การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วงจรเข้ารหัสและถอดรหัสแบบการคูณประสาน)

VERY LARGE SCALE INTERGATED CIRCUIT DESIGN USING VHDL

(CONVOLUTION ENCODE AND DECODE CIRCUIT)

โดย

นาย นพดล ม่วงเฉย

นาย บุญช่วย รตะไพบูลย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็คทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วงจรเข้ารหัสและถอดรหัสแบบการคูณประสาน)

VERY LARGE SCALE INTERGATED CIRCUIT DESIGN USING VHDL

(CONVOLUTION ENCODE AND DECODE CIRCUIT)

โดย

นาย นพคล ม่วงเฉย รหัส 36014203

นาย บุญช่วย รตะไพบูลย์ รหัส 36014233

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็คทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ ปีการศึกษา ๒๕๓๕

ภาควิชา อิเล็คทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล
(วงจรเข้ารหัสและถอดรหัสแบบการคูณประสาน)

ผู้จัดทำ

1. นาย นพดล ม่วงเฉย รหัส 36014203
2. นาย บุญช่วย รตะไพบูลย์ รหัส 36014233

อาจารย์ที่ปรึกษา
(ดร. สมศักดิ์ ชุมช่วย)

การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วงจรเข้ารหัสและถอดรหัสแบบการคูณประสาน)

VERY LARGE SCALE INTERGATED CIRCUIT DESIGN USING VHDL

(CONVOLUTIONAL ENCODE AND DECODE CIRCUIT)

1. นาย นพดล ม่วงเฉย รหัส 36014203

2. นาย บุญช่วย รตะไพบูลย์ รหัส 36014233

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ ชุมช่วย)

การออกแบบวงจรเข้ารหัสและถอดรหัสโดยใช้ภาษา วีเอชดีแอล

นาย นพดล ม่วงเฉย
นาย บุญช่วย รตะไพบูลย์
อาจารย์ สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา

ปีการศึกษา 2539

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้เป็นการศึกษาและออกแบบวงจรเข้ารหัสและถอดรหัสในการรับส่งข้อมูลทางดิจิตอล โดยใช้หลักการของ คอลโวลูชั่นโคด แบบ ซีสทีเมติกโคด เนื่องจาก ซีสทีเมติกโคด จะมีคุณสมบัติที่เป็น นอนแคสทาโธปิกโคด ซึ่งโคด ชนิดนี้เมื่อผ่านการเข้ารหัสแล้วจะไม่มีปัญญาในการหาเมตริกซ์ผกผันของเจนเนอร์เรเตอร์เมตริกซ์ ซึ่งจะทำให้ได้ข้อมูลที่ถูกต้อง ยกเว้นกรณีที่เกิดความผิดพลาดในตัวข้อมูลอันเนื่องมาจากสัญญาณรบกวนอื่นๆ โดยการเข้ารหัสจะแบ่งข้อมูลที่เข้ามาให้เป็นส่วนๆ โดยแต่ละส่วนจะมีขนาดko บิทเรียกว่า อินฟอร์เมชั่นเฟรม และเมื่อผ่านการเข้ารหัสแล้วจะได้ โคดเวิร์ดเฟรม มีขนาด noบิท โดยหลักการที่สำคัญของการเข้ารหัสจะใช้ เจนเนอร์เรเตอร์โพลิโนเมียล เป็นส่วนหลัก ซึ่งสามารถเขียนในรูปเมตริกซ์ G(X) =| I| P(X) | และในส่วนของวงจรถอดรหัสจะสามารถถอดรหัสโดยอาศัย อินเวิร์ดเมตริกซ์ G(X) เป็นตัวถอดรหัส โดยในส่วนของวงจรถอดรหัสจะสามารถตรวจจับและแก้ไขความผิดพลาดที่เกิดขึ้นในการรับส่งข้อมูล โดยอาศัยเมตริกซ์ H ในการหาซินโดรม ที่เกิดขึ้น ซึ่งจะสัมพันธ์กับ ความผิดพลาดที่เกิดกับข้อมูล

ในทางปฏิบัติเราสามารถใช้อุปกรณ์ทางดิจิตอลต่างๆ เช่น ชิพท์รีจีสเตอร์ เอ็กซ์คลูซีฟออร์ และเกทต่างๆ นำมาต่อร่วมกันเพื่อให้เกิดคุณสมบัติของวงจรเข้ารหัสและถอดรหัส ตามที่ได้ออกแบบไว้ โดยได้มีการใช้ ซอร์พแวร์ มาช่วยในการออกแบบวงจรในระดับ เกท เพื่อให้ได้คุณสมบัติของวงจรตามต้องการ ซึ่งในโครงงานฉบับนี้ได้นำภาษา วีเอชดีแอล (VHSIC Hardware Description Language) มาใช้ในการออกแบบวงจรเข้ารหัสและถอดรหัส ซึ่งภาษา วีเอชดีแอล มีความสามารถแทนการทำงานของวงจรด้วยอุปกรณ์ต่างๆและสามารถ ชิมมูเลต การทำงานของวงจรที่ออกแบบได้ จึงมีส่วนช่วยในการออกแบบวงจรดิจิตอลเป็นอย่างมาก

ซึ่งในปริญญานิพนธ์ฉบับนี้ได้ทำการออกแบบ คอลโวลูชั่นนอล โคด แบบ ซีสทีเมติก โคด โดยจะมีขนาด (12,6) โดยในการเข้ารหัสจะมีอินพุท 1 บิท และเอาท์พุท 2 บิท และในภาคถอดรหัสจะมีการตรวจสอบข้อผิดพลาด เมื่อข้อมูลเกิดข้อผิดพลาดขึ้น โดยเราจะใช้ภาษา วีเอชดีแอล มาช่วยในการออกแบบและทดสอบการทำงานของวงจร

ENCODE DECODE CIRCUIT DESIGN USING VHDL

MR NOPPADOL MUANGCHEOY

MR BUNCHUAY RATTAPAIBOON

Dr. SOMSAK CHUMCHUAY ADVISOR

ACADEMIC YEAR 1996

ABSTRACT

This project concerns the design of Encoder and Decoder circuit for transferring digital information . Convolutional codes method is used because the systematic code had the qualification of noncastastrophic . When this code has already passed through the Encode process, it will be no problem about inverse matrix of generator in the Decoding process. These lead to the correct information encode in case of it is mistaken owing to the factors. The information will be devided into a small size in the Encode process that Ko bit is one of each part size and we call it the information frame then it passed the Encode process to be codeword frame which is no bit size . The Encode Generator polynomial is mean by used as G(X) = [I| P(X) | , on the Decode side circuit can be done by using Inverse matrix G(X). Decode circuit can check and solve the occurrence of information mistake by the parity matrix H to find syndrome which is related with the error pattern of information.

In practice , we can use the other digital accessory such as shift register , exclusive OR and other gates , combined together in order to get the qualification of Encoder and Decoder circuit . VHDL (VHSIC Hardware Description Language) is used in this project to design Encode and Decode. With VHDL , the design task can be simpler and design time can saved.

In this thesis, the hardward of Systematic Convolutional code is implemented . Size of the code is (12,6) . VHDL language is used in designing and testing of the circuit . The encoder consists of 1 bit input and 2 bit output.

สารบัญ

สารบัญรูป

รูป หน้าที่

บทที่ 1
บทนำ

ปริญญานิพนธ์ฉบับนี้นำเสนอการออกแบบวงจรเข้ารหัสและถอดรหัสด้วยภาษาวีเอชดีแอล โดยใช้คอมพิวเตอร์ช่วยในการออกแบบ โดยหลักการออกแบบวงจรเข้ารหัสและถอดรหัสที่ได้ศึกษาเป็นรหัสแบบ คอนโวลูชั่น โคด (Convolution Code) ชนิดซีสทีเมติก โคด (Systematic Code) ซึ่งเป็นรหัสที่นิยมใช้กันทั่วไป โดยส่วนข้อมูลเข้าจะเรียกว่าอินฟอร์เมชั่นเฟรม (Information Frame) และเมื่อผ่านการเข้ารหัสแล้วจะได้สัญญาณออกที่เรียกว่า โคดเวิร์ดเฟรม (Codeword Frame) โดยส่วนสำคัญที่ใช้ในการเข้ารหัส คือ เจนเนอร์เรเตอร์โพลิโนเมียล (Generator Polymomial) และเราจะใช้อินเวอร์เมตริกซ์ของเจนเนอร์เรเตอร์เมตริกซ์ในการถอดรหัสที่ได้จากวงจรเข้ารหัส

ในการออกแบบวงจรเข้ารหัสและถอดรหัสที่ใช้ภาษาวีเอชดีแอลเป็นการออกแบบในแนวใหม่ ซึ่งสะดวกรวดเร็วและมีประสิทธิ์ภาพ ภาษาวีเอชดีแอลเป็นภาษาบรรยายการทำงานของฮาดแวร์เพื่อใช้อธิบายการทำงานของวงจรดิจิตอลตัวภาษานั้นประกอบไปด้วยรายละเอียดและส่วนประกอบต่างๆ ซึ่งใช้อธิบายพฤติกรรม(Behavioral) หรือโครงสร้าง (Structural) ของระบบ โดยพิจารณาถึงฐานเวลา (Timing) ของระบบเป็นหลักเนื่องจากว่าระบบที่เราจะออกแบบโดยวีเอชดีแอลนั้น ผู้ออกแบบไม่จำเป็นต้องรู้ถึงวงจรภายในว่ามีอะไรต่อกันอย่างไรบ้าง เพียงแต่กำหนดอินพุท เอาท์พุท และฟังก์ชั่นการทำงานของระบบเขียนเป็นโปรแกรมแสดงการไหลของข้อมูล (Dataflow) การทำงานของระบบหรือโครงสร้างของระบบขึ้นมา จากนั้นก็ทำการคอมไพล์และซิมูเลทโมเดลที่ออกแบบมาเพื่อทำการวิเคราะห์ดูฟังชั่นฐานเวลาของระบบว่าตรงตามต้องการหรือไม่ ถ้ามีการแก้ไขอย่างไรก็ทำการแก้ไขซอสโคด (Source code) ใหม่แล้วทำการคอมไพล์และซิมูเลทซ้ำๆ จนกว่าจะได้โมเดลที่มีฐานเวลาของระบบตามต้องการ ซึ่งมีความง่ายและสะดวกรวดเร็วกว่าเมื่อก่อนมากเพราะไม่ต้องเสียเวลากับการสร้างวงจรทางฮาดแวร์จริงๆ ขึ้นมาทดสอบ ซึ่งทำให้เสียเวลาและค่าใช้จ่ายสูง โมเดลที่ออกแบบโดยใช้วีเอชดีแอลสามารถทำการสังเคราะห์ (Synthesis) เพื่อให้ได้ซีเมติกไดอะแกรม (Schematic Diagram) และเอาไปสร้างเป็นวงจรจริงๆได้ โดยอาจนำเอาไปสร้างเป็นเอเอสไอซี (Application Specific Integratd circuit) หรือนำไปเบิร์น (Burn) ลงอีพีแอสดี หรือเอฟพีจีเอ เพื่อนำไปใช้งานจริงต่อไป ประโยชน์ของภาษาวีเอชดีแอลใช้กันมากในการออกแบบชิปเอเอสไอซี หรือออกแบบไมโครโปรเซสเซอร์ โดยมีซอฟแวร์หลายตัวสนับสนุนตั้งแต่การคอมไพล์, ซิมูเลท, การสังเคราะห์ เช่น เมนเตอร์กราฟฟิก (Mentor Graphic) , วิวลอจิก (Viewlogic) ประกอบกับคอมพิวเตอร์ระดับเวอร์กสเตชันที่มีระบบกราฟิกที่ดีและการประมวลผลด้วยความเร็วสูงในปัจจุบัน ทำให้การออกแบบทุกขั้นตอนเป็นไปอย่างรวดเร็วและมีประสิทธิ์ภาพ ซอฟแวร์ซีเออี (CAE) ที่ใช้ในการพัฒนาโครงงาน

นี้ขึ้นมาคือ วิวลอจิก (View Logic) ตั้งแต่การเขียนซอสโคด การคอมไพล์วีเอชดีแอลซอสโคด การทดสอบฟังก์ชันการทำงาน การจำลองการทำงานตรวจสอบความถูกต้องของวงจร เมื่อได้ซอสโคดที่สมบูรณ์ของโมเดลแล้วสามารถนำไปทำการแปลงซอสโคดให้อยู่ในรูปชีเมติกไดอะแกรม (Schematic diagram) โครงงานนี้ได้ทดลองทำการออกแบบวงจรเข้ารหัสถอดรหัสข้อมูล โดยใช้อัลกอริทึมดีอีเอส (Data Encryption Standard' s Algorithm) ซึ่งเป็นอัลกอริทึมการเข้ารหัสและถอดรหัสที่ใช้กันเป็นมาตราฐานสากลและมีความปลอดภัยสูง โดยผลที่ออกมานั้น สามารถนำมาใช้งานได้จริง ในการออกแบบวงจรเข้ารหัสและถอดรหัสโดยใช้ภาษาวีเอชดีแอลนั้น ผู้จัดทำของยอมรับว่ายังมีข้อผิดพลาดหลายประการในรายงานนี้ ซึ่งผู้จัดทำยินดีรับคำติชมและจะปรับปรุงแก้ไขให้ดีขึ้นต่อไป หวังว่ารายงานฉบับนี้จะเป็นประโยชน์ต่อท่านผู้ที่สนใจเพื่อเป็นแนวทางการศึกษาการเข้ารหัสและถอดรหัสโดยใช้ภาษาวีเอสดีแอลต่อไป

ผู้จัดทำ

บทที่ 2

หลักการทั่วไปของวงจรเข้ารหัสและถอดรหัส

ในปัจจุบันระบบการสื่อสารจำเป็นต้องมีการออกแบบการส่งข้อมูล ด้วยอัตราความเร็วสูง โดยในบางครั้งมีความจำเป็นที่ต้องใช้ถึงขนาดเป็นล้านบิตต่อวินาที รวมทั้งยังต้องการระบบการตรวจสอบข้อมูล เพื่อกันความผิดพลาดของข้อมูล และระบบ จึงได้มีการนำ block code เข้ามาใช้ โดยมี K เป็นสัญลักษณ์แทนข้อมูลที่เข้ามาใน blocks และหลังจากผ่านการเข้ารหัสก็จะได้ nซึ่งแทนสัญลักษณ์ของ code word

โดยทั่วไปหลักการเข้ารหัสจะอาศัยการแบ่งข้อมูลให้มีขนาดเล็กลง เป็นขนาด Ko ซึ่งเราจะเรียกว่า information frames ซึ่งจะถูกนำไปเข้ารหัสให้เป็น code frame ที่มีความยาว h_o โดยที่ information frame 1 ตัว จะให้ code word frame ออกมา 1 ตัว เช่นกัน จากกระบวนการทั้งหมดก็จะรวมกันเรียกว่า encoding procedure และ code ที่ได้รับเราจะเรียกว่า tree code ซึ่ง tree code ที่เราจะกล่าวถึง คือ convolutional code ซึ่งเป็น tree code ที่มีคุณสมบัติ ของ tinearlity,Time invariance ซึ่งหัวข้อแรกที่จะกล่าวถึง คือ หลักการของ tree code โดยทั่วไปแต่จุดประสงค์หลักของเราคือ Convolutional code ซึ่งเป็นกรณีพิเศษของ tree code

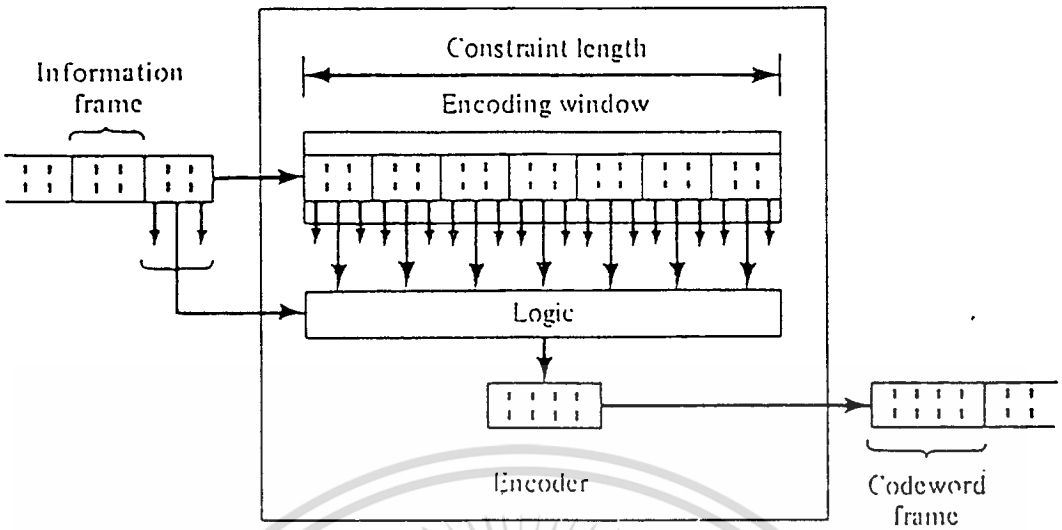
2.1 แผนภาพ tree code และกราฟ trellis code

จากที่กล่าวมาข้างต้น ส่วนประกอบหลักของ tree code คือ shifted-register ดังรูปที่ 1 โดยข้อมูลจะเริ่มต้น shifted เข้ามาที่เวลา O และจะ shifted ต่อไปเรื่อยๆ และลำดับข้อมูล จะถูกแยกเป็นส่วนๆ และแทนด้วย Ko โดยจะเรียก Ko ว่า information frame และตัววงจรรหัสจะสามารถเก็บข้อมูลได้ m frame และในแต่ละช่วงเวลาข้อมูลใหม่จะถูก shifted เข้ามาใน shifted-register และข้อมูลตัวเก่าจะถูก shifted ออกไป

จากข้อมูลที่เข้ามาและ m frame ที่ถูกเก็บไว้ วงจรเข้ารหัสจะนำมาคำนวณ หลังจากการคำนวณจะได้ code word 1 frame ที่แทนด้วยสัญลักษณะ n_o และ code word frame นี้จะถูก shift ออกจากวงจรเข้ารหัสเมื่อข้อมูล frame ใหม่เข้ามา และเมื่อเสร็จ กระบวนการต่างๆ ก็จะได้ code word n_o สำหรับแต่ละข้อมูล k_o

code word ที่ได้มาซึ่งเป็นผลมาจากลำดับของ input ซึ่งเราเรียก (n_o,k_o) tree code โดยใช้สัญลักษณ์ $R = k_o / n_o$แทน Rate ของ tree code

กำหนดให้ $v = mk_o$ซึ่งvจะถูกใช้ในการบรรยาย convolutional code ซึ่งเราเรียกว่า Constraint length ซึ่งเราจะกล่าวถึงอีกครั้ง จากวงจร Encodet รูปที่ 1 เราจะได้ $k_o = 3$, $k_o = 5$, $v = 21$



รูปที่ 2.1 ตัวอย่าง shift-register ของวงจรเข้ารหัส

ต่อไปนี้จะเป็นส่วนที่จำเป็นต้องกล่าวถึงใน tree code คือ k และ n โดย

$k = (n+1)\, k_o$ เรียกว่า wordlength

$n = (m+1)\, n_o$ เรียกว่า blocklength

และจากรูปที่ 1 เราจะได้ blocklength คือ 40 ซึ่ง block length ตัวนี้ก็คือ code word ซึ่งเป็นผลมาจาก single information frame แต่ใน tree code มักนิยมที่จะกล่าวถึงเฉพาะ (n_o,k_o)

ใน tree code จะประกอบไปด้วยคุณสมบัติที่สำคัญ 4 ข้อดังต่อไปนี้

1. Finite Constraint length ค่า Constraint lengthอาจจะมีค่าจำกัดหรือไม่จำกัดก็ได้ แต่ในทางปฏิบัติ tree code มักจะมี Constraint length เป็นค่าจำกัดแต่ถ้ากล่าวถึงในทางทฤษฎีค่า Constraint length มักจะมีค่าไม่จำกัด อย่างไรก็ตาม (n_o,k_o) tree code ที่มี Constraint length เป็นค่าจำกัด v, wordlength k 1 และ block length n อาจเขียนในรูป (n,k) ซึ่งเรียกว่า trellis code

2. Time invariace คือถ้ามี input 2 input ที่แตกต่างกันเข้ามาในช่วงเวลา frame เดียวกันก็จะได้ code word ออกมาในช่วงเวลาเดียวกัน

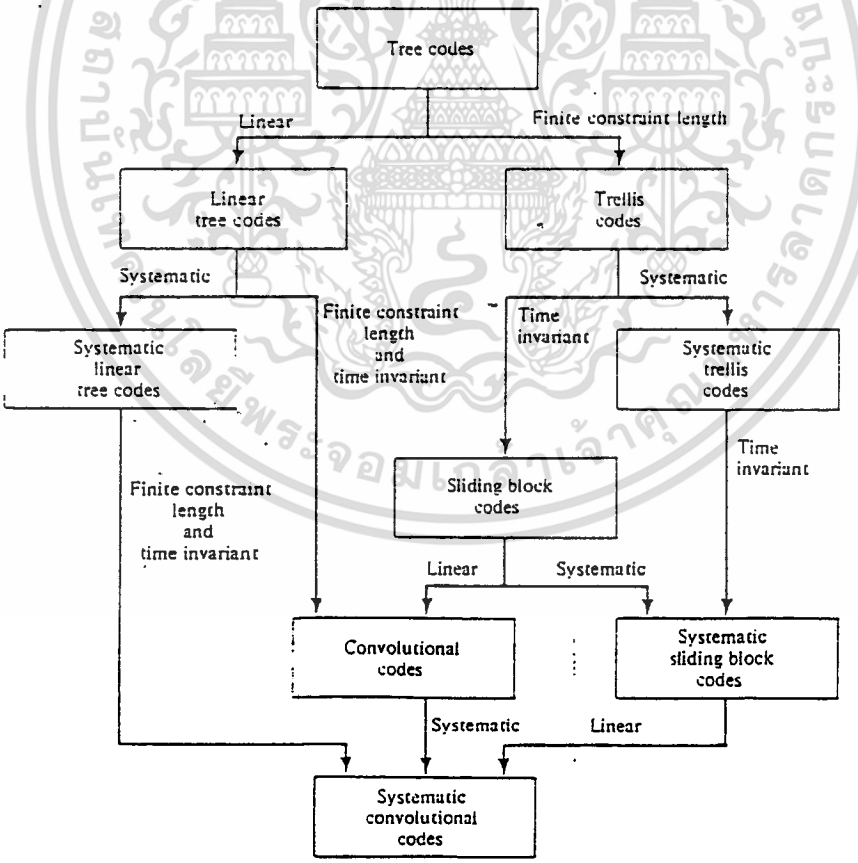
3. linearity คือ ถ้ามี input 2 ตัว ที่มีคุณสมบัติการรวมเป็นเชิงเส้นก็จะได้ code word ที่มีคุณสมบัติเป็นเชิงเส้นตาม input เช่นถ้า d_1,d_2 เป็น input และมี code word เป็น $G(d_1),G(d_2)$ แต่ถ้า input เป็น $ad_1 + ad_2$ ก็จะได้ code wordเป็น $G(ad_1 + ad_2) = aG(d_1) + bG(d_2)$

4. Systematic คือ การที่แต่ละ bit ของ information frame จะปรากฏที่ k_o bit แรกของ code word frame

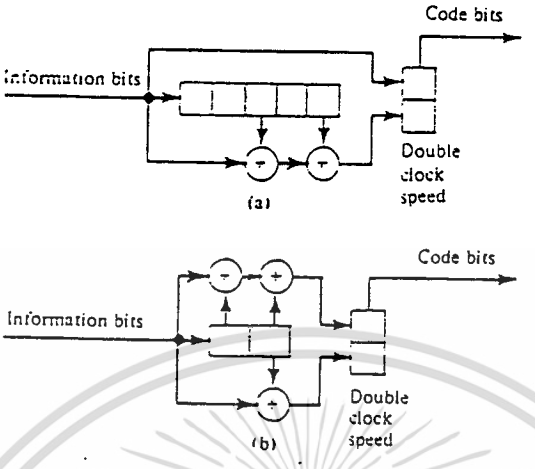
นิยาม ถ้ามี (n_o,k_o) tree code ที่มีคุณสมบัติ linear, time invariant และมี wordlength ที่มีความยาวจำกัด $k = (m + 1)\,k_o$ซึ่งเราจะเรียกว่า (n,k) Convolution code และถ้า (n,k)Convolution code มีคุณสมบัติ Systematic Convolution code

นิยาม ถ้า $(h_{o1}k_o)$ tree code มีคุณสมบัติtime invariant และ Finite wordlength k เราจะเรียก (n,k) sliding block code และมีคุณสมบัติ linear ก็จะเรียกว่า linear Sliding block

จากรูปที่ 2 จะแสดงถึงความสัมพันธ์กันระหว่างแบบต่าง ๆ ของ tree code ที่เป็นไปได้และรูปที่ 3 จะแสดงถึง ตัวอย่างของ encoder 2 แบบที่มีความแตกต่างกันของ Convolution code แต่ในทั้งสองแบบจะมีค่า $n_o = 2$, $k_o = 1$ เหมือนกัน จากรูป 3a จะเป็น encoder แบบ Systermatic Convolution code ขนาด (12,6) ที่มีค่า constraint length เท่ากับ 5 รูปที่ 3b เป็น encoder แบบ honsystermutic convolution code ขนาด (6,3) และมี Constraint length เท่ากับ 2 และในทั้งสองกรณีเมื่อมี input เข้ามาผ่านการเข้ารหัสก็จะได้ output ออกมาในแต่ละ clock time โดย output จะเก็บไว้ใน buffer ขนาด 2 bit

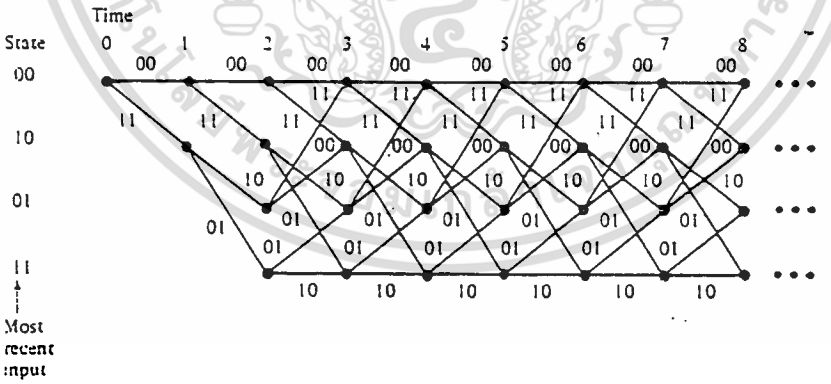


รูปที่ 2.2 แผนภาพแสดงคุณสมบัติของ tree code



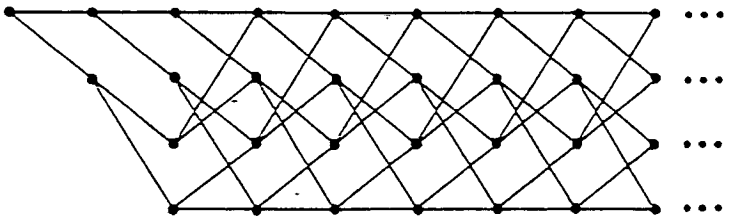
รูปที่3 ตัวอย่าง convolution encoders a) encoder สำหรับ binary ขนาด (12,6)
b) encoder สำหรับ binary ขนาด (6,3)

Convolution code และ trellis code สามารถบรรยายการทำงานในลักษณะกราฟได้ ซึ่งกราฟนี้จะเรียกว่า trellis ซึ่งกราฟจะมีลักษณะเป็น code บนตาราง พื้นผ้า และด้านขวาของกราฟจะเป็นลักษณะ Semi-infinte และ จำนวนของ code ในแต่ละ colume จะมีจำนวนจำกัด ซึ่งลักษณะของกราฟจะแตกเป็นสาขาเชื่อมต่อระหว่าง code ของ colume กับ code ในอีก Colume หนึ่ง



รูปที่ 2.4 กราฟ trellis ใช้แสดงการทำงานของ convolutional code

จากรูปที่ 4 จะเป็นตัวอย่างของ gtuph trellise สำหรับ binary code และรูปที่ 5 จะแสดง trellise ของตัวอย่างในรูปที่ 3b ซึ่งเป็นตารางที่ใช้ในการ encoder และ node ในแต่ละ colume ของ trellise จะแทน q state ซึ่งสมมุติให้เป็น shiff-register และจุดย่อยใน colume จะแทน set ของสถานะแต่ละเวลา



รูปที่2.5แผนภาพ trellis สำหรับ (6,3) convolutional code

ฉะนั้นในแต่ละ trellis จะแทน Convolution code ได้โดยทางผ่านต่าง ๆ จากซ้ายไปขวา ซึ่งใช้แทนค่า code word

2.2 บรรยาย Polynomial ของ convolutional code

(m+1) ho, $(m+1)k_o$ convolutional code และ constraint length $v = mk_o$ จะสามารถ encode ได้โดย n_o set ของ Finiter-impulse-response (FIR) Filters และในแต่ละ set จะประกอบด้วย k.FIR litters โดยจะมีอัตราการไหลของ input เป็น K_o bit/time และจะมี output ส่งออกในอัตรา no bit time จากรูปที่ 6 จะแสดง encoder สำหรับ binary Convolution code ซึ่งมีขนาด $n_o = 5$, $k_o = 1$ ซึ่ง encoder จะประกอบด้วยแถวขอ Litter รวมทั้ง output timing buffer สำหรับ binary convolutional code ขนาด $n_o = 5$, $k_o = 3$ โดยที่ input buffer จะต้องเข้ากับ input rate และ Filter rate

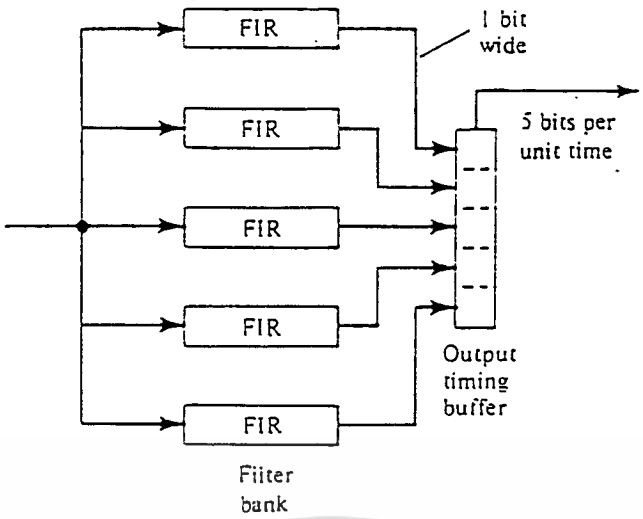
FIR แต่ละตัวจะใช้แทน Polynomial ที่มี degree มากที่สุดไม่เกิน m และถ้า input สามารถเขียนในรูปผลคูณของ Polynomial ได้ซึ่งในที่นี้ encoder ของ Convolutional code สามารถแทนได้ด้วย Polynomial ฉะนั้น code word ของมันสามารถแทนได้ด้วย Polynomial ได้เช่นเดียวกัน และ Polynomial ของ filter จะถูกเรียกว่า generator polynomial ซึ่ง generator polynomial จะมี degree มากที่สุดเท่ากับ m

ในทางตรงข้าม ถ้าเป็น block code จะมี generator polynomial เพียงตัวเดียวแต่ถ้าเป็น Convolutional code จะต้องการ generator polynomial หลายตัว ซึ่งเมื่อรวมกันแล้วจะมีขนาดเท่า k_o n_o Polynomials

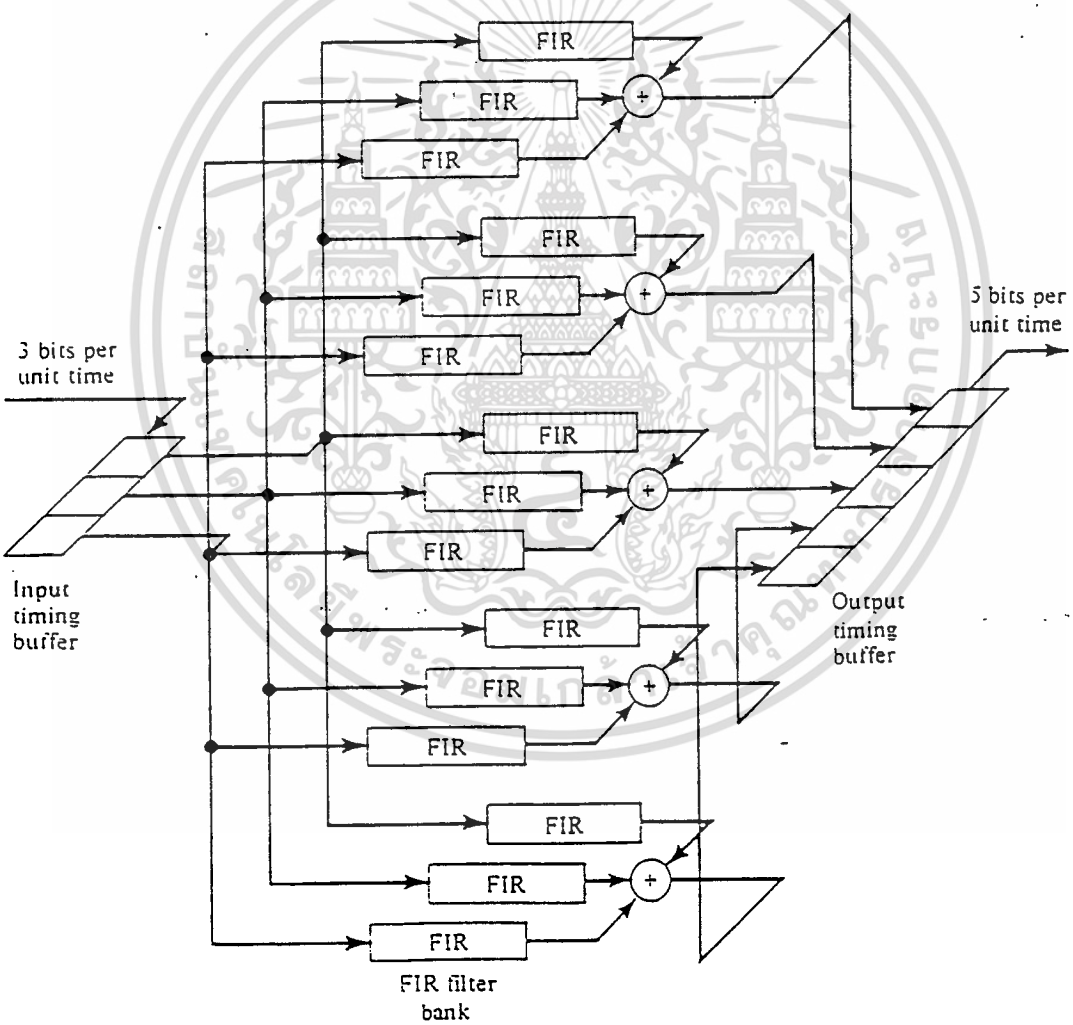
กำหนดให้ $g_{ij}(x)$ โดยที่ $i = 1,2...k_o$ และ $i = 1,2...n_o$ เป็น set ของ generator Polynomial เราจะสามารถนำมาเขียนในรูปของ Matix ที่มีขนาด k_o x n_o

$G(x) = [g_{ij}(x)]$

โดยที่ k_o มีค่ามากกว่า 1 เสมอ



รูปที่ 2.6 วงจรเข้ารหัสแบบ convolutional code



รูปที่ 2.7 วงจรเข้ารหัสแบบ convolutional code ขนาด ($n_o = 5, k_o = 3$)

ตัวอย่าง จากรูปที่ 3 เราสามารถหา Matrix ของ generator polynomial ได้คือ

$G(x) = [1\ x^5 + x^3 + 1]$ และ

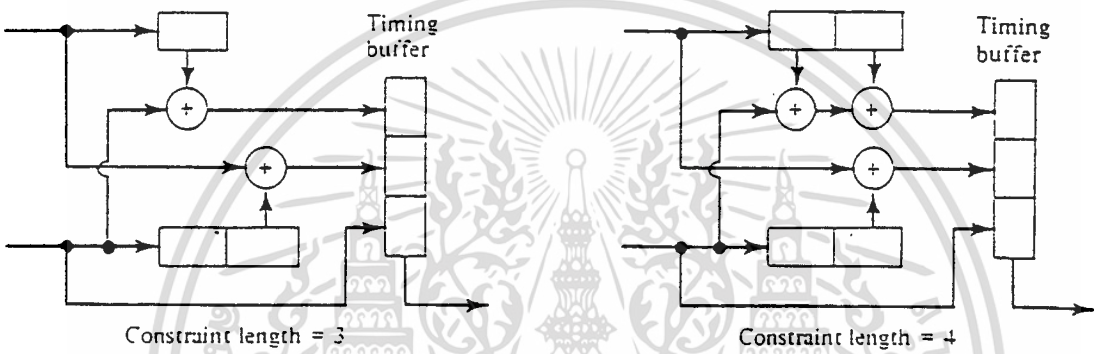
$G(x) = [x^2 + x + 1\ x^2 + 1]$

นิยาม ถ้ามี matrix ของ generator polynomial $[g_{ij}(x)]$ เราสามารถหา Constraint length ของ Convolutional code ได้เป็น

$v = \sum_{i=1}^{k_o} \max [\deg g_{ij}(X)]$ และ

word length $k = k_o \max [\deg g_{ij}(x) + 1]$

block length $n = n_o \max [\deg g_{ij}(x) + 1]$



รูปที่ 2.8 ตัวอย่างวงจรเข้ารหัสด้วยอัตรา 2/3

ตัวอย่าง จากรูปที่ 8 Convolutional code ซึ่งสามารถหาค่า v,k และ n ดังนี้

รูป 8a จะมีค่า v = 3, k = 6 และ n = 9

8b จะมีค่า v = 4, k = 6 และ n = 9

ถ้าเราพิจารณา input frame k_o เข้ามาในลักษณะ ขนาน และ

พิจารณาลำดับของinput frame k_o ที่เข้ามาขนานกันซึ่งเราสามารถแทนโดย k_o informution polynomial di(X) โดยที่ $i = 1,2...k_o$ หรือ แสดงในลักษณะ ของ voctor Polynomials

$d(X) = [d_1(X), d_2(X),.....dk_o(X)]$

ในลักษณะเดียวกัน output code word สามารถแทนได้ด้วย n_o code word polynomial Cg(X) โดยที่ $j = 1,2...h_o$ หรือแทนในลักษณะ Vector ดังนี้

$C(X) = [Cj(X)] = [C_1(X), C_2(X),.....Cn_o(X)]$

และสามารถเขียน C(X) เมื่อเสร็จการ encoding ได้ดังนี้

$C(X) = d(X)\ G(X)$ หรือ

$Cj(X) = \quad di(X)\, g_{ij}(X)$

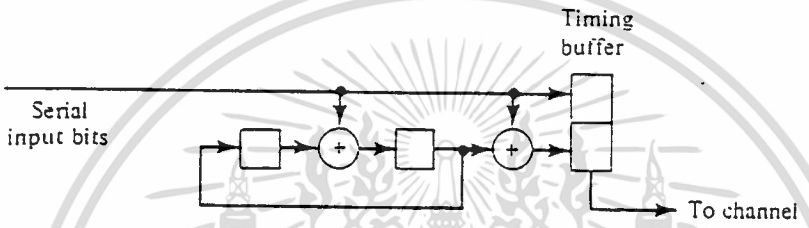
A parity - check - polynomial matrix H(X) เป็นเมตริกซ์ที่มีขนาด $(n_o - k_o) \times n_o$ ซึ่งเราสามารถหารูป H(X)ที่เหมาะสมคือ

$G(X)\, H(X)^T = 0$

Syndrome - polynomial vector ซึ่งเราสามารถเขียนได้ดังนี้

$S(X) = V(X)\, H(X)^T$ โดยที่ s(X) จะมีแถวเวกเตอร์ เป็น $(n_o - k_o)$ แถว

สำหรับในกรณีของ Systematic encoder เราสามารถเขียน generator - polynomial ในรูปเมตริกซ์ได้เป็น $G(X) = [Ip(x)]$



รูปที่ 2.9 วงจร convolutional code แบบ systematic และมีการป้อนกลับขนาด (6,3)

เมื่อ I มีขนาด $k_o - k_o$ เป็นเมตริกซ์เอกลักษณ์

p(X) มีขนาด $k_o \times (n_o - k_o)$ เป็นเมตริกซ์ของ Polynomial

และ parity - check - polynomial ของ Syatematic convolutional code เราสามารถเขียนได้เช่นเดียวกัน คือ $It(X) = [-p(X)^T : I]$

เมื่อเมตริกซ์ I มีขนาด $(n_o - k_o) \times (n_o - k_o)$

สำหรับ H(X) ของ systematic convolutional code สามารถตรวจเช็คได้เช่นเดียวกันคือ $G(X)\, H(X)^T = o$

เรามักจะให้ความสำคัญกัล systematic encoder มากกว่า systematic convolutional code เพราะเหตุที่ systematic cncoder มีความเหมาะสมมากกว่า จากการที่เราสามารถมองเห็น และอ่าน information ได้โดยตรงถ้าหาก ไม่มี error ใดๆ เกิดขึ้น ซึ่งมน block code ไม่สามารถทำได้ อย่างไรก็ตาม Convolutional code ทุกตัว ก็ไม่จำเป็นต้องเป็น Systematic Convolutional code และในบางครั้ง nonsystematic Convolutional code ก็อาจมีคุณสมบัติตรงที่สามารถประหยัดโครงสร้างได้ดีกว่า systematic Convolutional code และจากเหตุผลที่ code word ใดๆ ที่ไม่ผ่านการเข้ารหัสโดยวิธี Systematic ก็จะทำให้ code word ไม่สามารถอ่าน formation นั้นได้โดยตรงฉะนั้นเราจะต้องออกแบบ

ให้การเข้ารหัสนั้นจะต้องทำให้ได้มา formation กลับมาอย่างถูกต้อง โดยปราศจาก error ซึ่งสามารถทำได้โดยให้ encoder เหล่านั้นเป็น feed forward โดยการให้ polynomial มีการ teedback กลับเข้ามาในวงจรดังตัวอย่างในรูปที่ 9 ซึ่งจะกลายเป็น Systematic teedback encoder เช่นเดียวกับในรูปที่ 5

ต่อไปนี้เราจะชี้ให้เห็นความสำคัญในกรณีที่ $k_o = 1$ ซึ่งเราจะเห็นว่าเมื่อ $k_o = 1$ เราสามารถเขียน

$G(X) = [g_1(X_1)g_2(X)...g_{no}(X)]$

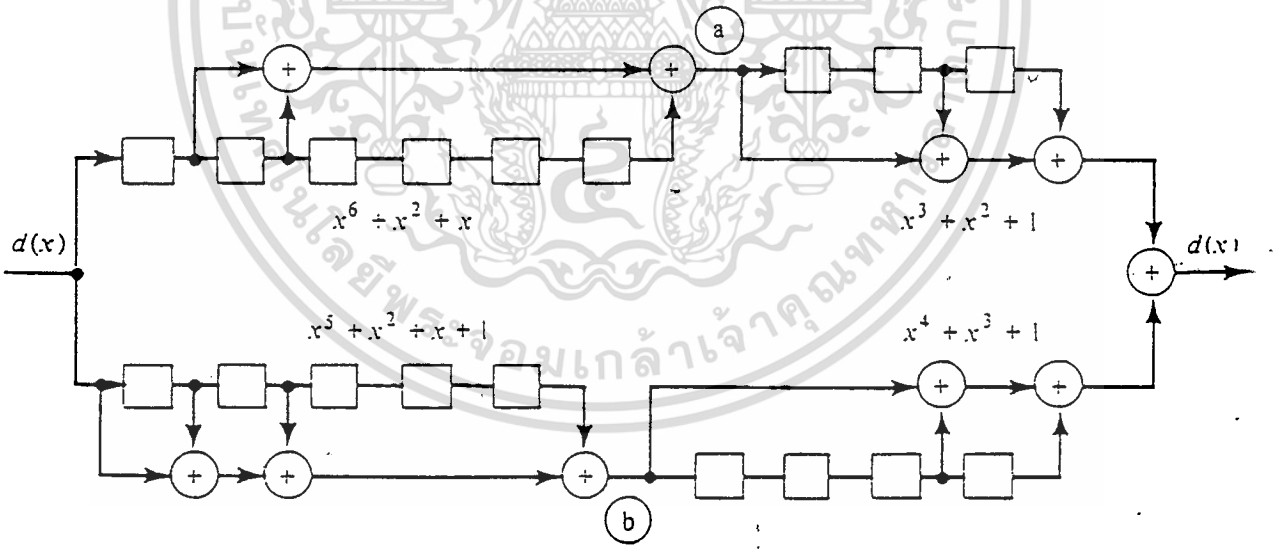
และ $C_j(X) = d(X)g_j(X)$ เมื่อ $j = 1,2...k_o$

นิยาม Convolutional code ซึ่ง generator polynomial $g_1(X),g_2(X)...g_{no}(X)$ ที่เหมาะสมจะเขียนในรูป GCD $[g_1(X),g_2(X)...g_{no}(X)] = x^9$ ได้ [GCD คือตัวหารร่วมมาก] โดยที่ a บางคำเท่านั้น จะถูกเรียกว่า noncastastrophic Convolutional นอกนั้นจะถูกเรียกว่า Castastrophic Convolutional code

ถ้าหาก $x^y + 1$ เราจะเรียกว่าใกล้เกิดการสูญเสียขึ้น เพราะถ้า $x^y \# 1$ มันจะเกิดการ delay ขึ้นใน filter

สำหรับ noncastastrophic Convolutional code สามารถถอดรหัสโดยปราศจาก error ได้โดยวิธี Euclidean algoithm ซึ่งจะมี polynomial เป็น $a_1(X)...a_{no}(X)$ กับ $a_1(X)g_1(X)+...+ a_{no}(X) = 1$

ดังนั้นถ้าเรามี d(X) เป็น data polynomial เราสามารถ encoded ได้โดย $C_1(X) = d(X)g_j(X)$ เมื่อ $j = 1,2...n_o$ และเราสามารถ decoded กลับมาได้โดย $d(X) = a_1(X)c_1(X)+...+ a_{no}(X)\ c_{no}(X)$



รูปที่ 2.10 ตัวอย่างวงจรเข้ารหัสและถอดรหัส

ตัวอย่าง จากรูปที่ 10 data polynomial จะเข้ามาภายในวงจรจากทางซ้าย และข้อมูลจะออกไปทางขยายวงจร โดยที่จุด a และ b จะเป็นจุดที่มีอัตราการรวม bit ซึ่งในตัวอย่างนี้จะมีอัตราการรวม bit เท่ากับ 1/2 และในบางครั้งจุดนี้ จะถูกใช้ในการหาและตรวจสอบ error ต่างๆ ซึ่งจะทำให้ code มีคุณภาพดีขึ้น

ถ้าจะพูดถึง Convolutional code ที่ดีเราก็มักจะพูดถึงความสัมพันธ์ของ generrator polynomial ที่ดีก่อน ซึ่งจะยากมากในการสร้างความสัมพันธ์ดังกล่าว โดยเฉพาะวิธีการที่เราจะทำอย่างไร ให้มันสามารถให้วงจรตรวจสอบ error ที่ดีได้

โดยทั่วไปk_o มักจะมีค่ามากกว่า 1 ซึ่งใน noncatastrophic code จะถูกเขียนอยู่ในรูป (n_o) ซึ่งแตกต่างจากรูปเมตริกซ์ย่อย k_o x k_o ของ G(X) และจะถูกชี้แสดงโดยเมตริกซ์เอกลักษณ์ I ถ้ากำหนดให้ $_I$(X) เป็น determinant ของ I (k_o x k_0)

นิยาม Convolutional code ที่มี Generator - polynomial matrix G(X) ที่มี determinant

$_I$(X) โดยที่ I = 1...( ) โดยที่จะมีรูปที่เหมาะสมคือ

GCD [$_I$(X) I = 1...( ) = x^a

โดยแท้จริงแล้ว noncastastrophic code สามารถเขียน matrix of polynomial (n_o x k_o) $G^*(X)$ ได้โดย $G^*(X)$ G(X) = x^aI

เมื่อ I มีขนาด k_o x k_o เป็นเมตริกซ์เอกลักษณ์ และ x^a จะใช้แทนค่า delay ที่ถูก fixed เอาไว้ซึ่งจะยากมากในการหา $G^*(X)$ และในทางปฏิบัติก็ไม่นิยมทำกัน แต่ถ้าเราใช้วิธี systematic code ก็จะได้ตามนิยามข้างบนและจะมีคุณสมบัติเป็น noncastastrophic เสมอด้วย

2.3 การตรวจสอบ error และการหา distance

ทุกๆ ครั้งเมื่อ Convolutional code ผ่านเข้าทาง chanhel ต่างๆ ก็มักจะทำให้เกิด error ขึ้นในขณะนั้นเสมอ ดังนั้นวงจร decode ซึ่งควรที่จะตรวจสอบ error เหล่านั้นได้ โดยทำจาก Codeword ที่ได้มา แต่วงจรdecoded ก็จะสามารถจดจำส่วนที่ error ได้เพียงบางส่วนในช่วงเวลาหนึ่งๆ เท่านั้นถึงแม้ว่า Codeword จะมีความยาวมากก็ตาม และการตัดสินใจของ decode ก็จะทำกับส่วนของ code word ที่มีขนาดจำกัด เท่านั้น แต่จริงๆ แล้วโครงสร้างของ codeไม่มีวิธีการที่จะแยกออกเป็นส่วนๆ ได้ ดังนั้นเราจะใช้ประโยชน์ของ information ที่ decoded ไม่จำเป็นต้องใช้แทน

โดยทั่วไปในการศึกษาเกี่ยวกับการตรวจสอบ error ของ decoded นั้นมักนิยมที่จะกล่าวถึงเฉพาะใน first frame เท่านั้น ซึ่งถ้า frame แรกสามารถแก้ไข error และสามารถ decode ได้ซึ่งจะทำให้เราได้ information ของ first frame และจากการที่ได้ information บางส่วนจาก codeword frame ซึ่งสามารถคำนวณและดึงออกจาก Code word หลักได้นั้น ก็จะทำให้ Second code word frame ถูก

คำนวณ และหาออกมาได้เหมือนใน first frame ซึ่งถ้าเราคิดต่อไปว่าถ้า first j frame ประสบผลสำเร็จในการแก้ไข error ขณะเดียวกันการ decode ของ (j+1) frame ก็จะเป็นปัญหาเช่นเดียวกันดังใน first frame ซึ่งก็มีหลายวิธีด้วยกันที่ decode จะสามารถทำได้สำหรับวิธีที่ใช้ information เข้าใน error ในบาง frame นั้นจะถูกเรียกว่า feedback precedure

วิธีการ decode อื่นๆ ก็จะคำนวณและจัดรูปแบบเพื่อที่ให้มีคุณสมบัติที่ง่ายขึ้น ส่วนมากแล้วโอกาส first frame ของ code word จะไม่สามารถแก้ไข error ได้ก็เป็นเพราะ มี error เกิดขึ้นมากเกินไป ซึ่งในบางครั้ง decode error ใน infinite number ของ error ใน code word ซึ่งหลักวิธีนี้จะเรียกว่า error propagation

การที่มี error propagation ประจำตัว ในวิธีการของ decoding algorithm ซึ่งในกรณีนี้จะเรียกว่า ordinary error propagation ซึ่งจะเกิดขึ้นประจำใน Castastrophic generator potyhomial และจะเรียกว่า castastrophic error propagation ซึ่งคุณสมบัตินี้ในการออกแบบอาจที่จะหลีกเลี่ยงได้ ถ้าเป็นไปได้

สำหรับวิธีที่ decoder สามารถ store ได้นั้นจะเรียกว่า decoding window width โดยทั่วไป ถ้าเราพยายามตรวจสอบ error ที่เป็นไปได้ ในขณะเดียวกันก็สามารถทำได้ดีขึ้นโดย การเพิ่ม decoding - window width แต่เหตุการณ์ที่มาถึงจุดนี้จะถูกทำให้กลับไปมีขนาดลดลง ฉะนั้น decoding - window width จึงควรที่จะมีขนาดใหญ่เท่ากับ blocklength n และบ่อยๆ ที่ต้องใช้เวลามากๆ

Convolution code มีวิธีที่สั้นๆ มากมาย เมื่อพิจารณาความยาวเริ่มต้นของ code word ที่มีระยะสั้นๆ ซึ่งตัวระยะทางนี้จะถูกกำหนดว่า ถ้าทั้งสอง code word ถูก decode อยู่ใน first information เดียวกัน ขณะนั้นเองตัวมันจะถูกพิจารณาเหมือนกัน

นิยาม I^{th} minimum destance d_j^* ของ Convolutional code มีขนาดเล็กที่สุดข้อควรระวังคือ ระยะทางระหว่าง 2 code word เดิมในส่วน j frame ซึ่งไม่ถูกยอมรับใน frame เดิมถ้า I เท่ากับ m + 1 ซึ่งในขณะนี้จะเรียกว่า minimum destonce และแทนด้วย d^* และลำดับ d_1^*, d_2^*, d_3^* จะถูกเรียกว่า distance profile ของ Convolutional code

ถ้าเราสมมติ Convolutional code มี I^{th} minimum destance d_I^* และจะมี error ที่เหมาะสม t จะเขียนได้ว่า

$$2t + 1 < d_I^*$$

ซึ่งสามารถเกิดขึ้นได้ใน first j frame ขณะเดียวกัน first frame จะสามารถแก้ไข error ได้ ในทางปฏิบัติจะให้ I = m + 1 ; the minimum destance d_I^* ของ Code word เท่ากับ d^*_{m+t} ซึ่งค่า f ที่เหมาะสมคือ

$2t + 1 \leq d^*$

และ code จะสามารถถูกแก้ไข error ใน first frame ได้ถ้าเกิด error ขึ้นมากที่สุด t ตัว ใน first block tength ซึ่งจะเรียกว่า t-error-correcting convolutional code

นิยาม free distance ของ Convolutional code L ถูกเขียนเป็น d_{∞}

$= m\alpha_1 x d_I$ ซึ่งผลก็คือ $d_m+1<d_m+2<...d_{\infty}$

นิยาม free length n ของ Convolotionsl code เป็นความยาวของส่วน

ที่ไม่เป็นศูนย์ของส่วนที่มีน้ำหนักน้อยที่สุดของ Convolutional code word ของน้ำหนักที่ไม่เป็นศูนย์ ดังนั้น $d_I=d_{\infty}$ ถ้า $I=n_{\infty}$ และ $d_I<d_{\infty}$

ถ้า $I<n_{\infty}$

ในรูปที่ 5 จะมี free distance เท่ากับ 5 และ free length เท่ากับ 6mในตัวอย่างนี้จะมี free length เท่ากับ block length ของ Convolutional code แต่โดยทั่วไปควรจะมีค่ามากกว่า block longth

2.4 การเขียนเมตริกซ์ของ convolutional code

Convolutional code มักจะอยู่ในรูป intinite code word จะมีความเป็นเชิงเส้นและสามารถบรรยายได้โดย intinite generator matrixและ generator matrix ก็สามารถบรรยายลักษณะของ code ได้เป็นเดียวกัน แต่จะไม่มีความสะดวกในการใช้ ยกเว้นแต่ generator matrix จะมีลักษณะที่ดีเท่านั้น generator matrix ที่ใช้กับ Convolutional code จะมีความซับซ้อนมากกว่า generator matrix ที่ใช้กับ Block code

genetator polynomial ที่มีตัวชี้คือ i และ j สามารถเขียนเป็น

$g_{ij}(x) = \sum g_{ij}Ix^l$

และเพื่อให้ได้ genetator matrix สัมประสิทธิ์ g_{ijI} จะถูกอยู่ใน

รูป matrix สำหรับ I จะได้ G_I เป็น $k_o x n_o$ matrix

$G_I g_{ijI}$

ขณะนี้จะได้ generator matrix ที่ใช้สำหรับ Convolutional code ที่เป็นส่วนตัดของ block code และมี blocklength nคือ

$$G^{(m)} = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m \\ 0 & G_0 & G_1 & \cdots & G_{m-1} \\ 0 & 0 & G_0 & \cdots & G_{m-2} \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & & G_0 \end{bmatrix},$$

เมื่อ o คือ $k_o x k_o$ matrix ของศูนย์ ซึ่ง genetator matrix สำหรับ Convolutional code คือ

$$G=\begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & 0 & 0 & 0 & 0 & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & 0 & 0 & 0 & \cdots \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \cdots \\ \vdots & & & & & & & & & \end{bmatrix}.$$

เมื่อ matrix ที่ได้นี้จะเป็นไปในลักษณะที่ไม่แน่นอนในแนวลงและทางขวายกเว้นแต่ในแนวเส้นทะแยงมุมของเมการิกซ์ ย่อย m ที่ไม่เป็นศูนย์ในขณะที่ส่วนอื่นๆ จะเป็นศูนย์

สำหรับในกรณี Systematric convolutional code ซึ่งจะสามารถเขียนได้ 2 รูปแบบคือ

$$G^{(n)}=\left[\begin{array}{cc|cc|cc|c|cc} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m \\ 0 & 0 & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} \\ 0 & 0 & 0 & 0 & I & P_0 & \cdots & 0 & P_{m-2} \\ \vdots & & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & I & P_0 \end{array}\right].$$

and

$$G=\left[\begin{array}{cc|cc|cc|c|cc|cc|c} I & P_0 & 0 & P_1 & 0 & P_2 & \cdots & 0 & P_m & 0 & 0 & \cdots \\ 0 & 0 & I & P_0 & 0 & P_1 & \cdots & 0 & P_{m-1} & 0 & P_m & \cdots \\ 0 & 0 & 0 & 0 & I & P_0 & \cdots & 0 & P_{m-2} & 0 & P_{m-1} & \cdots \\ \vdots & & & & & & & \vdots & & 0 & P_{m-2} & \cdots \\ & & & & & & & & & \vdots & & \end{array}\right]$$

เมื่อมีลักษณะ รูปแบบซ้ำๆ เลื่อนไปทางขวาทุกๆ แถว และจะไม่มีลักษณะชี้เฉพาะของเมตริกซ์ และ I ในที่นี้จะเป็นเมตริกซ์เอกลักษณ์ $K_o x K_{o\ 1}$ o จะเป็นเมตริกซ์ศูนย์ $K_o x_{Ko}$ และ $P_o ... P_m$ จะเป็นเมตริกซ์ $K_o x(h_o - K_o)$ โดยที่แถวแรกจะใช้บรรยายการ encoding ของ first intormation frame ใน first m codeword frame และที่สำคัญเราควรจะสามารถแสดงเมตริกซ์ นี้ให้อยู่ในเทอมของ shift-register ได้

first information frame ที่ถูกใช้ encode ภายใน first code word frame จะเป็นส่วนของซ้ายบนสุดของ G คือ

$G^{(no)}=[IPo]$

และ first two intormation frame ที่ถูก encode ภายใน first frame code word frame คือ

$$G^{(2no)}=\begin{bmatrix} I & P_0 & 0 & P_1 \\ 0 & 0 & I & P_0 \end{bmatrix}.$$

Aparity check matrix ในเมตริกซ์ H ที่เหมาะสมคือ

$$G^{(ln_o)}\, H^{(ln_o)T} = 0 \qquad l = 0,1,2,.....$$

เมื่อ $g^{(ln_o)}$ และ $H^{(ln_o)}$ เป็นช้ายบนสุด ของเมการิกซ์ย่อย G และ H เช่นเดียวกันกับใน 1 trame A purity check matrix H ซึ่งเป็น เมตริกซ์ในลักษณะทะแยงมุม และมีความยาวเป็น infinite สามารถสร้างได้จาก เมตริกซ์ G ดังนี้

$$H = \begin{bmatrix} P_0^T & -I & & & & & & & & \cdots \\ P_1^T & 0 & P_0^T & -I & & & & & & \\ P_2^T & 0 & P_1^T & 0 & P_0^T & -I & & & & \\ \vdots & & & & & & & & & \\ P_m^T & 0 & P_{m-1}^T & 0 & P_{m-2}^T & 0 & \cdots & P_0^T & -I & \cdots \\ & & P_m^T & 0 & P_{m-1}^T & 0 & \cdots & & & \\ & & & & P_m^T & 0 & \cdots & & & \end{bmatrix},$$

เมื่อในส่วนของช่องว่างที่ไม่ระบุคือ เมตริกซ์ศูนย์

ตัวอย่าง Systematic (4,2) binary convolutional code เมื่อ $K_o = 1$ และ m =1 จะบรรยายโดยเมการิกซ์ 1x1 และ $P_o = 1$ ดังนั้น

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & & & \\ & 1 & 1 & 0 & 1 & & \\ & & 1 & 1 & 0 & 1 & \\ & & & \cdots & & & \cdots \end{bmatrix} \quad \text{and} \quad H = \begin{bmatrix} 1 & 1 & & & \\ 1 & 0 & 1 & 1 & \\ & 1 & 0 & 1 & 1 \\ & & \cdots & & \cdots \end{bmatrix}.$$

ตัวอย่าง Systematic (12,4) binary convolutional code เมื่อ $K_o = 1$ และ m = 3 จะให้ค่า Po = [1 1], P_1 =[0 1], P_2 = [1 0] และ P_3 = [1 1] จะได้

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & & & & & & & & & & & & \\ & & & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & & & & & & & & & \\ & & & & & & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & & & & & & \\ & & & & & & & & & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & & & \\ & & & & & & & & & & & & \vdots & & & & & & & & & & & \end{bmatrix}$$

and

$$H=\begin{bmatrix}
1 & 1 & 0 & & & & & & & & & \\
1 & 0 & 1 & & & & & & & & & \\
0 & 0 & 0 & 1 & 1 & 0 & & & & & & \\
1 & 0 & 0 & 1 & 0 & 1 & & & & & & \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & & & \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & & & \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
 & & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & & & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 & & & \vdots & & & & & & & & & \\
\end{bmatrix}.$$

2.5 ตัวอย่างของวงจร convolutional code

convolutional codes ที่รู้จักกันในปัจจุบันมีเพียงไม่กี่ชนิดเท่านั้น และใน code เหล่านั้นไม่มี code แบบใด ที่มีโครงสร้างที่ดีเหมือนกับ t-error-correcting BCH code ซึ่งเป็นที่รู้จักกันดี ซึ่งในปัจจุบันเรายังไม่มีวิธีใดที่จะใช้สำหรับหา convolutional codes ที่มี constraint length ที่ยาวเพียงพอ โดย convolutional coder ที่รู้จักและใช้กันในปัจจุบันได้ถูก คิดค้นจากการใช้ computer ช่วยในการออกแบบ code เหล่านั้น

ในส่วนที่เราจะกล่าวถึงแบบพื้นฐานของ single-error-correcting convolutional coder

โดยแบบของ sinfe-error-correcting binary convolutional coder ที่เรียกว่า wyner-Ash coder จะเป็นรูปแบบหนึ่งของ ltamming coder โดยสำหรับแต่ละจำนวนเต็มบวกหาจะได้นิยาม $[(m+1)2^m,(m+1)(2^m-1)]$ Wyner-Ash code ซึ่ง code นี้จะมี parity- check matrix H' ของ Hamming $(2^m-1,2^m-1-m)$ code โดยที่ parity- check เมตริกซ์จะมีขนาดของหลักทั้งหมด 2^m-1 หลัก และเราอาจเขียนเมตริกซ์ นี้โดยใช้เมตริกซ์แถวซึ่งมีขนาด 1 by(2^m-1) ซึ่งแทนด้วย $P^T1,...,P^T$ ซึ่ง P^T เป็น rowvector ซึ่งมีสมาชิก 2^m-1 ตัว โดยสามารถเขียน parity check matrix สำหรับ Wyner-Ash code ได้ดังนี้

$$H=\begin{bmatrix}
 & & & & & & & \vdots & \\
P_0^T & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
P_1^T & 0 & P_0^T & 1 & 0 & 0 & 0 & 0 & \\
P_2^T & 0 & P_1^T & 0 & P_0^T & 1 & 0 & 0 & \\
\vdots & & \vdots & & P_1^T & & & & \\
P_m^T & 0 & & & & & & & \\
0 & 0 & P_m^T & 0 & & & & & \\
\vdots & \vdots & & & & & & &
\end{bmatrix}$$

and

$$H^{((m-1)2m)} = \begin{bmatrix} P_0^T & 1 & 0 & 0 & 0 & 0 & & \\ P_1^T & 0 & P_0^T & 1 & 0 & 0 & & \\ P_2^T & 0 & P_1^T & 0 & P_0^T & 1 & & \\ \vdots & & & & & & & \\ P_m^T & 0 & P_{m-1}^T & 0 & P_{m-2}^T & 0 & \cdots & P_0^T \quad 1 \end{bmatrix}$$

ในที่นี้ 1 ถือเมตริกซ์ 1x1 ที่มีสมาชิก คือ 1 และ 0 คือเมตริกซ์1x1 ที่มีสมาชิกคือ 0

จากลักษณะของ Wyner-Ask code ซึ่งเป็น single-error-correcting convolutional code และจาก สูตร $2t+1<d_1^*$ เมื่อเราแทน t=1 เราจะได้ minimum distance d^* มีค่าเท่ากับ 3

สำหรับตัวอย่างของ Wyner-fisk (12,9) code ซึ่งมี m=2 จะได้ parity-check matrix คือ

$$H = \begin{bmatrix} 1\,1\,1\,1 & & & & \\ 1\,1\,0\,0 & 1\,1\,1\,1 & & & \\ 1\,0\,1\,0 & 1\,1\,0\,0 & 1\,1\,1\,1 & & \\ 0\,0\,0\,0 & 1\,0\,1\,0 & 1\,1\,0\,0 & 1\,1\,1\,1 & \cdots \\ 0\,0\,0\,0 & 0\,0\,0\,0 & 1\,0\,1\,0 & 1\,1\,0\,0 & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{bmatrix}$$

เมื่อ code นี้คิดเฉพาะ block Length 12 จะได้ parity-check matrix คือ

$$H^{12} = \begin{bmatrix} 1\,1\,1\,1 & 0\,0\,0\,0 & 0\,0\,0\,0 \\ 1\,1\,0\,0 & 1\,1\,1\,1 & 0\,0\,0\,0 \\ 1\,0\,1\,0 & 1\,1\,0\,0 & 1\,1\,1\,1 \end{bmatrix}$$

และ generator matrix คือ

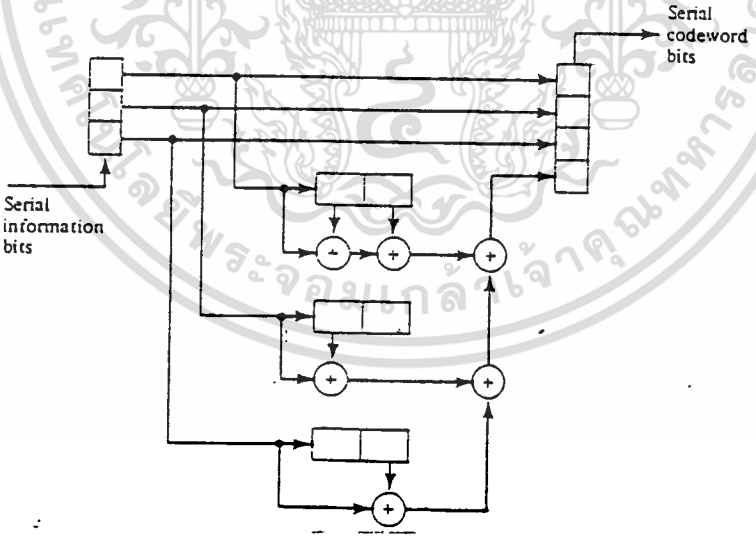
The generator matrix

$$G=\begin{bmatrix}
1&0&0&1&0&0&0&1&0&0&0&1 & & & & & & & & & & \\
0&1&0&1&0&0&0&1&0&0&0&0 & & & & & & & & & & \\
0&0&1&1&0&0&0&0&0&0&0&1 & & & & & & & & & & \\
 & & & &1&0&0&1&0&0&0&1&0&0&0&1 & & & & & & \\
 & & & &0&1&0&1&0&0&0&1&0&0&0&0 & & & & & & \\
 & & & &0&0&1&1&0&0&0&0&0&0&0&1 & & & & & & \\
 & & & & & & & &1&0&0&1&0&0&0&1&0&0&0&1 & \\
 & & & & & & & &0&1&0&1&0&0&0&1&0&0&0&0 & \cdots \\
 & & & & & & & &0&0&1&1&0&0&0&0&0&0&0&1 & \\
 & \vdots & & & & & & & & \vdots & & & & & & & & & & &
\end{bmatrix}.$$

และ generator matrix สำหรับ block length 12 คือ

$$G^{(12)}=\begin{bmatrix}
1&0&0&1&0&0&0&1&0&0&0&1 \\
0&1&0&1&0&0&0&1&0&0&0&0 \\
0&0&1&1&0&0&0&0&0&0&0&1
\end{bmatrix}.$$

โดยวงจร encoder สำหรับ (12,9) Wyner-Ash code ได้แสดงในรูปที่ 11 โดยวงจรดังรูปจะแสดง FIR filter สำหรับแต่ละ generator polynomial



รูปที่ 2.11 วงจรเข้ารหัสขนาด (12,9) แบบ Wyner - Ash code

Convolutional coder ที่ได้รับการยอมรับโดยส่วนมาก มักได้มาจากการใช้ compater ช่วยในการคำนวณหา coder เหล่านี้ได้แสดงในรูป 12

(n, k, d_∞)	Matrix of Generator Polynomials (in Terms of Polynomial Coefficients)			
(6, 3, 5)	$(x^2 + 1)$	101	$(x^2 + x + 1)$	111
(8, 4, 6)	$(x^3 - x + 1)$	1011	$(x^3 + x^2 + x + 1)$	1111
(10, 5, 7)		11001		10111
(12, 6, 8)		110101		101111
(14, 7, 10)		1101101		1001111
(16, 8, 10)		11100101		10011111
(18, 9, 12)		100011101		110101111
(20, 10, 12)		1110111001		1010011011
(22, 11, 14)		10111011001		10001101111
(24, 12, 15)		101110110001		11001011101
(26, 13, 16)		110110101001		10001101111111
(28, 14, 16)		101111011100001		11001010011101
(9, 3, 8)		101	111	111
(12, 4, 10)		1101	1011	1111
(15, 5, 12)		10101	11011	11111
(18, 6, 13)		111001	110101	101111
(21, 7, 15)		1101101	1010011	1011111
(24, 8, 16)		10101001	10011011	11101111
(27, 9, 18)		111101101	110011011	100100111
(30, 10, 20)		1111001001	1010111101	1101100111
(33, 11, 22)		11010111001	10011101101	10111110011
(36, 12, 24)		111011111001	110010111101	101011010011
(39, 13, 24)		110110101001	101111010001	100011011111
(42, 14, 26)		101000101110001	100011011110111	110110100111111
(12, 3, 10)	101	111	111	111
(16, 4, 13)	1101	1011	1011	1111
(20, 5, 16)	10101	11011	10111	11111
(24, 6, 18)	110101	111011	100111	101111
(28, 7, 20)	1011101	1011101	1110011	1100111
(32, 8, 22)	10111001	10111101	11010011	11110111
(36, 9, 24)	110011001	101110101	110110111	101001111
(40, 10, 27)	1111001001	1010111101	1101100111	1101010111
(44, 11, 29)	11101011001	11010111001	10011101101	10111110011
(48, 12, 32)	111011111001	110010111101	101011010011	101101001111
(52, 13, 33)	1010011001001	1111110010101	1101111011011	1110101110111
(56, 14, 36)	11010010010001	10111110011001	11101010110111	11111010110111

รูปที่ 12 เป็นการแสดง Noncatastrophic binary convolutional coder กับค่า maximam fru distance.

2.6 วิธีการหา syndrome ของวงจรถอดรหัส

โดยทั่วไปในภาครับหากเราได้รับลำดับที่มีความยาวอนันต์ของ v ซึ่งประกอบด้วย convolutional code word และ error patlern โดย

$$V = c+e$$

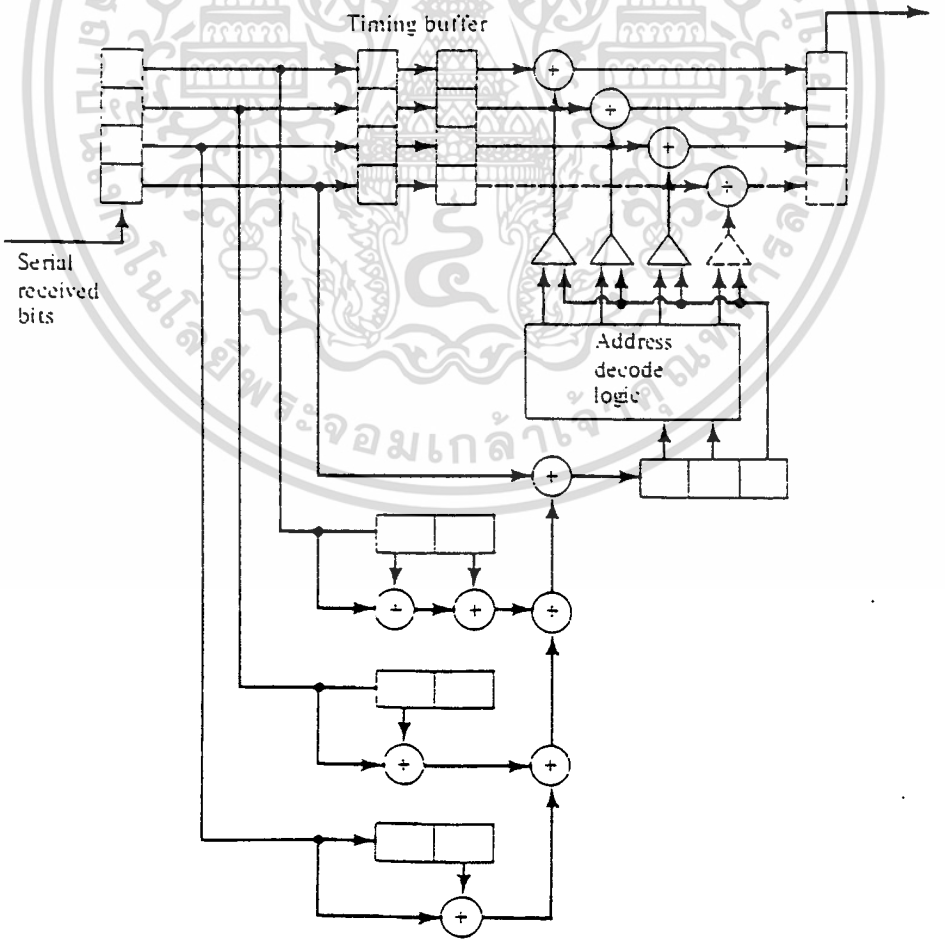
ซึ่งเราสามารถคำนวณ syndrame ได้จาก

$$S = VH^T = eH^T$$

โดย S แทน syndrome ที่เกิดขึ้นจาก error

อย่างไรก็ตาม syndrome ของ infinite length วงจร decoder สามารถมี syndrome ทั้งหมดได้แต่ decoder จะพิจารณาโดยเริ่มคำนวณ syndrome จาก ลำดับของ S ที่เข้ามาในแต่ละเวลาและจะสามารถตรวจแก้ลำดับของ S ที่เกิดการผิดจาก syndrome ที่มีอยู่ ซึ่ง decoder จะมีตารางที่แสดงส่วนของ syndrome และ error-pattern segment ที่เป็นสาเหตุที่ทำให้เกิดsyndrome แต่ละแบบ เมื่อ decoder มองเห็น syndrome ที่อยู่ในตารางก็จะ สามารถตรวจแก้ส่วนของ code word ที่ผิดให้ถูกได้

เราจะศึกษารูปแบบของ syndrome ต่างๆ โดยในตัวอย่างแรกเราจะศึกษาวงจร decoder จาก (12,9) Wyner-Ash code ซึ่งแสดงในรูป 13



รูปที่ 2.13 วงจรถอดรหัสขนาด (12,9) แบบ Wyner -Ash code

จากรูปที่ 13 codeword จะไหลเข้าสู่วงจร decoder แบบอนุกรม และจะแปลงให้อยู่ในลักษณะของเส้นขนาน n_0 เส้น และsyndrome จะคำนวณย้อนกลับของ parity bit จาก information bits ที่รับเข้ามาและจะเปรียบเทียบผลที่ได้กับ paritybit ที่รับเข้ามา เราจำเป็นต้องคำนวณเพียง
syndrome bit ที่เกิด single error ในเฟรมแรกซึ่งสามารถแก้ไขได้ โดยรูปแบบของ single error ที่เป็นไปได้ใน3เฟรมแรกและ 3 bit แรกของ syndrome ได้แสดงในรูปที่ 14 จากรูปที่ 14 สังเกตุได้ว่าหาก error เกิดในเฟรมแรก บิทแรกของ syndrome bit จะเป็นบิทที่บอกว่าเกิด error ได้ เกิดขึ้นใน frame แรกและอีก 2 บิทที่เหลือจะบอกตำแหน่งของ bit ที่เกิด error

ในรูปที่ 13 ส่วนที่แสดงเส้นประจะถูกใช้ในวงจรหากวงจรถูกออกแบบให้หาโครงสร้างของ codeword ที่ได้รวมกับ parity bit แต่ถ้าไม่ต้องการก็สามารถตัดทิ้งได้

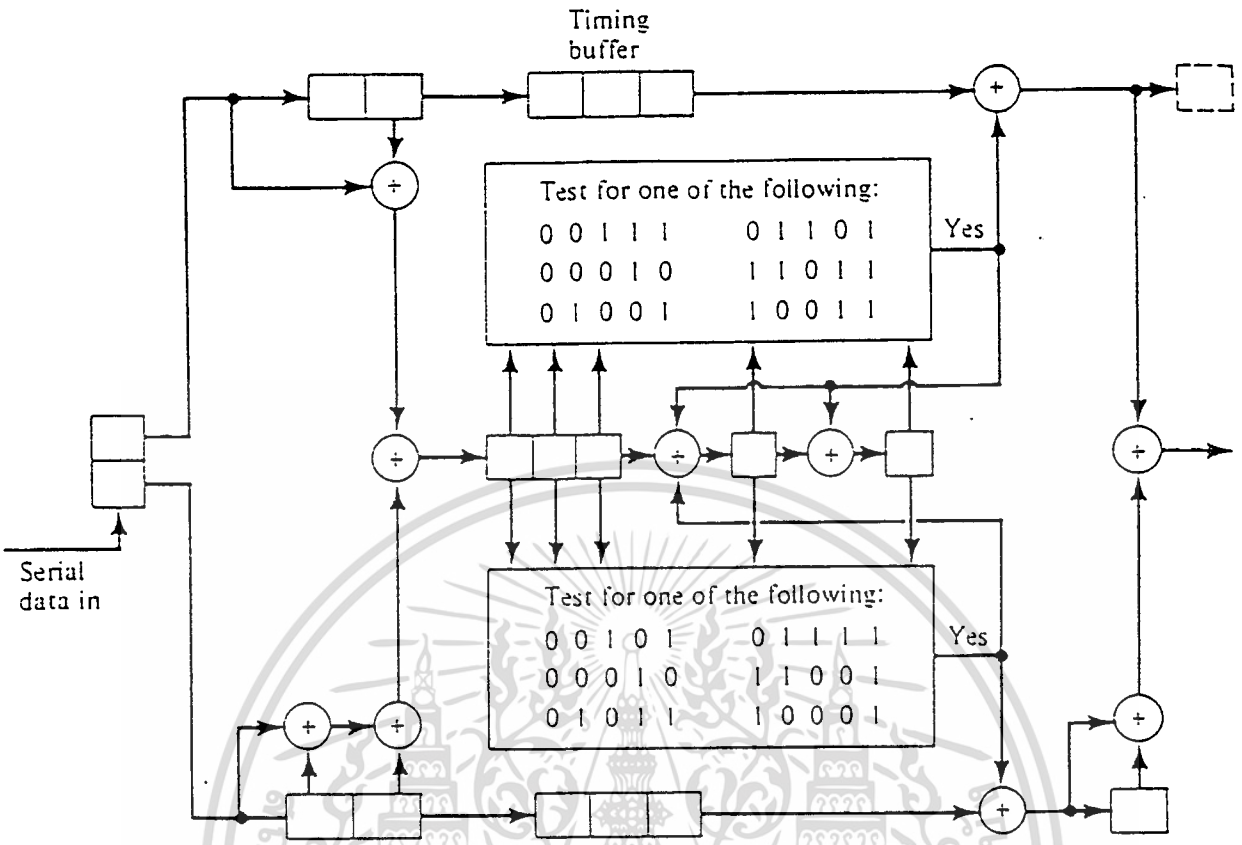
ปัญหาหนึ่งที่เกิดขึ้นกับวงจร decoder ดังรูปส่งหลังวงจรเฟรมแรกถูกแก้แล้ว syndrome reyster จะต้องถูกแก้ไข หากไม่แก้ไขจะส่งผลให้ไม่สามารถตรวจแก้เฟรมต่อไปได้ ซึ่งวิธีที่จะใช้ในการแก้ไข syndrome register มีอยู่หลายวิธีด้วยกัน ซึ่งเราอาจแก้ไขโดยใช้วิธี ดังนี้

1. โดยการ set syndrome register ให้เป็น 0 ในแต่ละเวลาที่ error ได้ถูกแก้แล้ว
2. อาจลบ syndrome ที่ใช้ตรวจแก้จาก syndrome register

ในการแก้โดยวิธีแรกเป็นลักษณะพิเศษของ single-error-correcting coder ส่วนในวิธีที่ 2 เราจะใช้ในรูปแบบที่ยากขึ้น

Error Pattern				Syndrome
Fourth Frame	Third Frame	Second Frame	First Frame	
.....	0 0 0 0	0 0 0 0	0 0 0 1	1 1 1
	0 0 0 0	0 0 0 0	0 0 1 0	0 1 1
	0 0 0 0	0 0 0 0	0 1 0 0	1 0 1
	0 0 0 0	0 0 0 0	1 0 0 0	0 0 1
	0 0 0 0	0 0 0 1	0 0 0 0	1 1 0
	0 0 0 0	0 0 1 0	0 0 0 0	1 1 0
	0 0 0 0	0 1 0 0	0 0 0 0	0 1 0
	0 0 0 0	1 0 0 0	0 0 0 0	0 1 0
	0 0 0 1	0 0 0 0	0 0 0 0	1 0 0
	0 0 1 0	0 0 0 0	0 0 0 0	1 0 0
	0 1 0 0	0 0 0 0	0 0 0 0	1 0 0
	1 0 0 0	0 0 0 0	0 0 0 0	1 0 0

รูปที่ 2.14 แสดงรูปแบบของ error ขนาด (12,9) Wyner-Ash code



รูปที่ 2.15 วงจรถอดรหัสขนาด (6,3)

Error Pattern				Syndrome
Fourth Frame	Third Frame	Second Frame	First Frame	
.....	0 0	0 0	0 1 ...	... 0 0 1 1 1
	0 0	0 0	1 0	0 0 1 0 1
	0 0	0 0	1 1	0 0 0 1 0
	0 0	0 1	0 1	0 1 0 0 1
	0 0	1 0	0 1	0 1 1 0 1
	0 1	0 0	0 1	1 1 0 1 1
	1 0	0 0	0 1	1 0 0 1 1
	0 0	0 1	1 0	0 1 0 1 1
	0 0	1 0	1 0	0 1 1 1 1
	0 1	0 0	1 0	1 1 0 0 1
	1 0	0 0	1 0	1 0 0 0 1

รูปที่ 2.16 แสดงรูปแบบ error ของ convolutional code ขนาด (6,3)

จากรูปที่ 3 วงจร (6,3) double-error correcting code จะมีวงจร decoder ดังรูป 15 โดยวงจร decoder นี้จะมีตาราง syndrome ดังรูป 16 โดยในวงจรนี้ จะสามารถตรวจแก้ two errors ที่ เกิดภายใน 6 บิทแรก โดยการ feedback จะออกจาก syndrome register จะช่วยในการแก้ error ที่เกิดขึ้นเพราะว่า code นี้ไม่ใช้ syntematic code information bit จะถูกถอดรหัสจาก codeword ที่ถูกต้องแล้วโดยวงจรที่ใช้ จะมี relationship $I = GCD[x^2+x+1_9x^2+1]=x(x^2+x+1)+(x+1)(x^2+1)$ ในการใช้ถอดรหัส

วงจรถอดรหัสจากรูป 15 ยังเป็นวงจรที่ไม่สมบูรณ์ เพราะยังมีsyndromer ที่เป็นไปได้อีกมากที่ยังไม่ได้นำมาใช้ โดยหลักการแล้วการถอดรหัสจะต้องใช้ error pattern โดยใช้ตาราง syndromer ที่สามารถขยาย syndrome memory ได้ แต่ในการปฏิบัติการเลือก code ที่มี block length ตามต้องการน่าจะเป็นทางเลือกที่ดีกว่าในการปรับปรุง performane

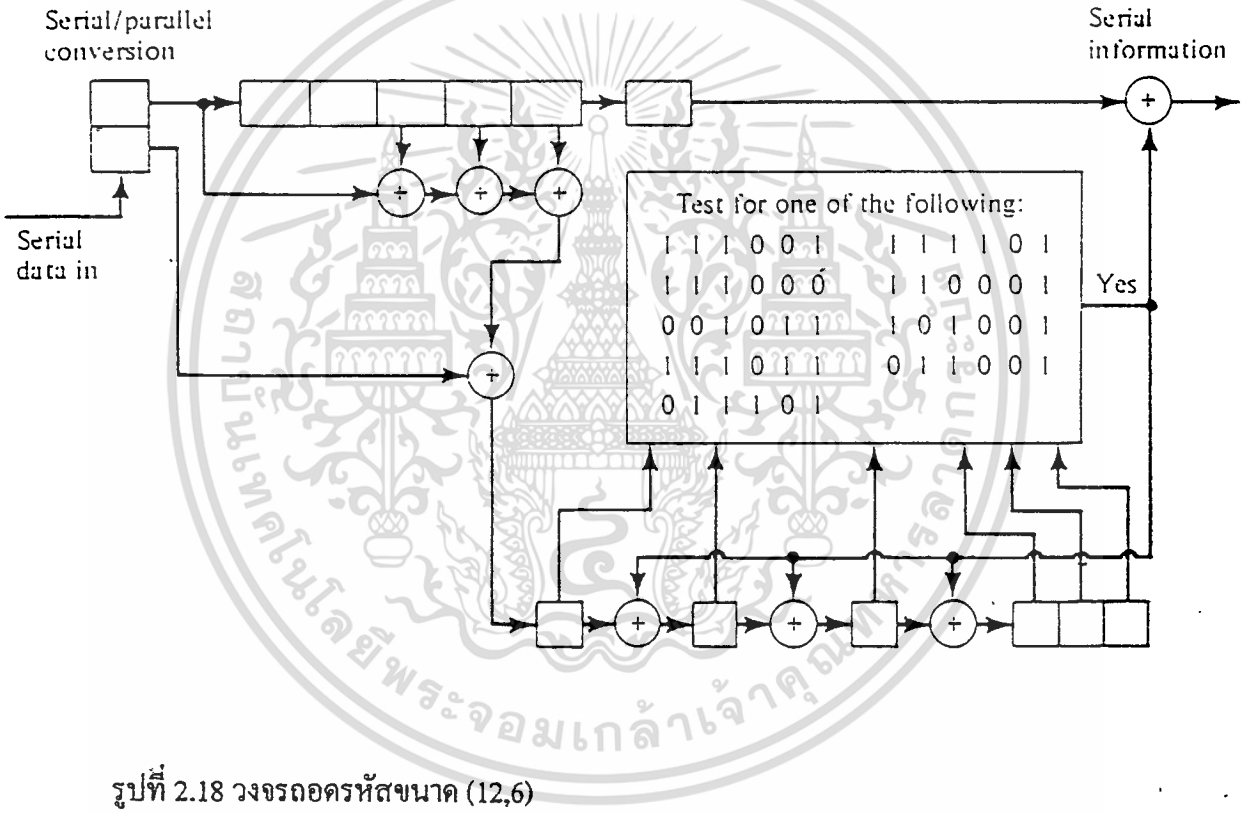
Error Pattern	Syndrome
... 0 0 0 0 0 0 0 0 0 0 0 0 1	... 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 1	1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 1	0 0 1 0 1 1
0 0 0 0 0 0 0 0 0 1 0 0 1	1 1 1 0 1 1
0 0 0 0 0 0 0 0 1 0 0 0 1	0 1 1 1 0 1
0 0 0 0 0 0 0 1 0 0 0 0 1	1 1 1 1 0 1
0 0 0 0 0 0 1 0 0 0 0 0 1	1 1 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 0 1	1 1 0 0 0 1
0 0 0 0 1 0 0 0 0 0 0 0 1	1 0 1 0 0 1
0 0 0 1 0 0 0 0 0 0 0 0 1	1 1 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 1	0 1 1 0 0 1
0 1 0 0 0 0 0 0 0 0 0 0 1	0 1 1 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1	0 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 0	1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 1 0 1 0	0 0 0 0 1 1
0 0 0 0 0 0 0 0 1 0 0 1 0	1 0 0 1 0 1
0 0 0 0 0 0 0 1 0 0 0 1 0	0 0 0 1 0 1
0 0 0 0 0 0 1 0 0 0 0 1 0	0 0 1 0 0 1
0 0 0 0 0 1 0 0 0 0 1 0	0 0 1 0 0 1
0 0 0 0 1 0 0 0 0 0 1 0	0 1 0 0 0 1
0 0 0 1 0 0 0 0 0 0 1 0	0 1 0 0 0 1
0 0 1 0 0 0 0 0 0 0 1 0	1 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0 1 0	1 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 1 0	1 0 0 0 0 1

รูปที่ 2.17 แสดงรูปแบบ error ของ convolutional code ขนาด (12,6)

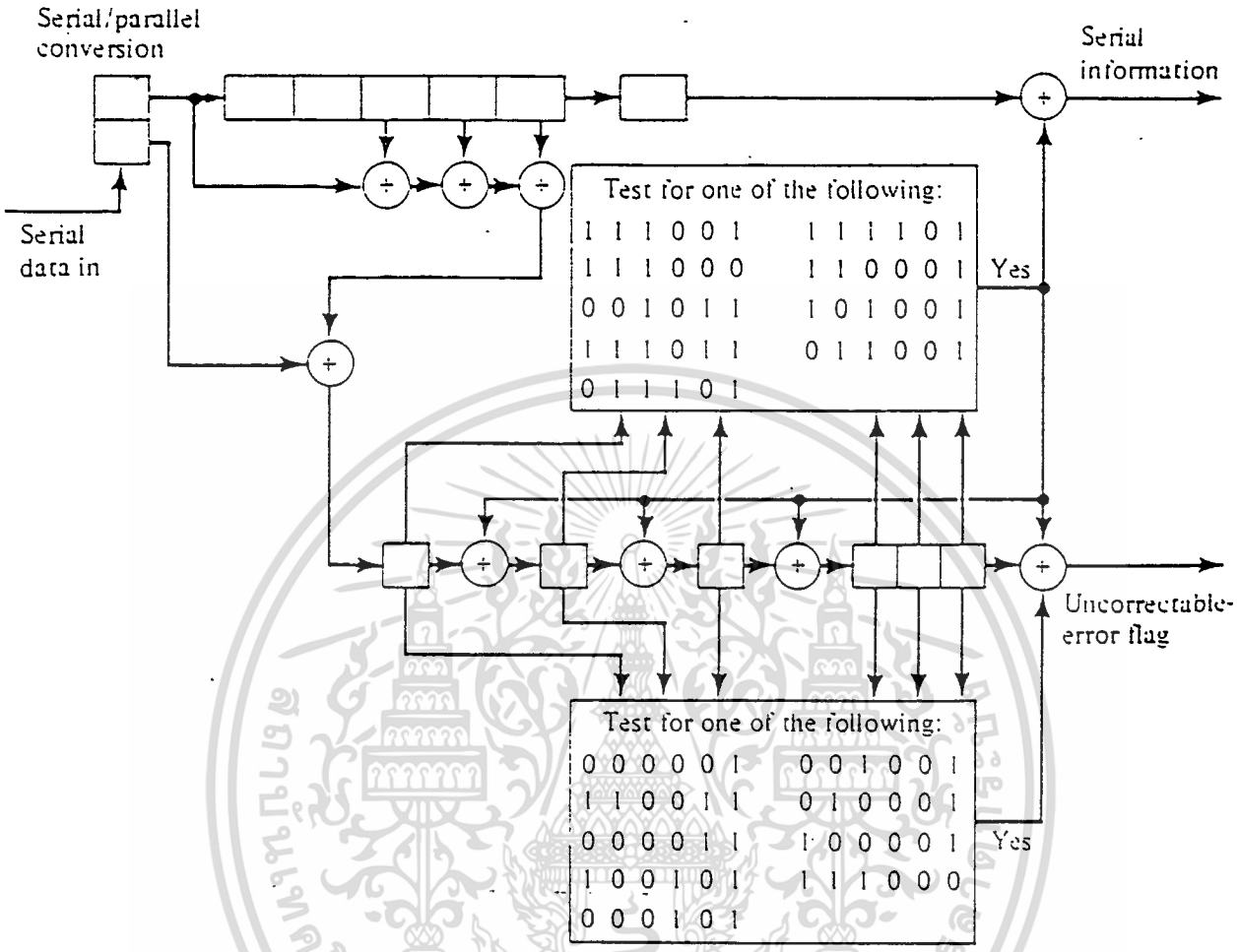
ในตัวอย่างสุดท้ายเราจะพิจารณา (12,6) convolutional code ซึ่ง code นี้สามารถแก้ error 2 บิท ใน 12 บิทแรก โดยจะใช้ syndrome decoding ดังแสดงในตารางรูปที่ 17 จากรูปที่ 18 ส่วนบนจะแสดง error patterns ที่ error เกิดใน information bit และในส่วนล่างคือ error pattern ที่เกิด error ใน parity bit แต่ไม่เกิด error ใน information bit

ในรูปที่ 18 เป็นวงจร decoder ที่ตรวจสอบเพียง syndromer ที่เกิดจาก error ที่เกิดใน information bit แต่ decoder จะไม่สามารถ ตรวจจับ uncorrectable

ส่วนในรูปที่ 19 เป็นวงจร decoder ที่สามารถตรวจจับ errorได้และสามารถแก้ไขได้เฉพาะ error ที่เกิดภายใน limits ของการ design



รูปที่ 2.18 วงจรถอดรหัสขนาด (12,6)



รูปที่ 2.19 ตัวอย่างวงจรถอดรหัสขนาด (12,6)

บทที่3

หลักการของภาษาวีเอชดีแอล

วีเอชดีแอล (VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC ย่อมาจาก Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) ซึ่งใช้อธิบายการทำงานของระบบดิจิตอลฮาร์ดแวร์ สามารถใช้อธิบายฟังก์ชันการทำงานได้หลาย ๆระดับ ตั้งแต่ระดับบล็อกจนถึงระดับเกท ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับเกท ประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษาวีเอชดีแอลนั้นจะประกอบไปด้วย 2 ส่วนใหญ่ ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล (Siqutmtial Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การโปรแกรมด้วยภาษาวีเอชดีแอลสามารถเขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใด ๆ ย่อมจะมีการทำงานในแบบ ซีเควนเชียลและ คอนเคอร์เรนท์ อยู่รวมกัน นอกจากนี้ตัวภาษาวีเอชดีแอลยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อย ๆ เข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษาวีเอชดีแอลนอกจากจะกำหนดรูปแบบไวยากรณ์ (Syntax) ของตัวภาษาแล้ว ยังมีการตรวจสอบความหมายของตัวภาษาว่าจะซิมูลเลชัน (Simulation) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการซิมูลเลชันซีแมนติค (Semantics) อย่างไรก็ตามแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานโดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมดเนื่องจากตัวภาษาวีเอชดีแอลออกแบบมาให้ใช้สำหรับการออกแบบตั้งแต่วงจรที่มีขนาดเล็กถึงวงจรที่มีขนาดใหญ่และซับซ้อน

3.1ความสามารถของภาษาวีเอชดีแอล (CAPABILITY)

- ตัวภาษาวีเอชดีแอลสามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิปกับผู้ออกแบบ (CAD Tools)

- ใช้เป็นการสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่างซีเออี (CAE) และซีเอดีทูล (CAD Tools) เช่นตัวภาษาซ้อสโค้ด (Source Code) ของวีเอชดีแอล สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และซิมูลเลเตอร์ (Simulator) ได้หลายตัวแตกต่างกัน

- ภาษาวีเอชดีแอลสนับสนุนการออกแบบ แบบท็อปดาวน์ (Top Down Design) และแบบบัททอมอับ (Bottom Up Design) หรือผสมกันทั้ง 2 แบบ

- ตัวภาษาวีเอชดีแอลเป็นแบบทั่วไป (Generic) คือ ไม่อิงเทคโนโลยีอันใดอันหนึ่ง สามารถอิงเทคโนโลยีใดก็ได้ และในขณะเดียงกันก็สามารถสนับสนุนหลายๆ เทคโนโลยี

- สนับสนุนการออกแบบทั้งระบบซิงโครนัส (Synchronous) และอะซิงโครนัส (Asynchronous)

- สนับสนุนการออกแบบระบบดิจิตอล ในหลาย ๆ เทคนิค เช่นไฟไนท์สเตทแมชชีน (Finite State Machine), อัลกอริทะมิก (Algorithmic) หรือสมการบูลีน (Boolean Equation)

- ตัวภาษาวีเอชดีแอลเป็นมาตราฐานรับรองโดย IEEE และ ANSI ทำให้โมเดลสามารถเคลื่อนย้ายไปยังระบบใด ๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้

- ภาษาวีเอชดีแอลสามารถอ่านและทำความเข้าใจได้โดยมนุษย์

- ภาษาวีเอชดีแอลสนับสนุนรูปแบบการเขียนถึง 3รูปแบบ ได้แก่ แบบบีเฮฟวีเออร์ (Behavioral Style) แบบสทรักเทอรัล (Structural Style) แบบดาต้าโฟลว์ (Data Flow) หรือสามารถเขียนรวมกันทั้ง 3 รูปแบบ

- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ (Component), ฟังก์ชั่นโพลซีเดอร์ (Function Procedure) และแพคเกต (Package)

- ไม่จำเป็นต้องศึกษาซอฟท์แวร์ (Software) ซิมูลเลเตอร์เพราะซิมูลเลชันโมเดลสามารถเขียนได้โดยใช้ภาษาวีเอชดีแอล เช่นกัน

- สามารถเขียนดมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับซอฟท์แวร์)

- สามารถเขียนอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation Delay, Min-Max Delay, Setup, Holding Time, Spige Detection สามารถอธิบายได้ภายในตัวภาษา

- เจเนริกส์ (GENERICS) ช่วยให้เราสามารถสร้างตัวแปรของรูปแบบ (Design)

- โมเดลที่สร้างด้วยภาษาวีเอชดีแอลนั้น ไม่เพียงแต่จะอธิบายฟังก์ชั่นการทำงานเท่านั้น แต่ยังสามารถอธิบายถึงรายละเอียดของตัวโมเดล เช่น Total Area และ Speed ของโมเดล

- ภาษาวีเอชดีแอลเป็นมาตราฐานที่ใช้โดยบริษัทและผู้ออกแบบหลาย ๆ แห่ง ฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้ว่าจะมาจากแหล่งต่าง ๆ

- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปลภาษาได้ตรวจสอบไวยากรณ์ทางด้านซิมูลเลชันซิแมนติคไว้ด้วย

- การอธิบายโมเดลด้วยแบบบีเฮฟวีเออร์สามารถ Synthesis ไปเป็นระดับเกทเลเวลได้ถ้าทำตามกฎของ Synthesis Guideline

- มีความสามารถที่ให้เราออกแบบข้อมูลชนิดใหม่ ๆ ได้ทำให้วีเอชดีแอลโมเดล เป็นการออกแบบในระดับสูง ที่ไม่ต้องคำนึงถึงว่าจะสร้างตัวโมเดลนั้นขึ้นมาได้อย่างไร

3.2 ประวัติความเป็นมาของภาษาวีเอชดีแอล

ความต้องการภาษานี้เริ่มจากโครงการวีเอชเอสไอซี (VHSIC) ของ Deparment Of Defence ของสหรัฐอเมริกาเนื่องจากมีบริษัทที่สร้างวีเอชเอสไอซีชิป (Chip) หลายบริษัทได้ร่วมโครงการที่จะพัฒนา ในขณะนั้นหลายบริษัทใช้ภาษาวีเอชดีแอล ซึ่งแตกต่างกันในการที่จะอธิบายการทำงานชิปของตน ด้วยเหตุนี้ทำให้เกิดความแตกต่าง แต่ละบริษัทไม่สามารถแลกเปลี่ยนเทคโนโลยีให้กันและกันได้ ทำให้ DOD เกิดปัญหาในการที่จะพัฒนาและซ่อมบำรุงในภายหลังจึงเกิดความต้องการภาษาวีเอชดีแอล ซึ่งเป็นมาตราฐานในการที่จะอธิบายถึงตัวแบบนั้น ๆ ดังนั้น DOD จึงมอบให้บริษัทไอบีเอ็ม (IBM) , เท็กซัสอินสทรูเมนต์ (TEXAS INSTRUMENT) และ อินเตอร์เมติกส์ (INTERMETICS) 3 บริษัทร่วมกันพัฒนาและกำหนดมาตราฐานของวีเอชดีแอลขึ้นมาในปี 1983 หลังจากนั้น วีเอชดีแอลเวอชั่น 7.2 (VHDL VERTION 7.2) ได้ทำการพัฒนาและออกเผยแพร่ต่อสาธารณะชนในปี 1985 ได้รับความสนใจเป็นอย่างมากในอุตสาหกรรม โดยเฉพาะอย่างยิ่งบริษัทที่ทำวีเอชเอสไอซีชิป จากผลสำเร็จนี้ทำให้เกิดมาตราฐาน IEEE ของวีเอชดีแอลในปี 1986 หลังจากนั้นก็มีการพัฒนาขยายขีดความสามารถของภาษาวีเอชดีแอลเพิ่มขึ้น และ DOD ก็มีการปรับปรุงและจดมาตราฐานใหม่ IEEE ในปี 1988 อีกครั้ง ซึ่งเป็นที่รู้จักกันในชื่อของ IEEE STD 1076-1987 หลังจากกันยายน 1988 บริษัทใด ๆ ที่ทำการพัฒนาเอเอสไอซี (ASIC) ชิป ใน Department Of Defence ของอเมริกาต้องส่งตัววีเอชดีแอลโมเดลพร้อมกับชุดทดสอบตามมาตราฐานที่ได้กำหนดเอาไว้

3.3หลักการสร้างโมเดลโดยภาษาวีเอชดีแอล (General VHDL Modelling Principles)

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ ซึ่งช่วยในการสร้างและออกแบบวงจรรวมดิจิตอล และ ส่วนประกอบต่าง ๆ อาจใช้อธิบายระบบทั้งระบบ หรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ Component Block จากนั้นก็ทำการจำลองการทำงาน (Sinulate) โดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริง หรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังการจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจรเกทเลเวลต่อไป

ประโยชน์จริงของการใช้ วีเอชดีแอลเป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริง คือเราสามารถอธิบาย Product Idea , product Proposal , Product Specification เป็นลักษณะในรูปของ Text จากนั้นก็นำไปคอมไพล์เพื่อดู Timing การทำงาน จากนั้นก็ทำการ Refine แก้ไขจนกว่าจะได้ Specification ตามต้องการ เมื่อ Product ได้ผลตามที่ต้องการแล้วจึงนำไปเข้าสู่การ Synthesis เพื่อให้ได้เกทเลเวล Schematic แล้วนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบที่สร้างนั้นทำงานได้จรองเพราะได้ทำการ Simulate เรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

เนื่องจากว่าภาษาวีเอชดีแอลเป็นภาษาที่มีประสิทธิภาพสูง เราจึงใช้ภาษาวีเอชดีแอลอธิบายฮาร์ดแวร์ เพราะว่ามีข้อดี 2 ประการ คือ

1. เข้าใจได้ง่าย
2. สามารถแก้ไขได้ง่าย

การเข้าใจได้ง่ายมีประโยชน์ต่อใครก็ได้ซึ่งมีความจำเป็นที่จะต้องอ่านรหัส ที่ได้ออกแบบมาแล้วโดยไม่จำเป็นต้องให้ผู้ออกแบบมาอธิบายให้ฟัง ตัวภาษาวีเอชดีแอลอธิบายการทำงานภายในตัวอยู่แล้ว ส่วนอีกประการหนึ่งก็คือ ความต้องการในการเปลี่ยนแปลงฮาร์ดแวร์ ที่ได้ออกแบบแล้ว คือว่า หลังจากทดสอบแล้วพบข้อผิดพลาดซึ่งต้องแก้ไข หรือว่า ระหว่างพัฒนามีการเปลี่ยนแปลงความต้องการของระบบ หรือต้องการเพื่อการทำงานบางส่วนลงไป ตัวภาษาวีเอชดีแอล นั้นสนับสนุนหลักการต่าง ๆ ให้เขียนแก้ไข และบำรุงรักษาวงจรดิจิตอล ที่มีความซับซ้อนเป็นไปอย่างรวดเร็ว และมีประสิทธิภาพ ซึ่งหลักการมีดังนี้

1. Top Down Design
2. Modularity
3. Abstraction
4. Information Hiding
5. Uniformity

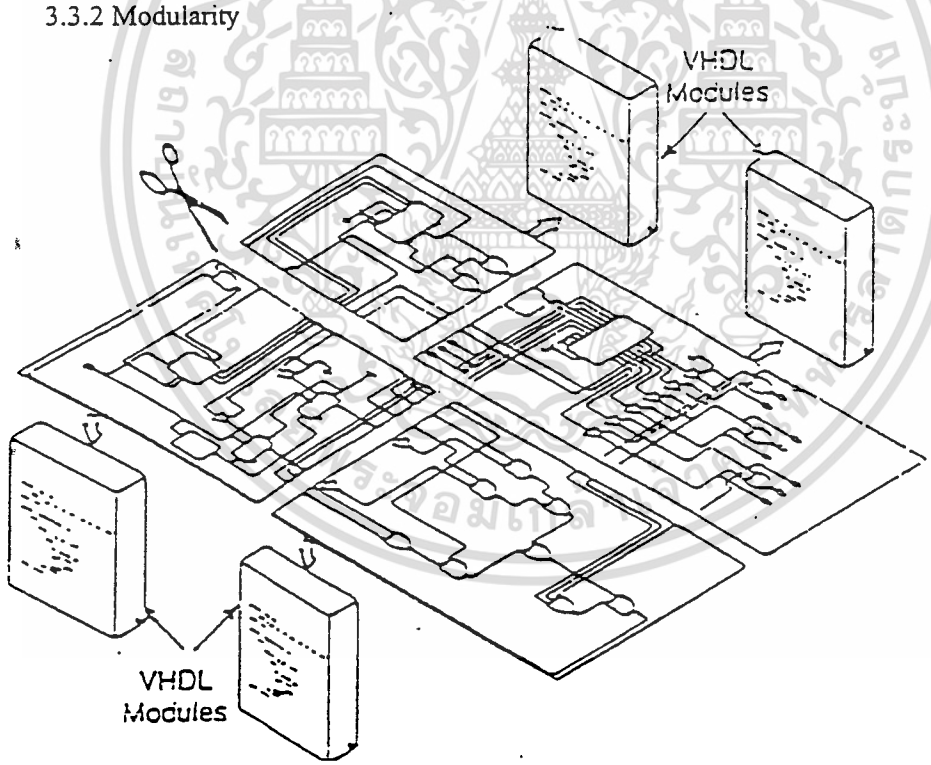
ซึ่งจะอธิบายประโยชน์ของหลักการต่าง ๆ ในหัวข้อต่อไป และแสดงให้เห็นว่า ภาษาวีเอชดีแอลนั้นช่วยในการพัฒนาวงจรดิจิตอล ขนาดใหญ่และซับซ้อนนั้นให้อ่านเข้าใจได้ง่าย และแก้ไขได้ง่ายอย่างไร

3.3.1Top down design

ในการพัฒนาวงจรร่วมดิจิตอล ขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Appication Specific Integrate Circuit) วิศวกร หรือผู้ออกแบบมักจะมองรูปแบบให้อยู่ในรูปของ Block Diagram เสียก่อน ก่อนที่จะออกย่อยรูปแบบ ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษาวีเอชดีแอลนั้นอนุญาตให้

- อธิบายการทำงานของแต่ละ Block
- วิเคราะห์การทำงาน (Analyze)
- จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการ ก่อนที่จะทำการออกแบบให้ละเอียดลึกลงในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการแก้ไขในช่วงของการพัฒนาในระดับสร้างซิลิกอนชิป (Silicon Chip)

3.3.2 Modularity

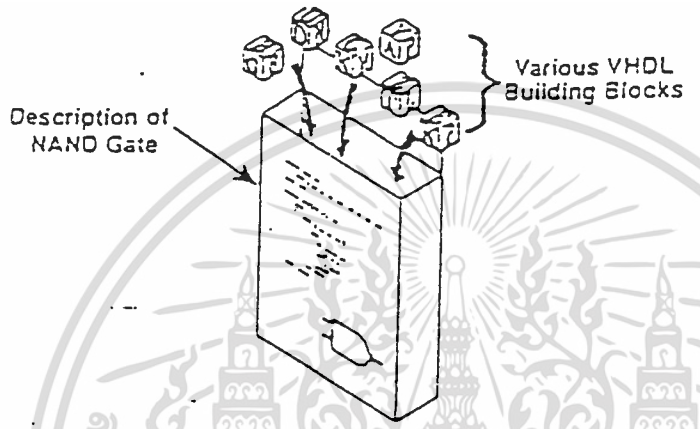


รูปที่ 3.1 การแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์

Modularity คือ หลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ ออกเป็นส่วนย่อยเล็ก ๆ ลงไป ซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ต้องประกอบด้วยฮาร์ดแวร์ย่อย ๆ ลงไป ดังรูปที่ 2.2 แสดงรูปแบบ ซึ่งแสดงวงจรทั้งหมดในรูปเดียง (Flatten Design) หลังจากนั้นตัดออกเป็นส่วนย่อย ๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ภาษาวีเอชดีแอลหน้าที่การทำงานของแต่ละส่วน

สามารถอธิบายได้โดย Module ของ Code (คล้าย Function หรือ Procedure) ซึ่งแสดงการทำงานของส่วนจ่อยนั้นอย่างชัดเจน ซึ่งการแยกรูปแบบใหญ่ ๆ ออกเป็นส่วนย่อย ๆนี้ทำให้ง่ายต่อการจัดาการและง่ายต่อการทำความเข้าใจ

ตัวภาษาวีเอชดีแอลประกอบขึ้นมาด้วย Language Building Block ซึ่งประกอบไปด้วย 75 Reverse Word และมากกว่า 200 Combination Words รูปที่ 3.2แสดงให้เห็นว่า ภาษาวีเอชดีแอลแต่ละ Module นั้นประกอบด้วย Language Building Block อะไรบ้าง และอธิบายการทำงานของ NAND เกท



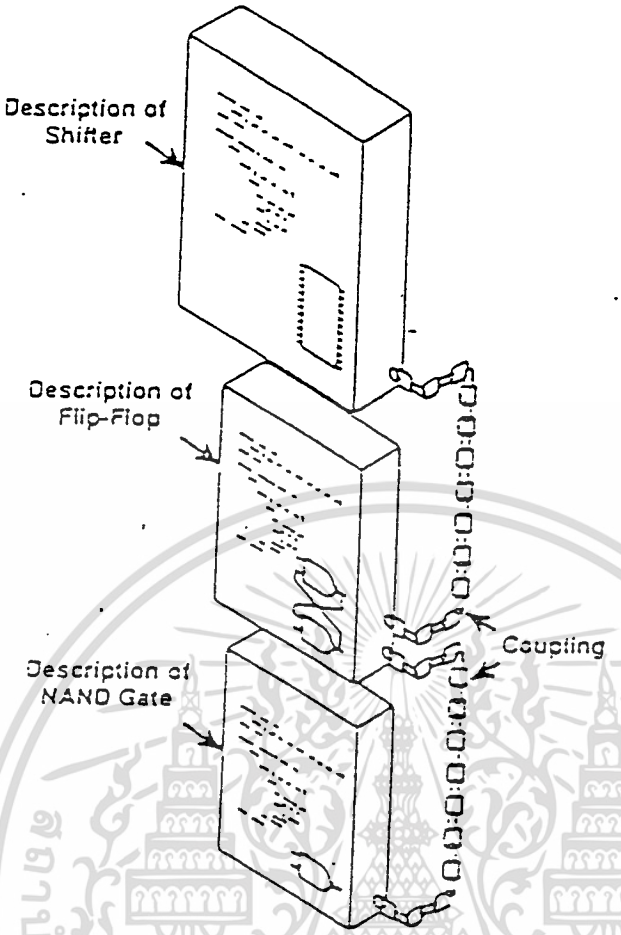
รูปที่ 3.2 ฮาร์ดแวร์โมดูลซึ่งสร้างจากการสร้างบล็อคของ VHDL

จากรูปที่ 3.3 แสดง Hierarchy- Method โดยการแยกส่วนรูปแบบออกเป็นส่วนย่อย ๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่าง ๆ ลงมาคือการแยกส่วนของ Shifter ออกเป็นฟลิปฟลอป จาก ฟลิปฟลอป แยกเป็น NAND เกท ภายใน Shifter ได้อธิบายการทำงานโดยใช้การต่อกันของฟลิปฟลอป ในระดับต่ำลงมา ฟลิปฟลอปก็เกิดจากการใช้ NAND เกทต่อกัน 2 ตัว ในระดับต่ำลงมาอีกก็เป็น NAND เกท ซึ่งมีการอธิบายการทำงานอยู่ภายใน ซึ่งแต่ละ Module จะมีคำอธิบายการทำงานในตัวของมันเองอยู่แล้ว คำอธิบายภายในแต่ละ Module ก็อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ NAND เกท ในระดับล่างสุดได้

ประโยชน์อย่างหนึ่งของการแยกส่วนฟลิปฟลอป และ NAND เกท ออกจากกัน เนื่องจากว่าทำให้ง่ายในการที่จะใช้ NAND เกทตัวนี้ในรูปแบบไฮเลเวล (High Level) ตัวอื่น ๆ ทำให้นำออกใช้ได้อีก และลดความซับซ้อนในการใช้อุปกรณ์ส่ง เป็นการง่ายที่จะแก้ไขการทำงานของ Shifter โดยปราศจากการแก้ไข Filp-Flop และ NAND เกท จากประโยชน์ที่ได้ของ Modularity นี้ทำให้รูปแบบที่เราออกแบบนี้ง่ายต่อการเข้าใจและแก้ไขได้เสมอ

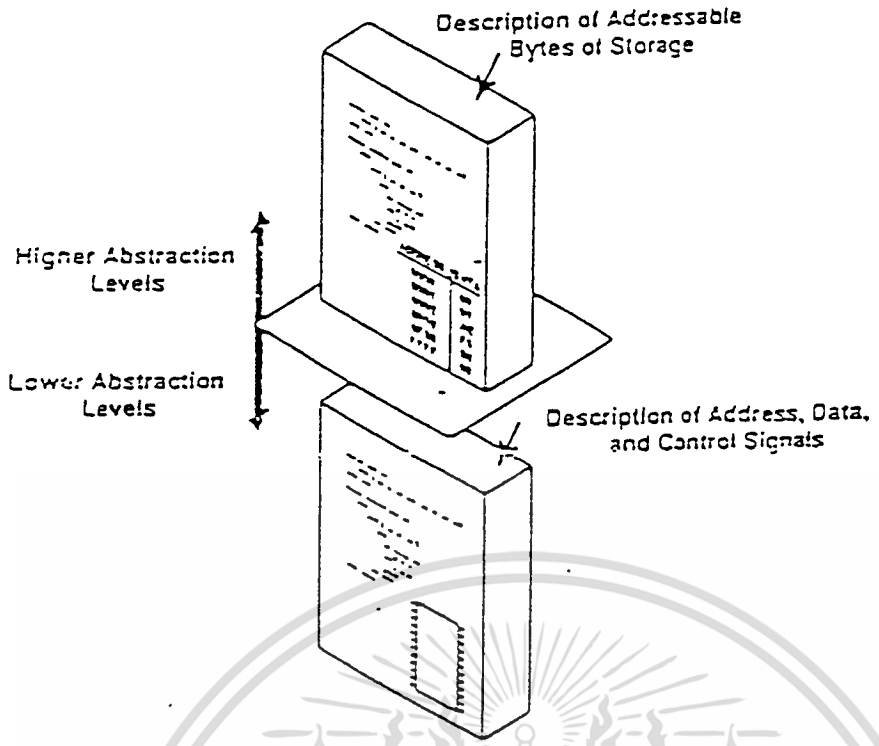
3.3.3Abstraction

คำนิยามของรูปแบบจะอธิบายการทำงานของตัวรูปแบบ มากกว่าที่จะอธิบายถึงว่าจะพัฒนาตัว รูปแบบนั้นอย่างไร หลักการนี้จะมีความสัมพันธ์อย่างใกล้ชิดกับหลักการ Modularity ในรูปที่3.3 Flip-Flop เป็นนิยามในการใช้ NAND เกท และ Shifter นิยามการใช้ฟลิปฟลอป



รูปที่ 3.3 การแบ่งแบบ Hierarchy ของ VHDL Shifter Description

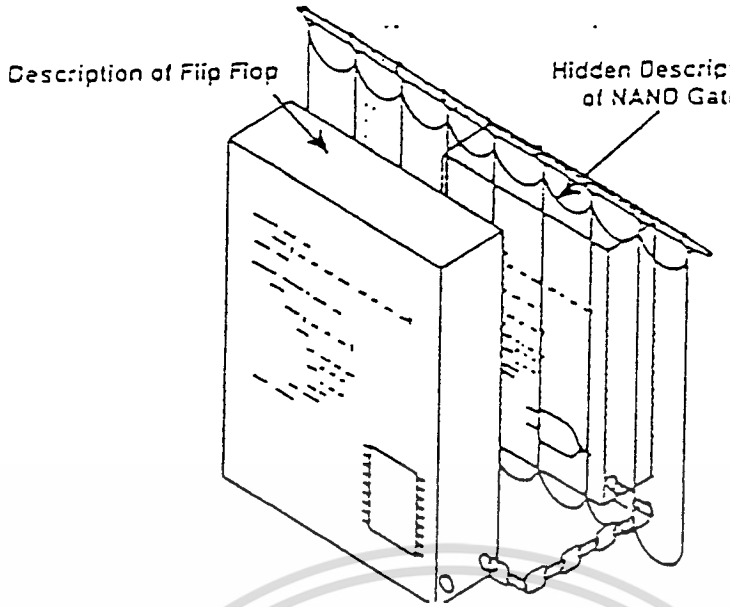
รูปที่ 3.4 แสดงอีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของรูปแบบโดยใช้ VHDL ในหลาย ๆ ระดับของการนิยาม ROM (Read Only Memory) อธิบายโดยใช้ภาษาระดับสูง ไฮเลเวล แสดงถึงสัญญาณควบคุมต่างๆ ซึ่งเก็บข้อมูลไว้ในตำแหน่งนั้น ๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Lin , Data Line หรือ Control Line เราสามารถเพ่งจิดสนใจไปที่ขนาดของข้อมูง โดยไม่ต้องคิดถึงสัญญาณควบคุมต่าง ๆ มากมายภายใน เพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ถ้าต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรขึ้นมาแก้ไขในระดับที่สูงขึ้นมา (High Level) จะทำให้ง่ายกว่าในการที่จะควบคุมสัญญาณภายใน ซึ่งเราจะเห็นว่าในแต่ละระดับมีความเหมาะสมแตกต่างกันไป และตรงจุดนี้เองทำให้รูปแบบที่เราออกแบบง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction



รูปที่ 3.4 Applying Absraction to a ROM Discription

3.3.4 Information Hiding

เมื่อเราทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์แวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่จะซ่อนรายละเอียดการพัฒนา Module นั้น ๆ โดยไม่ต้องการให้ส่วน Module อื่น ๆ รู้การทำงานภายใน Information Hiding มีประโยชน์คือ ทำให้รูปแบบภาษาวีเอชดีแอลนั้นสามารถจัดการได้ง่าย และสามารถอ่านและทำความเข้าใจได้ง่ายกว่า หลักการนี้จะใช้สนับสนุนหลักการ Abstraction คือจะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่ารูปแบบนั้นจะถูกสร้างขึ้นมาอย่างไร มีวงจรอย่างไรบ้าง เป็นต้น การซ่อนรายละเอียดภายใน Module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ในรูปที่ 3.3 คำอธิบายของ NAND เกทนั้นจะถูกปิดบังเอาไว้จากคนที่เขียนอธิบายฟลิปฟลอป ดูรูปที่ 3.5 คนที่เขียนอธิบายการทำงานของฟลิปฟลอป ไม่ต้องสนใจเลยว่า NAND เกท จะทำงานอย่างไร จะต่อกันภายในอย่างไร โดย NAND เกท สามารถเขียนขึ้นมาแล้วคอมไพล์เก็บไว้ใน Library ผู้ที่ออกแบบฟลิปฟลอประดับสูงขึ้นมาเพียงแต่ต้องรู้ว่าจะเชื่อมต่ออินพุท/เอาท์พุทของ NAND เกท มาใช้งานได้อย่างไรโดยไม่ต้องสนใจว่า NAND เกท จะถูกสร้างและพัฒนามาอย่างไร แบะประโยชน์อีกอย่างของ Information Hiding ก็คือป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย วีเอชดีแอลโมเดลไปยังที่อื่น ๆ โดยเป็นการแจกจ่าย อาจส่งไปแค่ภาษาวีเอชดีแอลที่คอมไพล์แล้ว ไม่ต้องส่งตัวซ้อสโค้ดไปทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในอีกระดับหนึ่ง



รูปที่ 3.5 การซ่อนรายละเอียดที่ไม่จำเป็นของระดับ NAND เกท

3.3.5 Uniformility

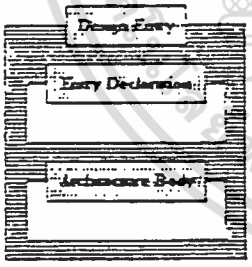
Uniformility เป็นหลักการอีกอย่างที่ช่วยในการอธิยายฮาร์แวร์ด้วยภาษาวีเอชดีแอลหมายถึงการสร้าง Module ของรหัสในลักษณะคล้ายกันโดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียน รหัสที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้คำอธิบาย (Comment) เป็นต้น ทำให้การพัฒนา Module อ่านและทำการเข้าใจง่าย

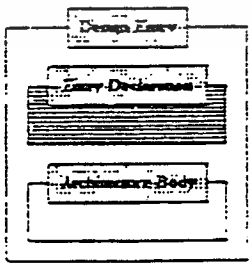
3.4 รูปแบบพื้นฐานของภาษาวีเอชดีแอล (Primary Language Abstraction VHDL)

ระหว่างการออกแบบนักออกแบบมักจะพยายามย่อยแตกวงจรออกเป็นส่วนย่อย ๆ เพื่อที่จะจัดการได้ง่ายขึ้นภาษาวีเอชดีแอล สนับสนุนการแยกส่วนฮาร์ดแวร์ และทำให้ง่ายที่จะเขียนพฤติกรรมการทำงานของแต่ละส่วนย่อย ๆ นั้น บางที Unit ย่อย ๆ สามารถใช้ได้ในหลาย ๆ ส่วนของวงจรที่เราจะออกแบบ หรือแม้กระทั่งนำไปใช้ได้ในรูปแบบอื่น ๆ รูปแบบพื้นฐานของภาษาวีเอชดีแอล ในการสร้าง Hardware โมเดล ก็คือ Design Entity ซึ่งสามารถแทน เซล , ชิป , บอร์ด หรือ ระบบย่อย (Subsystem) ได้

Design Entity ประกอบด้วย 2ส่วนใหญ่คือ ส่วนของ Entity Declaration และส่วนของ Architecture Body

Entity Declaration และ Architecture Body เป็น 2 ส่วนหลักที่สำคัญของภาษา VHDL Language Library คือส่วนของ Hardware Description หรือโมเดลซึ่งสามารถจะอยู่ใน Design File ใด ๆ หรือถูกคอมไพล์อยู่ใน Design File หลาย ๆ File ที่แยกจากกัน ความสามารถอันนี้ทำให้นักออกแบบสามารถจัดวางให้เป็น Module เขียนอธิบายแต่ละ Module จากนั้นก็คอมไพล์แต่ละ Entity แยกจากกัน โดยมี Architecture Body ของแต่ละ Entity ที่แตกต่างกันออกไป การประกาศ Entity Declaration เป็นการกำหนดการเชื่อมต่อ (Interface) ระหว่าง Design Entity กับวงจรส่วนอื่น ๆ ภายนอก โครงสร้างของ Entity Declaration แสดงตัวอย่างต่อไปนี้

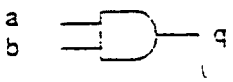




```
Entity indentifier is
        Entity_header
        --(generic and/or port clause)
        Entity_declarative_part
        --(Declarations for subprograms
        ,type,signal,..)
begin
        Entity_Statement_part
end identifier;
```

Entity Identifier คือชื่อของ Entity ที่เรากำหนดขึ้น แต่ละ Design Entity รับข้อมูลจากภายนอกโดย Port Mode In และส่งข้อมูลออกไปภายนอกโดย Port Mode Out ดูรูปที่ 2.7

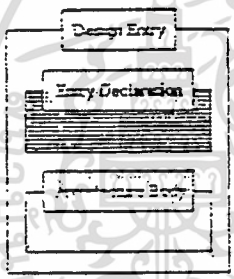
ตัวอย่างดังต่อไปนี้แสดง Entity Declaration (รวม Port Claus ด้วย) ของ AND เกท 2 Input




```
ENTITY and2 Is
        PORT (a,b: IN bit,
              c: OUT bit);
END and2;
```

รูปที่ 3.6 แสดงพอร์ท In และ Out ของ AND เกท

Architecture Body อธิบายความสัมพันธ์ระหว่างอินพุทและเอาท์พุทของ Design Entity โครงสร้างของ Architecture แสดงดังต่อไปนี้

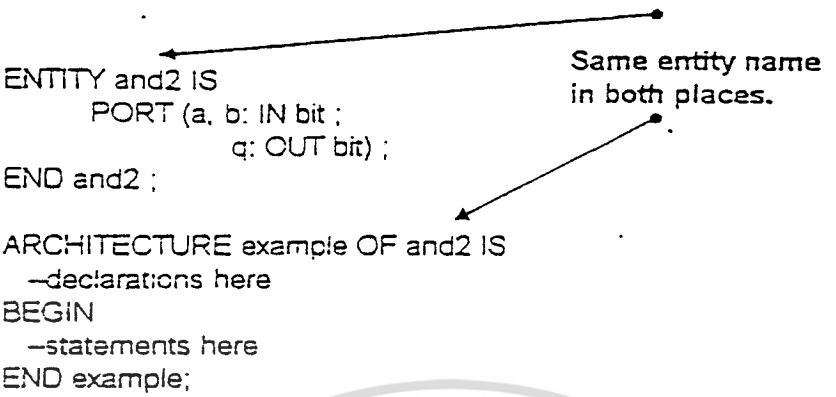


```
Architecture identifier of Entity_name is
        Architecture_declarative_part
begin
        Architecture_statement_part
end identifier;
```

Identifier และ Entity name คือ words ที่คุณจะต้องเขียนไว้ใน ภาษาวีเอชดีแอลโค้ด สิ่งสำคัญที่จะต้องคำนึงถึงก็คือ ชื่อของ Entity name ใน architecture Body จะต้องสัมพันธ์กัน ดังแสดงในรูปที่ 3.7

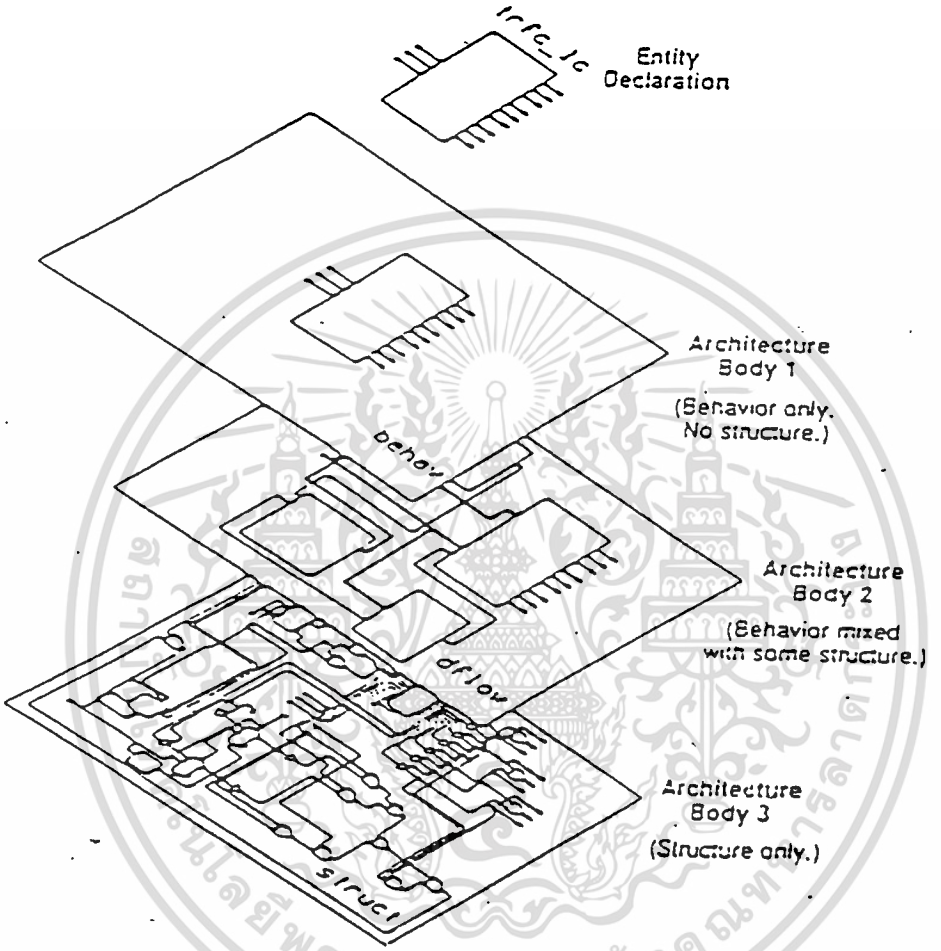
นักออกแบบกำหนดพฤติกรรมการทำงานหรือโครงสร้างของ Design Entity ใน Architecture Body โดยใช้วิธีการเขียนอธิบายโดยใช้ภาษาวีเอชดีแอล (Design Description Methods) ดังรูปที่ 3.8 และDesign Entity ซึ่งมี Architecture Body มากกว่า 1 Architecture โดยนักออกแบบจะต้องเขียน Entity Declaration (ชื่อควรเป็น "Trfc_lc") แล้วก็คอมไพล์ หลังจากนั้นจึงเขียนและคอมไพล์ส่วนของการบรรยายแบบบีเฮฟวีเออร์ของวงจรซึ่ง Architecture name ควรเป็น "Behav" ดังแสดงที่มุมล่างด้านขวาของ Architecture Body 1 เมื่อพอใจกับการทำงาน บีเฮฟวีเออร์ของวงจรที่ออกแบบแล้ว (ที่ระดับ High-Abstraction) เราสามารถที่จะเขียน Architecture Body ขึ้นมาอีกเพื่อเป็นการทดสอบ การทำงานของวงจร ณ ที่ระดับ Low-Abstraction Architecture 2 รูปที่ 3.8 แสดงโครงสร้างและรายละเอียดการไหลของข้อมูล (Data

Flow) ซึ่งมี Architecture name คือ "Dflow" หลังจากนั้นเราจึงทำการ Simulate Design ที่ระดับนี้จากนั้นทำการแก้ไขปรับปรุงจนได้ผลการทำงานที่น่าพอใจ



รูปที่ 3.7 การใช้ชื่อ Entity ใน Entity Declaration และ Architecture Body

ระดับการเขียนอธิบายด้วยภาษาวีเอชดีแอล ขั้นต่ำสุด (Lowest Abstraction Level) คือ การเขียนในแบบโครงสร้าง Structural Description ซึ่งเป็นการอธิบายการทำงานของวงจรระดับ Component Level Architecture Body 3 รูปที่ 3.8 แสดงให้เห็นถึงการอธิบายในระดับ Structural ชื่อ Architecture นี้คือ "Struct" ในการใช้วิธีการอธิบายที่แตกต่างกัน 3วิธีนี้ ทำให้เราสามารถพัฒนาออกแบบ Design 1 ตัว โดยใช้วิธีการ Top-Down Design ในแต่ละ Abstraction Level จะถูกเขียนและเก็บไว้ใน Design File แยกจากกัน



รูปที่ 3.8 หลาย Architecture Body สำหรับหนึ่ง Entity Declaration

3.5 วิธีการเขียนอธิบายในรูปแบบต่าง ๆ (Design Description Methodes)

ภาษาวีเอชดีแอล เป็นวิธีการเขียนอธิบายการทำงานของฮาร์แวร์ ในลักษณะของ Texual Format แทนที่จะใช้ Schemanc Diagram เหมือนเมื่อก้อน หัวข้อต่าง ๆ ดังนี้คือ วิธีการเขียนอธิบายภาษาวีเอชดีแอลในหลาย ๆ วิธีเพื่อที่จะอธิบาย Hardware Architecture

- Structural Description Method อธิยายตัวรูปแบบ ในรูปแบบของการเชื่อมต่อ Component ต่าง ๆ เข้าด้วยกัน

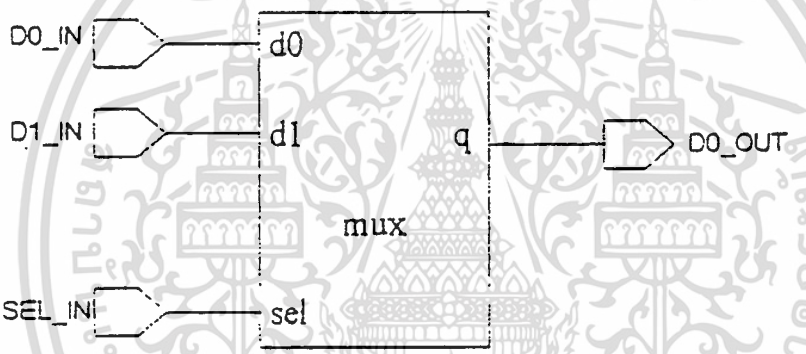
- วิธีการบรรยายแบบบีเฮฟวีเออร์อธิบายฟังก์ชันการทำงานของรูปแบบฮาร์ดแวร์ ในรูปแบบของ Circuit , Signal ที่ตอบสนองกับสัญญาณที่รับเข้ามาจากภายนอก พฤติกรรมการทำงาน (Hardware Behavior) จะถูกอธิบายด้วย Algorithm โดยไม่ว่าจะสร้างขึ้นมาอย่างไร

- Data Flow Description Method มีความคล้ายคลึงกับ Register-Tramfer Language วิธีการนี้อธิบายฟังก์ชันการทำงานของรูปแบบ โดยการทำหนดการไหล (Flow) ของข้อมูลจากอินพุท หรือ รีจิสเตอร์ ไปยังตัวเอาท์พุท หรือ รีจิสเตอร์ อีกตัววิธีการทั้ง 3 แบบที่ใช้อธิยาย Architecture ของ Hardware สามารถอธิบายร่วมกันโดยใช้ทั้ง 3 แบบต่อ 1 รูปแบบก็ได้

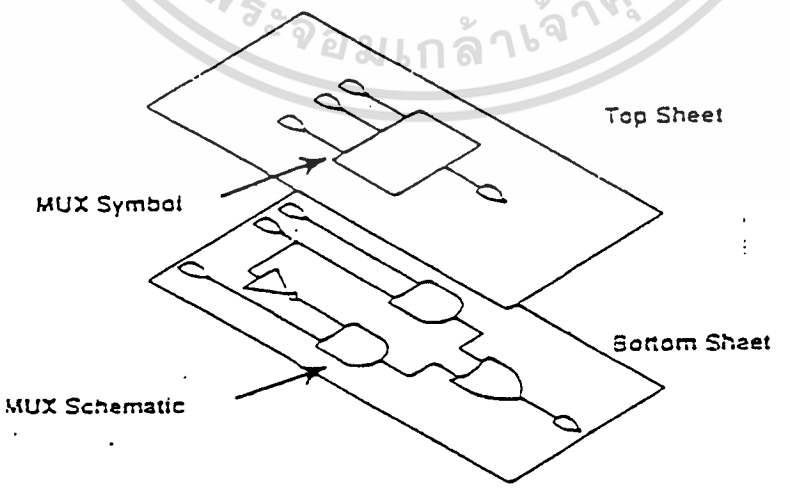
3.5.1 การอธิบายโดยใช้แบบ Structural

ในส่วนนี้จะอธิบายในส่วนของภาษาเพียงบางส่วนในการที่จะแสดง VHDL Structural Description โดยใช้ตัวอย่างคือ มัลติเพล็กเซอร์ 2 อินพุท จะไม่ได้อธิบายโครงสร้างของภาษาในส่วนนี้ทั้งหมด เพียงต้องการยกตัวอย่างให้เห็นเท่านั้น รายละเอียดทั้งหมดให้ดูได้ที่ VHDL Reference Manual

VHDL Structural Description เป็นวิธีการอธิบายตัวรูปแบบฮาร์ดแวร์ ที่คล้ายกับแสดงโดยใช้ Schematic diagram เพราะว่าแสดงให้เห็นถึงการเชื่อมต่อของส่วนประกอบ ตัวอย่างต่าง ๆ ที่จะแสดงให้เห็นในส่วนต่อไปนี้ เป็นการเปรียบเทียบวงจรง่าย ๆ 1วงจร ซึ่งแทนด้วย VHDL และแทนด้วย Schematic Diagram ว่าเป็นอย่างไร

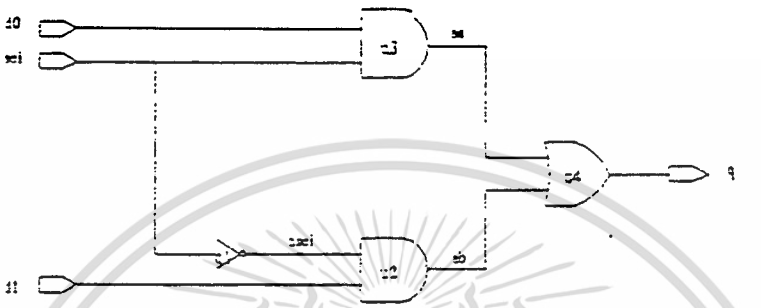


รูปที่ 3.9 สัญลักษณ์แสดงสองอินพุทมัลติเพลกเซอร์



รูปที่ 3.10 การออกแบบ Hierachy ใน Schematic Editor ของมัลติเพล็กเซอร์

รูปที่ 3.9 แสดงสัญลักษณ์ของ มัลติเพล็กเซอร์ 2 อินพุท วงจร MUX นี้เป็นการออกแบบในลักษณะของ Hierarchical Design (รูปที่ 3.10) ซึ่งวงจรในระดับล่างสุดเป็น Schematic Diagram หรืออยู่ในรูปแบบของการอธิบายถึงการเชื่อมต่อภายใน ดังรูปที่ 3.11 (หมายเหตุ จะสังเกตเห็นว่าชื่อ Pin ใน MUX Symbol ในรูปที่ 3.9 จะตรงกับชื่อ Net ของอินพุท/เอาท์พุทของ Schematic ในรูปที่ 3.11



รูปที่ 3.11 แสดงระดับเกทของสองอินพุทมัลติเพลกเซอร์

รูปที่ 3.12 แสดง VHDL Structural Description ของมัลติเพล็กเซอร์ 2 อินพุท โดย VHDL Code สามารถเขียนส่วนประกอบ โดยการเขียน Double Dash (--) ข้อความหรืออักษรใด ๆ ที่อยู่หลังจากเครื่องหมายจะถูกมองว่าเป็น Comment ไม่สนใจโดย Compiler (บรรทัด 1,2,5,7,17,19 ถึง 21,24 ในรูปที่ 3.12) Comment ทำให้รหัสอ่านง่าย

```
1 ENTITY  mux  IS        -- Entity  Declaration
2 PORT (d0,d1,sel :IN bit ; q :OUT bit );  --Port  Clause
3 END  mux;
4
5                          -- Architecture  Body
6 ARCHITECTURE  struct OF  mux  IS
7 COMPONENT  and2
8 PORT  (A,B   : IN bit; c: OUT bit);
9 END  COMPONENT;
10 COMPONENT  or2
11 PORT  (a,b : IN bit; c : OUT bit);
12 END  COMPONENT;
13 COMPONENT  inv
```

```
14 PORT  (a : IN bit; c : OUT bit);
15 END  COMPONENT;
16
17 SIGNAL  aa,ab,nsel : bit;      --Signal  Declaration
18
19 FOR  u1 :inv  USE  ENTNTY  WORK inv t(behav);  --config.
20 FOR u2,u3 : and2 USE  ENTITY  WORK and_gt(dFlow);  --specif.
21 FOR u4 : or2 USE ENTITY  WORK  or_gt(archl); --
22
23 BEGIN
24        u1 : inv  PORT  MAP (sel,nsel);  -- Architecture  Statement  Part
25        u2 : and2  PORT  MAP (nsel,d1,ab);
26        u3 : amd2 PORT  MAP (d0,sel,aa);
27        u4 : or2 PORT  MAP (zz,zb,q);
28 END  struct;
```

รูปที่ 3.12 ตัวอย่างโปรแกรมของ Structural Description สำหรับมัลติเพลกเซอร์

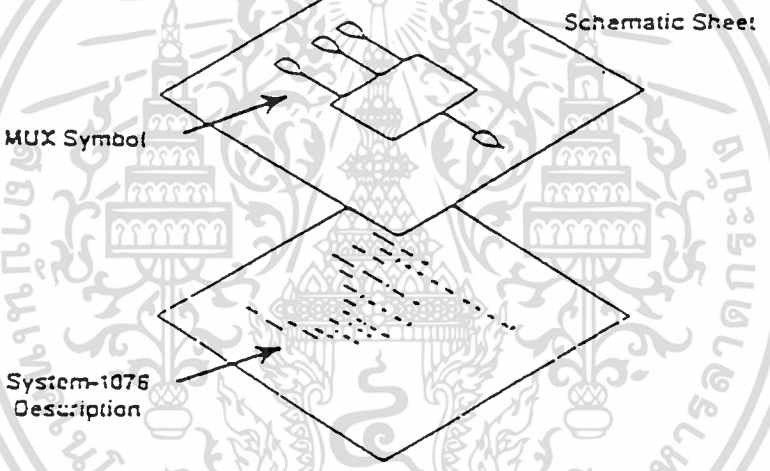
มัลติเพล็กเซอร์ 2 อินพุท แสดงในรูปที่ 3.12 เป็นวงจรพื้นฐาน Emtity Declaration อยู่ที่ด้านบน บรรทัด 1 ถึง 3 กำหนดการ Interface Design Entity และสภาพแวดล้อม หรือวงจรอื่น ๆ ภายนอก

Entity Declaration มี Port Clause ซึ่งบอกช่องสัญญาณอินพุท (Input Channels d0,d1 และ set) และช่องสัญญาณเอาท์พุท (Output Channels) (q) สัญญาณของแต่ละ Channel ถูกกำหนดให้เป็น บิท คือมีสถานะเป็น 0 กับ 1 ซึ่ง Entity Declaration นี้สามารถเปรียบเทียบกับ MUX Symbol ใน Schematic รูปที่ 3.10ได้

Architecture Body ในรูปที่ 3.12 (บรรทัดที่6 ถึง 28) อธิบายความสัมพันธ์ระหว่าง Design Entity Input และ Output ในลักษณะของโครงสร้าง Architecture นี้สามารถทำงานได้เหมือนกับ schematic ดังรูปที่ 3.10 Component หลาย ๆ ตัว (AND2 , OR , INV) ที่ประกอบ / รวมกันจนเป็น MUX Design Entity ในรูปที่ 3.12 ถูกประกาศไว้ในส่วนของ Architecture Declaration (บรรทัด 7 ถึง 15) Signal (aa , bb , nset) ถูกประกาศไว้ใน Architecture Body (บรรทัดที่ 17) ด้วยเช่นกัน เพื่อที่จะแทน เอาท์พุทของ AND เกท 2 ตัว (U2 , U3) และ Inverter (U1)

Configuration Specification ในบรรทัดที่ 19 ถึง 21 เชื่อมเอาส่วนประกอบแต่ละตัวที่ประกาศไว้เข้ามายัง Design Entity เพื่อบอก Design Entity ว่าส่วนประกอบแต่ละตัวทำงานอย่างไร ยกตัวอย่างเช่น ส่วนประกอบ U1 ในบรรทัดที่ 24 รูปที่ 3.12 ถูกกระโดดมายัง Architecture Body ที่ชื่อ "Behav" สำหรับ Design Entity ที่เรียกว่า Invrt

Architecture Statement Part (บรรทัดที่ 24 ถึง 27) อธิบายการเชื่อมต่อ (Donnection) ระหว่างส่วนประกอบที่ประกอบด้วย Design Unit ในส่วนนี้จะมีการประกาศการใช้ Component รูปที่ 3.13 แสดงให้เห็นว่า Svhematin Sheet ที่มี MUX Symbol มีความเกี่ยวข้องกับ VHDL Structural Description อย่างไร แทนที่จะใช้การลากเส้นใน Schematic Sheet VHDL Structure Description กำหนดการเชื่อมต่อภายในของส่วนประกอบ



รูปที่ 3.13 สองอินพุทมัลติเพลกเซอร์ซึ่งมี Structural Description ที่เกี่ยวข้องกัน

3.5.2 การอธิบายและการใช้ Behavirol

ในส่วนนี้จะเป็นการอธิบายถึงส่วนสำคัญของภาษาส่วนหนึ่ง นั้นคือ Behavioral Description โดยใช้วงจรซึ่งได้ยกตัวอย่างมาแล้วก่อนหน้านี้ คือ MUX และ 4 BIT SHIFTER หลังจากอ่านส่วนที่ได้อธิบายไปก่อนหน้านี้ คือ Stuctual Description เราสามารถเปรียบเทียบกันได้ระหว่างการอธิบายด้วย Behavioral Description กับ Structural Methrod ว่าแตกต่างกันอย่างไร

VHDL Behavioral description แทนฟังก์ชันการทำงานของรูปแบบ ในรูปแบบของวงจรและสัญญาณที่ตอบสนองต่อการกระตุ้นจากภายนอก แสดงในรูปที่3.10 ถึง 3.13พฤติกรรมการทำงานของ MUX ถูกกำหนดด้วยการเชื่อมต่อระหว่าง Inverter , AND gate และ OR gate ซึ่งฟังก์ชันของเกทแต่ละตัวนี้เป็นที่เข้าใจกันดีอยู่แล้ว ในรูปแบบที่มีความซับซ้อนมากขึ้นส่วนประกอบ U1 ถึง U5 ในรูปที่3.12 ต้องสามารถแทนด้วย Entity ที่มีฟังก์ชันการทำงานแต่ละส่วนประกอบ ด้วย Behavioral Description

VHDL Description แสดงในรูปที่ 3.13 แสดง Behavioral Description แทนที่จะเป็นแบบ Strcture Description เหมือนหัวข้อก่อนหน้านี้ ในครั้งนี้เราสามารถวาง MUX Symbol ลงใน Schematic Sheet แต่ในที่เราใช้ Behavioral ของส่วนประกอบระหว่างการทำ Circuit ซิมูลเลชัน รูปที่3.14 แสดงภาษาวีเอชดีแอลโค้ด ซึ่งกำหนดการทำงานของ MUX ในรูปแบบ Behavioral

Behavioral Description ในรูปที่ 3.14 และ Structal Description ในรูปที่3.12ทั้งคู่มี Entity Description และ Architeture Body เช่นกัน ในทางปฏิบัติเราไม่จำเป็นจะต้องเขียน 2Architecture Body ไว้ในไฟล์เดียวกัน (ซึ่งสามารถทำได้) เราควรเขียน Entity Declaration ไว้ไฟล์หนึ่ง แล้วเขียน Behaviroral Description ไว้อีก ไฟล์ หนึ่ง และ Structural Description ไว้อีกไฟล์เช่นกัน ในการออกแบบจริงๆ หลังจากการเขียน และ คอมไฟล์ Entity Description เสร็จเรียบร้อยแล้ว ขั้นตอนต่อไปควรเขียน Behavioral Architecture เป็นขั้นตอนต่อไปในการที่จะทดสอบการทำงานของวงจรโดยรวมทั้งหมด หลังจากนั้นเราจึงทำการเขียน Simulte และ Refine ฟังก์ชันการทำงานของโมเดลจนกว่าจะทำงานได้ถูกต้อง จึงจะทำการเขียน Structural Architecture จากนั้นก็เปลียน Sructrual Description เข้าไปแทนที่ Behavioral Description เพื่อที่จะทำการ Simulate ดูอีกครั้ง

```
1  ENTITY mux IS              -Entity Declaration
2 PORT (do,d1,set :IN bit,q:OUT bit );   --Port Clause
3END mux;
4
5
6BEGIN
7 fl :                        --Process Statement
8 PROCESS (d0,d1,set)                    --Sensitivity Part
```

```
9 BEGIN
10 IF sel = ' 0 ' THEN          --Process Statement Part
11q <= d1;
12 ELSE
13 q  <=  d0;
14 END IF;
15 END PROCESS fl ;
16 END behav;
```

รูปที่3.14 ตัวอย่างไปรแกรมของ Behavioral Description สำหรับมัลติเพลกเซอร์

Behavvior Description Mode มีประโยชน์ต่อการกำเหนิดอินพุทขึ้นมาทดสอบ VHDL โมเดล ในส่วนอื่นๆ เมื่อต้องการทำซิมูเลชัน เช่นเราต้องการออกแบบ Traffic Light Controller โดยใช้ Structual Description และเราต้องการทดสอบโมเดลนั้น โดยที่ Traffic Light Contoller นั้น อินพุทจะรับจาก Sensor ที่ต่อเข้ามา สำหรับการซิมูเลชัน เราต้องเขียน Behavior Description โมเดล ขึ้นมาเพื่อทำจำลองสัญญาณที่ออกจาก Sensor (แทนการสร้างจริงๆ)

ข้อแตกต่างที่เห็นได้ชัดระหว่าง Structural และ Behavioral Description ของ MUX คือ Architecture Body ดังรูปที่ 3.14 มี Process Statement ซึ่งแทนที่การทำงานที่เป็นอิสระกำหนดพฤติกรรมการทำงานของฮาร์ดแวร์ หรือส่วนใดส่วนหนึ่งของรูปแบบ รูปแบบของการเขียน Process Statement แสดงดังต่อไปนี้

```
Process Statement ..............label
        Process (sensitvity_list )
                Process_decrarative_part
        begin
                Process_Statement_part
        end  Process label;
```

Process statement ในรูปที่ 3.14 Process Label fl ตามด้วยเครื่องหมาย Colon ; (บรรทัดที่ 7) Process Label เป็น Optional แต่มีประโยชน์ในการที่จะช่วยแยกให้เห็นความแตกต่าง Process หลายตัวให้รูปแบบใหญ่ ๆ

ในวงเล็บที่ต่อจาก " Process " คือ Option Sensitivity List ดังรูปที่ 3.14 (บรรทัด ที่ 8) ประกอบไปด้วยสัญญาณ (d0,d1,set) ระหว่างการทำซิมูเลต ถ้าสัญญาณใดสัญญาณหนึ่งใน Sensitivity List เกิดการเปลี่ยนแปลง State ,Process จะทำการ Execute และ State ของเอาท์พุท ก็

จะเป็นไปตามเช่นกัน แต่ละ Process ในการออกแบบวีเอชดีแอล จะ Execute 1 ครั้งระหว่างการทำงาน Initialize VHDL Hardware โมเดล

หัวใจสำคัญของ Process Statement ในรูปที่ 3.14 คือ If Statement ที่อยู่ภายใน Process Statement Part รูปแบบพื้นฐานของการเขียน If statement แสดงดังข้างล่าง

```
If Statement ..... if condition then
                        sequence_of_statements
                   else if condition then
                        sequece_of_Statements
                        else
                        sequence_of_Statements
                   end if;
```

If Statement ในภาษาวีเอชดีแอล จะถูกดีความคล้ายๆกับประโยคในภาษาอังกฤษ ยกตัวอย่างจากประโยคข้างล่าง

If the traffic light isgreen then process across the intersection or else (if the traafic light is not green) remain stopped.

ในประโยคนี้จะอยู่ในสภาวะการทำงานก็ต่อเมื่อไฟเป็นสีเขียวก่อนที่คำสั่งต่าง ๆจะถูก Execute "else " Statement จะเป็นทางเลือกอีกทางหนึ่ง เมื่อสถานะอื่นที่ไฟไม่เป็นสีเขียวตามเงื่อนไข

If Statement ในรูปที่3.15 (บรรรทัดที่ 10 ถึง 14) สามารถเขียนได้ใหม่ดังรูปต่อไปนี้

if Signal sel (select) isequal to 0, then assign the value of the Wavefrom on Signal d1 to target Signal q or else assign the value of the Wavefrom on Signal d0 to target Signal q.

เมื่อใดก็ตามที่สภาวะภายใน if condition หรือ else condition ในตัวอย่าง ได้รับสภาพตรงตามเงื่อนไข(sel = '0' หรือสภาวะตรงข้ามของ sel < > '0') target Signal q จะถูก Modified ตามความเหมาะสม ซึ่งขึ้นอยู่กับ Signal Assignment Statement รูปแบบพื้นฐานของ Signal Assignment มีดังนี้

Signal Assignment State : target (= transport Wavefrom)

(Note transport Optional)

Signal Assignment Statement ประโยคแรกในรูปที่ 3.15 คือ

q < = d1

ประโยคนี้จะทำการกำหนดสัญญาณ d1 ให้แก่สัญญาณ q (Optional transport) ไม่ได้ใช้ในตัวอย่างนี้ Singal Assignment delimeter ประกอบด้วยตัวอักษรพิเศษ 2 ตัวคือ บางครั้งเรียกว่า compound delimeter ประโยค Signal Assignment Statement อันที่สองคือ

q < = d0;

จะทำการ Assign Wavegorm ให้กับสัญญาณ q การใช้งานอีกอย่างของ compound delimeter คือใช้เป็น relational operater "น้อยกว่าหรือเท่ากับ " เช่นประโยคตัวอย่าวดังนี้

If Z < = '1 ' Then

สรุปว่า Signal Assignment delimeter < = ใช้สำหรับ Assign ค่าที่อยู่ทางขวามือให้กับสัญญาณที่อยู่ทางด้านซ้ายมือ และ delimeter ตัวเดียวกันนั้นก็ใช้ในการทำ relational opertor เปรียบเทียบ (น้อยกว่าหรือเท่ากับ) ใช้ในการเปรียบเทียบ condition ใน If statement ตัวอย่างในรูปที่ 3.15 แสดงการเขียน VHDL Behavioral description ของ 4 bit shifter เพื่อแสดงให้เห็นว่า accurate และ succinet ของ VHDL Description เป็นอย่างไรเมื่อเทียบกับรูปแบบของ textual description ดังนี้

The four bit shifter has four input data line and two control line. When both control line 1 are low , the input level are passed directly to the coorresponding output . When control line 0 is high and control line is low , output line 0 is low ; input line 0 is passecd to output line 1; input line 1 is passed to output line 2; and input line 2 is passed to output line 3.

When control line 0 is low and control line 1 is high; input line 2 is passed to output line 1 ;input line 3 is passed to output line 2 ; and output line 3 is low . When both control lines are high , input line 0 is passed to both output line 0 and line 1; input line 1 is passed to output line 2; and input line 2 is passed is passed to output line 3.

```
1 ENTTTY Shifter is          --Entity Declaration
2 PORT  (shfin      : IN bit _ vector 0  to 3 );     -- Port Clause
3         shfout    : OUT bit _ vector ( 0 to 3);
4         shfcrl    : IN bit _ vector (0 to 1 );
5 END Shifter ;
6
7 ARCHITECTURE behav OF Shifer is          -- Architecture Body
8 BEGIN
9         f2 :          -- Process Declaranon Port
12 BEGIN
13 CASE shiftctl IS
14        WHEN "00"  => shifted  = shftin ;
15        WHEN" 01"  => shifted  =  shtin (1 to 3 ) & '0';
16        WHEN "10"  => shited   = '0' & shftin ( 0 to 2 );
```

```
17 WHEN "11"  => shifted  = shftin (0) & shftin (0 to 2);
18 END CASE;
19 shftout <= shifted AFTER 10 ns;
20END PROCESS f2;
21END behav;
```

รูปที่ 3.15 ตัวอย่างโปรแกรมของ Behavioral Description สำหรับ Shifter

3.5.3 Stuctuaral and Behavioral Description Surnmary

สรุป Strustural และ Behavioral Description ได้ดังนี้

VHDL Structural Description เป็นการกำหนดการเชื่อมต่อส่วนประกอบต่างๆ ในวงจร และการที่เราสร้างขึ้น ส่วน Behavioral Description เป็นการอธิบายถึง Algrorithm ของวงจร และการทำงานของ Input /Output ภายในว่ามีการตอบสนองอย่าวไรต่อสัญญาณภายนอก Design Entity โดยใช้ Entity Declaration ส่วน Architure Body ของแต่ละ Entity มีไว้เพื่ออธิบายความสัมพันธ์ระหว่างอินพุทและเอาท์ ของ Entity นั้นๆ

Strutural และ Behavirol Description มีความแตกต่างกันอย่างไรเห็นได้ชัดเจนในส่วนของ Architecture Body ดังแสดงไว้ในรูปที่ 3.16 เปรียบเทียบตัวอย่าง MUX Architecture Body ของ Strutural Description (ส่วนบนของรูปที่3.16) มีส่วนประกอบ Architecture Statement Body Part ซึ่งอิธบายการเชื่อมต่อของ Componet ที่อยู่ใน Design Entity ส่วน Architecture Body ของ Behavioral Description (ส่วนล่างของรูปที่ 3.16) มีส่วนของ Process Statement ซึ่งอธิบายการทำงานของ Design Entity นั้นๆ

```
1 ENTITY mux IS   -- STRUSTURAL -------Entity Declaration
2 PORT ( d0,d1, sel: IN bitq:OUT bit );  --Port Clause
3 End mux;
4
5          --Architecture Body
6 ARCHITECTURE struct OF mux IS
7COMPONENT and 2
8PORT(a,b: IN bit;c:OUT bit );
9END COMPONENT;
10 COMPONENT or 2
11 PORT (a,b :IN bit;c:OUT bit);
12 END COMPONENT;
13COMPONENT inv
```

```
14PORT (a: IN bit;c :OUT bit );
15END COMPONENT ;
16
17 SIGN AL,aa,ab.nsel :bit;      --Signal Declaration
18
19 FOR u1 : inv USE ENTITY WORK__invrubehav:  --Config.
20 FOR u2, u3 and 2 USE ENTITY WORK__and__gu (dflow);   --Specif
21FOR u4: or 2 USE ENTITY WORK or ) gt (archl);  --
22
23 BEGIN .
24      U1 : inv PORT MAP (sel, nsel);    --Architecture Statement Part
25      U2 and2 PORT MAP (nsel, d1, ab);
26      U3 :and2 PORT MAP (d0,sel,an );
27      U3 :or2 PORT MAP (aa, ab.q);
28END struct;
```

```
1 ENTITY mux IS ----BEHAVIORAL------Entity Declaration
2PORT (d0,d1,sel :IN bit q ;OUT bit);   -- Port Clause
3END mux;
4
5 ARCHITECTUER behav OF mux IS
6 BEGIN
7       fl :            --Process  Statement
8       PROCESS (d0,d1,sel)        --Sensitivity List
9       BEGIN
10              IF sel = '0' THEN    --Process Statement Part
11              q <= d1;
12              ELSE
13              q <= d0;
14              END IF;
15      END  PROCESS fl;   .
16 END behav;
```

รูปที่ 3.16 **เปรียบเทียบตัวอย่างมัลติเพลกเซอร์แบบ** Strutural **และ** Behavioral Description

3.5.4 การอธิบายโดยใช้แบบ Data Flow

VHDL Data Flow Description และ Register Transfer Language มีความคล้ายคลึงกัน คือ ทั้งคู่อธิบายการทำงานของรูปแบบ โดยกำหนดการไหลของข้อมูลจากอินพุทหนึ่ง หรือ รีจีสเตอร์หนึ่ง ไปยังเอาท์พุทหรืออีกรีจีสเตอร์หนึ่ง Data Flow และ Behavioral Description มีความคล้ายครึงกันคือ ทั้งคู่ใช้จำนวน Process น้อยกว่าโดยภายในแต่ละ Process ใช้ลักษณะของซีเควนเชียล Signal Assignment หลายๆ คำสั่งภายใน Process ในทางตรงกันข้าม Data Flow Description ใช้จำนวนของคอนเคอร์เรนท์ Signal Assignment มาก คอนเคอร์เรนท์ Statement ใช้ใน Data Flow Description ประกอบด้วย

-Block Statement

-คอนเคอร์เรนท์ Procedure Call

-คอนเคอร์เรนท์ Assertation Statement

-คอนเคอร์เรนท์ Signal Assignment Statement

นอกจากนี้ Process Statement , Genetate Statement และ Component Instamtiation Statement เป็น คอนเคอร์เรนท์ Statement ด้วยเหมือนกัน ซึ่งโครงสร้างเหล่านี้ไม่คอยพบในการอธิบายแบบ Data Flow Descrition

คอนเคอร์เรนท์ Statement กำหนดการเชื่อมต่อ Process Blocks ซึ่งทั้งหมดรวมๆกันจะอธิบายการทำงานของ Design คอนเคอร์เรนท์ Statement ทำการ Execute แบบ Asynchronous ร่วมไปกับ คอนเคอร์เรนท์ Statement อื่นๆ

รูปที่ 3.17 แสดงให้เห็นรูปแบบการเขียนอธิบาย Mux โดยวิธี Data Flow Description ซึ่งก่อนหน้านี้ใช้ Behavioral และ Structural Description เขียนอธิบายมาแล้ว ตัวอย่างนี้ง่ายเกินไปที่จะแสดงให้เห็นถึงประโยชน์จริงๆของ Data Flow Description เพราะว่ามีความคล้ายคลึงกับ Behavioral Description ในรูปที่ 3.14 มาก โดยทั้งสองตัวอย่างใช้ Process Statement แสดงด้วยคอนเคอร์เรนท์ Statement ในรูปที่ 3.17 (บรรทัดที่ 8ถึง 10) เพื่อกำหนด Signal Behavior

```
1 ENTITY mux IS
2 PORT ( d0,d1,sel:IN bit; OUT bit);        ---Port Clause
3end  MUX;
4
5
6 ARCHITECTURE data_flow OF mux IS
7 BEGIN
8         cls :
9         q <= d1 WHEN sel = '0' ELSE       -- Condition Signal Assignment
```

```
10      d0;
11 End data_flow;
```

รูปที่ 3.17 ตัวอย่างโปรแกรมของ Data Flow Description ของมัลติเพลกเซอร์

Data Flow Description ประกอบไปด้วย Entity Declaration (บรรทัดที่ 1 ถึง 3) เช่นเดียวกันกับตัวอย่างใน Stuctural และ Behavioral Description ที่อธิบายมาแล้วก่อนหน้านี้ ส่วนของ Architecture Body ประกอบไปด้วย คอนเคอรเรนท์ Signal Assignment Statement ซึ่งทำหน้าที่เทียบเท่ากับ Process Statement (ซึ่งมีความหมายเหมือนกัน) รูปแบบของการเขียนคอนเคอรเรนท์ Signal Assignment แสดงดังต่อไปนี้

```
Concurrent Signal Label : Conditional_signal_assignment
Assignment Statement.....
        label: selected signal_assignment
```

ในรูปที่ 3.17 (บรรทัดที่ 9 และ 10)Conditional Signal Assignment ทำหน้าที่เป็น Signal Assignment (q < d1 หรือ q < d0) ขึ้นอยู่กับสถานะที่กำหนดใน Condition Waveform รูปแบบของการทำ Condition Waveform มีดังนี้

```
Conditional Signal      target  <= option cinditional_waveform;
Assignment
condition waveforms     waveform when condition else
                --;
                waveform when condition else
                waveform
```

Condition Signal Assigment แทนการทำงานของ Process Statement ที่ใช้ IF Statement ในการเปลี่ยนแปลงลักษณะของสัญญาณ (Optional Guarded Transport) ไม่ได้แสดงให้เห็นในตัวอย่าง เพื่อการเปรียบเทียบ Beharioral Description ของ 4 bit Shifter รูปที่ 3.15 แสดงให้เห็นให้เห็นอีกครั้งในรูปที่ 3.18 (ด้านบนของรูป) พร้อมทั้ง Data Flow Description ซึ่งอธิบาย 4 bit Shifter เช่นกัน (ด้านล่างรูป)

ข้อแตกต่างซึ่งเห็นได้ชัดระหว่าวการอธิบายโดยใช้ 2 วิธี คือ 4 Process Statement ถูกแสดงเป็นนัยๆ ใน Data Flow Descrition และ 4 Conditional Signal Assignment (บรรทัด 9 ถึง 20) ใน วิธี Behavirol Descrition มี 1 Process Statement ใช้อย่างเห็นได้ (บรรทัด 9 ถึง 20)

ตัวอย่างของ Data Flow Descrition ในรูปที่ 3.18 มีการใช้ Entity Declaration เช่นเดียวกับ Behaviral Descrition (บรรทัดที่ 1 ถึง 5) Architecture Body ใน Data Flow Descritionใช้ คอนเคอรเรนท์ Signal Assignment ซึ่งประกอยด้วย 4 Conditional Signal Assignment มีสำหรับแต่ Element

ของ Shiftout Array (คอนเคอรเรนท์ Signal Assignment Statement ไม่ได้ใช้ Option Label เหมือนกับที่ใช้ในรูปที่ 3.18 บรรทัดที่ 8)

```
1 ENTITY shifter IS   –BEHAVIORAL ————entity declaration
2 PORT (shfin: IN bit_vector 0 to 3 );    --Port Clause
3       shfout:OUT bit_vector 0 to 3 );
4       shftout :IN bit_vector 0 t0 1) ;
5 END shifter;
6
7 ARCHITECTURE behav OF shifter IS      --Architecture Body
8 BEGIN
9       f2:                 --Process Statement
10      PROCESS (shftin, shfted)
11      VARIABLE shifted :bit_vector 0 to 3)     Process Dec;aration Part
12      BEGIN
13              CASE shfted IS
14                      WHEN "00" => shftin:=shftin;
15                      WHEN "01" =>shftin :=shftin (1 to 3) &'0';
16                      WHEN "10" =>shftin:='0'&shftin (0 to 2);
17                      WHEN"11" =>shftin:=shftin (0) &shftin(0 to 2);
18              END CASE;
19              shftout <= shifted AFTER 10 ns;
20      END PROCESS f2;
21 END behav;
```

```
1ENTITY shfter IS—DATA FLOW————Entity Declaration
2PORT ( shftin : IN bit_vector 0 to 3);    --Port Clause
3       shftout :OUT bit_vector 0 to 3);
4       shfted : IN bit_vector 0 to 3 );
5END shfter;
6
7ARCHTECTUER data_flow OF shifter IS  -- Architecture Body
8BEGIN
9       shftout(3) <= '0' AFTER 10 ns WHEN shfted = "01" ELSE
```

```
10              shftin (3) AFTER 10 ns WHEN shfted = "00" ELSE
11              shftin (2) AFTER 10 ns ;  --End Cond. Sig..Assign.1
12      shftout(2) <= shftin (3) AFTER 10 ns WHEN shfted = "01" ELSE
13              shftout (2) AFTER 10 ns WHEN shfted ="00" ELSE
14              shftin (1) AFTER 10 ns ; --END Cond . Sig . Assign.2
15      shftout (1)<= shftin (2) AFTER 10 ns WHEN shfted ="01" ELSE
16              shftin(1) AFTER 10 ns WHEN shftout ="00" ELSE
17              shftin(0) AFTER 10 ns  ---End Cond. Sig .Assign.3
18      shftout (0) <= shftin(1) AFTER 10 ns WHEN shftout ="01" ELSE
19              '0' AFTER 10 ns WHEN shftout ="10" ELSE
20              shftin (0) AFTER 10ns ;   --End Cond. Sig. Assign.4
21End data_flow;
```

รูปที่3.18 เปรียบเทียบตัวอย่าง Shifter แบบ Behavioral และ Data Flow Description

3.6 สรุป

-ภาษาวีเอชดีแอลคือตัวภาษาที่ออกแบบมาเฉพาะเพื่อใช้อธิบายการทำงานของ Hardware ให้อยู่ในรูปที่สามารถอ่านทำความเข้าใจได้ สามารถอธิบายได้ถึงการจัดระบบและการทำงานของวงจรดิจิตอล วงจรระดับบอรด์ และ อุปกรณ์ต่างๆ

-เหตุผลที่ทำให้ภาษาวีเอชดีแอล ใช้ในการออกแบบและจำลองการทำงานของ Product ตัวหนึ่งซึ่งยังไม่สามารถสร้างจริงๆ เพื่อดูการทำงานก่อนลงมือสร้าง หรืออาจใช้เป็นตัวแทนของความคิด Product นั้นๆ มีดังนี้

1. ภาษาวีเอชดีแอล อนุญาติให้เราออกแบบ จำลองการทำงาน และทดสอบระบบโดยใช้รูปแบบของภาษาระดับสูงจนถึงระดับเกทเลเวล
2. ภาษาวีเอชดีแอล ถ้าเราเขียนตามรูปแบบของ VHDL Synthesis Guide จะทำให้เราสามารถใช้วีเอชดีแอลโค้ด นั้นไปทำการสร้างวงจรโดยใช้ VHDL Synthesis Tools

3 เพราะว่าภาษาวีเอชดีแอล เป็นภาษาที่กำหนดเป็นมาตราฐาน IEEE 1076 -1087 IEEE Standard VHDL Reference Manual วิศวกรหรือผู้ออกแบบสามารถใช้ภาษานี้ในการพัฒนาได้เหมือนกัน ลดปัญหาการเข้ากันไม่ได้ลงไป

-ภาษาวีเอชดีแอล มีคุณสมบัติที่ดีที่ทำให้เราสามารถเขียนและแก้ไขวงจร Digital ที่มีขนาดใหญ่และซับซ้อนได้อย่างสะดวกรวดเร็ว และมีประสิทธิภาพ ดังนี้

1 Top Down Design วิธีการนี้ให้เราสามารถอธิบายการทำงานของระบบได้ในลักษณะของ Block ใหญ่ๆ จาก นั้นทำการวิเคราะห์จำลองการทำงานและแก้ไขให้ได้คุณสมบัติตามที่เราต้องการ ณ ระดับ Block ก่อนที่จะทำการลึกลงไป

2 Modularity วิธีการที่แยกส่วน (หรือการประกอบส่วนย่อยๆ ขึ้นมา) วงจรที่เราออกแบบออกเป็นส่วนย่อยๆ เล็กๆ ออกมา

3.Abstraction รายละเอียด ใน Module ซึ่งอธิบายการทำงานของ Module มากกว่าที่จะอธิบายถึงการพัฒนา และการสร้าง Module นั้น

4.Information Hiding การพัฒนา Module ใหม่จาก Module อื่นๆ ที่เราสร้างขึ้นมาแล้ว

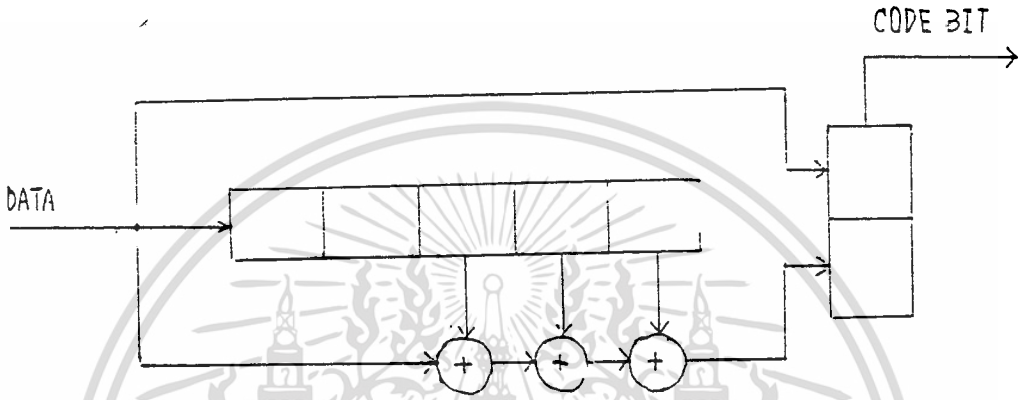
5. Uniformility สร้าง Module โดยใช้ตัวภาษา VHDL Building Blocks

บทที่4 วงจร convolution code ที่นำมาเขียนโปรแกรม

วงจรเข้ารหัสและถอดรหัสที่นำมาใช้ในการออกแบบ สามารถแสดงผลที่ได้จากการทดลองป้อนข้อมูลเพื่อเข้ารหัส และทดลองถอดรหัสดังนี้

-ส่วนของวงจรเข้ารหัส

จากรูปจะเป็นวงจรเข้ารหัสแบบ systematic convolutional code ซึ่งมีขนาด (12,6) และเป็นวงจรชนิด noncastatrophic ซึ่งจะไม่มีปัญหาในการถอดรหัส



รูปที่ 4.1 รูปวงจรเข้ารหัสแบบ convolutional code

จากวงจรข้างต้นเราสามารถเขียน generator polynomial ได้คือ g(x) = (100111) ซึ่งเป็นการเขียนในรูปแบบ binary และสามารถเขียนในรูปของเมตริกซ์ได้คือ

$$G = \begin{bmatrix} 11\ 00\ 00\ 01\ 01\ 01 \\ 00\ 11\ 00\ 00\ 01\ 01 \\ 00\ 00\ 11\ 00\ 00\ 01 \\ 00\ 00\ 00\ 11\ 00\ 00 \\ 00\ 00\ 00\ 00\ 11\ 00 \\ 00\ 00\ 00\ 00\ 00\ 11 \end{bmatrix}$$

และจะได้ Parity-check-matrix H(x) ดังนี้

$$H(x) = \begin{bmatrix} 11\ 00\ 00\ 00\ 00\ 00 \\ 00\ 11\ 00\ 00\ 00\ 00 \\ 00\ 00\ 11\ 00\ 00\ 00 \\ 10\ 00\ 00\ 11\ 00\ 00 \\ 10\ 10\ 00\ 00\ 11\ 00 \end{bmatrix}$$

10 10 10 00 00 11

โดยความสัมพันธ์กันของ G(x) และ $H(x)^T$ คือ $G(x)H(x)^T = 0$ สามารถแสดงได้ดังนี้

$$G(x)H(x)^T = \begin{bmatrix} 11\ 00\ 00\ 01\ 01\ 01 \\ 00\ 11\ 00\ 00\ 01\ 01 \\ 00\ 00\ 11\ 00\ 00\ 01 \\ 00\ 00\ 00\ 11\ 00\ 00 \\ 00\ 00\ 00\ 00\ 11\ 00 \\ 00\ 00\ 00\ 00\ 00\ 11 \end{bmatrix} \begin{bmatrix} 1\ 0\ 0\ 1\ 1\ 1 \\ 1\ 0\ 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0\ 1\ 1 \\ 0\ 1\ 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 0\ 0 \\ 0\ 0\ 0\ 1\ 0\ 0 \\ 0\ 0\ 0\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0\ 1\ 0 \\ 0\ 0\ 0\ 0\ 0\ 1 \\ 0\ 0\ 0\ 0\ 0\ 1 \end{bmatrix} = 0$$

จาก $c(x) = d(x)G(x)$ ถ้าเราสมมุติ $d(x) = (\ 110010\ )$ เราจะสามารถหา code word c(x) ได้ดังนี้

$$c(x) = (\ 110010) \begin{bmatrix} 11\ 00\ 00\ 01\ 01\ 01 \\ 00\ 11\ 00\ 00\ 01\ 01 \\ 00\ 00\ 11\ 00\ 00\ 01 \\ 00\ 00\ 00\ 11\ 00\ 00 \\ 00\ 00\ 00\ 00\ 11\ 00 \\ 00\ 00\ 00\ 00\ 00\ 11 \end{bmatrix}$$

$= (\ 11\ 11\ 00\ 01\ 11\ 00\ )$

และเราสามารถแสดงวิธีการหา syndrome ได้ดังนี้

$v(x) = c(x) + e(x) = (\ 11\ 11\ 00\ 01\ 11\ 00\ ) + (\ 10\ 00\ 00\ 00\ 00\ 00\ )$

$= (\ 01\ 11\ 00\ 01\ 11\ 00\ )$

$s(x) = v(x)\ H(x)$

โดยที่ s(x) คือ syndrome

e(x) คือ error ที่เกิดขึ้น

v(x) คือ สัญญาณที่เข้ามาในวงจรถอดรหัส

สามารถแสดงได้โดย

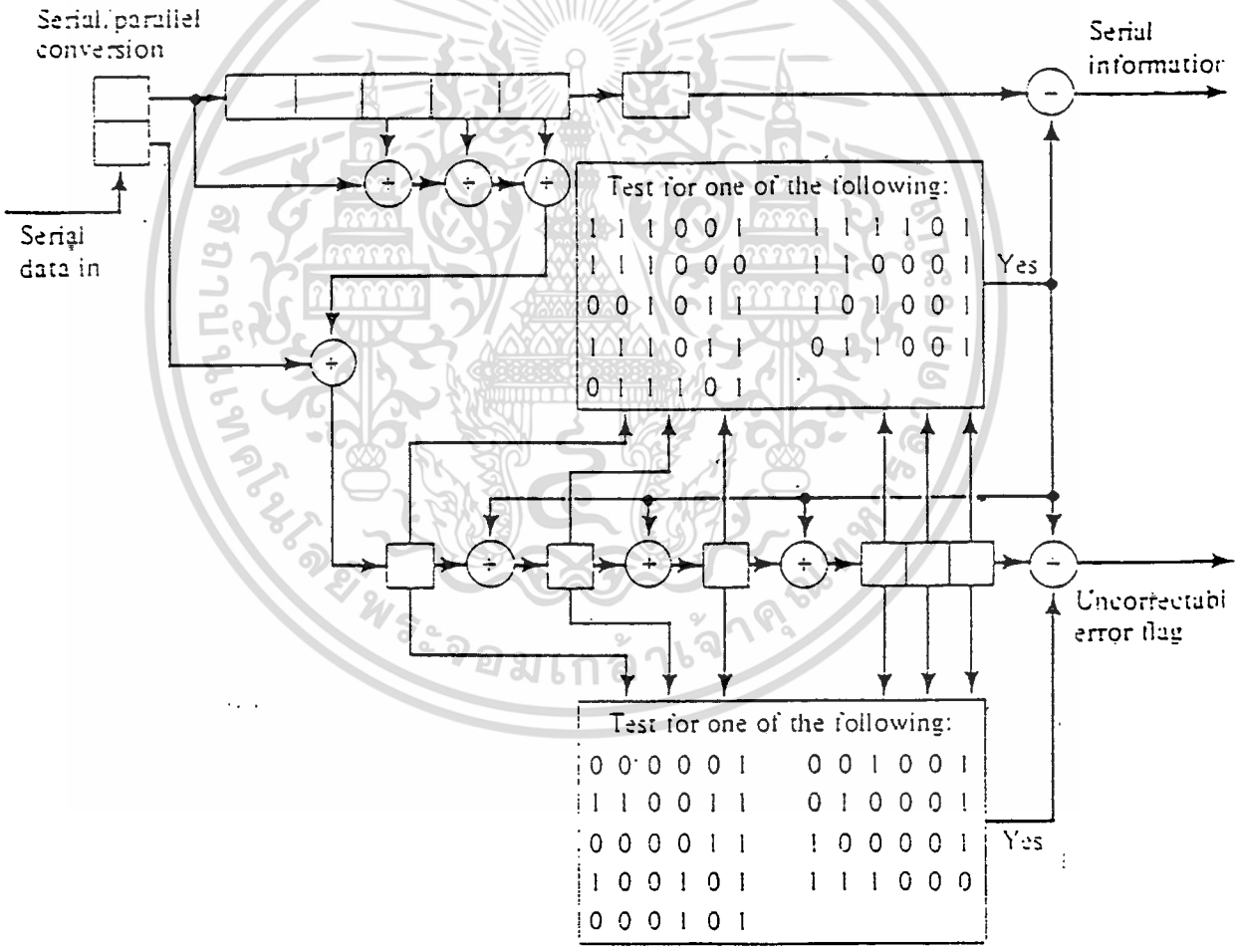
$$s(x) = (01\ 11\ 00\ 01\ 11\ 00)\begin{bmatrix} 1&0&0&1&1&1 \\ 1&0&0&0&0&0 \\ 0&1&0&0&1&1 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&1 \\ 0&0&1&0&0&1 \\ 0&0&0&1&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \\ 0&0&0&0&0&1 \end{bmatrix} = (1\ 0\ 0\ 1\ 1\ 1)$$

และสามารถเขียน syndrome ทั้งหมดได้ดังตาราง

Error Pattern	Syndrome
... 0 0 0 0 0 0 0 0 0 0 0 1	... 1 1 1 0 0 1
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 0 0 1 0 1	0 0 1 0 1 1
0 0 0 0 0 0 0 0 1 0 0 1	1 1 1 0 1 1
0 0 0 0 0 0 0 1 0 0 0 1	0 1 1 1 0 1
0 0 0 0 0 0 1 0 0 0 0 1	1 1 1 1 0 1
0 0 0 0 0 1 0 0 0 0 0 1	1 1 0 0 0 1
0 0 0 0 1 0 0 0 0 0 0 1	1 1 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 1	1 0 1 0 0 1
0 0 1 0 0 0 0 0 0 0 0 1	1 1 0 0 0 1
0 1 0 0 0 0 0 0 0 0 0 1	0 1 1 0 0 1
1 0 0 0 0 0 0 0 0 0 0 1	0 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1	1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 1 0 1 0	0 0 0 0 1 1
0 0 0 0 0 0 0 0 1 0 0 1 0	1 0 0 1 0 1
0 0 0 0 0 0 0 1 0 0 0 1 0	0 0 0 1 0 1
0 0 0 0 0 0 1 0 0 0 0 1 0	0 0 1 0 0 1
0 0 0 0 0 1 0 0 0 0 0 1 0	0 0 1 0 0 1
0 0 0 0 1 0 0 0 0 0 0 1 0	0 1 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 1 0	0 1 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 1 0	1 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0 0 1 0	1 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 1 0	

-ส่วนของวงจรถอดรหัส

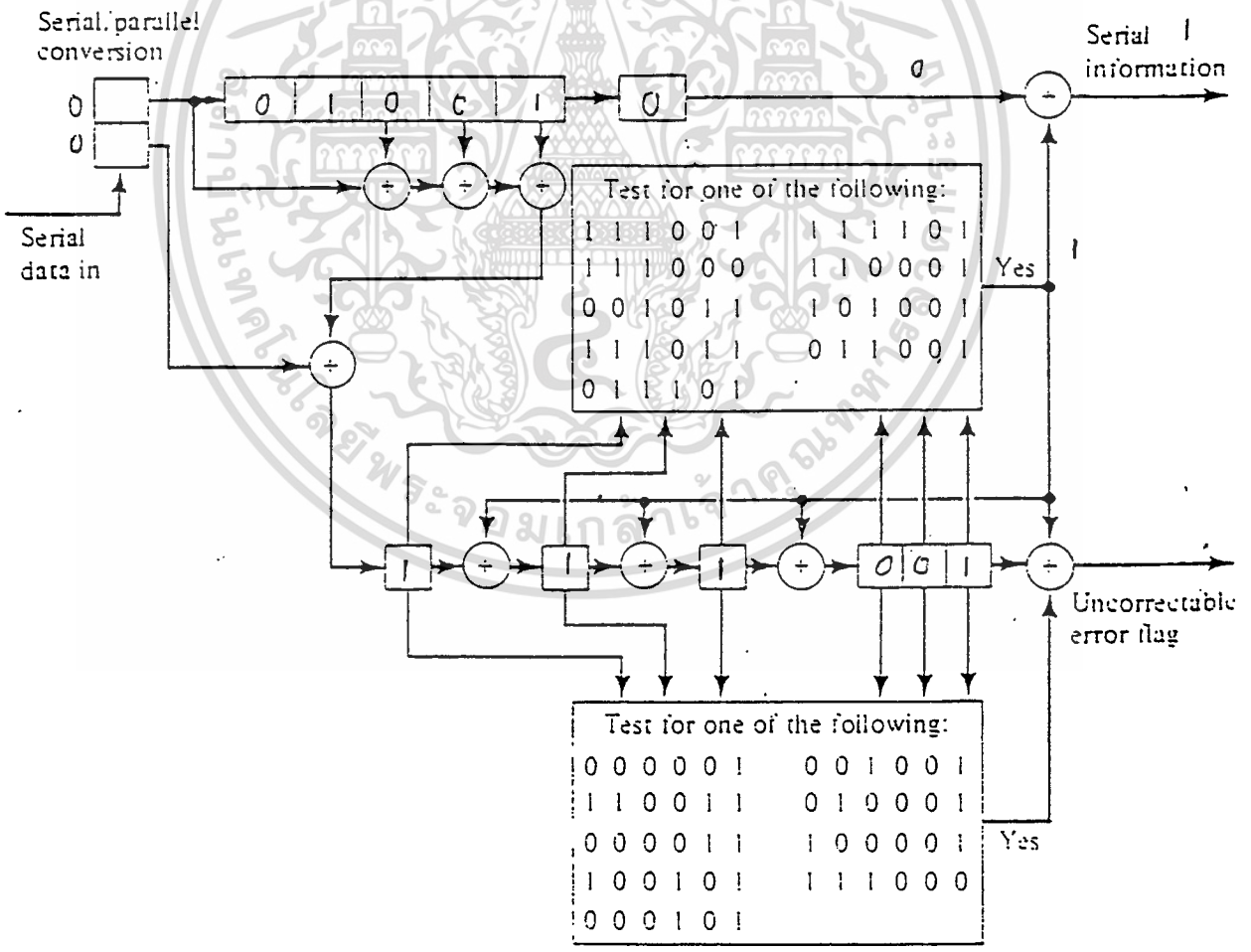
วงจรถอดรหัสนอกจากจะถอดรหัสได้แล้วยังสามารถที่จะแก้ไขความผิดพลาดได้ โดยอาศัยหลักการเปรียบเทียบกับ syndrome ที่หาได้จากข้างต้น และตัววงจรถอดรหัสจะสามารถแก้ไข error ที่เกิดขึ้นได้ 2 บิท ภายใน blocklength 12 บิท แต่ถ้าเกิด error เกินจากนี้ จะไม่สามารถแก้ได้ โดย uncorrectable error flag จะมีค่าเป็น 1



รูปที่ 4.2 รูปวงจรถอดรหัสที่สามารถตรวจจับและแก้ไข error ได้

-วิธีการแก้ไข error ที่เกิดขึ้นภายในวงจรถอดรหัส

จาก code word ที่ได้จากการเข้ารหัสข้างต้น คือ (11 11 00 01 11 00) แต่ถ้าเกิด error ขึ้น เช่นสมมุติ code word ที่ผิดคือ (01 11 00 01 11 00) ซึ่งบิทแรกแทนที่จะเป็น 1 แต่จะกลายเป็น 0 วงจรถอดรหัสจะสามารถแก้ไขให้ถูกต้องโดย จะได้ syndrome ตรงกับตารางข้างต้นซึ่งจะทำให้วงจรสามารถที่จะแก้ไข error ให้ถูกต้องได้ โดยหลังจากได้แก้ error แล้วจะมีการป้อนกลับไปทำการปรับ syndrome register เพื่อที่จะสามารถแก้ error ต่อไปได้ ดังจะแสดงสถานะได้ดังรูปข้างล่าง



รูปที่ 4.3 รูปแสดงการแก้ข้อผิดพลาดของวงจรถอดรหัส

สรุปและวิจารณ์ผล

จากการดำเนินงานที่ผ่านมาในภาคเรียนนี้สามารถสรุปผลที่ได้ดำเนินงานมาดังนี้

1. ในการออกแบบวงจรเข้ารหัสและถอดรหัสแบบคอนโวลูชั่นที่ได้ศึกษานั้น จะเป็นชนิด Systematic Convolutional Code ซึ่ง จะใช้ในการเข้ารหัสและถอดรหัสข้อมูลในการรับส่งข้อมูลเพื่อป้องกันความผิดพลาดที่อาจเกิดกับข้อมูล โดยการเข้ารหัสและถอดรหัสแบบนี้จะสามารถตรวจสอบข้อผิดพลาดและแก้ไขข้อผิดพลาดได้

2. ในการเข้ารหัสจะอาศัย เจนเนอร์เรเตอร์ เมตริกส์ เป็นตัวเข้ารหัส และการถอดรหัสเราจะใช้อินเวอร์เมตริกส์ ของ เจนเนอร์เรเตอร์เมตริกส์ เป็นตัวถอดรหัสเพื่อให้ได้ข้อมูลจากรหัสที่เข้ามา ดังนั้น การหาเมตริกส์ที่จะนำมาใช้ในการถอดรหัสจะต่อมี อินเวอร์เมตริกส์ของตัวมันเองจึงจะสามารถนำมาใช้ได้

3. การออกแบบวงจรถอดรหัสนอกจากจะต้องใช้ อินเวอร์เมตริกส์ของ เจนเนอร์เรเตอร์เมติกส์แล้ว ยังต้องใช้ Parity-check-polynomial matrix H(x) ซึ่งทรานสโพสของมันจะใช้ในการหา Syndrome ของerror ที่เกิดขึ้นในการรับส่งข้อมูล

วิจารณ์ผลการดำเนินงาน จากการดำเนินงานที่ผ่านมาเราได้เลือกศึกษาวงจร Systematic Convolutional Code เนื่องจากการเข้ารหัสวิธีนี้จะสามารถถอดรหัสได้เลยโดยไม่ต้องคำนึงถึง Delay ที่จะต้องเพื่อเข้าไป เหมือนวงจร Nonsystematic Convolutional Code ทำให้ลดความยุ่งยากในการออกแบบวงจร อีกทั้งยังเป็นการเพิ่มความมั่นใจได้ว่าจะสามารถถอดรหัสได้เนื่องจาก Systematic Convolutional Code เป็นวงจรแบบ Noncatastrophic Convolutional Code เสมอ

บทที่ 5
โปรแกรมเข้ารหัสถอดรหัส

5.1 ลำดับขั้นตอนการเขียนโปรแกรม

ในส่วนของการเขียนโปรแกรม เราจะใช้ภาษา วีเฮชดีแอล ในการเขียนโดยใช้ซอฟแวร์ วีซิสเตม ซึ่งจะได้โปรแกรมออกมาเป็นนามสกุล วีเฮชดี โดยในขั้นต้นหลังจากการเขียนโปรแกรมได้แล้วเราจะนำมาทำการ คอมไพล์ และ ซิมูเลชั่น เพื่อจะจำลองการทำงานของวงจร และเมื่อได้คุณสมบัติของวงจรถูกต้องตามต้องการ ต่อจากนั้นก็จะทำการสังเคราะห์ ก็จะได้ออกมาเป็นนามสกุล เอ็กซเอ็นเอฟ และจะนำมาทำการเมิจเพื่อทำการใส่ดีเลย์ จากนั้นจึงนำไปทำการ ซิมูเลชั่น เพื่อดูการทำงานว่ามีคุณสมบัติตามต้องการหรือไม่

5.2 ส่วนของโปรแกรมเข้ารหัส

ในส่วนของวงจรเข้ารหัสจะประกอบไปด้วยส่วนที่สำคัญ ได้แก่ ชิฟท์รีจิสเตอร์ 5 บิท ซึ่งจะทำหน้าที่เป็นตัวเจนเนอร์เรเตอร์ และจะมีชิฟท์รีจิสเตอร์ที่เอาท์พุท ทำหน้าที่ส่งข้อมูลออก

จากบทที่แล้วหลักในการเขียนโปรแกรม จะเขียนโปรแกรมเป็นลำดับชั้น โดยโปรแกรมหลัก(อุปกรณ์หลัก)จะเรียกใช้โปรแกรมย่อย(อุปกรณ์ย่อย) โดยจะมีซเททเมทชีน ทำการควบคุมอุปกรณ์ย่อยและอุปกรณ์หลักให้ทำงานที่เวลาเดียวกัน เพื่อให้ได้การทำงานที่สอดคล้องกันตามต้องการ

ในส่วนของวงจรเข้ารหัสเราจะมีโปรแกรม encode1 เป็นโปรแกรมหลักที่ทำการเข้ารหัส โดยโปรแกรม encode1 จะเรียกใช้โปรแกรมย่อยที่เป็นส่วนประกอบในการเข้ารหัสมาใช้งานดังนี้

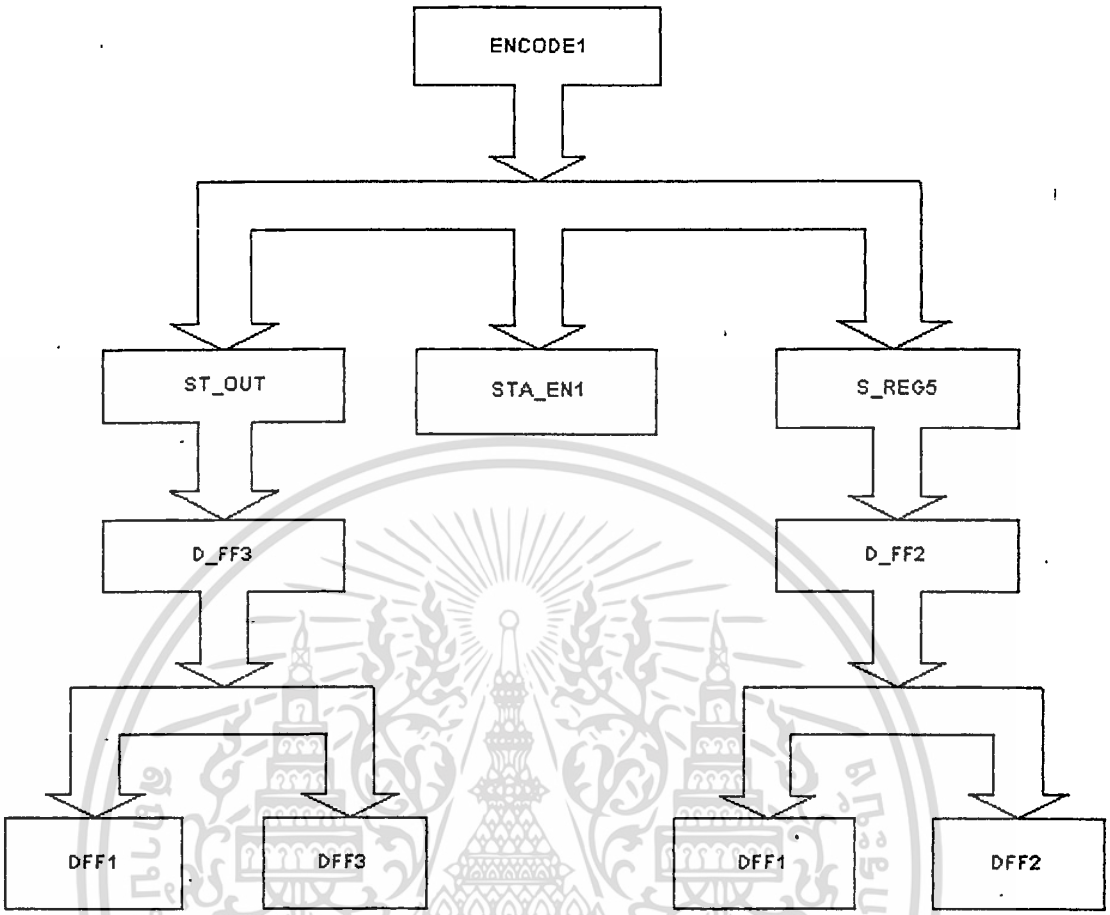
1. โปรแกรม s_reg5 โดยโปรแกรม s_reg5 เป็นโปรแกรมชิฟรีจิสเตอร์ ใช้ในการเข้าการรหัสโดยโปรแกรม s_reg5 จะทำการเรียกใช้โปรแกรมย่อยอีกคือ โปรแกรมd_ff2 และโปรแกรม d_ff2 จะเรียกใช้ โปรแกรม dff1 และ โปรแกรม dff2 มาใช้

2. โปรแกรมst_out เป็นโปรแกรมที่ใช้ในการส่งข้อมูลออกในลักษณะอนุกรม โดยโปรแกรม st_out จะประกอบไปด้วยโปรแกรมย่อย คือ โปรแกรมd_ff3ซึ่งจะมีโปรแกรมย่อย คือ โปรแกรม dff1 และ โปรแกรม dff3

3. โปรแกรม sta_en1 เป็นโปรแกรมซเททเมทชีน ซึ่งจะเป็นโปรแกรมหลักที่คอยควบคุมการทำงานของโปรแกรมอื่นๆ ให้ทำงานตามเวลาที่ต้องการ โดยจะทำการควบคุมการทำงานของชิฟรีจีสเตอร์(s_reg5) และ ภาคส่งข้อมูล (st_out) รวมทั้งควบคุมการทำงานของ โปรแกรมย่อยต่างๆ เช่นโปรแกรม d_ff2,d_ff3

โดยแผนภาพข้างล่างจะแสดงลำดับชั้นของการเขียนโปรแกรมดังที่ได้อธิบายไว้แล้วข้างต้น โดย โปรแกรม encode1 จะเป็นอุปกรณ์หลักที่ใช้ในการเข้ารหัสซึ่งจะทำการเรียกโปรแกรมย่อย

ต่างๆ ดังแสดงในแผนภาพ



รูปที่ 5.1 แผนภาพลำดับชั้นการเขียนโปรแกรม

จากรูปที่ 5.1 เป็นแผนภาพแสดงลำดับชั้นของโปรแกรม ซึ่งโปรแกรมที่เขียนสามารถแปรความได้สัมพันธ์กับการเชื่อมต่อของวงจรเข้ารหัสที่เรานำมาใช้(ในบทที่4) ซึ่งโปรแกรมต่างๆ ที่ใช้ในการเข้ารหัสจะแสดงเป็นส่วนๆ ได้ดังนี้

5.2.1 ส่วนโปรแกรมของวงจรพื้นฐาน

โปรแกรมของวงจรพื้นฐาน ประกอบด้วย

1.โปรแกรม dff1

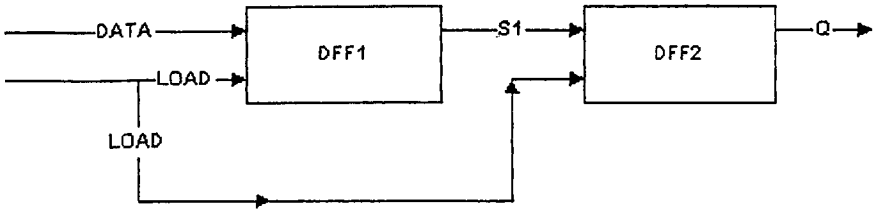
2.โปรแกรม dff2

3.โปรแกรม dff3

4.โปรแกรม d_ff2

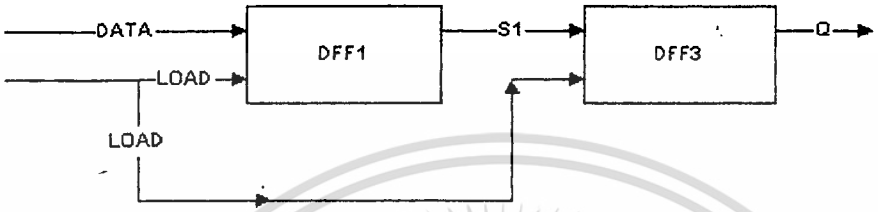
5.โปรแกรม d_ff3

โดยโปรแกรม d_ff2 จะได้จากการนำโปรแกรม dff1 และ โปรแกรม dff2 มาใช้ดังแสดงในรูป 5.2 ซึ่งเป็นแผนภาพแสดงการนำ โปรแกรม dff1 และ โปรแกรม dff2 มาใช้



รูปที่ 5.2 แสดงแผนภาพของโปรแกรม d_ff2

และในส่วนของโปรแกรม d_ff3 ก็สามารถแสดงการเชื่อมต่อได้เช่นเดียวกันดังรูปที่ 5.3



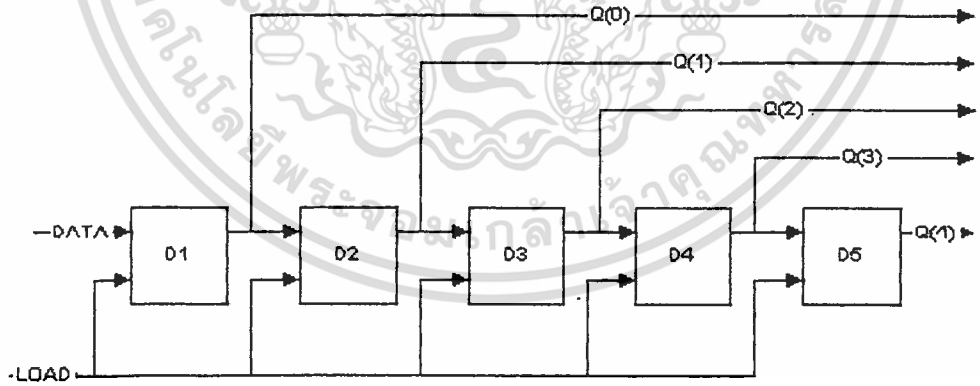
รูปที่ 5.3 แสดงแผนภาพของโปรแกรม d_ff3

5.2.2 ส่วนโปรแกรมของวงจรที่นำมาใช้ในโปรแกรมวงจรเข้ารหัส

ส่วนของโปรแกรมที่นำมาใช้ประกอบเป็นวงจรเข้ารหัสประกอบด้วย

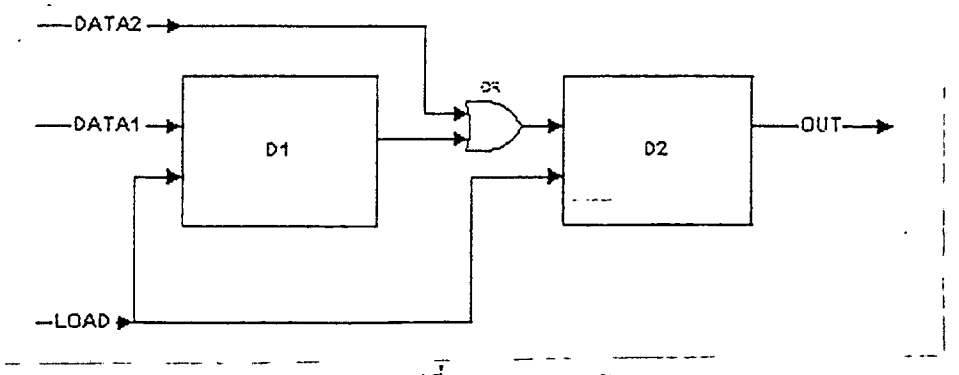
1. โปรแกรม s_reg5
2. โปรแกรม st_out

โดยโปรแกรม s_reg5 ได้นำ โปรแกรม d_ff2 มาใช้ ดังรูปที่ 5.4



รูปที่ 5.4 แผนภาพโปรแกรม s_reg5

โดยโปรแกรม st_out ได้นำโปรแกรม d_ff3 มาใช้โดยแสดงได้ดังรูป5.5



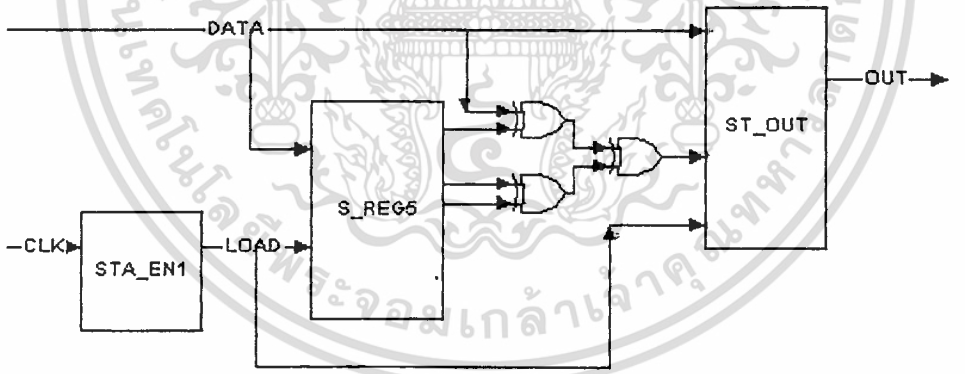
รูปที่ 5.5 แผนภาพโปรแกรม st_out

5.2.3 ส่วนโปรแกรมที่ใช้ในการควบคุม(statemachine)

โปรแกรมที่ใช้ในการควบคุมหรือ สเตทแมทชีน คือโปรแกรมที่ใช้ในการควบคุมอุปกรณ์ต่าง ๆ ให้ทำงานร่วมกันได้อย่างถูกต้อง โดยส่วนของโปรแกรมที่ใช้ในการควบคุมจะส่งสัญญาณโหลดไปให้แก่โปรแกรม st_out , s_reg5 เพื่อใช้เป็นสัญญาณควบคุมจังหวะการทำงานของโปรแกรมย่อยต่างๆ

5.2.4 ส่วนโปรแกรมของวงจรเข้ารหัส

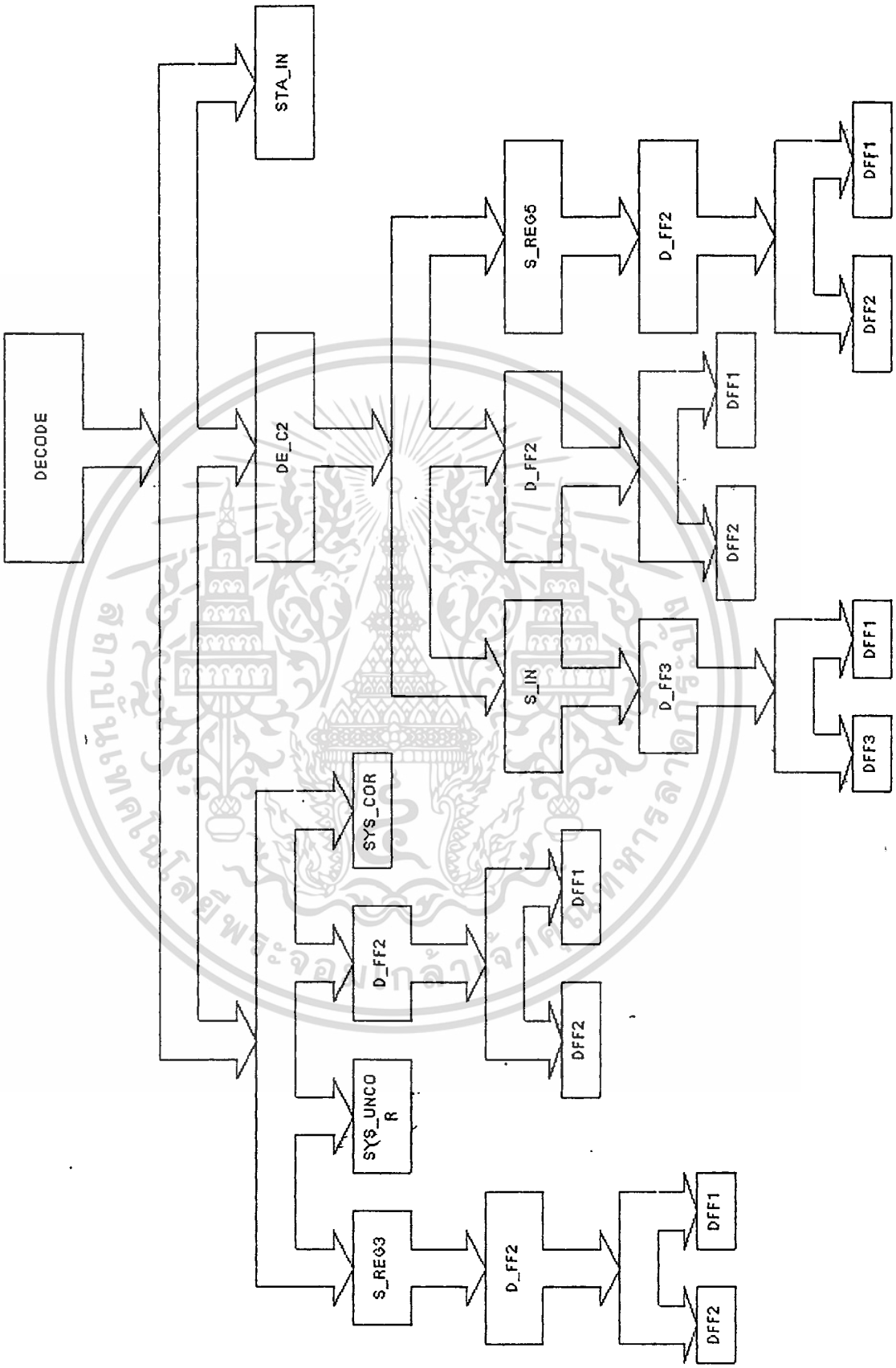
ส่วนโปรแกรมของวงจรเข้ารหัสจะมีการนำโปรแกรมst_out,s_reg5,sta_en1 มาใช้ ซึ่งสามารถแสดงแผนภาพของโปรแกรมได้ดังรูปที่ 5.6



รูปที่ 5.6 แสดงแผนภาพของโปรแกรมที่ใช้ในการเข้ารหัส

จากรูปที่ 5.6 แสดงให้เห็นว่า โปรแกรมเข้ารหัสได้นำโปรแกรม s_reg5 มาใช้แทน shift register 5 bit และได้นำโปรแกรมย่อย st_out มาช่วยในการส่งสัญญาณออก หรือเรียกอีกอย่างหนึ่งว่าเป็น บัฟเฟอร์ข้อมูลขาออก และจะมีส่วน โปรแกรม sta_en1 เป็นส่วนที่ใช้ในการควบคุมการทำงานของโปรแกรมย่อย st_out , s_reg5 ให้ทำงานเข้ากันได้และจะมีขา reset เป็นสัญญาณที่ส่งมา set สถานะต่างๆ ของโปรแกรมย่อย

5.3ส่วนของโปรแกรมถอดรหัส



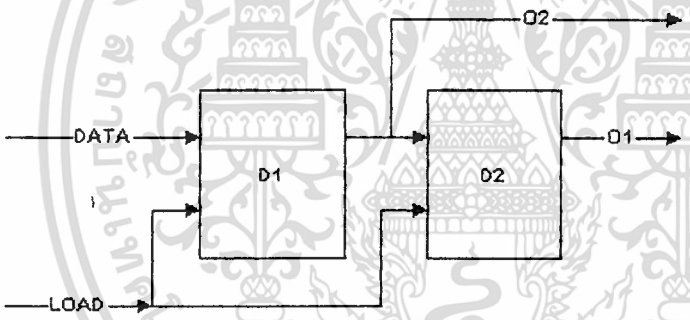
รูปที่ 5.7 แผนภาพแสดงลำดับชั้นของโปรแกรมถอดรหัส

ในส่วนของการเขียนโปรแกรมถอดรหัสก็มีหลักการเขียนเช่นเดียวกับภาคเข้ารหัส คือ อาศัยหลักลำดับขั้นของโปรแกรมโดยในส่วนนี้จะมีโปรแกรมหลัก คือ โปรแกรม decode1 ซึ่งเป็นโปรแกรมที่ใช้ในการถอดรหัส โดยที่โปรแกรม decode1 จะเรียกใช้โปรแกรมย่อย คือ โปรแกรม sta_in โปรแกรม de_c2 และโปรแกรม check_er2 โดยสามารถแสดงลำดับชั้นของโปรแกรมได้ดังรูปที่ 5.7 ซึ่งโปรแกรมย่อยแต่ละโปรแกรมทำหน้าที่แตกต่างกันไปดังนี้ โดยแบ่งได้เป็น

5.3.1 ส่วนโปรแกรมที่ทำหน้าที่รับข้อมูลและถอดรหัส

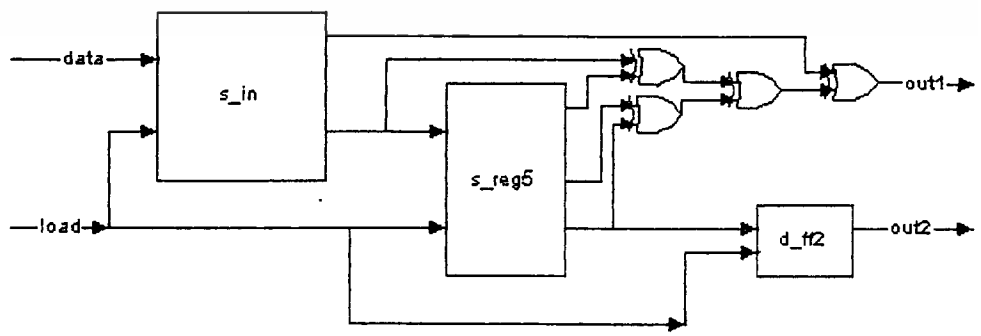
โปรแกรมที่ทำหน้าที่นี้ คือ โปรแกรม de_c2 ซึ่งโปรแกรมนี้จะมีโปรแกรมย่อยในส่วนรับข้อมูลก็ คือ โปรแกรม s_in ซึ่งจะรับข้อมูลที่ส่งมาแล้วแปลงให้เป็นแบบขนาน จากนั้นจะเข้าสู่โปรแกรมย่อยในส่วนการถอดรหัส ซึ่งก็จะประกอบด้วยโปรแกรม s_reg5 และโปรแกรม d_ff2

ในส่วนโปรแกรมย่อย s_in นั้นจะมีการเรียกใช้โปรแกรม d_ff3 ซึ่งได้แสดงแผนภาพของโปรแกรม s_in ดังรูปที่ 5.8



รูปที่ 5.8 แผนภาพโปรแกรม s_in

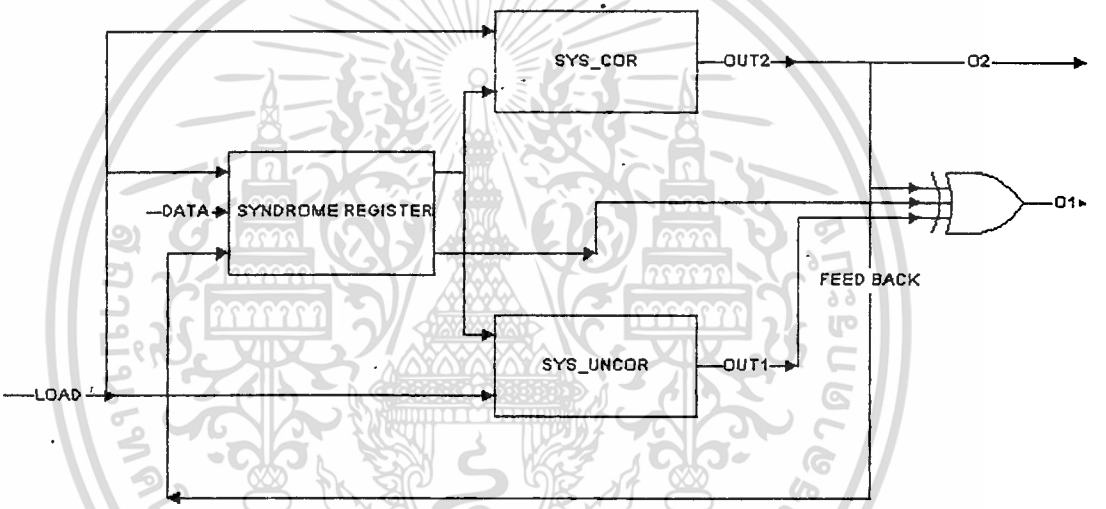
โดยในส่วนโปรแกรม de_c2 ที่มีการเรียกใช้ โปรแกรม s_reg5,d_ff2 และ s_in สามารถแสดงแผนภาพของโปรแกรมได้ดังรูปที่ 5.9



รูปที่ 5.9 แผนภาพของโปรแกรม de_c2

5.3.2 ส่วนโปรแกรมที่ทำหน้าที่ตรวจสอบและแก้ไข ความผิดพลาด

โปรแกรมที่ทำหน้าที่แก้ไขและตรวจสอบข้อผิดพลาด คือโปรแกรม check_er2 ซึ่งโปรแกรมนี้จะมีการเรียกใช้โปรแกรมย่อย sys_cor , sys_uncor มาใช้เป็นตัวเปรียบเทียบข้อมูลที่ส่งมากับ ลักษณะของข้อผิดพลาด โดยการเปรียบเทียบ ซินโดมรีจิสเตอร์ กับ รูปแบบของข้อผิดพลาดที่เก็บในโปรแกรมย่อย sys_cor , sys_uncor โดยจะใช้โปรแกรมย่อย s_reg3 , d_ff2 เป็นโปรแกรมที่ทำหน้าที่เป็น ซินโดมรีจิสเตอร์ หลักการทำงานของ โปรแกรม check_er2 จะทำงานโดยโปรแกรมย่อยsys_cor , sys_uncor จะอ่านค่าจากเอาท์พุทของ ซินโดมรีจิสเตอร์ มาเปรียบเทียบกับค่าที่อยู่ในโปรแกรมแล้วจะส่งค่าไปยังเอาท์พุทของ check_er2 และไปยัง ซินโดมรีจีสเตอร์ซึ่งสามารถแสดงแผนภาพลำดับชั้นของโปรแกรม check_er2 ดังรูปที่ 5.10



รูปที่ 5.10 แผนภาพของโปรแกรม check_er2

5.3.3 ส่วนของโปรแกรมที่ทำหน้าที่ควบคุม

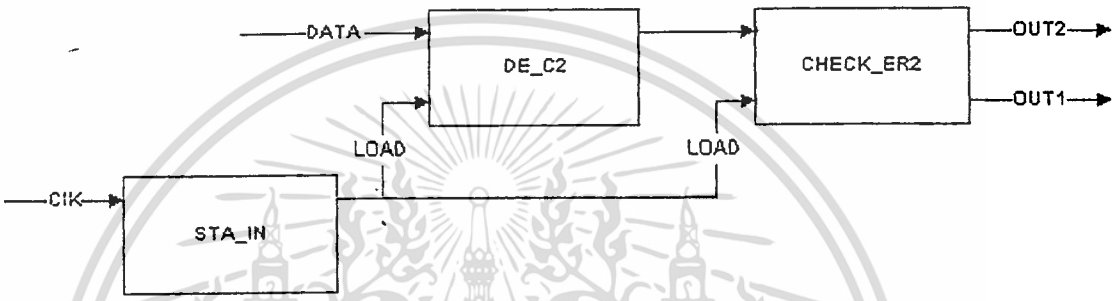
โปรแกรมที่ทำหน้าที่ส่วนนี้ก็ คือ sta_in ซึ่งโปรแกรมนี้จะให้สัญญาณเอาท์พุทคือสัญญาณโหลด ไปยังโปรแกรมย่อยอื่นๆในโปรแกรม decode1 ซึ่งโปรแกรมส่วนควบคุมนี้ก็มีลักษณะการทำงานคล้ายกับส่วนที่ใช้ในการควบคุมส่วนเข้าของการเข้ารหัสซึ่งจะใช้ในการควบคุมการทำงานของโปรแกรมย่อยส่วนต่างๆ

5.3.4 ส่วนของโปรแกรมหลักในการถอดรหัส

ในส่วนนี้จะใช้โปรแกรม decode1 ซึ่งโปรแกรม decode1 จะเรียกใช้โปรแกรมย่อย sta_in , de_c2 , check_er2 โดยโปรแกรม sta_in จะเป็นตัวกำหนดจังหวะการทำงานให้แก่ โปรแกรม de_c2 , check_er2 ให้ทำงานเข้ากันได้ซึ่งส่วนของโปรแกรม de_c2 จะเป็นภาครับข้อมูลและถอด

รหัสส่วนโปรแกรม check_er2 จะทำหน้าตรวจ และแก้ไขข้อผิดพลาด ซึ่งได้อธิบายไว้แล้วข้างต้น และจะมีขา reset ไว้ในการ set สถานะเริ่มต้นในการทำงานของโปรแกรม ย่อยต่างๆ

เราสามารถแสดงลำดับชั้นของโปรแกรม decode1 ได้ดัง รูปที่ 5.11



รูปที่ 5.11 แผนภาพแสดงโปรแกรม decode1

บทที่ 6 ผลการทำงานของโปรแกรม

6.1 ผลการทำงานของโปรแกรมวงจรในส่วนเข้ารหัส

โปรแกรมของวงจรภาคเข้ารหัสนั้นจะมีโปรแกรมหลัก คือ โปรแกรม encode1 ซึ่งจะเป็นโปรแกรมที่ทำหน้าที่ในการเข้ารหัส โดยโปรแกรมนี้จะ นำโปรแกรม sta_en1,st_out,s_reg5 มาใช้ร่วมกัน โดยโปรแกรม sta_en1 จะเป็นโปรแกรมควบคุม(statemechine) ที่จะเป็นตัวให้กำเนิดสัญญาณ โหลด ส่งให้แก่โปรแกรม st_out,s_reg5 เพื่อให้โปรแกรมแต่ละตัวสามารถทำงานร่วมกันได้อย่างถูกต้อง ซึ่งผลการทำงานของโปรแกรม sta_en1 นี้ จะมีผลต่อการทำงานของโปรแกรมอื่นที่นำสัญญาณ โหลด ไปใช้ดังนั้นโปรแกรม sta_en1 จึงมีความสำคัญต่อการทำงานของโปรแกรมรวมซึ่งผลการทำงานของโปรแกรม sta_en1 ที่ได้จากการจำลองการทำงานได้ผลออกมาถูกต้องตามต้องการ ซึ่งจะมีผลให้การทำงานของโปรแกรม อื่นๆ นั้นสอดคล้องกันด้วย

ส่วนของโปรแกรม st_out เป็นโปรแกรมที่ทำหน้าที่ส่งข้อมูลออก ซึ่งจะมีลักษณะที่คล้ายกับบัฟเฟอร์(buffer) ในตอนส่งข้อมูลออกและจะมีลักษณะที่พิเศษอีกอย่าง คือ การทำงานของโปรแกรม st_out จะมีจังหวะการทำงานที่เร็วเป็น 2 เท่าเมื่อเทียบกับการทำงานของ โปรแกรม s_reg5 เนื่องจากจะต้องทำการรับข้อมูลในลักษณะของ ข้อมูลขนาน(parallel) และจะต้องส่งข้อมูลออกในลักษณะอนุกรม(series) จึงต้องมีจังหวะการทำงานที่เร็วเป็น2 เท่าเมื่อเทียบกับส่วนที่ส่งข้อมูลมาให้

ซึ่งจากการจำลองการทำงานของโปรแกรม st_out ได้ผลการทำงานตามที่ต้องการ และในส่วนของโปรแกรม s_reg5 ซึ่งเป็นโปรแกรมที่ทำหน้าที่เป็น ชิพท์รีจีสเตอร์ นั้นจากการจำลองการทำงานของโปรแกรมก็ได้ผลออกมาถูกต้องเช่นกัน แต่ในตอนแรกของการเขียนโปรแกรมยังมีปัญหาเรื่องการแล็ซ(latch) เนื่องจากการทำงานของ ชิพท์รีจีสเตอร์ที่ต้องการนั้นจะคล้ายกับบัฟเฟอร์ คือ จะมีการหน่วงข้อมูลในหนึ่งช่วงการทำงาน จึงไม่สามารถที่จะเรียกใช้ ฟลิปฟลอปทั่วไปได้จึงได้มีการนำฟลิปฟลอป 2 ตัวมาต่อเชื่อมกัน เพื่อให้ได้การทำงานตามต้องการ(เกิดการหน่วง) และได้อาศัย โปรแกรม sta_en1 เข้ามาช่วยในการให้จังหวะการทำงานที่เหมาะสม ซึ่งทำให้สามารถทำการจำลองการทำงานของวงจรได้ผลการทำงานตามที่ต้องการเช่นกัน

ในส่วนของโปรแกรมหลักในภาคเข้ารหัสนั้นก็ คือ โปรแกรม encode1 นั้นได้มีการนำโปรแกรมย่อยต่างๆ ข้างต้นมาต่อเชื่อมกัน ซึ่งหลังจากการทำการจำลองการทำงานของโปรแกรมเข้ารหัสก็ได้ผลการทำงานตามที่ต้องการ และได้ทำการสังเคราะห์ส่วนของโปรแกรมเข้ารหัสทั้งในส่วนของโปรแกรมย่อย และในส่วนของโปรแกรมหลักแล้ว ก็จะได้วงจรลอจิก(logic)ออกมาตามต้องการ จากนั้นก็จะนำ ไพล์ จุดวีเอชดีที่ได้จากการทำการสังเคราะห์มาทำการ จำลองการทำงานของโปรแกรมอีกครั้ง โดยใช้โปรแกรมทดสอบมาทำการทดสอบการทำงานของโปรแกรมที่ได้มาใหม่ หลังจากทำการทดสอบการทำงานของโปรแกรมดูแล้วได้พบว่าการทำงานที่ได้นั้น ถูกต้องตามต้องการ จึงนำโปรแกรม จุด xnf ไปทำการรวม ซึ่งการรวมนี้จะเป็นการนำโปรแกรม

ย่อยต่างๆ ที่ใช้ในโปรแกรมเข้ารหัสมาต่อเชื่อมกันออกมาเป็น วงจรเข้ารหัสและจะทำการใส่ดีเลย์เข้าไปด้วยเพื่อให้ได้การทำงานที่ถูกต้องตามความเป็นจริง จากนั้นก็จะนำไปทำการจำลองการทำงานอีกครั้งเป็นโดยใช้โปรแกรมทดสอบในการทดสอบการทำงานของโปรแกรมวงจรที่ได้

6.2สรุปผลการทำงานของโปรแกรมวงจรถอดรหัส

โปรแกรมที่ใช้ในการถอดรหัสนั้นจะมีโปรแกรม decode1 เป็นโปรแกรมหลัก โดยจะมีการเรียกใช้โปรแกรมย่อยได้แก่ โปรแกรม sta_in ,de_c2,check_er2 มาใช้ร่วมกันในการถอดรหัส ซึ่งโปรแกรม sta_in จะเป็นโปรแกรมควบคุมการทำงานของโปรแกรม de_ce2,check_er2 ดังนั้นการทำงานของโปรแกรม sta_in นั้นจะมีผลต่อการทำงานของโปรแกรมอื่นๆ ด้วย ซึ่งจากการออกแบบและจำลองการทำงานของโปรแกรมแล้ว ได้ผลออกมาถูกต้องตามต้องการ ในส่วนของโปรแกรม de_c2 นั้น ซึ่งเป็นโปรแกรมในส่วนของการรับข้อมูลที่ส่งมา แล้วทำการแปลงข้อมูลที่ส่งมาแบบอนุกรมให้เป็นแบบขนาน และจะนำไปทำการถอดรหัสต่อไป ซึ่งจากการจำลองการทำงานของโปรแกรมส่วนนี้ก็ได้ผลการทำงานถูกต้องตามต้องการ

ส่วนโปรแกรมที่ทำการตรวจสอบและแก้ไขข้อผิดพลาดของข้อมูลที่ได้จากการเข้าและถอดรหัส ซึ่งก็คือโปรแกรม check_er2 นั้นได้มีการเรียกใช้โปรแกรมต่างๆ ตามที่ได้กล่าวไว้ในบทที่ 5 นั้นจากการนำโปรแกรมย่อยต่าง ๆไปทำการจำลองการทำงานนั้น ก็ได้ผลการทำงานที่ถูกต้องตามต้องการ จากนั้นจึงนำโปรแกรม check_er2 ไปจำลองการทำงานก็ได้ผลการทำงานที่ถูกต้องตามต้องการ ต่อจากนั้น จึงทำการรวมโปรแกรมย่อย ภาค รับข้อมูล , ภาคตรวจสอบและแก้ไขข้อผิดพลาด และภาคควบคุมการทำงาน มารวมกันเป็นโปรแกรม decode1 ซึ่งเป็นโปรแกรมที่ใช้ในการถอดรหัส จากนั้นจึงได้ทำการจำลองการทำงานของโปรแกรมถอดรหัส ซึ่งก็ได้ผลการทำงานของโปรแกรมถูกต้องตามที่ได้ออกแบบไว้ แล้วจึงนำโปรแกรมต่างๆ ที่ได้จำลองการทำงานแล้วไปทำการ สังเคราะห์เป็นวงจร ดิจิตอล เช่นเดียวกับในส่วนโปรแกรมของภาคเข้ารหัส ก็ได้วงจรถอดรหัสตามต้องการ

เมื่อทำการจำลองการทำงานเรียบร้อยแล้วก็จะนำไปทำการรวมโปรแกรมย่อยต่างๆ ที่ใช้ในการถอดรหัสให้เป็นวงจรเดียวกัน ซึ่งจะทำการต่อเชื่อมขาของอุปกรณ์ต่างๆเข้าด้วยกันและจะทำการใส่ดีเลย์ให้กับอุปกรณ์ต่างๆเพื่อให้การทำงานของโปรแกรมวงจรที่ได้มีความใกล้เคียงกับการทำงานของวงจรจริง และเพื่อไม่ให้เกิดปัญหาภายหลังจากการนำไปสร้างเป็นวงจรจริงแล้ว จากนั้นจะนำโปรแกรมที่ได้ไปทำการจำลองและทดสอบการทำงานของโปรแกรมดูอีกเป็นครั้งสุดท้าย

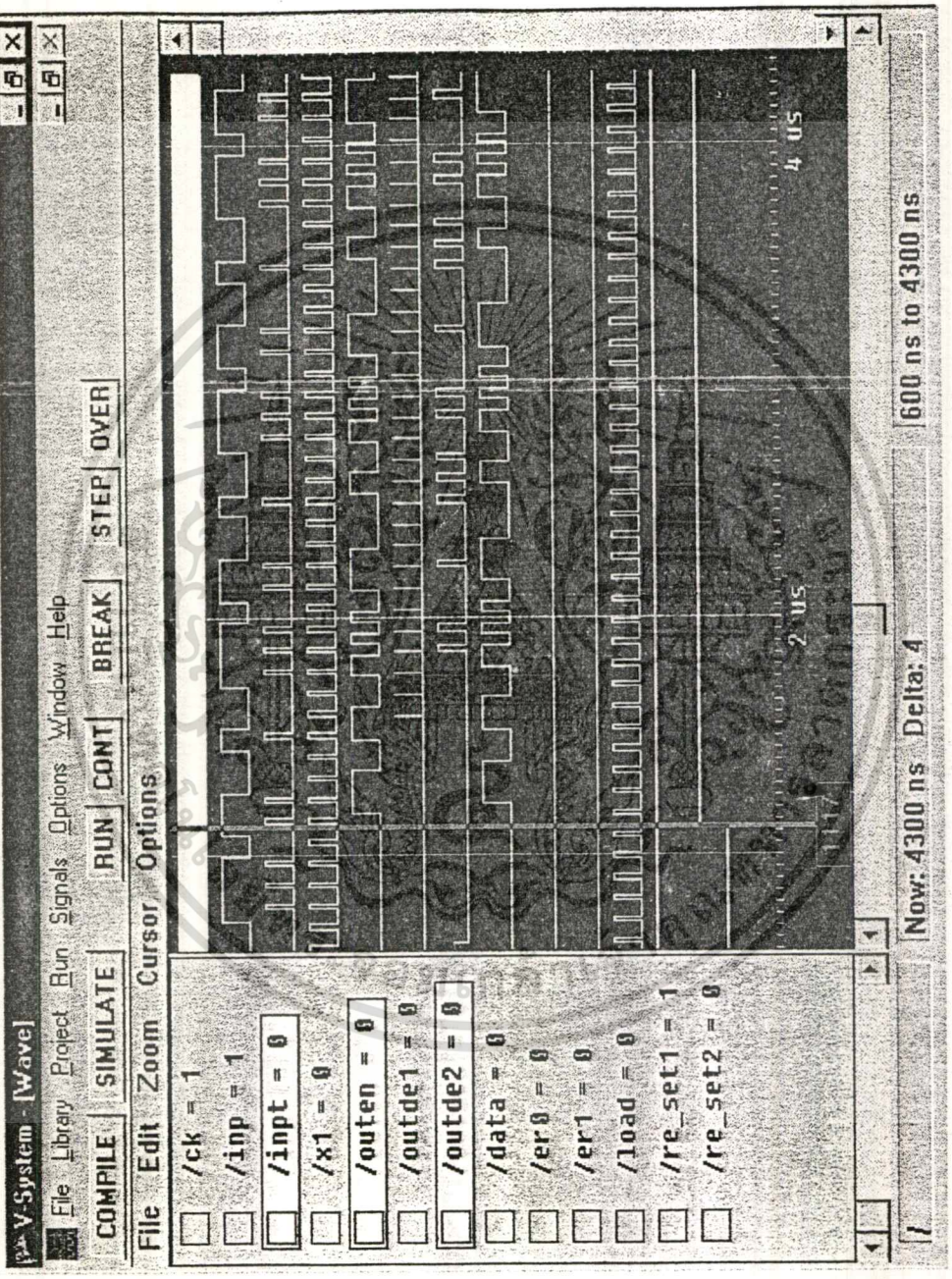
ในการทดสอบการทำงานของโปรแกรมส่วนเข้ารหัสและถอดรหัสนั้นเนื่องจากว่าการทำงานของทั้งสองส่วนมีความสอดคล้องกันจึงได้ใช้โปรแกรมทดสอบเป็นโปรแกรมเดียวกัน โดยโปรแกรมที่ใช้ในการทดสอบการทำงานจะมีการเรียกใช้โปรแกรมเข้ารหัสและโปรแกรมถอดรหัสมาใช้ร่วมกัน โดยโปรแกรมที่ใช้ทดสอบจะทำการกำเนิดความถี่นาฬิกาให้กับโปรแกรมเข้ารหัสและโปรแกรมถอดรหัสได้นำไปใช้ในการกำหนดสถานะการทำงานให้สอดคล้องกัน และจะนำ

เอาท์พุทที่ได้จากโปรแกรมวงจรเข้ารหัสมาเป็นอินพุทให้กับโปรแกรมวงจรถอดรหัส ซึ่งจะสามารถถอดรหัสได้ข้อมูลเดิมที่ทางโปรแกรมทดสอบส่งให้กับโปรแกรมเข้ารหัส ซึ่งเป็นการแสดงให้เห็นว่าการทำงานของทั้งสองโปรแกรมทำงานได้ถูกต้องหรือไม่ หรืออาจตรวจสอบได้จากส่วนย่อยๆคือ สามารถดูการทำงานของโปรแกรมเข้ารหัสได้จากเอาท์พุทของโปรแกรม และสามารถดูผลการทำงานของโปรแกรมถอดรหัสได้จากเอาท์พุททั้งสองของโปรแกรม โดยเอาท์พุทที่หนึ่งจะเป็นส่วนที่ใช้ในการตรวจสอบความผิดพลาดที่อาจเกิดขึ้นในระหว่างการส่งข้อมูลโดยจะมีค่าเป็นหนึ่งเมื่อเกิดความผิดพลาดในการส่งข้อมูล และเอาท์พุทที่สองจะเป็นส่วนที่ใช้ในการถอดรหัสในกรณีที่ไม่เกิดความผิดพลาดในการส่งเอาท์พุทที่สองนี้จะแสดงค่าข้อมูลเดิมที่ได้จากการถอดรหัสแล้ว

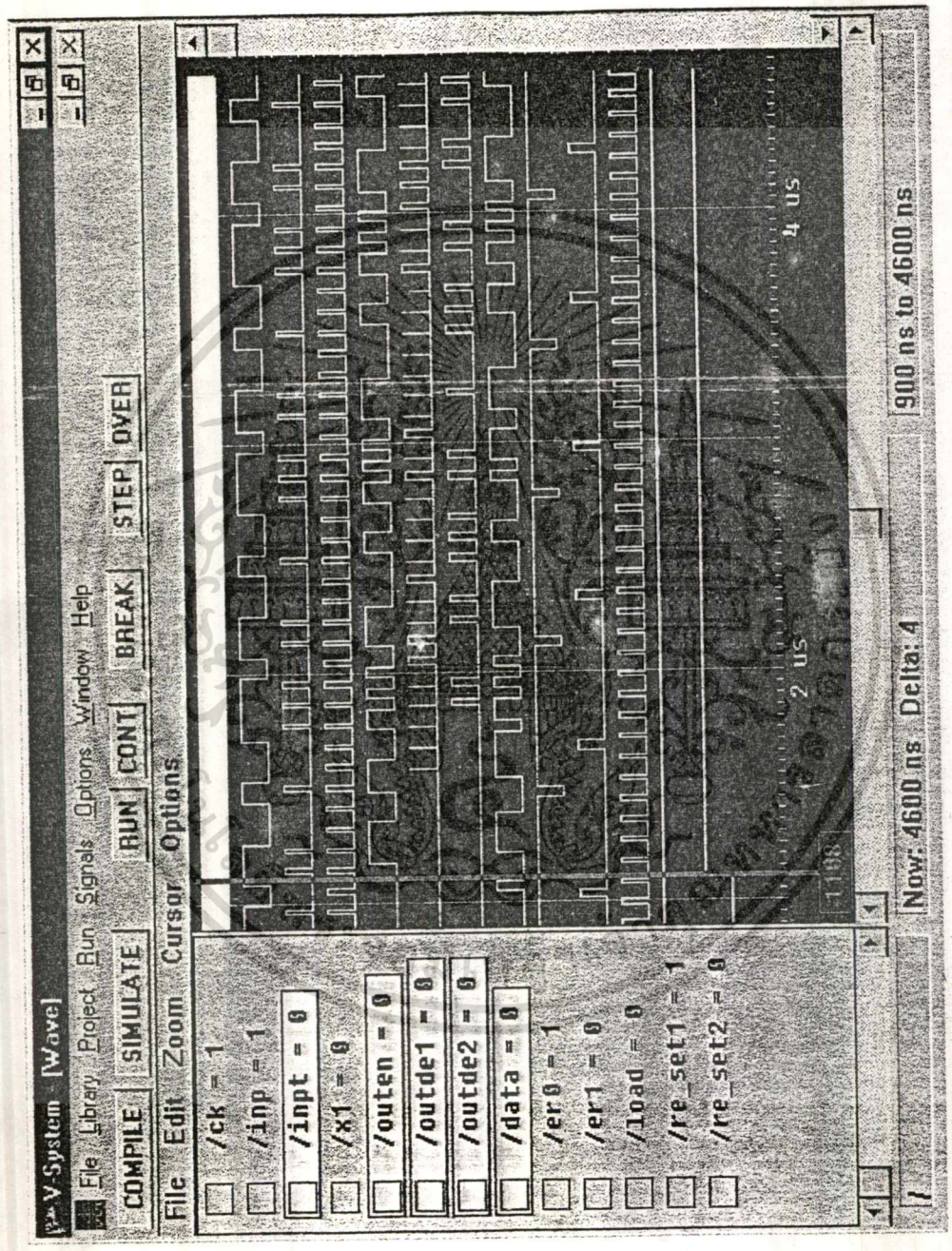
ในส่วนของการทดสอบการทำงานนั้นได้ทำการทดสอบ 2 ส่วนคือ

1.ได้ทำการทดสอบการทำการเข้ารหัสและถอดรหัสของโปรแกรมเข้ารหัสและโปรแกรมถอดรหัส โดยที่ไม่มีข้อผิดพลาดเกิดขึ้นระหว่างทำการส่งข้อมูล ซึ่งผลที่จากการทดสอบนั้นทั้งภาคเข้ารหัสและภาคถอดรหัสทำงานได้ถูกต้องตามที่ต้องการ โดยได้แสดงกราฟของผลจากการทำงานทั้งสองภาคดังในรูปที่ 6.1 โดยจากรูป 6.1 อธิบายได้ว่า การทำงานของโปรแกรมจะทำงานก็ต่อเมื่อมีการ เช็คสถานะการทำงานเรียบร้อยแล้วโดยดูจากขา reset 2 จะมีค่าเป็น หนึ่ง ซึ่งอินพุทของภาคเข้ารหัสจะมีค่าเป็น 1100100111011 ซึ่งภาคเข้ารหัสจะเข้ารหัสและให้เอาท์พุทออกมาเป็น 11 11 00 01 11 00 01 10 10 10 01 11 10 00 ซึ่งค่านี้จะเป็นอินพุทให้แก่ ภาคถอดรหัส ซึ่งจะถอดรหัสได้ออกมาเป็น 1100100111011 ซึ่งตรงกับอินพุทของภาคเข้ารหัสนั้นเองซึ่งแสดงให้เห็นว่าการทำงานของภาคถอดรหัสถูกต้อง

2.ได้ทำการทดสอบการเข้ารหัสและถอดรหัสโดยที่ได้มีการทำให้เกิดข้อผิดพลาดขึ้นในระหว่างการส่งข้อมูลทำให้อินพุทของภาคถอดรหัสผิดพลาด ซึ่งผลจากการถอดรหัสนั้นภาคถอดรหัสสามารถตรวจได้ว่าเกิดข้อผิดพลาดได้ โดยเอาท์พุทที่หนึ่งของภาคถอดรหัสเกิดเป็นหนึ่งเมื่อมีข้อผิดพลาดขึ้นในการส่งข้อมูล ซึ่งสามารถแสดงผลการทำงานของการเข้ารหัสและการถอดรหัสในขณะเกิดข้อผิดพลาดได้ดังในรูปที่ 6.2 จากรูปที่ 6.2 สามารถอธิบายได้ดังนี้ ในส่วนของภาคเข้ารหัสนั้นมีการทำงานเช่นเดียวกับในรูปที่ 6.1 แต่อินพุทของภาคถอดรหัสจะมีการรับข้อมูลโดยสมมุติให้เกิด error ขึ้นในการส่ง ทำให้ได้อินพุทเป็น 11 11 00 00 11 11 10 11 01 01 00 11 11 ซึ่งในการถอดรหัสนั้นสามารถตรวจสอบการทำงานที่ผิดพลาดไปได้โดยขาเอาท์พุทที่ 1 (outdel) จะมีค่าเป็นหนึ่งซึ่งแสดงให้เห็นว่ามีข้อผิดพลาดเกิดขึ้นกับข้อมูล



รูปที่ 6.1 กราฟแสดงการทำงานของการเข้ารหัสและถอดรหัสโดยไม่มีข้อผิดพลาดเกิดขึ้นในขนาดทำการส่งสัญญาณ



รูปที่ 6.2 กราฟแสดงการทำงานของการเข้ารหัสและถอดรหัสโดยที่เกิดข้อผิดพลาดขึ้นในขนาดทำการส่งสัญญาณ

บทที่ 7 สรุปและวิจารณ์ผลการทำงาน

7.1 สรุปผลการทำงานของโปรแกรมในส่วนเข้ารหัส

7.1.1 จากผลการทำงานได้จากบทที่แล้ว ซึ่งเป็นผลที่ได้จากการจำลองและทดสอบการทำงานของโปรแกรมเข้ารหัสหลังจากการใส่ ดีเลย์เข้าไปแล้วทำให้สามารถจำลองการทำงานของโปรแกรมได้ใกล้เคียงกับความเป็นจริงกับการทำงานของวงจรจริง และอีกทั้งยังสามารถนำโปรแกรมที่ได้จากการออกแบบนี้ ไปทำการสร้างเป็นวงจรรวม โดยจะสามารถลดโอกาสการทำงานผิดพลาดของวงจรได้เนื่องจากได้มีการใส่ค่าดีเลย์เข้าไปแล้ว โดยค่าดีเลย์ขึ้นอยู่กับเทคโนโลยีที่เลือกใช้ขณะทำการออกแบบวงจร ซึ่งจะต้องเป็นเทคโนโลยีที่สอดคล้องกับโรงงานที่จะนำไปทำการสร้าง อีกทั้งหลังจากการใส่ดีเลย์แล้วสามารถทำให้พอประมาณได้ว่าวงจรที่ได้ออกแบบนั้นสามารถทำงานได้ในช่วงความถี่เท่าใด

7.1.2 จากผลการทำงานของภาคเข้ารหัสจะสังเกตุได้ว่ามีการหน่วงเกิดขึ้นระหว่างอินพุทกับเอาท์พุทของภาคเข้ารหัสซึ่งเกิดจากการการหน่วงของชิพท์รีจีสเตอร์ที่ใช้พักข้อมูลก่อนที่จะส่งออกไปยังภาคถอดรหัส

7.1.3 จากรูปที่ 6.1 และ 6.2 จะสังเกตุเห็นได้ว่าสัญญาณเอาท์พุทจากภาคเข้ารหัสจะไม่ส่งให้กับทางอินพุทของภาคถอดรหัสโดยตรง แต่จะมีการหน่วงสัญญาณไว้ช่วงเวลาหนึ่งก่อนที่จะเข้ามายังภาคถอดรหัสทั้งนี้ก็เพื่อให้การทำงานของภาคเข้ารหัสและภาคถอดรหัสทำงานสอดคล้องกันนั้นเอง เนื่องจากทั้งสองภาคมีจังหวะการทำงานที่ไม่เหมือนกันจึงต้องพิจารณาถึงจังหวะการทำงานของทั้งสองภาคด้วย โดยการหน่วงสัญญาณนี้ในทางปฏิบัติอาจใช้ชิพท์รีจีสเตอร์เป็นตัวบัฟเฟอร์ข้อมูลไว้ก็ได้

7.1.4 จากรูปที่ 6.1,6.2 จะสังเกตุเห็นว่าอินพุทจากโปรแกรมทดสอบจะถูกนำมาแอนด์กับสัญญาณ x1 ซึ่งหมายความว่าอินพุทจะถูกโหลดเข้าสู่อินพุทของภาคเข้ารหัสเฉพาะช่วงที่สัญญาณ x1 มีค่าเท่ากับ 1 ทั้งนี้เพราะว่าการทำงานของภาคเข้ารหัสนั้นจะโหลดข้อมูลเข้าเฉพาะช่วง x1 เท่านั้น

7.2 สรุปผลการทำงานของโปรแกรมส่วนถอดรหัส

7.2.1การทำงานของโปรแกรมถอดรหัสหลังจากการใส่ดีเลย์แล้ว (จากในบทที่แล้ว) สังเกตุเห็นได้ว่าเกิด กริด ขึ้นในระหว่างช่วงการเปลี่ยนสถานะซึ่งหากเกิดขึ้นกับอินพุทอาจมีผลทำให้การทำงานของวงจรผิดพลาดได้ แต่หาก กริด เกิดขึ้นทางเอาท์พุทจะไม่ส่งผลต่อการทำงาน ซึ่งวิธีในการแก้กริดนี้อาจแก้โดยใช้แก็บที่จังหวะโหลดในขณะที่สัญญาณโหลดเปลี่ยนสถานะเพื่อไม่ให้เกิดกริดได้

7.2.2 จากผลการทำงานของวงจรถอดรหัสในรูปที่ 6.1,6.2 นั้นจะเห็นได้ว่าสามารถถอด

รหัสได้อย่างถูกต้องและสามารถที่จะตรวจสอบข้อผิดพลาดที่เกิดในขณะส่งข้อมูลได้ ในการทำงานของภาคถอดรหัสในขณะที่ไม่เกิดข้อผิดพลาดระหว่างการส่งข้อมูลนั้น จะสามารถถอดรหัสได้เอาท์พุทตรงกับอินพุทของภาคเข้ารหัสซึ่งแสดงได้ว่าการทำงานที่ได้ถูกต้อง แต่จะสังเกตุได้ว่าสัญญาณเอาท์พุทของภาคถอดรหัสเมื่อเทียบกับสัญญาณอินพุทของทางภาคเข้ารหัสจะถูกหน่วงไป 8 ช่วงจังหวะการโหลด ที่เป็นเช่นนี้ก็เพราะว่าเกิดจากการหน่วงของชิพท์รีจีสเตอร์ที่ใช้ในการพักข้อมูลก่อนที่จะส่งออกไปของภาคเข้ารหัส 1 ช่วงจังหวะการโหลด และจะเกิดการหน่วงของ ชิพท์รีจีสเตอร์ที่ใช้ในการรับข้อมูลของภาคถอดรหัสอีกหนึ่งช่วงจังหวะการโหลด และจะถูกหน่วงเนื่องจาก ชิพท์รีจีสเตอร์ที่ใช้ในส่วนของภาคถอดรหัสอีก 6 จังหวะการโหลดร่วมแล้วจึงเป็น 8 จังหวะการโหลด

7.3 วิจารณ์ผลการดำเนินงาน

จากการศึกษาหลักการทำงานของวงจรเข้ารหัส และถอดรหัสแบบ คูณประสานทำให้ทราบถึงคุณสมบัติของโคดว่ามีหลักการทำงานเช่นใด สามารถนำโคดชนิดนี้มาใช้ประโยชน์ในการรับ และส่งข้อมูลทาง ดิจิตอล รวมทั้งในบางรูปแบบอาจมีคุณสมบัติในการตรวจสอบ และแก้ไขข้อผิดพลาดที่อาจจะเกิดขึ้นในระหว่างการส่งข้อมูล

และจากการศึกษาภาษา วีเอชดีแอล ซึ่งเป็นภาษาที่มีคุณสมบัติในการออกแบบวงจรรวมที่มีขนาดใหญ่ทำให้เราสามารถนำภาษาวีเอชดีแอลมาใช้ในการออกแบบวงจรการเข้ารหัส และถอดรหัสให้มีคุณสมบัติตรงตามจุดประสงค์ที่เราได้กำหนดไว้ โดยเราจะแยกส่วนของวงจรเข้ารหัสและถอดรหัสออกเป็น 2 ชิฟ เพื่อให้มีความสะดวกในการนำไปใช้งานต่อไป

หนังสืออ้างอิง

1. Richard E. Blahat , " Theory and Practic of Erroe Control Codes " ,
Addison - Wesley Publishing company , 1984

2. Jayarm Bhasker, " A VHDL Primer" ,Prentice Hall, 1992

3. บวร ปภัสราทร , ประเสริฐ คันธมานนท์ และสุเมธ อังคะศิกุล , " เทคโนโลยีการออกแบบวงจรรวม" ศูนย์เทคโนโลยีอิเล็คทรอนิคส์และคอมพิวเตอร์แห่งชาติ กระทรวงวิทยาศาสตร์และการพลังงาน หน้า 56- 74 ,2533

กิตติกรรมประกาศ

รายงานนี้สำเร็จลงได้ก็เนื่องจากคำแนะนำในการแก้ไขปัญหาและการให้คำปรึกษาจาก อาจารย์ สมศักดิ์ ชุมช่วย และความช่วยเหลือจากพี่ ๆ และเพื่อน ๆ ที่คอยให้คำแนะนำและความช่วยเหลือต่าง ๆ จึงขอขอบคุณมาในโอกาสนี้

ภาคผนวก

โปรแกรม dff1

```
entity dff1 is
   port ( d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit );
end dff1 ;
architecture dff1s of dff1 is
begin
   process (d,ld)
     begin
       if ld = "01" then
          q <= d ;
          end if ;
   end process;
end dff1s ;
configuration dff1con of dff1 is
   for dff1s
   end for;
end dff1con;
```

โปรแกรม dff2

```
entity dff2 is
   port ( d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit );
end dff2;
architecture dff2s of dff2 is
   begin
       process (d,ld)
        begin
         if ld = "11" then
            q <= d;
         end if ;
       end process;
end dff2s;
configuration dff2con of dff2 is
   for dff2s
   end for;
end dff2con;
```

โปรแกรม dff3

```
entity dff3 is
   port ( d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit ) ;
end dff3 ;
architecture dff3s of dff3 is
   begin
       process (d,ld)
        begin
         if ld = "10" then
            q <= d;
         end if ;
   end process ;
end dff3s ;
configuration dff3con of dff3 is
   for dff3s
   end for;
end dff3con;
```

โปรแกรม d_ff2

```
entity  d_ff2  is
  port ( d : in bit ;load : in bit_vector(0 to 1); q:out bit);
end  d_ff2 ;
architecture d_ffs2 of d_ff2 is
      component dff1
       port ( d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit);
      end component;
      component dff2
       port ( d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit);
      end component;
   signal s1 : bit;
begin
          d1 : dff1 port map (d,load,s1);
          d2 : dff2 port map (s1,load,q);
end d_ffs2;
configuration d_ff2con of d_ff2 is
   for d_ffs2
      for d1 : dff1 use entity work.dff1(dff1s);
      end for;
      for d2 : dff2 use entity work.dff2(dff2s);
      end for;
   end for;
end d_ff2con;
```

โปรแกรม d_ff3

```
entity  d_ff3  is
  port ( d : in bit ;load : in bit_vector(0 to 1); q:out bit);
end  d_ff3 ;
architecture d_ffs3 of d_ff3 is
        component dff1
         port  (d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit);
        end component;
        component dff3
         port (d : in bit ; ld : in bit_vector(0 to 1) ; q : out bit);
        end component;
   signal s1 : bit;
begin
        d1 : dff3 port map (d,load,s1);
        d2 : dff1 port map (s1,load,q);
end d_ffs3;
configuration d_ff3con of d_ff3 is
   for d_ffs3
      for d1 : dff3 use entity work.dff3(dff3s);
      end for;
      for d2 : dff1 use entity work.dff1(dff1s);
      end for;
   end for;
end d_ff3con;
```

โปรแกรม s_reg5

```
entity s_reg5 is
   port (d,x : in bit ;l : in bit_vector(0 to 1); q : out bit_vector(0 to 4));
 end s_reg5 ;
architecture  s_reg5s of s_reg5 is
        component d_ff2
           port (d : in bit ;load : in bit_vector(0 to 1); q :out bit );
        end component;
      signal s : bit_vector( 0 to 4);
        begin
            d1 : d_ff2 port map (d,l,s(0));
            d2 : d_ff2 port map (s(0),l,s(1));
            d3 : d_ff2 port map (s(1),l,s(2));
            d4 : d_ff2 port map (s(2),l,s(3));
            d5 : d_ff2 port map (s(3),l,s(4));
            q(0) <= s(0) and x;
            q(1) <= s(1) and x;
            q(2) <= s(2) and x;
            q(3) <= s(3) and x;
            q(4) <= s(4) and x;
end s_reg5s;

configuration s_reg5con of s_reg5 is
   for s_reg5s
      for all : d_ff2 use entity work.d_ff2(d_ffs2);
      end for;
   end for;
end s_reg5con;
```

โปรแกรม st_out

```
entity st_out is
  port ( d1,d2 : in bit ;l2 : in bit_vector(0 to 1) ; data : out bit );
end st_out;
architecture st_outs of st_out is
        component d_ff3
        port ( d : in bit  ;load : in bit_vector( 0 to 1) ; q : out bit );
        end component;
     signal s1,s2 : bit;
begin
          df1 : d_ff3 port map (d1,l2,s1);
          df2 : d_ff3 port map (s2,l2,data);
          s2 <= s1 or d2 ;
end st_outs;

configuration st_outcon of st_out is
   for st_outs
      for all : d_ff3 use entity work.d_ff3(d_ffs3);
      end for;
   end for;
end st_outcon;
```

โปรแกรม sta_en1

```
entity n_staen is
  port ( clk,reset : in bit ;x,load : out bit; load1,load2 : out bit_VECTOR(0 to 1));
end n_staen ;
architecture n_staens of n_staen is
  type state is (st0,st1,st2,st3,st4,st5,st6,st7,st8);
  signal p_sta,n_sta :state ;
begin
    a : process(clk)
    begin
     if clk = '1' then
        p_sta <= n_sta ;
     end if;
    end process a ;
    b : process (p_sta,reset)
    begin
     if reset = '0' then
      n_sta <= st0 ;
      load1 <= "00";
      load2 <= "00";
      x <= '0' ;   load <= '0' ;
    elsif reset = '1' then
    case p_sta is
      when st0 =>
      load1 <= "00";
      load2 <= "00";
      x <= '0' ;          load <= '0';
      n_sta <= st1;
      when st1 =>
        load1 <= "00";
        load2 <= "01";
        x <= '0' ;
```

```
    n_sta <= st2;     load <= '0' ;
  when st2 =>
    load1 <= "01";
    load2 <= "10";
    x <= '1';         load <= '0' ;
    n_sta <= st3;
  when st3 =>
    load1 <= "01";
    load2 <= "11";
    x <= '1' ;        load <= '0' ;
    n_sta <= st4;
  when st4 =>
    load1 <= "00";
    load2 <= "00";
    x <= '0' ;        load <= '0' ;
    n_sta <= st5;
  when st5 =>
    load1 <= "10";
    load2 <= "00";
    x <= '0' ;        load <= '1' ;
    n_sta <= st6;
  when st6 =>
    load1 <= "10";
    load2 <= "01";
    x <= '0';         load <= '1';
    n_sta <= st7;
  when st7 =>
    load1 <= "11";
    load2 <= "10";
    x <= '0' ;        load <= '0' ;
    n_sta <= st8;
  when st8 =>
```

```
            load1 <= "11";
            load2 <= "11";
            x <= '0' ;          load <= '0' ;
            n_sta <= st0;
        end case ;
        end if;
      end process b ;
end n_staens ;

configuration n_staencon of n_staen is
  for n_staens
  end for;
end n_staencon;
```

โปรแกรม encode1

```
entity n_en1 is
      port ( c,reset1,reset2,d : in bit ; data : out bit );
end n_en1 ;
architecture  n_en1s of n_en1 is
      component n_staen
        port ( clk,reset : in bit ; x,load : out bit ; load1,load2 : out bit_vector(0 to 1));
      end component;
      component st_out
        port ( d1,d2 : in bit ; l2 : in bit_vector(0 to 1) ; data : out bit );
      end component ;
      component s_reg5
        port ( d,x : in bit  ; l : in bit_vector(0 to 1) ;q : out bit_vector(0 to 4) );
      end component ;
  signal s5 : bit_vector(0 to 4);
  signal ld1,ld2 : bit_vector(0 to 1) ;
  signal s1,s2,s3,s4,x1,sd,load : bit ;
begin
      s_reg : s_reg5 port map (sd,x1,ld1,s5);
      s_out : st_out port map (s4,s1,ld2,data);
      sta_1 : n_staen port map (c,reset1,x1,load,ld1,ld2);
      sd <= d and reset2 ;
      s1 <= sd and x1;
      s2 <= s1 xor s5(2) ;
      s3 <= s5(3) xor s5(4) ;
      s4 <= s2 xor s3 ;
end n_en1s;

configuration n_en1con of n_en1 is
  for n_en1s
    for s_reg : s_reg5 use entity work.s_reg5(s_reg5s);
    end for;
```

```
    for s_out : st_out use entity work.st_out(st_outs);
    end for;
    for sta_1 : n_staen use entity work.n_staen(n_staens);
    end for;
  end for;
end n_enicon;
```

โปรแกรม de_c2

```
entity de_c2 is
   port (data,l1 : in bit ; ss2 : in bit_vector(0 to 2) ; o1,o2 : out bit );
end de_c2 ;
architecture de_c2s of de_c2 is
        component s_in
        port (d,l_d1 : in bit ; load : in bit_vector(0 to 1) ; o1,o2 : out bit );
        end component ;
        component s_reg5
        port (d,x : in bit ; l : in bit_vector(0 to 1) ; q : out bit_vector(0 to 4));
        end component ;
        component d_ff2
        port (d : in bit ; load : in bit_vector(0 to 1) ; q : out bit);
        end component ;
   signal ss1 : bit_vector(0 to 4);
   signal s1,s2,s3,s4,s5,s6,da,i1,i2 : bit;
begin
        s_i1   : s_in   port map (da,l1,ss2(1 to 2),s1,s2);
        s_re5  : s_reg5 port map (s2,l1,ss2(0 to 1),ss1);
        d_f2   : d_ff2  port map (ss1(4),ss2(0 to 1),s6);
        i2 <= s2 and l1 ;
        i1 <= s1 and l1 ;
        s3 <= i2 xor ss1(2) ;
        s4 <= ss1(3) xor ss1(4) ;
        s5 <= s3 xor s4 ;
        o1 <= i1 xor s5 ;
        o2 <= s6 and l1 ;
        da <= data and l1 ;
end de_c2s ;
```

```
configuration de_c2con of de_c2 is
   for de_c2s
      for s_i1 : s_in use entity work.s_in(s_ins);
      end for;
      for s_re5 : s_reg5 use entity work.s_reg5(s_reg5s);
      end for;
      for d_f2 : d_ff2 use entity work.d_ff2(d_ffs2);
      end for;
   end for;
end de_c2con;
```

โปรแกรม s_reg3

```
entity s_reg3 is
   port (d,x : in bit ;l : in bit_vector(0 to 1); q : out bit_vector(0 to 2));
end s_reg3 ;
architecture  s_reg3s of s_reg3 is
        component d_ff2
           port (d : in bit ;load : in bit_vector(0 to 1); q :out bit );
        end component;
      signal s : bit_vector( 0 to 2);
        begin
            d1 : d_ff2 port map (d,l,s(0));
            d2 : d_ff2 port map (s(0),l,s(1));
            d3 : d_ff2 port map (s(1),l,s(2));
            q(0) <= s(0) and x;
            q(1) <= s(1) and x;
            q(2) <= s(2) and x;
end s_reg3s;

configuration s_reg3con of s_reg3 is
   for s_reg3s
      for all : d_ff2 use entity work.d_ff2(d_ffs2);
      end for;
   end for;
end s_reg3con;
```

โปรแกรม s_in

```
entity s_in is
   port ( d,l_d1 : in bit ; load : in bit_vector(0 to 1) ; o1,o2 : out bit);
end s_in ;
architecture s_ins of s_in is
        component d_ff3
        port ( d : in bit ; load : in bit_vector(0 to 1) ; q : out bit);
          end component;
  signal s1 : bit;
begin
          d1 : d_ff3 port map (d,load,s1);
          d2 : d_ff3 port map (s1,load,o2);
          o1 <= s1 ;
end s_ins ;

configuration s_incon of s_in is
   for s_ins
      for all : d_ff3 use entity work.d_ff3(d_ffs3);
      end for;
   end for;
end s_incon;
```

โปรแกรม sys_cor

```
entity sys_cor is
        port ( load : in bit_vector(0 to 1); s :in bit_vector(0 to 5) ; cor : out bit);
end sys_cor;
architecture sys_cors of sys_cor is
        begin
         process (load,s)
           begin
           if load = "01" then
           case s is
              when "111001"|"111000"|"001011"|"011001"|
              "111011"|"011101"|"111101"|"110001"|"101001" => cor <= '1' ;
              when others => cor <= '0' ;
           end case;
          else cor <= '0'  ;
          end if;
         end process;
end sys_cors;

configuration sys_corcon of sys_cor is
   for sys_cors
   end for;
end sys_corcon;
```

โปรแกรม sys_uncor

```
entity sys_uncor is
        port ( load : in bit_vector(0 to 1); s :in bit_vector(0 to 5) ; uncor : out bit);
end sys_uncor;
architecture sys_uncors of sys_uncor is
        begin
         process (load,s)
           begin
           if load = "01" then
           case s is
              when "000001"|"110011"|"000011"|"100101"|
              "000101"|"001001"|"010001"|"100001"|"111000" => uncor <= '1' ;
              when others => uncor <= '0' ;
           end case;
          else uncor <= '0'  ;
          end if;
         end process;
end sys_uncors;

configuration sys_uncorcon of sys_uncor is
   for sys_uncors
   end for;
end sys_uncorcon;
```

โปรแกรม check_er2

```
entity check_er2 is
   port (d,l1 : in bit ; ld : in bit_vector(0 to 2) ; o1,o2 : out bit );
end check_er2 ;
architecture check_er2s of check_er2 is
        component sys_cor
          port (load : in bit_vector(0 to 1) ; s : in bit_vector(0 to 5) ; cor : out bit);
        end component;
        component sys_uncor
          port (load : in bit_vector(0 to 1) ; s : in bit_vector(0 to 5) ; uncor : out bit);
        end component;
        component s_reg3
          port (d,x : in bit ; l : in bit_vector(0 to 1) ; q : out bit_vector(0 to 2));
        end component;
        component d_ff2
          port (d : in bit ; load : in bit_vector(0 to 1) ; q : out bit);
        end component;
  signal ss1 : bit_vector(0 to 5);       signal s1,s2,s3,s4,sc,suc : bit;
begin
        d1  : d_ff2 port map (d,ld(0 to 1),ss1(0));
        d2  : d_ff2 port map (s1,ld(0 to 1),ss1(1));
        d3  : d_ff2 port map (s2,ld(0 to 1),ss1(2));
        s_reg : s_reg3 port map (s3,l1,ld(0 to 1),ss1(3 to 5));
        s_cor : sys_cor port map (ld(0 to 1),ss1,sc);
        s_unc : sys_uncor port map (ld(0 to 1),ss1,suc);
        s1 <= ss1(0) xor sc;
        s2 <= ss1(1) xor sc;
        s3 <= ss1(2) xor sc;
        s4 <= ss1(5) xor sc;
        o1 <= s4 xor suc;
        o2 <= sc ;
 end check_er2s;
```

```
configuration check_er2con of check_er2 is
  for check_er2s
    for all : d_ff2 use entity work.d_ff2(d_ffs2);
    end for;
    for s_reg : s_reg3 use entity work.s_reg3(s_reg3s);
    end for;
    for s_cor : sys_cor use entity work.sys_cor(sys_cors);
    end for;
    for s_unc : sys_uncor use entity work.sys_uncor(sys_uncors);
    end for;
  end for;
end check_er2con;
```

โปรแกรม sta_in

```
entity n_stain is
  port ( clk,reset : in bit ;l_data1,l_data2,l_ld3,l_ld4 : out bit ; load : out bit_vector(0 to 2) );
end n_stain ;
architecture  n_stains of n_stain is
   type stat is ( st0,st1,st2,st3,st4,st5,st6,st7,st8 );
   signal p_sta,n_sta : stat ;
begin
   a : process (clk)
      begin
       if clk = '1' then
         p_sta <= n_sta ;
       end if;
      end process a ;
   b : process ( p_sta,reset)
      begin
       if reset = '0' then
       n_sta <= st0;
       load <= "000" ;
       l_data1 <= '0';        l_ld4 <= '0' ;
       l_data2 <= '0';        l_ld3 <= '0' ;
      elsif reset = '1' then
       case p_sta is
         when st0 =>
          load <= "000" ;
          l_data1 <= '0';       l_ld4 <= '0' ;
          l_data2 <= '0';       l_ld3 <= '0' ;
          n_sta <= st1 ;
         when st1 =>
          load <= "001";
          l_data1 <= '0';       l_ld4 <= '0' ;
          l_data2 <= '0';       l_ld3 <= '0' ;
```

```
n_sta <= st2 ;
when st2 =>
load <= "010" ;
l_data1 <= '1';        l_ld4 <= '1' ;
l_data2 <= '1';        l_ld3 <= '0' ;
n_sta <= st3 ;
when st3 =>
load <= "011";
l_data1 <= '1';        l_ld4 <= '1' ;
l_data2 <= '1';        l_ld3 <= '0' ;
n_sta <= st4 ;
when st4 =>
load <= "000" ;
l_data1 <= '0';        l_ld4 <= '0' ;
l_data2 <= '1';        l_ld3 <= '0' ;
n_sta <= st5 ;
when st5 =>
load <= "100";
l_data1 <= '0';        l_ld4 <= '0' ;
l_data2 <= '0';        l_ld3 <= '1' ;
n_sta <= st6 ;
when st6 =>
load <= "101" ;
l_data1 <= '0';        l_ld4 <= '0' ;
l_data2 <= '0';        l_ld3 <= '1' ;
n_sta <= st7 ;
when st7 =>
load <= "110";
l_data1 <= '1';        l_ld4 <= '0' ;
l_data2 <= '0';        l_ld3 <= '0' ;
n_sta <= st8 ;
when st8 =>
```

```
        load <= "111";
        l_data1 <= '1';        l_ld4 <= '0' ;
        l_data2 <= '0';        l_ld3 <= '0' ;
        n_sta <= st0 ;
      end case;
     end if;
    end process b ;
end n_stains ;

configuration n_staincon of n_stain is
  for n_stains
  end for;
end n_staincon;
```

โปรแกรม decode1

```
entity n_dec1 is
  port (d,c,reset1,reset2 : in bit ; o : out bit );
end n_dec1 ;
architecture n_dec1s of n_dec1 is
      component n_stain
       port (clk,reset : in bit; l_data1,l_data2,l_ld3,l_ld4 : out bit ; load : out bit_vector(0 to 2));
      end component;
      component de_c2
       port (data,l1,l2 : in bit;  ss2 : in bit_vector(0 to 2); o1,o2 : out bit );
      end component;
signal sld : bit_vector(0 to 2);
 signal s1,s2,s3,s4,s5,sl1,sl2,sd,sre1,sre2,l_ld3,l_ld4 : bit;
begin
      sta : n_stain   port map (c,reset1,sl1,sl2,l_ld3,l_ld4,sld);
      dec : de_c2     port map (sd,sl1,sl2,sld,s1,s2);
sd <= d and reset2 ;
s5 <= s2 and sl2;
       o <= s5  ;
end n_dec1s;

configuration n_dec1con of n_dec1 is
  for n_dec1s
    for sta : n_stain use entity work.n_stain(n_stains);
    end for;
    for dec : de_c2 use entity work.de_c2(de_c2s);
    end for;
 end for;
 end n_dec1con;
```

โปรแกรมที่ใช้ในการทดสอบการเข้ารหัสและถอดรหัสโดยที่มีข้อผิดพลาดเกิดขึ้นระหว่างการส่งสัญญาณ

```
library ieee;
use ieee.std_logic_1164.all;
library xi4;
use xi4.all;
entity n_t_de2 is
end n_t_de2 ;
architecture n_test of n_t_de1 is
 component  n_en1
 port (c : in std_logic; reset1 : in std_logic;
     reset2 : in std_logic; d :in std_logic ;data : out std_logic);
 end component;
 component  n_dec1
 port (d : in std_logic ; c : in std_logic ; reset1 : in std_logic ;
     reset2 :in std_logic ; o2 : out std_logic ; o1 : out std_logic);
 end component;
 component  n_staen
 port (reset :in std_logic ; clk : in std_logic ; load2_1 : out std_logic;
    load2_0 : out std_logic; load1_1 : out std_logic ; load1_0 : out std_logic;
    load : out std_logic ; x : out std_logic);
 end component;
  signal ck,inp,inpt,x1,outen,outde1,outde2 : std_logic := '0' ;
  signal data : std_logic := '0' ;
  signal ld1_0,ld1_1,ld2_0,ld2_1,load : std_logic := '0' ;
  signal re_set1,re_set2 : std_logic  ;
begin
      encode : n_en1 port map (ck,re_set1,re_set2,inp,outen);
      decode : n_dec1 port map (data,ck,re_set1,re_set2,outde2,outde1);
      sta    : n_staen  port map (re_set1,ck,ld2_1,ld2_0,ld1_1,ld1_0,load,x1);
      data <= outen  after 40 ns;
```

```
    inpt  <= inp and x1 ;
    ck <= not ck  after 5 ns;
    T : process
    begin
     inp <= '0' ;
     wait for 90 ns;
     inp <= '1';
     wait  for  180 ns;
     inp <= '0';
     wait for 180 ns;
     inp <= '1';
     wait for 90 ns;
     inp <= '0' ;
     wait for 180 ns;
     inp <= '1' ;
     wait for 270 ns;
    end process T;
res : process
     begin
      re_set1 <= '0' ;
      re_set2 <= '0' ;
      wait for 40 ns;
      re_set1 <= '1';
      re_set2 <= '0';
      wait for 1080 ns;
      re_set1 <= '1';
      re_set2 <= '1';
      wait for 4000 ns;
     end process res ;

end n_test;
```

โปรแกรมที่ใช้ในการทดสอบการเข้ารหัสและถอดรหัสโดยที่ไม่มีข้อผิดพลาดระหว่างการส่งสัญญาณ

```
library ieee;
use ieee.std_logic_1164.all;
library xi4;
use xi4.all;
entity n_t_de1 is
end n_t_de1 ;
architecture n_test of n_t_de1 is
 component  n_en1
 port (c : in std_logic; reset1 : in std_logic;
      reset2 : in std_logic; d :in std_logic ;data : out std_logic);
 end component;
 component  n_dec1
 port (d : in std_logic ; c : in std_logic ; reset1 : in std_logic ;
      reset2 :in std_logic ; o2 : out std_logic ; o1 : out std_logic);
 end component;
 component  n_staen
 port (reset :in std_logic ; clk : in std_logic ; load2_1 : out std_logic;
     load2_0 : out std_logic; load1_1 : out std_logic ; load1_0 : out std_logic;
     load : out std_logic ; x : out std_logic);
 end component;
 signal ck,inp,inpt,x1,outen,outde1,outde2 : std_logic := '0' ;
 signal data,er0,er1 : std_logic := '0' ;
 signal ld1_0,ld1_1,ld2_0,ld2_1,load : std_logic := '0' ;
 signal re_set1,re_set2 : std_logic  ;
begin
      encode : n_en1 port map (ck,re_set1,re_set2,inp,outen);
      decode : n_dec1 port map (data,ck,re_set1,re_set2,outde2,outde1);
      sta    : n_staen  port map (re_set1,ck,ld2_1,ld2_0,ld1_1,ld1_0,load,x1);
      data <= ((outen and er0) or er1 ) after 40 ns;
```

```
inpt  <= inp and x1 ;
ck <= not ck  after 5 ns;
T : process
begin
 inp <= '0' ;
 wait for 90 ns;
 inp <= '1';
 wait  for  180 ns;
 inp <= '0';
 wait for 180 ns;
 inp <= '1';
 wait for 90 ns;
 inp <= '0' ;
 wait for 180 ns;
 inp <= '1' ;
 wait for 270 ns;
end process T;
er : process
 begin
  er0 <= '1' ;
  er1 <= '0' ;
  wait for 167 ns;
  er0 <= '0' ;
  er1 <= '0' ;
  wait for 40 ns;
  er0 <= '1';
  er1 <= '0';
  wait for 160 ns;
  er0 <= '1';
  er1 <= '1';
  wait for 40 ns;
  er1 <= '0';
```

```
        wait for 200 ns;
        er1 <= '0';
        wait for 40 ns;
    end process er;
    res : process
    begin
     re_set1 <= '0' ;
     re_set2 <= '0' ;
     wait for 40 ns;
     re_set1 <= '1';
     re_set2 <= '0';
     wait for 1080 ns;
     re_set1 <= '1';
     re_set2 <= '1';
     wait for 4000 ns;
    end process res ;

end n_test;
```