ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)

ปริญญา

วิศวกรรมไฟฟ้า	วิศวกรรมไฟฟ้า
สาขา	**ภาควิชา**

เรื่อง การออกแบบและสร้างระบบเพื่อใช้สำหรับการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี ด้วยบอร์ด FPGA

Design and Implementation System for Nonbinary Encoder and Decoder with FPGA Board

นามผู้วิจัย นายรัชนนท์ อินทรสกุล

ได้พิจารณาเห็นชอบโดย

ประธานกรรมการ .....

(ผู้ช่วยศาสตราจารย์อุศนา ตัณฑุลเวศม์, Ph.D.)

กรรมการ .....

(ผู้ช่วยศาสตราจารย์ศรีจิตรา เจริญลาภนพรัตน์, Ph.D.)

กรรมการ .....

(ผู้ช่วยศาสตราจารย์วชิระ จงบุรี, Ph.D.)

หัวหน้าภาควิชา .....

(รองศาสตราจารย์มงคล รักษาพัชรวงศ์, Ph.D.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

.....

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

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

วิทยานิพนธ์

เรื่อง

การออกแบบและสร้างระบบเพื่อใช้สำหรับการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีด้วยบอร์ด FPGA

Design and Implementation System for Nonbinary Encoder and Decoder with FPGA Board

โดย

นายรัชนนท์ อินทรสกุล

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)
พ.ศ. 2551

รัชนนท์ อินทรสกุล 2551: การออกแบบและสร้างระบบเพื่อใช้สำหรับการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีด้วยบอร์ด FPGA ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า) สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า ประธานกรรมการที่ปรึกษา: ผู้ช่วยศาสตราจารย์อุศนา ตัณฑุลเวศม์, Ph.D. 83 หน้า

การเข้ารหัสถอดรหัสช่องสัญญาณเป็นวิธีการหลักที่ทำให้ระบบสื่อสารมีความน่าเชื่อถือโดยเฉพาะอย่างยิ่งระบบสื่อสารไร้สาย ช่องสัญญาณเกือบทั้งหมดจะถูกออกแบบให้ใช้กับสัญลักษณ์ไบนารีเพื่อความสะดวก แต่ก็มีการใช้งานบางอย่างที่ต้องการใช้สัญลักษณ์นอนไบนารี เช่นในส่วนรหัสภายนอกของรหัสคอนคาทิเนต ตัวอย่างที่สนใจ คือ วิธีการถอดรหัสแบบเวกเตอร์ซิมโบลสำหรับรหัสคอนโวลูชันที่ถูกออกแบบให้ใช้สัญลักษณ์ขนาดใหญ่ ซึ่งโดยทั่วไปคือ 32 บิตต่อสัญลักษณ์ ก็จะทำให้วิธีการเข้ารหัสถอดรหัสช่องสัญญาณมีลักษณะที่แตกต่างไปจากเดิมที่ใช้กับสัญลักษณ์ไบนารี

งานวิจัยนี้เป็นการสร้างระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีใช้เทคโนโลยีการสร้างวงจรรวม ซึ่งระบบการเข้ารหัสคอนโวลูชัน (3, 2, 2) และระบบการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลที่สร้างขึ้นด้วยบอร์ด FPGA นั้นสามารถทำการเข้ารหัสคอนโวลูชันสัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์ได้และสามารถแก้ไขความผิดพลาดบางแบบซึ่งเป็นการทดสอบฟังก์ชันเบื้องต้นบางส่วนในการเชื่อมต่อประสานระหว่างบอร์ด FPGA กับเครื่องคอมพิวเตอร์

_____ / ___ / ___

ลายมือชื่อนิสิต ลายมือชื่อประธานกรรมการ

Rachanon Intharasakul 2008: Design and Implementation System for Nonbinary Encoder and Decoder with FPGA Board. Master of Engineering (Electrical Engineering), Major Field: Electrical Engineering, Department of Electrical Engineering. Thesis Advisor: Assistant Professor Usana Tuntoolavest, Ph.D. 83 pages.

Channel coding is the main method to increase reliability of data communications, especially for wireless communication systems. Almost all channel codes were designed for binary symbol for simplicity. However, some applications require the use of nonbinary symbol such as the outer codes of concatenated codes. A particular example is convolutional vector symbol decoding which was designed for large symbols. The typical size is 32 bits/symbol. In this case, the method and the system for encoding and decoding them need to be redesigned.

This thesis is focused on the design and implementation of nonbinary encoding and decoding system using integrated circuit technology by FPGA board. To test the designed systems and the interface between the FPGA board and the computer, the (3, 2, 2) nonbinary convolutional encoding with 32 bits/symbol and some functions of vector symbol decoding have been implemented. They worked exactly as designed.

_____ _____ ___ / ___ / ___

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

ผู้วิจัยขอกราบขอบพระคุณ ผศ. ดร.อุศนา ตัณฑุลเวศม์ ประธานกรรมการที่ปรึกษาวิทยานิพนธ์ ผศ. ดร.ศรีจิตรา เจริญลาภนพรัตน์ กรรมการที่ปรึกษาสาขาวิชาเอก และ ผศ. ดร.วชิระ จงบุรี กรรมการที่ปรึกษาสาขาวิชารอง ที่ให้คำปรึกษาในการเรียน การค้นคว้าวิจัย ตลอดจนการตรวจแก้ไขวิทยานิพนธ์จนกระทั่งเสร็จสมบูรณ์ และกราบขอบพระคุณ ผศ. ดร.ภุชงค์ อุทโยภาศ ผู้แทนบัณฑิตวิทยาลัย ที่ได้ให้ความกรุณาตรวจแก้ไขวิทยานิพนธ์ให้สมบูรณ์ยิ่งขึ้น

ขอกราบขอบพระคุณอาจารย์ภาควิชาวิศวกรรมไฟฟ้าทุกท่าน ที่ได้อบรมสั่งสอนและมอบความรู้อันเป็นประโยชน์อย่างยิ่งในการนำไปใช้ประโยชน์ต่อไป และขอกราบขอบพระคุณ รศ. ดร.มงคล รักษาพัชรวงศ์ ที่เอื้อเฟื้ออุปกรณ์เพื่อใช้ศึกษาทดลอง และขอขอบคุณ รุ่นพี่จากห้องวิจัย SCORPion ที่ให้ความช่วยเหลือและให้คำแนะนำต่างๆ รวมทั้ง คุณอรรถภัทร์ สืบเนื่อง ที่ให้ความช่วยเหลือในการเขียนโปรแกรมภาษา C++ เพื่อใช้ทดสอบ และขอขอบคุณสถาบันวิจัยและพัฒนาแห่งมหาวิทยาลัยเกษตรศาสตร์ที่ให้ทุนสนับสนุนการวิจัยครั้งนี้ด้วย

ด้วยความดีหรือประโยชน์อันใดเนื่องจากวิทยานิพนธ์เล่มนี้ ขอมอบแด่คุณพ่อ คุณแม่ ที่ได้อบรมและให้กำลังใจผู้วิจัยตลอดมาในทุกเรื่อง

รัชนนท์ อินทรสกุล

ตุลาคม 2551

สารบัญ

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

คำอธิบายสัญลักษณ์และคำย่อ

API	=	Application Programming Interface
DLL	=	Dynamic-Link Library
EEPROM	=	Electrically Erasable Programmable Read-Only Memory
FIFO	=	First In First Out
FPGA	=	Field-Programmable Gate Array
FSM	=	Finite-State Machines
GUI	=	Graphical User Interface
HDL	=	Hardware Description Language
JTAG	=	Joint Test Action Group
PC	=	Personal Computer
TTL	=	Transistor-Transistor Logic
USB	=	Universal Serial Bus
VHDL	=	Very High Speed Integrated Circuit HDL
VSD	=	Vector Symbol Decoding
XOR	=	Exclusive Or

การออกแบบและสร้างระบบเพื่อใช้สำหรับการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี ด้วยบอร์ด FPGA

Design and Implementation System for Nonbinary Encoder and Decoder with FPGA Board

คำนำ

ปัจจุบันเทคโนโลยีการสื่อสารโทรคมนาคมมีการพัฒนาไปอย่างรวดเร็ว โดยเฉพาะอย่างยิ่งเทคโนโลยีด้านระบบสื่อสารไร้สาย ซึ่งในระบบสื่อสารไร้สายจะนิยมใช้รูปแบบการสื่อสารแบบดิจิตอล เนื่องจากสามารถทนทานต่อสัญญาณรบกวนที่เกิดขึ้นในช่องสัญญาณได้ดีกว่าการสื่อสารแบบแอนะลอก

ระบบสื่อสารไร้สายใช้สายอากาศในการแพร่กระจายคลื่นแม่เหล็กไฟฟ้าผ่านช่องสัญญาณที่เป็นอากาศ ซึ่งในช่องสัญญาณมีสัญญาณรบกวนที่ไม่อาจหลีกเลี่ยงได้ จึงอาจทำให้การสื่อสารระหว่างฝั่งส่งและฝั่งรับเกิดความผิดพลาดขึ้นได้ ถ้าในระบบสื่อสารบางอย่างที่ความถูกต้องของข้อมูลมีความสำคัญมาก เช่น ระบบธนาคาร หากการสื่อสารข้อมูลเกิดความผิดพลาดก็จะทำให้เกิดความเสียหาย แต่ถ้าทางฝั่งรับสามารถตรวจสอบได้ว่าข้อมูลที่รับมานั้นมีความผิดพลาด ทางฝั่งรับอาจจะร้องขอให้ทางฝั่งส่งทำการส่งข้อมูลมาใหม่อีกครั้งก็ได้ และมีบางกรณีที่การส่งข้อมูลใหม่จากทางฝั่งส่งทำได้ยาก เช่น การสื่อสารผ่านดาวเทียม เนื่องจากในการรับส่งสัญญาณผ่านดาวเทียมมีค่าหน่วงเวลา ซึ่งหากมีการร้องขอให้ส่งข้อมูลใหม่ จะทำให้เวลาในการรับส่งข้อมูลเพิ่มขึ้นอีกมาก

การเข้ารหัสถอดรหัสช่องสัญญาณเป็นวิธีการหนึ่งที่นำมาใช้กับระบบสื่อสารไร้สาย เพื่อช่วยแก้ปัญหาสัญญาณรบกวนที่เกิดขึ้นในช่องสัญญาณ ทำให้ระบบสื่อสารมีความน่าเชื่อถือเพิ่มขึ้น โดยทางฝั่งส่งจะทำการเข้ารหัสช่องสัญญาณด้วยตัวเข้ารหัสช่องสัญญาณ ซึ่งจะเพิ่มข้อมูลส่วนเติมเต็มเข้าไปกับข้อมูลที่ส่ง ส่วนทางฝั่งรับจะทำการถอดรหัสช่องสัญญาณด้วยตัวถอดรหัสช่องสัญญาณ เพื่อตรวจสอบว่าข้อมูลที่ได้รับมานั้นมีความผิดพลาดเกิดขึ้นหรือไม่ ถ้าพบว่าข้อมูลมีความผิดพลาด ตัวถอดรหัสช่องสัญญาณก็จะทำการแก้ไขข้อมูลให้ถูกต้อง แต่ในกรณีที่ไม่สามารถแก้ไขข้อมูลให้ถูกต้องได้ ก็จะร้องขอให้ทางฝั่งส่งทำการส่งข้อมูลมาใหม่อีกครั้ง หรือในกรณีที่ข้อมูลไม่ได้มีความสำคัญมากนัก ก็อาจจะยอมให้เกิดความผิดพลาดได้

รหัสแบบบล็อกและแบบคอนโวลูชันมีการใช้งานกันอย่างแพร่หลาย ซึ่งส่วนใหญ่จะนิยมใช้กับสัญลักษณ์ไบนารี แต่สำหรับสัญลักษณ์นอนไบนารีจะนำไปใช้เป็นรหัสภายนอก (Outer Codes) ของรหัสคอนคาทิเนต (Concatenated Codes) ซึ่งมักเป็นรหัสแบบบล็อก จะใช้รหัสรีด-โซโลมอน (Reed-Solomon Code)ในส่วนของการถอดรหัสภายนอกของรหัสคอนคาทิเนต แต่ยังมีการถอดรหัสแบบหนึ่งที่เรียกว่าการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล สามารถใช้ได้กับทั้งรหัสแบบบล็อกและแบบคอนโวลูชันสัญลักษณ์นอนไบนารี โดยสำหรับรหัสคอนโวลูชันสัญลักษณ์นอนไบนารีนั้น ต้องสร้างระบบและตัวเข้ารหัสที่เหมาะสมขึ้นมาเองเพื่อใช้งานเฉพาะ จึงเป็นที่มาของหัวข้อวิทยานิพนธ์นี้

การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลเหมาะกับรูปแบบข้อมูลที่มีจำนวนระดับขั้นค่าสูงมาก เช่น 2^{32} ระดับ เป็นต้น และด้วยเหตุนี้ การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล จึงมีความเหมาะสมกับสัญลักษณ์นอนไบนารี เมื่อเปรียบเทียบกับการถอดรหัสด้วยวิธีการอื่น เช่น วิธีการแบบวิเทอร์บิ ซึ่งไม่เหมาะกับการนำมาใช้งานในทางปฏิบัติ เพราะเมื่อจำนวนระดับขั้นมีค่าสูงมาก จะต้องใช้หน่วยความจำเพิ่มขึ้น และการทำงานมีความซับซ้อนเพิ่มขึ้นเช่นกัน แต่สำหรับการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล สามารถทำได้ดีกว่าโดยที่ยังมีอัตราความผิดพลาดของการถอดรหัสต่ำ

การออกแบบและสร้างระบบการเข้ารหัสถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล ได้ใช้เทคโนโลยีการสร้างวงจรรวม เนื่องจากกระบวนการออกแบบวงจรรวมสามารถจำลองการทำงานของวงจรรวมบนเครื่องคอมพิวเตอร์ได้ จึงช่วยทำให้ลดความผิดพลาดและยังทำให้การพัฒนาวงจรรวมสามารถทำได้อย่างรวดเร็ว กระบวนการที่ว่านี้จะใช้ภาษาบรรยายฮาร์ดแวร์ในการออกแบบการทำงาน และการจำลองการทำงานของวงจรรวม เพื่อให้มั่นใจว่าวงจรรวมที่ออกแบบสามารถทำงานได้อย่างถูกต้อง ก่อนจะนำไปโปรแกรมลงในชิป FPGA สำหรับทดสอบการทำงานจริงต่อไป

เนื่องจากการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลเป็นวิธีถอดรหัสที่ค่อนข้างใหม่ และเท่าที่ได้ศึกษาค้นคว้ามา พบว่ายังไม่มีการสร้างชิ้นงานการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลมาก่อน งานวิจัยนี้จึงได้ออกแบบและสร้างระบบการเข้ารหัสคอนโวลูชันสำหรับสัญลักษณ์นอนไบนารีและระบบการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลสำหรับความผิดพลาดบางแบบที่แก้ไขได้โดยง่าย ด้วยบอร์ด FPGA เพื่อเตรียมความพร้อมเบื้องต้นสำหรับการออกแบบและสร้างชิ้นงานการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล ซึ่งองค์ความรู้ที่ได้จะเป็นประโยชน์สำหรับการพัฒนาชิ้นงานการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลต่อไปในอนาคต

วัตถุประสงค์

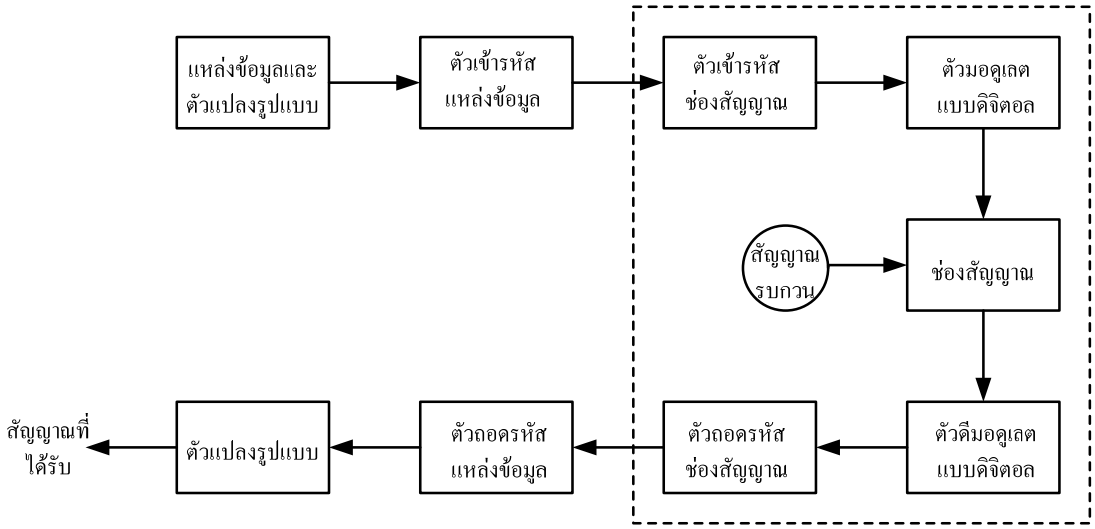
1. เพื่อออกแบบและสร้างระบบการเข้ารหัสคอนโวลูชันสำหรับสัญลักษณ์นอนไบนารีด้วยบอร์ด FPGA โดยสามารถทำการเข้ารหัสสัญลักษณ์นอนไบนารีได้

2. เพื่อออกแบบและสร้างระบบการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลบางส่วนด้วยบอร์ด FPGA โดยสามารถทำการแก้ไขข้อมูลที่มีความผิดพลาดบางส่วนได้

การตรวจเอกสาร

1. บทนำ

การสื่อสารแบบดิจิตอลมีส่วนประกอบหลายส่วน โดยการทำงานเริ่มต้นจากฝั่งส่งจะทำการเปลี่ยนแปลงข้อมูลจากแหล่งข้อมูลให้อยู่ในรูปแบบที่เหมาะสมด้วยการเข้ารหัสแหล่งข้อมูล จากนั้นจะนำข้อมูลที่ได้มาทำการเข้ารหัสช่องสัญญาณเพื่อป้องกันความผิดพลาดที่อาจจะเกิดขึ้นจากสัญญาณรบกวนในขณะที่ส่งผ่านช่องสัญญาณ สำหรับฝั่งรับจะนำข้อมูลที่รับมาทำการถอดรหัสช่องสัญญาณ เพื่อตรวจสอบและแก้ไขความผิดพลาดที่เกิดระหว่างส่งผ่านช่องสัญญาณ ก่อนทำการแปลงข้อมูลให้กลับไปอยู่ในรูปแบบเดิมด้วยวิธีการถอดรหัสแหล่งข้อมูล (Proakis, 2001)



ภาพที่ 1 ไดอะแกรมการสื่อสารแบบดิจิตอล

ส่วนที่ให้ความสนใจศึกษาวิจัยเป็นส่วนที่อยู่ภายในกรอบเส้นประของภาพที่ 1 โดยจะให้ความสำคัญในส่วนของตัวเข้ารหัสถอดรหัสช่องสัญญาณ ซึ่งเป็นส่วนที่นำมาออกแบบและสร้างระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีด้วยบอร์ด FPGA

วิธีการเข้ารหัสช่องสัญญาณที่นิยมใช้มี 2 แบบ คือ การเข้ารหัสแบบบล็อกและการเข้ารหัสแบบคอนโวลูชัน โดยการเข้ารหัสแบบบล็อกจะไม่มีการใช้หน่วยความจำและสามารถสร้างได้โดยใช้วงจรแบบคอมบิเนชัน แต่สำหรับการเข้ารหัสแบบคอนโวลูชันจะมีการใช้หน่วยความจำเพื่อเก็บข้อมูลก่อนหน้าและสามารถสร้างได้โดยใช้วงจรแบบซีเควนเชียล ซึ่งการเข้ารหัสแบบคอนโวลูชันเหมาะกับการนำไปใช้งานในระบบสื่อสารไร้สาย เนื่องจากมีโอกาสเกิดการจางของสัญญาณในช่องสัญญาณและมีผลทำให้เกิดความผิดพลาดเป็นแถบขึ้นได้ (Lin and Costello, 2004)

ตารางที่ 1 สรุปการตรวจเอกสารที่เกี่ยวข้องกับวิทยานิพนธ์

ชื่อเรื่อง	ผู้แต่ง	ปี	รายละเอียด
A general decoding technique applicable to replicated file disagreement location and concatenated code decoding.	Metzner and Kapturowski	1990	เป็นการนำเสนอเทคนิคการถอดรหัสแบบใหม่ที่ใช้กับสัญลักษณ์นอนไบนารี
Vector symbol decoding with list inner symbol decisions.	Metzner	2000	เป็นการนำเสนอการใช้ชุดข้อมูลสำรองช่วยให้การถอดรหัสด้วยวิธีเวกเตอร์ซิม โบลทำได้รวดเร็วและง่ายขึ้น
Vector symbol decoding with list symbol decisions and outer convolutional codes for reliable communications.	Tuntoolavest and Metzner	2002	เป็นการประยุกต์ใช้การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลกับรหัสคอนโวลูชันที่ไม่เป็นระบบสัญลักษณ์นอนไบนารี พร้อมทั้งใช้ชุดข้อมูลสำรอง

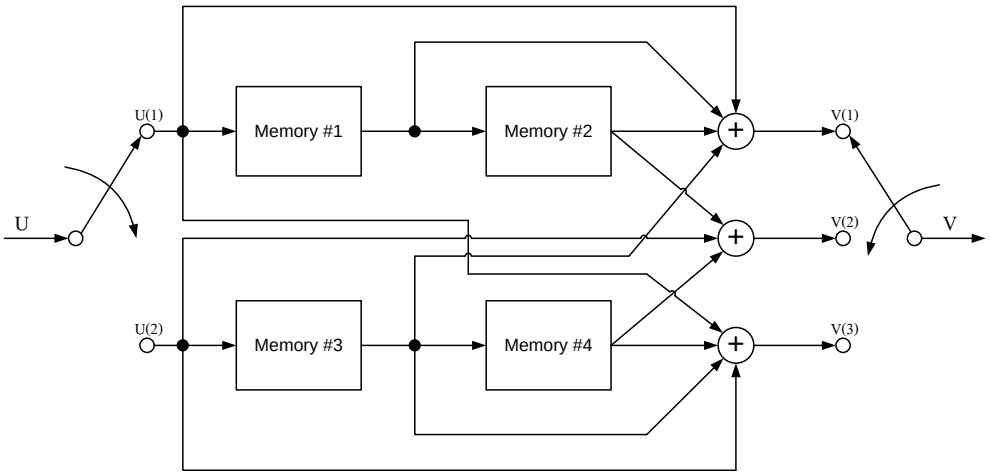
2. รหัสคอนโวลูชัน

2.1 การเข้ารหัสคอนโวลูชันสัญลักษณ์ไบนารี (Tuntoolavest and Intharasakul, 2006)

เพื่ออธิบายการเข้ารหัสแบบคอนโวลูชันสำหรับสัญลักษณ์ไบนารี จะใช้ตัวอย่างรหัสคอนโวลูชัน (3, 2, 2) ซึ่งมีเมตริกฟังก์ชันถ่ายโอน $G(D)$ ดังนี้

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

นั่นคือ ตัวเข้ารหัสจะมีจำนวนเอาต์พุต $n=3$ จำนวนอินพุต $k=2$ และจำนวนหน่วยความจำต่ออินพุต $m=2$ จากสมการที่ (1) สามารถเขียนเป็นลักษณะของไดอะแกรมได้ดังนี้



ภาพที่ 2 ไดอะแกรมของตัวเข้ารหัสแบบคอนโวลูชัน (3, 2, 2)

ถ้ากำหนดให้ข้อมูลลำดับเข้าเป็น **U** และเขียนให้อยู่ในรูปสมการ จะได้ดังนี้

$$U=\left(u_0^{(1)}u_0^{(2)},u_1^{(1)}u_1^{(2)},u_2^{(1)}u_2^{(2)},\ldots\right) \quad (2)$$

สามารถเขียนให้อยู่ในรูปลำดับของอินพุตจำนวน 2 อินพุต คือ

$$\begin{aligned} U^{(1)} &= \left(u_0^{(1)}, u_1^{(1)}, u_2^{(1)}, \ldots\right) \\ U^{(2)} &= \left(u_0^{(2)}, u_1^{(2)}, u_2^{(2)}, \ldots\right) \end{aligned} \tag{3}$$

โดยสามารถเขียน **U** ให้อยู่ในรูปของเมตริกได้เป็น

$$U = \left[U^{(1)} \quad U^{(2)}\right] \tag{4}$$

สำหรับในส่วนของลำดับเอาต์พุตจำนวน 3 เอาต์พุต สามารถเขียนสมการเป็น

$$\begin{aligned} V^{(1)} &= \left(v_0^{(1)}, v_1^{(1)}, v_2^{(1)}, \ldots\right) \\ V^{(2)} &= \left(v_0^{(2)}, v_1^{(2)}, v_2^{(2)}, \ldots\right) \\ V^{(3)} &= \left(v_0^{(3)}, v_1^{(3)}, v_2^{(3)}, \ldots\right) \end{aligned} \tag{5}$$

จากลำดับเอาต์พุตจะทำให้ได้คำรหัส **V** ดังสมการต่อไปนี้

$$V = \left(v_0^{(1)} v_0^{(2)} v_0^{(3)}, v_1^{(1)} v_1^{(2)} v_1^{(3)}, v_2^{(1)} v_2^{(2)} v_2^{(3)}, \ldots\right) \tag{6}$$

ในสมการที่ (1) เนื่องจากเป็นระบบเชิงเส้น จึงสามารถนำวิธีการโอนย้ายโดเมนมาใช้กับวิธีการเข้ารหัสคอนโวลูชันได้ ซึ่งทำให้การคำนวณคำรหัสง่ายขึ้น โดยใช้สมการ

$$V(D) = U(D) \cdot G(D) \tag{7}$$

เมื่อนำสมการที่ (1) และ (4) มาแทนลงใน (7) จะได้

$$V(D) = \left[U^{(1)}(D) \quad U^{(2)}(D)\right] \begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{bmatrix} \tag{8}$$

จากสมการที่ (8) จะได้

$$
\begin{aligned}
V^{(1)}(D) &= U^{(1)}(D) + U^{(1)}(D)\cdot D + U^{(1)}(D)\cdot D^2 + U^{(2)}(D)\cdot D \\
V^{(2)}(D) &= U^{(1)}(D)\cdot D^2 + U^{(2)}(D) + U^{(2)}(D)\cdot D^2 \\
V^{(3)}(D) &= U^{(1)}(D) + U^{(2)}(D) + U^{(2)}(D)\cdot D + U^{(2)}(D)\cdot D^2
\end{aligned} \tag{9}
$$

เนื่องจาก $n = 3$ จึงทำให้ได้คำรหัสเป็น

$$
V(D) = V^{(1)}(D^3) + D\cdot V^{(2)}(D^3) + D^2\cdot V^{(3)}(D^3) \tag{10}
$$

โดยหากเปรียบเทียบสมการที่ (9) กับไดอะแกรมในภาพที่ 2 จะพบว่าหน่วยความจำของไดอะแกรมมีความสัมพันธ์กับสมการ คือ Memory#1 มาจาก $U^{(1)}(D)\cdot D$, Memory#2 มาจาก $U^{(1)}(D)\cdot D^2$, Memory#3 มาจาก $U^{(2)}(D)\cdot D$ และ Memory#4 มาจาก $U^{(2)}(D)\cdot D^2$ สำหรับเครื่องหมายบวกในสมการ คือ ตัวดำเนินการแบบเอ็กซ์คลูซีฟออร์ ซึ่งทั้งหน่วยความจำและตัวดำเนินการแบบเอ็กซ์คลูซีฟออร์ที่ใช้กับสัญลักษณ์ไบนารีสามารถออกแบบและสร้างโดยใช้ไอซีประเภททีทีแอลได้

2.2 การเข้ารหัสคอนโวลูชันสัญลักษณ์นอนไบนารี (Tuntoolavest and Intharasakul, 2006)

การเข้ารหัสคอนโวลูชันสัญลักษณ์นอนไบนารีจะมีความแตกต่างจากการเข้ารหัสคอนโวลูชันสัญลักษณ์ไบนารี ในส่วนขนาดของข้อมูลลำดับ **U** ขนาดของหน่วยความจำ ขนาดของตัวดำเนินการแบบเอ็กซ์คลูซีฟออร์ และขนาดของคำรหัส **V** ยกตัวอย่างเช่น ข้อมูลที่มีขนาด 32 บิตต่อสัญลักษณ์ ข้อมูลลำดับ **U** จะมีขนาด 32 บิต ซึ่งหน่วยความจำที่ใช้สำหรับเก็บข้อมูลก็ต้องใช้ 32 บิตเช่นกัน ซึ่งการคำนวณคำรหัส **V** ก็ต้องใช้ตัวดำเนินการแบบเอ็กซ์คลูซีฟออร์ที่สามารถประมวลผลข้อมูล 32 บิตเช่นกัน หากใช้ไอซีประเภททีทีแอลในการสร้างจะมีความยุ่งยากและซับซ้อนมาก

ถ้ากำหนดให้ข้อมูลลำดับเข้าเป็น **U** และเขียนให้อยู่ในรูปคล้ายสมการที่ (1) แต่มีลำดับสัญลักษณ์อินพุตที่สมมติให้มีขนาด 3 บิตต่อสัญลักษณ์

$$
U = \left(u_0^{(1)}u_1^{(1)}u_2^{(1)}, u_0^{(2)}u_1^{(2)}u_2^{(2)}, u_3^{(1)}u_4^{(1)}u_5^{(1)}, u_3^{(2)}u_4^{(2)}u_5^{(2)}, \ldots\right) \tag{11}
$$

เมื่อแยกลำดับสัญลักษณ์อินพุตออกเป็น 2 ส่วน โดยสามารถนำหลักการในการเข้ารหัสเช่นเดียวกันกับสัญลักษณ์ไบนารีมาใช้กับสัญลักษณ์นอนไบนารีได้

$$\begin{aligned} U^{(1)} &= \left(u_0{}^{(1)} u_1{}^{(1)} u_2{}^{(1)}, u_3{}^{(1)} u_4{}^{(1)} u_5{}^{(1)}, \ldots\right) \\ U^{(2)} &= \left(u_0{}^{(2)} u_1{}^{(2)} u_2{}^{(2)}, u_3{}^{(2)} u_4{}^{(2)} u_5{}^{(2)}, \ldots\right) \end{aligned} \tag{12}$$

3. การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล

3.1 ความเป็นมา

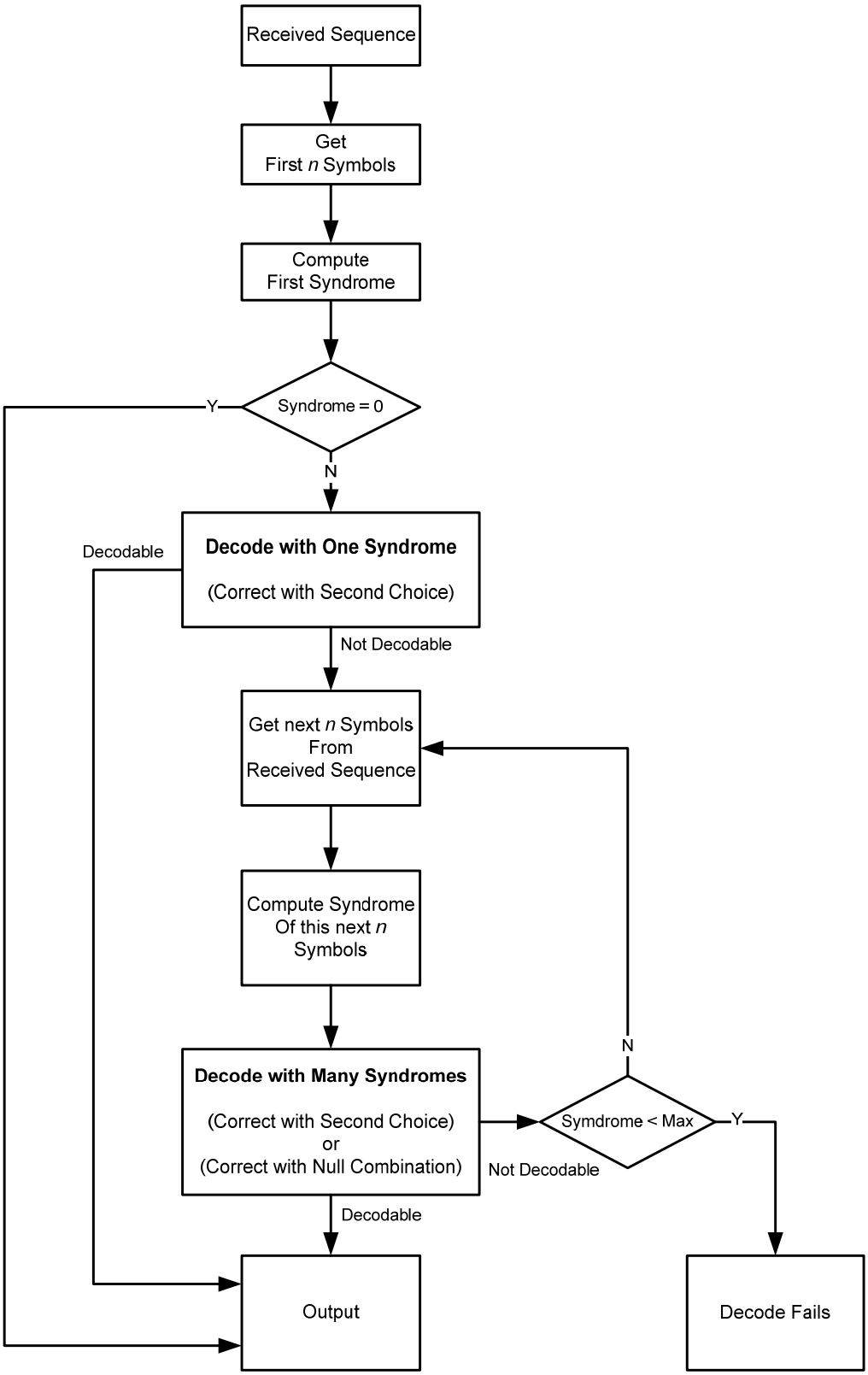
การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลคิดค้นในปี ค.ศ. 1990 (Metzner and Kapturowski, 1990) และถูกค้นพบอีกครั้งโดย Haslash and Vinck ในปี ค.ศ. 1999 ในการถอดรหัสที่นำเสนอครั้งแรกจะเน้นการถอดรหัสแบบบล็อกและต่อมาได้มีการนำชุดข้อมูลสำรองมาใช้ (Metzner, 2000) โดยวิธีการถอดรหัสนี้เป็นวิธีการถอดรหัสที่ใช้งานกับสัญลักษณ์นอนไบนารี สามารถใช้ถอดรหัสที่มีการเข้ารหัสเป็นแบบบล็อกหรือแบบคอนโวลูชันได้ สำหรับการประยุกต์ใช้กับรหัสคอนโวลูชันที่เป็นระบบถูกเสนอโดย Seo ในปี ค.ศ. 1991 แต่ยังไม่มีการนำชุดข้อมูลสำรองมาใช้ ซึ่งต่อมามีการประยุกต์ใช้กับรหัสคอนโวลูชันที่ไม่เป็นระบบพร้อมกับการนำชุดข้อมูลสำรองมาใช้ (Tuntoolavest and Metzner, 2002)

3.2 ขั้นตอนการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล สำหรับรหัสคอนโวลูชันที่มีอัตรา $(n-1)/n$ และมีจำนวนรายการตัวเลือก 2 สรุปได้ดังนี้

3.2.1 ตัวถอดรหัสเริ่มรับข้อมูลสัญลักษณ์เข้ามาเป็นลำดับ และคำนวณหาค่าซินโดรมจาก n สัญลักษณ์แรกที่รับเข้ามา เช่น รหัสคอนโวลูชันที่มีพารามิเตอร์ (3, 2, 2) จะนำ 3 สัญลักษณ์แรกที่รับเข้ามาใช้ในการคำนวณหาค่าซินโดรม ถ้าค่าซินโดรมที่คำนวณได้มีค่าเป็นศูนย์ แสดงว่าสัญลักษณ์ที่รับเข้ามานั้นไม่มีความผิดพลาด แล้วจึงทำการคำนวณหาค่าซินโดรมในสัญลักษณ์ n สัญลักษณ์ถัดไป หากคำนวณค่าซินโดรมของสัญลักษณ์ทั้งหมดแล้วค่าซินโดรมยังเป็นศูนย์ ก็แสดงว่าในการส่งข้อมูลไม่มีความผิดพลาดเกิดขึ้นเลย

3.2.2 ถ้าตัวถอดรหัสคำนวณค่าซินโดรมจาก n สัญลักษณ์แรกที่รับเข้ามาแล้วมีค่าไม่เท่ากับศูนย์ ก็จะพยายามแก้ไขข้อมูลใน n สัญลักษณ์แรกที่รับเข้ามาให้ถูกต้อง โดยใช้วิธีถอดรหัสด้วยซินโดรมเดียว (Decode with One Syndrome) ซึ่งวิธีนี้จะแก้ไขได้เฉพาะกรณีที่เกิดความผิดพลาดเพียงสัญลักษณ์เดียวจาก n สัญลักษณ์แรกที่รับเข้ามา และในตำแหน่งของตัวเลือกที่ 2 ที่ตรงกับตัวเลือกแรกจะต้องไม่มีความผิดพลาด

3.2.3 ถ้าตัวถอดรหัสไม่สามารถแก้ไขความผิดพลาดโดยวิธีถอดรหัสด้วยซินโดรมเดียว ก็จะทำการคำนวณค่าซินโดรมเพิ่มขึ้นโดยคำนวณจาก n สัญลักษณ์ถัดไป แล้วใช้วิธีถอดรหัสด้วยซินโดรมหลายตัว (Decode with Many Syndromes) โดยเริ่มต้นที่ค่าซินโดรมมากกว่าหรือเท่ากับ 2 ไปจนถึงค่ามากที่สุดที่กำหนด และขั้นตอนการทำงานเริ่มใช้ค่าซินโดรมเท่ากับ 2 มาใช้แก้ไขก่อนด้วยวิธี Correct with Second Choice ถ้าแก้ไขได้ก็ไปทำที่ตำแหน่งผิดพลาดถัดไป แต่ถ้าแก้ไขยังไม่ได้ก็จะใช้วิธี Correct with Null Combination ถ้าแก้ไขได้ก็ไปทำที่ตำแหน่งผิดพลาดถัดไป แต่ถ้าหากยังไม่สามารถแก้ไขความผิดพลาดได้ก็จะเพิ่มจำนวนซินโดรมเข้าไปทีละหนึ่งค่า แล้วก็เริ่มแก้ไขด้วยวิธี Correct with Second Choice ก่อนเหมือนกับที่ค่าซินโดรมเป็น 2 โดยทำอย่างนี้ไปเรื่อยๆ จนกว่าค่าซินโดรมเพิ่มขึ้นถึงค่าซินโดรมที่กำหนด ถ้าหากยังไม่สามารถแก้ไขความผิดพลาดได้ ตัวถอดรหัสก็จะปล่อยให้สัญลักษณ์นั้นเกิดความผิดพลาดไป แล้วรายงานว่าเกิดความล้มเหลวในการถอดรหัส



ภาพที่ 3 ผังงานขั้นตอนการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล

ถอดรหัสด้วยซินโดรมเดียวเป็นขั้นตอนหนึ่งของการถอดรหัสด้วยวิธีแก้ไขด้วยตัวเลือกที่สอง โดยการใช้ซินโดรมเมตริกที่มีจำนวนซินโดรมเพียงตัวเดียวเท่านั้น ในการถอดรหัสจะพิจารณารหัสคอนโวลูชัน (3, 2, 2) ซึ่งมีเมตริกตรวจสอบพาริตี้ตามสมการที่ (13)

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & & & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & & & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & & & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \qquad (13)$$

เมื่อรับข้อมูลชุดสัญลักษณ์แรกเข้ามาครบ 3 สัญลักษณ์ โดยสมมติให้เป็น **a**, **b** และ **c** แต่มีความผิดพลาดเท่ากับ **e2** เกิดขึ้น ทำให้ชุดสัญลักษณ์แรกจากชุดข้อมูลหลักเป็น **a**, $\mathbf{b} \oplus \mathbf{e2}$ และ **c** ตามลำดับ โดยแต่ละตัวเป็นเมตริกของสัญลักษณ์ขนาด 32 บิต และ $\oplus$ เป็นการบวกแบบมอดูโล 2 หรือการทำ XOR และสมมติว่าสัญลักษณ์ชุดแรกจากชุดข้อมูลสำรองเป็น $\mathbf{a} \oplus \mathbf{e1}$, **b** และ $\mathbf{c} \oplus \mathbf{e3}$ ซึ่งถ้าตัวถอดรหัสทำงานได้ถูกต้อง จะต้องนำสัญลักษณ์ตัวที่สองจากชุดข้อมูลสำรองไปแทนที่สัญลักษณ์ตัวที่สองในชุดข้อมูลหลัก เพื่อทำให้ชุดข้อมูลหลักถูกต้องทุกสัญลักษณ์

วิธีถอดรหัสด้วยซินโดรมเดียวจะทำการคำนวณหาค่าซินโดรมของสัญลักษณ์ที่เข้ามาโดยนำชุดสัญลักษณ์แรกจากชุดข้อมูลหลักไปคูณกับเมตริกตรวจสอบพาริตี้ที่ใช้กับชุดข้อมูลชุดแรกซึ่งมีค่าเป็นสมการที่ (14)

$$\mathrm{h}_1 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \qquad (14)$$

โดยจะได้ซินโดรมตัวแรกของชุดข้อมูลหลักเป็น

$$\mathrm{S}_1 = \mathrm{HY} = \mathrm{h}_1\mathrm{Y} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{a} & (\mathrm{b} \oplus \mathrm{e2}) & \mathrm{c} \end{bmatrix}^{\mathrm{T}} = \mathrm{a} \oplus \mathrm{b} \oplus \mathrm{e2} \oplus \mathrm{c} \qquad (15)$$

ทฤษฎีพื้นฐานของการถอดรหัสกล่าวว่า เมตริกตรวจสอบพาริตี้คูณกับคำรหัสที่ไม่มีความผิดพลาดแล้วจะได้เท่ากับเวกเตอร์ศูนย์เสมอ ดังนั้น

$$HV = h_1 V = [1 \quad 1 \quad 1][a \quad b \quad c]^T = a \oplus b \oplus c = 0 \tag{16}$$

จากสมการที่ (16) จะได้ค่าซินโดรมของความผิดพลาดที่เกิดขึ้นดังสมการที่ (17)

$$S_1 = e2 \tag{17}$$

จากนั้นจะทำการคำนวณผลต่างระหว่างสัญลักษณ์ของชุดข้อมูลหลัก และชุดข้อมูลสำรอง โดยทำการ XOR กัน ซึ่งจะพบว่าผลต่างของสัญลักษณ์ทั้งสามตำแหน่งจะเป็น **e1**, **e2** และ **e3** ตามลำดับ แล้วนำค่าผลต่างที่ได้มาเปรียบเทียบกับค่าซินโดรมที่คำนวณได้เพื่อหาค่าดัชนีบอกตำแหน่งของสัญลักษณ์ชุดข้อมูลหลักที่มีความผิดพลาด เมื่อทราบตำแหน่งแล้ว จะนำผลต่างที่ได้มาทำการแก้ไขสัญลักษณ์ชุดข้อมูลหลักให้ถูกต้องโดยใช้การทำ XOR กันระหว่างผลต่างกับสัญลักษณ์ชุดข้อมูลหลัก

การถอดรหัสด้วยวิธี Correct with Second choice สำหรับซินโดรมที่มีค่ามากกว่าหนึ่ง เรียกว่า Correct with Null Combination นั้น เกินจากขอบเขตของวิทยานิพนธ์นี้ เนื่องจากวิทยานิพนธ์นี้ เน้นที่การออกแบบระบบเพื่อพัฒนาชิ้นงาน จึงจะทดสอบการทำงานของระบบโดยใช้การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลที่ใช้เพียงซินโดรมเดียวเท่านั้น สำหรับผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมได้จากเอกสารในภาคผนวก ข

4. การออกแบบอุปกรณ์ดิจิตอลที่มีความซับซ้อน

4.1 ขั้นตอนการออกแบบ FPGA

การออกแบบวงจรด้วย FPGA (Field-Programmable Gate Array) จะเริ่มต้นจากการสร้างข้อกำหนดของการออกแบบ ซึ่งเป็นการสร้างข้อกำหนดต่างๆ ของวงจร เช่น กำหนดค่าความถี่ที่ใช้งาน กำหนดฟังก์ชันการทำงาน เป็นต้น ขั้นตอนต่อมาคือ การจำลองการทำงานโมเดลวงจรระดับฟังก์ชัน ซึ่งเป็นการตรวจสอบการทำงานของโมเดลวงจร หลังจากนั้นจะทำการสังเคราะห์

และออปติไมซ์วงจร ซึ่งเป็นการสร้างแผนภาพวงจรจากโมเดลวงจรระดับฟังก์ชันให้อยู่ในรูปของลอจิกเกต โดยต้องอาศัยซอฟต์แวร์ช่วยในการสังเคราะห์วงจร ต้องเลือกใช้เทคโนโลยี FPGA ที่ผู้ออกแบบต้องการเลือกใช้ ซึ่งขึ้นอยู่กับบริษัทผู้ผลิต FPGA จัดเตรียมซอฟต์แวร์ไว้ให้พัฒนาชิป FPGA เมื่อสังเคราะห์เรียบร้อยแล้วขั้นตอนต่อไปคือ การจำลองการทำงานของวงจรในระดับลอจิกเกต โดยต้องคำนึงถึงเรื่องเกตดีเลย์ด้วย ต้องมีการตรวจสอบค่าเวลาในการทำงานว่าเป็นไปตามที่ต้องการหรือไม่ จากนั้นจะเป็นขั้นตอนของการวางและเชื่อมต่อเซลล์ภายในของ FPGA และหลังจากนั้นก็จะต้องทำการจำลองการทำงานของวงจรระดับฐานเวลาจริงอีกครั้ง ซึ่งเป็นขั้นตอนสุดท้ายของการตรวจสอบความถูกต้อง ก่อนที่จะนำวงจรที่ออกแบบไปโปรแกรมลงสู่ชิป FPGA ต่อไปเพื่อทดสอบการทำงานของวงจรที่ออกแบบ (ชำนาญ และ วัชรากร, 2547)

4.2 ภาษา VHDL

ภาษา VHDL (Very High Speed Integrated Circuit HDL) เป็นภาษาที่พัฒนาขึ้นโดยกระทรวงกลาโหมของสหรัฐอเมริกา เพื่อให้สามารถใช้ในการออกแบบวงจรที่มีความซับซ้อน และให้เป็นภาษามาตรฐานที่เป็นสากล ข้อดีของภาษา VHDL คือ เป็นภาษาที่มีความยืดหยุ่นในการพัฒนา ไม่ยึดติดอยู่กับซอฟต์แวร์ที่ใช้ในการออกแบบ สามารถเปลี่ยนแปลงแก้ไขวงจรได้ง่าย

การออกแบบหน่วยควบคุมจะใช้การออกแบบ FSM ซึ่งปกติแล้วจะบรรยายด้วยแผนภาพไดอะแกรมที่แสดงการเปลี่ยนแปลงสเตตการทำงานของวงจร โดยพื้นฐานของการออกแบบ FSM จะมี 2 รูปแบบ คือ แบบ Mealy Machine และแบบ Moore Machine ซึ่งรูปแบบที่ใช้ในการออกแบบระบบการเข้ารหัสถอดรหัสเป็นแบบผสมทั้ง 2 แบบ (Douglas, 2002)

5. ระบบบัส USB

5.1 ความเป็นมา

ในปัจจุบันนิยมใช้การเชื่อมต่อด้วยระบบบัส USB เนื่องจากใช้งานสะดวก และยังส่งถ่ายข้อมูลได้รวดเร็วเมื่อเทียบกับการเชื่อมต่อแบบขนานและแบบอนุกรม ซึ่งจุดเริ่มต้นของระบบบัส USB เกิดจากการที่นักพัฒนาระบบการเชื่อมต่อได้ร่วมมือกันสร้างมาตรฐานและจัดการประเด็นปัญหาต่างๆ ที่พบในการพัฒนาระบบการเชื่อมต่อ เพื่อแก้ไขปัญหาระบบการเชื่อมต่อมีหลากหลาย

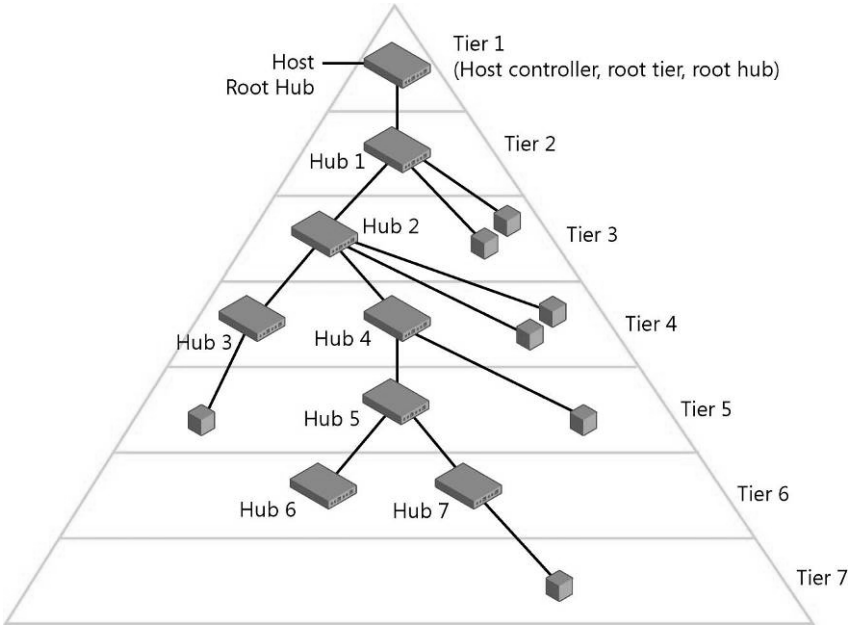
และไม่เป็นมาตรฐาน โดยอุปกรณ์ USB มีโหมดความเร็วของการส่งถ่ายข้อมูล 3 แบบ คือ แบบโลว์สปีด (Low Speed) ที่มีอัตราการส่งถ่ายข้อมูลเป็น 1.5 เมกะบิตต่อวินาที, แบบฟูลสปีด (Full Speed) ที่มีอัตราการส่งถ่ายข้อมูลเป็น 12 เมกะบิตต่อวินาที และแบบไฮสปีด (High Speed) ที่มีอัตราการส่งถ่ายข้อมูลเป็น 480 เมกะบิตต่อวินาที แต่ในทางปฏิบัติแล้วอัตราการส่งถ่ายข้อมูลจะมีค่าต่ำกว่าค่าที่กำหนดไว้ข้างต้น เนื่องจากระบบบัสมีการส่งถ่ายสัญญาณข้อมูลจำนวนมากและมีการแบ่งช่องสัญญาณกรณีที่มีอุปกรณ์รอบข้างหลายตัวเชื่อมต่ออยู่ภายในระบบบัส (วรเทพ และ บุญอนันต์, ม.ป.ป.)

5.2 ภาพรวมสถาปัตยกรรม

ระบบบัส USB มีองค์ประกอบที่สำคัญดังต่อไปนี้

5.2.1 พอร์ต (Port) หมายถึง คอนเนคเตอร์แต่ละตัวที่อยู่บนระบบบัส ถึงแม้จะมีพอร์ตหลายพอร์ตอยู่ในระบบบัส แต่จะมีเส้นทางข้อมูลเพียงเส้นเดียวที่ใช้งาน

5.2.2 โทโพโลยี (Topology) หมายถึง การจัดรูปแบบการเชื่อมต่อบนระบบบัส USB ซึ่งจะอยู่ในรูปแบบสตาร์แบบลำดับชั้น การเชื่อมต่อประกอบไปด้วยโครงสร้างแบบสตาร์หลายๆ ชุดเชื่อมต่อกัน เนื่องจากการหน่วงเวลาของสัญญาณที่เกิดขึ้นภายในสายเคเบิลและฮับ ทำให้จำนวนชั้นสูงสุดที่ยอมให้ใช้ได้ภายในโครงสร้างนี้จะมีทั้งหมดเพียง 7 ชั้น รวมชั้นที่เป็นโฮส



ภาพที่ 4 โทโพโลยีของระบบบัส USB

ที่มา: วรเทพ และ บุญอนันต์ (ม.ป.ป.)

5.2.3 โฮส USB (Host) คือ เครื่องคอมพิวเตอร์ที่ประกอบด้วยโฮสคอนโทรลเลอร์และรูทฮับอยู่ภายใน ซึ่งทั้งสองตัวนี้จะทำงานร่วมกัน เพื่อทำให้ระบบปฏิบัติการสามารถสื่อสารกับอุปกรณ์บนระบบบัสได้

5.2.4 ฮับ USB (Hub) จะมีพอร์ตที่ใช้สำหรับเชื่อมต่อกับอุปกรณ์ USB อยู่หนึ่งพอร์ตหรือมากกว่านั้น หน้าที่หลัก คือ การเป็นตัวทวนสัญญาณ และช่วยจัดการเรื่องความเร็วระหว่างโหมดที่ต่างกัน

5.2.5 อุปกรณ์ USB หรืออุปกรณ์รอบข้าง USB (Devices) เป็นสิ่งที่นำมาเชื่อมต่อเข้ากับพอร์ต USB บนเครื่องคอมพิวเตอร์หรือฮับ อาจเรียกอีกชื่อว่าฟังก์ชัน หน้าที่หลัก ได้แก่ ตรวจจับการสื่อสารที่เกิดขึ้น, ตอบสนองคำร้องขอมาตรฐาน, ตรวจสอบความผิดพลาด, จัดการพลังงาน และแลกเปลี่ยนข้อมูลกับโฮส

5.2.6 คอนเนคเตอร์ (Connector) ใช้สำหรับเชื่อมต่อ ซึ่งมีคำศัพท์ที่เกี่ยวข้อง 2 คำ ระบุไว้ในข้อกำหนด USB ได้แก่ อัพสตรีม และดาวน์สตรีม โดยอัพสตรีม คือ การไหลของข้อมูลที่มีทิศทางไปยังโฮสคอมพิวเตอร์ และดาวน์สตรีม คือ การไหลของข้อมูลที่มีทิศทางออกไปจากโฮสคอมพิวเตอร์ ลักษณะของอัพสตรีมพอร์ตจะเป็นพอร์ตที่อยู่บนตัวอุปกรณ์และรับข้อมูลดาวน์สตรีม ส่วนดาวน์สตรีมพอร์ตจะเป็นพอร์ตที่อยู่บนโฮสคอมพิวเตอร์และรับข้อมูลอัพสตรีม

5.2.7 โพรโตคอล (Protocol) สร้างขึ้นจากโพรโตคอลที่ค่อนข้างซับซ้อนและมีข้อกำหนดที่เคร่งครัด แต่ในทางปฏิบัติโพรโตคอลจะถูกจัดการด้วยไอซีคอนโทรลเลอร์ USB

5.2.8 เอนด์พอยน์ (Endpoint) คือ บัฟเฟอร์ซึ่งใช้สำหรับรับหรือส่งข้อมูล โดยเอนด์พอยน์จะเกิดขึ้นที่จุดสิ้นสุดของเส้นทางการสื่อสาร

5.2.9 ไปป์ (Pipe) เป็นการเชื่อมต่อทางลอจิกระหว่างโฮสและเอนด์พอยน์ โดยขณะอุปกรณ์ส่งหรือรับข้อมูลบนเอนด์พอยน์ ซอฟต์แวร์จะทำการรับส่งข้อมูลผ่านทางไปป์

อุปกรณ์และวิธีการ

อุปกรณ์

1. เครื่องคอมพิวเตอร์ (Personal Computer)
2. บอร์ดทดลอง FPGA รุ่น FPGA Discovery-III XC3S200F4
3. โมดูล USB รุ่น Ezy USB-M01
4. เครื่องมือวัดมัลติมิเตอร์
5. สายแพ (Ribbon Cable)
6. คอนเนคเตอร์สำหรับเชื่อมต่อขาสัญญาณ

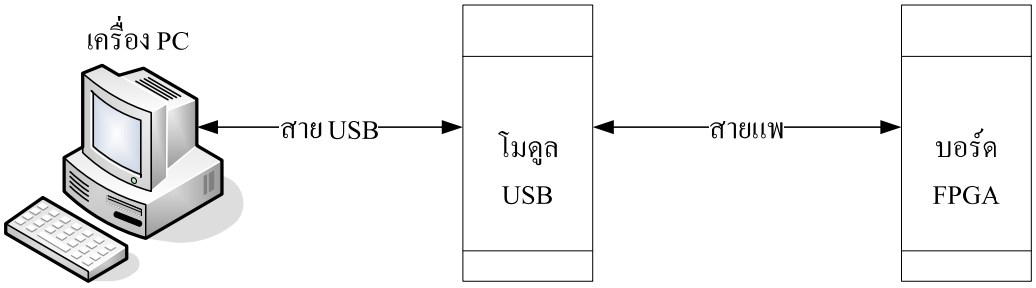
วิธีการ

1. การออกแบบระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี

การออกแบบระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีในภาพรวม จะเริ่มต้นการทำงานโดยให้เครื่อง PC ส่งข้อมูลจากแฟ้มข้อมูลผ่านทางโมดูล USB ไปยังบอร์ด FPGA เพื่อทำการประมวลผลการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี และเมื่อบอร์ด FPGA ประมวลผลเสร็จ ก็จะส่งข้อมูลผ่านทางโมดูล USB กลับไปยังเครื่อง PC เพื่อจัดเก็บข้อมูลที่ผ่านการประมวลผลด้วยบอร์ด FPGA ให้อยู่ในรูปแบบแฟ้มข้อมูล ซึ่งใช้สำหรับตรวจสอบความถูกต้องในการทำงานของระบบการเข้ารหัสถอดรหัสที่ได้ออกแบบ

การเชื่อมต่อประสานอุปกรณ์ต่างๆ เข้าด้วยกันต้องคำนึงถึงอุปกรณ์ที่เกี่ยวข้องทั้งหมดและรวมถึงการพัฒนาโปรแกรมคอมพิวเตอร์สำหรับการติดต่อสื่อสารกับอุปกรณ์ โดยมีองค์ประกอบที่สำคัญ คือ โมดูล USB ใช้สำหรับทำหน้าที่สื่อสารข้อมูลของอุปกรณ์ผ่านระบบบัส USB, บอร์ด FPGA ที่มีการโปรแกรมให้ทำหน้าที่รับส่งข้อมูลและทำหน้าที่เข้ารหัสถอดรหัส, เครื่อง PC ที่สนับสนุนการสื่อสารข้อมูลผ่านระบบบัส USB และมีการติดตั้งไดร์เวอร์ไว้, ซอฟต์แวร์สำหรับ

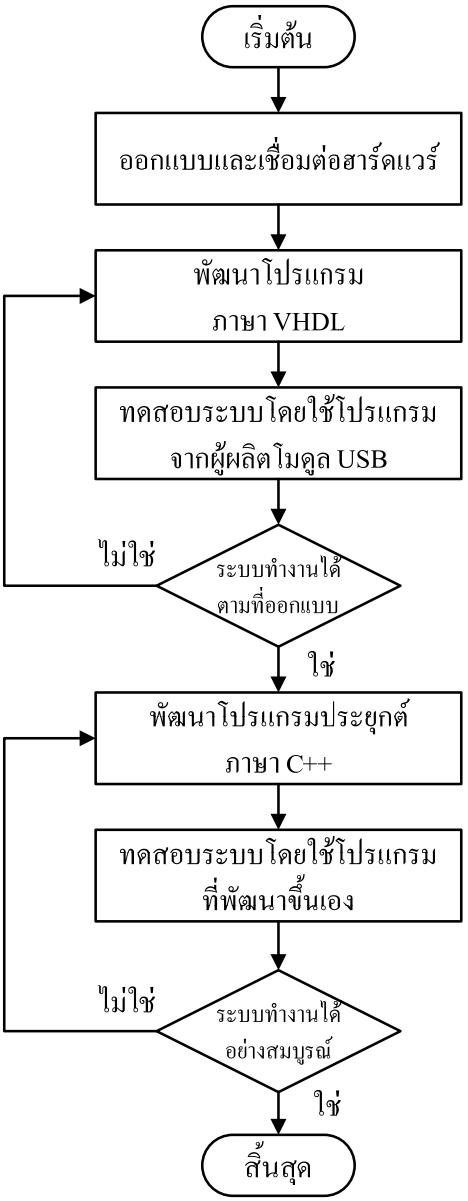
เขียนโปรแกรมภาษา VHDL, เครื่องโปรแกรมบอร์ด FPGA, ซอฟต์แวร์สำหรับเขียนโปรแกรมภาษา C++ และซอฟต์แวร์อื่นๆ เช่น Notepad เป็นต้น



ภาพที่ 5 การเชื่อมต่อประสานอุปกรณ์

ขั้นตอนการพัฒนาระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี จะเริ่มต้นจากการออกแบบฮาร์ดแวร์ของระบบ โดยศึกษาการใช้งานบอร์ด FPGA และโมดูล USB จากคู่มือ เพื่อทำความเข้าใจการทำงานและสามารถนำมาประยุกต์ใช้งานได้อย่างถูกต้อง เมื่อเข้าใจการทำงานแล้ว ขั้นตอนต่อไปก็จะทำการเชื่อมต่อประสานบอร์ด FPGA และโมดูล USB เข้าด้วยกัน ซึ่งในขั้นตอนนี้จะต้องทำการออกแบบการเชื่อมต่อประสานโดยการกำหนดขาสัญญาณของบอร์ด FPGA ที่ใช้งาน เนื่องจากขาสัญญาณของบอร์ด FPGA มีความสัมพันธ์กับการพัฒนาโปรแกรมภาษา VHDL

เมื่อออกแบบฮาร์ดแวร์ของระบบเรียบร้อยแล้ว ในขั้นต่อไปจะออกแบบซอฟต์แวร์ของระบบ ในขั้นต้นต้องออกแบบโปรแกรมภาษา VHDL เพื่อใช้สำหรับการติดต่อสื่อสารระหว่างเครื่อง PC และบอร์ด FPGA ก่อน โดยเริ่มออกแบบการรับข้อมูลจากเครื่อง PC เข้าสู่บอร์ด FPGA เพียงอย่างเดียว, การส่งข้อมูลจากบอร์ด FPGA ไปยังเครื่อง PC เพียงอย่างเดียว และการรับส่งข้อมูลระหว่างบอร์ด FPGA กับเครื่อง PC ที่มีความซับซ้อนยิ่งขึ้น เป็นต้น ซึ่งขั้นตอนนี้ยังไม่จำเป็นต้องออกแบบโปรแกรมประยุกต์ภาษา C++ ขึ้นมาใช้งาน เนื่องจากสามารถใช้ตัวอย่างโปรแกรมประยุกต์ที่ได้จากบริษัทผู้ผลิตโมดูล USB สำหรับทดสอบการทำงานได้ โดยหลังจากที่ทำให้เครื่อง PC และบอร์ด FPGA สามารถติดต่อสื่อสารกันได้แล้ว ขั้นตอนต่อไปจะเป็นการพัฒนาโปรแกรมภาษา VHDL ให้สามารถทำการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีได้ แล้วจึงทำการพัฒนาโปรแกรมประยุกต์ด้วยภาษา C++ ขึ้นมาเพื่อใช้ในการทดสอบการทำงานของระบบเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีที่ออกแบบในภายหลัง



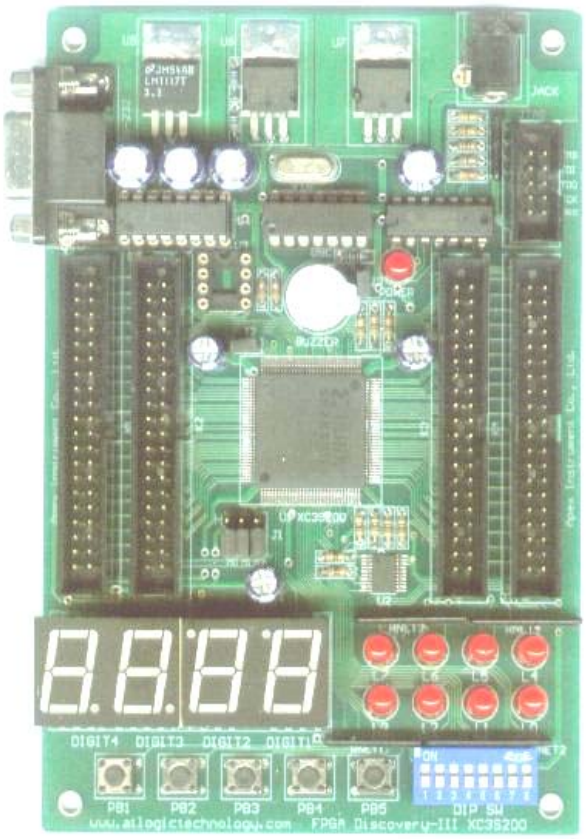
ภาพที่ 6 ผังงานการออกแบบและพัฒนาระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี

2. การออกแบบฮาร์ดแวร์ของระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี

2.1 บอร์ด FPGA

ในการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์นั้น หากนำอุปกรณ์ไอซีประเภท TTL มาใช้ในการสร้างจะทำให้ฮาร์ดแวร์ของระบบมีขนาดใหญ่ และอาจไม่

สามารถประมวลผลฟังก์ชันที่มีลักษณะการทำงานซับซ้อนมากได้ ดังนั้น จึงต้องอาศัยเทคโนโลยีการสร้างวงจรรวมมาใช้ในการออกแบบและสร้างระบบ โดยเลือกใช้ชิป FPGA เป็นอุปกรณ์หลัก เนื่องจากชิป FPGA สามารถโปรแกรมให้ทำงานเป็นวงจรดิจิตอลตามความต้องการ ยังสามารถบรรจุวงจรดิจิตอลที่มีขนาดใหญ่และมีลักษณะการทำงานซับซ้อนมากได้ โดยสามารถทำงานได้อย่างรวดเร็ว และมีความสะดวกในการแก้ไขการทำงานของวงจรดิจิตอลด้วยการโปรแกรมซ้ำ

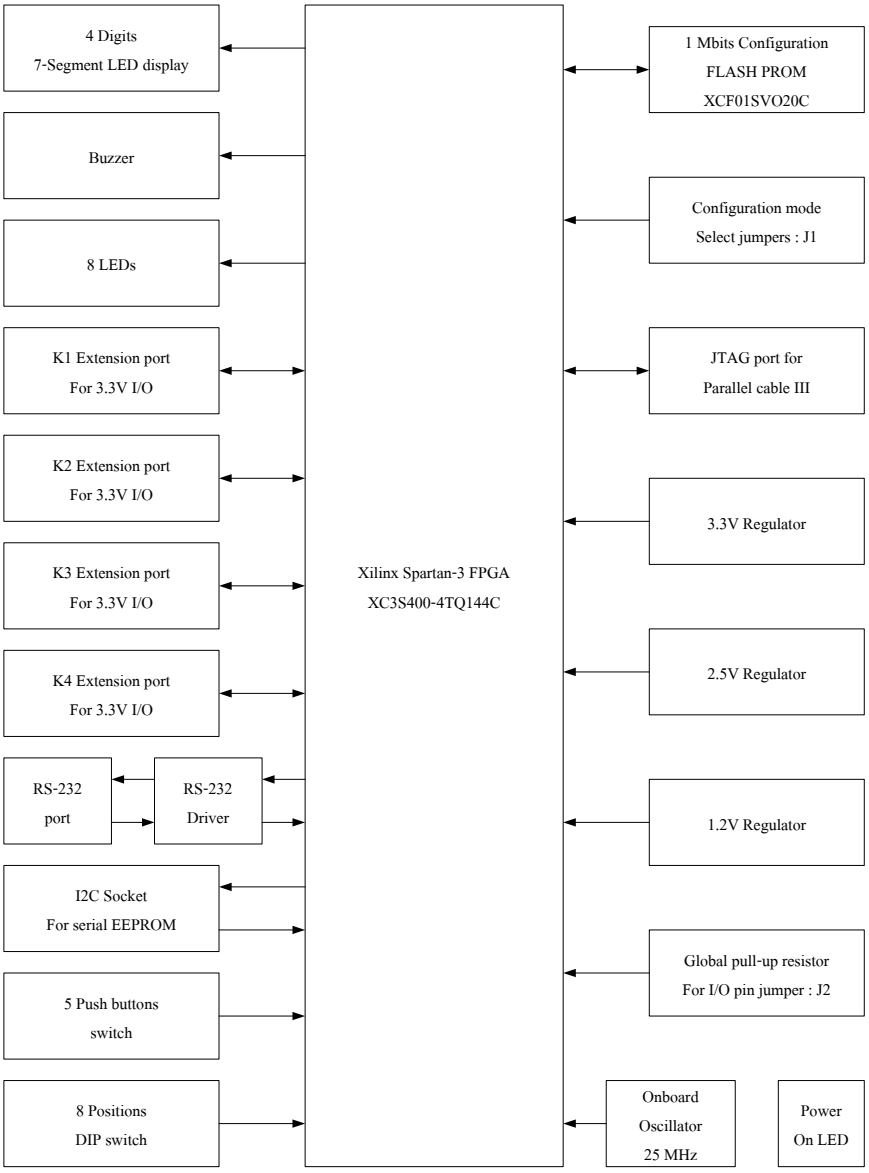


ภาพที่ 7 บอร์ด FPGA รุ่น FPGA Discovery-III XC3S200F4

ที่มา: APEX Instrument (n.d.)

บอร์ด FPGA ที่นำมาใช้สร้างระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีเป็นบอร์ดทดลองอเนกประสงค์รุ่น FPGA Discovery-III XC3S200F4 ของบริษัท เอเพก อินสตรูเมนส์ จำกัด ซึ่งใช้ชิป FPGA ตระกูล Spatan-3 ของบริษัท Xilinx เบอร์ XC3S400-4TQ144C ที่มีขนาด

ความจุวงจรเท่ากับ 400,000 เกต ใช้งานร่วมกับ Platform Flash PROM เบอร์ XCF01VO20C ซึ่งสาเหตุที่ต้องใช้ Platform Flash PROM เนื่องจากชิป FPGA ไม่สามารถเก็บค่าข้อมูลวงจรในขณะที่ไม่มีไฟจ่ายให้ชิปได้ จึงต้องอาศัย Platform Flash PROM เป็นตัวเก็บค่าข้อมูลวงจรไว้ โดยเมื่อเริ่มจ่ายไฟเลี้ยงให้กับบอร์ด FPGA ค่าข้อมูลวงจรที่ถูกโปรแกรมไว้ใน Platform Flash PROM จะถูกโหลดเข้าไปในชิป FPGA อย่างอัตโนมัติ ก่อนเริ่มการทำงานทุกครั้ง



ภาพที่ 8 บล็อกไดอะแกรมส่วนประกอบของบอร์ด FPGA Discovery-III XC3S200F4

ที่มา: APEX Instrument (n.d.)

อุปกรณ์อินพุต/เอาต์พุตและพอร์ตต่างๆ ของบอร์ด FPGA มีรายละเอียดดังตารางผนวกที่ ก1 สำหรับการนำมาใช้งานจะต้องคำนึงถึงระดับแรงดันไฟฟ้าด้วย เนื่องจากบอร์ด FPGA ทำงานที่ระดับแรงดันไฟฟ้า 3.3 โวลต์ ซึ่งสามารถนำสัญญาณเอาต์พุตของบอร์ด FPGA ไปใช้กับอุปกรณ์ภายนอกที่มีอินพุตระดับ 3.3 โวลต์และ 5 โวลต์ได้ แต่ไม่สามารถนำเอาต์พุตจากอุปกรณ์ภายนอกที่มีระดับ 5 โวลต์มาใช้งานกับบอร์ด FPGA เนื่องจากไม่สามารถรับระดับแรงดันที่มีค่าสูงกว่าได้

การเชื่อมต่อประสานบอร์ด FPGA กับโมดูล USB จะเลือกใช้คอนเนคเตอร์ K2 ที่ติดตั้งอยู่บนบอร์ด FPGA มีรายละเอียดดังตารางผนวกที่ ก2 ซึ่งจะพบว่ามีขาของคอนเนคเตอร์ K2 บางขาที่ถูกใช้งานไปแล้ว เช่น ขาที่ 1, 3, 5 และ 7 ได้ถูกนำไปใช้งานเป็นขา Common Cathode ของ 7-Segment ดังนั้น จึงต้องเลือกขาที่ว่างไปใช้ในการเชื่อมต่อประสาน ได้แก่ ขาที่ 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29 และ 31 รวมทั้งสิ้น 12 ขา ซึ่งเพียงพอกับจำนวนขาสัญญาณของโมดูล USB ที่ต้องการใช้งาน ส่วนสวิตช์กดติดปล่อยดับ PB5 จะใช้งานเพื่อเป็นปุ่มรีเซตการทำงานของระบบ โดยมีลักษณะการทำงานแบบ Active Low คือ หากไม่มีการกดสวิตช์ สัญญาณที่ขานี้จะมีสถานะเป็นลอจิกสูง และหากมีการกดสวิตช์ สัญญาณที่ขานี้จะมีสถานะเป็นลอจิกต่ำ ซึ่งมีตำแหน่งขาตรงกับขา P51 ของชิป FPGA และเนื่องจากการทำงานของระบบมีการประมวลผลในลักษณะเป็นลำดับต่อเนื่อง ซึ่งจำเป็นต้องใช้สัญญาณนาฬิกาในการประมวลผล โดยบอร์ด FPGA ได้ติดตั้งอุปกรณ์ออสซิลเลเตอร์ที่ผลิตสัญญาณนาฬิกาความถี่ 25 เมกะเฮริตซ์ไว้ให้แล้ว ซึ่งมีตำแหน่งขาตรงกับขา P127 ของชิป FPGA

การนำค่าข้อมูลวงจรโปรแกรมลงในชิป FPGA จะต้องใช้คอนเนคเตอร์ JTAG เชื่อมต่อกับสายดาวน์โหลด JTAG และเชื่อมต่อเข้ากับพอร์ตขนานของเครื่อง PC ซึ่งบอร์ด FPGA ได้ออกแบบให้สามารถโปรแกรม Platform Flash PROM และชิป FPGA ผ่านสายดาวน์โหลด JTAG และสามารถตั้งค่าโหมดการทำงานให้เป็นแบบ Master Serial เพื่อให้ Platform Flash PROM ทำการโหลดค่าข้อมูลวงจรลงในชิป FPGA อย่างอัตโนมัติทุกครั้งที่จ่ายไฟเลี้ยงให้กับวงจร ซึ่งในการเลือกโหมดจะต้องเซตจัมพ์เปอร์ J1 บนบอร์ด FPGA ให้ M0, M1 และ M2 มีค่าเป็น “000”

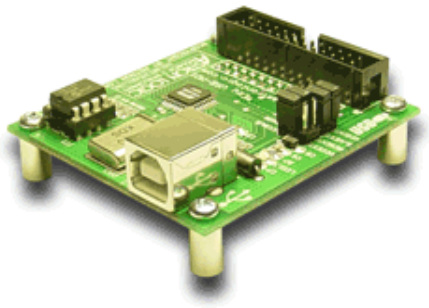
ตารางที่ 2 รายการโหมด Configuration ของบอร์ด FPGA

Configuration Mode	M0	M1	M2
Master Serial	0	0	0
Slave Serial	1	1	1
Master Parallel	1	1	0
Slave Parallel	0	1	1
JTAG	1	0	1

ที่มา: APEX Instrument (n.d.)

2.2 โมดูล USB

โมดูล USB ที่ใช้สำหรับเชื่อมต่อประสานเครื่อง PC และบอร์ด FGPA เข้าด้วยกันเป็นโมดูล USB รุ่น Ezy USB-M01 ของบริษัท แอสทรอน ลอจิก รีเสิร์จแอนดิวิลอปเมนต์ จำกัด ใช้ชิปที่ควบคุมการสื่อสารของระบบบัส USB ของบริษัท FTDI เบอร์ FT245BM ซึ่งมีลักษณะการส่งถ่ายข้อมูลภายในแบบ FIFO โดยโมดูล USB มีความสามารถในการจัดการโพรโตคอล USB ได้อย่างสมบูรณ์แล้วในตัวโมดูล นอกจากนี้ยังมีไดร์ฟเวอร์ซึ่งอนุญาตให้โปรแกรมประยุกต์สามารถเรียกใช้ไฟล์ไลบรารี่ (DLL) ผ่านทางฟังก์ชัน API ที่มีชื่อว่า D2XX Direct Driver ซึ่งทำให้การพัฒนาโปรแกรมประยุกต์เพื่อติดต่อสื่อสารกับโมดูล USB ทำได้สะดวกและรวดเร็วยิ่งขึ้น

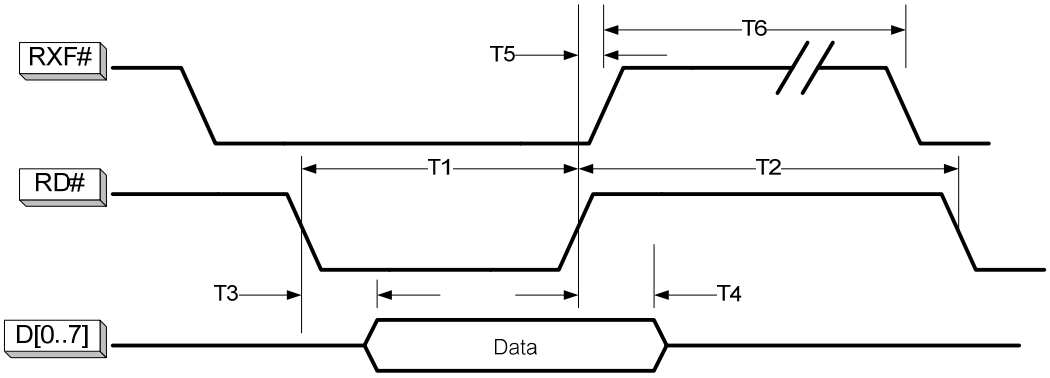


ภาพที่ 9 โมดูล USB รุ่น Ezy USB-M01

ที่มา: Astronlogic Research and Development (n.d.)

จุดเด่นของโมดูล USB คือ มีอัตราการส่งถ่ายข้อมูล 1 เมกะไบต์ต่อวินาที หรือ 8 เมกะบิตต่อวินาที, มีบัฟเฟอร์ FIFO ขนาด 384 ไบต์สำหรับด้านส่ง และขนาด 128 ไบต์สำหรับด้านรับ, สามารถเชื่อมต่อวงจรภายนอกที่ใช้แรงดัน 5 โวลต์และ 3.3 โวลต์, มีวงจรรีเซทขณะเริ่มทำการจ่ายไฟอยู่ภายใน, สนับสนุน USB 1.1 และ USB 2.0 (เฉพาะโหมดฟูลสปีด), มี EEPROM ภายนอกสำหรับเก็บค่าพารามิเตอร์ต่างๆ, สามารถโปรแกรม EEPROM ที่อยู่บนโมดูลผ่านสาย USB ได้โดยตรง, มีจัมพ์เปอร์ตั้งค่าการทำงานให้แก่โมดูล และเชื่อมต่อกับไมโครคอนโทรลเลอร์หรือ FPGA ได้ง่าย

การทำงานของโมดูล USB จะแบ่งเป็น 2 รูปแบบ คือ การอ่านข้อมูลออกจากโมดูล USB และการเขียนข้อมูลลงไปยังโมดูล USB โดยในการอ่านข้อมูลออกจากโมดูล USB นั้นจะเกิดขึ้นก็ต่อเมื่อเครื่อง PC ส่งข้อมูลเข้ามาในโมดูล USB หรือในบัฟเฟอร์ด้านรับ จากนั้นตัวโมดูล USB จะทำให้ขา RXF# เปลี่ยนแปลงเป็นสภาวะลอจิกต่ำ เพื่อเป็นการบอกวงจรภายนอกที่เชื่อมต่อว่าขณะนั้นมีข้อมูลถูกส่งมาแล้วอย่างน้อยขนาดหนึ่งไบต์ และพร้อมให้วงจรที่เชื่อมต่อสามารถอ่านข้อมูลออกไปด้วยการทำให้ขา RD# อยู่ในสภาวะลอจิกต่ำแล้วจึงอ่านข้อมูลที่ปรากฏบนบัสข้อมูลไปเก็บไว้ในบัฟเฟอร์ของวงจรที่เชื่อมต่อ โดยข้อมูลจะถูกอ่านออกไปจนกว่าข้อมูลในบัฟเฟอร์ด้านรับของโมดูล USB จะว่างลง จากนั้นขา RXF# ก็จะเปลี่ยนสภาวะลอจิกกลับไปเป็นสูง



ภาพที่ 10 ไดอะแกรมเวลาสำหรับการอ่านข้อมูลออกจากโมดูล USB

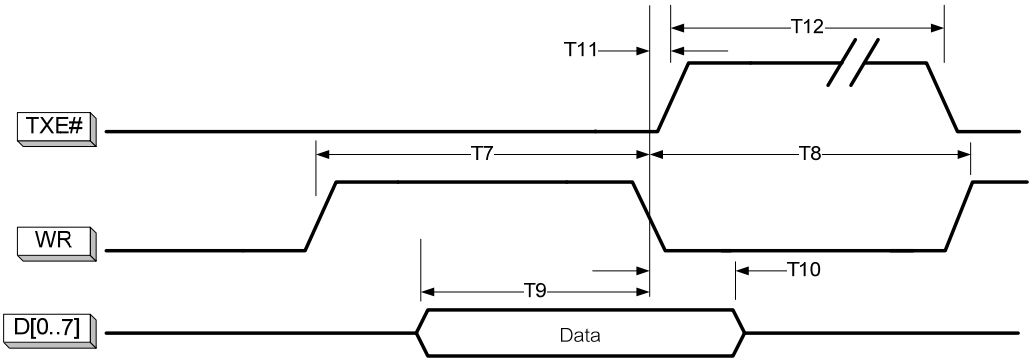
ที่มา: Astronlogic Research and Development (n.d.)

ตารางที่ 3 ค่าเวลาต่างๆ สำหรับการอ่านข้อมูลออกจากโมดูล USB

เวลา	คำอธิบาย	ค่าต่ำสุด (ns)	ค่าสูงสุด (ns)
T1	ความกว้างของพัลส์ในช่วงเวลาที่ RD# ทำงาน	50	-
T2	ค่าเวลาพรีชาร์ทระหว่างที่ RD# หยุดทำงานถึง RD# เริ่มทำงานอีกครั้ง	50	-
T3	ช่วงเวลาที่ RD# ทำงานเพื่อรับข้อมูล	-	30
T4	ช่วงเวลาข้อมูลยังคงอยู่หลังจากที่ RD# หยุดทำงาน	10	-
T5	ช่วงเวลาที่ RD# หยุดทำงานเมื่อเทียบกับ RXF#	5	25
T6	ช่วงเวลาที่ RXF# หยุดทำงานหลังจากกระบวนการอ่าน	80	-

ที่มา: Astronlogic Research and Development (n.d.)

การเขียนข้อมูลไปยังโมดูล USB จะเกิดขึ้นได้ก็ต่อเมื่อขา TXE# อยู่ในสภาวะลอจิกต่ำ โดยวงจรภายนอกที่เชื่อมต่อสามารถส่งข้อมูลไปยังเครื่อง PC ผ่านทางโมดูล USB หรือบัฟเฟอร์ด้านส่งด้วยการทำให้ขา WR เปลี่ยนแปลงเป็นสภาวะลอจิกสูง เพื่อนำข้อมูลที่ปรากฏบนบัสข้อมูลไปเก็บไว้ในบัฟเฟอร์ด้านส่งของโมดูล USB แต่ถ้าบัฟเฟอร์ด้านส่งถูกเขียนข้อมูลลงไปแล้ว หรือยังอยู่ในสภาวะที่ยังไม่ว่าง เนื่องจากการเขียนข้อมูลในครั้งก่อนหน้ายังไม่เสร็จสิ้น โมดูล USB จะให้ขา TXE# มีสภาวะเป็นลอจิกสูง เพื่อหยุดการเขียนข้อมูล จนกว่าข้อมูลในบัฟเฟอร์ด้านส่ง ถูกส่งไปยังเครื่อง PC แล้วจึงเปลี่ยนสภาวะเป็นลอจิกต่ำ



ภาพที่ 11 ไดอะแกรมเวลาสำหรับการเขียนข้อมูลไปยังโมดูล USB

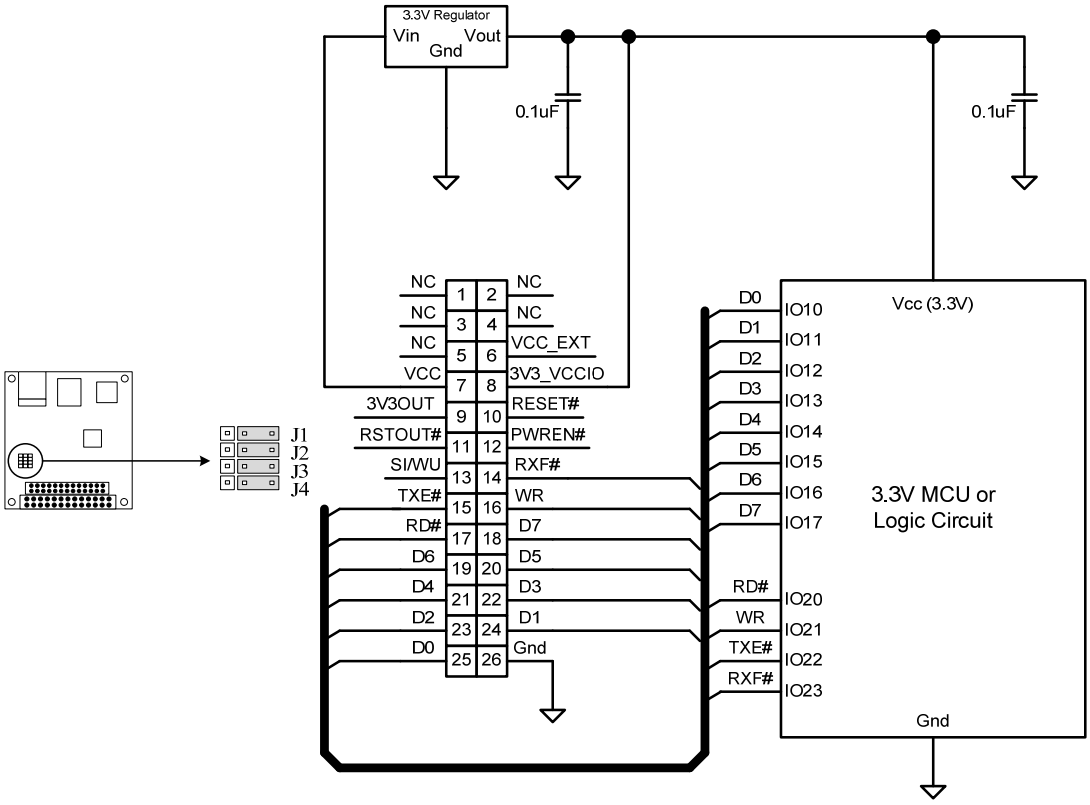
ที่มา: Astronlogic Research and Development (n.d.)

ตารางที่ 4 ค่าเวลาต่างๆ สำหรับการเขียนข้อมูลไปยังโมดูล USB

เวลา	คำอธิบาย	ค่าต่ำสุด (ns)	ค่าสูงสุด (ns)
T7	ความกว้างของพัลส์ในช่วงเวลาที่ WR ทำงาน	50	-
T8	ค่าเวลาพรีชาร์ทระหว่างที่ WR หยุดทำงานถึง WR เริ่มทำงานอีกครั้ง	50	-
T9	ช่วงเวลาที่ WR ทำงานเพื่อเขียนข้อมูล	-	20
T10	ช่วงเวลาข้อมูลยังคงอยู่หลังจากที่ WR หยุดทำงาน	10	-
T11	ช่วงเวลาที่ WR หยุดทำงานเมื่อเทียบกับ TXE#	5	25
T12	ช่วงเวลาที่ TXE# หยุดทำงานหลังจากกระบวนการเขียน	80	-

ที่มา: Astronlogic Research and Development (n.d.)

ตำแหน่งขาสัญญาณของโมดูล USB ที่นำมาใช้งาน คือ ขาที่ 8 สัญญาณ 3V3_VCCIO ใช้ต่อกับแหล่งจ่ายแรงดัน 3.3 โวลต์จากภายนอก และต้องเซทจัมพ์เปอร์ J4 ไปที่ VIOEXT ด้วย, ขาที่ 14 สัญญาณ RXF# เมื่อเป็นลอจิกสูงห้ามทำการอ่านข้อมูลออกจากโมดูล และเมื่อเป็นลอจิกต่ำสามารถอ่านข้อมูลออกจากโมดูลได้, ขาที่ 15 สัญญาณ TXE# เมื่อเป็นลอจิกสูงห้ามทำการเขียนข้อมูลไปยังโมดูล และเมื่อเป็นลอจิกต่ำสามารถเขียนข้อมูลไปยังโมดูลได้, ขาที่ 16 สัญญาณ WR เมื่อเปลี่ยนสภาวะจากลอจิกสูงไปเป็นลอจิกต่ำจะทำให้มีการเขียนข้อมูลที่ปรากฏบนบัสข้อมูลลงในบัฟเฟอร์ด้านส่ง, ขาที่ 17 สัญญาณ RD# เมื่อเป็นลอจิกต่ำจะเปิดข้อมูลให้ไปปรากฏบนบัสข้อมูล และเมื่อเปลี่ยนสภาวะจากลอจิกต่ำไปเป็นลอจิกสูงจะทำให้มีการดึงข้อมูลไบต์ถัดไปมาจากบัฟเฟอร์ด้านรับ, ขาที่ 18 สัญญาณ D7 เป็นบัสข้อมูลแบบ FIFO บิตที่ 7, ขาที่ 19 สัญญาณ D6 เป็นบัสข้อมูลแบบ FIFO บิตที่ 6, ขาที่ 20 สัญญาณ D5 เป็นบัสข้อมูลแบบ FIFO บิตที่ 5, ขาที่ 21 สัญญาณ D4 เป็นบัสข้อมูลแบบ FIFO บิตที่ 4, ขาที่ 22 สัญญาณ D3 เป็นบัสข้อมูลแบบ FIFO บิตที่ 3, ขาที่ 23 สัญญาณ D2 เป็นบัสข้อมูลแบบ FIFO บิตที่ 2, ขาที่ 24 สัญญาณ D1 เป็นบัสข้อมูลแบบ FIFO บิตที่ 1, ขาที่ 25 สัญญาณ D0 เป็นบัสข้อมูลแบบ FIFO บิตที่ 0 และขาที่ 26 สัญญาณ GND เป็นกราวด์ของโมดูล



ภาพที่ 12 การเชื่อมต่อโมดูล USB กับวงจรภายนอกที่ใช้แรงดัน 3.3 โวลต์และการเซตจัมพ์เปอร์

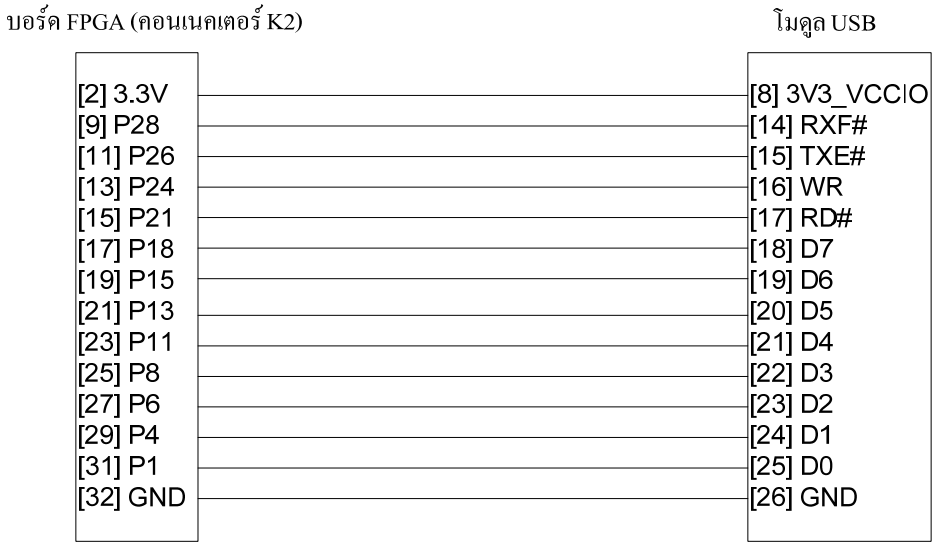
ที่มา: Astronlogic Research and Development (n.d.)

2.3 การเชื่อมต่อประสาน

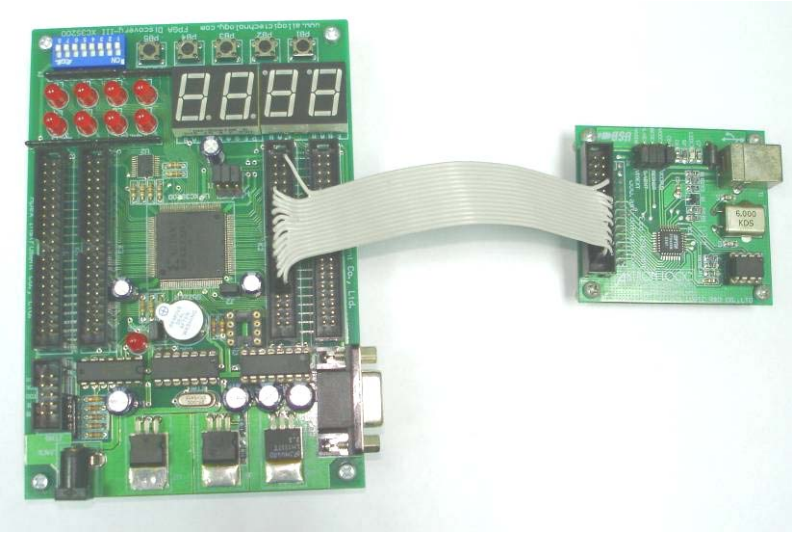
การเชื่อมต่อประสานระหว่างบอร์ด FPGA กับโมดูล USB ใช้คอนเนคเตอร์ K2 ของบอร์ด FPGA เนื่องจากเป็นคอนเนคเตอร์ที่มีตำแหน่งใกล้กับชิป FPGA จึงทำให้ลดปัญหาสัญญาณรบกวนที่อาจเกิดขึ้นจากสายสัญญาณที่ใช้เชื่อมต่อประสาน และยังมีตำแหน่งขาที่ยังไม่ได้ถูกนำไปใช้งานอื่น โดยตารางที่ 5 เป็นการเทียบตำแหน่งขาสัญญาณที่ใช้เชื่อมต่อประสานระหว่างบอร์ด FPGA และโมดูล USB เข้าด้วยกัน ซึ่งมีจำนวนสัญญาณทั้งหมด 14 สัญญาณ นั่นคือ ต้องใช้สายสัญญาณ 14 เส้น และเชื่อมต่อดังภาพที่ 13

ตารางที่ 5 การเทียบตำแหน่งขาสัญญาณบอร์ด FPGA กับโมดูล USB และขาของชิป FPGA

ขาสัญญาณบอร์ด FPGA	ขาสัญญาณโมดูล USB	ขาของชิพ FPGA
สัญญาณนาฬิกา OSC (Clock)	-	P127
ปุ่มสัญญาณรีเซท PB5 (Reset)	-	P51
ขาที่ 2 ของคอนเนคเตอร์ K2 (3.3V)	ขาที่ 8 ของโมดูล (3V3_VCCIO)	3.3V
ขาที่ 9 ของคอนเนคเตอร์ K2	ขาที่ 14 ของโมดูล (RXF#)	P28
ขาที่ 11 ของคอนเนคเตอร์ K2	ขาที่ 15 ของโมดูล (TXE#)	P26
ขาที่ 13 ของคอนเนคเตอร์ K2	ขาที่ 16 ของโมดูล (WR)	P24
ขาที่ 15 ของคอนเนคเตอร์ K2	ขาที่ 17 ของโมดูล (RD#)	P21
ขาที่ 17 ของคอนเนคเตอร์ K2	ขาที่ 18 ของโมดูล (D7)	P18
ขาที่ 19 ของคอนเนคเตอร์ K2	ขาที่ 19 ของโมดูล (D6)	P15
ขาที่ 21 ของคอนเนคเตอร์ K2	ขาที่ 20 ของโมดูล (D5)	P13
ขาที่ 23 ของคอนเนคเตอร์ K2	ขาที่ 21 ของโมดูล (D4)	P11
ขาที่ 25 ของคอนเนคเตอร์ K2	ขาที่ 22 ของโมดูล (D3)	P8
ขาที่ 27 ของคอนเนคเตอร์ K2	ขาที่ 23 ของโมดูล (D2)	P6
ขาที่ 29 ของคอนเนคเตอร์ K2	ขาที่ 24 ของโมดูล (D1)	P4
ขาที่ 31 ของคอนเนคเตอร์ K2	ขาที่ 25 ของโมดูล (D0)	P1
ขาที่ 32 ของคอนเนคเตอร์ K2 (GND)	ขาที่ 26 ของโมดูล (GND)	GND



ภาพที่ 13 สายสัญญาณที่เชื่อมต่อระหว่างบอร์ด FPGA และโมดูล USB



ภาพที่ 14 การเชื่อมต่อประสานบอร์ด FPGA และโมดูล USB

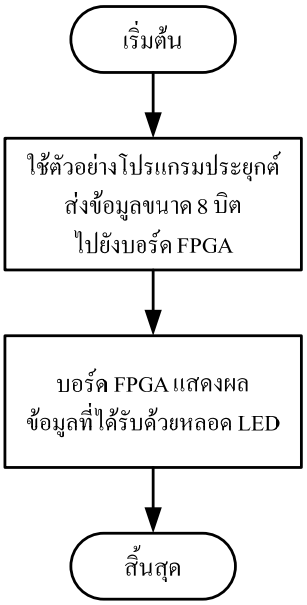
3. การออกแบบซอฟต์แวร์ของระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี

3.1 การติดต่อสื่อสารระหว่างเครื่อง PC และบอร์ด FPGA

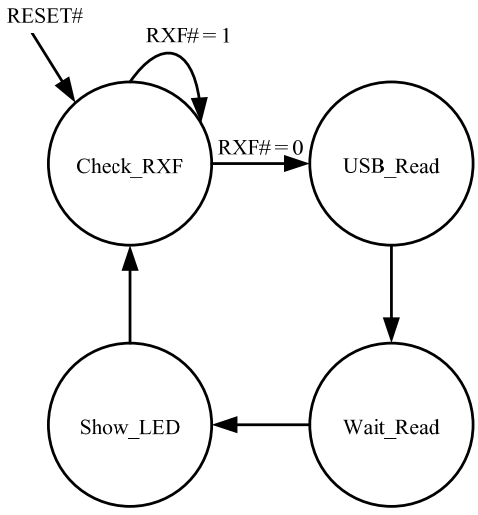
3.1.1 การรับข้อมูลที่ส่งมาจากเครื่อง PC เข้าสู่บอร์ด FPGA

เนื่องจากโมดูล USB มีขนาดบัสข้อมูลเท่ากับ 8 บิต จึงทำให้บอร์ด FPGA สามารถรับข้อมูลจากโมดูล USB ได้เพียงครั้งละ 8 บิตเช่นกัน และจากไดอะแกรมเวลาสำหรับการอ่านข้อมูลออกจากโมดูล USB จะพบว่าในการทำงานมีขาสัญญาณที่เกี่ยวข้องจำนวน 3 สัญญาณ ได้แก่ สัญญาณ RXF#, สัญญาณ RD# และสัญญาณ D[0..7] ดังนั้น ในการออกแบบเพื่อให้บอร์ด FPGA สามารถรับข้อมูลจากเครื่อง PC ได้ จะต้องทำความเข้าใจถึงการทำงานของสัญญาณข้างต้น และทำการออกแบบแผนภาพสเตตการทำงาน จากนั้นจึงเขียนโปรแกรมภาษา VHDL เพื่อทดสอบการทำงานจริงต่อไป

การออกแบบการรับข้อมูลจะเริ่มต้นโดยให้เครื่อง PC ใช้ตัวอย่างโปรแกรมประยุกต์ส่งข้อมูลขนาด 8 บิตผ่านทางโมดูล USB มายังบอร์ด FPGA แล้วแสดงผลข้อมูลที่ได้รับด้วยหลอด LED



ภาพที่ 15 ผังงานการรับข้อมูลขนาด 8 บิตจากเครื่อง PC เข้าสู่บอร์ด FPGA



ภาพที่ 16 แผนภาพสเตตการรับข้อมูลขนาด 8 บิตจากเครื่อง PC เข้าสู่บอร์ด FPGA

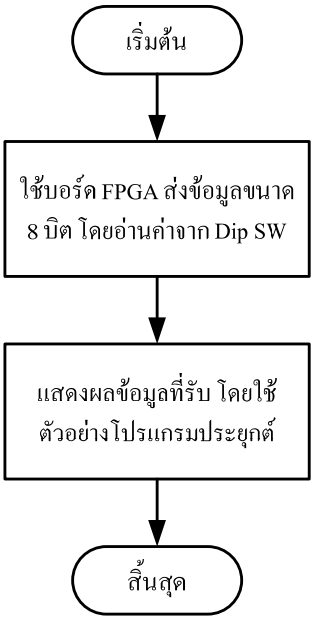
การทำงานของแผนภาพสเตตการรับข้อมูลขนาด 8 บิต จะเริ่มต้นที่สเตต Check_RXF เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ RXF# ของโมดูล USB หากพบว่ามีสภาวะลอจิกสูง ก็จะยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Read เพื่อทำการอ่านข้อมูลจากโมดูล USB และจะเปลี่ยนแปลงสเตตไปเป็น Wait_Read เพื่อ

รอให้กระบวนการอ่านข้อมูลเสร็จสมบูรณ์ แล้วจากนั้นจะเปลี่ยนแปลงสเตตไปเป็น Show_LED เพื่อนำข้อมูลที่ได้รับมาแสดงผลด้วยหลอด LED ก่อนกลับไปเริ่มต้นทำงานที่สเตต Check_RXF อีกครั้ง

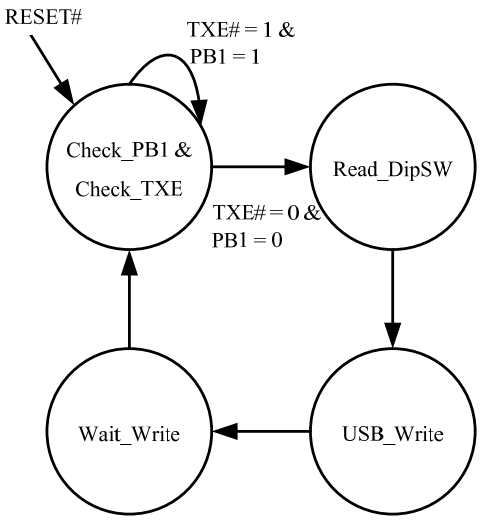
3.1.2 การส่งข้อมูลจากบอร์ด FPGA ไปยังเครื่อง PC

การส่งข้อมูลจากบอร์ด FPGA ไปยังเครื่อง PC ผ่านทางโมดูล USB สามารถส่งข้อมูลได้ครั้งละ 8 บิตเช่นเดียวกัน และจากไดอะแกรมเวลาสำหรับการเขียนข้อมูลไปยังโมดูล USB จะพบว่าในการทำงานมีสัญญาณที่เกี่ยวข้องจำนวน 3 สัญญาณ ได้แก่ สัญญาณ TXE#, สัญญาณ WR และสัญญาณ D[0..7]

การออกแบบการส่งข้อมูลจะเริ่มต้นโดยหากมีการกดปุ่ม PB1 บนบอร์ด FPGA ก็จะอ่านสภาวะลอจิกจาก Dip SW จำนวน 8 บิต แล้วส่งผ่านโมดูล USB ไปยังเครื่อง PC เพื่อแสดงผลข้อมูลที่ได้รับโดยใช้ตัวอย่างโปรแกรมประยุกต์



ภาพที่ 17 ผังงานการส่งข้อมูลขนาด 8 บิตจากบอร์ด FPGA ไปยังเครื่อง PC



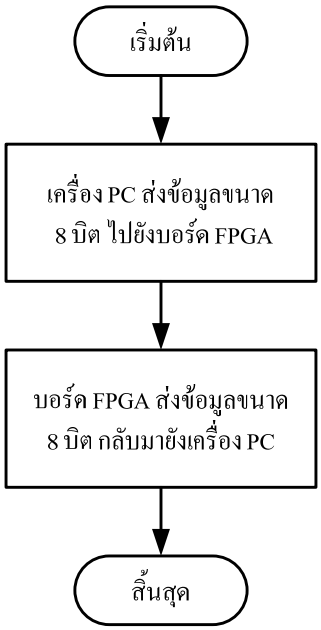
ภาพที่ 18 แผนภาพสเตตการส่งข้อมูลขนาด 8 บิตจากบอร์ด FPGA ไปยังเครื่อง PC

การทำงานของแผนภาพสเตตการส่งข้อมูลขนาด 8 บิต จะเริ่มต้นที่สเตต Check_PB1 & Check_TXE เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ PB1 และ TXE# หากพบว่าทั้ง 2 สัญญาณหรือสัญญาณใดสัญญาณหนึ่งมีสภาวะลอจิกสูง ก็จะยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่าทั้ง 2 สัญญาณมีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น Read_DipSW เพื่ออ่านสภาวะลอจิกจาก Dip SW จำนวน 8 บิต และเปลี่ยนแปลงสเตตไปเป็น USB_Write เพื่อทำการเขียนข้อมูลไปยังโมดูล USB แล้วเปลี่ยนแปลงสเตตไปเป็น Wait_Write เพื่อรอให้กระบวนการเขียนข้อมูลเสร็จสมบูรณ์ ก่อนกลับไปเริ่มต้นทำงานที่สเตต Check_PB1 & Check_TXE อีกครั้ง

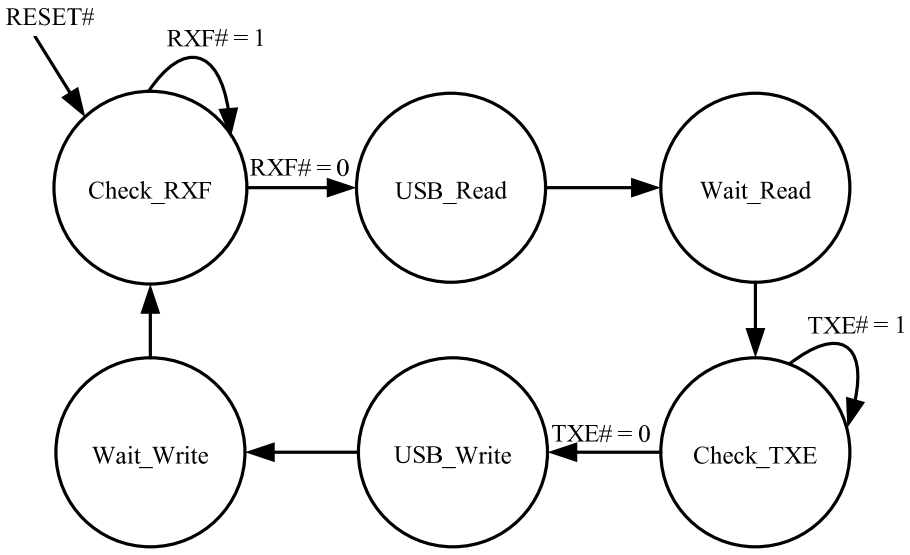
3.1.3 การรับและส่งข้อมูลระหว่างบอร์ด FPGA กับเครื่อง PC

ก. กรณีข้อมูลที่รับส่งมีขนาดเท่ากัน

การออกแบบการรับส่งข้อมูลที่มีขนาดเท่ากันจะเริ่มต้นโดยให้เครื่อง PC ใช้ตัวอย่างโปรแกรมประยุกต์ส่งข้อมูลขนาด 8 บิตผ่านทางโมดูล USB ไปยังบอร์ด FPGA แล้วให้บอร์ด FPGA ส่งข้อมูลขนาด 8 บิตที่ได้รับนั้นกลับมายังเครื่อง PC



ภาพที่ 19 ผังงานการรับส่งข้อมูลขนาด 8 บิตระหว่างบอร์ด FPGA กับเครื่อง PC



ภาพที่ 20 แผนภาพสเตตการรับส่งข้อมูลขนาด 8 บิตระหว่างบอร์ด FPGA กับเครื่อง PC

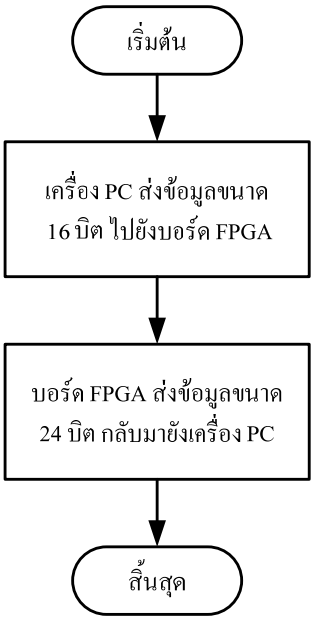
การทำงานของแผนภาพสเตตการรับส่งข้อมูลขนาด 8 บิต จะเริ่มต้นที่สเตต Check_RXF เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ RXF# ของโมดูล USB หากพบว่ามีสภาวะลอจิกสูง ก็จะยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น

USB_Read เพื่อทำการอ่านข้อมูลจากโมดูล USB ไปเก็บไว้ในบัฟเฟอร์ที่เตรียมไว้ในบอร์ด FPGA และเปลี่ยนแปลงสเตตไปเป็น Wait_Read เพื่อรอให้กระบวนการอ่านข้อมูลเสร็จสมบูรณ์ จากนั้นจะเปลี่ยนแปลงสเตตไปเป็น Check_TXE เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ TXE# หากพบว่ามีสภาวะลอจิกสูง ก็จะยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Write เพื่อทำการเขียนข้อมูลที่อยู่ในบัฟเฟอร์บนบอร์ด FPGA ไปยังโมดูล USB แล้วเปลี่ยนแปลงสเตตไปเป็น Wait_Write เพื่อรอให้กระบวนการเขียนข้อมูลเสร็จสมบูรณ์ ก่อนกลับไปเริ่มทำงานที่สเตต Check_RXF อีกครั้ง

ข. กรณีข้อมูลที่รับส่งมีขนาดไม่เท่ากัน

เนื่องจากอัตราที่ใช้ในการเข้ารหัสคอนโวลูชันและจำนวนรายการตัวเลือกของการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล จึงทำให้ขนาดข้อมูลที่รับและส่งมีขนาดไม่เท่ากัน คือ กรณีอัตราการเข้ารหัสคอนโวลูชันเป็น 2 ต่อ 3 จึงทำให้ข้อมูลที่รับเข้ามีขนาดเท่ากับ 2 สัญลักษณ์ และข้อมูลที่ส่งออกมีขนาดเท่ากับ 3 สัญลักษณ์ และกรณีจำนวนรายการตัวเลือกที่ใช้งานในการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลเป็น 2 ตัวเลือก จึงทำให้ข้อมูลที่รับเข้ามีขนาดเป็น 2 เท่าเมื่อเทียบกับข้อมูลที่ส่งออก โดยสิ่งที่เพิ่มเข้ามาในกรณีข้อมูลที่รับส่งมีขนาดไม่เท่ากันและมีขนาดมากกว่า 8 บิต คือ ตัวนับจำนวนนั่นเอง

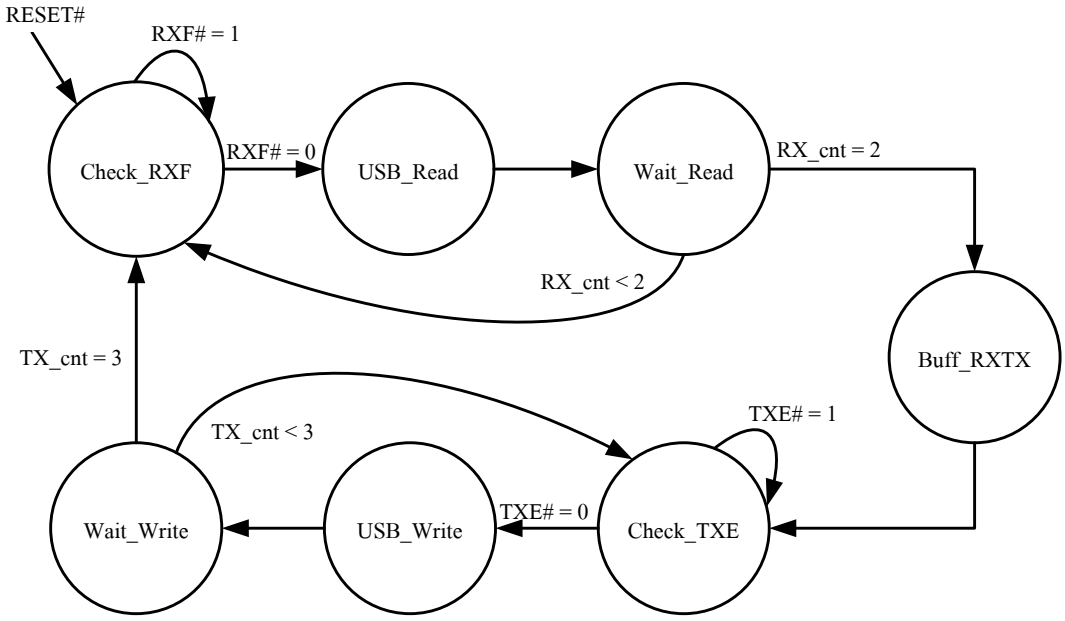
การออกแบบการทำงานกรณีข้อมูลที่รับมีจำนวนน้อยกว่าข้อมูลที่ส่งจะเริ่มต้นโดยให้เครื่อง PC ใช้ตัวอย่างโปรแกรมประยุกต์ส่งข้อมูลขนาด 16 บิตผ่านทางโมดูล USB ไปยังบอร์ด FPGA แล้วให้บอร์ด FPGA ส่งข้อมูลขนาด 24 บิตกลับมายังเครื่อง PC



ภาพที่ 21 ผังงานกรณีข้อมูลที่รับมีจำนวนน้อยกว่าข้อมูลที่ส่ง

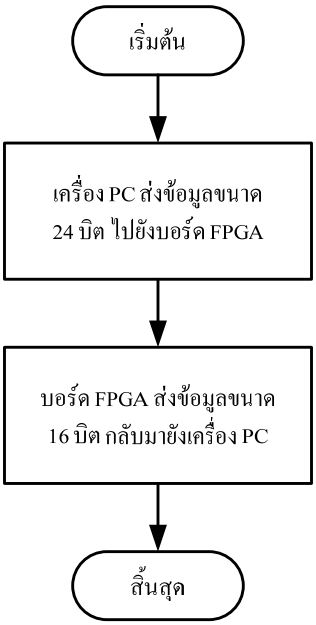
การทำงานของแผนภาพสเตตการรับข้อมูลขนาด 16 บิต และส่งข้อมูลขนาด 24 บิต จะเริ่มต้นที่สเตต Check_RXF เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ RXF# ของโมดูล USB หากพบว่ามีสภาวะลอจิกสูง ก็จะยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Read เพื่อทำการอ่านข้อมูลจากโมดูล USB ไปเก็บไว้ในบัฟเฟอร์ด้านรับและทำการเพิ่มค่าตัวนับ RX_cnt ขึ้นหนึ่งค่า จากนั้นจะเปลี่ยนแปลงสเตตไปเป็น Wait_Read เพื่อรอให้กระบวนการรับข้อมูลเสร็จสมบูรณ์และทำการตรวจสอบค่า RX_cnt ว่ามีค่าเท่ากับ 2 หรือไม่ ถ้าค่า RX_cnt ยังมีค่าน้อยกว่า 2 ก็จะเปลี่ยนแปลงสเตตไปเป็น Check_RXF เพื่อรอรับข้อมูลเข้ามาเพิ่มอีก และเนื่องจากบัสข้อมูลมีขนาด 8 บิต จึงทำให้รับข้อมูลได้ครั้งละ 8 บิต เช่นกัน ถ้าหากค่า RX_cnt มีค่าเท่ากับ 2 ก็แสดงว่ารับข้อมูลได้ครบ 16 บิตแล้ว จึงเปลี่ยนแปลงสเตตไปเป็น Buff_RXTX เพื่อนำข้อมูลที่รับทั้งหมดเก็บลงในบัฟเฟอร์ข้อมูล จากนั้นเปลี่ยนแปลงสเตตไปเป็น Check_TXE เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ TXE# หากพบว่ามีสภาวะลอจิกสูง ก็ยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Write เพื่อทำการส่งข้อมูลที่อยู่ในบัฟเฟอร์ด้านส่งไปยังโมดูล USB และทำการเพิ่มค่าตัวนับ TX_cnt ขึ้นหนึ่งค่า แล้วเปลี่ยนแปลงสเตตไปเป็น Wait_Write เพื่อรอให้กระบวนการส่งข้อมูลเสร็จสมบูรณ์และทำการตรวจสอบค่า TX_cnt ว่ามีค่าเท่ากับ 3 หรือไม่ ถ้าค่า TX_cnt ยังมีค่าน้อยกว่า 3 ก็จะเปลี่ยนแปลงสเตตไปเป็น Check_TXE เพื่อส่งข้อมูลชุดถัดไป เนื่องจากบัสข้อมูลมีขนาด 8 บิต จึง

ทำให้ส่งข้อมูลได้ครั้งละ 8 บิต แต่ถ้าค่า TX_cnt มีค่าเท่ากับ 3 ก็แสดงว่าส่งข้อมูลได้ครบ 24 บิต ก่อนกลับไปเริ่มทำงานที่สเตต Check_RXF อีกครั้ง

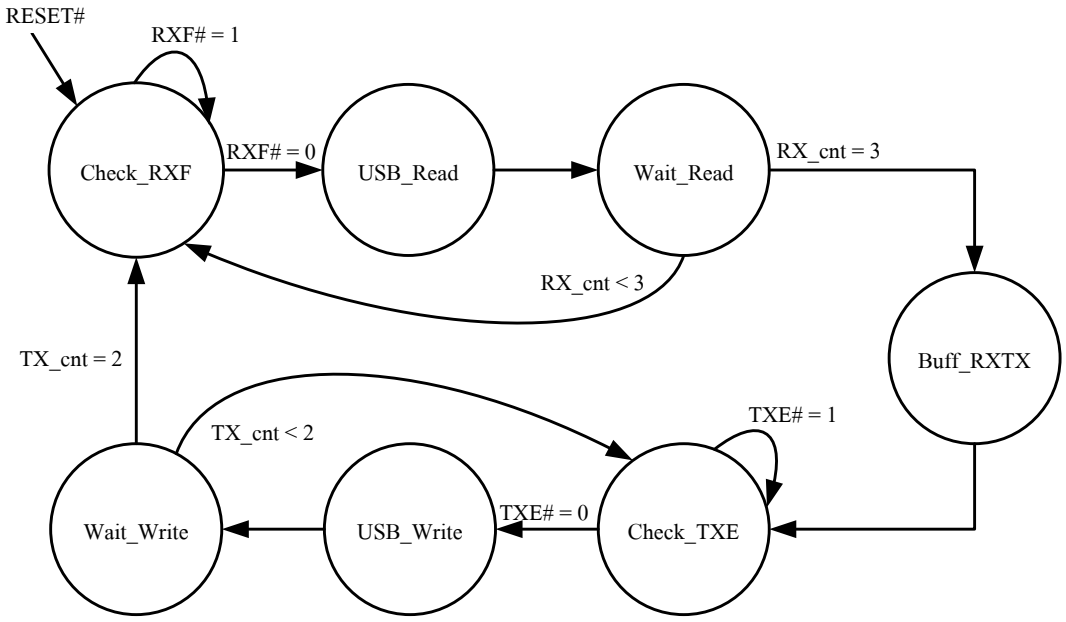


ภาพที่ 22 แผนภาพสเตตกรณีข้อมูลที่รับมีจำนวนน้อยกว่าข้อมูลที่ส่ง

การออกแบบการทำงานกรณีข้อมูลที่รับมีจำนวนมากกว่าข้อมูลที่ส่งจะเริ่มต้นโดยให้เครื่อง PC ใช้ตัวอย่างโปรแกรมประยุกต์ส่งข้อมูลขนาด 24 บิต ผ่านทางโมดูล USB ไปยังบอร์ด FPGA แล้วให้บอร์ด FPGA ส่งข้อมูลขนาด 16 บิตกลับมายังเครื่อง PC



ภาพที่ 23 ผังงานกรณีข้อมูลที่รับมีจำนวนมากกว่าข้อมูลที่ส่ง



ภาพที่ 24 แผนภาพสเตตกรณีข้อมูลที่รับมีจำนวนมากกว่าข้อมูลที่ส่ง

การทำงานของแผนภาพสเตตการรับข้อมูลขนาด 24 บิต และส่งข้อมูลขนาด 16 บิต จะเริ่มต้นที่สเตต Check_RXF เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ RXF# ของโมดูล

USB หากพบว่ามีสภาวะลอจิกสูง ก็ยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Read เพื่อทำการรับข้อมูลจากโมดูล USB ไปเก็บไว้ในบัฟเฟอร์ด้านรับและทำการเพิ่มค่าตัวนับ RX_cnt ขึ้นหนึ่งค่า จากนั้นจะเปลี่ยนแปลงสเตตไปเป็น Wait_Read เพื่อรอให้กระบวนการรับข้อมูลเสร็จสมบูรณ์และทำการตรวจสอบค่า RX_cnt ว่ามีค่าเท่ากับ 3 หรือไม่ ถ้าค่า RX_cnt ยังมีค่าน้อยกว่า 3 ก็จะเปลี่ยนแปลงสเตตไปเป็น Check_RXF เพื่อรอรับข้อมูลเข้ามาเพิ่มอีก แต่ถ้าค่า RX_cnt มีค่าเท่ากับ 3 ก็แสดงว่ารับข้อมูลได้ครบ 24 บิตโดยจะเปลี่ยนแปลงสเตตไปเป็น Buff_RXTX เพื่อนำข้อมูลที่รับทั้งหมดเก็บลงในบัฟเฟอร์ข้อมูล จากนั้นเปลี่ยนแปลงสเตตไปเป็น Check_TXE เพื่อตรวจสอบสภาวะลอจิกที่ขาสัญญาณ TXE# หากพบว่ามีสภาวะลอจิกสูง ก็ยังไม่เปลี่ยนแปลงสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ จะเปลี่ยนแปลงสเตตไปเป็น USB_Write เพื่อทำการส่งข้อมูลที่อยู่ในบัฟเฟอร์ด้านส่งไปยังโมดูล USB และทำการเพิ่มค่าตัวนับ TX_cnt ขึ้นหนึ่งค่า แล้วเปลี่ยนแปลงสเตตไปเป็น Wait_Write เพื่อรอให้กระบวนการส่งข้อมูลเสร็จสมบูรณ์และทำการตรวจสอบค่า TX_cnt ว่ามีค่าเท่ากับ 2 หรือไม่ ถ้าค่า TX_cnt ยังมีค่าน้อยกว่า 2 ก็จะเปลี่ยนแปลงสเตตไปเป็น Check_TXE เพื่อส่งข้อมูลชุดถัดไป แต่ถ้าค่า TX_cnt มีค่าเท่ากับ 2 ก็แสดงว่าส่งข้อมูลได้ครบ 16 บิต ก่อนกลับไปเริ่มทำงานที่สเตต Check_RXF อีกครั้ง

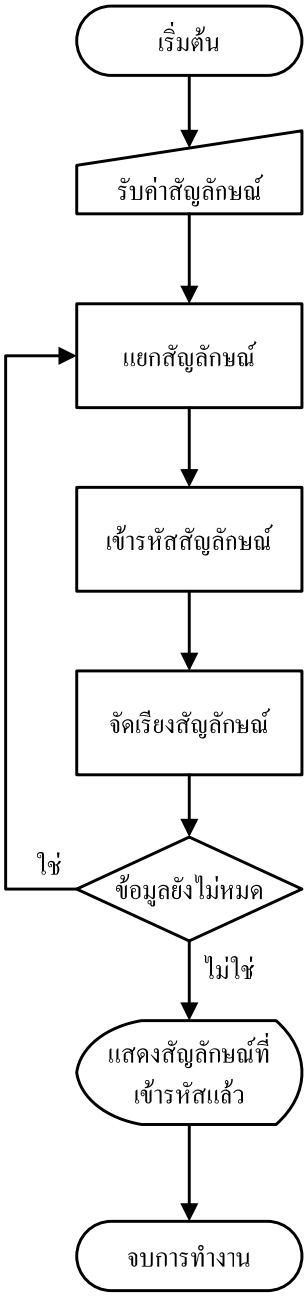
3.2 การเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์

3.2.1 การออกแบบโปรแกรมจำลองการทำงาน

ฟังก์ชันการเข้ารหัสคอนโวลูชัน (3, 2, 2) จะใช้ตัวดำเนินการเอ็กซ์คลูซีฟออร์ (XOR) และหน่วยความจำในการเข้ารหัสข้อมูล แต่เนื่องจากข้อมูลที่จะนำมาทำการเข้ารหัสเป็นแบบสัญลักษณ์นอนไบนารีที่มีขนาด 32 บิตต่อสัญลักษณ์ ดังนั้น ตัวดำเนินการเอ็กซ์คลูซีฟออร์และหน่วยความจำที่ใช้ต้องรองรับข้อมูลขนาด 32 บิตเช่นกัน เพื่อจำลองการทำงานของฟังก์ชันการเข้ารหัสคอนโวลูชัน (3, 2, 2) จึงเขียนโปรแกรมภาษา C++ และนำผลลัพธ์ที่ได้มาใช้สำหรับตรวจสอบการทำงานโปรแกรมที่เขียนด้วยภาษา VHDL ต่อไป

การทำงานของโปรแกรมภาษา C++ จะเริ่มจากการแยกข้อมูลสัญลักษณ์ที่รับเข้ามาเป็นลำดับให้เป็น 2 ชุด โดยแต่ละชุดสัญลักษณ์มีขนาดเท่ากับ 32 บิต จากนั้นนำข้อมูลทั้ง 2 ชุดสัญลักษณ์ มาทำการเข้ารหัสโดยใช้ตัวดำเนินการเอ็กซ์คลูซีฟออร์กระทำกับสัญลักษณ์ชุดใหม่และสัญลักษณ์ชุดเก่าในหน่วยความจำในลักษณะบิตต่อบิต แล้วจึงทำการเลื่อนสัญลักษณ์ชุดเก่า

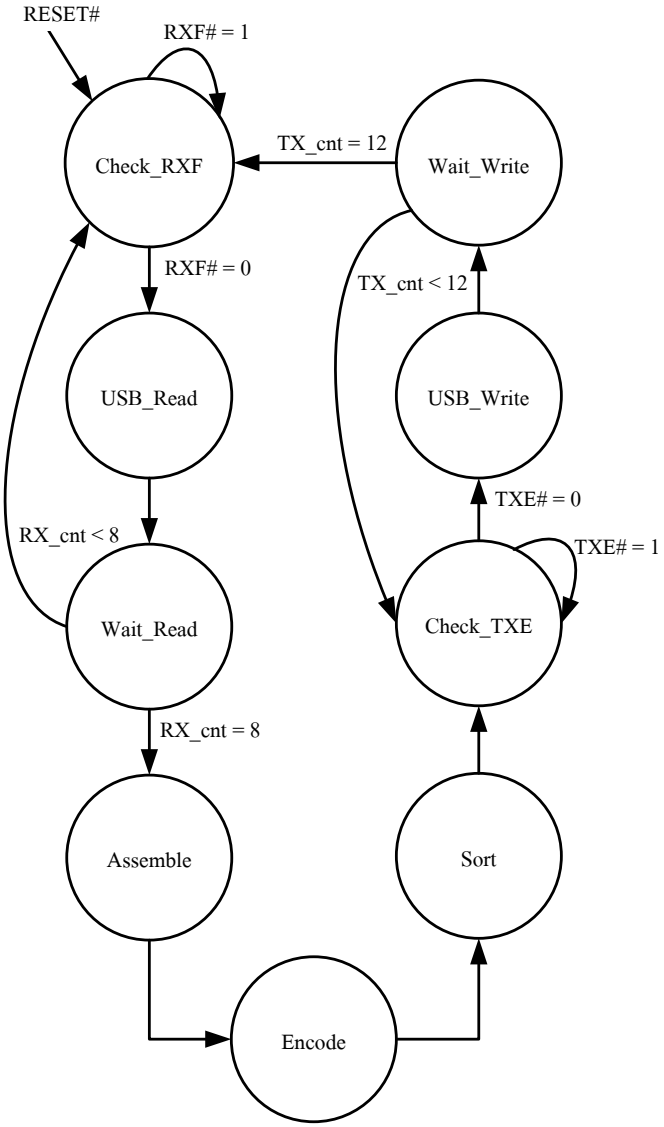
พร้อมกับเก็บสัญลักษณ์ชุดใหม่ไว้ในหน่วยความจำในลักษณะบิตต่อบิตเช่นกัน ซึ่งผลลัพธ์จากการเข้ารหัสจะทำให้ได้สัญลักษณ์ที่ผ่านการเข้ารหัสแล้วจำนวน 3 ชุดสัญลักษณ์ หรือมีขนาด 96 บิต โดยมีการจัดเรียงลำดับก่อนหลังไว้อย่างถูกต้อง และหลังจากทำการเข้ารหัสข้อมูล 2 ชุดสัญลักษณ์แรกเสร็จ ก็จะนำข้อมูล 2 สัญลักษณ์ชุดถัดไปเข้ามาประมวลผลต่อไป จนกว่าข้อมูลจะหมด



ภาพที่ 25 ผังงานการเข้ารหัสคอนโวลูชันโดยเขียนโปรแกรมภาษา C++ จำลองการทำงาน

3.2.2 การออกแบบแผนภาพสเตต

จากผังงานการเข้ารหัสคอนโวลูชัน พบว่ากระบวนการหลักประกอบด้วย 3 กระบวนการ คือ การแยกข้อมูล, การเข้ารหัสข้อมูล และการจัดเรียงข้อมูล ซึ่งสามารถนำมาออกแบบแผนภาพสเตตการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์ โดยสามารถนำวิธีการออกแบบในกรณีข้อมูลที่รับมีจำนวนน้อยกว่าข้อมูลที่ส่งมาประยุกต์ใช้งานได้



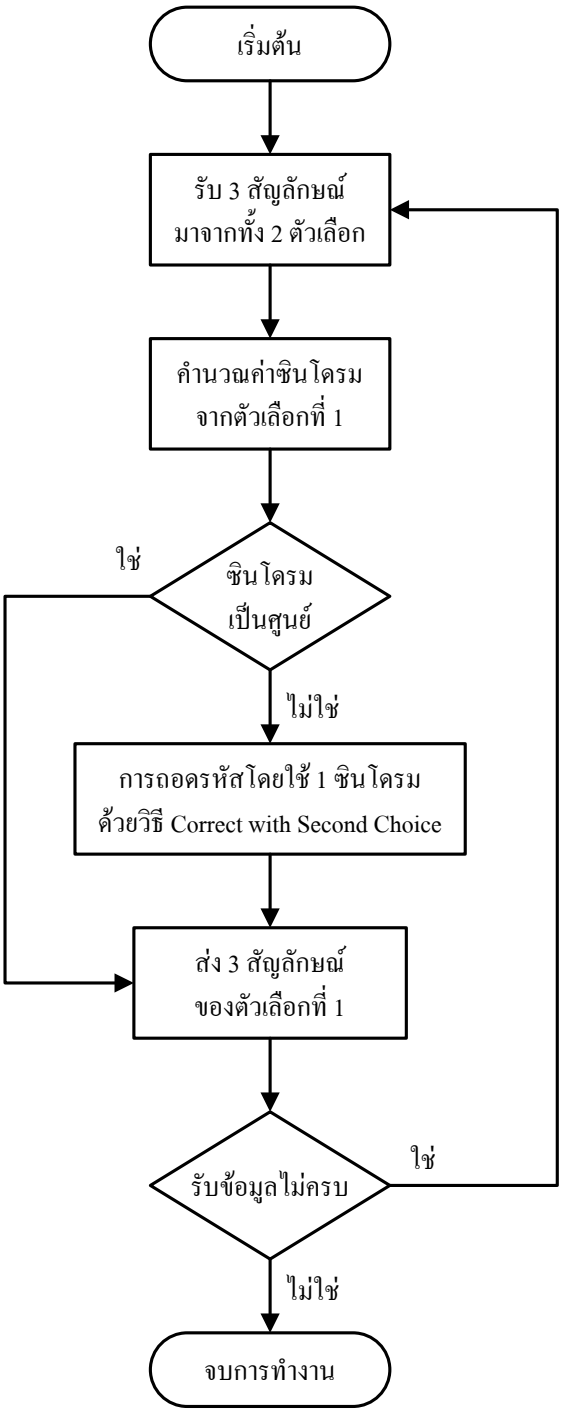
ภาพที่ 26 แผนภาพสเตตการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารี

การทำงานของแผนภาพสเตตการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์จะเริ่มต้นที่สเตต Check_RXF เพื่อตรวจสอบว่าโมดูล USB ได้รับข้อมูลจากเครื่อง PC แล้วหรือไม่ หากพบว่ามีสภาวะลอจิกสูง ก็หมายถึงยังไม่มีข้อมูลถูกส่งมา จึงไม่เปลี่ยนสเตต แต่หากพบว่ามีสภาวะลอจิกต่ำ ก็หมายถึงมีข้อมูลถูกส่งมาแล้ว จึงเปลี่ยนสเตตไปเป็น USB_Read เพื่อทำการรับข้อมูลจากโมดูล USB ไปเก็บไว้ในบัฟเฟอร์ด้านรับและทำการเพิ่มค่าตัวนับ RX_cnt ขึ้นหนึ่งค่า จากนั้นจึงเปลี่ยนสเตตไปเป็น Wait_Read เพื่อรอให้กระบวนการรับข้อมูลเสร็จสมบูรณ์และทำการตรวจสอบค่า RX_cnt ว่ามีค่าเท่ากับ 8 หรือไม่ ถ้าค่า RX_cnt ยังมีค่าน้อยกว่า 8 ก็หมายถึงยังรับข้อมูลเข้ามาไม่ครบตามที่ต้องการ จึงเปลี่ยนสเตตไปเป็น Check_RXF เพื่อรอรับข้อมูลเข้ามาเพิ่มอีก แต่ถ้าค่า RX_cnt มีค่าเท่ากับ 8 ก็แสดงว่ารับข้อมูลได้ครบแล้ว จึงเปลี่ยนสเตตไปเป็น Assemble เพื่อจัดเรียงข้อมูลที่ได้รับมาทั้งหมดเป็น 2 สัญลักษณ์ แล้วจึงเปลี่ยนสเตตไปเป็น Encode เพื่อนำชุดสัญลักษณ์มาเข้ารหัสโดยการทำเอ็กซ์คลูซีฟออร์สัญลักษณ์ชุดใหม่และสัญลักษณ์ชุดเก่าในหน่วยความจำ แล้วทำการเลื่อนสัญลักษณ์ชุดเก่าพร้อมกับเก็บสัญลักษณ์ชุดใหม่ไว้ในหน่วยความจำ เมื่อดำเนินการเสร็จจะเปลี่ยนสเตตไปเป็น Sort เพื่อการจัดเรียงลำดับสัญลักษณ์ที่ผ่านการเข้ารหัสแล้วจำนวน 3 สัญลักษณ์ให้ถูกตำแหน่ง ก่อนจะเปลี่ยนสเตตไปเป็น Check_TXE เพื่อทำการส่งข้อมูลผ่านโมดูล USB ไปยังเครื่อง PC โดยมีการตรวจสอบค่า TX_cnt ว่ามีค่าเท่ากับ 12 หรือไม่ ถ้าค่า TX_cnt ยังมีค่าน้อยกว่า 12 ก็หมายถึงยังส่งข้อมูลไม่หมด แต่ถ้าค่า TX_cnt มีค่าเท่ากับ 12 ก็แสดงว่าได้ส่งข้อมูลออกไปครบ 3 สัญลักษณ์แล้ว จากนั้นจะกลับไปเริ่มทำงานที่สเตต Check_RXF เพื่อรอรับข้อมูลอีก 2 สัญลักษณ์ถัดไปมาทำการเข้ารหัส

3.3 การถอดรหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์ด้วยวิธีเวกเตอร์ซิมโบล

3.3.1 การออกแบบฟังก์ชัน

การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล มีการใช้รายการตัวเลือกจำนวน 2 ตัวเลือก เพื่อช่วยให้การถอดรหัสสามารถทำได้รวดเร็วยิ่งขึ้น และเมื่อพิจารณาจำนวนข้อมูลที่รับเข้ามาพบว่าในแต่ละตัวเลือกจะต้องรับข้อมูลเข้ามาครั้งละจำนวน 3 สัญลักษณ์ ซึ่งหากรวมกันทั้ง 2 ตัวเลือกแล้ว จะต้องรับข้อมูลเข้ามาทั้งหมดครั้งละจำนวน 6 สัญลักษณ์ แต่สำหรับการส่งข้อมูลที่ได้ผ่านการถอดรหัสแล้ว จะส่งเฉพาะข้อมูลของตัวเลือกที่ 1 ที่ถูกต้องเท่านั้น จึงสามารถนำวิธีการออกแบบในกรณีข้อมูลที่รับมีจำนวนมากกว่าข้อมูลที่ส่งมาประยุกต์ใช้งานได้



ภาพที่ 27 ผังงานการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลด้วยซินโดรมเดียว

จากผังงานการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลที่ออกแบบเป็นเพียงบางส่วนของการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล ในขั้นต้นจะออกแบบให้สามารถทำการถอดรหัสโดยใช้จำนวนซินโดรมเท่ากับหนึ่งเท่านั้น ด้วยวิธีการ Correct with Second Choice ซึ่งเป็นวิธีการที่ช่วยให้

การถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลทำได้รวดเร็ว โดยอาศัยสัญลักษณ์ในตัวเลือกที่ 2 ที่มีความถูกต้องในตำแหน่งเดียวกันกับสัญลักษณ์ในตัวเลือกที่ 1 ที่มีความผิดพลาด เพื่อช่วยแก้ไขสัญลักษณ์ให้ถูกต้อง แต่สำหรับในกรณีที่ซินโดรมมีค่ามากกว่า 1 หรือสัญลักษณ์ในตัวเลือกที่ 2 ไม่สามารถช่วยได้ ก็จะยังไม่สามารถแก้ไขสัญลักษณ์ให้ถูกต้องได้

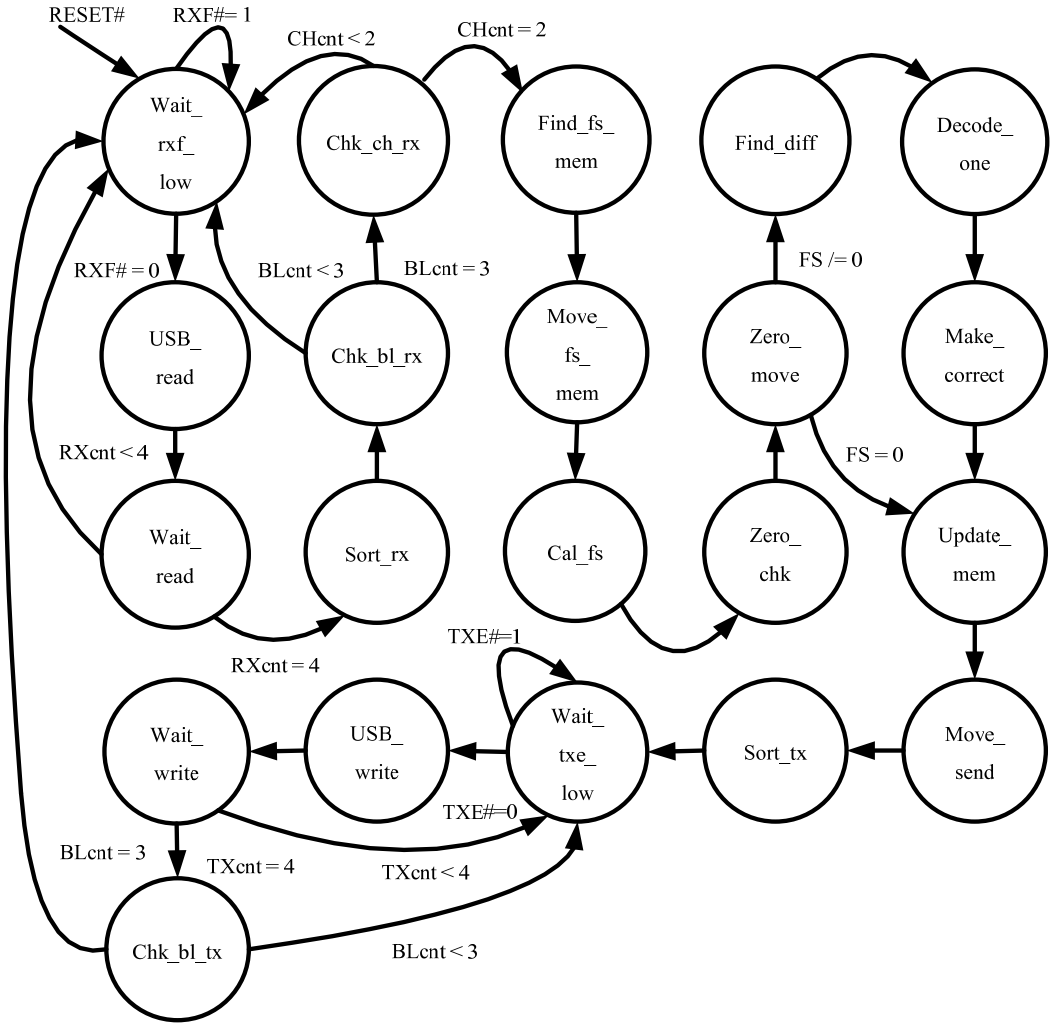
เมื่อเริ่มต้นการทำงานจะรับสัญลักษณ์ของตัวเลือกที่ 1 เข้ามาจำนวน 3 สัญลักษณ์ และรับสัญลักษณ์ของตัวเลือกที่ 2 เข้ามาจำนวน 3 สัญลักษณ์ แล้วทำการคำนวณหาค่าซินโดรมจากสัญลักษณ์ของตัวเลือกที่ 1 เพื่อตรวจสอบว่ามีความผิดพลาดเกิดขึ้นกับสัญลักษณ์ใดสัญลักษณ์หนึ่งในตัวเลือกที่ 1 หรือไม่ หากพบว่าซินโดรมมีค่าเป็นศูนย์ ก็แสดงว่าไม่มีความผิดพลาดเกิดขึ้น ก็จะทำการส่งสัญลักษณ์ของตัวเลือกที่ 1 ทั้ง 3 สัญลักษณ์ออกไปได้ แต่หากพบว่าซินโดรมมีค่าไม่เป็นศูนย์ ก็แสดงว่ามีความผิดพลาดเกิดขึ้นกับสัญลักษณ์ใดสัญลักษณ์หนึ่งในตัวเลือกที่ 1 ก็จะทำการแก้ไขด้วยวิธี Correct with Second Choice โดยทำการคำนวณหาค่าผลต่างระหว่างสัญลักษณ์ของตัวเลือกที่ 1 และตัวเลือกที่ 2 เพื่อหาตำแหน่งสัญลักษณ์ที่มีความผิดพลาดของตัวเลือกที่ 1 แล้วแก้ไขสัญลักษณ์ที่ผิดพลาดให้ถูกต้อง ก่อนส่งสัญลักษณ์ของตัวเลือกที่ 1 ทั้ง 3 สัญลักษณ์ออกไป และจะกลับไปเริ่มต้นการทำงานใหม่อีกครั้ง

3.3.2 การออกแบบแผนภาพสเตต

การทำงานของแผนภาพสเตตการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบล เมื่อเริ่มต้นทำงานหรือมีการกดปุ่มรีเซต สเตตเริ่มต้นจะเป็น Wait_rxf_low ในสเตตนี้มีการตรวจสอบสัญญาณ RXF# จากโมดูล USB ถ้าสัญญาณ RXF# ยังมีสภาวะเป็นลอจิกสูง สเตตก็จะยังไม่เปลี่ยนแปลง จนกว่าสัญญาณ RXF# จะมีสภาวะเป็นลอจิกต่ำ สเตตก็จะเปลี่ยนไปเป็น USB_read โดยในสเตตนี้บอร์ด FPGA จะมีการส่งสัญญาณ RD ออกไปยังโมดูล USB เพื่ออ่านข้อมูลจากโมดูล USB เข้ามาในบอร์ด FPGA ผ่านบัสข้อมูล D[0..7] โดยข้อมูลที่เข้ามานี้จะถูกนำไปเก็บไว้ในบัฟเฟอร์ฝั่งรับบนบอร์ด FPGA แล้วจะเปลี่ยนสเตตไปเป็น Wait_read โดยในสเตตนี้จะรอให้การทำงานต่างๆ ที่ทำในสเตต USB_read มีการเปลี่ยนแปลง และมีการตรวจสอบค่าตัวแปร RX_cnt ซึ่งใช้สำหรับนับจำนวนครั้งในการรับข้อมูล เนื่องจากข้อมูลที่ใช้การประมวลผลมีขนาด 32 บิต แต่การทำงานของโมดูล USB สามารถรับหรือส่งข้อมูลได้เพียงครั้งละ 8 บิตเท่านั้น ดังนั้น จะต้องมีการรับข้อมูลเข้ามาจำนวน 4 ครั้งจึงจะทำให้ได้ขนาดของข้อมูลตามที่ต้องการ การรับข้อมูลเข้ามาเพิ่มจะมีการเปลี่ยนสเตตกลับไปเป็น Wait_rxf_low แล้วจึงเริ่มทำงานเหมือนเดิมจนกว่า RX_cnt จะมีค่าเท่ากับ

4 หลังจากนั้นก็จะเปลี่ยนสเตตไปเป็น Sort_rx โดยในสเตตนี้จะทำการย้ายข้อมูลที่อยู่ในบัฟเฟอร์ฝั่งรับเข้าไปเก็บไว้ในบัฟเฟอร์ที่เตรียมไว้สำหรับเก็บข้อมูลใน FPGA ที่มีขนาดเท่ากับ 32 บิต เพื่อใช้ในการประมวลผลต่อไป ในสเตตถัดไปจะเป็นสเตต Chk_bl_rx จะตรวจสอบค่าตัวแปร BL_cnt เป็นตัวแปรที่ใช้สำหรับนับจำนวนสัญลักษณ์ที่รับเข้ามาแล้ว ซึ่งจำนวนสัญลักษณ์ที่ต้องการสำหรับการทำงานของตัวถอดรหัสนี้เท่ากับ 3 สัญลักษณ์ เมื่อรับจำนวนของสัญลักษณ์เข้ามาครบแล้ว สเตตต่อไปที่ทำงาน คือ Chk_ch_rx ในสเตตนี้จะทำการตรวจสอบค่าตัวแปร CH_cnt ตัวแปรนี้ใช้สำหรับการนับจำนวนรายการตัวเลือกที่รับเข้ามาแล้ว โดยกำหนดค่าตัวเลือกเท่ากับ 2 และถ้าหาค่าตัวเลือกยังไม่ครบก็จะกลับไปรับข้อมูลใหม่ในสเตต Wait_rxf_low จนกว่าจะได้ข้อมูลครบทั้ง 2 ตัวเลือก เมื่อได้รับข้อมูลครบแล้วสเตตจะเปลี่ยนไปเป็น Find_fs_mem ในสเตตนี้จะทำการคำนวณค่าข้อมูลก่อนหน้าในหน่วยความจำ เพื่อใช้สำหรับหาค่าซินโดรมต่อไปในสเตต Cal_fs แต่ก่อนที่จะไปทำสเตต Cal_fs จะต้องไปทำการปรับปรุงค่าตัวแปรที่ได้มาจากการคำนวณในสเตต Find_fs_mem ในสเตต Move_fs_mem แล้วจึงไปทำการคำนวณค่าซินโดรมของข้อมูลในตัวเลือกที่หนึ่ง (First choice) ในสเตต Cal_fs เมื่อคำนวณเสร็จสเตตจะเปลี่ยนไปเป็น Zero_chk เพื่อทำการตรวจสอบค่าซินโดรมที่คำนวณได้ว่ามีค่าเท่ากับศูนย์หรือไม่ แล้วสเตตจะเปลี่ยนไปเป็น Zero_move ในสเตตนี้จะนำค่าของซินโดรมมาตัดสินใจว่าสเตตต่อไปจะให้ไปทำงานอะไร ถ้าค่าซินโดรมที่คำนวณได้มีค่าเป็นศูนย์ สเตตต่อไปก็จะนำข้อมูลในตัวเลือกที่หนึ่งไปเก็บไว้ในหน่วยความจำสำหรับใช้ในการคำนวณค่าซินโดรมของข้อมูลชุดถัดไป แล้วส่งข้อมูลออกไป ถ้าค่าซินโดรมที่คำนวณได้มีค่าไม่เป็นศูนย์ สเตตต่อไปจะนำค่าในตัวเลือกที่หนึ่งไปทำการแก้ไขให้ถูกต้องโดยใช้ข้อมูลในชุดที่สอง (Second choice) มาช่วยในการแก้ไข สเตต Find_diff เป็นสเตตที่ใช้คำนวณหาค่าผลต่างระหว่างข้อมูลในตัวเลือกที่หนึ่งและตัวเลือกที่สอง และสเตตถัดไป Decode_one ในสเตตนี้จะนำค่าผลต่างที่ได้มาจากสเตต Find_diff มาใช้ในการหาค่าตำแหน่งที่มีค่าความผิดพลาด แล้วจะใช้ค่านี้ไปแก้ไขข้อมูลในตัวเลือกที่หนึ่งให้ถูกต้องในสเตต Make_correct เมื่อแก้ไขข้อมูลให้ถูกต้องแล้ว จะนำข้อมูลในตัวเลือกที่หนึ่งไปเก็บไว้ในหน่วยความจำ สำหรับใช้ในการคำนวณค่าซินโดรมของข้อมูลชุดถัดไป ซึ่งจะทำในสเตต Update_mem เมื่อสเตตนี้ทำงานเสร็จ ต่อไปจะมีการนำค่าข้อมูลในตัวเลือกที่หนึ่งเข้าไปเก็บไว้มนบัฟเฟอร์ฝั่งส่งในสเตต Move_send เพื่อจะนำข้อมูลในบัฟเฟอร์ไปแยกเป็นส่วนย่อยๆ ขนาดส่วนละ 8 บิต สเตต Sort_tx นี้จะทำให้ขนาดของข้อมูลเหมาะสมกับการส่งข้อมูลกลับไปให้โมดูล USB โดยการส่งข้อมูลจะเริ่มการทำงานในสเตต Wait_txe_low จะมีวิธีการทำงานเหมือนกันกับการส่งข้อมูลที่กล่าวมาแล้วข้างต้น แต่มีการตรวจสอบค่าของตัวแปร TX_cnt และมีสเตต Chk_bl_tx เพิ่มเข้ามา เพื่อตรวจสอบจำนวนสัญลักษณ์ที่ส่งออกไป โดยการตรวจสอบตัวแปร BL_cnt ว่ามีค่าเท่ากับที่ต้องการหรือยัง ถ้า

ค่ายังไม่เท่ากับที่ต้องการก็จะกลับไปทำที่สเตต Wait_txe_low อีกครั้งจนกว่าจะมีการส่งข้อมูลออกไปจนครบ 3 สัญลักษณ์แล้ว เมื่อส่งข้อมูลออกไปจนครบแล้วสเตตการทำงานก็จะกลับไปเริ่มต้นรับข้อมูลชุดใหม่มาถอดรหัสที่สเตต Wait_rxf_low



ภาพที่ 28 แผนภาพสเตตการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลด้วยซินโครมเดียว

ผลและวิจารณ์

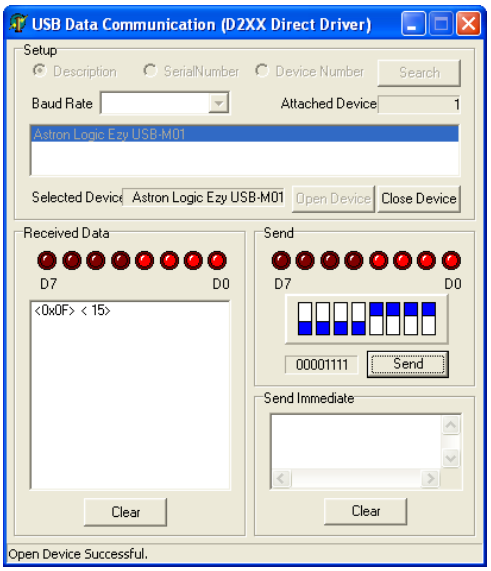
ผล

1. การติดต่อสื่อสารระหว่างเครื่อง PC และบอร์ด FPGA

1.1 การรับและส่งข้อมูลระหว่างบอร์ด FPGA กับเครื่อง PC

1.1.1 กรณีข้อมูลที่รับส่งมีขนาดเท่ากัน

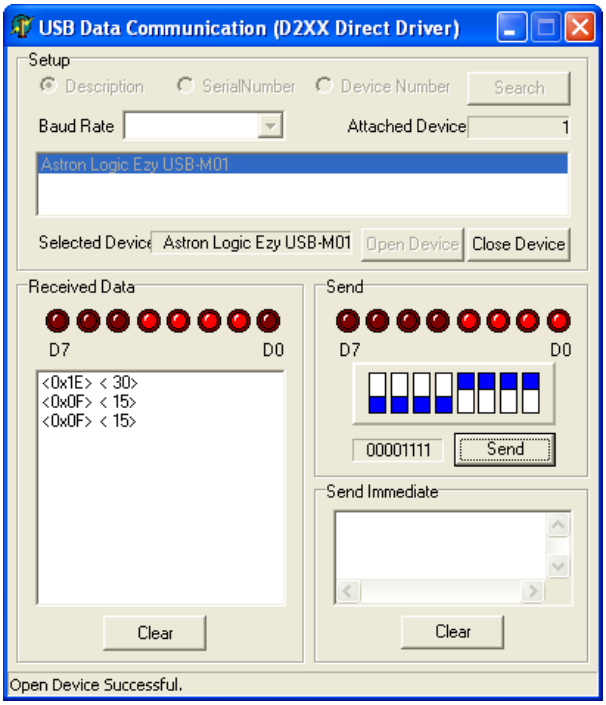
การทดลองจะใช้โปรแกรมประยุกต์ตัวอย่างทำการส่งข้อมูลขนาด 8 บิต จากเครื่อง PC ไปสู่บอร์ด FPGA แล้วให้บอร์ด FPGA ส่งข้อมูลขนาด 8 บิตที่ได้รับกลับมายังเครื่อง PC อีกครั้ง โดยสามารถตรวจสอบผลการทำงานของบอร์ด FPGA จากโปรแกรมประยุกต์ตัวอย่าง ซึ่งได้ผลการทดลองดังภาพที่ 29 ได้ทำการส่งข้อมูลจากเครื่อง PC เป็นเลขฐานสอง คือ “00001111” ในส่วนของ Send และได้รับข้อมูลจากบอร์ด FPGA กลับมาแสดงผลเป็นเลขฐานสิบหกเป็น “0x0F” และเลขฐานสิบเป็น “15” ซึ่งเป็นข้อมูลค่าเดียวกันกับที่ได้ส่งออกไป



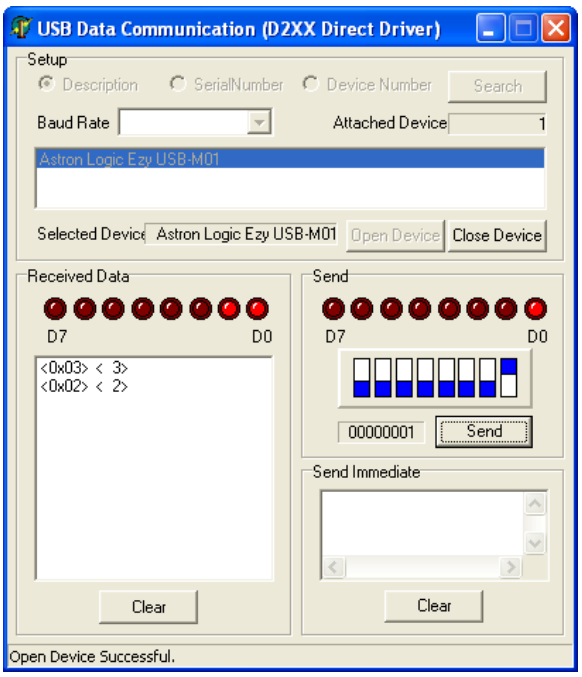
ภาพที่ 29 ผลการทดลองกรณีข้อมูลที่รับส่งมีขนาดเท่ากัน

1.1.2 กรณีข้อมูลที่รับส่งมีขนาดไม่เท่ากัน

การทดลองจะแบ่งออกเป็น 2 กรณี คือ กรณีข้อมูลที่บอร์ด FPGA รับเข้ามีจำนวนน้อยกว่าข้อมูลที่ส่งออก และกรณีข้อมูลที่บอร์ด FPGA รับเข้ามีจำนวนมากกว่าข้อมูลที่ส่งออก โดยในกรณีแรกจะทำการส่งข้อมูลจากเครื่อง PC ออกไปจำนวน 2 ชุด ขนาดชุดละ 8 บิต รวมแล้วเป็น 16 บิต เข้าสู่บอร์ด FPGA แล้วให้บอร์ด FPGA ทำการบวกข้อมูลทั้ง 2 ชุดเข้าด้วยกัน แล้วส่งข้อมูล 2 ชุดแรกออกมาพร้อมกับข้อมูลชุดที่ 3 ดังภาพที่ 30 ซึ่งข้อมูล 2 ชุดแรกที่ส่งออกจากเครื่อง PC มีค่าเท่ากับ “15” ในฐานสิบ และข้อมูลชุดที่ 3 เกิดจากข้อมูล 2 ชุดแรกบวกกันจึงมีค่าเท่ากับ “30” ในเลขฐานสิบ รวมข้อมูลที่เครื่อง PC ได้รับทั้งหมดคือ 24 บิต จะเห็นว่ามีการส่งข้อมูลออกจากเครื่อง PC จำนวนน้อยกว่าข้อมูลที่เครื่อง PC ได้รับกลับจากบอร์ด FPGA สำหรับในกรณีที่สองจะทำการส่งข้อมูลจากเครื่อง PC ออกไปจำนวน 3 ชุด ขนาดชุดละ 8 บิต รวมแล้วเป็น 24 บิต เข้าสู่บอร์ด FPGA แล้วให้บอร์ด FPGA ทำการบวกข้อมูลชุดที่ 1 เข้ากับข้อมูลชุดที่ 3 และบวกข้อมูลชุดที่ 2 เข้ากับข้อมูลชุดที่ 3 ดังภาพที่ 31 ซึ่งจะเห็นว่ามีข้อมูล 2 ชุดที่แสดงผลอยู่เกิดจากการส่งข้อมูลชุดที่ 1 เป็น “1” ในเลขฐานสิบ ส่งข้อมูลชุดที่ 2 เป็น “2” ในเลขฐานสิบ และส่งข้อมูลชุดที่ 3 เป็น “1” ในเลขฐานสิบ โดย “2” เกิดจาก “1” บวกกับ “1” และ “3” เกิดจาก “2” บวกกับ “1”



ภาพที่ 30 ผลการทดลองกรณีข้อมูลที่รับเข้ามีจำนวนน้อยกว่าข้อมูลที่ส่งออก



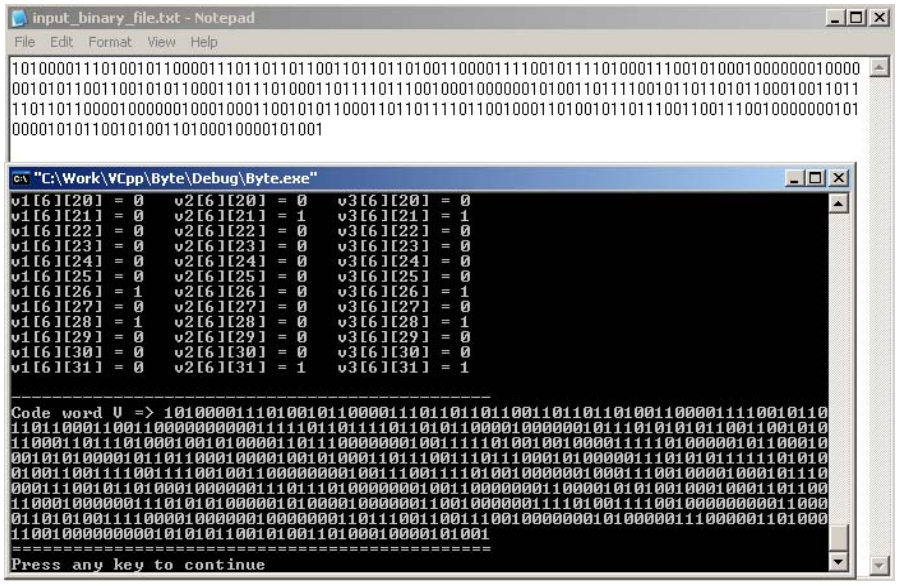
ภาพที่ 31 ผลการทดลองกรณีข้อมูลที่รับเข้ามีจำนวนมากกว่าข้อมูลที่ส่งออก

2. การเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต

2.1 การจำลองการทำงาน

การจำลองการเข้ารหัสคอนโวลูชัน (3, 2, 2) จะเขียนโปรแกรมด้วยภาษา C++ ขึ้นมาแล้วทำการป้อนข้อมูลเพื่อทำการเข้ารหัสข้อมูล โดยผลการจำลองการทำงานที่ได้จะนำไปใช้สำหรับเปรียบเทียบผลการทำงานของตัวเข้ารหัสคอนโวลูชันที่สร้างขึ้นด้วยบอร์ด FPGA ต่อไป จากภาพที่ 32 เป็นผลการจำลองการเข้ารหัสคอนโวลูชัน (3 ,2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิตด้วยโปรแกรมภาษา C++ โดยจะทำการป้อนข้อมูลที่อยู่ในรูปแบบไบนารีดังรูปเข้าสู่โปรแกรม ซึ่งโปรแกรมจะทำการคำนวณหาคำรหัสให้ คำรหัสที่ได้จะมีความยาวมากกว่าข้อมูลที่ป้อนตามทฤษฎี เนื่องจากอัตรารหัสเป็น 2/3 นั่นเอง สำหรับในภาพที่ 33 จะเป็นการจำลองการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต ที่เขียนด้วยโปรแกรมภาษา VHDL โดยจะมีการกำหนดให้มีสัญญาณต่างๆ ได้แก่ สัญญาณนาฬิกา (CLK) ใช้สำหรับกระตุ้นการทำงาน สัญญาณรีเซต (RST) ใช้สำหรับทำให้ค่าสัญญาณเอาต์พุตเป็นค่าศูนย์ สัญญาณอินพุตขนาด 32 บิต จำนวน 2 อินพุต คือ u1[31:0] และ u2[31:0] โดยมีการเรียงลำดับข้อมูลบิตที่มีความสำคัญสูงสุดอยู่ด้านซ้ายไปหาบิตที่มีความสำคัญต่ำสุดอยู่ด้านขวา และสัญญาณเอาต์พุตขนาด 32 บิต จำนวน 3

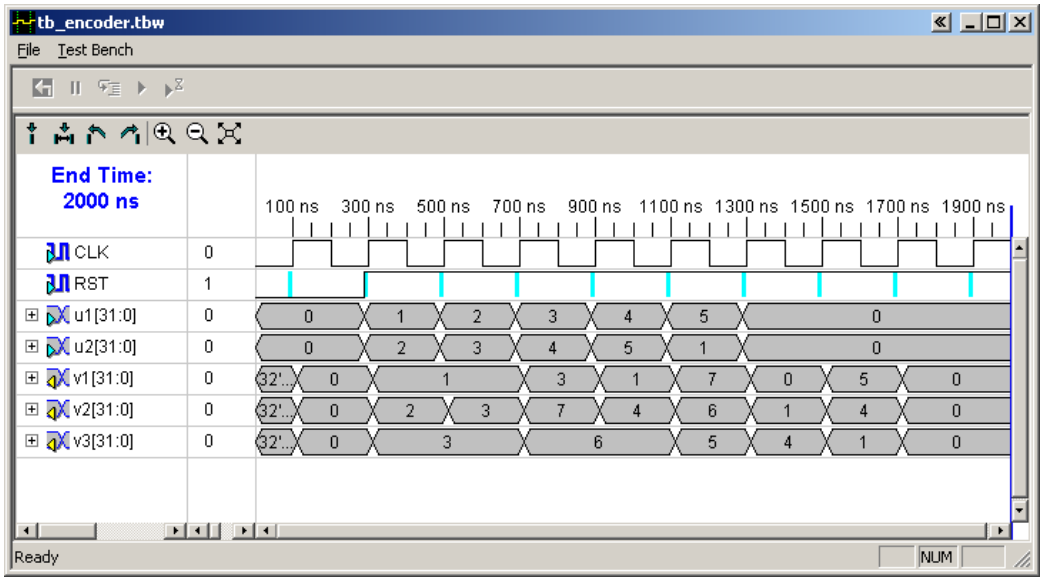
เอาต์พุต คือ v1[31:0], v2[31:0] และ v3[31:0] โดยมีการเรียงลำดับข้อมูลเช่นเดียวกันกับสัญญาณอินพุต ในการทดลองจะกำหนดชุดข้อมูลเข้าจำนวน 2 ชุด และเพื่อให้ง่ายในการตรวจสอบการทำงานจะกำหนดให้มีการแสดงผลเป็นเลขฐานสิบ โดยหลังจากที่มีการรีเซตจะกำหนดให้ข้อมูลอินพุต u1[31:0] เป็น 1, 2, 3, 4, 5 และ 0 ตามลำดับ กำหนดให้อินพุต u2[31:0] เป็น 2, 3, 4, 5, 1 และ 0 ตามลำดับ ซึ่งจะทำให้ได้ชุดข้อมูลเอาต์พุต 3 ชุด ได้แก่ ในขณะที่ u1[31:0] เป็น 1 และ u2[31:0] เป็น 2 จะได้ v1[31:0] เป็น 1 , v2[31:0] เป็น 2 และ v3[31:0] เป็น 3 ในขณะที่ u1[31:0] เป็น 2 และ u2[31:0] เป็น 3 จะได้ v1[31:0] เป็น 1 , v2[31:0] เป็น 3 และ v3[31:0] เป็น 3 ในขณะที่ u1[31:0] เป็น 3 และ u2[31:0] เป็น 4 จะได้ v1[31:0] เป็น 3 , v2[31:0] เป็น 7 และ v3[31:0] เป็น 6 ในขณะที่ u1[31:0] เป็น 4 และ u2[31:0] เป็น 5 จะได้ v1[31:0] เป็น 1 , v2[31:0] เป็น 4 และ v3[31:0] เป็น 6 ในขณะที่ u1[31:0] เป็น 5 และ u2[31:0] เป็น 1 จะได้ v1[31:0] เป็น 7 , v2[31:0] เป็น 6 และ v3[31:0] เป็น 5 และเมื่อไม่มีการป้อนข้อมูลอินพุตหรือข้อมูลอินพุตเป็น 0 ก็ยังมีค่าเอาต์พุตที่เกิดจากค่าสัญญาณที่เก็บไว้ในหน่วยความจำอีก 2 ชุด ได้แก่ v1[31:0] เป็น 0 , v2[31:0] เป็น 1 และ v3[31:0] เป็น 4 เป็นชุดแรก v1[31:0] เป็น 5 , v2[31:0] เป็น 4 และ v3[31:0] เป็น 1 เป็นชุดสุดท้าย



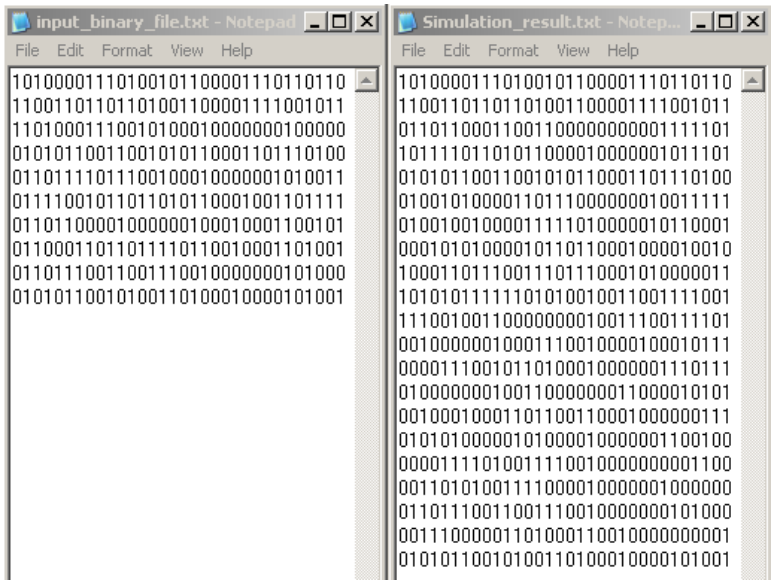
ภาพที่ 32 ผลการจำลองการเข้ารหัสคอนโวลูชันด้วยโปรแกรมภาษา C++

ในการใช้งานจริงข้อมูลที่ออกจากระบบการเข้ารหัสคอนโวลูชัน (3, 2, 2) จะมีการเรียงต่อกันดังภาพที่ 34 ซึ่งจะเปรียบเทียบให้เห็นจำนวนข้อมูลในแฟ้มข้อมูลที่เป็นข้อมูลเข้าและใน

แฟ้มข้อมูลที่เป็นข้อมูลออก โดยในแฟ้มข้อมูลเข้าจะมีจำนวนสัญลักษณ์ขนาด 32 บิต เป็น 10 สัญลักษณ์ และในแฟ้มข้อมูลออกจะมีจำนวนสัญลักษณ์ขนาด 32 บิต เป็น 21 สัญลักษณ์



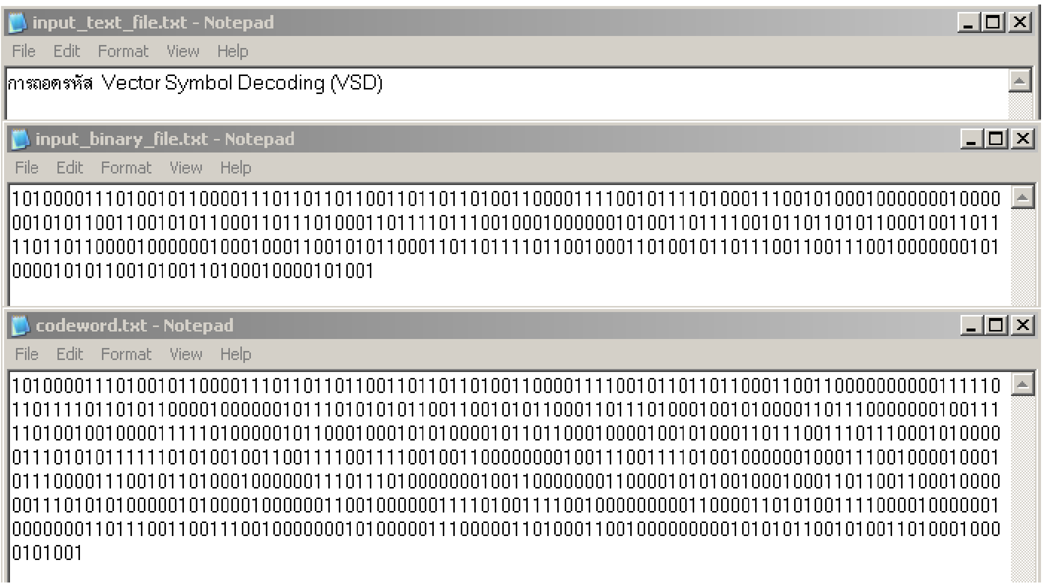
ภาพที่ 33 ผลการจำลองการเข้ารหัสคอนโวลูชันด้วยโปรแกรมภาษา VHDL



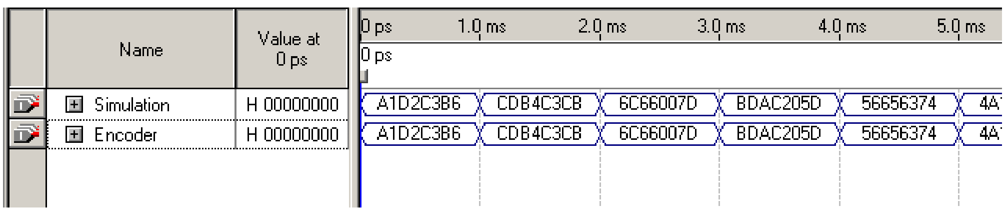
ภาพที่ 34 ผลการจำลองการเข้ารหัสคอนโวลูชันในรูปแบบแฟ้มข้อมูล

2.2 การประมวลผลด้วยบอร์ด FPGA

การทดลองจะเริ่มจากนำข้อมูลที่อยู่ในรูปแบบข้อความ (Text) มาทำการแปลงให้อยู่ในรูปแบบไบนารี แล้วจากนั้นจะทำการป้อนข้อมูลที่อยู่ในรูปแบบไบนารีเข้าสู่บอร์ด FPGA ด้วยโปรแกรมภาษา C++ ที่เขียนขึ้น โดยกำหนดให้มีการส่งข้อมูลจากเครื่อง PC เข้าสู่บอร์ด FPGA รอบละ 32 บิตจำนวน 2 สัญลักษณ์ รวมเป็น 64 บิต แล้วรอรับผลที่ได้จากการทำงานของบอร์ด FPGA กลับมาบันทึกลงในเครื่อง PC ให้อยู่ในรูปแบบไบนารี ดังภาพที่ 35 แฟ้มข้อมูลบนสุดที่แสดงเป็นข้อมูลในรูปแบบข้อความที่จะนำมาเข้ารหัส แฟ้มข้อมูลในตำแหน่งกลางเกิดจากการแปลงรูปแบบข้อมูลให้เป็นแบบไบนารี หรือมองเป็นแบบนอนไบนารี ส่วนแฟ้มข้อมูลล่างสุดที่แสดงเป็นข้อมูลที่ผ่านการเข้ารหัสแล้ว ในภาพที่ 36 เปรียบเทียบผลการเข้ารหัสด้วยบอร์ด FPGA และผลการจำลองการทำงานด้วยโปรแกรมภาษา C++ ซึ่งได้ผลการทำงานที่ตรงกัน



ภาพที่ 35 ผลการทดลองการเข้ารหัสคอนโวลูชันด้วยบอร์ด FPGA



ภาพที่ 36 เปรียบเทียบผลการเข้ารหัสคอนโวลูชันด้วยบอร์ด FPGA และผลการจำลองการทำงาน

2.3 ประสิทธิภาพการเข้ารหัสคอนโวลูชันด้วยบอร์ด FPGA

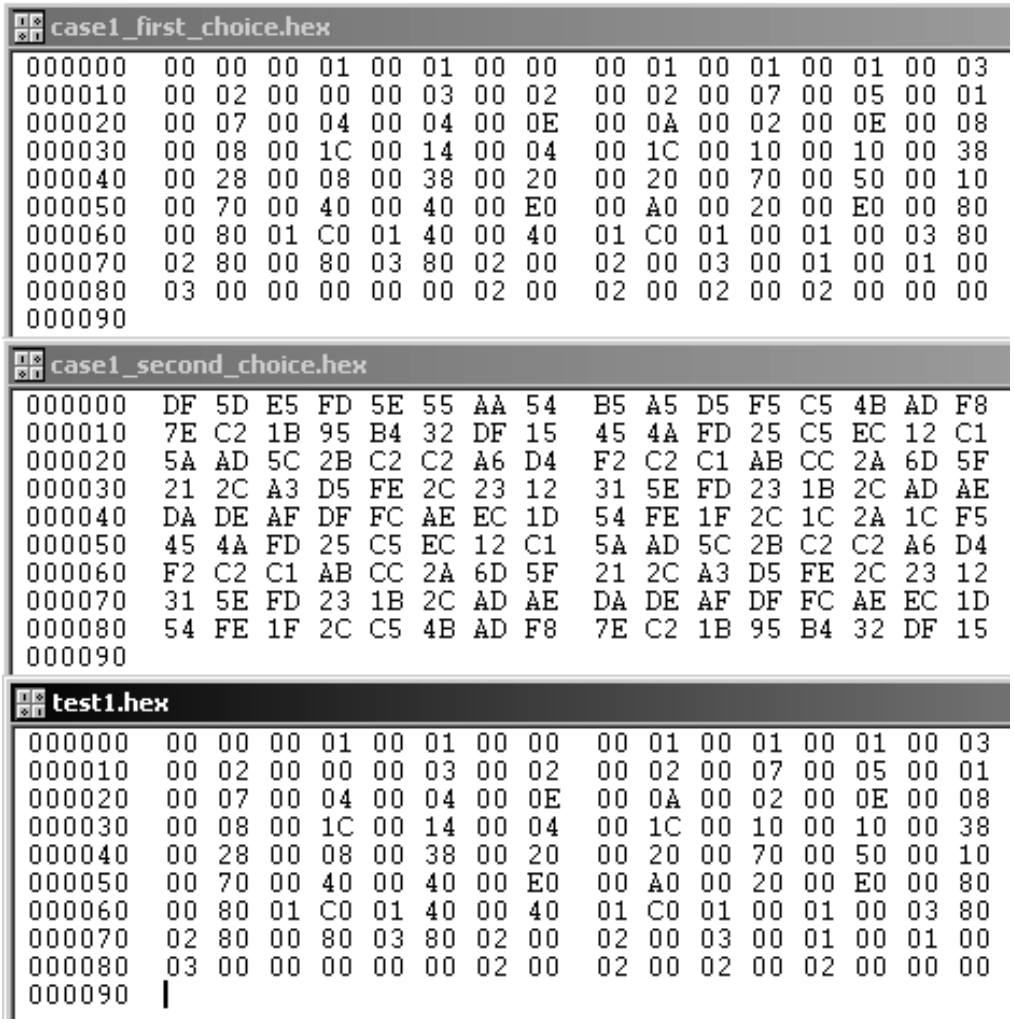
จากระบบการเข้ารหัสคอนโวลูชันที่ได้ออกแบบ มีวัตถุประสงค์หลักเพื่อใช้สำหรับทดสอบความถูกต้องในการทำงานเท่านั้น โดยประสิทธิภาพการทำงานในแง่ของความเร็วการประมวลผล พบว่ามีข้อจำกัดในส่วนของอุปกรณ์ที่นำมาใช้ออกแบบ ประการแรก คือ บอร์ด FPGA สามารถประมวลผลได้ที่ความเร็วสูงสุด 25 เมกะเฮริตซ์ ซึ่งระบบที่ออกแบบสามารถทำงานได้ที่ความเร็วสูงสุดของบอร์ด FPGA ที่นำมาใช้งาน ประการที่สอง คือ โมดูล USB ที่นำมาใช้งานมีอัตราการรับส่งข้อมูลสูงสุดที่ 1 เมกะไบต์ต่อวินาที หรือ 8 เมกะบิตต่อวินาที จึงทำให้เกิดคอขวดในการทำงานของระบบที่จุดของการเชื่อมต่อนี้ สำหรับการนำอุปกรณ์การเข้ารหัสไปใช้งานจริงจะมีความแตกต่างจากระบบการเข้ารหัสที่ได้ออกแบบ นั่นคือจะมองระบบการเข้ารหัสเป็นเพียงฟังก์ชันหนึ่งที่อยู่ในระบบ ซึ่งทำหน้าที่การเข้ารหัสเพียงอย่างเดียวเท่านั้น

3. การถอดรหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต ด้วยวิธีเวกเตอร์ซิมโบล

3.1 การถอดรหัสคอนโวลูชันด้วยวิธีเวกเตอร์ซิมโบลโดยใช้บอร์ด FPGA

การทดลองการประมวลผลการถอดรหัสจะแบ่งออกเป็น 2 กรณี คือ กรณีสัญลักษณ์ในชุดข้อมูลหลักถูกต้องทั้งหมด และกรณีสัญลักษณ์ในชุดข้อมูลหลักมีความผิดพลาดโดยที่สัญลักษณ์ในชุดข้อมูลสำรองมีค่าที่ถูกต้องในตำแหน่งเดียวกันกับชุดข้อมูลหลัก โดยทำการป้อนข้อมูลเข้าไปสู่บอร์ด FPGA จำนวน 2 ชุดข้อมูลในรูปแบบที่แสดงผลเป็นเลขฐานสิบหก โดยแบ่งเป็น 4 ชุดที่แสดง มีค่าเท่ากับ 1 สัญลักษณ์ เนื่องจากสัญลักษณ์ที่ใช้มีขนาดเป็น 32 บิต และในส่วนของผลการทำงานก็จะบันทึกกลับสู่เครื่อง PC ก็แสดงผลในรูปแบบเลขฐานสิบหกเช่นกัน เพื่อความสะดวกในการตรวจสอบการทำงาน

กรณีสัญลักษณ์ในชุดข้อมูลหลักถูกต้องทั้งหมด จะส่งข้อไฟล์ข้อมูลชื่อ case1_first_choice.hex เป็นชุดข้อมูลหลัก และไฟล์ข้อมูลชื่อ case1_second_choice.hex เป็นชุดข้อมูลสำรอง โดยกำหนดให้ข้อมูลในชุดข้อมูลหลักไม่มีความผิดพลาดเลย และในส่วนของชุดข้อมูลสำรองมีความผิดพลาดแบบสุ่ม ซึ่งผลการทดลองที่ได้จะเป็นดังภาพที่ 37 คือไฟล์ชื่อ test1.hex จะมีข้อมูลเหมือนกันกับไฟล์ชื่อ case1_second_choice.hex เนื่องจากไม่มีความผิดพลาดในชุดข้อมูลหลักเลย



ภาพที่ 37 ผลการทดลองกรณีสัญลักษณ์ในชุดข้อมูลหลักถูกต้องทั้งหมด

กรณีสัญลักษณ์ในชุดข้อมูลหลักมีความผิดพลาดโดยที่สัญลักษณ์ในชุดข้อมูลสำรองมีค่าที่ถูกต้องในตำแหน่งเดียวกันกับชุดข้อมูลหลัก จะส่งข้อไฟล์ข้อมูลชื่อ case2_first_choice.hex เป็นชุดข้อมูลหลัก และไฟล์ข้อมูลชื่อ case2_second_choice.hex เป็นชุดข้อมูลสำรอง โดยกำหนดให้ข้อมูลในชุดข้อมูลหลักมีความผิดพลาดเพียง 1 สัญลักษณ์ใน 3 สัญลักษณ์ และในส่วนของชุดข้อมูลสำรองไม่มีความผิดพลาดในตำแหน่งเดียวกันกับชุดข้อมูลหลักที่มีความผิดพลาด ซึ่งผลการทดลองที่ได้จะเป็นดังภาพที่ 38 คือไฟล์ชื่อ test2.hex จะแก้ไขชุดข้อมูลหลักในตำแหน่งสัญลักษณ์ที่มีความผิดพลาดให้ถูกต้อง โดยอาศัยชุดข้อมูลสำรอง

case2_first_choice.hex

```
000000  00 00 00 01 00 01 00 00  00 01 00 01 C5 4B AD F8
000010  00 02 00 00 00 03 00 02  00 02 00 07 00 05 00 01
000020  00 07 00 04 00 04 00 0E  F2 C2 C1 AB 00 0E 00 08
000030  00 08 00 1C 00 14 00 04  31 5E FD 23 00 10 00 38
000040  00 28 00 08 FC AE EC 1D  00 20 00 70 00 50 00 10
000050  00 70 00 40 00 40 00 E0  00 A0 00 20 00 E0 00 80
000060  00 80 01 C0 CC 2A 6D 5F  01 C0 01 00 FE 2C 23 12
000070  02 80 00 80 03 80 02 00  02 00 03 00 01 00 01 00
000080  03 00 00 00 00 00 02 00  02 00 02 00 02 00 00 00
000090
```

case2_second_choice.hex

```
000000  DF 5D E5 FD 5E 55 AA 54  B5 A5 D5 F5 00 01 00 03
000010  7E C2 1B 95 B4 32 DF 15  45 4A FD 25 C5 EC 12 C1
000020  5A AD 5C 2B C2 C2 A6 D4  00 0A 00 02 CC 2A 6D 5F
000030  21 2C A3 D5 FE 2C 23 12  00 1C 00 10 1B 2C AD AE
000040  DA DE AF DF 00 38 00 20  54 FE 1F 2C 1C 2A 1C F5
000050  45 4A FD 25 C5 EC 12 C1  5A AD 5C 2B C2 C2 A6 D4
000060  F2 C2 C1 AB 01 40 00 40  21 2C A3 D5 01 00 03 80
000070  31 5E FD 23 1B 2C AD AE  DA DE AF DF FC AE EC 1D
000080  54 FE 1F 2C C5 4B AD F8  7E C2 1B 95 B4 32 DF 15
000090
```

test2.hex

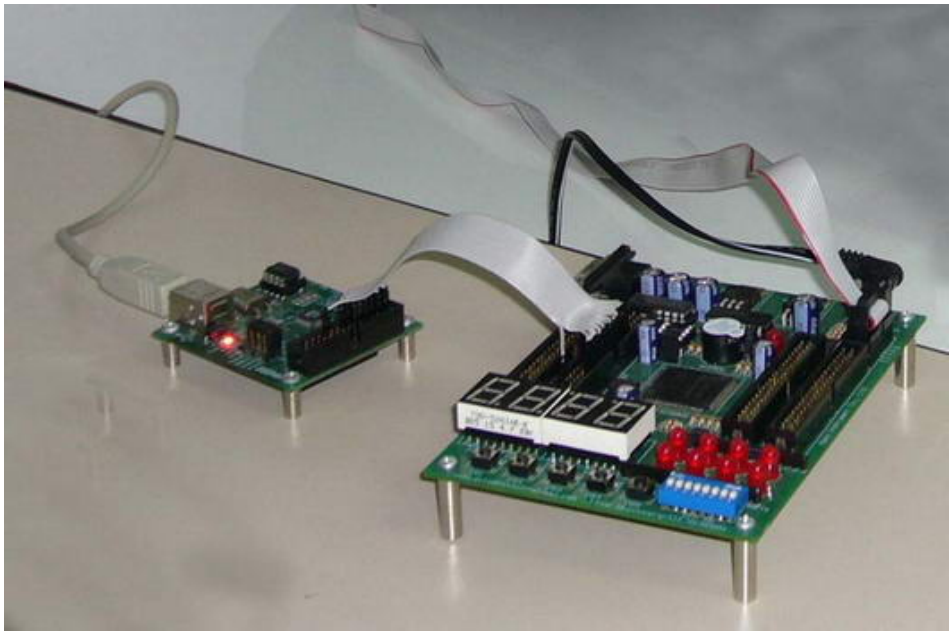
```
000000  00 00 00 01 00 01 00 00  00 01 00 01 00 01 00 03
000010  00 02 00 00 00 03 00 02  00 02 00 07 00 05 00 01
000020  00 07 00 04 00 04 00 0E  00 0A 00 02 00 0E 00 08
000030  00 08 00 1C 00 14 00 04  00 1C 00 10 00 10 00 38
000040  00 28 00 08 00 38 00 20  00 20 00 70 00 50 00 10
000050  00 70 00 40 00 40 00 E0  00 A0 00 20 00 E0 00 80
000060  00 80 01 C0 01 40 00 40  01 C0 01 00 01 00 03 80
000070  02 80 00 80 03 80 02 00  02 00 03 00 01 00 01 00
000080  03 00 00 00 00 00 02 00  02 00 02 00 02 00 00 00
000090
```

ภาพที่ 38 ผลการทดลองกรณีสัญลักษณ์ในชุดข้อมูลหลักมีความผิดพลาดโดยสัญลักษณ์ในชุดข้อมูลสำรองมีค่าที่ถูกต้อง

3.2 ประสิทธิภาพการถอดรหัสคอนโวลูชันด้วยวิธีเวกเตอร์ซิมโบลโดยใช้บอร์ด FPGA

เช่นเดียวกันกับระบบการเข้ารหัสคอนโวลูชันที่ได้ออกแบบ คือ มีวัตถุประสงค์หลักเพื่อใช้สำหรับทดสอบความถูกต้องในการทำงานเท่านั้น โดยมีข้อจำกัดในส่วนของอุปกรณ์ที่นำมาใช้ออกแบบ จึงทำให้เกิดคอขวดในการทำงานของระบบที่จุดของการเชื่อมต่อ สำหรับการนำอุปกรณ์การถอดรหัสไปใช้งานจริงจะมีความแตกต่างจากระบบที่ออกแบบ ซึ่งยังต้องมีการพัฒนาระบบต่อไป เพื่อให้สามารถนำไปใช้งานจริงได้

ภาพที่ 39 ระบบการเข้ารหัสถอดรหัสที่เชื่อมต่อกับเครื่องคอมพิวเตอร์



ภาพที่ 40 ระบบการเข้ารหัสถอดรหัสที่สร้างด้วยบอร์ด FPGA

วิจารณ์

1. การติดต่อสื่อสารระหว่างเครื่อง PC และบอร์ด FPGA

ในการรับส่งข้อมูลระหว่างเครื่อง PC และบอร์ด FPGA นั้น การเข้าใจการทำงานโมดูล USB นับว่าเป็นส่วนที่สำคัญในการออกแบบระบบเชื่อมต่อ ดังนั้น จึงจำเป็นต้องเข้าใจการรับส่งข้อมูลก่อน เพื่อสามารถนำมาประยุกต์ใช้งานต่อไปได้ ในการทดลองได้ทำการทดลองรับส่งข้อมูลในรูปแบบที่มีการรับข้อมูลเพียงอย่างเดียวและการส่งข้อมูลเพียงอย่างเดียวก่อน โดยหลังจากเข้าใจการทำงานแล้วก็สามารถเพิ่มรูปแบบที่มีการรับข้อมูลแล้วทำการส่งข้อมูลกลับมาด้วย ซึ่งได้ทำการทดลองให้เห็นไปแล้ว และหลังจากนั้นก็สามารถนำแนวคิดการรับส่งข้อมูลมาประยุกต์ใช้กับรูปแบบที่ต้องการได้ เช่น การรับส่งข้อมูลที่มีขนาดไม่เท่ากัน โดยอาจจะมีการรับข้อมูลจำนวนมากกว่าการส่งข้อมูล หรืออาจจะมีการรับข้อมูลจำนวนหนึ่งแล้วมีการส่งข้อมูลที่จำนวนมากกว่าออกไป เป็นต้น ข้อจำกัดของโมดูล USB ก็คือขนาดบัสข้อมูลที่มีขนาด 8 บิต แต่ในการใช้งานจริงข้อมูลมีขนาด 32 บิตต่อสัญลักษณ์ ดังนั้น จึงต้องมีการประยุกต์ให้เข้ากับความต้องการ

2. การเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต

การทดลองในส่วนของการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต จะแบ่งออกเป็น การทดลองโดยการจำลองการทำงานด้วยซอฟต์แวร์ทั้งที่เป็นโปรแกรมที่เขียนด้วยภาษา C++ และที่เป็นโปรแกรมที่เขียนด้วยภาษา VHDL แล้วนำผลที่ได้มาเปรียบเทียบกันเพื่อตรวจสอบความถูกต้องในการออกแบบการทำงาน ซึ่งได้ทำการทดลองให้เห็นไปแล้ว และจากนั้นจะทำการจัดสร้างให้สามารถเข้ารหัสด้วยบอร์ด FPGA โดยต้องอาศัยวิธีการรับส่งข้อมูลที่กล่าวไปแล้วข้างต้นในกรณีที่มีการรับข้อมูลมีจำนวนที่น้อยกว่าการส่งข้อมูลมาใช้งานด้วย เนื่องจากอัตราของการเข้ารหัสข้อมูลเป็น 2 ต่อ 3 และรวมถึงการออกแบบให้สามารถประมวลผลการเข้ารหัสด้วย

ผลการทดลองการเข้ารหัสแสดงผลในรูปแบบข้อมูลเลขฐานสอง เพื่อความสะดวกในการตรวจสอบผลการทำงานของบอร์ด FPAG ที่ทำงานเป็นตัวเข้ารหัส สามารถนำผลที่ได้มาทำการเปรียบเทียบกับผลการจำลองการทำงานได้ง่าย สำหรับประสิทธิภาพการทำงานในแง่ความเร็วในการประมวลผลมีข้อจำกัดในส่วนของโมดูล USB ที่รองรับอัตราการส่งถ่ายข้อมูลเพียง 1 เมกะไบต์ต่อวินาทีเท่านั้น

3. การถอดรหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต ด้วยวิธีเวกเตอร์ซิมโบล

การทดลองในส่วนของการถอดรหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต ด้วยวิธีเวกเตอร์ซิมโบลด้วยซินโดรมเดียว จะกำหนดกรณีเป็น 2 กรณีในการทดลอง โดยกรณีที่สัญลักษณ์ในชุดข้อมูลหลักถูกต้องทั้งหมด ผลที่ได้จะเหมือนกันกับสัญลักษณ์ในชุดข้อมูลหลักทุกประการ เนื่องจากไม่ต้องทำการแก้ไข และในกรณีนี้ก็ไม่จำเป็นต้องใช้ชุดข้อมูลสำรองมาช่วย แต่ในกรณีที่สัญลักษณ์ในชุดข้อมูลหลักมีความผิดพลาดไม่เกิน 1 สัญลักษณ์ใน 3 สัญลักษณ์ โดยที่สัญลักษณ์ในชุดข้อมูลสำรองในตำแหน่งตรงกันกับสัญลักษณ์ในชุดข้อมูลหลักมีค่าที่ถูกต้อง ผลที่ได้ก็จะเหมือนกันกับการทดลองในกรณีที่สัญลักษณ์ในชุดข้อมูลหลักถูกต้องทั้งหมด เนื่องจากตัวถอดรหัสได้ทำการแก้ไขสัญลักษณ์ที่ผิดพลาดในชุดข้อมูลหลักให้ถูกต้องโดยใช้สัญลักษณ์ในชุดข้อมูลสำรองได้

ผลการทดลองการถอดรหัสแสดงผลในรูปแบบของเลขฐานสิบหก เนื่องจากสัญลักษณ์ที่ใช้งานมีขนาด 32 บิตต่อสัญลักษณ์ ซึ่งทำให้มีความง่ายในการตรวจสอบการทำงานของบอร์ด FPGA ที่ทำงานเป็นตัวถอดรหัส โดยในส่วนของประสิทธิภาพการทำงานในแง่ของความเร็วในการประมวลผลก็มีข้อจำกัดเช่นเดียวกันกับระบบการเข้ารหัส ซึ่งเกิดจากระบบทดสอบที่ได้ออกแบบ

สรุปและข้อเสนอแนะ

สรุป

การออกแบบระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีเลือกใช้ข้อมูลที่เป็นสัญลักษณ์นอนไบนารีขนาด 32 บิตต่อสัญลักษณ์ ใช้วิธีการเข้ารหัสแบบคอนโวลูชัน (3, 2, 2) ในส่วนของระบบการเข้ารหัส ใช้วิธีการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลในส่วนของระบบการถอดรหัส โดยทำการสร้างระบบที่ได้ออกแบบลงบนบอร์ด FPGA เพื่อศึกษาการทำงานของระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารี ซึ่งจะทำให้ทราบข้อดีและข้อด้อยของวิธีการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีในการนำมาสร้างเป็นอุปกรณ์จริง และผลการทดลองที่ได้สามารถนำไปใช้พัฒนาระบบการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลต่อไป

ระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีจะต้องมีการออกแบบ 2 ส่วนหลัก คือ การออกแบบส่วนฮาร์ดแวร์และการออกแบบส่วนซอฟต์แวร์ซึ่งมีความสัมพันธ์กัน โดยในส่วนฮาร์ดแวร์นั้น จะต้องทำความเข้าใจการทำงานของอุปกรณ์ที่ใช้งานก่อน เนื่องจากจะช่วยลดความผิดพลาดในการออกแบบซอฟต์แวร์ การเชื่อมต่อประสานฮาร์ดแวร์เข้าด้วยกันก็นับว่ามีความสำคัญ โดยข้อมูลในการเชื่อมต่อประสานต้องนำไปใช้ในการออกแบบซอฟต์แวร์ด้วย เพื่อทำให้อุปกรณ์ในระบบสามารถติดต่อสื่อสารกันได้ ซึ่งในระบบการเข้ารหัสถอดรหัสนี้ได้เลือกโมดูล USB ในการติดต่อสื่อสารระหว่างเครื่อง PC และบอร์ด FPGA โดยการติดต่อสื่อสารของระบบการเข้ารหัสถอดรหัสสัญลักษณ์นอนไบนารีจะมีความคล้ายคลึงกัน ซึ่งประยุกต์การทำงานให้เหมาะสมกับระบบที่ออกแบบ

ระบบการเข้ารหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต มีการออกแบบโดยจำลองการทำงานก่อนด้วยโปรแกรมภาษา C++ และโปรแกรมภาษา VHDL เพื่อตรวจสอบการทำงานของระบบว่าสามารถทำงานได้ถูกต้องตามที่ออกแบบ แล้วหลังจากนั้นจะทำการออกแบบแผนภาพสเตตการทำงาน เพื่อใช้สำหรับการเขียนโปรแกรมภาษา VHDL ก่อนโปรแกรมลงสู่บอร์ด FPGA ซึ่งผลการทดลองพบว่าระบบที่ออกแบบสามารถทำงานได้ผลตรงกันกับผลการจำลองการทำงาน

ระบบการถอดรหัสคอนโวลูชัน (3, 2, 2) สัญลักษณ์นอนไบนารีขนาด 32 บิต ด้วยวิธีเวกเตอร์ซิมโบลบางส่วน ซึ่งได้ออกแบบแผนภาพสเตตการทำงานก่อน เช่นเดียวกันกับระบบการเข้ารหัส โดยจากผลการทดลองพบว่าระบบสามารถแก้ไขสัญลักษณ์ที่มีความผิดพลาดในชุดข้อมูลหลักโดยใช้ลัญลักษณ์ในชุดข้อมูลสำรองได้ตามที่ออกแบบไว้

ข้อเสนอแนะ

การออกแบบและสร้างระบบการเข้ารหัสถอดรหัสจำเป็นต้องออกแบบทั้งในส่วนของฮาร์ดแวร์และส่วนของซอฟต์แวร์ โดยควรทำการศึกษาการทำงานของฮาร์ดแวร์ที่ใช้ก่อนดำเนินการออกแบบซอฟต์แวร์ เนื่องจากการออกแบบซอฟต์แวร์มีความยืดหยุ่นมากกว่า ในการออกแบบระบบการเข้ารหัสถอดรหัสด้วยโปรแกรมภาษา VHDL จำเป็นต้องมีการออกแบบแผนภาพสเตตการทำงาน ซึ่งจะทำให้ช่วยลดข้อผิดพลาดในการเขียนโปรแกรม และทำให้ผู้ที่นำระบบไปพัฒนาต่อสามารถทำความเข้าใจได้ง่ายอีกด้วย

งานในอนาคต

พัฒนาระบบการถอดรหัสด้วยวิธีเวกเตอร์ซิมโบลให้มีประสิทธิภาพ โดยสามารถประมวลผลฟังก์ชันที่มีความซับซ้อนมากขึ้นได้ และนำเทคโนโลยีการออกแบบวงจรรวมใหม่ๆ มาประยุกต์ใช้กับการพัฒนาระบบ

เอกสารและสิ่งอ้างอิง

ชำนาญ ปัญญาใส และ วัชรากร หนูทอง. 2547. **ภาษา VHDL สำหรับการออกแบบวงจรดิจิตอล.** ซีเอ็ดยูเคชั่น, กรุงเทพฯ.

วราเทพ ไพบูลย์รัตนากร และ บุญอนันต์ เกียงเอีย. ม.ป.ป.. **สัมผัสโลก USB ด้วย Ezy USB Module.** แอสทรอนลอจิกรีเสิร์ชแอนดิวิลอปเมนต์, กรุงเทพฯ.

อุศนา ตัณฑุลเวศม์. 2549. **รายงานโครงการวิจัยชิ้นงานต้นแบบเครื่องถอดรหัสแบบเวกเตอร์ซิมโบล สำหรับรหัสคอนโวลูชันเฟสที่หนึ่ง.** สถาบันวิจัยและพัฒนาแห่งมหาวิทยาลัยเกษตรศาสตร์.

อุศนา ตัณฑุลเวศม์. 2550. **รายงานโครงการวิจัยชิ้นงานต้นแบบเครื่องเข้ารหัสที่เหมาะสมกับเครื่องถอดรหัสแบบเวกเตอร์ซิมโบล.** สถาบันวิจัยและพัฒนาแห่งมหาวิทยาลัยเกษตรศาสตร์.

APEX Instrument. n.d.. **User Manual: FPGA Discovery-III XC3200.** Bangkok.

Astronlogic Research and Development. n.d.. **User Manual: Ezy USB-M01.** Bangkok.

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

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

L, Douglas Perry. 2002. **VHDL: programming by example.** McGraw-Hill, Boston.

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

R, Intharasakul 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.**

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

U, Tuntoolavest 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.

U, Tuntoolavest and R. Intharasakul. 2006. Nonbinary Convolutional Encoder for Vector Symbol Decoding for Vector Symbol Decoding on FPGA board. **IEEE International Conference on Communications Technology 2006.**

ภาคผนวก

ภาคผนวก ก

ข้อมูลทางฮาร์ดแวร์

ตารางผนวกที่ ก1 รายละเอียดของอุปกรณ์ที่ต่ออยู่กับขา FPGA

7-Segment	FPGA Pinout	Descriptions
A	p40	A
B	p35	B
C	p32	C
D	p30	D
E	p27	E
F	p25	F
G	p23	G
Dp	p20	Decimal Point
DIGIT1	p31	DIGIT1, COMMON CATHODE
DIGIT2	p33	DIGIT2, COMMON CATHODE
DIGIT3	p36	DIGIT3, COMMON CATHODE
DIGIT4	p41	DIGIT4, COMMON CATHODE

Push Button	FPGA Pinout	Descriptions
PB1	p44	Push Button No. 1
PB2	p46	Push Button No. 2
PB3	p47	Push Button No. 3
PB4	p50	Push Button No. 4
PB5	p51	Push Button No. 5

EEPROM	FPGA Pinout	Descriptions
I2C-SCL	p128	24LCXX
I2C-SDA	p129	24LCXX

BUZZER	FPGA Pinout	Descriptions
BUZ	p125	BUZZER

ตารางผนวกที่ ก1 (ต่อ)

LED	FPGA Pinout	Descriptions
L0	p70	L0
L1	p77	L1
L2	p69	L2
L3	p76	L3
L4	p74	L4
L5	p79	L5
L6	p73	L6
L7	p78	L7

Dip SW	FPGA Pinout	Descriptions
1	p52	Dip Switch No. 1
2	p53	Dip Switch No. 2
3	p55	Dip Switch No. 3
4	p56	Dip Switch No. 4
5	p59	Dip Switch No. 5
6	p60	Dip Switch No. 6
7	p63	Dip Switch No. 7
8	p68	Dip Switch No. 8

RS-232	FPGA Pinout	Descriptions
TX	p131	ICL3232CP
RX	p132	ICL3232CP

Oscillator	FPGA Pinout	Descriptions
OSC	p127	25MHz, GCLK6

ตารางผนวกที่ ก2 รายละเอียดขา I/O ของคอนเนคเตอร์ K2

K2 CONNECTOR						
Descriptions	FPGA Pinout	K2 Pinout		K2 Pinout	FPGA Pinout	Descriptions
GND	-	40		39	p129	I/O, I2C-SDA
GND	-	38		37	p131	I/O, RS232(TX)
GND	-	36		35	p135	I/O
GND	-	34		33	p140	I/O
GND	-	32		31	p1	I/O
GND	-	30		29	p4	I/O
GND	-	28		27	p6	I/O
GND	-	26		25	p8	I/O
GND	-	24		23	p11	I/O
GND	-	22		21	p13	I/O
GND	-	20		19	p15	I/O
GND	-	18		17	p18	I/O
GND	-	16		15	p21	I/O
GND	-	14		13	p24	I/O
GND	-	12		11	p26	I/O
GND	-	10		9	p28	I/O
GND	-	8		7	p31	I/O, DIGIT1
GND	-	6		5	p33	I/O, DIGIT2
GND	-	4		3	p36	I/O, DIGIT3
+3.3V	-	2		1	p41	I/O, DIGIT4

ภาคผนวก ข

ผลงานที่ได้รับการตีพิมพ์

State diagram design for implementing phase I of a Vector Symbol Decoder on an FPGA board

Rachanon Intharasakul and Usana Tuntoolavest*
Department of Electrical Engineering, Faculty of Engineering
Kasetsart University, 50 Phahonyothin rd., Chatuchak, Bangkok, 10900, Thailand
Phone +66-2-942-8555 ext1565 Fax +66-2-942-8555 ext1550
E-mail: rachanon_i@hotmail.com, fengunt@ku.ac.th*

***Abstract*— The principle of vector symbol decoding (VSD) has been shown as a powerful decoding technique. This paper focused on the implementation of the phase I of convolutional VSD decoder that can correct some error pattern using only one syndrome. Specifically, it is on how to design the state diagram for the interface of the test system and the state diagram for the decoding algorithm. The state design was for the VHDL programming and the implementation on an FPGA board. The test results showed that the phase I VSD decoder worked as designed.**

I. INTRODUCTION

Reliable data communication is important for most applications since the destination usually requires the correct copy of the transmitted data. For wireless channels, reliable data communication is challenging because the channels sometimes has low quality due to interferences and noise. To obtain low probability of decoding errors for this type of channel, powerful error-correcting codes can be the solution. There are several codes that have been implemented and are in use widely nowadays such as Reed-Solomon codes [1].

To use a code, there must be both the encoder and the decoder. Many codes have several ways to decode them, but usually a decoding technique was designed for a particular type of codes. This paper will focus on a decoding technique that is general in the sense that its principle can be applied for any linear block or convolutional codes, but the structure of the codes needs to be modified to use large nonbinary symbols. This technique is called vector symbol decoding (VSD). Its principle was first proposed by Metzner and Kapturowski in 1990 [2] and again by C. Haslach and A.J. Han Vinck in 1999 [3]. Some extended results can be found in the work by Tuntoolavest [4, 5] and Metzner [4, 6]. VSD is a powerful error-correcting decoding technique and suitable for some special applications such as for the channels with low quality where most errors occur in burst or as an outer decoder of a concatenated code.

To further develop and understand the advantages and limitations of this decoding technique, the authors are implementing the prototype of VSD for a specific convolutional code. This paper focuses on designing the state diagram of the phase I of the decoder, which will be the fundamentals for further development. The implementation of the phase I of VSD was done on an electronic board with FPGA (Field Programmable Gate Array) chip. The FPGA is selected because the algorithm can be simulated in the computer by using the VHDL (Very high speed integrated circuit Hardware Description Language) language first.

II. BACKGROUND

When alternative choices of the received sequence are available, VSD can simplify the decoding and make the decoding faster by adding additional steps that can correct some simple error patterns before passing the received sequence to the main decoding step [5].

Due to the complexity of the decoder, the implementation was divided into two parts as shown in Fig. 1. Phase I of the decoder called "Decode with one syndrome" involves this extra step of using the alternative choice. In this phase, the decoder checks whether there are any errors in the received sequence. It can also correct some error patterns by replacing the error symbols in the main received sequence by the correct symbols from the alternative choice given that no more than one error symbol occurs in each syndrome and the alternative choice has correct symbol at that positions. This means that the phase I of the decoder needs only one syndrome to correct each error symbol.

The next phase of the decoder called "Decode with many syndromes" is still under development. It will be for the error patterns that contain more error symbols in one syndrome or that the alternative choices are not correct at those positions. This paper focuses on designing the state diagram for implementing the decoder that can receive the received sequence as an input file from a personal computer, process the file and return the corrected sequence as an output file to the computer. The test result of the phase I decoder is shown as the test result of this system.

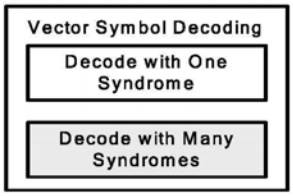


Fig. 1. The two phases of implementing Vector symbol decoder

Supported by Kasetsart University Research and Development Institute

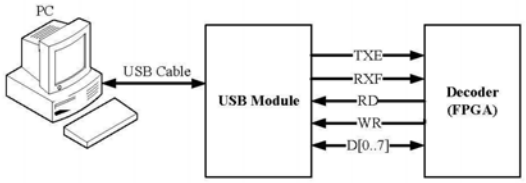


Fig. 2. Decoder system interface

III. METHOD AND STATE DESIGN RESULT

A. System Design and Interface

The test system was designed as a communication system for the receiver side after the demodulator part. The input file was transferred from the personal computer (PC), which is the source, to the FPGA board via a USB (Universal Serial Bus) module. There are five main signalling ports between the USB module and the VSD decoder on the FPGA board as shown in Fig. 2. The USB module was selected as the interface because the transmission rate is much faster than the serial port. The selected USB module called Ezy USB-M01 also has library files that help simplify the required interface program, which was written in C++ language. This interface program is needed for the USB module to talk with the PC and the FPGA board.

When the FPGA board receives enough information from the file, it will decode and correct the errors as necessary. Then it will transmit the corrected version back to the PC via the USB module. The output will be stored at the PC in a text file, which can be opened easily.

The architecture of the phase I VSD decoder was designed in behavioral style [7]. The main process operates in sequential style and consists of many states. The operation starts in the first state and then goes on to the next state according to the state diagram. In some states, there is an input signal that will control which next state the process will operate. There are several input signals as shown in Fig. 2. They will be discussed in detail in section B and C.

B. State Design for USB module

The first part of the state design was on interfacing between the USB module and the FPGA board. To design the state for the interface, the protocol of data transmission must be understood first.

The protocol of data transmission between the USB module and the FPGA was straight forward. First, the FPGA waits for the received flag RXF (Active low) from the USB module. When it receives the flag, the FPGA will send the signal RD to the USB module. This allows the data in the Rx (Receive) buffer of the USB module to appear on the data bus. Note that the USB module contains two buffers. The first one is the Rx buffer and the second one is the Tx (Transmit) buffer. The Rx buffer receives data from the PC and the Tx buffer sends data to PC. The buffers work in the FIFO (First In First Out) mode. The RXF signal will remain in logic "low" until all data in the Rx buffer have been transmitted. Then it will change to logic "high", which ends the data receiving process.

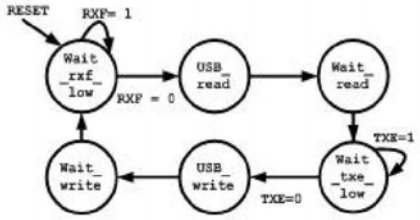


Fig. 3. State diagram for the data transmission between the USB module and the FPGA board

To transmit data from the FPGA board to the USB module, the FPGA checks whether the active low signal TXE is active. If it is not active (Logic "high"), the bus is busy and is not available to receive data from the FPGA because the transmitting data from the FPGA may collide with the data on the data bus. However, if the TXE signal is active (Logic "low"), the bus is available so the FPGA may send data to the transmit buffer in the USB module by first sending the signal WR to the USB module. This allows the data on the data bus to be written onto the Tx buffer of the USB module.

From the protocol, the state diagram for the interface between the USB module and the FPGA board was designed to have six states. Fig. 3. shows that the FPGA board is in the Wait_rxf_low state at the beginning of the operation or after the Reset button is pushed. When the signal RXF is active ("low"), it will go to state USB_read and send the Read signal RD to the USB module to allow the USB module to transmit data to the FPGA via the data bus D[0..7]. Then, it will go to the Wait_read state to read the data into the buffer of the FPGA board. After the FPGA received data, it will perform the decoding algorithm, whose state diagram will be considered in section 3.3, and then it will send the result back to the PC. Therefore, the next state will be for the event that the FPGA wants to send data back to the board. This means that the FPGA will be in the Wait_txe_low state. When the TXE signal is active, it will go to state USB_write and put data in the data bus to send to the USB module. In the Wait_write state, the USB module will put the data from the data bus into its buffer to be transmitted to the PC later. This completes a cycle for one set of data. The FPGA will go back to the Wait_rxf_low to wait for the next set of data.

C. Decoder phase I State Design and Design Result

In the first phase of the decoder, the decoder checks whether the received symbols are correct and attempts to correct error using only one syndrome at a time. The decoder needs to interface with the PC to receive each set of received symbols and to transmit the decoded symbols back to the PC. This is the first part of the state design that is similar to the one explained in section 3.2. The second part will be the state design for the decoding algorithm in phase I itself.

For the selected (3,2,2) convolutional code at the encoder and the assumption that the decoder is provided two alternative choices of the received symbols, the decoder in this phase requires two different sets of three symbols at a time. Moreover, VSD decoder uses very large symbols,

typically 32-bit symbols, which was the selected size for the implementation. The large symbol size is used to satisfy the linearly independent assumption of the error symbols because VSD principle is based on the idea that error patterns for large nonbinary symbols are almost never linearly dependent [6]. This makes the state design for the first part more complicated since the USB module was designed for the data transmission of 8 bits at a time. Therefore, the state design will include several counters to check that all data have been transmitted or received.

For the state design of the decoder, consider the decoding step of the VSD decoder phase I. The decoder will wait to receive all required data to decode with one syndrome. The three received symbols from the main choice will be used with the previous received symbols in the memory of the decoder to compute the current syndrome. If the syndrome is a zero vector, the decoder will decide that there is no error in this set of received symbols. In this case, these three received symbols will be transmitted to the output buffer to be sent back to the PC and the decoder will wait for the next set of the received symbols and repeat the procedure.

If the syndrome is not zero, the decoder will compute the differences between the received symbols in the first and in the second choices. If any one of the three differences is the same as the syndrome value, the symbol position corresponding to the difference contains error and the error value is the difference value of that position. When the error symbol position and the error value are both found, the decoder can make the correction.

Fig. 4. shows that state diagram of the decoder, which includes both the interface between the decoder and the PC and the decoding algorithm of phase I VSD decoder. The first two states are the same as in section 3.2. In the Wait_read state, the decoder will check whether it has received a complete 32-bit symbol by using the variable RX_cnt that counts each 8-bit subblock of the receive data. If RX_cnt is less than 4, the decoder will go to Wait_rxf_low state to wait for more data. When the RX_cnt is 4, which means that the decoder has received a complete symbol, the decoder will go to Assemble_rx state. In this state, the decoder will put the symbol that is assembled from the 4 subblocks in the input buffer.

After it receives each complete symbol, the decoder will go to State Chk_complete_rx, which will check whether it has received a complete block of 3 symbols by using BL_cnt variable to count the number of received data block (3-symbols per block). If BL_cnt is not equal to 3 the decoder will go back to Wait_rxf_low state and repeat the process until it received all three symbols of the first choice. Then, it will go to the Chk_ch_rx state to obtain all three received symbols of the second choice with similar procedure.

When symbols from both choices have been received, the decoder will then compute the current syndrome from the three received symbols of the first choice. However, it has to take the effect of previous symbols into account because the convolutional encoder has memory. Therefore, the decoder

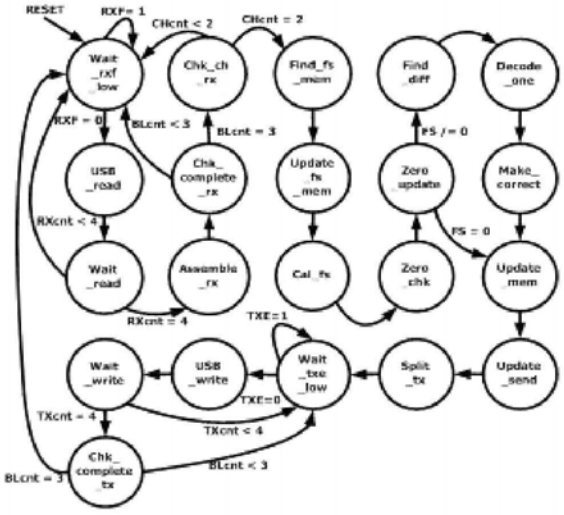


Fig. 4. State diagram of the decoder

will first go to Find_fs_mem state to calculate this effect and update this value in the Update_fs_mem state. Then the decoder will go to Cal_fs state and compute the current syndrome (32-bit), which is always called first syndrome (FS) since the decoder uses only one syndrome in phase I. Next, the decoder will go to Zero_chk state to compare the syndrome value to a 32-bit zero vector and the comparison result is shown by a flag called FS in the next state (Zero_update). If the flag FS is zero (Syndrome is $\bar{0}$), the decoder decides that there is no error and these received symbols are to be transmitted back to the PC. Before sending them back, the decoder needs to go to Update_mem state first to record the current received symbols effect that will be needed in the calculation of the syndrome for the next set of received symbols. But, if the flag FS is not zero (Syndrome is not $\bar{0}$), this means that the decoder detected error and needs to go to Find_diff state to calculate the difference values between the symbol values in the first choice and the second choice buffers.

When the decoder goes to Decode_one state, the decoder will find the error index by using the difference values. The error index will be used to correct the error symbol in the first choice buffer at Make_correct state. After it has been corrected, the decoder will go to Update_mem state to record the current received symbols effect that will be needed in the calculation of the syndrome for the next set of received symbols. After that, the decoder will go to Update_send state to move data from the first choice buffer to the Send buffer. After the Send buffer has received a complete set of data, the decoder will go to Split_tx state to separate each symbol to 4 subblocks of one byte each for the transmission back to PC.

The transmission back to PC is similar to what explained in section 3.2. The additions are on counters such as TX_cnt to count the number of subblocks that has been transmitted and BL_cnt to count the number of symbols that has been

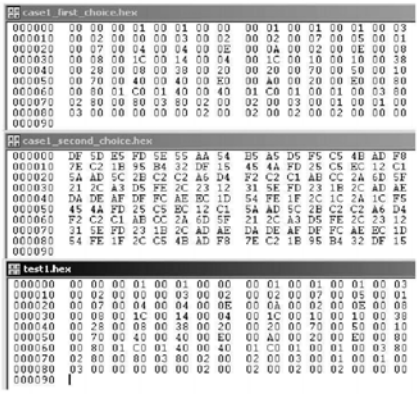


Fig. 5. The test result for the no errors case

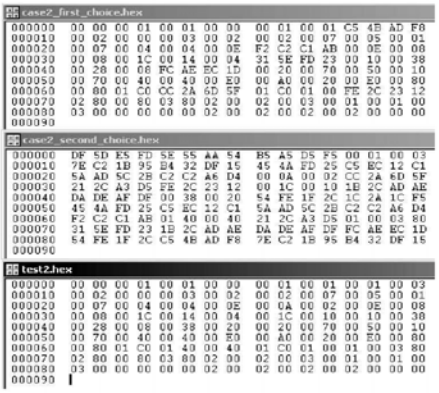


Fig. 6. The test result for the errorous case that the decoder can correct

transmitted. When the decoder has transmitted all decoded data in the Send buffer, it will go back to Wait_rxf_low state to wait for the next set of received symbols repeat the whole procedure.

IV. Test Results

The state design shown in Fig. 4. was for the implementation of the phase I VSD decoder on an FPGA board with VHDL language. The decoder implemented with this design worked successfully. To test the decoder, the decoding algorithm was written in VHDL and simulated in the computer first. Then the programs were burn into the FPGA board. The interface between USB module and the PC needs additional programming, which was written in C++ language. This allows the user to send files to the decoder and receive the decoded file back to display onscreen.

Two examples of the test results are shown. First, the file that contained first choice of the received symbol sequence had no error, so the decoded symbols should be the same as the first choice received symbols. The test was done by sending the file *case1_first_choice.hex* and the file *case1_second_choice.hex* for the first and the second choices respectively from the PC to the decoder. After the decoding was complete, the decoded data was returned to the PC and saved in the file *test1.hex* as shown in Fig. 5. It is seen that the file *case1_first_choice.hex* is the same as *test1.hex*. Therefore, the decoder worked correctly in this case.

Second, there are some error symbols in the first choice received symbols, but no more than one error symbols in each set of three received symbols and the second choice at those positions are correct. The test was done by adding errors in the file *case1_first_choice.hex* to get *case2_first_choice.hex* and put the correct symbol in the *case2_second_choice.hex* at those positions. Then, send the new two files to the decoder and received the file *test2.hex* back. Fig. 6. shows that *test2.hex* is the exactly same as *test1.hex* even though there are errors in the *case2_first_choice.hex*. This means that the decoder can correct the errors and transmitted back the correct decoded symbols.

V. Discussions and Conclusions

The state design was for the implementation with VHDL language and FPGA board. It is seen from Fig. 4. that the states are very detailed. For example, when the decoder calculates the syndrome value in a state and needs to compare the result to a zero vector, it must go to another state (the next clock) to do so. This very detailed state is a way to simplify the VHDL programming. This idea of state design is useful and can be extended for the next phases of VSD decoder.

The state design for the interface is very important to see if the decoder works properly. The test of the state design for the decoder with the interface was done on the actual hardware and the test results were sent back as a file to a computer, which made them easy to display. The results also showed that the decoder worked as designed.

The prototype of the VSD decoder was necessary for the understanding of the advantages and the limitations of this algorithm. There are many insights that can only be obtained from the physical hardware and not from the mathematical proof. This phase I of VSD decoder is a major start on implementing the complete prototype.

References

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

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

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

[4] 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, n 2, 2002, p 101-116, IOS Press, ISBN 1069-2509

[5] U. Tuntoolavest, "A simple method to improve the performance of convolutional vector symbol decoding with small symbol size." *IEEE TENCON 2004*, November, 21-24, 2004, Chiang Mai, Thailand.

[6] J.J. Metzner, "Vector symbol decoding with list inner symbol decisions," *IEEE Trans Comm.*, vol 51, No. 3, March 2003.

[7] L. Douglas Perry, VHDL: programming by example, McGraw-Hill, Boston, 2002.

Nonbinary Convolutional Encoder for Vector Symbol Decoding on FPGA board

Usana Tuntoolavest* and Rachanon Intharasakul
Department of Electrical Engineering, Faculty of Engineering,
Kasetsart University, Bangkok, Thailand
Phone:+66-2-942-8555 ext1565 Email: fengunt@ku.ac.th*, rachanon_i@hotmail.com

Abstract—Convolutional codes are widely used for reliable data communications. Conventionally, the codes use binary symbols because their decoders such as Viterbi decoder or Sequential Decoder usually were designed for binary symbols. In order to develop the system for nonbinary convolutional codes that use the vector symbol decoding algorithm, there is a need for a nonbinary convolutional encoder that works with large symbol size since the typical size of this decoding algorithm is 32 bits/symbol. For ease of implementation, this encoder was developed on an FPGA (Field Programmable Gate Array) board.

I. INTRODUCTION

Convolutional codes were first presented by Elias [1] in 1955. Several decoding algorithms have been proposed, but the most well-known is probably Viterbi algorithm [2]. Details on the codes and the decoding algorithms can be found in [3]. As with most codes, convolutional codes typically use binary symbols.

Vector symbol decoding (VSD) algorithm was first proposed by Metzner and Kapturowski in 1990 [4] and was rediscovered by C. Haslach and A.J. Han Vinck in 1999 [5]. It was further developed by Seo [6], Metzner [7-8], Tuntoolavest [8-10] and Interasakul [10]. This decoding technique works with large nonbinary symbols (usually 32 bits/symbol) and one application is to be an outer decoder for concatenated codes. The basic principle of VSD algorithm can be applied for both block codes and convolutional codes. To develop the system for large nonbinary symbol channel coding, it is necessary to consider the encoding part as well as the decoding part.

This paper focuses on the encoder of a nonbinary convolutional code that is suitable for vector symbol decoding algorithm. After the simulation with C++ and VHDL language, this encoder was then implemented on a FPGA (Field programmable Gate Array) board.

II. BACKGROUND

The concept of convolutional encoder can be readily extended for nonbinary symbols. To see this, consider

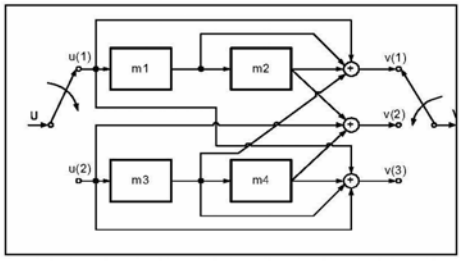


Figure 1. A (3,2,2) convolutional encoder.

a (3,2,2) convolutional code [3] with the transfer function

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

This block diagram of the encoder for this code is shown in Fig. 1. where $\oplus$ means Modulo-2 addition or the equivalent XOR operation.

For nonbinary symbols, the switch will alternate between up at down at every symbol instead of every bit. Consider 3-bit symbols case. The input sequence is the same, but the subsequences are changed as follows:

Let $\mathbf{U} = (\underline{u_1, u_2, u_3}, \underline{u_4, u_5, u_6} \ldots\ldots\ldots\ldots)$ (2)

Then, $\mathbf{u(1)} = (\underline{u_1, u_2, u_3}, \underline{u_7, u_8, u_9}, \ldots\ldots\ldots)$ (3)

And $\mathbf{u(2)} = (\underline{u_4, u_5, u_6}, \underline{u_{10}, u_{11}, u_{12}}, \ldots\ldots)$ (4)

Where u_i represents one data bit.

The Vector Symbol decoding algorithm typically uses 32 bits/symbol. Therefore the encoder in consideration would use 32 bits/symbol, which is a very large symbol size. The output sequence v(1),v(2) and v(3) can be obtained by performing the XOR operations as shown in Fig. 1.

In the implementation point of view, Fig. 1. shows that for this code, four memory blocks of 32 bits are needed with the exclusive-or operations. In addition to the circuit for the actual encoding process, there would be an interface module. This module would connect the encoder circuit in the FPGA

Supported by Kasetsart University Research and Development Institute

board to a PC (personal computer) so that files could be transmitted from the PC to be encoded by the encoder and the output code words could be viewed by the user via PC. For the interface, a USB (Universal Serial Bus) module would be used with some C++ programming part to let the user control the sending and receiving files process.

III. METHOD

The first step was to write the C++ program to perform the encoding operation. This would be used to verify that the hardware worked properly. The C++ code is very simple as it followed the XOR operations shown in Fig. 1. directly. Fig. 2. shows that main process in C++ without the header. All programs shown in this paper will focus on the main process or function only. The headers will be omitted to keep it concise.

In this program, the input sequence was separated into two subsequences. In the inner for-loop, each symbol from the two subsequences would be encoded by the XOR operations on the bit-by-bit basis. This could be done because the each nonbinary symbol in consideration was from GF(2^{32}) [3] and could be represented by a 32-bit vector [7]. After the XOR operations, the memories were updated by shifting the input in bit-by-bit. The outer for-loop set the number of input symbols and tails to put into the encoder.

After the C++ programming, the next step is to write the encoding operation using Hardware Description Language (HDL) called VHDL (Very High Speed Integrated circuit HDL) language [11]. The language can be written in both sequential format and concurrent format. The concurrent format makes it suitable to describe hardware. To use VHDL, we have to design in terms of Hardware and concern with the interface, which is more complicated than the C++ programming.

```
for (i=0;i < (((L/BPS)/2)+F);i++)
{
    for (int j=0;j < BPS;j++)
    {
        v1[i][j] = u1[i][j]^m1[j]^m2[j]^m3[j];
        v2[i][j] = u2[i][j]^m2[j]^m4[j];
        v3[i][j] = u1[i][j]^u2[i][j]^m3[j]^m4[j];
        m2[j] = m1[j];
        m1[j] = u1[i][j];
        m4[j] = m3[j];
        m3[j] = u2[i][j];
    }
}
```

L = the length of the sequence in bits.

BPS = number of bits per symbol.

F = number of zero symbols added at the tail of the encoding sequence to flush (clear) the memory of the encoder.

u1 and u2 = the input subsequence.

m1,m2,m3 and m4 = the memory positions as indicated in Fig. 1.

v1,v2 and v3 = the encoded subsequences

Figure 2. C++ code for the encoding operation

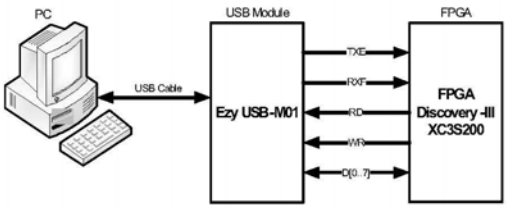


Figure 3. Interface for encoder test system

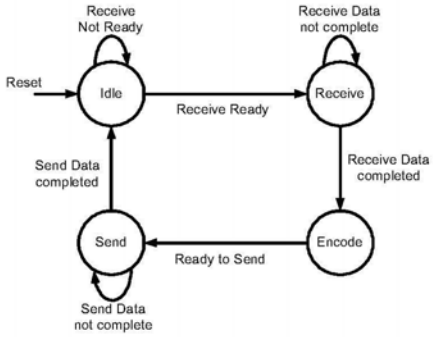


Figure 4. State diagram for the encoder

Fig. 3. shows that interface between three main units, which are a personal computer (PC), the USB module and the FPGA board.

More specifically, the designed state diagram for the encoder is shown in Fig. 4. At the beginning or after the reset button was pushed, the encoding operation is in Idle state. When the Receive Ready signal comes in, which means that USB module transmits data, the encoder will go from Idle state to Receive state. In this state, the encoder will receive data from USB module and put them in a buffer to be encoded later. When enough data was received, the state will change to Encode state. For the (3,2,2) convolutional encoder with 32-bit symbols in use, the state is changed each time 2 symbols comes in. When the encoding is done, the state will change to Send state and transmit the encoded sequence to the computer via the USB module.

The advantage of using VHDL to implement was that each function block could be simulated in the computer with the software from Xilinx. This greatly simplified the implementation process. The encode data process of VHDL program is shown in Fig. 5. The program in Fig. 5. worked at the positive edge of the clock signal. It would check whether the reset signal (RST) is 0 or 1. If it was 0, the program would clear the memory and the output would be zero. If the RST is 1, it would perform the encoding operation and update the memory.

```
--Encode data process
encode_pr : process(CLK)
begin
        if CLK'event and CLK = '1' then
          if RST = '0' then
                m1 <= (others=>'0');
                m2 <= (others=>'0');
                m3 <= (others=>'0');
                m4 <= (others=>'0');
                v1 <= (others=>'0');
                v2 <= (others=>'0');
                v3 <= (others=>'0');
          else
                v1 <= u1 xor m1 xor m2 xor m3;
                v2 <= u2 xor m2 xor m4;
                v3 <= u1 xor u2 xor m3 xor m4;
                m2 <= m1;
                m1 <= u1;
                m4 <= m3;
                m3 <= u2;
          end if;
        end if;
end process encode_pr;
-------------------------------------------------------------
Where CLK is the clock signal.
RST is the reset signal.
m, u and v are the same as in Fig. 1.
```

Figure 5. VHDL code for the convolutional encoder

The hardware selected for the encoder is the integrated circuit (IC) technology on FPGA chip. The model of the FPGA board used was FPGA Discovery-III XCS200 from APEX Instrument with the processor chip FPGA SPATAN-3 from Xilinx with 200,000 gates. It should be noted that the number of gates was much more than this application needed and the board was selected because the authors plan to extend the work to include other modules of the communication system as well. To interface with PC, a USB module in use is Ezy USB-M01 by Astron Logic Research & Development Co.,ltd. This module uses the IC FT245BM by FTDI (Future Technology Devices International Limited). Fig. 5. shows the hardware where the left board is the FPGA board and the right board is the USB module.

In addition to the function to perform encoding, there is also the interfacing part. The C++ program for transmitting and receiving files to and from the board is too lengthy to present here and it was modified from the standard send and receive files.

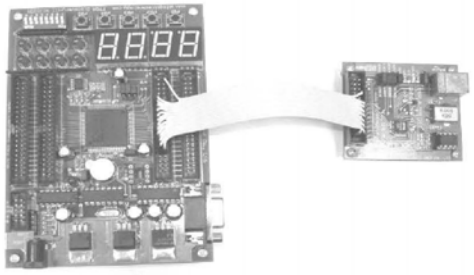


Figure 6. FPGA board and USB module

Figure 7. The interface for PC screen for send/receive file

Only the interface of the program on the PC screen is shown in Fig. 7.

After the VHDL program worked, it was burned onto the FPGA board. To test, a file that contained data was transmitted from the PC through the USB module to the encoding board and the file that contained the encoded sequence was received through the USB module back to the PC and saved in the name the user selected.

IV. RESULTS

The results are divided into two parts. The first part was the result from simulating with the VHDL program. The second part was the result from the hardware. An example of the simulation result with VHDL is shown in Fig. 8. The RST signal was in logic "low" at first. When it changed to high, the program obtained the information from u1 and u2, which were displayed in decimal format. A decimal number in each clock period represented the value of a 32-bit number. For example: "1" = 000…..0001, "7" = 000….0111. To see the numbers clearly, the example shown in Fig. 8. used very low values.

For the hardware part, the data in the input file was transformed into binary format first. This can be done by the binary converter program [12]. Then the input file in binary format named input_binary_file.txt was input from the interface on PC in Fig. 7. through USB module as shown in Fig. 3. to the FPGA board that performed the encoding operation. After the encoding was done, the encoded file was returned to the PC via USB module and the output file could be saved and viewed. The output file was saved in the name codeword.txt. Fig. 9. shows the input files in text and in binary format as well as the output file that shows the encoded sequence (codeword).

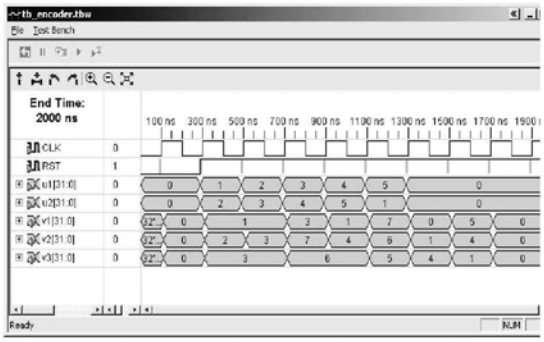


Figure 8. Simulation result from VHDL program

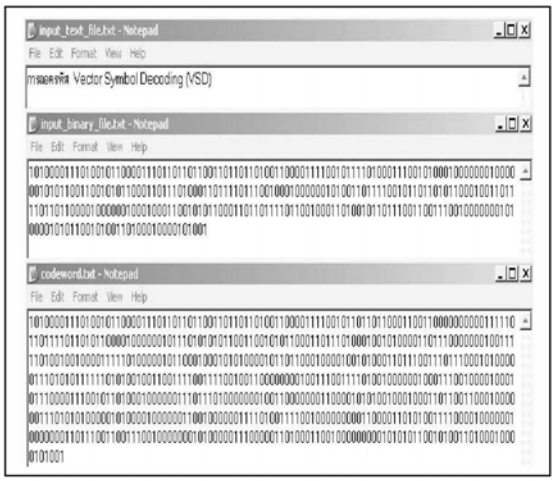


Figure 9. The input file in text and in binary format and the output file

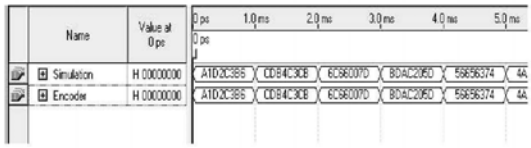


Figure 10. Compare the results from the simulation and the hardware. The numbers shown are in hexadecimal.

The results from the simulation and from the hardware were put on top of each other for clear comparison in Fig. 10. It was verified that the results from both parts are the same and it was also the same as the result from C++ program, which are not shown due to lack of space.

V. DISCUSSIONS

This paper showed a method to implement a (3,2,2) convolutional encoder for nonbinary symbols. This method can be applied for other nonbinary convolutional encoders with slight modifications such as the XOR operation may be performed on different set of inputs and memories based on the connections of the code in consideration. In fact, the basic principle is the same as the convolutional encoder for binary symbols since the nonbinary symbols selected was from $GF(2^n)$ and can be described as binary vectors. The differences of this binary and nonbinary encoder were in the implementation because each symbol was very large (for example 32-bit per symbol). The USB module in used can send or receive only 8-bit at a time, therefore each 32-bit symbol was divided into 4 subsymbols and transmitted in sequence. In addition, the input rate to the FPGA board and the output rate are not equal. With this selected (3,2,2) code, the rate is 2/3. This problem was solved by using state diagram as shown in Fig. 4. To verify the result from the hardware, the simulation was first done in C++, then in VHDL. It was found that these results were the same as the result from the hardware.

VI. CONCLUSIONS

The convolutional encoder for large nonbinary symbols is needed at the transmitter for a system that uses convolutional Vector Symbol Decoding (VSD) as the channel code decoding technique at the receiver. This paper shows a way to implement this encoder on FPGA board. By using Xilinx, it was simple to run the simulation on the software first and test it before put the code in the hardware. The hardware works properly as expected.

ACKNOWLEDGMENT

The authors would like to thank Mr. Audaphat Subnaung, a graduate student at Kasetsart University for the C++ interface part. Moreover, we would like to thank Kasetsart University Research Development Institute (Kurdi) for the research grant.

REFERENCES

[1] P. Elias, "Coding for noisy channels," IRE. Conv. Rec. Part 4, pp. 37-47, 1955.

[2] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory, vol. IT-13, pp. 260-269, Apr 1967.

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

[4] 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.

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

[6] Y.S. Seo, A new decoding technique for convolutional codes, Ph.D. Thesis, Pennsylvania State University, May 1991.

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

[8] 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.

[9] U. Tuntooolavest, "A simple method to improve the performance of convolutional vector symbol decoding with small symbol size," IEEE TENCON 2004, November, 21-24, 2004, Chiang Mai, Thailand.

[10] R. Interasakul and U. Tuntoolavest, "State diagram design for implementing phase I of a Vector Symbol Decoder on an FPGA board," submitted to ISCIT2006, October 18-20,2006, Bangkok, Thailand.

[11] L. Douglas Perry, VHDL: programming by example, McGraw-Hill, Boston, 2002.

[12] C. Wei, http://students.washington.edu/cwei/tools/ binary.shtml

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 ©

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

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

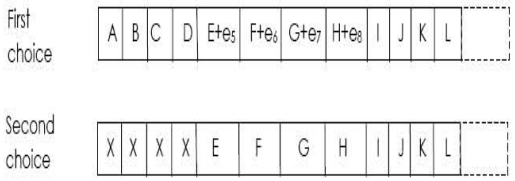


Fig 1. The first and second choice of the received sequence where e_i represent the error for the i^{th} received symbol
× represents a wrong received symbol

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,

$$\mathbf{S}_4 = \mathbf{H}_4\mathbf{Y} = \mathbf{H}_4\mathbf{E} \quad (3)$$

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.

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

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

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

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**.

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

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].

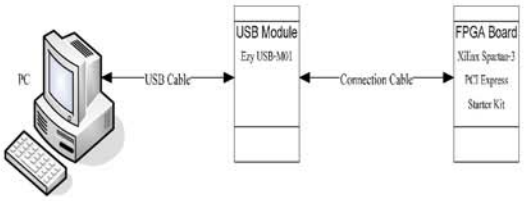


Fig. 2 System setup showing the interface between the decoder board and a computer

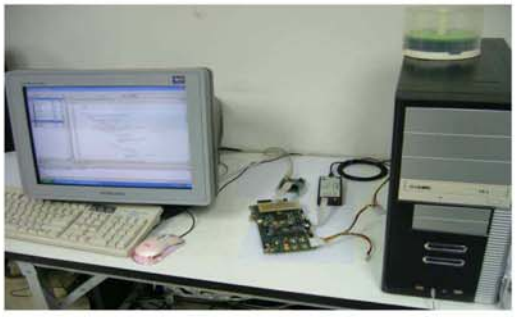


Fig 3. System setup in the lab

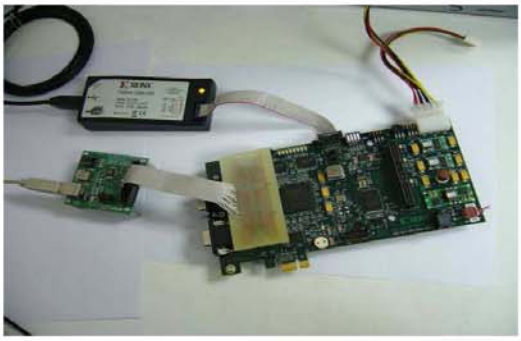


Fig. 4 The FPGA board with the USB module and the Platform cable USB.

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

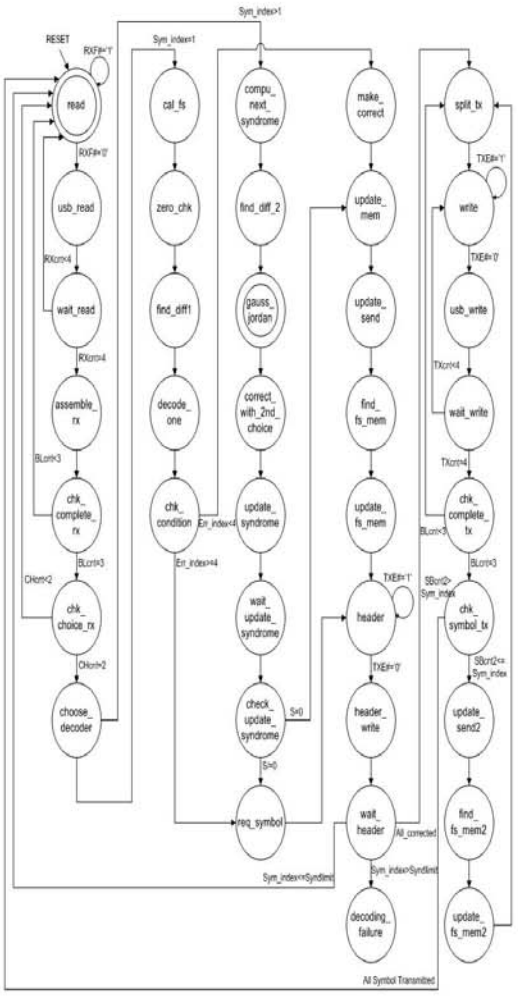


Fig. 5 State diagram design for Phase II conv. VSD decoder

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 diff2* 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. Conclusions

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	นายรัชนนท์ อินทรสกุล
วัน เดือน ปี ที่เกิด	วันที่ 7 กรกฎาคม 2524
สถานที่เกิด	อำเภอเมืองยะลา จังหวัดยะลา
ประวัติการศึกษา	วศ.บ. (วิศวกรรมไฟฟ้า) สถาบันเทคโนโลยีราชมงคล
ตำแหน่งหน้าที่การงานปัจจุบัน	วิศวกร 5
สถานที่ทำงานปัจจุบัน	บมจ. กสท โทรคมนาคม
ผลงานดีเด่นและรางวัลทางวิชาการ	
ทุนการศึกษาที่ได้รับ	