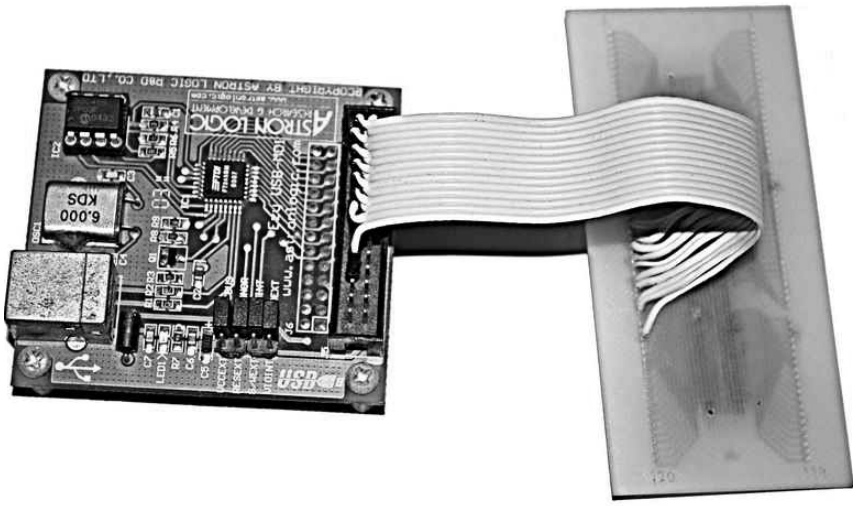
1.1.2 เครื่องดาวน์โหลดโปรแกรม Platform Cable USB

งานวิจัยนี้ ใช้เครื่องดาวน์โหลดโปรแกรมของ Xilinx Platform cable USB ดังรูปที่ 12 ซึ่งสามารถใช้ได้กับการดาวน์โหลดลงชิป FPGA หรือ EEPROM ที่อยู่บนบอร์ด FPGA Virtex5 XC5VLX110 โดยมีช่องต่อเชื่อมสองช่อง คือ ช่องต่อ USB ใช้ต่อกับเครื่องคอมพิวเตอร์ อีกช่องหนึ่งใช้ต่อกับบอร์ด FPGA เป็นสายแพแบบ 14 เส้น การเชื่อมต่อประสานทั้งสองช่องนี้ ได้เป็นไปตามมาตรฐานของ Xilinx จึงไม่ขออธิบายรายละเอียดทางไฟฟ้าไว้ในวิทยานิพนธ์ฉบับนี้

ภาพที่ 12 เครื่องดาวน์โหลดโปรแกรม Xilinx Platform Cable USB

1.1.3 USB-module

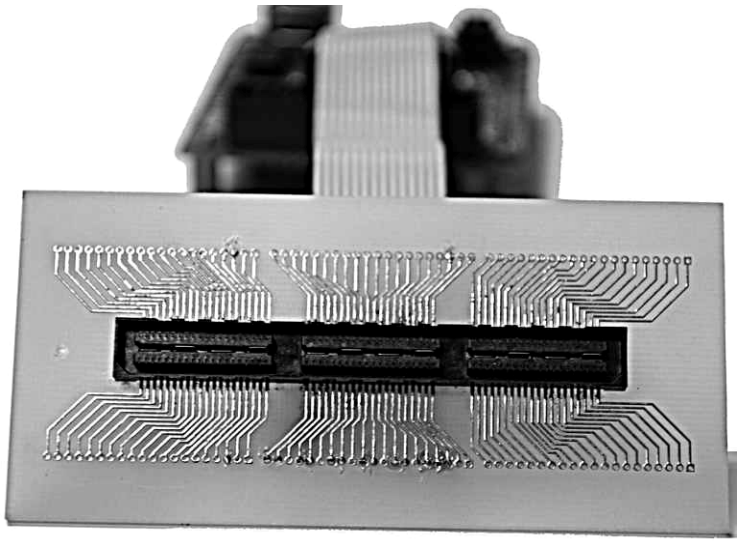
สำหรับการเชื่อมต่อตัวถอดรหัสหรือบอร์ด FPGA เข้ากับคอมพิวเตอร์นั้น จะใช้ USB-module รุ่น Ezy USB-M01 ของบริษัท แอสทรอนลอจิก จำกัด ดังรูปที่ 13 โดย USB-module จะทำหน้าที่เป็นตัวรับข้อมูลจากคอมพิวเตอร์เพื่อส่งข้อมูลไปให้ตัวถอดรหัส และเป็นตัวส่งข้อมูลที่ได้จากการประมวลผลของตัวถอดรหัสกลับไปให้คอมพิวเตอร์ สามารถส่งถ่ายข้อมูลด้วยอัตรา 1 ล้านไบท์ต่อวินาทีหรือ 8 ล้านบิตต่อวินาที และยังรับส่งข้อมูลในลักษณะบัส (Bus) คือสามารถรับส่งข้อมูลได้ครั้งละ 8 บิตในเวลาเดียวกันได้



ภาพที่ 13 USB-module รุ่น Ezy USB-M01 (ด้านซ้าย) เชื่อมต่อประสานกับ PCB ของหัวเชื่อมต่อ QSE (ด้านขวา)

1.1.4 หัวเชื่อมต่อ QSE

สำหรับเชื่อมต่ออุปกรณ์ภายนอกของบอร์ด FPGA นั้นจะเชื่อมต่อผ่านช่อง EXP JX1 ที่มีหัวเชื่อมต่อเป็นหัวแบบ QTE ของบริษัท SAMTEC จำกัด ซึ่งหัวเชื่อมต่อนี้เป็นตัวผู้ โดยต้องใช้หัวแบบ QSE ที่เป็นตัวเมียในการเชื่อมต่อ แต่หัวเชื่อมต่อ QSE นี้ เป็นอุปกรณ์ชนิด Surface Mount คือ ต้องเชื่อมต่อด้านหนึ่งของหัวเชื่อมต่อ QSE ลงบนแผ่น PCB ก่อน ถึงจะนำไปเชื่อมต่อสายสัญญาณ เมื่อเชื่อมต่อสายสัญญาณกับ PCB และ PCB กับหัว QSE แล้ว จะมีหน้าตาเป็นดังรูปที่ 14 โดยอีกด้านหนึ่งของขาสัญญาณจะเชื่อมต่อกับ USB-module

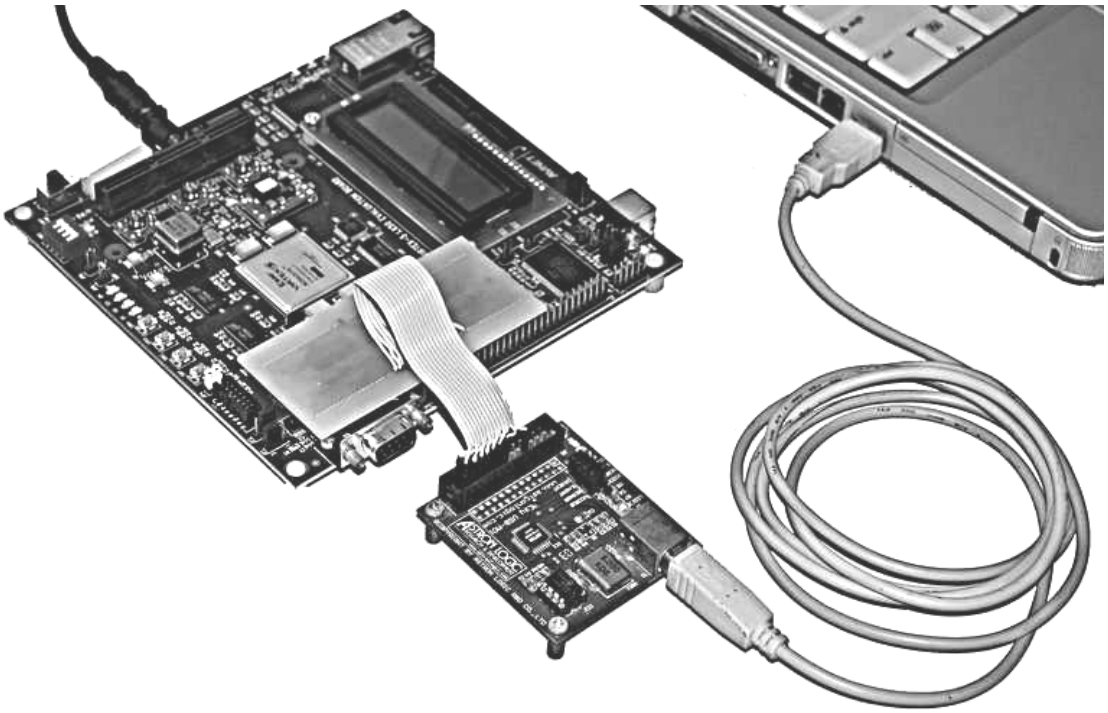


ภาพที่ 14 PCB และหัวเชื่อมต่อ QSE

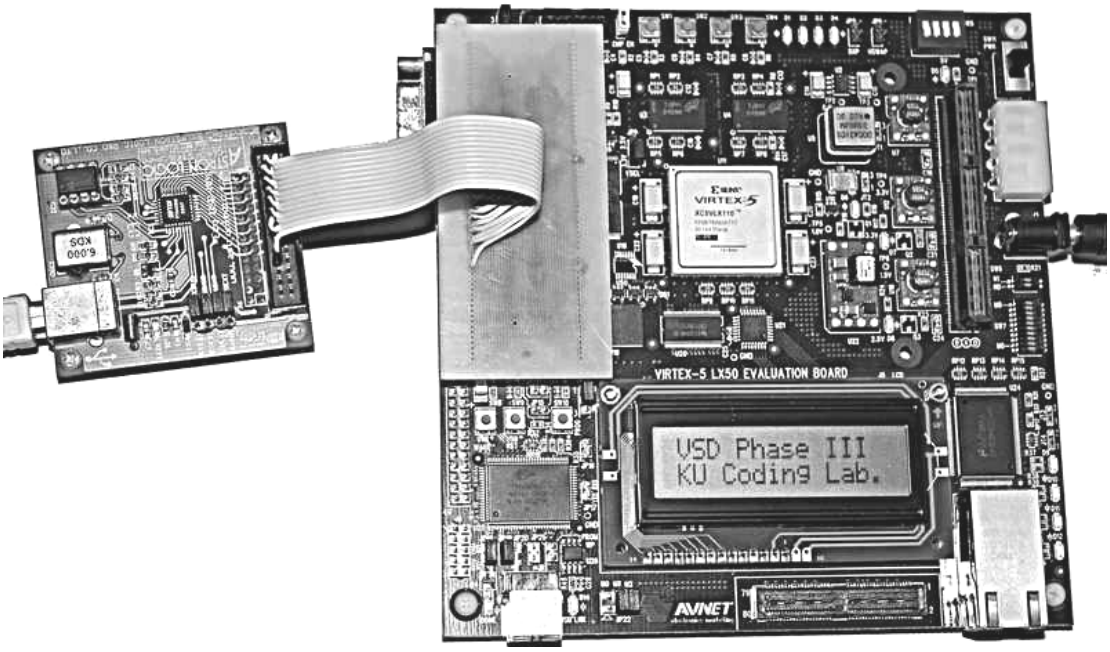
1.2 การเชื่อมต่อประสาน (Interface)

การเชื่อมต่อประสานอุปกรณ์ ในงานวิจัยนี้ได้แบ่งออกเป็น 2 กรณี ตามลักษณะการใช้งาน คือ การเชื่อมต่อประสานอุปกรณ์สำหรับดาวน์โหลดโปรแกรม และการเชื่อมต่อประสานอุปกรณ์สำหรับรับส่งข้อมูล ซึ่งทั้ง 2 กรณีเป็นการเชื่อมต่อประสาน ระหว่างคอมพิวเตอร์กับบอร์ด FPGA โดยต่างที่อุปกรณ์ที่เป็นสื่อกลางในเชื่อมต่อ โดยในกรณีใช้สำหรับดาวน์โหลดโปรแกรมจะใช้เครื่องดาวน์โหลดโปรแกรม Platform Cable USB คั่นระหว่างบอร์ด FPGA กับคอมพิวเตอร์ ดังรูปที่ 15 ส่วนกรณีใช้รับส่งข้อมูลจะใช้ USB-module ที่ต่อกับ PCB และหัว QSE ดังรูปที่ 13 มาต่อเข้ากับคอมพิวเตอร์และบอร์ด FPGA ดังรูปที่ 16 และ 17

ภาพที่ 15 การเชื่อมต่อประสานอุปกรณ์ สำหรับดาวน์โหลดโปรแกรม



ภาพที่ 16 การเชื่อมต่อประสานอุปกรณ์ สำหรับรับส่งข้อมูล



ภาพที่ 17 การเชื่อมต่อประสานอุปกรณ์ สำหรับรับส่งข้อมูล แสดงเฉพาะ USB-module กับ FGPA

สำหรับขาสัญญาณในการเชื่อมต่อ ในกรณีใช้สำหรับดาวน์โหลดโปรแกรม จะไม่กล่าวถึงเนื่องจากเป็นอุปกรณ์มาตรฐานของบริษัท Xilinx จำกัด งานวิจัยนี้ไม่ได้มีการปรับแต่งหรือเพิ่มเติมใดๆ โดยสามารถดูรายละเอียดได้จาก (Avnet electronics marketing, 2008) ส่วนขาสัญญาณในกรณีที่ใช้รับส่งข้อมูลสามารถดูการเทียบขาสัญญาณได้จากตารางที่ 1 โดยในหลักที่แรกเป็นขาสัญญาณบนบอร์ด FPGA หลักที่สองเป็นขาสัญญาณบน USB-module และหลักสุดท้ายเป็นขาสัญญาณของชิป FPGA ซึ่งจะเป็นขาสัญญาณอ้างอิงที่จะต้องนำไปใช้ในการเขียนโปรแกรม แล้วกำหนดลงบน User constraint file

ตารางที่ 1 การเทียบตำแหน่งขาสัญญาณบอร์ด FPGA กับ USB-module และขาของชิป FPGA

ขาสัญญาณบอร์ด FPGA	**ขาสัญญาณ USB-module**	**ขาของชิป FPGA**
User clock input (CLK)	-	E16
Push switch SW4 (RST)	-	F17
LED0 (ALM)	-	E11
ขาที่ 86 ของ EXP JX1 (3.3V)	ขาที่ 8 ของ USB-module (3V3_VCCIO)	-

ตารางที่ 1 (ต่อ)

ขาสัญญาณบอร์ด FPGA	ขาสัญญาณ USB-module	ขาของชิป FPGA
ขาที่ 80 ของ EXP JX1	ขาที่ 14 ของ USB-module (RXF#)	H1
ขาที่ 78 ของ EXP JX1	ขาที่ 15 ของ USB-module (TXE#)	G1
ขาที่ 74 ของ EXP JX1	ขาที่ 16 ของ USB-module (WR)	W1
ขาที่ 72 ของ EXP JX1	ขาที่ 17 ของ USB-module (RD#)	Y1
ขาที่ 68 ของ EXP JX1	ขาที่ 18 ของ USB-module (D7)	E3
ขาที่ 66 ของ EXP JX1	ขาที่ 19 ของ USB-module (D6)	F3
ขาที่ 62 ของ EXP JX1	ขาที่ 20 ของ USB-module (D5)	F4
ขาที่ 60 ของ EXP JX1	ขาที่ 21 ของ USB-module (D4)	F5
ขาที่ 56 ของ EXP JX1	ขาที่ 22 ของ USB-module (D3)	E1
ขาที่ 54 ของ EXP JX1	ขาที่ 23 ของ USB-module (D2)	E2
ขาที่ 50 ของ EXP JX1	ขาที่ 24 ของ USB-module (D1)	R2
ขาที่ 48 ของ EXP JX1	ขาที่ 25 ของ USB-module (D0)	R3
ขาที่ 46 ของ EXP JX1 (GND)	ขาที่ 26 ของ USB-module (GND)	-

2. การรับส่งข้อมูล

งานวิจัยนี้ ได้ออกแบบระบบที่ใช้ในการทดสอบการถอดรหัสบนบอร์ด FPGA โดยใช้คอมพิวเตอร์เป็นตัวส่งข้อมูลที่จะนำไปถอดรหัส เข้าไปถอดรหัสยังบอร์ด FPGA และเมื่อถอดรหัสสำเร็จแล้ว ตัวถอดรหัสจะส่งข้อมูลที่ถอดรหัสแล้ว จากบอร์ด FPGA กลับไปยังเครื่องคอมพิวเตอร์ ซึ่งการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับบอร์ด FPGA ได้ใช้ USB-module เป็นตัวเชื่อม จึงทำให้การรับส่งข้อมูลเป็นแบบอนุกรมมีขนาดบัส (Bus) ขนาด 8 บิต ถ้าหากกำหนดให้ 1 สัญลักษณ์ มีขนาด 32 บิต ดังนั้นต้องทำการรับหรือส่งเป็นจำนวน 4 คาบ จึงจะรับหรือส่งข้อมูลครบ 1 สัญลักษณ์ แต่ในการถอดรหัสต้องมีการรับส่งข้อมูลมากกว่า 1 สัญลักษณ์จึงต้องมีการออกแบบการรับส่งข้อมูลให้ถูกต้อง ทั้งนี้การนำ USB-module มาใช้งานทำให้ประหยัดเวลาการวิจัย ไม่ต้องทำการโปรแกรม เพื่อใช้เชื่อมต่อระหว่างระบบปฏิบัติการบนคอมพิวเตอร์ กับฮาร์ดแวร์ที่นำมาต่อเชื่อมกับคอมพิวเตอร์ เนื่องจาก USB-module ที่ใช้มีชุดเครื่องมือสำเร็จรูปใช้ให้งาน

2.1 รูปแบบการจัดเรียงข้อมูล และการสื่อสารกันระหว่างอุปกรณ์

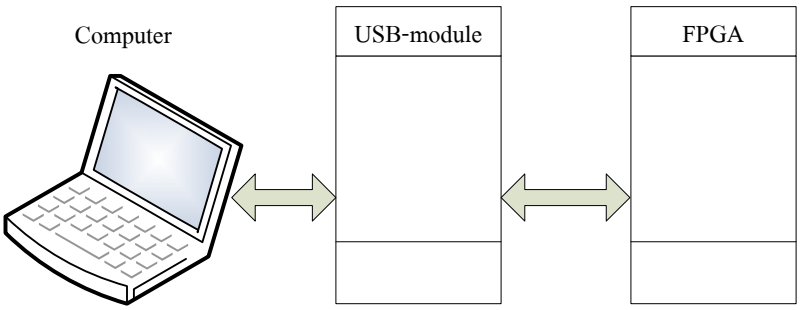
ด้วยระบบที่ได้ออกแบบไว้นั้น การถอดรหัสของคำรหัสหนึ่งคำของงานวิจัยนี้ จะต้องนำข้อมูลตัวเลือกอันดับหนึ่งและสองที่สร้างเป็นไฟล์ไว้ในคอมพิวเตอร์ ส่งไปถอดรหัสยังบอร์ด FPGA ซึ่งข้อมูลทั้งสองตัวเลือกมีขนาดตัวเลือกละ 36 สัญลักษณ์ รวม 72 สัญลักษณ์ แต่ในการถอดรหัสคอนโวลูชัน (3, 2, 2) ในแต่ละรอบนั้น ตัวถอดรหัสจะใช้เพียงตัวเลือกละ 3 สัญลักษณ์ ก็สามารถถอดรหัสข้อมูลที่มีความผิดพลาดอย่างง่ายได้แล้ว หากการถอดรหัสโดยใช้ตัวเลือกละ 3 สัญลักษณ์ยังไม่สามารถถอดรหัสได้จึงค่อยนำข้อมูลตัวเลือกถัดไปอีกตัวเลือกละ 3 สัญลักษณ์มาช่วยในการถอดรหัส ดังนั้นจึงไม่จำเป็นต้องรับข้อมูลมาให้ครบทั้ง 72 สัญลักษณ์แล้วค่อยทำการถอดรหัส ซึ่งจะทำให้ใช้สมบัติความเป็นรหัสคอนโวลูชันได้ไม่เต็มที่

เนื่องจากข้อมูลนำมาถอดถูกเข้ารหัสด้วยรหัสคอนโวลูชัน (3, 2, 2) ในกรณีที่มีตัวเลือกเดียวจะต้องทำการถอดรหัสจะทำครั้งละ 3 สัญลักษณ์ หรือเรียกว่าถอดรหัสด้วยซินโดรมเดียว แต่ในกรณีที่มีตัวเลือกเดียวนั้น จะยังไม่สามารถถอดรหัสด้วย 3 สัญลักษณ์ได้ วิธีเวกเตอร์ซิมโบลดีโคดดิงจะนำข้อมูลอีก 3 สัญลักษณ์ถัดไปมาช่วยถอดรหัส หรือเรียกว่าถอดรหัสด้วยซินโดรมหลายตัว ซึ่งจะรวมเป็นทำการถอดรหัสครั้งละ 6 สัญลักษณ์ เพื่อให้ได้ผลลัพธ์ 4 สัญลักษณ์ ดังรูปที่ 19 ถ้าไม่ยังสามารถถอดรหัสด้วย 6 สัญลักษณ์ได้ จะข้อมูลอีก 3 สัญลักษณ์ถัดไปมาช่วยถอดรหัส เป็นเช่นนี้เรื่อยๆ จนถึงค่าที่จำกัดไว้ โดยในงานวิจัยนี้กำหนดไว้ไม่เกิน 30 สัญลักษณ์ต่อการถอดรหัสหนึ่งครั้ง หรือไม่เกินรอบละ 10 ซินโดรม ส่วนในกรณีที่มี 2 ตัวเลือก จะทำการถอดรหัสตัวเลือกละ 3 สัญลักษณ์ต่อครั้ง ผลลัพธ์การถอดรหัสที่ได้ จะเหมือนกับกรณีที่มีตัวเลือกเดียวคือมีจำนวน 2 สัญลักษณ์ นั่นก็คือในกรณีที่มี 2 ตัวเลือกจำนวนผลลัพธ์ที่ได้จะเท่ากับในกรณีที่มีตัวเลือกเดียว แต่จำนวนสัญลักษณ์ที่ป้อนเข้าตัวถอดรหัส จะมีมากกว่ากรณีที่มีตัวเลือกเดียวสองเท่า อย่างไรก็ตามก็หากสัญลักษณ์มีความผิดพลาด จำนวนสัญลักษณ์ขั้นต่ำที่จะสามารถถอดรหัสได้สำเร็จคือ 6 สัญลักษณ์ ซึ่งกรณีที่มีตัวเลือกเดียวต้องใช้ตัวเลือกอันดับหนึ่ง 6 สัญลักษณ์ ส่วนกรณีที่มีตัวเลือก 2 ตัวต้องใช้ตัวเลือกอันดับหนึ่งและสองอย่างละ 3 สัญลักษณ์

การถอดรหัสของคำรหัสหนึ่งคำของงานวิจัยนี้ ต้องทำการส่งข้อมูลตัวเลือกอันดับหนึ่งและสอง จากคอมพิวเตอร์เข้าสู่บอร์ด FPGA รวม 72 สัญลักษณ์ สัญลักษณ์ละ 32 บิต คิดเป็น 2304 บิต และผลลัพธ์ที่ได้จากการถอดรหัสจะส่งจากบอร์ด FPGA เข้าสู่คอมพิวเตอร์มีขนาด 24 สัญลักษณ์ คิดเป็น 768 บิต แต่เนื่องจากได้นำ USB-module ที่สามารถรับหรือส่งได้ครั้งละ 8 บิตมา

ใช้งาน ซึ่งทำให้ไม่สามารถรับส่งข้อมูลภายในครั้งเดียว จึงต้องสร้างข้อตกลงระหว่างคอมพิวเตอร์และ FPGA ไว้ในรูปที่ 18 แล้วเขียนโปรแกรมทั้งสองส่วนให้สอดคล้องกัน เพื่อให้รับส่งข้อมูลได้ครบถ้วน โดยในสถานะเริ่มต้นคอมพิวเตอร์กับ USB-module ต้องทำการสร้างการเชื่อมต่อกันก่อน ส่วน FPGA จะรอรับข้อมูลที่จะส่งมาจาก USB-module ในสถานะส่งข้อมูลที่ต้องการถอดรหัสไปยังบอร์ด FPGA จะเป็นการส่งข้อมูลครั้งละ 8 บิตจากคอมพิวเตอร์ผ่าน USB-module เข้าสู่ FPGA แล้วเขียนลงโลจิกเกท แล้วจึงรับส่งข้อมูล 8 บิตต่อไปจนข้อมูลครบ 6 สัญลักษณ์ เมื่อครบทุกสัญลักษณ์แล้วจะเข้าสู่สถานะ FPGA กำลังถอดรหัส ซึ่งคอมพิวเตอร์และ USB-module จะอยู่ในภาวะรอสัญญาณจาก FPGA ที่กำลังถอดรหัส ซึ่งในการถอดรหัสผลลัพธ์ที่ได้จะมี 3 กรณี คือ ยังไม่สามารถถอดรหัสได้ต้องใช้ข้อมูลในการถอดรหัสเพิ่มเติม ถอดรหัสได้สำเร็จ และถอดรหัสล้มเหลว ซึ่งได้แยกข้อตกลงการรับส่งออกเป็น 3 สถานะตามลำดับดังนี้ สถานะ FPGA ร้องขอข้อมูลเพิ่มเมื่อยังถอดรหัสไม่ได้ สถานะ FPGA ถอดรหัสสำเร็จ และสถานะ FPGA ถอดรหัสล้มเหลว

สถานะ FPGA ร้องขอข้อมูลเพิ่มเมื่อยังถอดรหัสไม่ได้ เป็นการที่ FPGA จะทำการส่ง Header ขนาด 32 บิตผ่าน USB-module ไปบอกให้คอมพิวเตอร์ส่งข้อมูลที่ต้องการถอดรหัสมาให้เพิ่มเติม ในสถานะ FPGA ถอดรหัสสำเร็จ จะเป็นการส่งข้อมูลที่ถอดรหัสแล้วกลับสู่คอมพิวเตอร์ผ่าน USB-module โดยต้องทำการส่ง Header ก่อน แล้วตามด้วยข้อมูลที่ถอดรหัสแล้ว ในสถานะสุดท้ายสถานะ FPGA ถอดรหัสล้มเหลว จะเป็นการแสดงผลว่าการถอดรหัสล้มเหลว (Decoding failure) โดยทำการแสดงผลที่ LED0 บนบอร์ด FPGA



สถานะ			
เริ่มต้น	สร้างการเชื่อมต่อกับ USB	สร้างการเชื่อมต่อกับ PC	รอรับสัญญาณจาก USB
ส่งข้อมูลที่ต้องการถอดรหัสไปยังบอร์ด FPGA	ส่งข้อมูลครั้งละ 8 บิทจนครบ	ส่งผ่านข้อมูลครั้งละ 8 บิทไปยัง FPGA	รับข้อมูลครั้งละ 8 บิท แล้วเขียนลงโลจิกเกท
FPGA กำลังถอดรหัส	รอรับ Header 32 บิท	รอ	ประมวลผลการถอดรหัส
FPGA ร้องขอข้อมูลเพิ่มเมื่อยังถอดรหัสไม่ได้	รับ Header จนครบ	ส่งผ่านข้อมูลครั้งละ 8 บิทไปยัง PC	ส่ง Header ขนาด 32 บิท ทีละ 8 บิทจนครบ
FPGA ถอดรหัสสำเร็จ	รับ Header จนครบ แล้วจึงรับข้อมูลที่ถอดรหัสแล้วมาเขียนลงไฟล์	ส่งผ่านข้อมูลครั้งละ 8 บิทไปยัง PC	ส่ง Header ขนาด 32 บิท ทีละ 8 บิทจนครบ ตามด้วยส่งข้อมูลที่ถอดรหัสแล้ว ทีครั้งละ 8 บิทจนครบ
FPGA ถอดรหัสไม่สำเร็จ	รอรับ Header 32 บิท	รอ	แสดงสัญญาณไฟ LED0

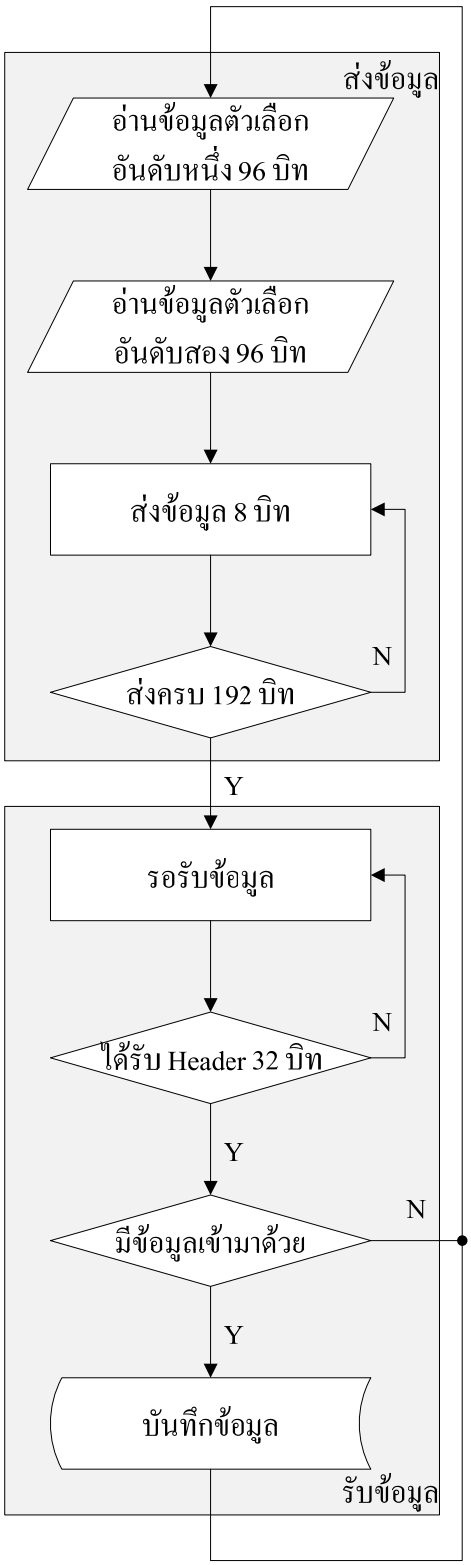
ภาพที่ 18 ข้อตกลงการรับส่งข้อมูลระหว่างคอมพิวเตอร์ USB-module และบอร์ด FPGA

หลังจากมีการสร้างข้อตกลงการรับส่งข้อมูลแล้ว จะเป็นการกำหนดการรับส่งที่คอมพิวเตอร์ เริ่มด้วยการส่งข้อมูลตัวเลือกทั้งสองไปยังบอร์ด FPGA โดยมีการเรียงข้อมูลที่จะส่งจากคอมพิวเตอร์ จะมีลำดับดังรูปที่ 19 โดยแต่ละช่องจะหมายถึงสัญลักษณ์ขนาด 32 บิต การส่งในแต่ละรอบการถอดรหัสจะส่งครั้งละ 6 สัญลักษณ์ตามลำดับที่ระบุ ประกอบด้วยตัวเลือกอันดับหนึ่ง 3 สัญลักษณ์ และตัวเลือกอันดับสอง 3 สัญลักษณ์ โดยในรอบแรกจะเป็นการส่งตัวเลือกอันดับหนึ่งสัญลักษณ์ลำดับที่ 1 ถึง 3 ในรูปที่ 19 แล้วจึงเป็นการส่งตัวเลือกอันดับสองสัญลักษณ์ลำดับที่ 4 ถึง 6 ในรูปที่ 19 ซึ่งการส่งแต่ละรอบการถอดรหัสจะส่งรวม 6 x 32 = 192 บิต และถ้าส่งด้วย USB-module ที่มีบัสขนาด 8 บิต จะต้องส่ง 192 / 8 = 24 คาบ จึงจะครบ 192 บิต เมื่อส่งครบ 192 บิต หรือ

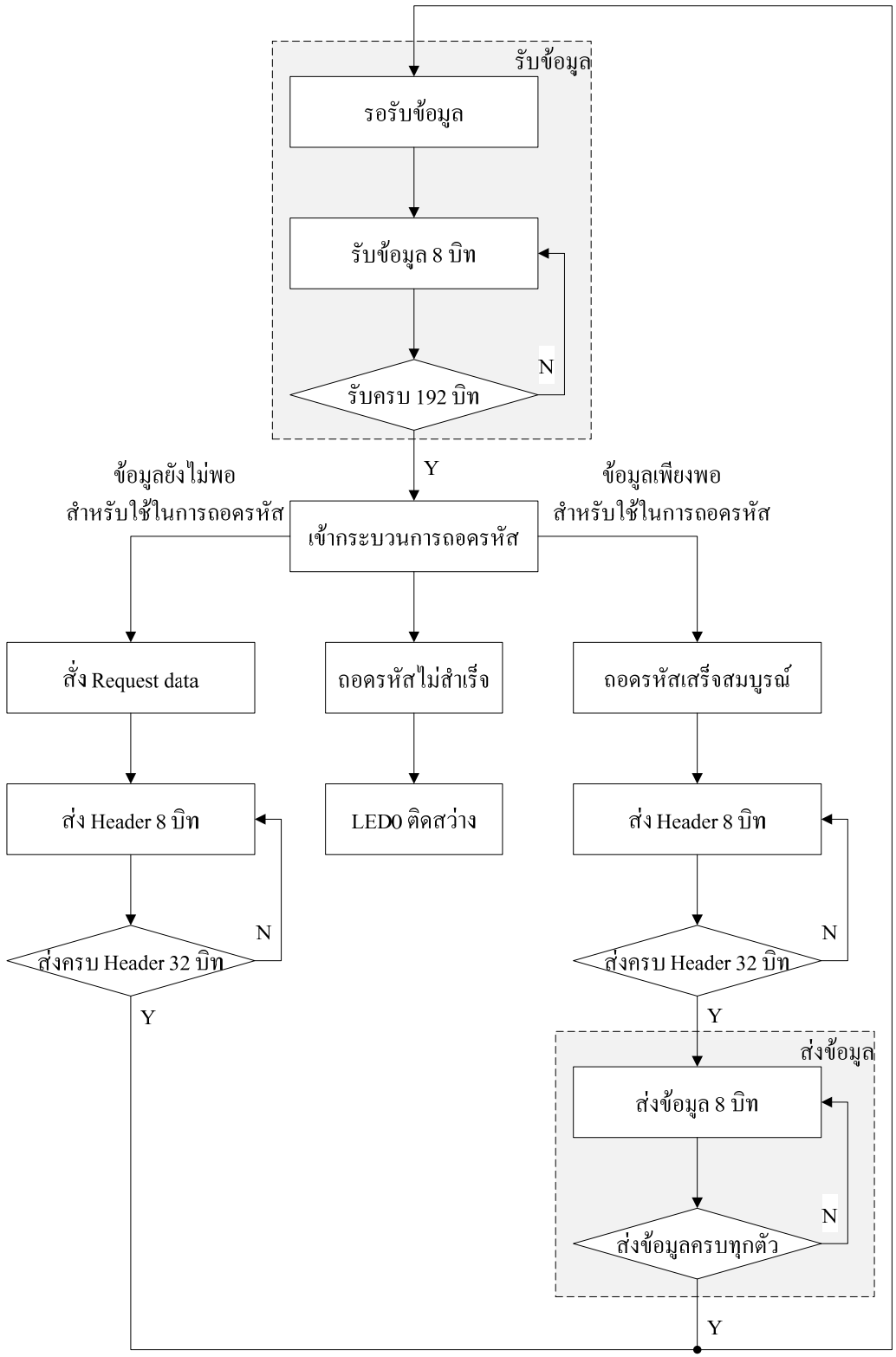
6 สัญลักษณ์แล้ว จะรอรับข้อมูลที่ส่งมาจากบอร์ด FPGA หากได้รับข้อมูลส่วนหัว (Header) จะทำการเขียนข้อมูลที่ได้รับต่อจากข้อมูลส่วนหัวลงคอมพิวเตอร์ จากนั้นจึงจะเริ่มส่งข้อมูลในรอบถัดไปอีก 192 บิต คือลำดับที่ 7 ถึง 12 ในรูปที่ 8 ส่วนถ้าไม่มีข้อมูลอื่นได้รับแนบมากับข้อมูลส่วนหัว ให้ข้ามขั้นตอนการเขียนข้อมูลไปทำการส่งข้อมูลเลย ซึ่งได้เขียนเป็นขั้นตอนไว้ในรูปที่ 20

ภาพที่ 19 ลำดับการรับส่งข้อมูลแต่ละสัญลักษณ์

ส่วน FPGA ได้กำหนดขั้นตอนการรับส่งข้อมูลไว้ในรูปที่ 21 อธิบายได้ดังนี้ ตัวถอดรหัสจะรอรับข้อมูลจากคอมพิวเตอร์ ถ้ามาข้อมูลเข้ามาจะทำการรับข้อมูลครั้งละ 8 บิต จนครบ 192 บิต จากนั้นจะเข้าสู่กระบวนการถอดรหัส ซึ่งจะได้ผลลัพธ์สองแบบคือ ถอดรหัสสำเร็จก็จะทำการส่งข้อมูลส่วนหัว (Header) และข้อมูลที่ถอดรหัสแล้วกลับสู่คอมพิวเตอร์ ครั้งละ 8 บิตจนครบ อีกกรณีคือยังถอดรหัสไม่ได้ ก็จะส่งเฉพาะข้อมูลส่วนหัวให้แก่คอมพิวเตอร์ เพื่อร้องขอข้อมูล ส่วนกรณีสุดท้ายคือการถอดรหัสล้มเหลวจะทำการแสดงผลที่ LED0 บนบอร์ด FPGA



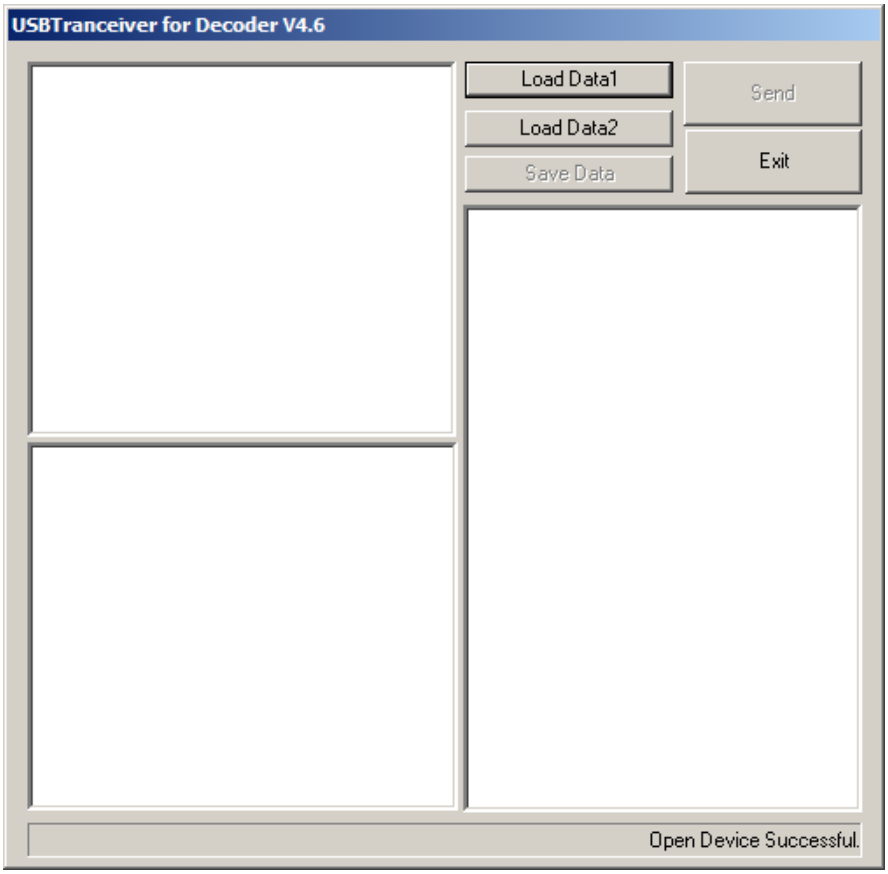
ภาพที่ 20 ขั้นตอนการรับส่งข้อมูลของโปรแกรมคอมพิวเตอร์ที่ใช้ติดต่อกับ USB-module



ภาพที่ 21 ขั้นตอนการรับส่งข้อมูลของโปรแกรมบน FPGA ที่ใช้ติดต่อกับ USB-module

2.2 โปรแกรมคอมพิวเตอร์ที่ใช้ติดต่อกับ USB-module

งานวิจัยนี้ ได้นำบอร์ด FPGA มาเชื่อมต่อกับคอมพิวเตอร์ โดยใช้การเชื่อมต่อประสานอุปกรณ์ (Interfacing) เป็น USB-module ที่ใช้ไอซีรุ่น FT245BM ของบริษัท FTDI จำกัด ซึ่งการนำมาใช้งานจะต้องโปรแกรมที่บอร์ด FPGA และคอมพิวเตอร์ ให้ทำงานสัมพันธ์กับ USB-module โดยการโปรแกรมที่คอมพิวเตอร์ได้นำ D2XX Direct Drivers (Future Technology Devices International Ltd. [FTDI], 2002) ของ FTDI มาใช้งาน ซึ่งมีความสามารถสร้างการสื่อสารระหว่าง USB-module กับระบบปฏิบัติการคอมพิวเตอร์ได้ โดยการสร้างโปรแกรมที่ใช้รับส่งข้อมูลบนคอมพิวเตอร์นี้ สามารถสร้างมาจากการนำเครื่องมือของ D2XX Direct Drivers ไปผนวกลงใน Microsoft Visual C++ ซึ่งโปรแกรมที่ใช้สื่อสารกับ USB-module ในงานวิจัยนี้มีชื่อว่า “USB Transceiver” มีหน้าตาดังรูปที่ 22



ภาพที่ 22 หน้าต่างของโปรแกรม USB Transceiver

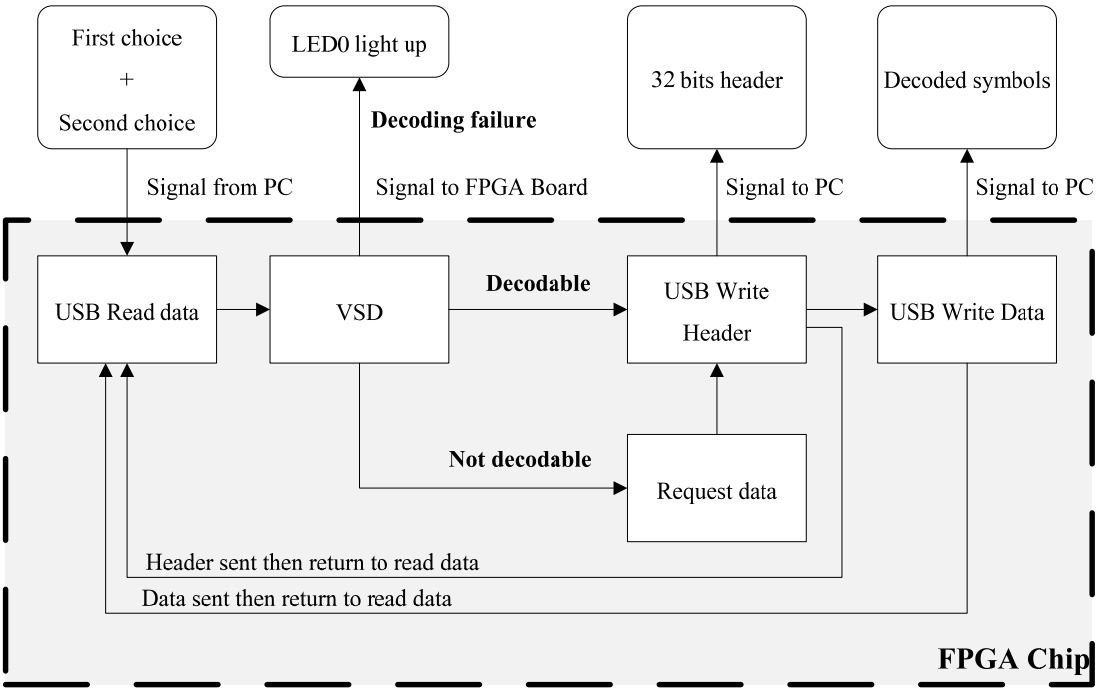
USB Transceiver มีความสามารถในการสร้างการติดต่อกับ USB-module และรับส่งข้อมูลที่เป็นไฟล์ขนาดไม่เกิน 1 ล้านไบท์ผ่านทาง USB-module โปรแกรมนี้จะเริ่มต้นทำงานด้วยการสร้างการติดต่อระหว่างคอมพิวเตอร์กับ USB-module หลังจากที่สร้างการติดต่อสำเร็จแล้ว โปรแกรมจะให้เลือกไฟล์ที่ต้องการส่ง แล้วจึงเริ่มแบ่งส่งไฟล์ข้อมูลไปยัง USB-module และรับข้อมูลที่มาจาก USB-module มาเก็บลงไฟล์ที่ระบุ โดยการรับส่งข้อมูลมีการจัดลำดับของข้อมูลดังรูปที่ 19 และมีขั้นตอนการทำงานในการรับส่งข้อมูลดังแผนภาพในรูปที่ 20 สุดท้ายโปรแกรมจะยุติการติดต่อกับ USB-module โดยโปรแกรมจะเรียกใช้ฟังก์ชันที่มากับ D2XX Direct Drivers (FTDI, 2002) ในทุกขั้นตอนที่มีการเรียกใช้งาน USB-module

3. การออกแบบสเตต (State) การทำงาน

การออกแบบสเตตการทำงานนี้ เป็นการออกแบบการทำงานเพื่อที่จะนำไปเขียนเป็นภาษา VHDL จึงต้องมีการกำหนดลำดับขั้นตอนและเงื่อนไขต่างๆอย่างละเอียด แต่ตัวถอดรหัสมีความซับซ้อน จึงต้องแบ่งสเตตการทำงานเป็น 2 กลุ่มใหญ่ เพื่อให้การออกแบบการทำงานของตัวถอดรหัสเป็นไปอย่างมีระบบ ซึ่งได้แก่ กลุ่มแรกเป็นกลุ่มของสเตตของตัวจัดการข้อมูล เป็นสเตตที่ไม่เกี่ยวข้องกับการถอดรหัส เช่น การรับส่งข้อมูล การร้องขอข้อมูล เป็นต้น กลุ่มที่สองเป็นกลุ่มสเตตของตัวถอดรหัส ซึ่งล้วนเป็นขั้นตอนหนึ่งในการถอดรหัสทั้งสิ้น เช่น การเลือกตัวถอดรหัส การถอดรหัสด้วยซินโดรมเดียว การแก้ไขด้วยวิธีใช้ตัวเลือกอันดับสอง การแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ เป็นต้น

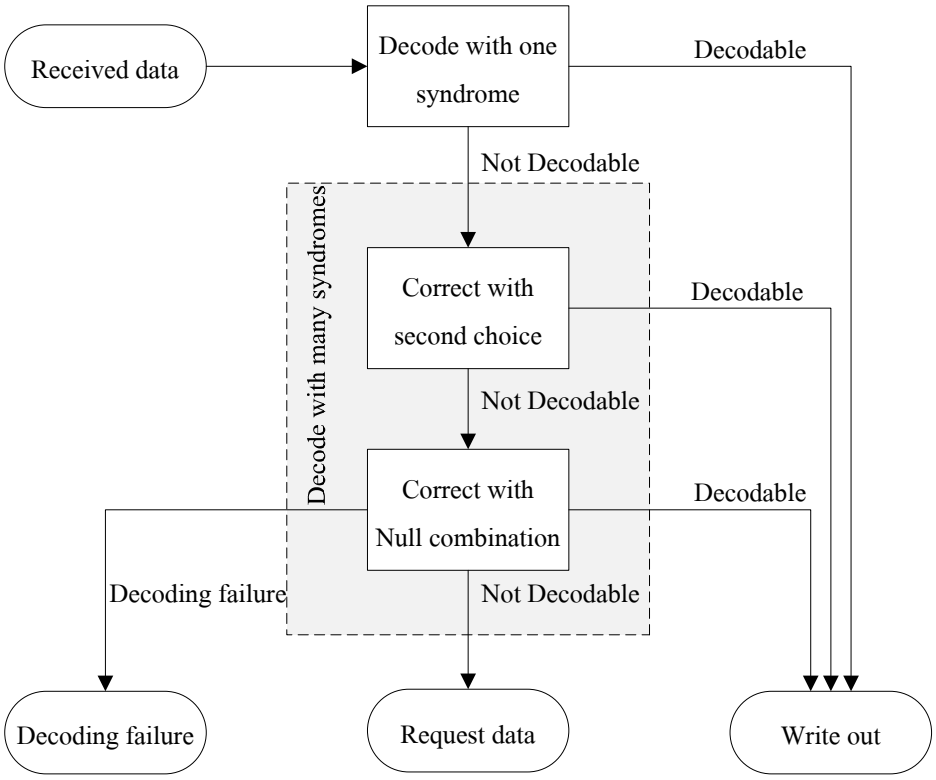
ก่อนที่จะออกแบบสเตตได้นั้น ต้องร่างแผนภาพรวมของการทำงานก่อน เพื่อให้ทราบถึงลำดับขั้นตอนการทำงาน โดยจะเริ่มที่การร่างแผนภาพรวมของการทำงานของตัวจัดการข้อมูล ในรูปที่ 23 ประกอบด้วยส่วนที่อยู่ในกรอบแสดงภาพรวมการทำงานของชิป FPGA และส่วนที่อยู่นอกกรอบเป็นสัญญาณภายนอกที่เชื่อมต่อกับชิป FPGA โดยสามารถอธิบายการทำงานได้ดังนี้ เริ่มต้นที่ USB Read data เป็นขั้นตอนของการรับข้อมูลตัวเลือกอันดับหนึ่งและสอง มาจากคอมพิวเตอร์ แล้วป้อนข้อมูลที่รับได้เข้าไปยัง VSD ซึ่งเป็นกลุ่มก้อนการทำงานของตัวถอดรหัส ที่ให้ผลลัพธ์การถอดรหัสแตกต่างออกไป ได้แก่ การถอดรหัสล้มเหลว จะแสดงผลที่ไฟ LED ของบอร์ด FPGA ถ้าถอดรหัสสำเร็จจะส่งข้อมูลออกไปยังขั้นตอน USB Write header ซึ่งจะทำการส่ง Header ขนาด 32 บิตสู่คอมพิวเตอร์แล้วส่งต่อให้ขั้นตอน USB Write data เพื่อส่งข้อมูลที่ถอดรหัสแล้วต่อท้ายไปกับ Header เข้าสู่คอมพิวเตอร์ จากนั้นจึงกลับสู่ขั้นตอน USB Read data ย้อนกลับไปยังผลลัพธ์การ

ถอดรหัสอีกกรณีหนึ่งที่ยังไม่ได้อธิบายคือ เมื่อตัวถอดรหัสยังไม่สามารถถอดรหัสได้จะเป็นหน้าที่ของขั้นตอน Request data ที่จะแจ้งไปยัง USB Write header ให้ทำการส่ง Header ขนาด 32 บิต เพื่อเป็นการแจ้งให้คอมพิวเตอร์ส่งข้อมูลเพิ่มเติม เมื่อส่ง Header แล้วจะเข้าสู่ขั้นตอน USB Read data เพื่อรอรับข้อมูลที่ได้ร้องขอจากคอมพิวเตอร์



ภาพที่ 23 แผนผังการทำงานของตัวจัดการข้อมูลให้กับตัวถอดรหัส

สำหรับแผนภาพการทำงานของตัวถอดรหัสนั้น เพื่อให้เข้าใจได้ง่ายจึงได้ร่างแผนผังการทำงานอย่างง่ายขึ้นมาก่อนตามรูปที่ 24 ประกอบด้วยขั้นตอนการถอดรหัส 3 ขั้นตอน ซึ่งอธิบายได้ดังนี้ ในขั้นแรกตัวถอดรหัสจะรับข้อมูลเข้ามาถอดรหัสด้วยขั้นตอนการถอดรหัสด้วยซินโดรมเดียว (Decode with one syndrome) ถ้าถอดรหัสได้จะเขียนข้อมูลที่ถอดรหัสแล้วออกไปสู่คอมพิวเตอร์ หากถอดรหัสไม่ได้จะส่งให้ขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง (Correct with second choice) ทำการถอดรหัส ถ้าถอดรหัสได้จะเขียนข้อมูลที่ถอดรหัสแล้วออกไป หากถอดรหัสไม่ได้จะส่งให้ขั้นตอนการแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with Null combination) ทำการถอดรหัส ถ้าถอดรหัสได้จะเขียนข้อมูลที่ถอดรหัสแล้วออกไป หากถอดรหัสล้มเหลวจะแสดงผลออกทาง LED บนบอร์ด FPGA หากยังไม่สามารถถอดรหัสได้จะทำการร้องขอข้อมูลเพิ่มเติมที่ขั้นตอน Request data เพื่อให้คอมพิวเตอร์ส่งข้อมูลมาเพิ่มเติมแล้วจึงเริ่มถอดรหัสอีกครั้ง

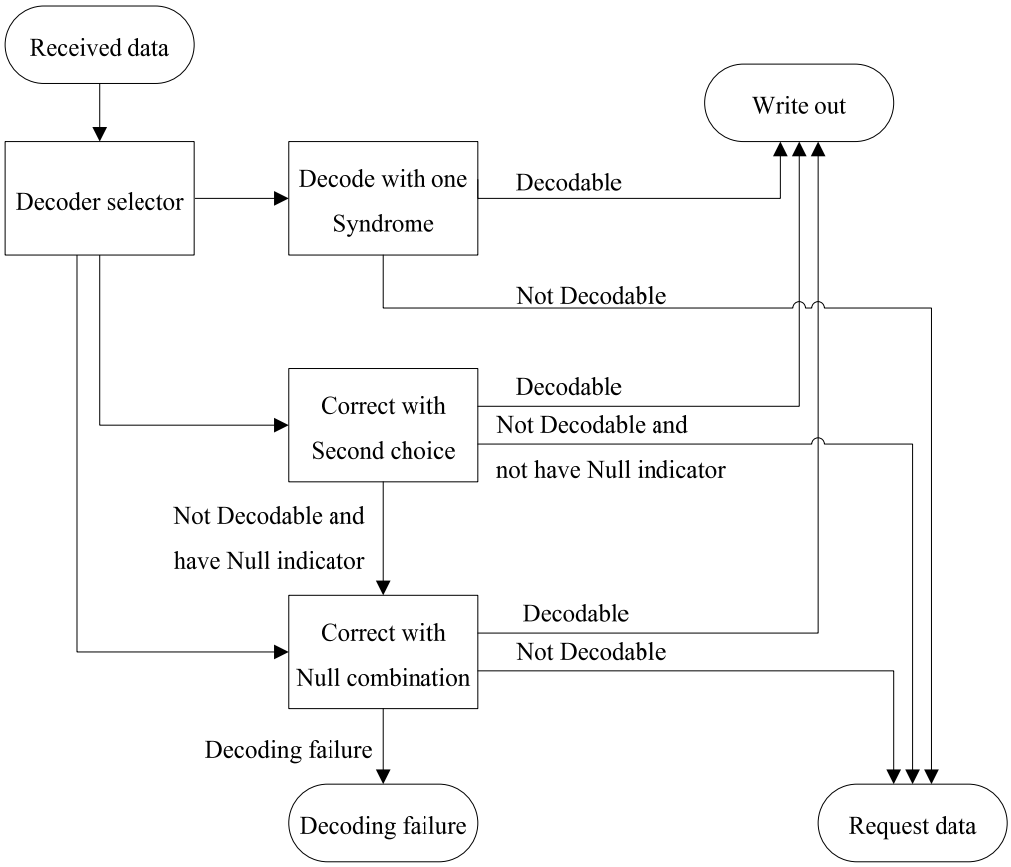


ภาพที่ 24 แผนผังการทำงานอย่างง่ายของตัวถอดรหัส

ในรูปที่ 24 เป็นเพียงการทำงานอย่างง่ายเท่านั้น หากจะนำไปใช้งานจริงยังต้องมีการกำหนดขั้นตอนเพิ่มเติม เพื่อให้ขั้นตอนการถอดรหัสกระชับและรัดกุม ดังแสดงไว้ในรูปที่ 25 ซึ่งขั้นตอนที่เพิ่มมาคือ การเลือกวิธีถอดรหัส (Decoding selector) มีหน้าที่ในการเลือกวิธีถอดรหัสให้ถูกต้อง สาเหตุที่ต้องมีขั้นตอนนี้เนื่องจากการถอดรหัสนั้นมีหลายเงื่อนไขและไม่ได้ทำงานต่อเนื่องกันทั้ง 3 ขั้นตอน จึงต้องมีขั้นตอนที่เลือกว่า จากข้อมูลที่มีสามารถใช้การถอดรหัสใดจึงจะเหมาะสม โดยจะมีการอธิบายเงื่อนไขในการเลือกวิธีการถอดรหัสไว้ในการอธิบายเกี่ยวกับสเตตของการเลือกวิธีถอดรหัสซึ่งเป็นเนื้อหาต่อจากนี้

จากรูปที่ 25 สามารถอธิบายขั้นตอนการทำงานได้ดังนี้ เมื่อรับข้อมูลมาจากคอมพิวเตอร์แล้ว ขั้นตอนการเลือกวิธีถอดรหัสจะทำการเลือกวิธีการที่เหมาะสม ซึ่งทุกวิธีการจะมีผลลัพธ์ที่เหมือนกันสองกรณีคือ กรณีที่ถอดรหัสได้จะเขียนข้อมูลออกไปยังคอมพิวเตอร์ กับกรณีที่ไม่สามารถถอดรหัสได้จะร้องขอข้อมูลจากคอมพิวเตอร์ โดยผลลัพธ์ที่ได้จากวิธีแก้ไขด้วยตัวเลขอันดับสองจะมีผลลัพธ์ที่ได้เพิ่มเติมจากที่กล่าวอยู่หนึ่งกรณีคือ เมื่อไม่สามารถถอดรหัสได้และมี

ข้อมูลที่เพียงพอต่อการใช้วิธีถอดรหัสด้วยการรวมกันทางพีชคณิตได้ศูนย์ จะส่งต่อให้ขั้นตอนการถอดรหัสด้วยการรวมกันทางพีชคณิตได้ศูนย์ทำการถอดรหัสต่อ นอกจากนี้ในขั้นตอนการถอดรหัสด้วยการรวมกันทางพีชคณิตได้ศูนย์ยังมีผลลัพธ์ที่ได้เพิ่มเติมจากที่กล่าวอยู่หนึ่งกรณีคือการถอดรหัสล้มเหลว ที่จะทำการรายงานผลไปยัง LED บนบอร์ด FPGA

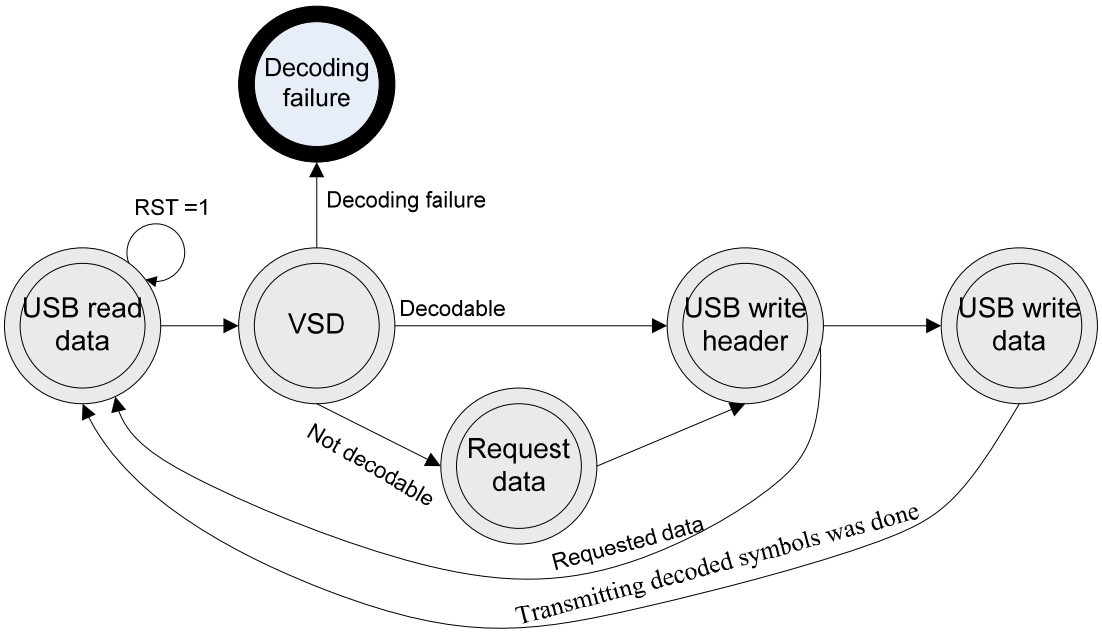


ภาพที่ 25 แผนผังการทำงานของตัวถอดรหัสที่ใช้งาน

3.1 กลุ่มสเตตของตัวจัดการข้อมูล (Group of data management states)

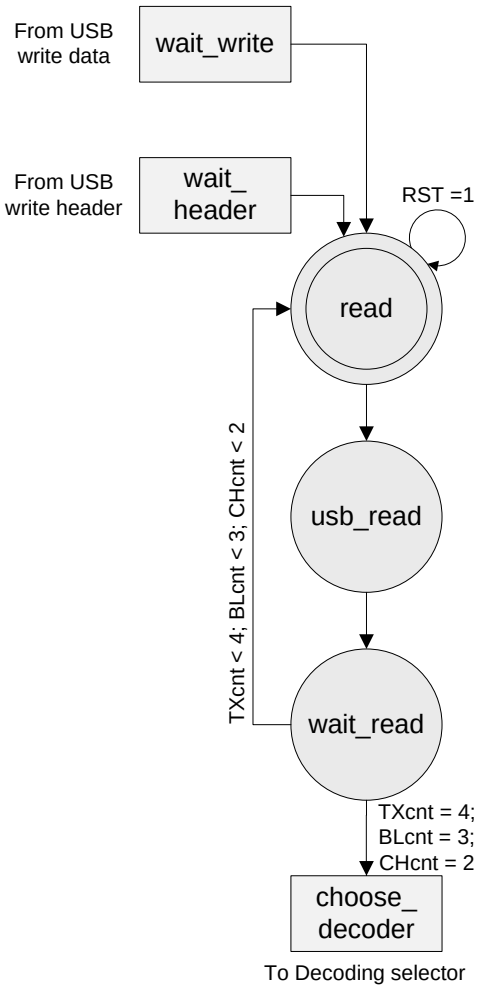
จากรูปที่ 23 สามารถเขียนให้เป็นสเตตการทำงานได้ดังรูปที่ 26 ซึ่งมี 4 กลุ่มย่อยเป็นสเตตของตัวจัดการข้อมูล คือ USB read data, Request data, USB write header และ USB write data ส่วนอีก 2 สเตตคือ Decoding failure เป็นสเตตสิ้นสุดการทำงานเมื่อไม่สามารถถอดรหัสได้ และสเตต VSD เป็นกลุ่มใหญ่ของตัวถอดรหัส ในหัวข้อนี้จะเป็นการแสดงให้เห็นถึงรายละเอียดของสเตตของตัวจัดการข้อมูลในแต่ละกลุ่มย่อยที่ได้ใช้งานจริงนั้นมีชื่อสเตตและสัญญาณอย่างไร ซึ่ง

สเตตของตัวจัดการข้อมูลนี้ ในกลุ่มย่อย USB read data และ USB write data นั้น ได้พัฒนามาจาก รัชนนท์ (2551) โดยได้มีการเพิ่มและลดสเตตไปหลายตัวโดยสเตตที่มิได้เปลี่ยนแปลงจากเดิมคือ read (wait_rxf_low), usb_read, split_tx (sort_tx), write (wait_txe_low) และ usb_write ส่วนสเตตที่เหลือเป็นสเตตที่ได้ทำการออกแบบเองทั้งสิ้น



ภาพที่ 26 แผนภาพสเตต แสดงภาพรวมของตัวจัดการข้อมูล

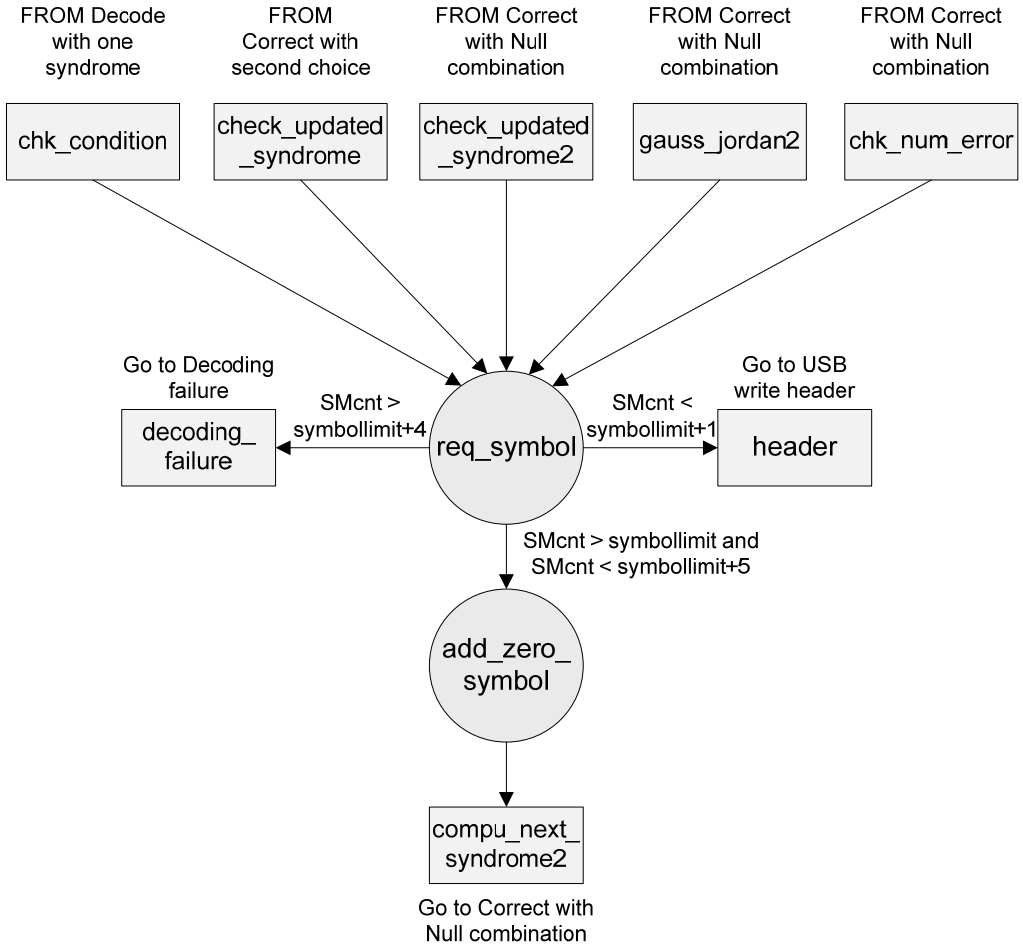
จากรูปที่ 27 เป็นสเตตการทำงานในขั้นตอน USB Read data ซึ่งจะเริ่มการทำงานที่สเตต read เป็นสเตตในการรับข้อมูลที่ส่งจากคอมพิวเตอร์ผ่านมาทาง USB-module ถ้าเป็นข้อมูลชุดแรกสุดการทำงานจะเริ่มขึ้นเมื่อมีข้อมูลส่งเข้ามา ถ้าข้อมูลที่จะถอดรหัสไม่ใช่สุดแรก การทำงานของสเตตนี้จะถูกกระตุ้นมาจากสเตต wait_write หรือ wait_header ซึ่งมีข้อแตกต่างกันคือ หากกระตุ้นด้วยสเตต wait_write เป็นรับข้อมูลเพื่อเริ่มถอดด้วยซินโดรมเดียว ส่วนถ้ากระตุ้นด้วย wait_header เป็นรับข้อมูลเพื่อนำไปถอดรหัสด้วยซินโดรมหลายตัว โดยการทำงานจะรับข้อมูลที่สเตต usb_read ครั้งละ 8 บิต แล้วตรวจสอบเงื่อนไขว่าได้รับข้อมูลครบหรือยังที่สเตต wait_read โดยจะมีตัวแปรสำหรับนับค่า 3 ตัวคือ TXcnt ทำการนับว่ารับครบหนึ่งสัญลักษณ์แล้วหรือไม่ BLcnt ทำการนับว่ารับมาครบ 3 สัญลักษณ์แล้วหรือไม่ CHcnt ทำการนับว่ารับมาครบทั้งสองตัวเลือกแล้วหรือไม่ หากยังไม่ครบให้เริ่มอ่านข้อมูลชุดต่อไปที่สเตต read จะส่งให้ขั้นตอนการเลือกวิธีถอดรหัส (Decoding selector) เข้าขั้นตอนการถอดรหัสต่อไป



ภาพที่ 27 แผนภาพสเตตตัวจัดการข้อมูล ขั้นตอน USB Read data

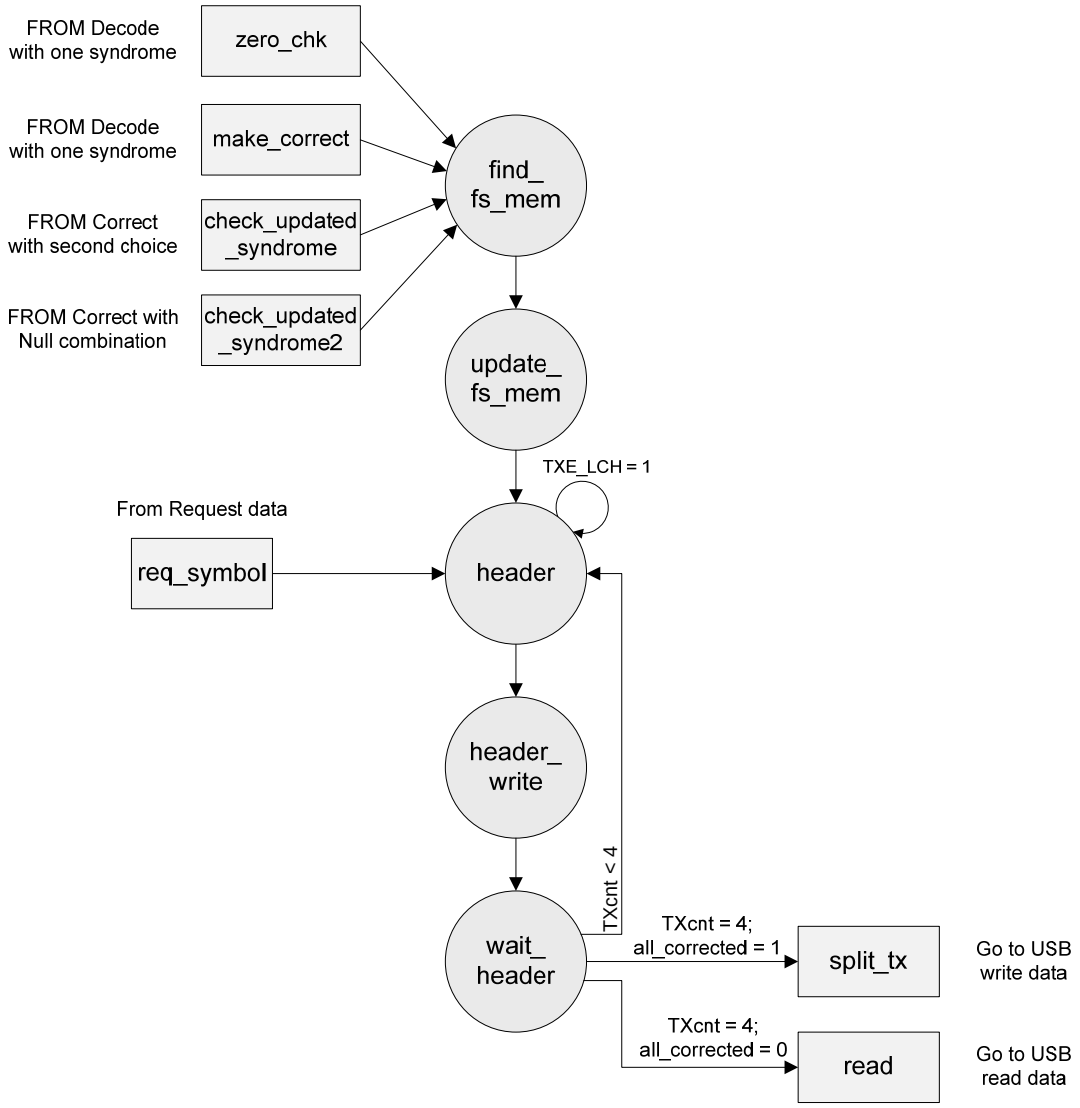
จากรูปที่ 28 เป็นสเตตการทำงานในขั้นตอน Request data จะทำหน้าที่ในการร้องขอสัญลักษณ์ชุดต่อไปมาช่วยถอดรหัสในกรณีที่ยังไม่สามารถถอดรหัสได้ การกระตุ้นการทำงานของ req_symbol จะมาจากหลายสเตตในกลุ่มสเตตของตัวถอดรหัส ซึ่งสเตต req_symbol จะเลือกระหว่างร้องขอสัญลักษณ์จากคอมพิวเตอร์โดยผ่านสเตต header ซึ่งจะทำการส่ง Header ขนาด 32 บิตไปยังคอมพิวเตอร์ กับอีกวิธีหนึ่งคือทำการร้องขอไปยัง add_zero_symbol ก็ต่อเมื่อข้อมูลในคอมพิวเตอร์ถูกส่งมาหมดแล้ว สเตตนี้จะทำหน้าที่เพิ่มสัญลักษณ์ศูนย์ให้ทำการถอดรหัสต่อไป นอกจากนี้หากได้ร้องขอข้อมูลเกินมากกว่าที่ได้กำหนดไว้จะถือว่าการถอดรหัสล้มเหลวจะรายงานไปยัง decoding_failure ต่อไป

เงื่อนไขในการเลือกว่าจะร้องขอข้อมูลจากสเตตใด จะใช้ตัวแปรนับ SMcnt เป็นตัวนับว่ามีค่าเกินกว่า symbollimit หรือไม่ และเกินกว่ามากเพียงใด โดยค่า symbollimit เป็นค่าที่กำหนดว่าคำรหัสมีความยาวเท่าใด เช่นในที่นี้รหัสคอนโวลูชัน (3, 2, 2) มีความยาวคำรหัส 36 สัญลักษณ์ จะได้ symbollimit = 36 / 3 = 12 เป็นต้น พิจารณาเงื่อนไขในรูปที่ 28 จะได้ว่า จะเลือกสเตต header เมื่อ SMcnt ไม่มากกว่า symbollimit + 1 หมายความว่าจะร้องขอข้อมูลจากคอมพิวเตอร์เมื่อข้อมูลจากคอมพิวเตอร์ยังไม่หมด เงื่อนไขถัดมาคือ จะร้องขอข้อมูลจาก add_zero_symbol เมื่อ symbollimit < SMcnt < symbollimit + 5 หมายถึงจะทำการเติมสัญลักษณ์ที่เป็นศูนย์ให้เมื่อข้อมูลจากคอมพิวเตอร์หมดแล้วและการเติมสัญลักษณ์ยังอยู่ในเกณฑ์ที่จะสามารถช่วยถอดรหัสได้ ส่วนเงื่อนไขสุดท้ายคือ จะรายงานว่าการถอดรหัสล้มเหลวเมื่อ SMcnt > symbollimit+4 ซึ่งเงื่อนไขนี้จะเป็นไปได้เมื่อมีการเติมสัญลักษณ์ศูนย์แล้ว แต่ยังไม่สามารถช่วยในการถอดรหัสได้



ภาพที่ 28 แผนภาพสเตตตัวจัดการข้อมูล ขั้นตอน Request data

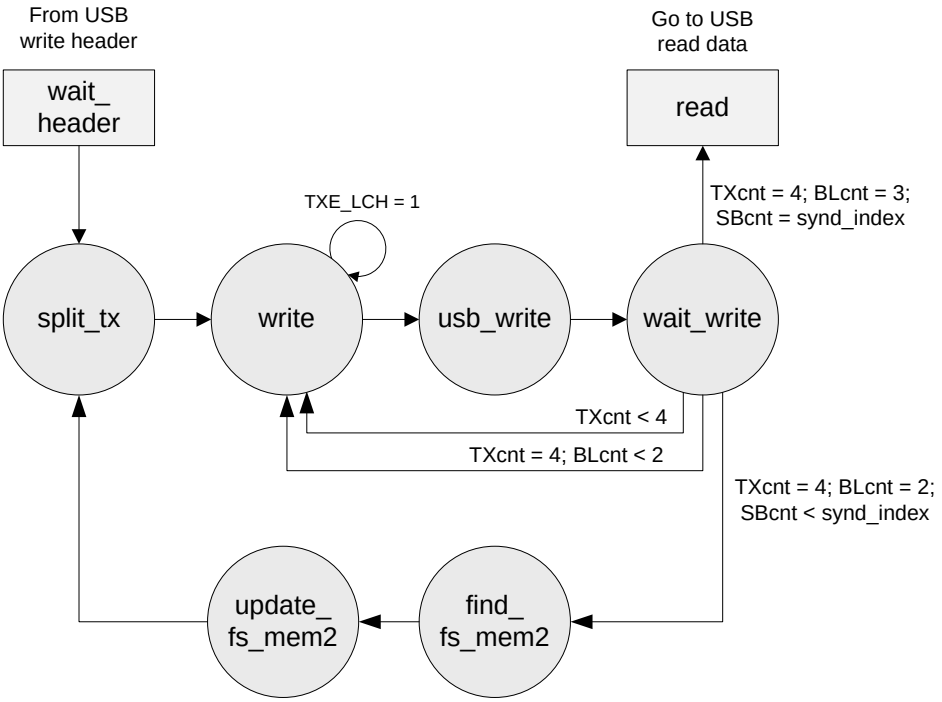
จากรูปที่ 29 เป็นสเตตการทำงานในขั้นตอน USB Write header มีหน้าที่หลักคือส่ง Header ขนาด 32 บิตไปยังคอมพิวเตอร์โดยผ่าน USB-module และยังมีหน้าที่อีกอย่างหนึ่งคือ รับข้อมูลที่ถอดรหัสสำเร็จแล้วมาหาค่า M_o ดังในสมการที่ 16 ซึ่งเป็นการหาค่าสำคัญสำหรับวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ก่อนที่จะส่งกลับไปยังคอมพิวเตอร์ ซึ่งจะทำที่สเตต find_fs_mem และ update_fs_mem โดยสเตต find_fs_mem จะถูกกระตุ้นมาจากหลายสเตตของตัวถอดรหัส ซึ่งล้วนแล้วเป็นสเตตที่ถูกกระตุ้นเมื่อสามารถถอดรหัสได้ ส่วนสเตตที่เหลืออีก 3 สเตต คือ header, header_write และ wait_header จะทำหน้าที่ส่ง Header ครั้งละ 8 บิต จนครบ 32 บิต โดยใช้ตัวแปรนับ TXcnt ในการนับและรอสัญญาณ Enable ในการส่งจาก TXE_LCH เมื่อส่ง Header ครบแล้ว สเตต wait_header จะตรวจสอบว่า สเตต Header ถูกสเตตใดกระตุ้น โดยใช้โลจิก all_correct เป็นตัวตัดสิน ถ้า all_correct เท่ากับโลจิกศูนย์จะทราบว่าเป็นสเตต req_symbol ซึ่งจะเป็นเพียงการร้องขอข้อมูล จะส่งต่อไปยังสเตต read เพื่อรอรับข้อมูลทันที ถ้า all_correct เท่ากับโลจิกหนึ่ง จะทราบว่าถูกกระตุ้นด้วยสเตต update_fs_mem จะเป็นการส่งข้อมูลที่ถอดรหัสเสร็จกลับไปยังคอมพิวเตอร์ จะต้องส่งผ่านไปยังสเตต split_tx เพื่อทำการส่งข้อมูลที่ถอดรหัสแล้วกลับออกไป



ภาพที่ 29 แผนภาพสเตตตัวจัดการข้อมูล ขั้นตอน USB Write header

จากรูปที่ 30 เป็นสเตตการทำงานในขั้นตอน USB Write data ทำหน้าที่ในการจัดเรียงและส่งข้อมูลที่ถอดรหัสแล้วกลับไปยังคอมพิวเตอร์ และยังมีหน้าที่หาค่า M_o สำหรับวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ในขั้นตอนนี้มีการกระตุ้นมาจากสเตต wait_header เท่านั้น ซึ่งหมายถึง การส่งข้อมูลที่ถอดรหัสแล้วกลับไปยังคอมพิวเตอร์จะเริ่มขึ้นเมื่อได้ทำการส่ง Header เสร็จเรียบร้อยแล้ว การส่งข้อมูลจะเริ่มที่สเตต split_tx ทำการเรียงข้อมูลที่ต้องการส่ง แล้วจึงตรวจสอบสัญญาณ Enable ของขา TXE เมื่อพร้อมจึงส่งออกไปยังคอมพิวเตอร์ผ่าน USB-module ที่สเตต usb_write โดยจะส่งครั้งละ 8 บิต เมื่อส่งแล้วจะตรวจสอบว่าส่งข้อมูลครบหรือยังที่สเตต wait_write ถ้าตรวจสอบตัวแปรนับ BLcnt แล้วได้ว่าข้อมูลยังส่งไม่ครบ 2

สัญลักษณ์จะทำการส่งต่อที่สเตต write แต่ถ้าส่งครบ 2 สัญลักษณ์แล้ว จะตรวจสอบว่าตัวแปรนับ SBcnt เป็นการถอดรหัสด้วยซินโดรมหลายตัวหรือไม่ ถ้าเป็นการถอดรหัสด้วยซินโดรมหลายตัว จะต้องทำการหาค่า M_o สำหรับวงจรดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับแล้วจึงทำการเขียนข้อมูลออกจนครบทุกสัญลักษณ์ที่ถอดรหัสแล้ว จึงจะส่งต่อไปยังสเตต read เพื่อทำการรับข้อมูลชุดถัดไปมาถอดรหัส

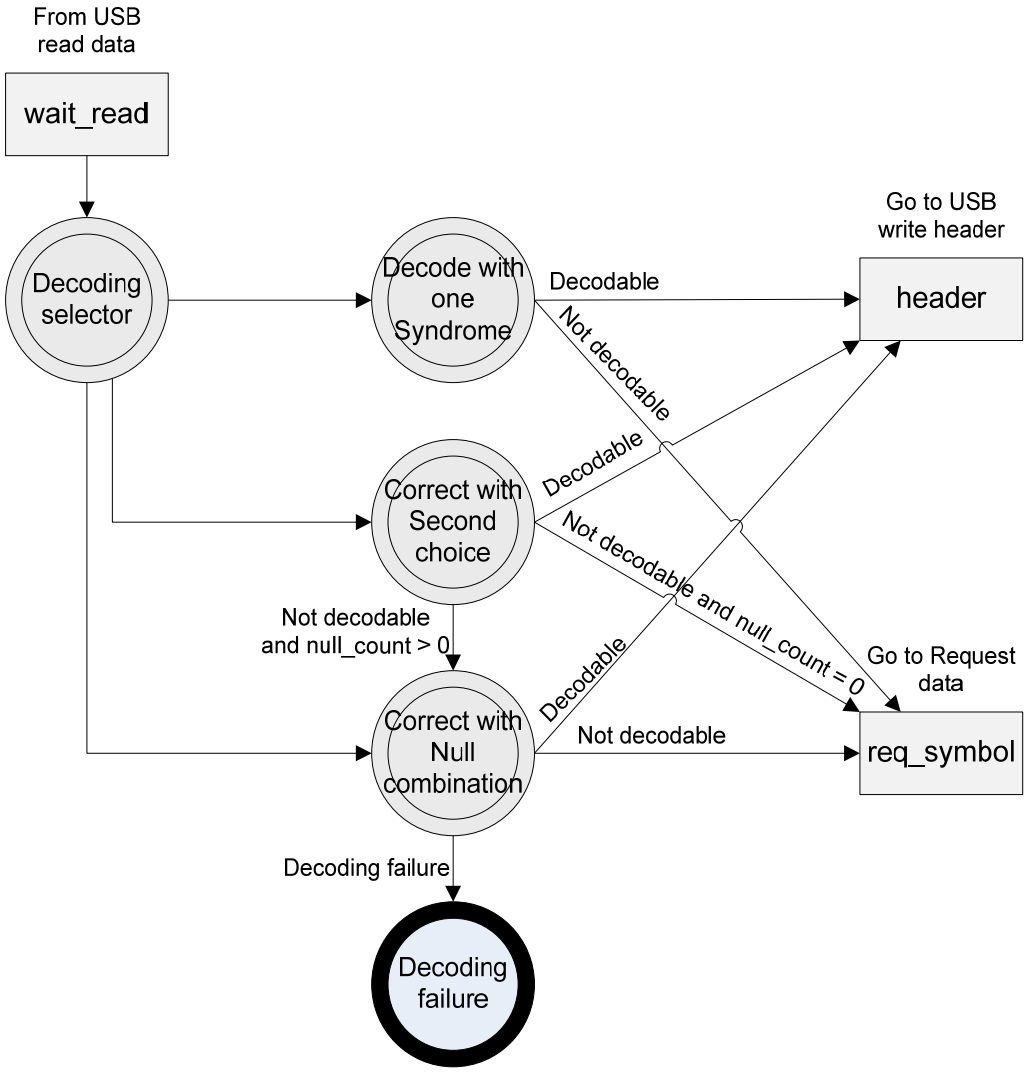


ภาพที่ 30 แผนภาพสเตตตัวจัดการข้อมูล ขั้นตอน USB Write data

3.2 กลุ่มสเตตของตัวถอดรหัส (Group of decoding states)

จากแผนผังการทำงานของตัวถอดรหัสที่ใช้งานในรูปที่ 25 สามารถเขียนให้เป็นแผนภาพสเตตได้ดังรูปที่ 31 โดยอธิบายได้ดังนี้ การถอดรหัสจะเริ่มทำงานเมื่อถูกกระตุ้นด้วยสเตต wait_read ซึ่งก็คือ เมื่อมีการรับข้อมูลครบตามจำนวนที่กำหนดแล้วจึงเริ่มถอดรหัสที่สเตต Decoding selector ซึ่งจะเป็นการเลือกว่าจะใช้วิธีการถอดรหัสแบบใด ซึ่งวิธีการถอดรหัสทั้ง 3 วิธี จะมีให้ผลลัพธ์ที่เหมือนกันอยู่ 2 กรณี คือเมื่อสามารถถอดรหัสได้ (Decodable) จะส่งต่อให้สเตต header ทำการส่ง Header พร้อมทั้งทำการส่งข้อมูลที่ถอดรหัสแล้วออกไป ส่วนผลลัพธ์ที่เหมือนกันอีกกรณีหนึ่งคือ ยังไม่สามารถถอดรหัสได้ (Not decodable) จะส่งคำสั่งร้องขอข้อมูลไปให้สเตต

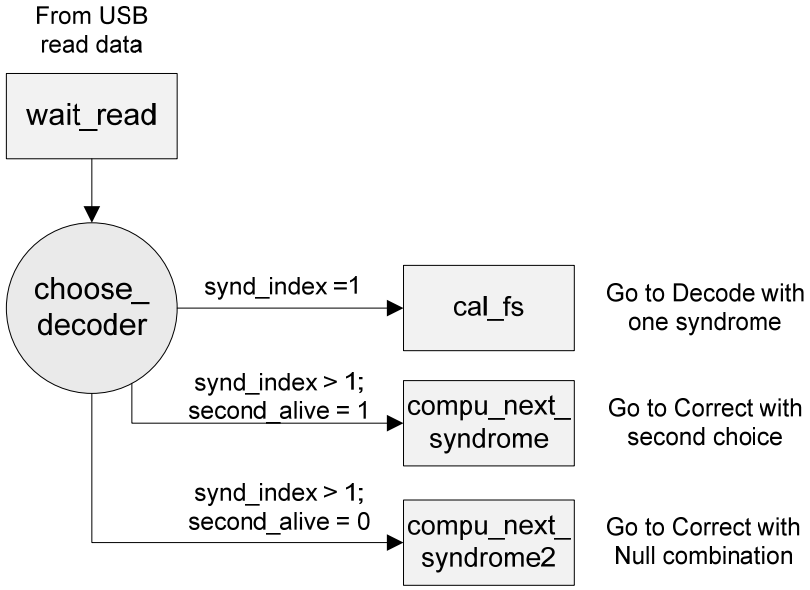
req_symbol นอกจากนี้ยังมีผลลัพธ์อีก 2 กรณีที่ต่างไปคือ ผลลัพธ์ของการแก้ไขด้วยตัวเลือกอันอับสอง (Correct with second choice) เมื่อยังไม่สามารถถอดรหัสได้ แต่ตรวจพบว่ามีข้อมูลเพียงพอที่จะแก้ไขด้วยการรวมกันทางพีชคณิตได้ศูนย์ (Correct with Null combination) ได้ จะส่งต่อไปให้วิธีดังกล่าวถอดรหัส และผลลัพธ์ในกรณีสุดท้ายคือ ผลลัพธ์ของการแก้ไขด้วยการรวมกันทางพีชคณิตได้ศูนย์ จะรายงานว่าการถอดรหัสล้มเหลว (Decoding failure) เมื่อการรหัสไม่สามารถดำเนินต่อได้เนื่องจากเงื่อนไขบางประการ เช่น แก้ไขความผิดพลาดด้วยวิธีการแก้ไขด้วยการรวมกันทางพีชคณิตได้ศูนย์แล้ว แต่ยังมีความผิดพลาดหลงเหลืออยู่ เป็นต้น โดยจะดูเงื่อนไขที่ทำให้การถอดรหัสล้มเหลวเนื่องจากแก้ไขด้วยการรวมกันทางพีชคณิตได้ศูนย์ ได้จากสเตตของการแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์



ภาพที่ 31 แผนภาพสเตตรวมของตัวถอดรหัส

3.2.1 สเตตของการเลือกวิธีการถอดรหัส (Decoding selector)

การเลือกวิธีการถอดรหัสทำได้โดยมีเงื่อนไขดังในรูปที่ 32 ซึ่งทำการถอดรหัสด้วยซินโดรมเดียว (Decode with one syndrome) ที่สเตต cal_fs เมื่อ Syndrome index เป็น 1 แล้วจะถอดรหัสด้วยวิธีแก้ไขด้วยตัวเลือกอันดับสอง (Correct with second choice) ที่สเตต compu_next_syndrome เมื่อ Syndrome index มากกว่า 1 และมีการรับข้อมูลตัวเลือกอันดับสองมาถอดรหัสด้วย ส่วนเงื่อนไขสุดท้ายคือ แก้ไขด้วยวิธีการรวมกันทางพีชคณิตได้ศูนย์ (Correct with null combination) ที่สเตต compu_next_syndrome2 เมื่อ Syndrome index มากกว่า 1 และไม่มีการรับข้อมูลตัวเลือกอันดับสองมาถอดรหัส

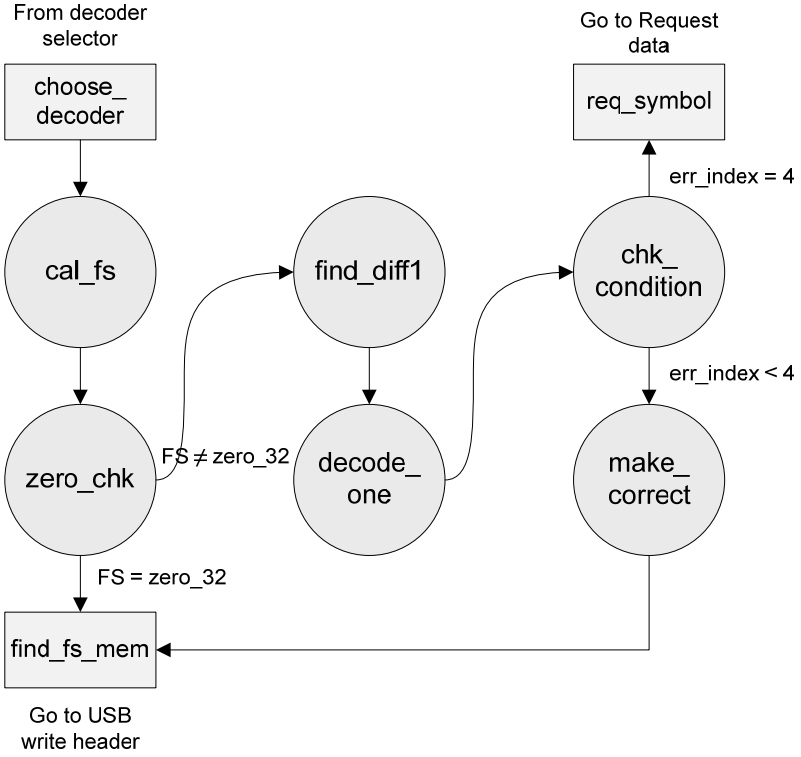


ภาพที่ 32 แผนภาพสเตตการเลือกวิธีการถอดรหัส

3.2.2 สเตตของการถอดรหัสด้วยซินโดรมเดียว (Decode with one syndrome states)

กลุ่มของสเตตนี้ ประกอบด้วย cal_fs, zero_chk, find_diff1, decode_one, chk_condition และ make_correct ดังแสดงไว้ในรูปที่ 33 โดยสเตตทั้งหมดที่กล่าวมารวมถึงการเขียนโปรแกรมได้นำมาจาก รัชนนท์ (2551) เว้นแต่ find_diff1 เป็นสเตตที่ได้ออกแบบการทำงานเอง เนื่องจากต้องทำการหาค่าผลต่างมีขนาดที่ใหญ่กว่าเดิม โดยมีการทำงานของแต่ละสเตตดังนี้ cal_fs จะเริ่มคำนวณหาค่าซินโดรมเมื่อได้รับคำสั่งมาจาก choose_decoder จากนั้นจะเมื่อคำนวณ

เสร็จแล้วจะส่งค่าซินโดรมไปยัง zero_chk เพื่อหาว่าซินโดรมเป็นศูนย์หรือไม่ ถ้าเป็นศูนย์จะส่งให้ find_fs_mem ดำเนินการส่งข้อมูลกลับออกไป ถ้าไม่เป็นศูนย์ต้องทำการแก้ไขโดยเริ่มจาก find_diff1 ทำการหาค่าผลต่างของตัวเลือกทั้งสอง แล้วส่งให้ decode_one ตรวจหาตำแหน่งของความผิดพลาด แล้วส่งให้ chk_condition ตรวจสอบว่าสามารถแก้ไขความผิดพลาดได้หรือไม่ ถ้าแก้ไขไม่ได้ให้รายงานผลไปยัง req_symbol ถ้าแก้ไขได้จะทำการแก้ไขที่ make_correct แล้วจึงส่งข้อมูลที่แก้ไขแล้วกลับออกไปผ่านสเตต find_fs_mem

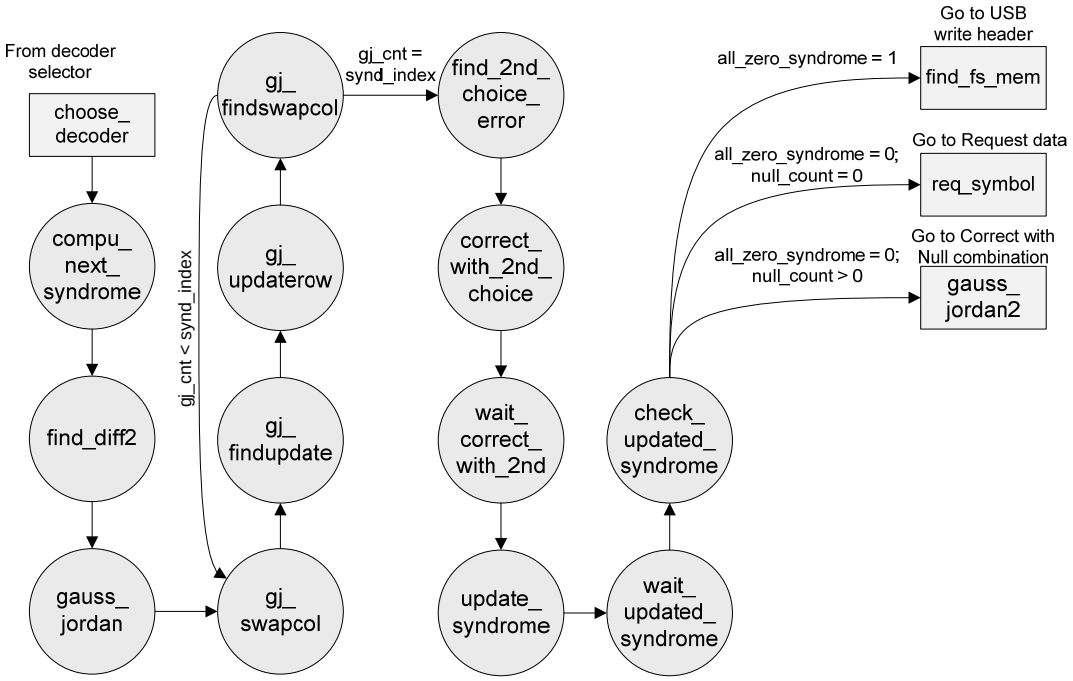


ภาพที่ 33 แผนภาพสเตตการถอดรหัสด้วยซินโดรมเดียว

3.2.3 สเตตของการแก้ไขด้วยตัวเลือกอันดับสอง (Correct with second choice states)

กลุ่มสเตตนี้ทำหน้าที่การถอดรหัสด้วยวิธีการแก้ไขด้วยตัวเลือกอันดับสอง เพื่อให้เข้าใจการทำงานของกลุ่มนี้ได้ง่ายให้ดูรูปที่ 34 ประกอบ การทำงานของกลุ่มสเตตนี้เริ่มทำงานเมื่อ choose_decoder เลือกว่าให้ใช้การถอดรหัสวิธีนี้ โดยเริ่มที่การหาค่าซินโดรมด้วย compu_next_syndrome แล้วหาค่าผลต่างระหว่างตัวเลือก อีก 5 สเตตถัดไป เป็นการทำ Gauss-Jordan reduction ให้กับซินโดรมดัดแปร (Modified syndrome) เสร็จแล้วจะให้สเตต

find_2nd_choice_error ทำการหาตำแหน่งของความผิดพลาดจากซินโดรมดัดแปรที่ทำ Gauss-Jordan reduction แล้ว จากนั้นจึงทำการแก้ไขค่าความผิดพลาดด้วย correct_with_2nd_choice และ wait_correct_with_2nd_choice ในส่วน 2 สเตตถัดมา update_syndrome และ wait_update_syndrome เป็นการหาค่าซินโดรมของข้อมูลที่แก้ไขค่าความผิดพลาดออกไปแล้ว และตรวจสอบค่าซินโดรมที่ check_updated_syndrome ถ้าซินโดรมเป็นศูนย์จะตัดสินใจว่าไม่มีความผิดพลาดเหลืออยู่ในข้อมูลที่ถอดรหัสแล้วจึงทำการส่งกลับไปยังคอมพิวเตอร์ผ่านสเตต find_fs_mem ถ้าซินโดรมไม่เป็นศูนย์จะตรวจสอบว่าจะสามารถแก้ไขได้ด้วยวิธีการรวมทางพีชคณิตได้ศูนย์ได้หรือไม่ ถ้าได้จะส่งให้ gauss_jordan2 ให้เริ่มทำการแก้ไข ถ้าไม่ได้จะทำการร้องขอข้อมูลเพิ่มด้วยสเตต req_symbol



ภาพที่ 34 แผนภาพสเตตการแก้ไขด้วยตัวเลือกอันดับสอง

3.2.4 สเตตของการแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination states)

สเตตกลุ่มนี้เป็นการทำงานของวิธีการรวมทางพีชคณิตได้ศูนย์ โดยจะเริ่มทำงานเมื่อได้รับคำสั่งจากสเตต choose_decoder หรือทำการถอดรหัสต่อจากขั้นตอนของแก้ไขด้วย

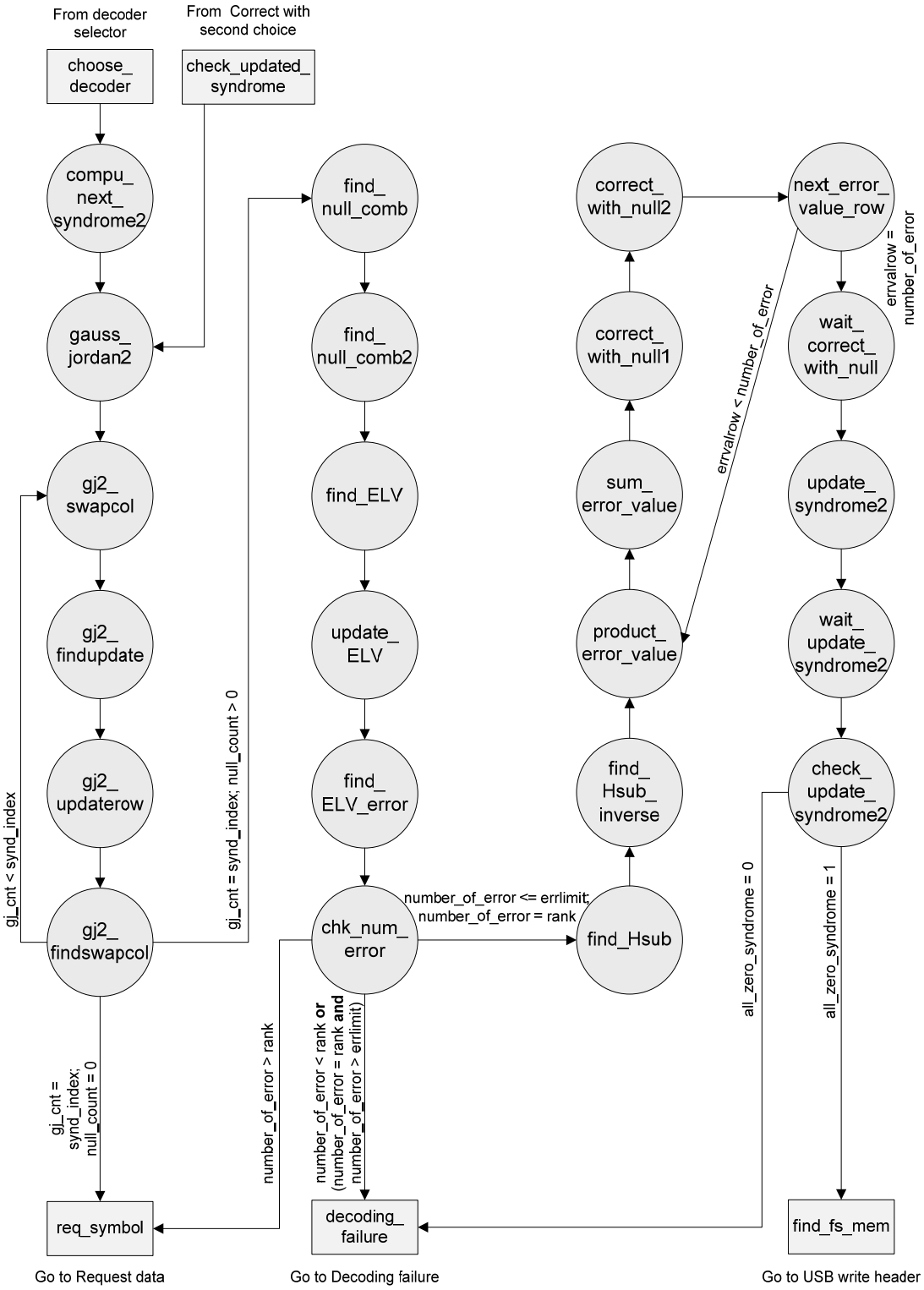
ตัวเลือกอันดับสองจากสเตต check_updated_syndrome ในกรณีที่รับคำสั่งจาก choose_decoder ต้องทำการหาค่าซินโดรมก่อนที่ compu_next_syndrome2 แล้วจึงทำ Gauss-Jordan reduction ให้กับซินโดรมที่สเตต gauss_jordan2 ถึงสเตต gj2_findswapcol ส่วนการถอดรหัสที่ทำต่อจาก check_updated_syndrome ให้เริ่มถอดรหัสที่ gauss_jordan2 เลย เมื่อได้ซินโดรมที่ลดรูปแล้ว สเตต gj2_findswapcol จะตรวจสอบว่าสามารถถอดรหัสด้วยวิธีการรวมทางพีชคณิตได้ศูนย์ได้หรือไม่ ถ้าไม่ได้ให้ทำการร้องขอข้อมูลที่สเตต req_symbol ถ้าได้ให้คำนวณหาค่าผลรวมทางพีชคณิตได้ศูนย์ (Null combination)

ที่สเตต find_null_comb และ find_null_comb2 โดยหาได้จากค่าบ่งชี้ศูนย์ (Null indicator) ที่หาได้จากขั้นตอนการทำ Gauss-Jordan reduction อีก 3 สเตตถัดมา ได้แก่ find_ELV, update_ELV และ find_ELV_error เป็นการหาเวกเตอร์ระบุตำแหน่งความผิดพลาด (Error locating vector) พร้อมทั้งหาค่าตำแหน่งความผิดพลาดและจำนวนสัญลักษณ์ที่ผิดพลาดที่ได้จาก ELV จากนั้น

ที่สเตต chk_num_error เป็นการตรวจสอบว่าจำนวนสัญลักษณ์ที่ผิดพลาดที่ตรวจพบนั้นมีมากหรือน้อยไปสำเร็จการถอดรหัสหรือไม่ ถ้าจำนวนสัญลักษณ์ที่ผิดพลาดที่ตรวจพบมีมากกว่าค่าอันดับ (Rank) ของซินโดรม หรือมีค่าเท่ากับค่าอันดับของซินโดรมแต่มีมากเกินกว่า 4 สัญลักษณ์จะถือว่าการถอดรหัสล้มเหลวและรายงานไปยังสเตต decoding_failure ถ้าจำนวนสัญลักษณ์ที่ผิดพลาดที่ตรวจพบมีมากกว่าค่าอันดับของซินโดรมจะถือว่าข้อมูลยังมีไม่เพียงพอที่จะให้ถอดรหัสให้ร้องขอข้อมูลเพิ่มที่ req_symbol ถ้าจำนวนสัญลักษณ์ที่ผิดพลาดที่ตรวจพบมีค่าเท่ากับค่าอันดับของซินโดรมและมีไม่เกิน 4 สัญลักษณ์ จะสามารถถอดรหัสได้ โดยจะต้องทำการหาค่าความผิดพลาดแล้วแก้ไขความผิดพลาด ซึ่งจะเริ่มที่สเตต find_Hsub

สเตต find_Hsub เป็นการหาค่า H_{sub} จากนั้นจึงนำไปหาค่าเมทริกซ์ผกผันที่ find_Hsub_inverse อีก 6 สเตตถัดมาได้แก่ product_error_value, sum_error_value, correct_with_null1, correct_with_null2, next_error_value_row และ wait_correct_with_null เป็นการคูณเมทริกซ์ทีละแถว เพื่อหาค่าความผิดพลาดทีละตัว แล้วนำไปแก้ไขให้ข้อมูลตัวเลือกอันดับหนึ่งถูกต้อง

เมื่อแก้ไขครบทุกตัวแล้วจะเป็นการหาค่าซินโดรมของข้อมูลที่ได้แก้ไขด้วยวิธีการรวมทางพีชคณิตได้ศูนย์แล้วด้วยสเตต update_syndrome2 และ wait_update_syndrome2 ส่วนสเตตสุดท้ายเป็นการตรวจสอบค่าซินโดรมหลังการถอดรหัส ถ้าเป็นศูนย์ตัวถอดรหัสจะถือว่าการถอดรหัสถูกต้องแล้วส่งข้อมูลที่ถอดรหัสแล้วกลับสู่คอมพิวเตอร์ ถ้าไม่เป็นศูนย์จะถือว่าการถอดรหัสล้มเหลวแล้วรายงานไปยังสเตต decoding_failure โดยขั้นตอนทั้งหมดแสดงไว้ในรูปที่ 35



ภาพที่ 35 แผนภาพสเตตการแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์

4. การเขียนโปรแกรมและจำลองการทำงาน สำหรับตัวถอดรหัส

การเขียนโปรแกรมสำหรับชิป FPGA ในงานวิจัยนี้ ได้เลือกใช้ภาษา VHDL และเลือกใช้โปรแกรม ModelSim 6.0 ในการตรวจสอบความถูกต้องของภาษาที่เขียนและใช้ในการจำลองการทำงาน ส่วนการสังเคราะห์วงจร (Synthesis) และการสร้างโปรแกรมสำหรับดาวน์โหลดลงบอร์ด FPGA ได้ใช้โปรแกรม Xilinx ISE 10.1

การเขียนภาษา VHDL ต้องกำหนด Entity ซึ่งเป็นส่วนหัวของโปรแกรมก่อน ตามรูปที่ 36 โดยได้กำหนดสัญญาณต่างๆ อิงจากตารางที่ 1 โดยมีสัญญาณขาเข้า 4 สัญญาณ สัญญาณขาออก 1 สัญญาณ และสัญญาณผสมทั้งขาเข้าและขาออก 3 สัญญาณ ดังนี้ สัญญาณขาเข้า 4 สัญญาณ ได้แก่ สัญญาณ RST และ CLK เป็นสัญญาณ User clock input และ Push switch SW4 ที่รับมาจากบอร์ด FPGA ทำหน้าเป็นสัญญาณ Reset และสัญญาณนาฬิกา ส่วนสัญญาณ TXE และ RXF เป็นสัญญาณด้านรับมาจาก USB-module ในขาที่ 15 และ 14 ซึ่งตรงกับขาสัญญาณบอร์ด FGPA ขาที่ 78 และ 80 โดย TXE และ RXF ทำหน้าที่เป็นสัญญาณ Enable สำหรับด้านส่ง TX และด้านรับ RX ถัดมาเป็นสัญญาณขาออก 1 สัญญาณ คือ สัญญาณ ALM เป็นสัญญาณส่งออกไปยัง LED0 บนบอร์ด FPGA ทำหน้าที่ในการแสดงผลการถอดรหัสล้มเหลว (Decoding failure) นอกจากนี้ยังมีสัญญาณผสมทั้งขาเข้าและขาออก 3 สัญญาณ คือ สัญญาณ WR และ RD เป็นขาสัญญาณที่ใช้รับส่งข้อมูลบ่งบอกสถานะพร้อมที่จะรับส่งข้อมูลของ USB-module โดยตรงกับขาสัญญาณ USB-module ในขาที่ 16 และ 17 แล้วยังตรงกับขาสัญญาณบนบอร์ด FPGA ขาที่ 74 และ 72 ส่วนสัญญาณตัวสุดท้ายเป็นสัญญาณ D เป็นสัญญาณแบบบัสเอาไว้รับส่งข้อมูล มี 8 ขาสัญญาณ คือ D7 ถึง D0 ใน USB-module เป็นขาที่ 18 ถึง 25 ซึ่งจะตรงกับขาที่ 68, 66, 62, …, 48 ของบอร์ด FPGA

```
entity VSD is
Port ( RST : in std_logic;       --RESET
       CLK : in std_logic;       --CLOCK
       TXE : in std_logic;       --TXE
       RXF : in std_logic;       --RXF
       ALM : out std_logic;      --ALARM
       WR : inout std_logic;     --WR
       RD : inout std_logic;     --RD
       D : inout std_logic_vector(0 to 7)); --Data bus
end VSD;
```

ภาพที่ 36 ขาสัญญาณที่กำหนดบนภาษา VHDL

จากนั้นจึงเริ่มเขียนโปรแกรมในส่วนที่เป็นเนื้อหาของโปรแกรมต่อไป ซึ่งโครงสร้างภายในประกอบไปด้วย function และ process ย่อย ที่จะทำหน้าที่ต่างกันไป แต่ทั้งหมดนี้ จะมี process เดียวเท่านั้นที่ทำหน้าที่ในการควบคุมการไหลของสเตตทั้งหมด ซึ่งจะเป็นการเชื่อมโยงความสัมพันธ์ของสเตตต่างๆ โดยมีตัวอย่างดังเช่นรูปที่ 37 อธิบายได้ว่า สเตตเริ่มต้นเป็น read เมื่ออยู่ในสเตต read แล้ว จะทำการตัดสินจากเงื่อนไขของ RXF_LCH ว่า จะก้าวไปสเตต usb_read เมื่อ RXF_LCH เป็นศูนย์ ถ้าไปสเตต usb_read แล้วสเตตถัดไปคือ wait_read เป็นต้น หลังจากเขียนโปรแกรมภาษา VHDL เสร็จแล้วจึงทำการทดสอบ และหาช่องโหว่ของโปรแกรม โดยได้ทำการจำลองการทำงานด้วยการใช้ ModelSim

```
state_pr : process(RST,CLK)
begin
    if RST = '1' then    --Reset
        state <= read;   --Initial state
    elsif CLK'event and CLK = '1' then
        case state is
            when read =>
                if RXF_LCH = '0' then    --receive enable
                    state <= usb_read;   --goto state usb_read
                else
                    state <= read;       --wait receive enable signal
                end if;
            when usb_read => state <= wait_read;     --goto state wait_read
            when wait_read => state <= next_state_name;
            when others => state <= read;
        end case;
    end if;
end process state_pr;
```

ภาพที่ 37 ตัวอย่างการกำหนดการทำงานของสเตตด้วยภาษา VHDL

เมื่อจำลองการทำงานเสร็จสิ้น จึงทำการกำหนดขาสัญญาณของชิป FPGA ที่จะใช้งานให้กับโปรแกรมที่เขียนขึ้นมาโดยกำหนดลงใน User constraint file ในรูปที่ 38 โดยอ้างอิงจากตารางที่ 1 ซึ่งจะเป็นการระบุให้ขั้นตอนการสร้างโปรแกรมไว้สำหรับดาวน์โหลดทราบว่า จะต้องเชื่อมโยงขาสัญญาณจากโปรแกรมที่เขียนไว้ขาใด เข้ากับขาสัญญาณขาใดของชิป FPGA ซึ่งขาของชิป FPGA แต่ละขานั้นมีการเชื่อมโยงไปหาอุปกรณ์ต่างๆกันบนบอร์ด FPGA โดยสามารถดูรายละเอียดได้จากภาคผนวก ก หลังจากระบุขาของชิป FPGA ที่ใช้ได้แล้วจึงทำการสังเคราะห์วงจรแล้วสร้างโปรแกรมสำหรับดาวน์โหลดลงบอร์ด FPGA ด้วย Xilinx ISE ต่อไป

```
NET "CLK"  LOC = "e16";
NET "D<0>"  LOC = "r3" ;
NET "D<1>"  LOC = "r2" ;
NET "D<2>"  LOC = "e2" ;
NET "D<3>"  LOC = "e1" ;
NET "D<4>"  LOC = "f5" ;
NET "D<5>"  LOC = "f4" ;
NET "D<6>"  LOC = "f3" ;
NET "D<7>"  LOC = "e3" ;
NET "RD"    LOC = "y1" ;
NET "RST"   LOC = "f17" ;
NET "RXF"   LOC = "h1" ;
NET "TXE"   LOC = "g1" ;
NET "WR"    LOC = "w1" ;
NET "ALM"   LOC = "e11";
```

ภาพที่ 38 ขาสัญญาณที่กำหนดบน User constraint file

5. การทดสอบการทำงานตัวถอดรหัสบน FPGA

หลังจากที่เตรียมอุปกรณ์ และเขียนโปรแกรมในทุกส่วนแล้ว ก็จะเป็นการทดสอบการทำงานของตัวถอดรหัส ในฝั่งคอมพิวเตอร์จะทำการส่งไฟล์ที่มีข้อมูลของตัวเลือกทั้งสอง โดยใช้โปรแกรม USB Transceiver ส่วนฝั่งของบอร์ด FPGA จะต้องทำการดาวน์โหลดโปรแกรมตัวถอดรหัสลงชิป FPGA ก่อน จากนั้นจึงทำการส่งไฟล์รหัสแต่ละตัวอย่างไปถอดรหัสแล้ว ทำการบันทึกผลต่อไป

5.1 การเตรียมไฟล์ที่จะนำไปถอดรหัส

ข้อมูลที่จะนำไปถอดรหัสนั้น ได้แบ่งออกเป็น 2 ไฟล์ คือ ไฟล์ของตัวเลือกอันดับหนึ่งหากข้อมูลไม่มีค่าความผิดพลาดจะเป็นข้อมูลของคำรหัส ส่วนไฟล์ที่สองเป็นข้อมูลของสัญญาณรบกวนที่สุ่มขึ้นมา ในการสร้างไฟล์ของคำรหัสได้ใช้โปรแกรม File Encoding ดังรูปที่ 39 ที่เขียนขึ้นมา เพื่อใช้สร้างคำรหัสบนคอมพิวเตอร์ โดยได้นำไฟล์ที่สร้างขึ้นขนาด 96 ไบท์ ในรูปที่ 40 มาเข้ารหัสได้เป็นไฟล์ที่ได้เข้ารหัสข้อมูลด้วยรหัสคอนโวลูชัน (3, 2, 2) รหัสหนึ่งที่มีขนาด 144 ไบท์ ดังรูปที่ 41 ซึ่งไฟล์คำรหัสที่ได้นั้น ได้นำไปทดสอบเทียบกับผลที่ได้จาก รัชนนท์ (2551) แล้วสามารถเข้ารหัสได้เหมือนกัน

ภาพที่ 39 โปรแกรมเข้ารหัสคอนโวลูชัน (3, 2, 2) ให้กับไฟล์คอมพิวเตอร์

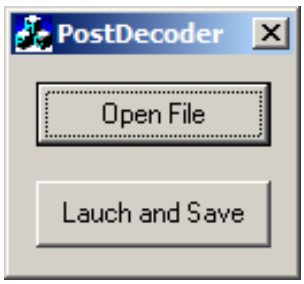
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000	00	00	00	01	00	01	00	00	00	00	00	02	00	02	00	00
00000010	00	00	00	04	00	04	00	00	00	00	00	08	00	08	00	00
00000020	00	00	00	10	00	10	00	00	00	00	00	20	00	20	00	00
00000030	00	00	00	40	00	40	00	00	00	00	00	80	00	80	00	00
00000040	00	00	01	00	01	00	00	00	00	00	02	00	02	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

ภาพที่ 40 ไฟล์คอมพิวเตอร์ต้นฉบับขนาด 96 ไบท์

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000	00	00	00	01	00	01	00	00	00	01	00	01	00	01	00	03
00000010	00	02	00	00	00	03	00	02	00	02	00	07	00	05	00	01
00000020	00	07	00	04	00	04	00	0e	00	0a	00	02	00	0e	00	08
00000030	00	08	00	1c	00	14	00	04	00	1c	00	10	00	10	00	38
00000040	00	28	00	08	00	38	00	20	00	20	00	70	00	50	00	10
00000050	00	70	00	40	00	40	00	e0	00	a0	00	20	00	e0	00	80
00000060	00	80	01	c0	01	40	00	40	01	c0	01	00	01	00	03	80
00000070	02	80	00	80	03	80	02	00	02	00	03	00	01	00	01	00
00000080	03	00	00	00	00	00	02	00	02	00	02	00	02	00	00	00

ภาพที่ 41 ไฟล์คอมพิวเตอร์ที่เข้ารหัสเข้ารหัสคอนโวลูชัน (3, 2, 2) เป็นคำรหัสขนาด 144 ไบท์

นอกจากนี้ยังได้ทำการทดสอบโปรแกรมดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ด้วยโปรแกรม Post Decoder ในที่รูปที่ 42 ที่ได้สร้างขึ้นมาเป็นการดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับบนคอมพิวเตอร์ โดยใช้คำรหัสในรูปที่ 41 มาดึงรหัสคอนโวลูชันแบบไม่เป็นระบบกลับคืนข้อมูลต้นฉบับ ทำให้ได้เป็นข้อมูลก่อนเข้ารหัสดังรูปที่ 40



ภาพที่ 42 โปรแกรมดึงรหัสคอนโวลูชันแบบไม่เป็นระบบ (3, 2, 2) กลับคืนข้อมูลต้นฉบับ ให้กับไฟล์คอมพิวเตอร์

5.2 การดาวน์โหลดโปรแกรมลงชิป FPGA

การดาวน์โหลดโปรแกรมลงชิป FPGA ทำได้โดยใช้เครื่องดาวน์โหลดโปรแกรม Xilinx Platform Cable USB นำมาต่อกับบอร์ด FPGA กับคอมพิวเตอร์ ตามรูปที่ 15 โดยใช้โปรแกรม iMPACT ที่มาในชุดของโปรแกรม Xilinx ISE ซึ่งโปรแกรมนี้ จะทำการดาวน์โหลดโปรแกรมลงชิป FPGA โดยโปรแกรมที่จะนำไปดาวน์โหลดได้จากขั้นตอนการ Implementation ของ Xilinx ISE

5.3 การส่งไฟล์ที่จะนำไปถอดรหัส และบันทึกผล

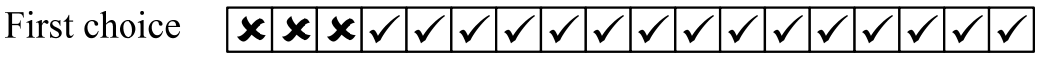
การส่งไฟล์จากคอมพิวเตอร์เข้าไปถอดรหัสยังบอร์ด FPGA ได้ใช้โปรแกรม USB Transceiver เป็นตัวควบคุมการรับส่งข้อมูล ในงานวิจัยนี้ข้อมูลที่ส่งเป็นการนำข้อมูลคำรหัสในรูปที่ 41 มาทำการเพิ่มสัญญาณรบกวนลงไปด้วยการสุ่ม ซึ่งกำหนดตามเงื่อนไขที่ตัวถอดรหัสสามารถทำการถอดรหัสได้ตามทฤษฎี จากนั้นจึงส่งข้อมูลที่ต้องการทดสอบลงไป แล้วทำการบันทึกผล แล้วจึงทำการเปลี่ยนสัญญาณรบกวนเป็นรูปแบบอื่นๆที่ตัวถอดรหัสสามารถถอดรหัสได้

ผลและวิจารณ์

ผล

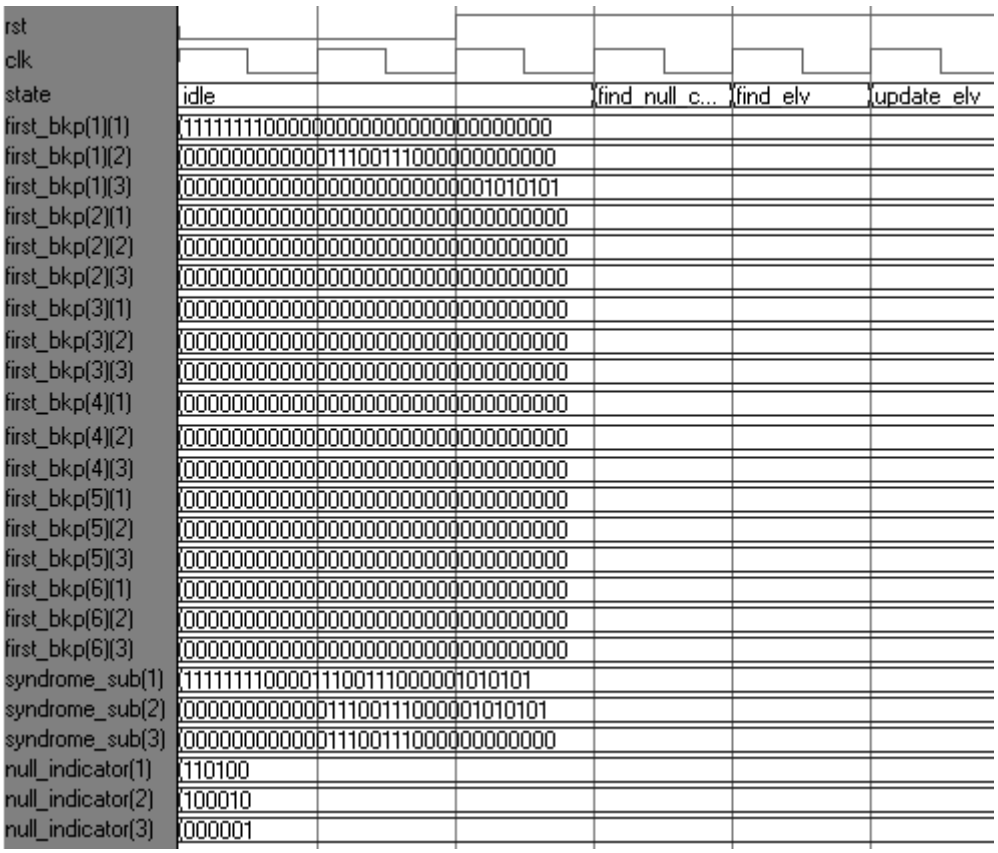
1. ผลการจำลองการทำงาน (Simulation) บนแผนภาพลูกคลื่นสัญญาณกับแกนเวลา

ในขั้นตอนการออกแบบตัวถอดรหัสนั้น ได้แบ่งขั้นตอนการถอดรหัสเป็นขั้นตอนย่อย ซึ่งแต่ละขั้นตอนย่อยได้มีการเขียนโปรแกรมด้วย VHDL แล้วทำการทดสอบการทำงาน ด้วยการจำลองการทำงานบนโปรแกรม ModelSim 6.0 โดยทำการป้อนลูกคลื่นสัญญาณที่ต้องการทดสอบลงไป แต่เนื่องด้วยขั้นตอนย่อยเหล่านี้มีเป็นจำนวนมาก ดังนั้นจึงได้นำผลการจำลองการทำงานที่ควรให้ความสนใจมาแสดงเท่านั้น โดยได้นำขั้นตอนย่อยบางส่วน ที่อยู่ในขั้นตอนการแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ (Correct with null combination) มาแสดงผล เนื่องจากเป็นแก่นของตัวถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง (Vector symbol decoding) โดยผลการจำลองการทำงานนี้ได้ทำการถอดรหัสของข้อมูลที่มีตัวเลือกเพียงตัวเดียว โดยกำหนดให้คำรหัสที่ถูกต้องเป็นศูนย์ทุกตัว และกำหนดให้มีสัญลักษณ์ที่ผิดติดกันอยู่ใน 3 สัญลักษณ์แรก ดังรูปที่ 43



ภาพที่ 43 แสดงรูปแบบของข้อมูลที่ใช้จำลองการทำงาน แต่ละช่องแทนสัญลักษณ์ขนาด 32 บิต

จากรูปที่ 44 เป็นค่าสัญญาณที่ได้ป้อนเข้าไป ประกอบด้วยสัญญาณ 6 ตัว ได้แก่ rst, clk, state, first_bkp, syndrome_sub และ null_indicatorโดยได้อธิบายความหมายของแต่ละสัญญาณไว้ในตารางที่ 2 สามารถอธิบายการทำงานได้ดังนี้ สัญญาณเวลานาฬิกา (Clock) สามคาบแรก มีสเตต (State) อยู่ที่สถานะ idle แต่ในคาบที่สามสัญญาณ RST ได้เปลี่ยนจาก Low เป็น High ทำให้คาบที่ 4 นั้น สเตตเปลี่ยนสถานะเป็น Find null combination ซึ่งก็คือตัวถอดรหัสเริ่มทำงานหลังจากสัญญาณ RST เปลี่ยนเป็น High แต่นี่เป็นเพียงการจำลองการทำงานเท่านั้น ถ้าจะนำไปใช้งานบน FPGA Virtex5 รุ่น XC5VLX110 จะต้องเปลี่ยน RST จาก Active high เป็น Active low และในส่วนของสัญญาณอีกสามตัวเป็นค่าตั้งต้นที่จะนำไปถอดรหัสที่ได้ทำการป้อนลงไป



ภาพที่ 44 ผลการจำลองการทำงาน แสดงค่าตั้งต้นที่ทำการป้อนลงไป

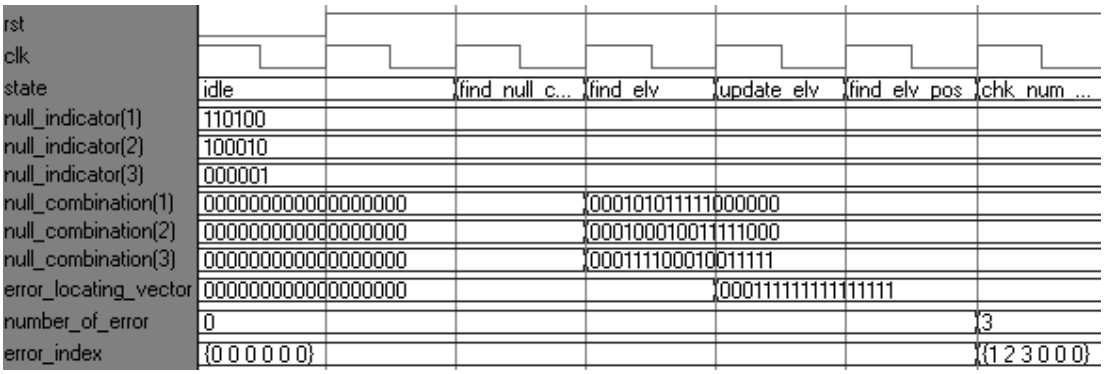
ตารางที่ 2 ชื่อและคำอธิบายของสัญญาณที่ใช้ในการจำลองการทำงาน

สัญญาณ	คำอธิบาย
rst	Reset
clk	สัญญาณนาฬิกา Clock
state	ชี้ค่าสเตต (State) การทำงาน
first_bkp	บัส (Bus) ของข้อมูลตัวเลือกอันดับหนึ่ง ความยาว 18 สัญลักษณ์
syndrome_sub	บัสของซินโดรมลดรูป S_{sub} มี 3 ตัว
null_indicator	บัสของตัวบ่งชี้ศูนย์ (Null indicator) มี 3 ตัว แต่ละตัวยาว 6 บิต
null_combination	บัสของผลรวมทางพีชคณิตได้ศูนย์ (Null combination) 3 ตัวตัวละ 18 บิต
error_locating_vector	เวกเตอร์ระบุตำแหน่งความผิดพลาด (ELV) ยาว 18 บิต
number_of_error	จำนวนความผิดพลาดที่ตัวถอดรหัสตรวจพบ

ตารางที่ 2 (ต่อ)

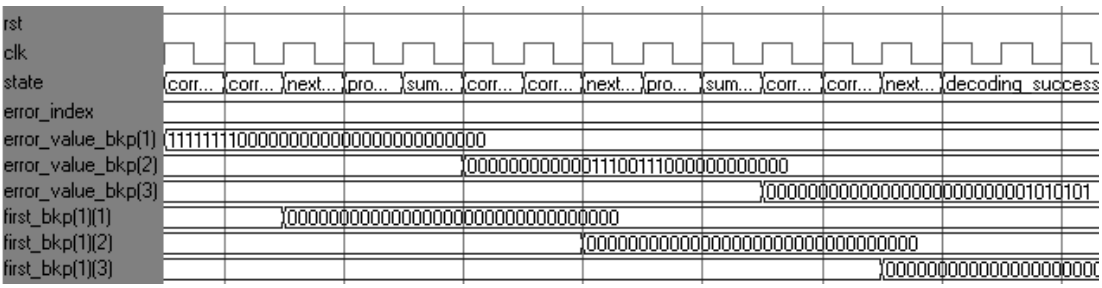
สัญญาณ	คำอธิบาย
error_index	ค่าระบุตำแหน่งความผิดพลาด (Error index) ที่ตัวถอดรหัสหาได้
error_value_bkp	ค่าความผิดพลาด (Error value) ที่ตัวถอดรหัสคำนวณได้

จากรูปที่ 45 เป็นผลการจำลองการทำงาน เพื่อหาค่าระบุตำแหน่งความผิดพลาด (Error index) ซึ่งอธิบายได้ดังนี้ ตัวถอดรหัสจะนำตัวระบุตำแหน่งศูนย์ มาใช้หาผลรวมทางพีชคณิตได้ศูนย์ที่สเตต Find null combination จากนั้นที่สเตต Find ELV และ Update ELV ก็จะทำการหา ELV จากการ OR กันของผลรวมทางพีชคณิตได้ศูนย์ที่หาได้ทั้ง 3 ตัว เมื่อได้ ELV แล้วตัวถอดรหัสจะทำการนับจำนวนค่าความผิดพลาด และค่าระบุตำแหน่งความผิดพลาดได้ที่สเตต Find ELV pos

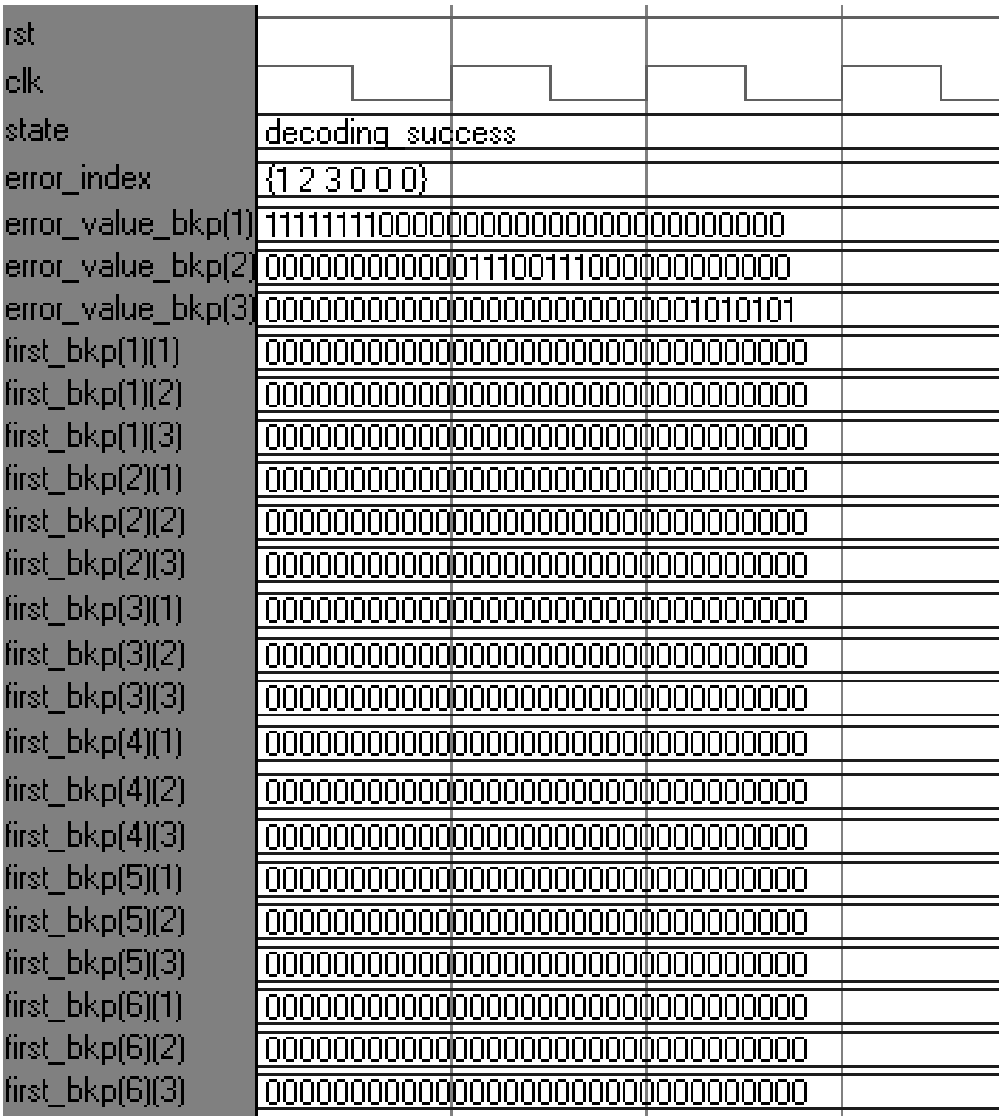


ภาพที่ 45 ผลการจำลองการทำงาน แสดงค่าระบุตำแหน่งความผิดพลาด (Error index) ที่หาได้

ส่วนรูปที่ 46 เป็นการหาค่าความผิดพลาด (Error value) โดยใช้ซินโดรมลดรูป S_{sub} ที่กำหนดไว้ในรูปที่ 44 และค่าระบุตำแหน่งความผิดพลาดในรูป 36 พร้อมด้วยค่าเมทริกซ์ตรวจสอบภาวะคู่หรือคี่ลดรูป H_{sub} มาทำการคำนวณหา ซึ่งการคำนวณจะทำการหาค่าความผิดพลาดครั้งหนึ่งสัญลักษณ์ แล้วนำไปแก้ไขให้กับตัวเลือกอันดับหนึ่งก่อน จึงค่อยคำนวณหาค่าความผิดพลาดอีกหนึ่งสัญลักษณ์ตัวถัดไป แล้วนำไปแก้ไข ทำเช่นนี้จนแก้ไขได้ครบตามจำนวนที่หาได้เป็นอันเสร็จกระบวนการถอดรหัส ซึ่งจะพบว่าค่าของตัวเลือกอันดับหนึ่งถูกแก้ไขให้เป็นศูนย์หมด โดยแสดงไว้ในรูปที่ 47 และคำรหัสที่ถูกต้องที่กำหนดไว้นั้น เป็นศูนย์หมด ดังนั้นการจำลองการถอดรหัสทำได้ถูกต้องตรงกับที่ได้กำหนดไว้



ภาพที่ 46 ผลการจำลองการทำงาน แสดงค่าความผิดพลาด (Error value) ที่ตัวถอดรหัสคำนวณได้



ภาพที่ 47 ผลการจำลองการทำงาน ผลลัพธ์การถอดรหัสตัวอย่างรหัสหนึ่ง

2. ผลการสังเคราะห์วงจร (Synthesis) สำหรับตัวถอดรหัส

ชิป FPGA Virtex5 รุ่น XC5VLX110 มีขนาดของ CLBs เท่ากับ 8640 เซลล์ ประกอบไปด้วย Flip-flops 69120 ตัว และ 6-input LUT 69120 ชุด หรือคิดเป็น 69120 x 2^6 = 4423680 โลจิกเกท จากผลการสังเคราะห์วงจร ทำให้ทราบค่าประมาณการใช้ทรัพยากรบนชิป FPGA ดังรูปที่ 48 โดย จำนวน Flip-flop ใช้ไป 5973 ตัว จากทั้งหมด 69120 ตัว หรือคิดเป็น 8% และจำนวน LUT ใช้ไป 13629 ตัว จากทั้งหมด 69120 ตัว หรือคิดเป็น 19% เนื่องจากแต่ละเซลล์ของ CLBs นั้นประกอบไปด้วย Flip-flop และ LUT ดังนั้นเมื่อรวมทั้งสองเข้าด้วยกันเป็น CLBs แล้วจะมีการใช้งานรวมเป็น 15713 ชุด จากทั้งหมด 69120 ชุด ซึ่งคิดเป็น 22.7% หรือ 1005632 โลจิกเกท ประกอบด้วย ใช้งานทั้ง Flip-flop และ LUT 3889 ชุด หรือ 24% ใช้งานเฉพาะ LUT 9740 ชุด หรือ 61% ใช้งานเฉพาะ Flip-flop 2084 ชุด หรือ 13% มีการใช้งานขาสัญญาณรับส่งข้อมูลของบอร์ดทดลอง FPGA ใช้ไป 15 ขา จากทั้งหมด 440 ขาสัญญาณ คิดเป็น 3% ประกอบด้วยขาสัญญาณควบคุม 7 ขา และขาสัญญาณข้อมูล 8 ขา

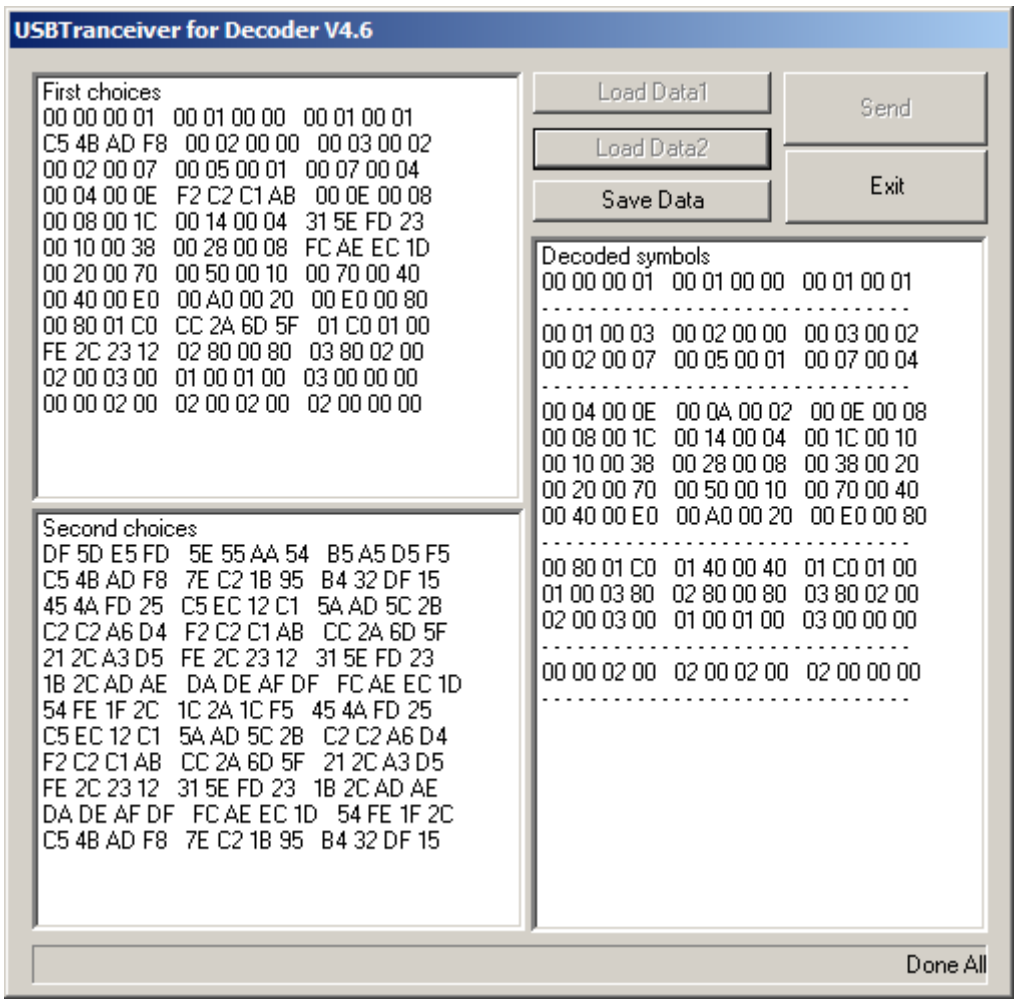
```
Device utilization summary:
---------------------------

Selected Device : 5vlx110ff676-1


Slice Logic Utilization:
 Number of Slice Registers:            5973  out of  69120     8%
 Number of Slice LUTs:                13629  out of  69120    19%
    Number used as Logic:             13629  out of  69120    19%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  15713
   Number with an unused Flip Flop:    9740  out of  15713    61%
   Number with an unused LUT:          2084  out of  15713    13%
   Number of fully used LUT-FF pairs:  3889  out of  15713    24%
   Number of unique control sets:        93

IO Utilization:
 Number of IOs:                          15
 Number of bonded IOBs:                  15  out of    440     3%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                2  out of     32     6%
```

ภาพที่ 48 รายงานการสังเคราะห์วงจร ด้วยโปรแกรม Xilinx ISE 10.1 โดยนำมาแสดงเป็นบางส่วน

3. วิธีการตีความผลการถอดรหัส

จากรูปที่ 49 เป็นโปรแกรม “USB Transceiver for Decoder V4.6” ที่ใช้ในการรับส่งข้อมูลกับ FPGA พร้อมทั้งมีการแสดงผลค่าที่ถูกส่งเข้า FPGA และค่าที่รับมาจาก FPGA ซึ่งประกอบไปด้วยหน้าต่างย่อย 3 ช่อง ได้แก่ หน้าต่าง “First choices” แสดงข้อมูลตัวเลือกอันดับหนึ่งที่ส่งเข้า FPGA หน้าต่าง “Second choices” แสดงข้อมูลตัวเลือกอันดับสองที่ส่งเข้า FPGA หน้าต่าง “Decoded symbols” แสดงข้อมูลสัญลักษณ์ที่ถอดรหัสเสร็จแล้วซึ่งส่งมาจาก FPGA โดยแต่ละหน้าต่างจะอ่านค่าจากซ้ายไปขวา แล้วอ่านบนลงล่าง ในแต่ละแถวจะแสดงผลข้อมูล 3 กลุ่ม แต่ละกลุ่มจะแทนค่าด้วยข้อมูล 1 สัญลักษณ์ แต่ละสัญลักษณ์มีขนาด 4 ไบท์ หรือ 32 บิต ซึ่งก็คือ ในหนึ่งแถวจะยาวเท่ากับ 3 สัญลักษณ์ แต่ละหน้าต่างมีข้อมูล 12 แถว หรือหมายถึงเมื่อถอดรหัสแล้วจะได้สัญลักษณ์ 36 สัญลักษณ์นั่นเอง ส่วนในหน้าต่าง “Decoded symbols” จะมีเส้นประ เป็นตัวเว้นวรรคการถอดรหัสในแต่ละครั้ง ซึ่งจะบ่งบอกว่า ในการถอดรหัสแต่ละครั้งได้ใช้ซินโดรมไปเป็นจำนวนเท่าใด เช่น ถ้ามีข้อมูลติดกันห้าแถว แล้วจึงมีเส้นประคั่น จะอ่านได้ว่า ได้ใช้ซินโดรมไปห้าค่าในการถอดรหัสสำหรับครั้งนั้น เป็นต้น

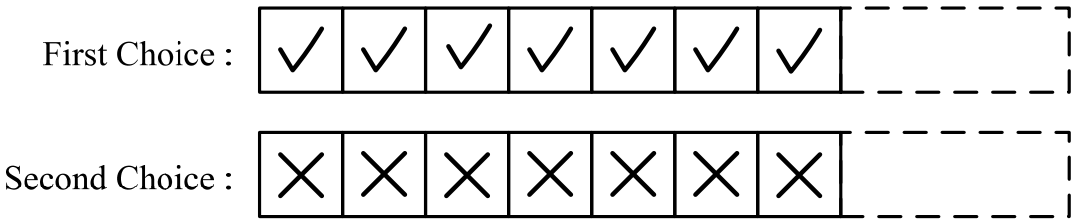
การอ่านค่าผลการถอดรหัสของรูปที่ 49 มีเส้นประ 5 เส้น หมายถึงตัวถอดรหัสได้ทำการถอดรหัส 5 ครั้ง ซึ่งจะอ่านได้ว่า ในครั้งแรกทำการป้อนตัวเลือกอันดับหนึ่งเป็น “00000001 00010000 00010001” และป้อนตัวเลือกอันดับสองเป็น “DF5DE5FD 5E55AA54 B5A5D5F5” ผลการถอดรหัสครั้งแรกใช้ซินโดรมไปหนึ่งตัว โดยได้ผลลัพธ์เป็น “00000001 00010000 00010001” ในครั้งที่สองทำการป้อนตัวเลือกอันดับหนึ่งเป็น “C54BADF8 00020000 00030002” และป้อนตัวเลือกอันดับสองเป็น “C54BADF8 7EC21B95 B432DF15” ผลการถอดรหัสครั้งที่สองใช้ซินโดรมไปสองตัว หมายความว่า ได้ทำการป้อนตัวเลือกอันดับหนึ่งและสอง เพิ่มเข้าไปอีกอย่างละแถว ซึ่งก็คือ “00020007 00050001 00070004” กับ “454AFD25 C5EC12C1 5AAD5C2B” แล้วจึงทำให้ได้ผลลัพธ์เป็น “00010003 00020000 00030002” กับ “00020007 00050001 0007 0004” ในครั้งที่สามที่หน้าต่าง “Decoded symbols” มีห้าแถวที่ติดกัน หมายถึง ในการถอดรหัสครั้งที่สามใช้ไปห้าซินโดรม ซึ่งก็คือได้ใช้ตัวเลือกอันดับหนึ่งและสอง ในแถวที่ 4 ถึง 8 หรืออย่างละ 5 แถว จึงจะทำให้ได้ผลลัพธ์ออกมา 5 แถว ในครั้งที่สี่ได้ผลลัพธ์ 3 แถว ซึ่งได้จากถอดรหัสตัวเลือกอันดับหนึ่งและสอง ในแถวที่ 9 ถึง 11 ในครั้งที่ห้าทำการถอดรหัสตัวเลือกอันดับหนึ่งและสองแถวที่ 12 ได้ผลลัพธ์เป็นแถวสุดท้ายในช่อง “Decoded symbols”



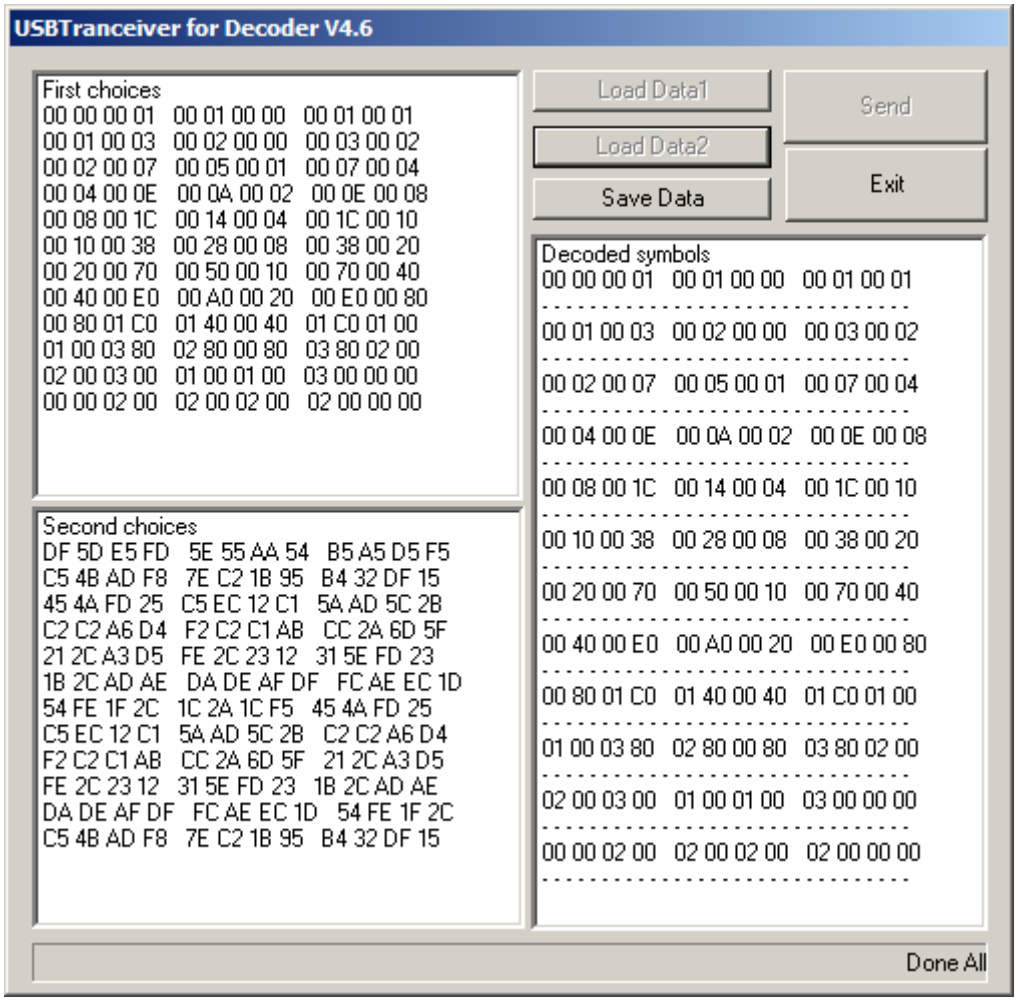
ภาพที่ 49 ตัวอย่างประกอบ วิธีการอ่านผลการถอดรหัส

4. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสเป็นข้อมูลที่ถูกต้องทั้งหมด

ในการถอดรหัสข้อมูลที่ถูกต้องทั้งหมด หมายถึง ทำการถอดรหัสข้อมูลตัวเลือกอันดับหนึ่งที่มีค่าเหมือนกับคำรหัสทุกประการ และข้อมูลตัวเลือกที่ไม่ใช่อันดับหนึ่งจะมีข้อมูลที่ผิดทั้งหมด ซึ่งจะมีรูปแบบดังรูปที่ 50 และผลการถอดรหัสแสดงไว้ในรูปที่ 51 ซึ่งทำการถอดรหัสทั้งหมด 12 ครั้ง แต่ละครั้งใช้เพียงหนึ่งซินโดรม

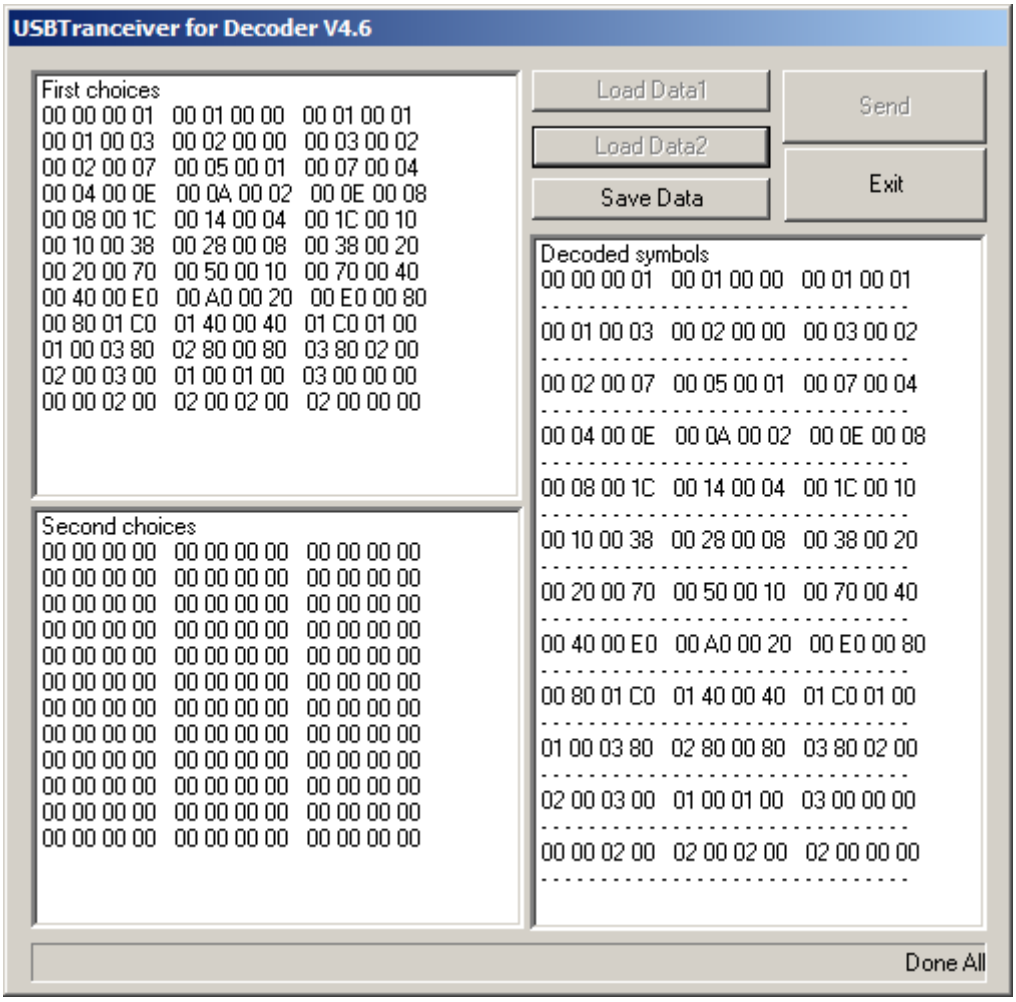


ภาพที่ 50 แสดงรูปแบบของข้อมูลในกรณีที่ข้อมูลที่นำมาถอดรหัสเป็นข้อมูลที่ถูกต้องทั้งหมด โดยแต่ละช่องแทน สัญลักษณ์ขนาด 32 บิต



ภาพที่ 51 ผลการถอดรหัส เมื่อข้อมูลที่นำมาถอดรหัสเป็นข้อมูลที่ถูกต้องทั้งหมด โดยที่ตัวเลือกอันดับสองเป็นข้อมูลที่ผิดทั้งหมด

กรณีที่มีเฉพาะตัวเลือกอันดับหนึ่ง แต่ไม่มีตัวเลือกอันดับสอง ในงานวิจัยนี้จะแทนค่าตัวเลือกอันดับสองด้วยศูนย์ทั้งหมด ซึ่งกรณีนี้ตัวถอดรหัสจะถอดรหัสได้เหมือนกับกรณีตัวเลือกมีสองตัว ซึ่งได้แสดงผลการถอดรหัสไว้ในรูปที่ 52



ภาพที่ 52 ผลการถอดรหัส เมื่อข้อมูลที่นำมาถอดรหัสเป็นข้อมูลที่ถูกต้องทั้งหมด และมีตัวเลือกเดียว

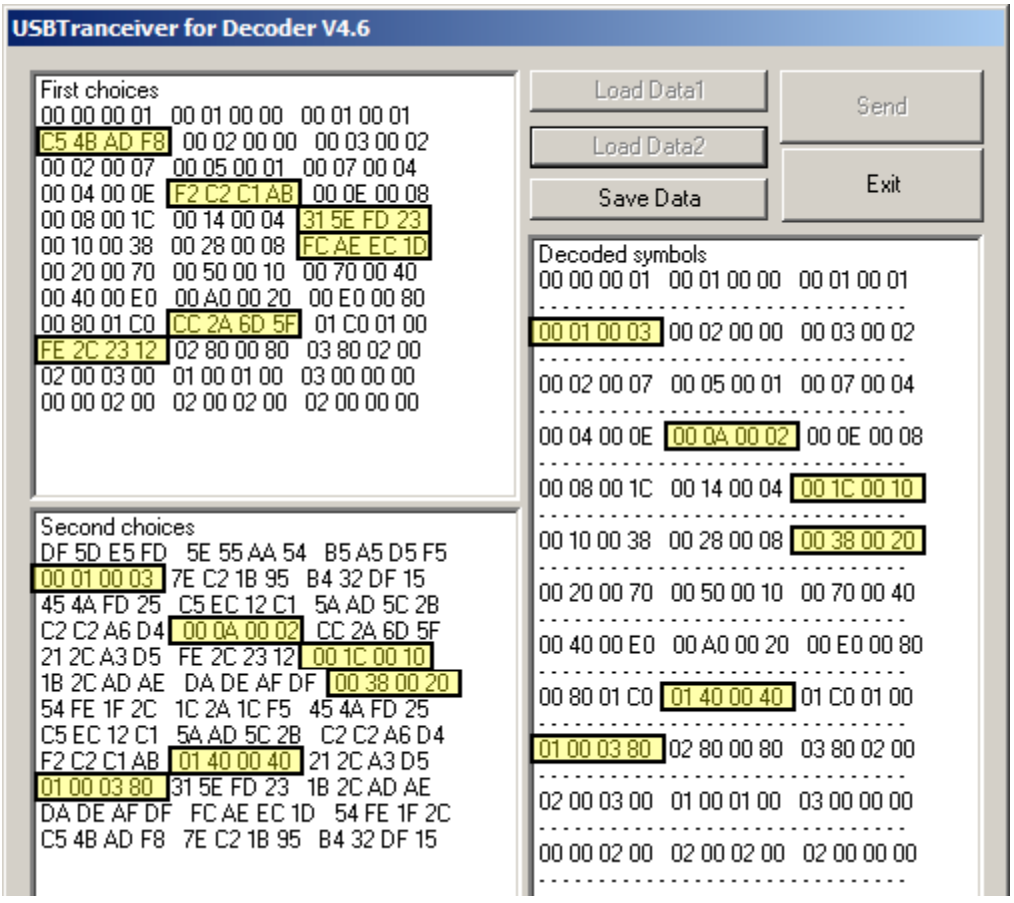
5. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสมี 2 ตัวเลือก

เมื่อข้อมูลที่นำมาถอดรหัสมี 2 ตัวเลือก หากตรวจเจอค่าความผิดพลาดตัวถอดรหัส จะทำการถอดรหัสด้วยซินโดรมเดียวในขั้นแรก ถ้าไม่สามารถถอดรหัสด้วยซินโดรมเดียวได้ จะทำการเพิ่มข้อมูลตัวเลือกทั้งสองอย่างละ 3 สัญลักษณ์ แล้วจึงแก้ไขข้อมูลที่ผิดพลาดด้วยวิธีถอดรหัสด้วย

ซินโดรมหลายตัว ซึ่งจะแบ่งเป็นขั้นตอนแก้ไขด้วยตัวเลือกอันดับสอง และขั้นตอนแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ โดยอาจจะใช้วิธีใดวิธีหนึ่ง หรืออาจจะใช้ทั้งสองวิธีก็ได้ หากแก้ไขแล้วแต่ยังมีค่าความผิดพลาดเหลืออยู่และซินโดรมขนาดยังไม่เกิน 10 ตัว ตัวถอดรหัสจะเพิ่มข้อมูลตัวเลือกทั้งสองอย่างละ 3 สัญลักษณ์ แก้ไขด้วยวิธีถอดรหัสด้วยซินโดรมหลายตัวอีกครั้ง แต่ถ้าแก้ไขแล้วแต่ยังมีค่าความผิดพลาดเหลืออยู่และซินโดรมขนาดเกิน 10 ตัวแล้ว การถอดรหัสจะไม่สำเร็จ

5.1 ถอดรหัสด้วยซินโดรมเดียว

การถอดรหัสด้วยซินโดรมเดียว จะเป็นการแก้ไขค่าความผิดพลาดด้วย การหาข้อมูลในตัวเลือกอันดับสองที่ถูกต้องมาแทนที่ข้อมูลตัวเลือกอันดับหนึ่งที่มีความผิดพลาด เพื่อให้ตำแหน่งที่มีความผิดพลาดนี้สังเกตได้ง่าย จึงล้อมกรอบไว้ด้วยเส้นทึบดังในรูปที่ 53



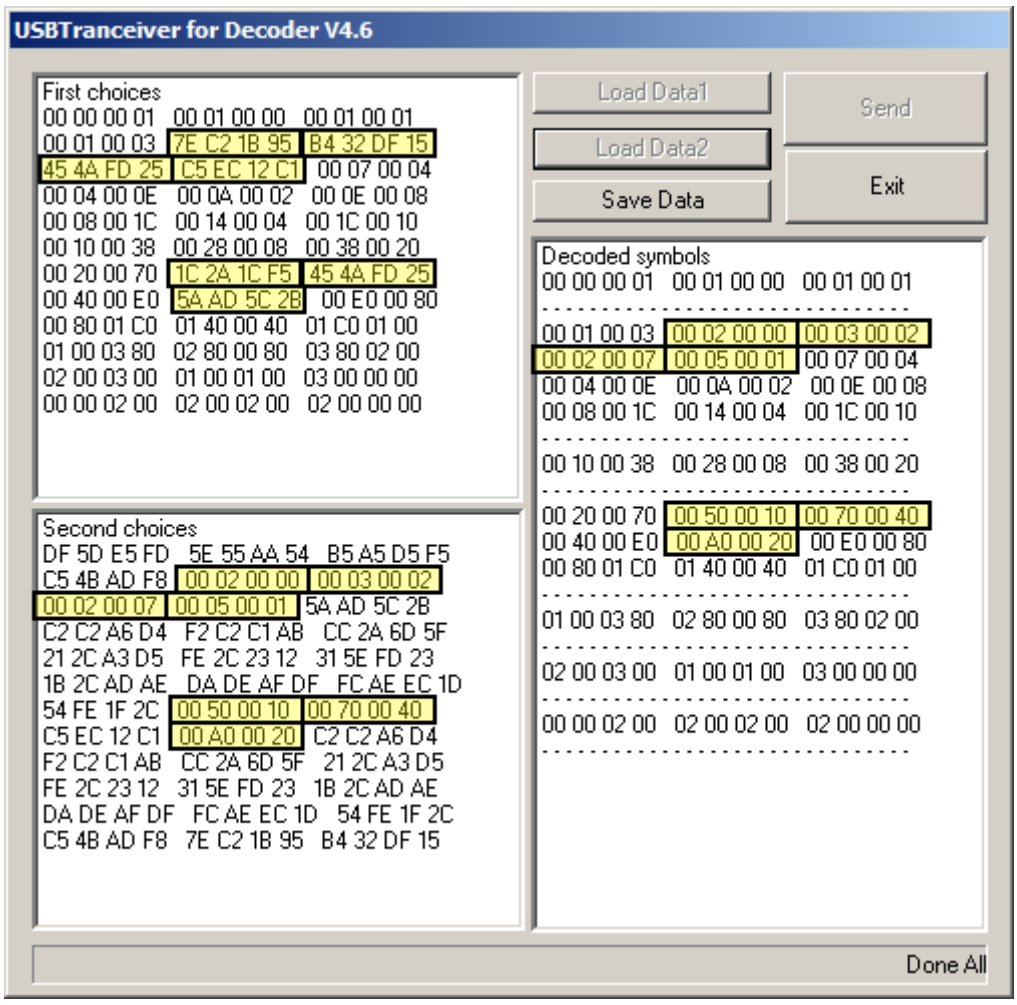
ภาพที่ 53 ผลการถอดรหัสด้วยซินโดรมเดียว

5.2 ถอดรหัสด้วยซินโดรมหลายตัว

การถอดรหัสด้วยซินโดรมหลายตัว ใช้ถอดรหัสในขั้นที่ซับซ้อน จะเริ่มถอดรหัสได้เมื่อมีจำนวนซินโดรมทั้งแต่ 2 ตัวขึ้นไป นั่นก็คือ ต้องมีตัวเลือกอันดับหนึ่งและสองอย่างละ 6 สัญลักษณ์เป็นอย่างน้อย จึงจะเริ่มถอดรหัสได้ ผลการถอดรหัสได้แบ่งเป็น 3 กรณี เพื่อให้การแสดงผลลัพธ์มีความชัดเจน

5.2.1 แก้ไขได้ด้วยตัวเลือกอันดับสอง

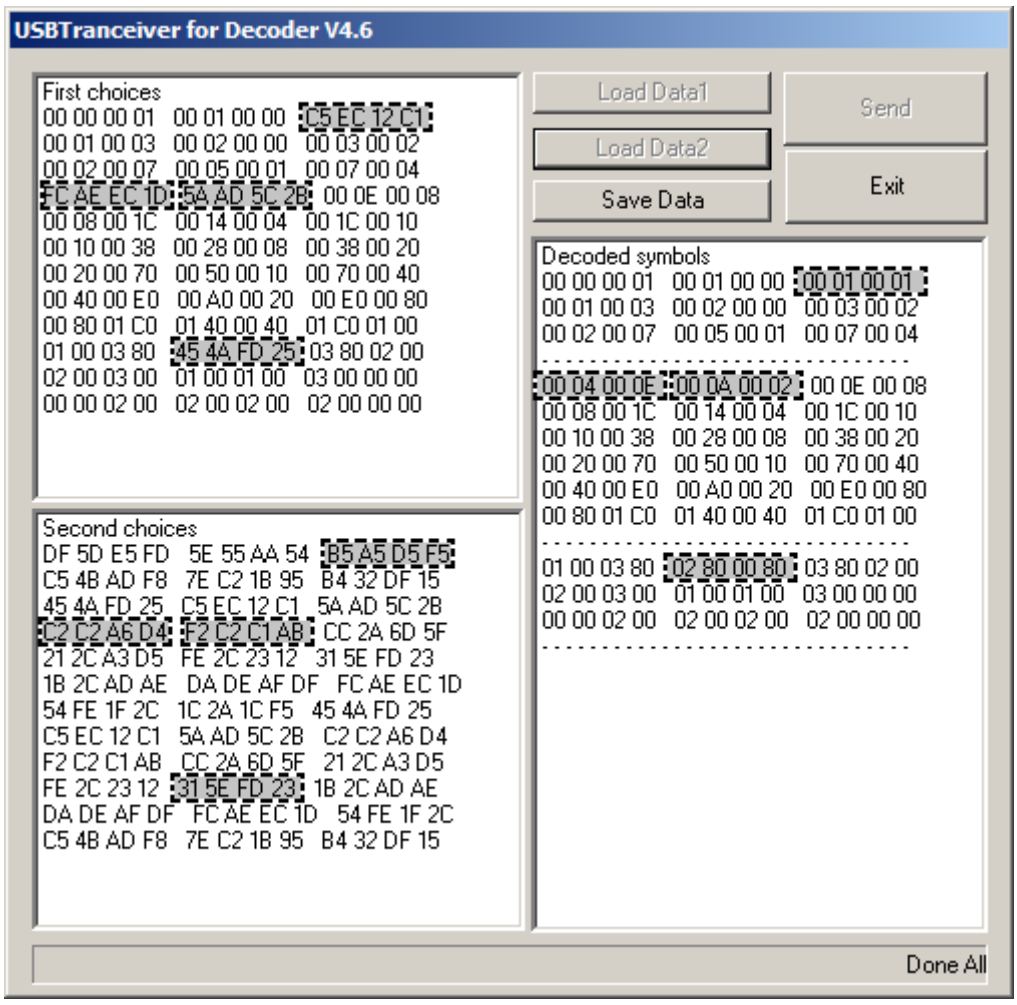
การถอดรหัสด้วยวิธีนี้ จะมีหลักการคล้ายกับการถอดรหัสด้วยซินโดรมเดียว คือ เมื่อตัวถอดรหัสจะแก้ไขค่าความผิดพลาด โดยใช้ข้อมูลในตัวเลือกอันดับสองที่ถูกต้อง มาแทนที่ในข้อมูลตัวเลือกอันดับหนึ่งที่ผิด โดยจะมีสิ่งที่ต่างกันคือ ค่าความผิดพลาดสามารถมีได้มากกว่า 1 สัญลักษณ์ต่อหนึ่งแถว หรือหมายถึงถ้าผิดทั้งแถว หรือเท่ากับ 3 สัญลักษณ์ก็สามารถถอดรหัสได้ เพื่อให้สังเกตได้ง่ายจึงได้ทำกรอบทึบล้อมตำแหน่งที่สัญลักษณ์มีความผิดพลาดเอาไว้ ดังรูปที่ 54



ภาพที่ 54 ผลการถอดรหัสด้วยซินโดรมหลายตัว ซึ่งแก้ไขได้ด้วยตัวเลือกอันดับสอง

5.2.2 แก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์

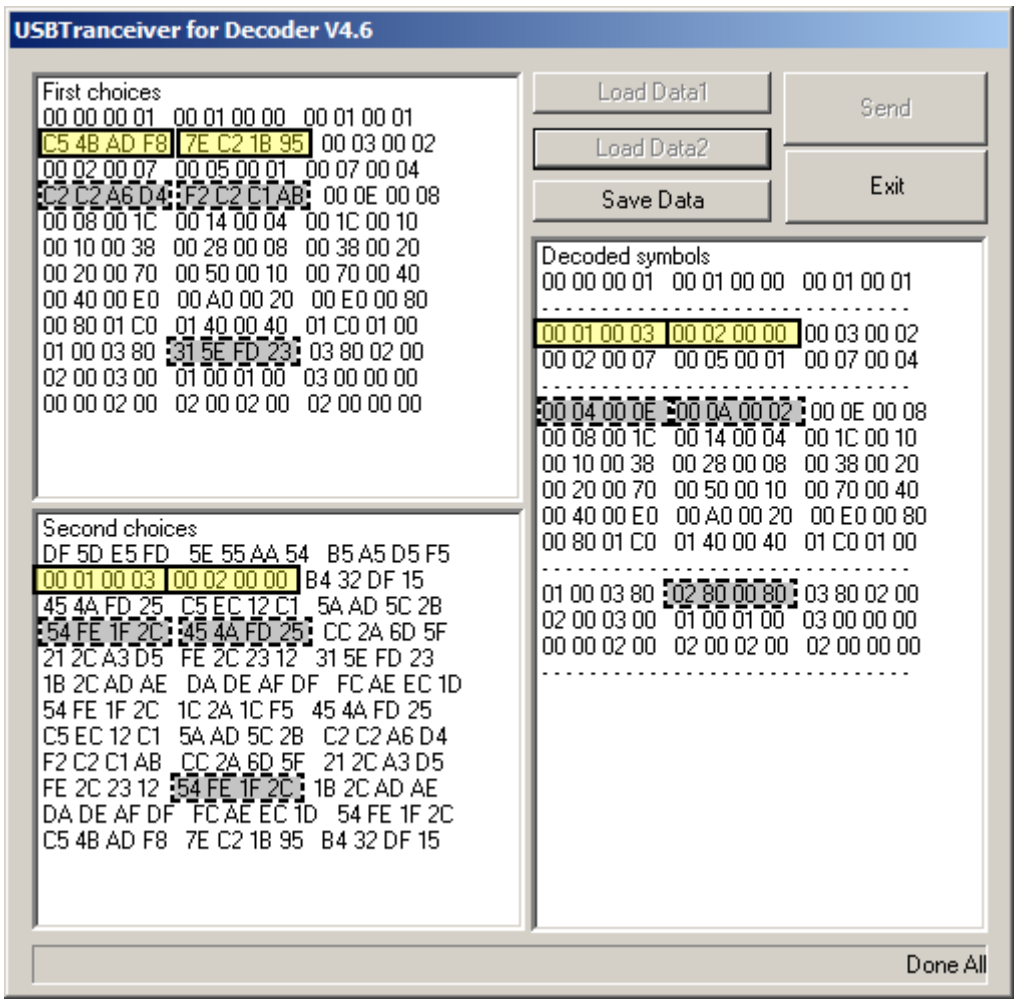
การถอดรหัสด้วยขั้นตอนนี้ จะไม่มีการนำตัวเลือกใดเลยมาใช้ นอกจากตัวเลือกอันดับหนึ่ง ไม่ว่าตัวเลือกอื่นๆ นั้นจะเป็นเช่นใดก็ตาม โดยทฤษฎีแล้วการถอดรหัสด้วยวิธีนี้ จะทำการหาตำแหน่งและค่าของความผิดพลาด จากสัมพันธ์ทางคณิตศาสตร์ของแต่ละซินโดรม เพื่อให้สังเกตเห็นตำแหน่งและค่าของความผิดพลาดได้ง่าย จึงได้ล้อมกรอบเส้นประไว้ ในรูปที่ 55



ภาพที่ 55 ผลการถอดรหัสด้วยซินโดรมหลายตัว ซึ่งแก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์

5.2.3 แก้ไขได้ด้วยตัวเลือกอันดับสอง และการรวมทางพีชคณิตได้ศูนย์

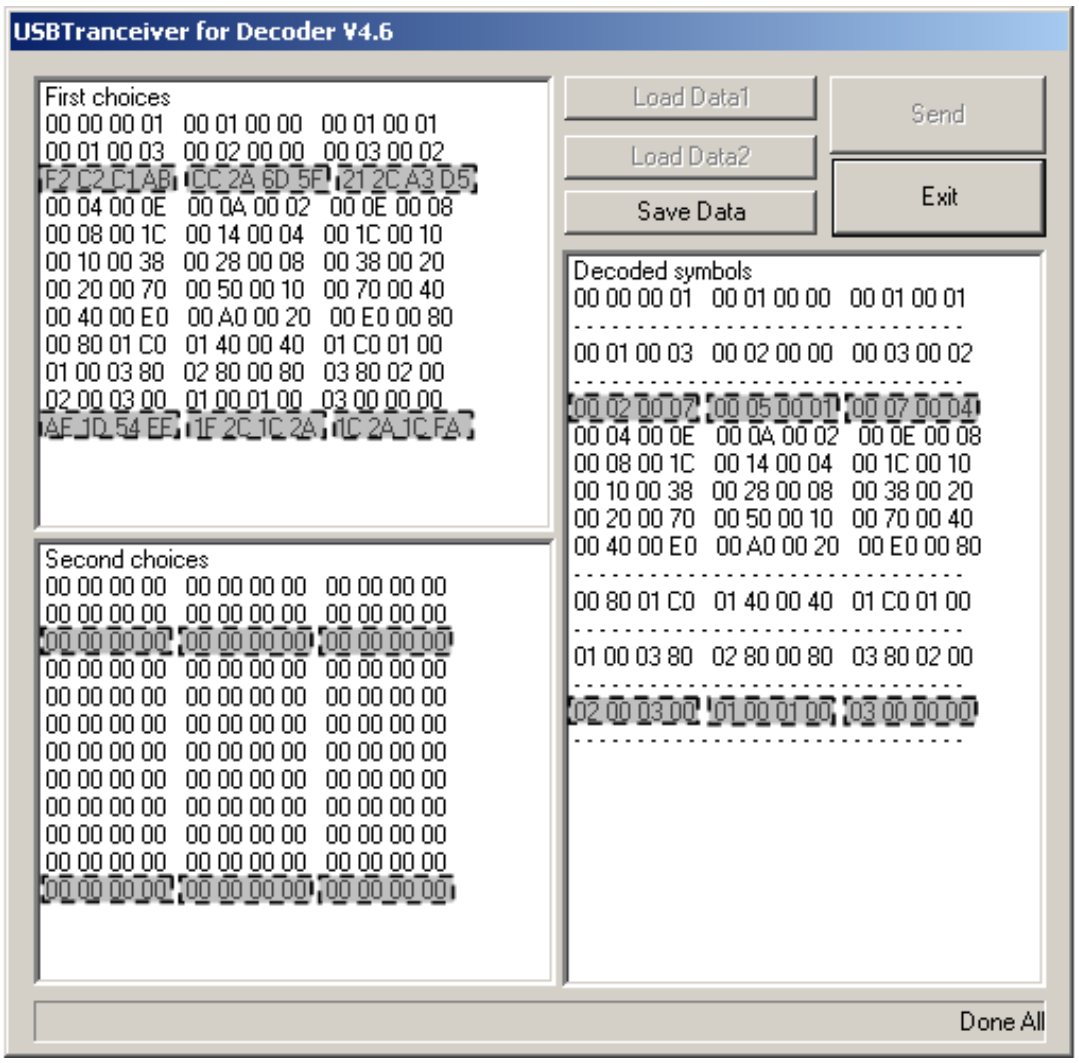
การถอดรหัสทั้งสองวิธีนี้ สามารถทำงานร่วมกันได้ โดยที่ตัวถอดรหัสจะเลือกใช้วิธีแก้ไขด้วยตัวเลือกอันดับสองก่อน ถ้าไม่สามารถแก้ไขได้หรือแก้ไขได้แต่ยังมีค่าความผิดพลาดเหลืออยู่ จะใช้วิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ เพื่อให้สังเกตเห็นได้ง่ายจึงได้ทำกรอบทึบล้อมตำแหน่ง ที่สามารถแก้ไขค่าความผิดพลาดได้ด้วยวิธีแก้ไขด้วยตัวเลือกอันดับสอง ส่วนกรอบเส้นประเป็นตำแหน่งที่สามารถแก้ไขค่าความผิดพลาดได้ด้วยวิธีการรวมทางพีชคณิตได้ศูนย์ โดยแสดงผลการถอดรหัสไว้ยังรูปที่ 56



ภาพที่ 56 ผลการถอดรหัสด้วยซินโดรมหลายตัว ซึ่งแก้ไขได้ด้วยตัวเลือกอันดับสอง และการรวมทางพีชคณิตได้ศูนย์

6. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสมีตัวเลือกเดียว

การถอดรหัสด้วยวิธีนี้ จะให้ผลลัพธ์การถอดรหัสเหมือนกัน ทั้งในกรณีมีตัวเลือก 2 ตัว หรือตัวเลือกตัวเดียว เนื่องการถอดรหัสวิธีนี้ จะทำการหาตำแหน่งและค่าของความผิดพลาด จากสัมพันธ์ทางคณิตศาสตร์ของแต่ละซินโดรมที่หาได้จากตัวเลือกอันดับหนึ่ง และในการถอดรหัสแต่ละครั้งจะต้องใช้ซินโดรมมากกว่าหนึ่งค่า เช่นเดียวกับกรณีที่มีตัวเลือก 2 ตัว เพื่อให้สังเกตเห็นตำแหน่งและค่าของความผิดพลาดได้ง่าย จึงได้ล้อมกรอบเส้นประไว้ ในรูปที่ 57 ซึ่งจะเห็นว่า มีค่าความผิดพลาดที่ตำแหน่งสุดท้ายของข้อมูล แต่ก็สามารถถอดรหัสได้ โดยใช้ซินโดรมที่สร้างจากตัวเลือกอันดับหนึ่งเพียงตัวเดียว ซึ่งในกรณีนี้จะอธิบายไว้ในหัวข้อการวิจารณ์ต่อไป



ภาพที่ 57 ผลการถอดรหัส เมื่อข้อมูลที่นำมาถอดรหัสมีตัวเลือกเดียว และความผิดพลาดเกิดในตำแหน่งสุดท้ายของชุดข้อมูล

วิจารณ์

จากผลการสังเคราะห์วงจรทำให้ทราบค่าประมาณของจำนวนโลจิกเกทที่ต้องใช้คือ 1005632 โลจิกเกท แต่จำนวนโลจิกเกทที่ชิป FPGA Virtex5 รุ่น XC5VLX110 มีคือ 4423680 โลจิกเกท โดยใช้โลจิกเกทไปเพียง 22.7% ของที่ชิป FPGA จะเห็นได้ว่าใช้ทรัพยากรที่มีไม่คุ้มค่า สาเหตุที่เป็นเช่นนี้เพราะแต่เดิมนั้นตัวรหัสถอดนี้ไม่ได้ออกมาแบบให้ใช้กับ ชิป FPGA Virtex5 รุ่น XC5VLX110 แต่ออกแบบมาให้ใช้กับชิป FPGA Xilinx รุ่น Spartan-3 XC3S1000 ที่มีขนาดของ

โลจิกเกทประมาณ 1 ล้านโลจิกเกท แต่ด้วยความผิดพลาดในการทดลองทำให้บอร์ด FPGA Spartan-3 ดังกล่าวเสียหาย จึงต้องหาบอร์ด FPGA ตัวอื่นมาทดแทน โดยบอร์ด FPGA ที่ห้องวิจัยมีให้ใช้ และมีขนาดของโลจิกเกทเพียงพอก็คือ บอร์ด FPGA Virtex5 ที่ใช้ในงานวิจัยนี้นั่นเอง

จากผลการถอดรหัสที่ได้ สามารถสรุปได้ว่าชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สำหรับรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีชุดข้อมูลขาเข้า 2 ตัวเลือกนี้ได้เป็นไปตามที่ได้ออกแบบไว้ทุกประการ ซึ่งตรงกับทฤษฎีที่มีอยู่ โดยที่ผลที่ได้จากการถอดรหัสด้วยซินโดรมเดียวเหมือนกับที่ รัชนนท์ (2551) ทำไว้ทุกประการ ซึ่งผลที่ได้นี้เป็นการสนับสนุนให้เครื่องถอดรหัสตัวนี้สามารถที่จะเป็นชิ้นงานต้นแบบได้ และสามารถนำไปพัฒนาต่อยอดเพื่อให้สามารถนำไปใช้งานจริงได้ต่อไป โดยสามารถวิจารณ์ผลแยกย่อยได้ดังนี้

1. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสเป็นข้อมูลที่ถูกต้องทั้งหมด

การถอดรหัสข้อมูลที่ถูกต้องทั้งหมด 36 สัญลักษณ์ สามารถถอดรหัสได้อย่างถูกต้อง ซึ่งได้ถอดรหัสทั้งหมด 12 ครั้ง ครั้งละ 3 สัญลักษณ์ ถูกต้องตรงตามทฤษฎี ในกรณีนี้จะเห็นได้ว่าตัวเลือกอันดับสอง ไม่มีประโยชน์ต่อการถอดรหัส เนื่องจากผลการถอดรหัสทั้งสอง กรณีแรกมีตัวเลือก 2 ตัว อีกกรณีมีตัวเลือก 1 ตัว ได้ผลลัพธ์ที่ตรงกัน

2. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสมี 2 ตัวเลือก

การถอดรหัสข้อมูลที่มี 2 ตัวเลือกนี้ จากผลการถอดรหัสทั้งสี่ตัวอย่าง สามารถสรุปได้ว่าการถอดรหัสเป็นไปอย่างถูกต้องตรงตามที่ได้ออกแบบไว้ ดังนี้

2.1 ถอดรหัสด้วยซินโดรมเดียว

การถอดรหัสด้วยซินโดรมเดียว จะทำการถอดรหัสครั้งละ 3 สัญลักษณ์ต่อตัวเลือก โดยที่ตำแหน่งสัญลักษณ์ที่มีความผิดพลาดในตัวเลือกอันดับหนึ่ง กับตำแหน่งสัญลักษณ์ที่ถูกต้องในตัวเลือกอันดับสอง และมีความผิดพลาดได้ไม่เกิน 1 สัญลักษณ์ต่อหนึ่งแถว จากผลการถอดรหัสจะเห็นว่า สามารถถอดรหัสได้ถูกต้องไม่ว่าความผิดพลาดจะในตำแหน่งใดของคำรหัส

2.2 ถอดรหัสด้วยซินโดรมหลายตัว

การถอดรหัสด้วยซินโดรมหลายตัว จะทำการถอดรหัสครั้งละ 6 สัญลักษณ์ต่อตัวเลือก เป็นอย่างน้อย แต่ไม่เกิน 30 สัญลักษณ์ต่อตัวเลือก มีผลการถอดรหัส 3 ตัวอย่าง โดย 2 ตัวอย่างแรก จะแสดงการถอดรหัสด้วยวิธีที่ต่างกัน เพื่อให้เห็นถึงความสามารถของแต่ละวิธีได้อย่างชัดเจน และในตัวอย่างสุดท้ายแสดงให้เห็นถึงความสามารถที่แท้จริงของการถอดรหัสด้วยซินโดรมหลายตัว

2.2.1 แก้ไขได้ด้วยตัวเลือกอันดับสอง

การถอดรหัสด้วยวิธีนี้ จะใช้สัญลักษณ์ที่ถูกต้องของตัวเลือกอันดับสอง มาแก้ไขให้กับ สัญลักษณ์ที่ผิดพลาดของตัวเลือกอันดับหนึ่งมีตำแหน่งเดียวกัน ซึ่งหลักการจะเหมือนกับการถอดรหัสด้วยซินโดรมเดียว แต่มีสัญลักษณ์ที่ผิดพลาดปะปนมากกว่า 1 สัญลักษณ์ต่อ 1 แถว ซึ่งจากผลการทดลองสามารถถอดรหัสได้อย่างถูกต้อง และมีข้อสังเกตอีกว่า ถ้าถอดรหัสได้ด้วยตัวเลือกอันดับสอง จำนวนซินโดรมที่ใช้จะเท่ากับจำนวนสัญลักษณ์ที่ผิดพลาด

2.2.2 แก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์

วิธีนี้จะไม่ใช้ตัวเลือกอันดับสองมาใช้ในการถอดรหัส สังเกตได้จากผลการถอดรหัส ที่ตำแหน่งของสัญลักษณ์ที่ผิดพลาดที่ล้อมกรอบทึบไว้ ในตัวเลือกอันดับสองเห็นได้ว่าเป็นสัญลักษณ์ที่ผิดทั้งหมด ดังนั้นการถอดรหัสจึงใช้เพียงตัวเลือกอันดับหนึ่งเท่านั้น แล้วใช้ความสัมพันธ์ระหว่างซินโดรมที่คำนวณมาจากตัวเลือกอันดับหนึ่ง ในการแก้ไขสัญลักษณ์ที่ผิดพลาด โดยจำนวนของซินโดรมที่ใช้ จะขึ้นอยู่กับตำแหน่งของสัญลักษณ์ที่ผิดพลาดว่ามีรูปแบบอย่างไร

2.2.3 แก้ไขได้ด้วยตัวเลือกอันดับสอง และการรวมทางพีชคณิตได้ศูนย์

การถอดรหัสกรณีนี้ มีความสมจริง คือ ตัวถอดรหัสใช้สองขั้นตอนในการถอดรหัส โดยตัวถอดรหัสจะเลือกใช้วิธีแก้ไขด้วยตัวเลือกอันดับสองก่อน หากแก้ไขไม่ได้แล้วจึงใช้วิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ ซึ่งสามารถสังเกตได้จากจำนวนซินโดรมที่ใช้ในการ

ถอดรหัสแต่ละครั้ง เช่น ในการถอดรหัสครั้งที่สองได้ใช้วิธีแก้ไขด้วยตัวเลือกอันดับสอง เนื่องจากซินโดรมที่ใช้มีค่าเท่ากับจำนวนสัญลักษณ์ที่มีความผิดพลาด คือ เท่ากับ 2

3. การถอดรหัสแบบคอนโวลูชัน (3, 2, 2) เมื่อข้อมูลที่นำมาถอดรหัสมีตัวเลือกเดียว

กรณีที่มีตัวเลือกเดียว การถอดรหัสจะใช้ได้เฉพาะวิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์เท่านั้น ซึ่งวิธีการและผลที่ได้ เหมือนกับวิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ในกรณีที่มีตัวเลือก 2 ตัว แต่ในผลการถอดรหัสสุดท้าย จะแสดงให้เห็นความสามารถของวิธีนี้เพิ่มเติม คือ การที่สามารถแก้ไขสัญลักษณ์ที่ผิดพลาด ที่เกิดในตำแหน่งสุดท้ายของชุดข้อมูลได้ ซึ่งจากผลการถอดรหัสจะพบว่า ในการถอดรหัสครั้งที่สาม มีตำแหน่งของสัญลักษณ์ที่ผิดพลาดที่มีรูปแบบเดียวกับ การถอดรหัสครั้งสุดท้าย โดยในการถอดรหัสครั้งที่สามใช้ซินโดรมไป 6 ตัว ซึ่งในการถอดรหัสครั้งสุดท้ายมีรูปแบบที่เหมือนกันก็ต้องใช้ซินโดรมในจำนวนที่เท่ากัน แต่ในหน้าต่าง “Decoded symbols” กลับระบุว่าใช้ซินโดรมไปเพียงซินโดรมเดียว เนื่องจากในขั้นตอนการเข้ารหัสข้อมูลในสองคาบสุดท้ายถูกปิดท้ายด้วยศูนย์ ดังสมการที่ (6) ทำให้ทราบได้ว่า สัญลักษณ์ตัวต่อไปที่ไม่ได้ส่งออกจากตัวเข้ารหัสก็คือ สัญลักษณ์ศูนย์ทั้งหมด ซึ่งเงื่อนไขนี้เองทำให้ตัวถอดรหัสทราบได้เองว่า จะต้องเติมสัญลักษณ์ศูนย์ต่อท้ายชุดข้อมูล ในกรณีที่ต้องถอดรหัสสัญลักษณ์ที่อยู่ในช่วงท้ายของชุดข้อมูล โดยคำรหัสศูนย์ที่เพิ่มเข้ามานี้ งานวิจัยนี้กำหนดให้เพิ่มเข้ามาได้ไม่เกิน 10 ซินโดรมเมื่อรวมกับซินโดรมของข้อมูลปกติแล้ว เช่นในผลการถอดรหัสต้องใช้ซินโดรม 6 ตัวจึงจะถอดรหัสได้ แต่มีซินโดรมของข้อมูลอยู่แล้ว 1 ตัว ดังนั้นตัวรหัสจึงเติมสัญลักษณ์ศูนย์ครั้งละ 3 ตัว จนสามารถถอดรหัสได้ โดยที่ได้เติมสัญลักษณ์ศูนย์ไปทั้งสิ้น 5 ซินโดรม หรือ 15 สัญลักษณ์

สรุปและข้อเสนอแนะ

สรุป

ฮาร์ดแวร์ต้นแบบใช้ชิป FPGA ที่ใช้ยี่ห้อ Xilinx ตระกูล Virtex5 รุ่น XC5VLX110 -1 FF676 C ความเร็วสูงสุดในการประมวลผล 550 MHz มีขนาดประมาณ 4.4 ล้านโลจิกเกท มีขาสัญญาณรับส่งข้อมูล 440 ขา โดยชิ้นงานต้นแบบได้ใช้ทรัพยากรดังนี้ ใช้โลจิกเกทประมาณ 1 ล้านโลจิกเกท ใช้งานขาสัญญาณรับส่งข้อมูล 15 ขา ได้แก่ ขาสัญญาณควบคุม 7 ขา และขาสัญญาณข้อมูล 8 ขา ประมวลผลด้วยความเร็วเดียวกับสัญญาณนาฬิกาภายนอกคือ 3.6864 MHz

ชิ้นงานต้นแบบของเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง สามารถถอดรหัสแบบคอนโวลูชัน (3, 2, 2) ที่มีค่าได้หลายระดับขั้น โดยที่ได้เข้ารหัสจากข้อมูลความยาว 20 สัญลักษณ์ไปเป็นคำรหัสความยาว 36 สัญลักษณ์ โดยที่แต่ละสัญลักษณ์มีขนาด 32 บิต หรือเท่ากับ 2^{32} ระดับขั้น หากมีชุดข้อมูลขาเข้าเพียงตัวเลือกเดียว จะทำให้การถอดรหัสในแต่ละรอบใช้ข้อมูลที่มีความยาว 36 x 32 = 1152 บิต แต่ถ้าใช้ชุดข้อมูลขาเข้า 2 ตัวเลือก ซึ่งจะทำให้ชุดข้อมูลที่ใช้ในการถอดรหัสมีความยาวรวม 72 สัญลักษณ์ หรือ 2304 บิตต่อการถอดรหัสในแต่ละรอบ

จากผลการทดสอบการทำงานชิ้นงานต้นแบบนี้ หากมีเพียงตัวเลือกเดียว ตัวถอดรหัสได้ทำการเลือกใช้แก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์มาใช้ในการถอดรหัส และสามารถถอดรหัสได้อย่างถูกต้อง หากมีการใช้ชุดข้อมูลขาเข้า 2 ตัวเลือก ตัวถอดรหัสได้เลือกการถอดรหัสด้วยซินโดรมเดียวในขั้นแรก แล้วจึงเลือกถอดรหัสด้วยวิธีแก้ไขด้วยตัวเลือกอันดับสองเป็นขั้นต่อไป ส่วนขั้นสุดท้ายตัวถอดรหัสจึงเลือกใช้แก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์ โดยกรณีที่ใช้ชุดข้อมูลขาเข้า 2 ตัวเลือกนี้ ตัวถอดรหัสสามารถถอดข้อมูลตัวอย่างที่กำหนดได้อย่างถูกต้อง แต่ตัวถอดยังไม่สามารถถอดรหัสโดยใช้วิธีแก้ไขด้วยตัวเลือกอันดับสอง พร้อมกับแก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์สำหรับการถอดรหัสข้อมูลชุดเดียวกันได้

ในส่วนของวิธีแก้ไขด้วยตัวเลือกอันดับสอง และวิธีแก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์ ในชิ้นงานต้นแบบนี้ สามารถรองรับค่าซินโดรมได้สูงสุด 10 ตัว หากเกินกว่านี้การถอดรหัสจะไม่สำเร็จ นอกจากนี้วิธีแก้ไขด้วยการรวมทางพีชคณิตได้ศูนย์ สามารถรองรับค่าความผิดพลาดได้สูงสุด 4 ตัว ซึ่งเพียงพอที่จะใช้กับค่าซินโดรม 10 ตัว

ข้อเสนอแนะ

1. การทดสอบการทำงานของโปรแกรม VHDL ในเบื้องต้นควรทดสอบสองขั้น คือ

1.1 การจำลองการทำงาน (Simulation) ในรูปแบบของแผนผังสัญญาณกับเวลา (Timing diagram) ในขั้นนี้จะช่วยตรวจหาคำสั่ง VHDL ที่ทำงานผิดพลาดได้ง่าย ด้วยการมองเห็นแผนภาพของสัญญาณที่เปลี่ยนไปตามเวลา

1.2 การสังเคราะห์วงจร (Synthesize) เป็นขั้นตอนที่ทำได้หลังจากทำการจำลองการทำงานแล้ว โดยจะให้รายละเอียดทางซอฟแวร์และฮาร์ดแวร์ ซึ่งจะสามารถประมาณขนาดของทรัพยากรต่างๆที่ต้องใช้ได้ เช่น จำนวนโลจิกเกทที่ใช้ เวลาในการประมวลผล ขาสัญญาณที่ใช้งาน เป็นต้น

2. โปรแกรมแปลงภาษา (Interpreter) ของส่วนที่ใช้ทำการจำลองการทำงาน กับส่วนที่ใช้สร้างระบบ (Implementation) เป็นคนละตัวกัน ทำให้เมื่อสั่งคำสั่ง VHDL ด้วยคำสั่งเดียวกัน แต่เมื่อแปลงเป็นภาษาเครื่อง (Machine language) แล้วอาจให้ผลลัพธ์ที่แตกต่างกันเป็นบางคำสั่ง ดังนั้นเมื่อจำลองการทำงานของซอฟแวร์ได้ผลลัพธ์ที่ถูกต้องแล้ว มิได้หมายความว่า เมื่อนำไปใช้งานจริงบน FGPA จะให้ผลลัพธ์ถูกต้องเหมือนตอนทำการจำลองการทำงาน

3. ภาษา VHDL มีความยืดหยุ่นไม่มากนัก หากนำมาใช้ออกแบบฮาร์ดแวร์ FPGA ที่มีความซับซ้อนจะมีความยุ่งยากในการเขียนซอฟแวร์ จึงควรเลี่ยงมาใช้ภาษา C ซึ่งมีอยู่ในชุดโปรแกรมออกแบบฮาร์ดแวร์ FPGA เช่น XILINX EDK ของบริษัท XILINX เป็นต้น โปรแกรมนี้เหมาะกับอุตสาหกรรมการออกแบบฮาร์ดแวร์ FPGA โดยยังไม่มีให้ใช้ในรูปแบบเพื่อการศึกษาและการวิจัย

งานในอนาคต

พัฒนาเครื่องถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลดีโคดดิง ให้สามารถใช้งานบนช่องสัญญาณจริงได้ ยังต้องมีการเพิ่มเติมในอีกหลายส่วน ได้แก่ การทดสอบประสิทธิภาพการทำงานเชิงฮาร์ดแวร์ การพัฒนาให้ใช้ได้กับโปรโตคอลมาตรฐานอื่นๆ เป็นต้น ซึ่งการทดสอบประสิทธิภาพ

การทำงานเชิงฮาร์ดแวร์ ยังไม่สามารถในการวิจัยนี้ เนื่องจากรูปแบบรับส่งข้อมูลที่ใช้ ไม่ได้ออกแบบมาให้ใช้กับการรับส่งข้อมูลที่มีอัตราเร็วที่สูงเพียงพอจะใช้ทดสอบ

ปรับปรุงการถอดรหัสให้สมบูรณ์ ซึ่งชิ้นงานต้นแบบนี้ยังไม่สามารถถอดรหัสโดยใช้วิธีแก้ไขด้วยตัวเลือกอันดับสอง พร้อมกับแก้ไขได้ด้วยการรวมทางพีชคณิตได้ศูนย์สำหรับการถอดรหัสข้อมูลชุดเดียวกันได้

การศึกษาประสิทธิภาพการถอดรหัสที่ได้จากชิ้นงานต้นแบบกับทฤษฎี ในเงื่อนไขต่างๆ เพื่อที่เป็นการยืนยันสมมติฐานในทฤษฎี ซึ่งผลที่ได้จะสามารถใช้ในการหาช่องสัญญาณที่เหมาะกับตัวถอดรหัส รวมถึงการปรับแต่งขนาดของตัวถอดรหัสให้เหมาะ ก่อนที่จะนำไปทดลองใช้ในช่องสัญญาณจริงต่อไป

เอกสารและสิ่งอ้างอิง

รัชนนท์ อินทราสกุล. 2551. **การออกแบบและสร้างระบบเพื่อใช้สำหรับการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีด้วยบอร์ด FPGA.** วิทยานิพนธ์ปริญญาโท, มหาวิทยาลัยเกษตรศาสตร์.

Avnet electronics marketing. 2008. **Xilinx Virtex -5 LX Evalution kit User guide.** User manual. Available Source: http://www.avnet.em.com, January 20, 2009.

Berrou, C., A. Glavieux and P. Thitimajshima. 1993. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. **Proc. IEEE Intl. Conf. Commun.** 1: 1064-1070.

Bose, R. C. and D. K. Ray-Chaudhuri. 1960. On a Class of Error Correcting Binary Group Codes. **Inform. Control**. 3: 68-79.

Future Technology Devices International Ltd. 2002. **D2XX Driver.** User manual. Available Source: http://www.ftdichip.com, January 20, 2009.

Haslach, C. and A. J. Han Vick. 1999. A Decoding algorithm with restrictions for array codes. **IEEE Transactions on Information Theory.** 45(7): 2339-2344.

Hocquenghem, A. 1959. Codes corecteurs k'erreurs. **Chiffres.** 2: 147-156.

Intharasakul, R. and U. Tuntoolavest. 2006. State diagram design for implementing phase I of a Vector Symbol Decoder on an FPGA board. **International Symposium on Communications and Information Technologies 2006.**

Lin, S. and D. J. Costello Jr. 2004. **Error Control Coding.** 2nd edition ed. Pearson Education, Upper Saddle River, NJ.

Metzner, J.J. 2000. Vector symbol decoding with list inner symbol decisions. **International Symposium on Information Theory 2000.**

Metzner, J.J. 2002. Vector symbol decoding with erasures, errors and symbol list decisions. **IEEE International Symposium on Information Technology 2002.** 1: 34.

Metzner, J.J. 2003. Vector symbol decoding with list inner symbol decisions. **IEEE Transactions on Commnication.** 51(3):371-380.

Metzner, J.J. and E.J Kapturowski. 1990. A general decoding technique applicable to replicated file disagreement location and concatenated code decoding. **IEEE Trans. Inf. Theory.** (36): 911-917.

Prange, E. 1957. Cyclic Error-Correcting Codes in Two Symbols. **Air Force Cambridge Research Center**. 57: 103.

Proakis, John G. 2001. **Digital communications.** McGraw-Hill, Singapore.

Reed, I. S. and G. Solomon. 1960. Polynomial codes over certain finite fields. **J. Soc. Ind. Appl. Math.** 8: 300-304.

Seo, Y.S. 1991. **A new decoding technique for convolutional codes**. Ph.D., Pennsylvania state university.

Tuntoolavest, U. 2002. **Vector symbol decoding with lists of alternative vector symbol choices and outer convolutional codes**. Ph.D., Pennsylvania state university.

Tuntooolavest, U. 2004. A simple method to improve the performance of convolutional vector symbol decoding with small symbol size. **IEEE TENCON 2004.**

Tuntoolavest, U. and J.J. Metzner. 2001. Vector symbol decoding with list inner symbol decisions,and outer convolutional codes for wireless communications, File name: TE301_3F.doc in the Conference Proceedings CD. **Electro/Information Technology Conference Proceeding 2nd IEEE.** Oakland University, USA.

Tuntoolavest, U. and J.J. Metzner. 2002. Vector symbol decoding with list symbol decisions and outer convolutional codes for reliable communications. **Integrated Computer-Aided Engineering Journal.** 2 (9): 101-116.

ภาคผนวก

ภาคผนวก ก

ข้อมูลขา FPGA

ตารางผนวกที่ ก1 รายละเอียดขา LVDS ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
TX_CLK_P	R21
TX_CLK_N	R20
TX_FRAME_P	U19
TX_FRAME_N	U20
TX_DATA0_P	T20
TX_DATA0_N	T19
TX_DATA1_P	AA20
TX_DATA1_N	AA19
TX_DATA2_P	AD19
TX_DATA2_N	AC19
TX_DATA3_P	AB20
TX_DATA3_N	AB19
TX_DATA4_P	V19
TX_DATA4_N	W19
TX_DATA5_P	AF23
TX_DATA5_N	AE23
TX_DATA6_P	Y21
TX_DATA6_N	Y20
TX_DATA7_P	V21
TX_DATA7_N	V22
TX_DATA8_P	Y23
TX_DATA8_N	Y22
TX_DATA9_P	W21
TX_DATA9_N	W20
RX_CLK_P	AB24
RX_CLK_N	AC24

ตารางผนวกที่ ก1 (ต่อ)

Signal name	FPGA Pin
RX_FRAME_P	AD24
RX_FRAME_N	AD23
RX_DATA0_P	AE26
RX_DATA0_N	AD26
RX_DATA1_P	AC21
RX_DATA1_N	AB21
RX_DATA2_P	AE25
RX_DATA2_N	AD25
RX_DATA3_P	W24
RX_DATA3_N	W23
RX_DATA4_P	AA23
RX_DATA4_N	AA24
RX_DATA5_P	AB22
RX_DATA5_N	AA22
RX_DATA6_P	V24
RX_DATA6_N	V23
RX_DATA7_P	V21
RX_DATA7_N	V22
RX_DATA8_P	U22
RX_DATA8_N	U21
RX_DATA9_P	T23
RX_DATA9_N	T22

ตารางผนวกที่ ก2 รายละเอียดขา DDR2 ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
DDR_A0	B11
DDR_A1	A14
DDR_A2	C11
DDR_A3	A12
DDR_A4	C12
DDR_A5	B12
DDR_A6	B16
DDR_A7	C14
DDR_A8	B15
DDR_A9	B14
DDR_A10	A13
DDR_A11	A15
DDR_A12	C16
DDR_BA0	C13
DDR_BA1	D16
DDR_CS#	D10
ODT	C9
DDR_WE#	B17
DDR_RAS#	A10
DDR_CAS#	D11
DDR_CLKEN	A17
DDR_LDM0	A22
DDR_UDM0	A23
DDR_LDM1	A8
DDR_UDM1	A9
DR_LDQS0 P,N	A20, B20

ตารางผนวกที่ ก2 (ต่อ)

Signal name	FPGA Pin
DR_UDQS0 P,N	C19, D19
DR_LDQS1 P,N	D8, C8
DR_UDQS1 P,N	B7, A7
DDR_D0	B24
DDR_D1	D24
DDR_D2	B25
DDR_D3	C24
DDR_D4	C23
DDR_D5	A25
DDR_D6	D23
DDR_D7	A23
DDR_D8	C21
DDR_D9	B19
DDR_D10	D21
DDR_D11	C18
DDR_D12	D18
DDR_D13	C22
DDR_D14	D20
DDR_D15	B21
DDR_D16	B5
DDR_D17	D5
DDR_D18	A5
DDR_D19	C6
DDR_D20	C7
DDR_D21	B6
DDR_D22	D6

ตารางผนวกที่ ก2 (ต่อ)

Signal name	FPGA Pin
DDR_D23	A4
DDR_D24	A3
DDR_D25	C2
DDR_D26	B4
DDR_D27	D1
DDR_D28	C1
DDR_D29	C4
DDR_D30	C3
DDR_D31	A2
DDR_CLK_FB_O	D3
DDR_CLK_FB_I	AD18

ตารางผนวกที่ ก3 รายละเอียดขา Flash memory ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
FLASH_A0	H8
FLASH_A1	H9
FLASH_A2	H18
FLASH_A3	G19
FLASH_A4	G9
FLASH_A5	G10
FLASH_A6	G17
FLASH_A7	H17
FLASH_A8	G11
FLASH_A9	H11
FLASH_A10	H16

ตารางผนวกที่ ก3 (ต่อ)

Signal name	FPGA Pin
FLASH_A11	G16
FLASH_A12	H12
FLASH_A13	G12
FLASH_A14	G15
FLASH_CE	AA10
FLASH_A15	F15
FLASH_A16	H13
FLASH_A17	H14
FLASH_A18	F13
FLASH_A19	G14
FLASH_A20	AB17
FLASH_A21	AA17
FLASH_A22	AA8
FLASH_A23	Y11
FLASH_A24	Y10
FLASH_D0	AA14
FLASH_D1	AA13
FLASH_D2	AB11
FLASH_D3	AA12
FLASH_D4	AB14
FLASH_D5	AA15
FLASH_D6	Y13
FLASH_D7	Y12
FLASH_OE	AA9
FLASH_WE	AB16
FLASH_RST	F18

ตารางผนวกที่ ก4 รายละเอียดขา On-board clock sources ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
CLK_100MHZ	E18
USB_IFCLK	E10
CLK_SYNTH0_P/N	AB10, AB9
CLK_SYNTH1_P/N	AC23, AC22
GMII_RX_CLK	AC8
GMII_TX_CLK	AC17
GBE_MCLK	AD8

ตารางผนวกที่ ก5 รายละเอียดขา User clock ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
CLK_SOCKET	E16

ตารางผนวกที่ ก6 รายละเอียดขา ICS8442 ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
DIFF_CLK_0_P	AB10
DIFF_CLK_0_N	AB9
DIFF_CLK_1_P	AB23
DIFF_CLK_1_N	AB22

ตารางผนวกที่ ก7 รายละเอียดขา On-board USB ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
USB_CTL0	AF18
USB_CTL1	AC9

ตารางผนวกที่ ก7 (ต่อ)

Signal name	FPGA Pin
USB_CTL2	AC16
USB_RDY0	AF19
USB_RDY1	AC18
USB_FD8	AE22
USB_FD9	AD21
USB_FD10	AF22
USB_FD11	AD20
USB_FD12	AE21
USB_FD13	AE20
USB_FD14	AF20
USB_FD15	AE18
USB_IFCLK	E10
USB_INT0#	AD11
USB_INT1#	AD10
USB_SLOE	Y8
USB_WU2	AB16
USB_FA0	AA18
USB_FA1	Y18
USB_PEND	AD14
USB_SLCS#	AC12
USB_RST#	AC14

ตารางผนวกที่ ก8 รายละเอียดขา Ethernet PHY ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
GBE_MDC	AE7

ตารางผนวกที่ ก8 (ต่อ)

Signal name	FPGA Pin
GBE_MDIO	AF7
GBE_MCLK	AD15
GMII_GTC_CLK	AD8
GMII_TXD0	AD16
GMII_TXD1	AE16
GMII_TXD2	AE15
GMII_TXD3	AF15
GMII_TXD4	AF13
GMII_TXD5	AF14
GMII_TXD6	AD13
GMII_TXD7	AC7
GMII_TX_EN	AF17
GMII_TX_ER	AE17
GMII_TX_CLK	AC17
GMII_RX_CLK	AC8
GBE_INT#	AF8
GBE_RST#	AC14
GMII_CRS	AE13
GMII_COL	AC13
GMII_RXD0	AE8
GMII_RXD1	AF9
GMII_RXD2	AD9
GMII_RXD3	AF10
GMII_RXD4	AE10
GMII_RXD5	AE11
GMII_RXD6	AC11

ตารางผนวกที่ ก8 (ต่อ)

Signal name	FPGA Pin
GMII_RXD7	AF12
GMII_RX_DV	AE12
GMII_RX_ER	AB12

ตารางผนวกที่ ก9 รายละเอียดขา RS232 ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
RS232_RXD	AB7
RS232_TXD	AC6
RS232_RTS	AD5
RS232_CTS	AB5

ตารางผนวกที่ ก10 รายละเอียดขา Push button ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
SWITCH_PB1	B1
SWITCH_PB2	B2
SWITCH_PB3	E8
SWITCH_PB4	F17

ตารางผนวกที่ ก11 รายละเอียดขา DIP switch ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
SWITCH0	B26
SWITCH1	C26
SWITCH2	D26

ตารางผนวกที่ ก11 (ต่อ)

Signal name	FPGA Pin
SWITCH3	D25

ตารางผนวกที่ ก12 รายละเอียดขา LED ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
LED0	E11
LED1	E17
LED2	F10
LED3	F19

ตารางผนวกที่ ก13 รายละเอียดขา SAM ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
CLOCK	F8
OEn	T4
WEn	U4
CEn	AB6
MPA0	T5
MPA1	U5
MPA2	T7
MPA3	V3
MPA4	V7
MPA5	V6
MPA6	W6
MPD00	U6
MPD01	V4

ตารางผนวกที่ ก13 (ต่อ)

Signal name	FPGA Pin
MPD02	Y3
MPD03	AA3
MPD04	W3
MPD05	W4
MPD06	AC3
MPD07	AB4
MPD08	AC4
MPD09	AA4
MPD10	W5
MPD11	AD3
MPD12	Y5
MPD13	AD4
MPD14	U7
MPD15	AA5
BRDY	Y6
IRQ	Y7
RESETn	AA7

ตารางผนวกที่ ก14 รายละเอียดขา EXP JX1 ที่ต่ออยู่กับขา FPGA

EXP Pin	Signal name	FPGA Pin
1	EXP1_SE_IO_1	D14
2	EXP1_SE_IO_0	G4
3	EXP1_SE_IO_3	G5
4	EXP1_SE_IO_2	H7
5	2.5V	-

ตารางผนวกที่ ก14 (ต่อ)

EXP Pin	Signal name	FPGA Pin
6	2.5V	-
7	EXP1_SE_IO_5	H4
8	EXP1_SE_IO_4	H6
9	EXP1_SE_IO_7	J5
10	EXP1_SE_IO_6	J4
11	2.5V	-
12	2.5V	-
13	EXP1_SE_IO_9	J6
14	EXP1_SE_IO_8	K7
15	EXP1_SE_IO_11	K6
16	EXP1_SE_IO_10	L4
17	2.5V	-
18	2.5V	-
19	EXP1_SE_IO_13	L3
20	EXP1_SE_IO_12	L7
21	EXP1_SE_IO_15	M6
22	EXP1_SE_IO_14	M5
23	2.5V	-
24	2.5V	-
25	EXP1_SE_IO_17	M1
26	EXP1_SE_IO_16	M2
27	EXP1_SE_IO_19	M4
28	EXP1_SE_IO_18	M7
29	2.5V	-
30	2.5V	-
31	EXP1_SE_IO_21	N7

ตารางผนวกที่ ก14 (ต่อ)

EXP Pin	Signal name	FPGA Pin
32	EXP1_SE_IO_20	N4
33	EXP1_SE_IO_23	N1
34	EXP1_SE_IO_22	N6
35	2.5V	-
36	2.5V	-
37	EXP1_SE_IO_25	N2
38	EXP1_SE_IO_24	P5
39	EXP1_SE_IO_27	N3
40	EXP1_SE_IO_26	P1
41	EXP1_SE_IO_28	P3
42	EXP1_DIFF_CLK_IN_p	E12
43	EXP1_SE_CLK_IN	D13
44	EXP1_DIFF_CLK_IN_n	F12
45	GND	-
46	GND	-
47	EXP1_SE_IO_29	R1
48	EXP1_SE_IO_30	R3
49	EXP1_SE_CLK_OUT	P4
50	EXP1_SE_IO_31	R2
51	GND	-
52	GND	-
53	EXP1_DIFF_p21	E6
54	EXP1_DIFF_p20	E2
55	EXP1_DIFF_n21	E5
56	EXP1_DIFF_n20	E1
57	GND	-

ตารางผนวกที่ ก14 (ต่อ)

EXP Pin	Signal name	FPGA Pin
58	GND	-
59	EXP1_SE_IO_32	T2
60	EXP1_DIFF_p18	F5
61	EXP1_SE_IO_33	T3
62	EXP1_DIFF_n18	F4
63	GND	-
64	GND	-
65	EXP1_DIFF_p19	E7
66	EXP1_DIFF_p16	F3
67	EXP1_DIFF_n19	F7
68	EXP1_DIFF_n16	E3
69	GND	-
70	GND	-
71	EXP1_DIFF_p17	F2
72	EXP1_DIFF_CLK_OUT_p	Y1
73	EXP1_DIFF_n17	G2
74	EXP1_DIFF_CLK_OUT_n	W1
75	GND	-
76	GND	-
77	EXP1_DIFF_p15	K3
78	EXP1_DIFF_p14	G1
79	EXP1_DIFF_n15	K2
80	EXP1_DIFF_n14	H1
81	EXP1_DIFF_p13	J1
82	EXP1_DIFF_p12	H3
83	EXP1_DIFF_n13	H2

ตารางผนวกที่ ก14 (ต่อ)

EXP Pin	Signal name	FPGA Pin
84	EXP1_DIFF_n12	J3
85	3.3V	-
86	3.3V	-
87	EXP1_DIFF_p11	L2
88	EXP1_RCLK_DIFF_p10	K5
89	EXP1_DIFF_n11	K1
90	EXP1_RCLK_DIFF_n10	L5
91	3.3V	-
92	3.3V	-
93	EXP1_DIFF_p9	G6
94	EXP1_DIFF_p8	P6
95	EXP1_DIFF_n9	G7
96	EXP1_DIFF_n8	R7
97	3.3V	-
98	3.3V	-
99	EXP1_DIFF_p7	R6
100	EXP1_DIFF_p6	U2
101	EXP1_DIFF_n7	R5
102	EXP1_DIFF_n6	U1
103	3.3V	-
104	3.3V	-
105	EXP1_DIFF_p5	AA2
106	EXP1_DIFF_p4	V2
107	EXP1_DIFF_n5	Y2
108	EXP1_DIFF_n4	V1
109	3.3V	-

ตารางผนวกที่ ก14 (ต่อ)

EXP Pin	Signal name	FPGA Pin
110	3.3V	-
111	EXP1_DIFF_p3	AC2
112	EXP1_DIFF_p2	AB2
113	EXP1_DIFF_n3	AC1
114	EXP1_DIFF_n2	AB1
115	3.3V	-
116	3.3V	-
117	EXP1_DIFF_p1	AE1
118	EXP1_DIFF_p0	AF2
119	EXP1_DIFF_n1	AD1
120	EXP1_DIFF_n0	AE2
121	GND	-
122	GND	-
123	GND	-
124	GND	-
125	GND	-
126	GND	-
127	GND	-
128	GND	-
129	GND	-
130	GND	-
131	GND	-
132	GND	-

ตารางผนวกที่ ก15 รายละเอียดขา EXP JX2 ที่ต่ออยู่กับขา FPGA

EXP Pin	Signal name	FPGA Pin
1	EXP2_SE_IO_1	E15
2	EXP2_SE_IO_0	H19
3	EXP2_SE_IO_3	G21
4	EXP2_SE_IO_2	H21
5	2.5V	-
6	2.5V	-
7	EXP2_SE_IO_5	G22
8	EXP2_SE_IO_4	H22
9	EXP2_SE_IO_7	H23
10	EXP2_SE_IO_6	J21
11	2.5V	-
12	2.5V	-
13	EXP2_SE_IO_9	J20
14	EXP2_SE_IO_8	J19
15	EXP2_SE_IO_11	J23
16	EXP2_SE_IO_10	K21
17	2.5V	-
18	2.5V	-
19	EXP2_SE_IO_13	K20
20	EXP2_SE_IO_12	L20
21	EXP2_SE_IO_15	L19
22	EXP2_SE_IO_14	L23
23	2.5V	-
24	2.5V	-
25	EXP2_SE_IO_17	L22
26	EXP2_SE_IO_16	M21

ตารางผนวกที่ ก15 (ต่อ)

EXP Pin	Signal name	FPGA Pin
27	EXP2_SE_IO_19	M22
28	EXP2_SE_IO_18	M25
29	2.5V	-
30	2.5V	-
31	EXP2_SE_IO_21	M24
32	EXP2_SE_IO_20	M20
33	EXP2_SE_IO_23	M26
34	EXP2_SE_IO_22	N22
35	2.5V	-
36	2.5V	-
37	EXP2_SE_IO_25	N23
38	EXP2_SE_IO_24	N21
39	EXP2_SE_IO_27	N24
40	EXP2_SE_IO_26	N19
41	EXP2_SE_IO_28	N26
42	EXP2_DIFF_CLK_IN_p	F14
43	EXP2_SE_CLK_IN	D15
44	EXP2_DIFF_CLK_IN_n	E13
45	GND	-
46	GND	-
47	EXP2_SE_IO_29	P24
48	EXP2_SE_IO_30	P19
49	EXP2_SE_CLK_OUT	M19
50	EXP2_SE_IO_31	P23
51	GND	-
52	GND	-

ตารางผนวกที่ ก15 (ต่อ)

EXP Pin	Signal name	FPGA Pin
53	EXP2_DIFF_p21	E25
54	EXP2_DIFF_p20	E22
55	EXP2_DIFF_n21	E26
56	EXP2_DIFF_n20	E23
57	GND	-
58	GND	-
59	EXP2_SE_IO_32	P25
60	EXP2_DIFF_p18	E21
61	EXP2_SE_IO_33	P26
62	EXP2_DIFF_n18	E20
63	GND	-
64	GND	-
65	EXP2_DIFF_p19	F24
66	EXP2_DIFF_p16	G24
67	EXP2_DIFF_n19	F25
68	EXP2_DIFF_n16	G25
69	GND	-
70	GND	-
71	EXP2_DIFF_p17	F22
72	EXP2_DIFF_CLK_OUT_p	V26
73	EXP2_DIFF_n17	F23
74	EXP2_DIFF_CLK_OUT_n	U26
75	GND	-
76	GND	-
77	EXP2_DIFF_p15	H24
78	EXP2_DIFF_p14	G26

ตารางผนวกที่ ก15 (ต่อ)

EXP Pin	Signal name	FPGA Pin
79	EXP2_DIFF_n15	J24
80	EXP2_DIFF_n14	H26
81	EXP2_DIFF_p13	J25
82	EXP2_DIFF_p12	G20
83	EXP2_DIFF_n13	J26
84	EXP2_DIFF_n12	F20
85	3.3V	-
86	3.3V	-
87	EXP2_DIFF_p11	K25
88	EXP2_RCLK_DIFF_p10	K23
89	EXP2_DIFF_n11	K26
90	EXP2_RCLK_DIFF_n10	K22
91	3.3V	-
92	3.3V	-
93	EXP2_DIFF_p9	L24
94	EXP2_DIFF_p8	P21
95	EXP2_DIFF_n9	L25
96	EXP2_DIFF_n8	P20
97	3.3V	-
98	3.3V	-
99	EXP2_DIFF_p7	R22
100	EXP2_DIFF_p6	R25
101	EXP2_DIFF_n7	R23
102	EXP2_DIFF_n6	R26
103	3.3V	-
104	3.3V	-

ตารางผนวกที่ ก15 (ต่อ)

EXP Pin	Signal name	FPGA Pin
105	EXP2_DIFF_p5	U24
106	EXP2_DIFF_p4	T24
107	EXP2_DIFF_n5	U25
108	EXP2_DIFF_n4	T25
109	3.3V	-
110	3.3V	-
111	EXP2_DIFF_p3	Y25
112	EXP2_DIFF_p2	W25
113	EXP2_DIFF_n3	Y26
114	EXP2_DIFF_n2	W26
115	3.3V	-
116	3.3V	-
117	EXP2_DIFF_p1	AC26
118	EXP2_DIFF_p0	AB25
119	EXP2_DIFF_n1	AB26
120	EXP2_DIFF_n0	AA25
121	GND	-
122	GND	-
123	GND	-
124	GND	-
125	GND	-
126	GND	-
127	GND	-
128	GND	-
129	GND	-
130	GND	-

ตารางผนวกที่ ก15 (ต่อ)

EXP Pin	Signal name	FPGA Pin
131	GND	-
132	GND	-

ตารางผนวกที่ ก16 รายละเอียดขา LCD ที่ต่ออยู่กับขา FPGA

Signal name	FPGA Pin
LCD_RS	AE3
LCD_E	AF3
LCD_DB4	AF4
LCD_DB5	AF5
LCD_DB6	AE6
LCD_DB7	AD6

ภาคผนวก ข

ผลงานที่ได้รับการตีพิมพ์

การศึกษาผลประสิทธิผลของการถอดรหัสเวกเตอร์ซิมโบลสำหรับรหัสคอนโวลูชั่นแบบที่มีตัวเลือกมากกว่าสองค่า และแบบที่มีตัวเลือกที่สองไม่ครบ

PERFORMANCE INVESTIGATION OF CONVOLUTIONAL VECTOR SYMBOL DECODING WITH LARGER THAN TWO CHOICES AND WITH INCOMPLETE SECOND CHOICES

อุศนา ตัณฑุลเวศม์[1], อรรถภัทร์ สืบเนื่อง[1]

Usana Tuntoolavest[1], Auttaphud Seubnaung[1]

บทคัดย่อ

เวกเตอร์ซิมโบลดีโคดดิ่ง (วีเอสดี) เป็นวิธีการถอดรหัส ซึ่งประยุกต์ใช้ได้กับรหัสทั้งแบบบล็อก และแบบคอนโวลูชันที่ใช้สัญลักษณ์แบบหลายระดับขั้น สำหรับรหัสคอนโวลูชัน จะเข้ารหัสคอนโวลูชันแบบหลายระดับขั้นและใช้วีเอสดีเป็นเครื่องถอดรหัส วีเอสดีสามารถนำการรับสัญญาณซ้ำมาประยุกต์ใช้ได้ง่าย โดยจะช่วยปรับปรุงประสิทธิผล และทำให้ขั้นตอนการถอดรหัสง่ายลง การใช้งานของการรับสัญญาณซ้ำเป็นวิธีหนึ่ง ที่จะเพิ่มจำนวนตัวเลือกของค่าสัญลักษณ์ที่ได้รับ ให้กับเครื่องถอดรหัส การทดสอบวีเอสดีที่ใช้ตัวเลือกก่อนหน้านี้ จะเน้นเฉพาะกรณีที่ทุกสัญลักษณ์มีจำนวนตัวเลือกสองตัวเสมอเพื่อความสะดวกในการทดสอบ ส่วนในบทความนี้ จะศึกษาและทดสอบประสิทธิภาพของวีเอสดีแบบคอนโวลูชัน ในกรณีที่มีจำนวนตัวเลือกได้ถึงสี่ตัวเลือก และในกรณีที่มีสองตัวเลือกแบบไม่ครบ ผลการทดลองพบว่า 1) แม้ตัวเลือกสองตัวจะเพิ่มประสิทธิผลของวีเอสดีได้อย่างชัดเจน การเพิ่มตัวเลือกเป็นสามและสี่นั้น แทบไม่เพิ่มประสิทธิผลเลย ซึ่งไม่คุ้มค่ากับความซับซ้อนของอุปกรณ์ที่เพิ่มขึ้น และ 2) วีเอสดีมีความยืดหยุ่น เพราะแม้ตัวเลือกจะไม่ครบก็ยังสามารถใช้ข้อมูลที่มีเพิ่มประสิทธิผลได้ สรุปได้ว่าควรใช้ตัวเลือกที่สองเท่าที่มีเสมอ แต่ไม่ควรใช้เกินสองตัวเลือก ซึ่งผลนี้มีประโยชน์ในการสร้างฮาร์ดแวร์อย่างมาก

ABSTRACT

Vector Symbol Decoding (VSD) is a decoding technique for both block and convolutional nonbinary encoders. Convolutional VSD uses a convolutional encoder with nonbinary symbols at the encoder and VSD at the decoder. VSD can employ diversity quite easily to improve the performance and simplify the decoding because diversity provides alternative choices for the decoder. Previous work always assumed two complete choices for simplicity. This paper explores the performance of a convolutional VSD for the case of up to four complete choices and the case of two incomplete choices. The results showed that 1) although the second choice clearly improved the performance, the third and fourth choices do not have enough effect to justify the higher complexity of the hardware. 2) VSD can use incomplete second choices to improve performance. In summary, second choices should be applied whenever possible, but no more than two choices should be used. These results are very useful for hardware implementation.

Key Words: vector symbol decoding, error correcting code, nonbinary, reliable communications

U. Tuntoolavest: **fengunt@ku.ac.th**

ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ 10903

Electrical Engineering Department, Faculty of Engineering, Kasetsart University 10903

Suupported by Kasetsart University Research and Development Institute (KURDI)

INTRODUCTION

In wireless communications, the channel sometimes has low quality and the condition is changing all the time. To increase the reliability of the received information, there are two main approaches. The first one is to use simple diversity such as space diversity by using multiple antennas. The second one is to use channel coding, which is actually another form of diversity that is more complex. Usually if the receiver uses simple diversity, it will make decision based on some simple rule such as selecting the choice that have highest signal to noise ratio and throw away the rest of the choices. It can also combine the received signal using maximum ratio combining (MRC) proposed by Breanan (1959) to get the highest signal to noise ratio. However, MRC assumes perfect knowledge of the channel attenuation and phase shifts. After that, it may input this result to the channel decoder if there is one. Some examples of macro and micro diversity system are in Chung et al. (2000) and Jakes et al. (1974). This paper is different from this usual application of diversity and channel coding in that the receiver will input all choices above some threshold to the channel decoder, which is convolutional VSD. Then VSD will use this information to help decode the received information. This is suitable because VSD concept allows for the use of list decoding where each received symbol has each own list and may have different number of alternative symbols in the list as shown by Metzner (2003).

The concept of VSD for block code was proposed by Metzner and Kapturowski (1990). Later, Haslach and Vinck rediscovered it (1999), where it was called an array code. Seo (1991) presented some concept for convolutional VSD. Metzner (2003) added the use of symbol list with block VSD. Previous simulations in Tuntooolavest (2004) on convolutional VSD with lists were for two complete choices only. That is all received symbols always had two complete alternative choices and these choices were always different. This was valid because it was assumed that the nonbinary symbols of VSD resulted from the use of a concatenated code where the inner decoder was the list Viterbi decoder (LVA) in Chen and Sundberg (2001) and the outer decoder is VSD.

This paper extends the work in two ways. 1) to show whether more than two choices should be used since the hardware would be more complex especially in terms of memory sizes. 2) to show whether VSD should make use of the second choices that are incomplete. The results will be used in implementation of VSD lab prototype under development by Intharasakul and Tuntoolavest (2006).

BACKGROUND

ALTERNATIVE CHOICES

To understand the idea of alternative choices in this paper, assume that the receiver uses 4 receiving antennas. Then, the receiver makes the preliminary decision of valid alternative choices by comparing the SNR of the received waveform for each received symbol at each antenna with a threshold. Up to 4 alternative choices for each received symbol that met threshold are considered

valid alternative choices and will be forwarded to VSD. The one with the highest SNR is called "first choice" or "main choice". The one with lower SNR is called "second choice", "third choice" and "fourth choice" respectively. It is important to note that from this setup, some symbols may have only the "first choice"; other symbols may have two, three or four choices. When all received symbols have all choices, the received sequence is referred as having complete alternative choices. If some received symbols do not have some choices, the received sequence is referred as having incomplete alternative choices.

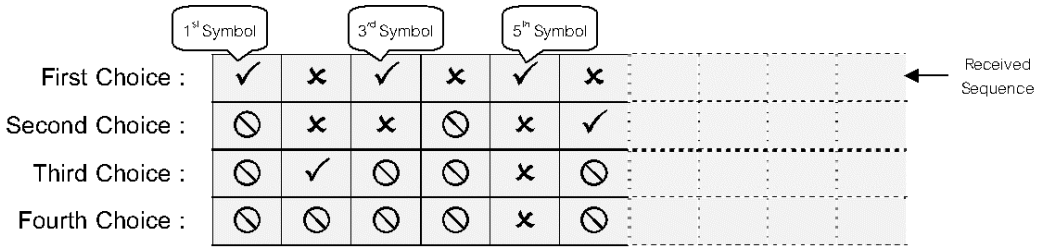


Figure 1 Example of a received sequence with incomplete alternative choices

[adapted from fig 2.1 in Nukmind and Suebnaung (2003)]

when ✓ represents correct received symbol

× represents erroneous received symbol

⊘ represents blank or no available choice

VSD: CORRECT WITH CHOICES

Vector symbol decoding (VSD) can correct with or without the help of alternative choices. If the choices are available, VSD will screen and attempt to correct as many error symbols as possible before performing the correction by verification process, which sometimes called "correct with null combination". The theory on correct with alternative choices and correct with null combinations can be found in Metzner (2003). This paper will show an example of the case when the choices are incomplete only.

Basically, VSD uses large nonbinary symbols typically of 32-bit size. The syndrome and the received sequence are considered in the matrix format.

$$S = HY \qquad (1)$$

Where **S** is the syndrome matrix, **H** is the parity check matrix and **Y** is the received matrix

Each row of the received matrix represents a 32-bit symbol and the number of columns represents the number of received symbols used in the calculation of the syndrome matrix at that time. The number of column varies because convolutional VSD computes the syndrome by one at a time. If it cannot correct with one syndrome (where the syndrome matrix contains one row), it will increase the number of syndrome to two (where the syndrome matrix contains two rows) and so on.

To understand why VSD can handle various list size for each symbol with little added complexity other than the increase in storage space, consider that VSD correct with choices by appending the differences between the first choice and each of the alternative choices at the bottom of the syndrome matrix. Any added row that is in the row space of the original syndrome matrix are found as the correct symbol and will be used to replaced the error symbols in the first choice. The index of the error symbols to be replaced is found by keeping the track of which differences that appended row comes from.

As an example, consider a case of one syndrome where each syndrome is computed by 3 received symbols. Figure 2 shows the received symbols available for the current computation. The position of the correct and the wrong symbols are the same as in figure 1.

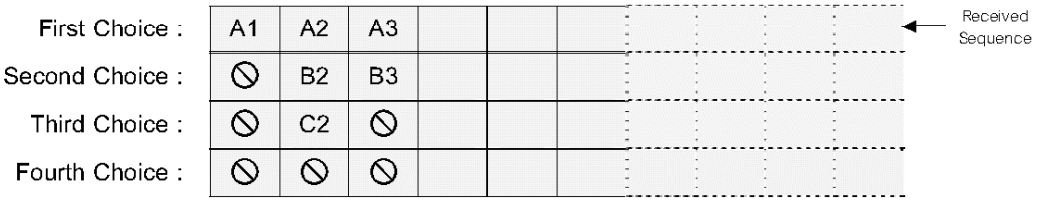


Figure 2 shows the received symbols available for one syndrome

The modified syndrome matrix in this case will be

$$S_{mod} = \begin{bmatrix} syndrome1 \\ A2 - B2 \\ A2 - C2 \\ A3 - B3 \end{bmatrix} \quad (2)$$

Since A2 is wrong and C2 is correct, the difference will be

A2-C2 = (error value $\underline{e}$ + correct symbol) (correct symbol) = error value $\underline{e}$ (3)

Also, the "*syndrome1*" is computed from A1, A2 and A3 and must be zero if all three are correct. Since only A2 is wrong, the "*syndrome1*" also equals to error value $\underline{e}$. By finding the differences in the appended row that has the same value as "*syndrome1*", the decoder finds the correct symbol and by keeping track of which differences that row came from, the decoder finds the C2 is correct and can correct the error in A2 by replacing A2 by C2. Note that for more syndromes, examples can be found in Nukmind and Suebnaung (2003).

METHOD

In the simulations, it was assumed that the encoder uses a (3,2,2) nonbinary convolutional code with 32-bit symbols and $G(D) = \begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{bmatrix}$.

The codeword is terminated at 21 encoded symbols. The vector symbol decoder used a maximum of six syndromes, which is less than previous work by Tuntoolavest (2003) since more than six syndromes are highly impractical in hardware. The simulation was divided into two main parts.

1. *Up to Four complete alternative choices*. In this part, the received sequences always had one, two, three or four complete alternative choices. For example, three complete choices mean there are three different alternative choices for every symbol. To simulate this part, the probabilities of symbols error for each alternative choice were assumed. Two different sets of probability of symbol error were used in the simulations. Case one assumed that $p_2 = 2p_1$, $p_3 = 4p_1$ and $p_4 = 8p_1$. Case two assumed that $p_2 = 2p_1$, $p_3 = 3p_1$ and $p_4 = 4p_1$ where

p_1 = probability of symbol error for the first choice

p_i = probability of symbol error for the i^{th} choice *given that the* $(i\text{-}1)^{th}$ *choice is wrong* when $i = 2, 3, 4$.

Note that p_i were defined in this format because there can be no more than one correct symbol in the list of four alternative choices for each symbol. If the first choice symbol at a particular position is correct, choice two to four at that position must be wrong. However, if the first choice symbol at a particular position is wrong, the second choice has a chance to be correct. Similarly, the fourth choice only has a chance to be correct if all three previous choices are wrong.

2. *Two incomplete alternative choices*. In this part, the received sequences always had the complete first choice but the second choices may be incomplete. The probability of not having second choices is P_n. For example if $P_n = 20\%$, each symbol have a 80% chance of having second choice and 20% chance of not having second choice. In the simulations, four different values of P_n were considered, i.e, $P_n = 0\%$, 25%, 50% and 100%. To obtain more realistic result, the probabilities of symbol error (p_1 and p_2) in this case were chosen from the simulation of the list Viterbi decoder (LVA) with the model from Tuntoolavest and Metzner (2002). This model assumed that concatenated code was used. The inner code was a convolutional code with List Viterbi Decoder (LVA) and the outer code use VSD as the decoder. The channel was a simplified two-state fading channel where the non-fade state provided perfectly demodulated symbols.

RESULTS

Figure 3 and Figure 4 show the symbol error probability after VSD finished the decoding process for the case of complete alternative choices. These two figures used two different sets of probabilities of the third and the fourth choices, but the results are almost identical. Both figures also contain four lines, but the results for using three and four complete choices are overlapping. The "i-choice(s)" in the graph refer to the number of complete choices that VSD was provided.

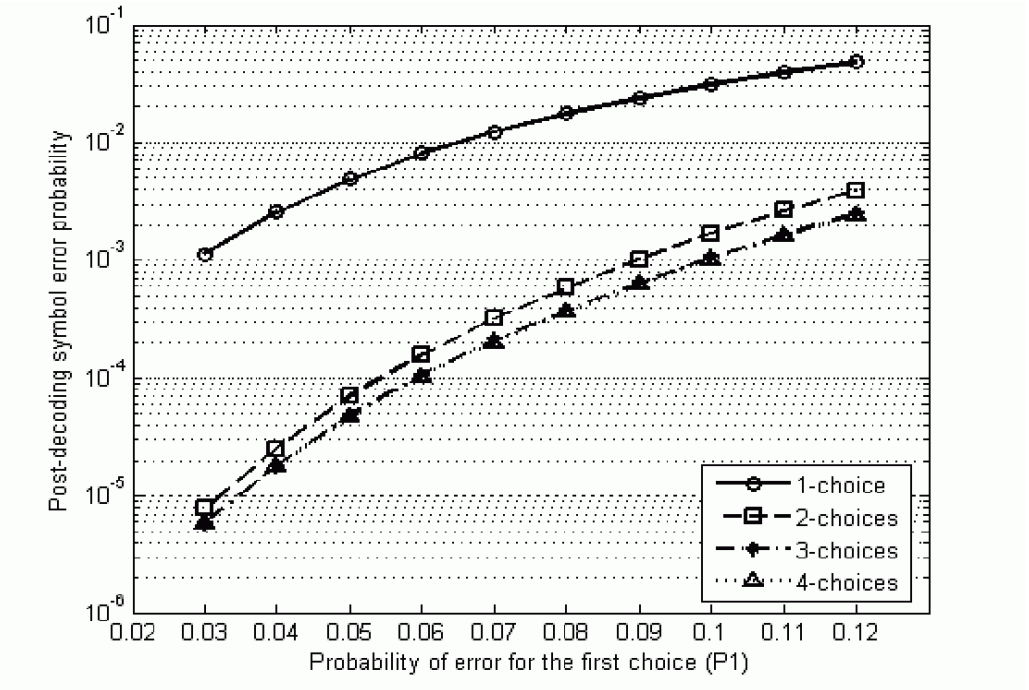


Figure 3 Post-decoding symbol error probability of VSD for the case of up to four complete alternative choices: input symbol error probabilities are $p_2 = 2p_1$, $p_3 = 4p_1$ and $p_4 = 8p_1$

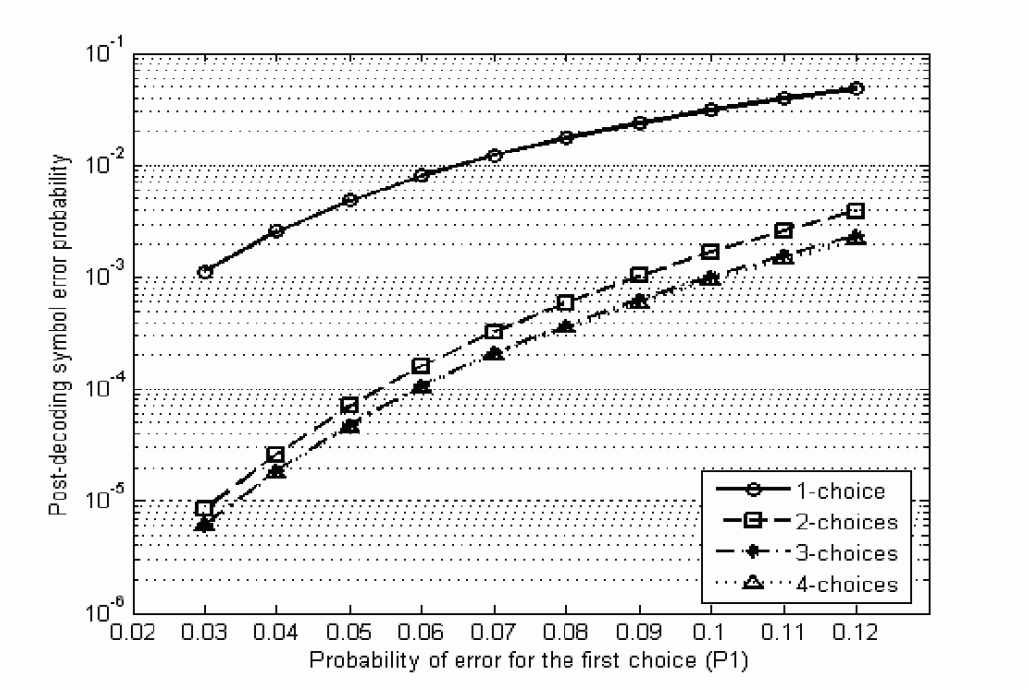


Figure 4 Post-decoding symbol error probability of VSD for the case of up to four complete alternative choices: input symbol error probabilities are $p_2 = 2p_1$, $p_3 = 3p_1$ and $p_4 = 4p_1$

Figure 5 shows symbol error probability after VSD finished the decoding process for the case of two incomplete alternative choices when the probabilities of not having the second choice (P_n) are

0%, 25%, 50% and 100%. It is clear that smaller P_n value gives better decoding performance than larger P_n as expected since smaller P_n means the decoder has more information.

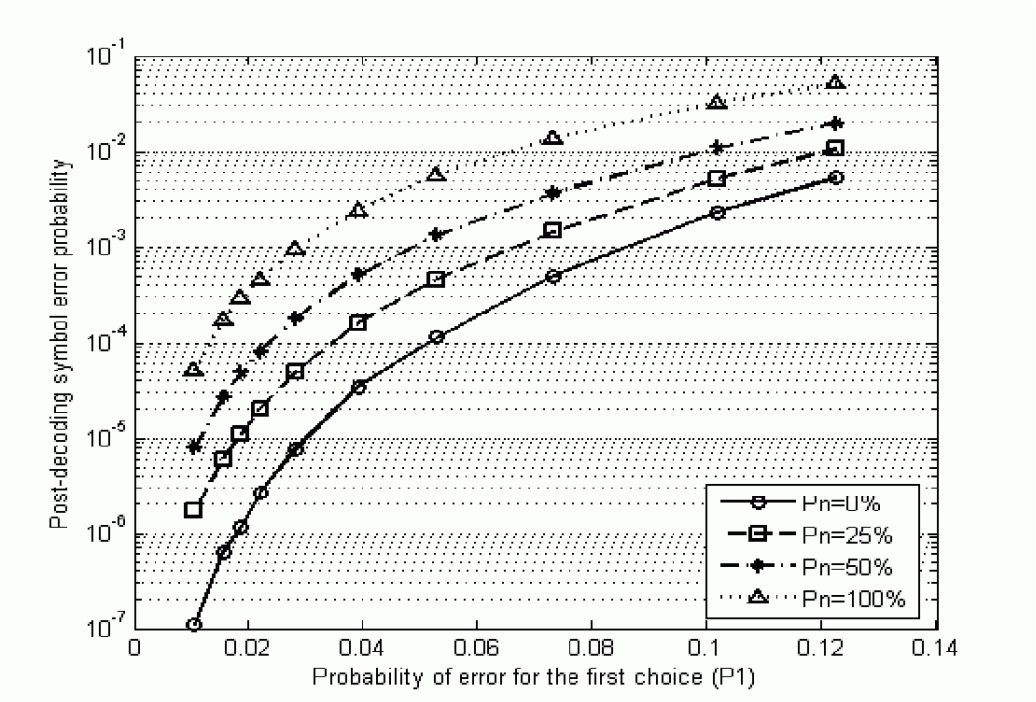


Figure 5 Post-decoding symbol error probability of VSD for the case of incomplete second choices: p1, p2 are from LVA simulation, probability of not having second choice P_n = 0%, 25%, 50%, 100%

DISCUSSIONS

From Figure 3 and Figure 4, the alternative choices can improve performance of VSD but two choices are enough since more than two choices slightly improve the performance. The two sets of input probabilities used were different only in the third and the fourth choices because the result of the second choice has been shown in previous work and was not the objective of the paper. LVA was not used in this part of the simulation because simulation of LVA with list of 4 is highly complex and Figure 3 and 4 showed that they did not affect the results.

Since the third and fourth choices contributed very little to the performance, the simulation in part two used only the first and second choices to simulate the incomplete alternative choices case. By consider only two incomplete choices instead of four, the complexity and the resource consumption especially the memory unit was reduced significantly. In Figure 5, the P_n = 0% curve means the second choice is complete. When P_n = 25% and 50%, only 75% and 50% of second choice were available to the decoder respectively. For the P_n = 100% case, there was no second choice at all. When the second choice is incomplete (P_n = 25% and 50%), the performance of VSD is lower than when it is complete. However, it is clear that the more the information on the second choice was provided, the higher the improvement on the performance will be. Therefore, this second choice should be used in the VSD decoding process as much as possible.

It should be noted that the case of incomplete alternative choice is more realistic than the case of complete alternative choices because when the channel is good, the correct one will be the only likely choice. This case also shows that VSD is flexible in handling additional information.

CONCLUSIONS

This paper showed that second choices should be applied whenever possible, but no more than two choices should be used. This information is very useful for the implementation of the VSD decoder, which is currently under development in Intharasakul and Tuntoolavest (2006). More than two choices would have much more delay during the interfacing and need much larger memory size as well as the number of gates required in the FPGA (Field Programmable Gate Array) board that is used for the lab prototype of the decoder.

REFERENCE

Brennan, D.G. 1959. Linear Diversity Combining Techniques. Proc. IRE., vol. 47, pp. 1075-1102.

Chen, B. and C-E. W. Sundberg. 2001. List Viterbi algorithms for continuous transmission. IEEE Transactions on Communications, vol.49, no.5, pp.784-792.

Haslach, C. and A.J. Han Vinck. 1999. A decoding algorithm with restrictions for array codes. IEEE Trans. Inform. Theory, Vol.45, No. 7, pp.2339-2344.

Intharasakul, R. and U. Tuntoolavest. 2006. State diagram design for implementing phase I of a Vector Symbol Decoder on an FPGA board. International Symposium on Communications and Information Technologies Proceeding (ISCIT 2006), Bangkok, Thailand.

Metzner, J.J. and E.J. Kapturowski. 1990. A general decoding technique applicable to replicated file disagreement location and concatenated code decoding. IEEE Trans. Inf. Theory, vol. 36, pp. 911-917.

Metzner, J.J. 2003. Vector symbol decoding with list inner symbol decisions. IEEE Trans. Comm., vol. 51, Issue 3, pp. 371 – 380.

Nukmind N. and A. Suebnaung. 2003. Effect of varying number of alternative choices on Vector Symbol Decoding. Senior project in Electrical engineering, Kasetsart University.

Seo, Y.S. 1991. A new decoding technique for convolutional codes, Ph.D. Thesis, Penn State U, USA.

Tuntoolavest, U., J.J. Metzner. 2002. Vector symbol decoding with list symbol decisions and outer convolutional codes for reliable communications. Integrated Computer-Aided Engineering Journal, vol. 9, no. 2, p. 101-116, IOS Press, ISBN 1069-2509.

Tuntoolavest, U. 2003. Performance comparison between a maximum and a non-maximum distance outer code decoding. The 3rd International Symposium on Communications and Information Technologies Proceeding (ISCIT 2003), Songkhla, Thailand.

Tuntooolavest, U. 2004. A simple method to improve the performance of convolutional vector symbol decoding with small symbol size. IEEE TENCON 2004, Chiang Mai, Thailand.

Convolutional Vector Symbol Decoder Phase II on FPGA: Correct with Second Choice

Usana Tuntoolavest, Auttaphud Seubnaung, and Rachanon Intharasakul
Department of Electrical Engineering
Kasetsart University, Bangkok, Thailand
Tel: +66-2-9428555 ext 1565, Fax: +66-2-942-8555 ext 1550
E-mail: fengunt@ku.ac.th, g4765184@ku.ac.th, rucha_nont@hotmail.com

***Abstract*— The concept of Vector Symbol Decoding (VSD) for convolutional codes was shown to be a good decoding technique for convolutional codes with large nonbinary symbols or packet-symbols. However, the hardware implementation in Phase I was designed for very simple error patterns, which can be decoded using only one syndrome vector. This new Phase II decoder extends its ability to decode more complicated error patterns using up to four syndrome vectors. For Phase II, the decoding was done by using the concept of correct with second choice only. Lab prototype was implemented on an FPGA board and worked exactly as expected. The next phase will be VSD decoder without the help of second choices.**

I. INTRODUCTION

Some types of information require high reliability in transmission such as personal identification information, banking information or even general file transfer. Channel coding or error correcting code had been used as a common method to increase the reliability of data transmission. The criteria of selecting the code depend on the level of reliability and the type of the transmission medium whether it is a wired or wireless medium.

Nonbinary codes are not commonly used except for the Reed-Solomon codes (RS codes) [1] because they are generally very complex. However for wireless channel where the channel condition can change from good to bad such as when fading occurs, the use of nonbinary codes may be preferred. RS codes have been used for space communications and satellite communications. Recently more interest is given to nonbinary code decoding in terms of packet decoding by viewing each packet as one symbol as described by Luby and Mitzenmacher [2] in 2005 or Metzner [3] in 2007. In addition, there was an idea on reducing the complexity of nonbinary decoding for low-density parity check code (LDPC) over GF(q) [4]. The work in [2, 4] focused on LDPC code, but the work in [3] was for general linear codes that used large nonbinary symbols. This idea [3] is related to the concept of Vector Symbol Decoding (VSD) first proposed by Metzner [5].

The principle of VSD decoding technique can be applied for any linear block or convolutional codes. One application of VSD is as an outer decoder of concatenated codes. When the inner code decoder can provide alternative choices for each received symbol, these choices will improve the performance of VSD and simplify the decoding steps. VSD for convolutional codes or conv. VSD with two alternative choices for each received symbol was described in [6] and modified in [7]. The performance of conv. VSD has been shown to be better than RS code for certain conditions [8]. Preliminary Implementation of VSD was done for some specific error patterns that can be corrected using only one syndrome vector and two alternative choices [9]. For the selected (3,2,2) convolutional code, these correctable patterns were the cases that no more than one error symbol occurred in each group of three received symbols and the alternative choice at the error position must contain correct symbol. The previous lab prototype for these specific error patterns was called VSD phase I.

This paper focuses on the implementation of conv. VSD phase II that is capable of correcting more complicated and more variety of error patterns using up to four syndrome vectors. The increase in the syndromes allows the decoder to correct error patterns that contain several consecutive error symbols. The decoding step is based on the use of second choice to correct errors. Therefore, this phase is labeled "correct with second choice". Although the phase I and phase II can both be viewed as using the second choice to decode, the main difference is that there requires many extra steps in order to employ more than one syndrome. This is because the error positions are not obvious in the case of multiple syndromes. The details can be found in [6, 7]. An example is also shown in section III. The next phase of VSD is under development and will be able to correct errors without the help of second choice.

The lab prototype of conv. VSD phase II was implemented on an FPGA board. The system was set up such that the input file can be input through a computer, which was connected to the FPGA board via a USB module and the decoded data were then transmitted back to the computer and saved as a text file. The program was written with VHDL (Very high speed integrated circuit Hardware Description Language) language [10, 11]. The program was downloaded onto the FPGA (Field Programmable Gate Array) board with the Platform cable USB.

Supported by Kasetsart University Research and Development Institute

1-4244-0977-2/07/$25.00 © 2007 IEEE

Section II describes the decoding steps of VSD: correct with second choices using an example. Section III demonstrates the set up of the test system. Section IV explained the state diagram design. Section V shows the decoding result from the hardware. Section VI and VII are the discussions and conclusions.

II. BACKGROUND

VSD decoder uses the verification based method where it tries to confirm the correct symbols. It can decode with or without the help of second choice received sequence. If the second choice received sequence is available, it was analyzed [12] that the decoder should always use this extra information.

The second choice received sequence may come from diversity reception such as the use of multiple antennas. It may also come from the inner decoder that can provide alternative received sequence for the same transmitted sequence such as List Viterbi Algorithm (LVA) [13]. By the help of the second choice received sequence, VSD can pick the correct symbols from the second choice to replace the wrong symbols in the first choice for many error patterns. This process was called "correct with second choice". The error patterns that can not be corrected would then go through another decoding step that will find error locating vector (or verification vector) and error values. This next step is not included in VSD phase II.

Although the "correct with second choice" step was usually considered as a pre-decoding step, but it could decode some error patterns by itself without the use of the main VSD decoding step. The necessary condition for the successful decoding of the "correct with second choice" step is that the positions that contain error symbols in the first choice received sequence must have correct symbols in the second choice sequence. Note that this is a necessary, but not sufficient condition. The details of the sufficient conditions can be found in [14].

The decoding step of conv. VSD for the "correct with second choice" was described with examples for two and three syndrome vectors in [7]. The steps for four syndromes vector are similar and will be showed as an example here. The code is a (3, 2, 2) nonbinary convolutional code with

$$G(D)=\begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{bmatrix} \quad (1)$$

And the correspond parity check matrix [6] is

$$\mathbf{H}=\begin{bmatrix} 1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&1&1&1&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&1&1&1&1&1&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0&1&1&\cdots&\cdots&\cdots&\cdots&\cdots&\cdots \\ 1&1&1&1&0&0&0&1&0&\cdots&\cdots&\cdots&1&1&1 \\ 0&0&0&1&1&1&1&0&0&\cdots&\cdots&\cdots&0&1&1 \\ 0&0&0&0&0&0&1&1&1&\cdots&\cdots&\cdots&0&1&0 \\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\cdots&\cdots&\cdots&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1 \end{bmatrix} \quad (2)$$

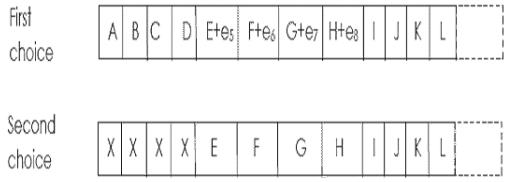


Fig 1. The first and second choice of the received sequence where e_i represent the error for the i^{th} received symbol
× represents a wrong received symbol

Example: Suppose the transmitted symbols are ***A, B, C, D, E, F,…,Z*** where each alphabet represents an r-bit vector. Suppose there are error symbols in the 5th, 6th, 7th and 8th symbols of the first choice received sequence. Fig. 1 shows the positions of correct and wrong symbols.

As shown in [7], the first step is to compute one syndrome vector using the first n received first choice symbols where n comes from the (n,n-1,m) code and so n = 3 in this case. It should be noted that this is a nonbinary decoder, so one syndrome vector is similar to one syndrome bit in the binary decoder case.

For this example, the first group of received symbols (A,B,C) were correct. The decoder realized this because the syndrome vector for this first group would be zero. Therefore, this first received symbol group can be transmitted out immediately. Then, the decoder considered the second group of received symbols (D,$(E+e_5)$,$(F+e_6)$) and found that they contain errors since the corresponding syndrome vector was not zero. This error pattern can not be corrected with one syndrome (Phase I VSD decoder) since there were more than one error symbol in one syndrome as explained in [9].

After the decoder failed to correct with one syndrome, it would increase the number of syndrome vectors to two by involving three new received symbols in the syndrome calculation. The decoder would continue to increase the number of syndromes until it can correct or has reached the maximum number of syndromes allowed.

For this error pattern, the decoder can not correct with one, two or three syndromes. Since the cases of two or three syndromes are similar to the four syndrome case, only the four syndrome vectors case is shown in detail here.

Consider the syndrome calculation for 4 syndrome vectors,

$$\mathbf{S_4} = \mathbf{H_4 Y} = \mathbf{H_4 E} \quad (3)$$

Where

$$\mathbf{H_4} = \begin{bmatrix} 011111000000000 \\ 010011111000000 \\ 100010011111000 \\ 111100010011111 \end{bmatrix} \quad (4)$$

$\mathbf{H_4}$ is the Parity check matrix with 4 rows starting from the second row to the fifth row because the nonzero syndrome was first discovered in the second group of received symbols. The number of columns of **H** equals to the last row index multiplied by n (5x 3 = 15 columns)

Y is the received symbol matrix whose size depends on the number of columns of **H** currently used.

$$\mathbf{Y} = [\mathrm{A\ B\ C\ D}\ (E+e_5)\ (F+e_6)\ (G+e_7)\ (H+e_8)\ \mathrm{I\ J\ K\ L\ M\ N\ O}]^T \quad (5)$$

E is the corresponding error matrix of **Y**

$$\mathbf{E} = [\mathbf{0\ 0\ 0\ 0}\ e_5\ e_6\ e_7\ e_8\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0}]^T \quad (6)$$

And $\mathbf{S_4}$ is the syndrome matrix with 4 syndrome vectors. From (3), (4) and (6),

$$\mathbf{S_4} = [(e_5+e_6)\ \ (e_5+e_6+e_7+e_8)\ \ (e_5+e_8)\ \ e_8]^T \quad (7)$$

After computing the syndrome matrix, the decoder would compute the difference between the first and the second choice values for each symbol by performing the XOR or the modulo-2 addition since each symbol is a binary vector over GF(2). The differences only have meanings and equal to the error value for that symbol when the first choice is wrong and the second choice is correct, which are the case of symbol 5, 6, 7 and 8. All differences are appended to the syndrome rows in (7) to obtain a new syndrome matrix called **Sa**.

$$\mathbf{S_a} = [(e_5+e_6)\ \ (e_5+e_6+e_7+e_8)\ \ (e_5+e_8)\ \ e_8\ \ w\ (e_5)\ (e_6)\ (e_7)\ (e_8)\ \ w\ w\ w\ w\ w\ w\ w]^T \quad (8)$$

Where w = wrong (no meaning) difference patterns

There are 12 differences for the 12 symbols under the decoding process (the 4th to 15th received symbols). Note that the first three symbols (A, B, C) were declared to be corrected before and no differences for these symbols were needed.

The decoder identified the meaningful patterns by finding the differences that were in the row space of $\mathbf{S_4}$. These differences were the actual error patterns and the positions of error symbols were known by construction [14]. Obviously, the difference for the 8th symbol was in the row space of $\mathbf{S_4}$. With more thorough investigation, the differences for the 5th, 6th and 7th symbol are found to be in the row space of $\mathbf{S_4}$ as well. Specially, the (mudolu-2) sum of row 3 and 4 of **Sa** revealed that e_5 was in the row space of $\mathbf{S_4}$. Similarly, the sum of row 1,3 and 4 revealed e_6 and the sum of row 1,2 and 4 revealed e_7.

In the implementation, Gauss-Jordan reduction was used to perform column operations on the modified syndrome **Sa** to find the difference rows that were in the row space of $\mathbf{S_4}$. The **w** patterns are very unlikely to be in the row space of $\mathbf{S_4}$ especially when the symbol size is large enough such as 32-bits/symbol in this implementation.

After discovering the error patterns and the positions of error symbols, the decoder could make correction at this point. To be certain, the decoder would compute the syndrome value again after the correction. If the syndrome matrix was zero, the correction was right. If not, the decoding failed and the decoder would report the failure.

III. SYSTEM SETUP

The selected FPGA board for phase II contained more gate than the one for phase I and also had external memory (RAM). The new board was the Xilinx Sparatn-3 PCI Express Starter Kit that uses FPGA chip number XC3S1000-4FG676C. This chip contained one million gates. This was more than necessary for phase II, bit it was selected to have room for the next phase III. The user guide of this board can be found in [15].

Downloading program for the FPGA chip and the Platform Flash PROM on this board can be done with Platform cable USB. The system was setup as shown in Fig. 2 and 3. The more close up of the board was shown in Fig 4.

The selected USB (Universal Serial Bus) module for the interface was the same as the one for VSD phase I since the new FPGA board was selected such that it used the same voltage level (3.3 V) as the old board. Specially, the USB module was Ezy USB-M01 from Astron Logic Research and Development. The details of the interfacing between USB module and the decoder on FPGA board including the state diagram design for the data transmission between the two boards was explained when the phase I was presented in [9].

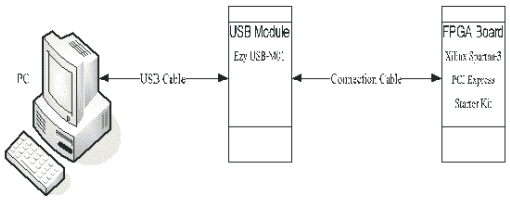


Fig. 2 System setup showing the interface between the decoder board and a computer

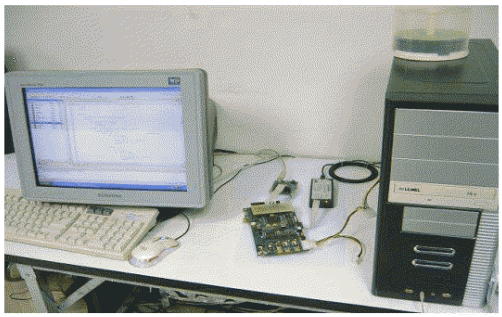


Fig 3. System setup in the lab

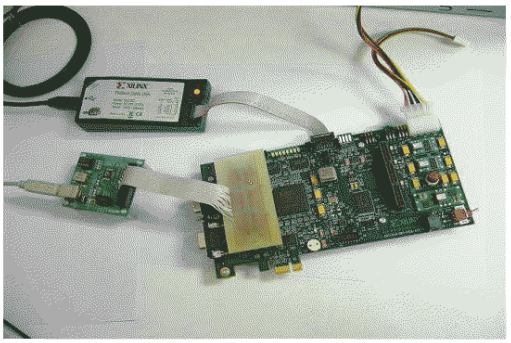


Fig. 4 The FPGA board with the USB module and the Platform cable USB.

IV. METHOD

The decoding concept for "correct with second choice" was analyzed and the state diagram was designed as shown in Fig 5. The program was written using the VHDL language and downloaded into the FPGA board described in section III. The decoder was implemented for the (3,2,2) nonbinary convolutional code described in (1) with 32-bits/symbol. Up to 4 syndromes were allowed although it could be extended to more syndromes with added computation and memory.

The states in Fig 5 may be viewed as the interface part nad the processing part. The interface between the decoder and the computer contained the "read" and the "write" procedure. The interface was very similar to the Phase I decoder [9] with some modifications. Basically, the "read" part, which included state *read* to state *chk_choice_rx*, read the received symbol from the computer in 8-bit blocks. The decoder then needed to assemble four 8-bit blocks to represent one 32-bit symbol. When it received a complete set of 3 first choice and 3 second choice symbols, it would start to process.

The processing part started by checking whether the number of syndromes was more than 4 syndromes. If it was, the decoder would report that decoding failed. If not, the decoder would go to the "decode with one syndrome" part, which was the part of Phase I decoder. This part included the state *Cal_fs* to state *Chk_condition*. If it could decode with one syndrome, the decoder would then go to the "make correction" part. This part included state *Make_correct* to state *Update_fs_mem*. Basically, it made the correction and then updated the memory of the decoder, so that the correct symbol values replaced the wrong symbol values in the decoder memory. This update was necessary because convolutional codes are codes with memory. If the memory was not updated, subsequence syndrome calculations would be wrong.

After the "make correction" part, the decoder then prepared to transmit the decoded symbols out by adding header and split the 32-bit symbols to four 8-bit blocks and started the "write" part. The decoder would keep writing out to the computer via the USB module until all decoded symbols were

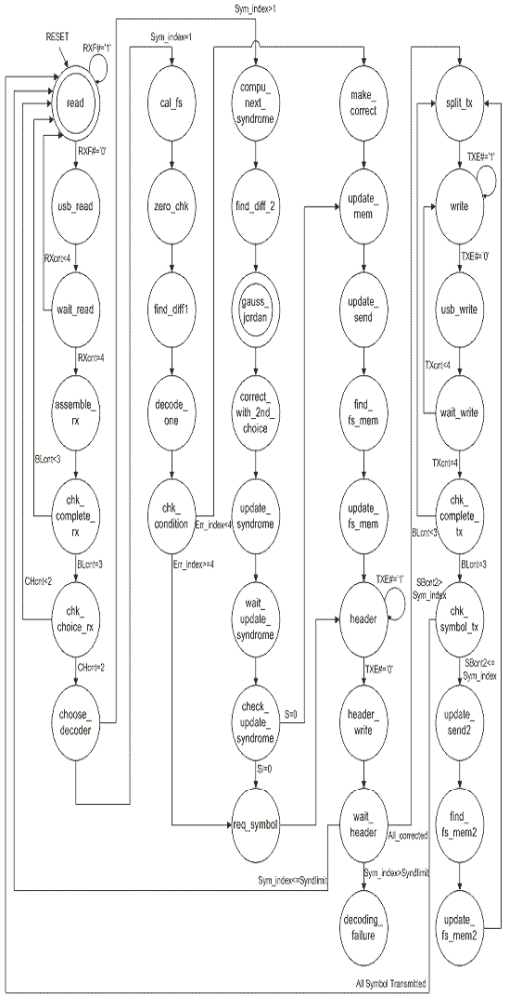


Fig. 5 State diagram design for Phase II conv. VSD decoder

transmitted out. Then it would go back and start reading new received symbols from the received sequence again.

If the decoder could not correct with one syndrome, it would request more symbols. This was not in Phase I decoder. This "decode with second choice (many syndromes)" part started with state *Req_symbol* and went through the whole "read" part and then went from state *Compu_next_syndrome* through state *Chk_update_syndrome*.

State *Comput_next_syndrome* would compute syndrome of new symbols block. State *Find d_j2* would compute the difference between the first and second choice by XOR operation. State *Gauss_jordan* would perform the column

operation to simplify the modified syndrome matrix. This helped the decoder identify the difference rows that were in the row space of the syndrome in state *Correct_with_2nd_choice*. After the correction, the decoder again updated the syndromes and verified that the correction was right and complete by checking that the update syndromes were all equal to zeroes. If updated syndromes were not all zero, the decoder would require more received symbols for the decoding. But if the updated syndromes were all zero, the decoder would go to state *Update_mem* to update the memory before sending the decoded symbols back to the computer.

V. RESULTS

The test result was obtained by inputting the error patterns that could not be corrected with fewer than four syndromes. An example of the received sequences with this condition was described in section II. Fig. 6 shows the input file in hexadecimal number format of the first choice received sequence that contains 36 symbols or 144 bytes. Each 32-bit symbol was represented by 8 hexadecimal numbers, so each row showed 3 symbols.

Fig. 7 shows the second choice received sequence for the same transmitted data sequence. Fig. 8 showed the correct position as "/" and error positions as "X". The first group of three received symbols (first row in Fig 6.) was correct, so their corresponding syndrome was zero. The decoder next considered the second group, which consisted of the 4th-6th symbols (second row in Fig. 6.). This group had errors in the 5th and the 6th symbols and their syndrome was nonzero. Since there were more than one error symbol for one syndrome, the phase I decoder (correct with one syndrome) failed. The decoder then needed to increase the number of received symbols in consideration.

As explained in detailed in section II, this set of errors (5th, 6th, 7th and 8th error symbols) can be corrected with four syndromes starting from the first nonzero syndrome found. This means that the decoder would be able to decode after it received the 4th – 15th symbols (4 syndromes need 4 x n = 12 symbols for (n,n-1,m) code).

```
00 00 00 01 00 01 00 00 00 01 00 01
00 01 00 03 7E C2 1B 95 B4 32 DF 15
45 4A FD 25 C5 EC 12 C1 00 07 00 04
00 04 00 0E 00 0A 00 02 00 0E 00 08
00 08 00 1C 00 14 00 04 00 1C 00 10
00 10 00 38 00 28 00 08 00 38 00 20
00 20 00 70 1C 2A 1C F5 45 4A FD 25
00 40 00 E0 5A AD 5C 2B 00 E0 00 80
00 80 01 C0 01 40 00 40 01 C0 01 00
01 00 03 80 02 80 00 80 03 80 02 00
02 00 03 00 01 00 01 00 03 00 00 00
00 00 02 00 02 00 02 00 02 00 00 A0
```

Fig 6 An example of tested first choice sequence

```
DF 5D E5 FD 5E 55 AA 54 B5 A5 D5 F5
C5 4B AD F8 00 02 00 00 00 03 00 02
00 02 00 07 00 05 00 01 5A AD 5C 2B
C2 C2 A6 D4 F2 C2 C1 AB CC 2A 6D 5F
21 2C A3 D5 FE 2C 23 12 31 5E FD 23
1B 2C AD AE DA DE AF DF FC AE EC 1D
54 FE 1F 2C 00 50 00 10 00 70 00 40
C5 EC 12 C1 00 A0 00 20 C2 C2 A6 D4
F2 C2 C1 AB CC 2A 6D 5F 21 2C A3 D5
FE 2C 23 12 31 5E FD 23 1B 2C AD AE
DA DE AF DF FC AE EC 1D 54 FE 1F 2C
C5 4B AD F8 7E C2 1B 95 B4 32 DF 15
```

Fig. 7 The corresponding second choice sequence

```
// // // // // // // // // // // //
// // // // XX XX XX XX XX XX XX XX
XX XX XX XX XX XX XX XX // // // //
// // // // // // // // // // // //
// // // // // // // // // // // //
// // // // // // // // // // // //
// // // // XX XX XX XX XX XX XX XX
// // // // XX XX XX XX // // // //
// // // // // // // // // // // //
// // // // // // // // // // // //
// // // // // // // // // // // //
// // // // // // // // // // // //
```

Fig. 8 The corresponding error patterns.

```
00 00 00 01 00 01 00 00 00 01 00 01
00 01 00 03 00 02 00 00 00 03 00 02
00 02 00 07 00 05 00 01 00 07 00 04
00 04 00 0E 00 0A 00 02 00 0E 00 08
00 08 00 1C 00 14 00 04 00 1C 00 10
00 10 00 38 00 28 00 08 00 38 00 20
00 20 00 70 00 50 00 10 00 70 00 40
00 40 00 E0 00 A0 00 20 00 E0 00 80
00 80 01 C0 01 40 00 40 01 C0 01 00
01 00 03 80 02 80 00 80 03 80 02 00
02 00 03 00 01 00 01 00 03 00 00 00
00 00 02 00 02 00 02 00 02 00 00 00
```

Fig. 9 The decoded sequence contained all correct symbols.

This means that the decoder would be able to decode after it received the 4th – 15th symbols (4 syndromes need 4 x n = 12 symbols for (n,n-1,m) code). After the correction, these 12 symbols would be transmitted out. Then conv. VSD decoder considered the 16th to 18th symbols and found that they were all correct since their syndrome was zero. Next, the decoder reached the group starting with the 19th symbols and found

that they contain errors. This second group of errors consisted of the 20th, 21st and 23rd symbols. This group can also be corrected within 4 syndromes.

When the decoder reached the second error set, the first error set had been cleared and it only had to handle the new error set. This clearing of the old errors was a reason the decoder can correct several groups of error symbols in a received sequence within 4 syndromes at a time. In the test several error patterns were tested and more than one set of error was included in the sequence to ensure that the decoder can actually handle more than one group of errors.

Fig. 9 shows the decoded sequence as a hexadecimal file. This file was received by the computer from the FPGA board through the USB module. It showed that the correct second choice replaced the wrong first choice for all error positions. The decoded sequence was exactly the same as the correct code word before it was corrupted by errors.

VI. Discussions

The results showed that the decoder can perform the "correct with second choice" operation as expected. Although other results were not shown here, several error patterns that could be correct with one, two, three and other four syndromes were also tested and worked. That is it also included the ability of phase I decoder. The number of syndrome vectors can also be increased with the same technique described here. Previous simulation used up to 16 syndrome vectors in the performance analysis [6,8]. However, the hardware will require more computations and memory especially when the verification part (Phase III) is added in the future.

The state diagram design was important in the implementation process and was explained in details here. It can be seen that Phase II needs many new added states. The state that was most complicated was the Gauss-Jordan reduction since it required several computation steps.

The specifications for Phase II conv. VSD were designed based on the complete conv. VSD that can correct with or without the help of the second choices. Many selected parameters were analyzed by simulations to obtain the values with good performance and without unnecessary complexity.

The electronics board with an FPGA chip was selected for the implementation because it is convenient to simulate the VHDL program on computer before the downloading it into the hardware.

VII. Ccnclusions

This work verifies the concept and the decoding procedure of VSD for convolutional codes with Hardware implementation. The Lab prototype of VSD phase II on FPGA board can correct all error patterns that are correctable with second choices using up to four syndrome vectors. The allowed VSD decoder phase II to correct several consecutive error symbols in addition to the specific error patterns of no more than one error symbol in each group of three received symbols as in phase I. The interface was designed so that the board received an input file and sent back the decoded file to a computer for the ease of monitoring. This lab prototype is an important step in extending the ability of the decoder. The future work is to build the prototype for VSD phase III that can correct with or without the help of second choice received sequence.

References

[1] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal on Applied Mathematics*, vol.8, 1960, pp. 300-304.

[2] M.G. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol.51, no.1, pp.120-127, Jan 2005.

[3] J.J. Metzner, "Packet-symbol decoding for reliable multipath reception with no sequence numbers," *To appear in proceedings of International Conference on Communications*, June 2007.

[4] D. Declercq, M. Fossorier "Decoding Algorithms for Nonbinary LDPC Codes Over GF(q)," *IEEE Trans. Comm.*, vol. 55, Issue 4, pp. 633 – 643, April 2007

[5] J.J. Metzner and E.J. Kapturowski, "A general decoding technique applicable to replicated file disagreement location and concatenated code decoding," *IEEE Trans. Inf. Theory*, vol.36, pp.911-917, July 1990.

[6] U. Tuntoolavest and J.J. Metzner, "Vector symbol decoding with list symbol decisions and outer convolutional codes for reliable communications," *Integrated Computer-Aided Engineering Journal*, vol. 9, no. 2, 2002, p. 101-116, IOS Press, ISBN 1069-2509.

[7] U. Tuntooolavest, "Additional steps of convolutional vector symbol decoding for general data sequence," *Proc. of the 2007 ECTI International Conference*,vol.2, pp.651-654, May 9-12, 2007, Mae Fah Luang University, Chang Rai, Thailand.

[8] U. Tuntoolavest, "Performance comparison between a maximum and a non-maximum distance outer code decoding," *The 3rd International Symposium on Communications and Information Technologies Proceeding (ISCIT 2003)*, September 3-5,2003, Songkhla, Thailand.

[9] R. Intharasakul and U. Tuntoolavest, "State Diagram Design for Implementing Phase I of a Vector Symbol Decoder on an FPGA Board." *Proceedings of International Symposium on Communications and Information Technologies 2006 (ISCIT 2006)*, Oct 18-20, 2006, Bangkok, Thailand.

[10] D. L. Perry, *VHDL: programming by example*. 4th edition. McGraw-Hill, Boston, 2002.

[11] A. P. Pedroni, *Circuit design with VHDL*. The MIT Press, 2004.

[12] U. Tuntooolavest and A. Seubnaung, "Perforamnce investigation of Convolutional Vector Symbol Decoding Convolutional Vector Symbol Decoding with Larger than Two Choices and with Incomplete Second Choices," *Proceedings of Kasetsart Univerisity Annul Conf.*, Jan 30 - Feb 2, 2007, Bangkok, Thailand.

[13] N. Seshadri and C-E. W. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Trans. Comm.*, vol. 42, pp. 313-323, Feb./Mar./Apr. 1994.

[14] J.J. Metzner, "Vector symbol decoding with list inner symbol decisions," *IEEE Trans. Comm.*, vol.51, Issue3, pp.371–380, Mar 2003.

[15] Spartan-3 for PCI Express Starter Kit Board User Guide v1.3, 2007 **http://www.xilinx.com/bvdocs/publications/ug256.pdf**

ประวัติการศึกษา และการทำงาน

ชื่อ –นามสกุล	นายอรรถภัทร์ สืบเนื่อง
วัน เดือน ปี ที่เกิด	วันที่ 8 มีนาคม 2525
สถานที่เกิด	อำเภอพญาไท จังหวัดกรุงเทพมหานคร
ประวัติการศึกษา	วศ.บ.(วิศวกรรมไฟฟ้า) มหาลัยวิทยาลัยเกษตรศาสตร์
ตำแหน่งหน้าที่การงานปัจจุบัน	วิศวกรระบบ
สถานที่ทำงานปัจจุบัน	บริษัท วิทยุการบินแห่งประเทศไทย จำกัด
ผลงานดีเด่นและรางวัลทางวิชาการ	
ทุนการศึกษาที่ได้รับ	