การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วิธีการแปลงกลับและการแปลงในสนามจำกัด)

VERY LARGE SCALE INTEGRATED CIRCUIT DESIGN USING VHDL

(FINITE FIELD INVERSE / FORWARD TRANSFORM MACHINE)

โดย

นางสาวพัชรี เมฆโพธิ์

นายสมพงษ์ ศรีเมธีพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็คทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วิธีการการแปลงกลับและการแปลงในสนามจำกัด)

VERY LARGE SCALE INTEGRATED CIRCUIT DESIGN USING VHDL

(FINITE FIELD INVERSE / FORWARD TRANSFORM MACHINE)

โดย

นักศึกษา

นางสาว พัชรี เมฆโพธิ์ รหัส 36014287

นาย สมพงษ์ ศรีเมธีพงษ์ รหัส 36014460

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ ชุมช่วย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็คทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

รายงาน ปีการศึกษา ๒๕๓๕

ภาควิชา อิเล็คทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วิธีการการแปลงกลับและการแปลงในสนามจำกัด)

ผู้จัดทำ

1. นางสาว พัชรี เมฆโพธิ์ รหัส 36014287
2. นาย สมพงษ์ ศรีเมธีพงษ์ รหัส 36014460

อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ ชุมช่วย)

การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล

(วิธีการแปลงกลับและการแปลงในสนามจำกัด)

VERY LARGE SCALE INTEGRATED CIRCUIT DESIGN USING VHDL

(FINITE FIELD INVERSE / FORWARD TRANSFORM MACHINE)

1. นางสาว พัชรี เมฆโพธิ์ รหัส 36014287
2. นาย สมพงษ์ ศรีเมธีพงษ์ รหัส 36014460

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ ชุมช่วย)

การออกแบบวงจรรวมขนาดใหญ่มากโดยใช้ภาษาวีเอชดีแอล
(เรื่องวิธีการการแปลงกลับและการแปลงในสนามจำกัด)

นางสาว พัชรี เมฆโพธิ์
นาย สมพงษ์ ศรีเมธีพงษ์
ดร. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอวิธีการออกแบบวงจรรวมสำหรับสร้างวงจรเข้ารหัสและถอดรหัส โดยใช้วิธีการแปลงในสนามจำกัดเพื่อลดข้อผิดพลาดในการรับส่งข้อมูลจึงได้นำการแปลงข้อมูลแบบรหัสรีด-โซโลมอน ซึ่งเป็นวิธีการแปลงข้อมูลเชิงความถี่ ในการถอดรหัสแบบนี้ต้องอาศัยการคำนวณเป็นจำนวนมาก โดยเฉพาะอย่างยิ่งการแปลงข้อมูลทั้งไปและกลับ จึงนำวิธีการตัวประกอบปฐม เพื่อแยกข้อมูลในการคำนวณให้มีขนาดเล็กลง และนำมาใช้ร่วมกับวิธีการแปลงข้อมูลจำนวนน้อยๆ เข้ามาช่วยลดเวลาการคำนวณลง โครงสร้างที่ได้จากวิธีตัวประกอบนี้สามารถจะออกแบบเป็นวงจรความถี่สูงได้ จากนั้นจะนำภาษาวีเอชดีแอลช่วยในการออกแบบ โดยใช้คอมพิวเตอร์ช่วยในการจำลองการทำงานและสังเคราะห์เป็นวงจรระดับเกท แล้วจึงนำมาสร้างเป็นฮาร์ดแวร์ โดยการโปรแกรมลงเอฟพีจีเอ

VERY LARGE SCALE INTEGRATED CIRCUIT DESIGN USING VHDL

(FINITE FIELD INVERSE / FORWARD TRANSFORM MACHINE)

Phatcharee Makpo

Sompong Srimeteepong

Dr. Somsak Choomchuay Advisor

1996

ABSTRACT

This thesis presents Very Large Scale Integrated Circuit (VLSI) design for encoding and decoding by using the Finite Field Fast Transform Algorithm . The Reed-Solomon Code which is the frequency-based transform were selected as a design excercise. This method of decoding demands a great amount of calculation, especially in Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) . Data are first decomposed by using Prime Factor Algorithm (PFA) method and then Short Length Algorithm (SLA) method is applied to each dimension transformation to improve the computation speed. The developed algorithm is then mapped into the hardware piece by mean of Very high speed integrated circuit Hardware Description Language (VHDL) and Field Programmable Gate Array (FPGA).

สารบัญ

สารบัญภาพ

บทที่ 1
บทนำ

ความสำคัญและระบบการ Encode และ Decode Reed-Solomon Code โดยวิธีการ Transform

ปัจจุบันเทคโนโลยีการผลิตอุปกรณ์ฮาร์ดแวร์ที่ใช้กันในอุตสาหกรรมการผลิต อุปกรณ์ชิ้นส่วนทางอิเล็คทรอนิคส์และคอมพิวเตอร์ได้มีการพัฒนาและก้าวล้ำไปอย่างมากอุปกรณ์ที่ผลิตขึ้นมีขนาดเล็กแต่มีความซับซ้อนภายในประกอบด้วยวงจรขนาดเล็กจำนวนมากมาย ดังนั้นในการออกแบบวงจรรวมที่มีขนาดใหญ่มาก (VLSI : Very Large Scale Integrated Circuit) มีความซับซ้อนและในขณะที่ขนาดและความซับซ้อนของระบบดิจิตอลเพิ่มมากขึ้นคอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD : Computer Aided Design) ก็ได้ถูกนำมาใช้ในขบวนการออกแบบฮาร์ดแวร์เพิ่มมากขึ้นเช่นกัน อุปกรณ์และวิธีการออกแบบใหม่ๆ ได้ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบและภาษาสำหรับอุปกรณ์ฮาร์ดแวร์ (HDL : Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการในการออกแบบระบบดิจิตอล

ซึ่งปริญญานิพนธ์นี้ได้ทดลองทำการออกแบบวงจรรวมการเข้ารหัสและถอดรหัสข้อมูลสำหรับรหัสแบบรีด-โซโลมอนโดยใช้ภาษา VHDL ช่วยในการออกแบบ ซึ่งรหัสรีด-โซโลมอนนี้เป็นอัลกอริธึมเข้ารหัสและถอดรหัสที่นิยมใช้กันแพร่หลายในการเข้ารหัสและถอดรหัสชนิดหนึ่งและการแปลงในสนามจำกัดก็เป็นขบวนการที่มีประโยชน์มากและใช้มากทั้งในวงจรเข้ารหัสและถอดรหัสแบบรีด-โซโลมอน เพื่อใช้ตรวจหาและแก้ไขข้อมูลที่ผิดพลาด รหัสรีด-โซโลมอนนี้สามารถทำได้ทั้งในเชิงเวลาและความถี่โดยความยุ่งยากในการทำงานจะอยู่ที่การถอดรหัส ซึ่งนิยมทำในเชิงความถี่ เนื่องจากการถอดรหัสในเชิงความถี่นี้จะต้องใช้การแปลง (Transform) และการแปลงกลับ (Inverse Transform) เป็นส่วนสำคัญนอกเหนือไปจากการคำนวณอื่นๆ การแปลงและการแปลงกลับจะต้องใช้การคำนวณเป็นจำนวนมาก เพราะฉะนั้น การแปลงในสนามจำกัด $GF(2^m)$จึง ใช้วิธีการตัวประกอบปฐม (Prime Factor Algorithm) มาใช้เพื่อลดจำนวนการคำนวณลงโดยใช้ร่วมกับวิธีการแปลงข้อมูลจำนวนน้อยๆ (Short Length Algorithm) การนำวิธีการดังกล่าวไปใช้งานจริงจะต้องพิจารณาเวลาที่ใช้ในการแปลงเป็นสำคัญ เพราะเวลาที่ใช้ในการแปลงจะมีผลอย่างมากต่อความเร็วในการแปลงโดยเฉพาะอย่างยิ่งในการรับส่งข้อมูลแบบเวลาจริงเช่น การรับส่งข้อมูลในระบบสื่อสารหรือการเขียนอ่านข้อมูลจำนวนมากในฮาร์ดดิสก์

ในปริญญานิพนธ์ฉบับนี้การออกแบบวงจรรวมจะใช้คอมพิวเตอร์ช่วยออกแบบ โดยใช้ภาษาวีเอชดีแอล (VHDL : VHSIC Hardware Description Language) ซึ่งเป็นภาษาบรรยายการทำงานของฮาร์ดแวร์ของวงจรดิจิตอล โดยตัวภาษาประกอบไปด้วยรายละเอียดและส่วนประกอบต่างๆ ที่ใช้อธิบายพฤติกรรม (Behavioral) หรือโครงสร้าง (Structural) ของระบบโดยพิจารณาฐานเวลา (Timing) ของระบบเป็นหลัก ระบบที่เราออกแบบโดย VHDL นั้น ผู้ออกแบบไม่จำเป็นต้องรู้ถึงวงจรภายในว่ามีอะไรต่อกันอย่างไรบ้าง เพียงแต่กำหนดอินพุท เอาท์พุทและฟังก์ชั่นการทำงานของระบบ เขียนเป็นโปรแกรมแสดงการไหลของข้อมูล (Dataflow), การทำงานของระบบ (Behavioral) หรือโครงสร้างของระบบ (Structural) จากนั้นก็ทำการคอมไพล์ (Compile) และซิมูเลท (Simulate)โมเดล ที่ออกแบบมาเพื่อวิเคราะห์ดูฟังก์ชั่นฐานเวลาของระบบ ว่าตรงตามต้องการหรือไม่ ถ้ามีการแก้ไขอย่างไรก็ทำการแก้ไขซอสโคด (Source Code) และทำการคอมไพล์แล้วซิมูเลทซ้ำๆ จนกว่าจะได้โมเดลที่มีฐานเวลาตามที่ระบบต้องการ จะเห็นว่าไม่ต้องเสียเวลากับการสร้างวงจรทางฮาร์ดแวร์จริงๆ ขึ้นมาทดสอบ ซึ่งจะทำให้เสียเวลาและค่าใช้จ่ายสูง โมเดลที่ออกแบบโดย VHDL สามารถทำการสังเคราะห์ (Synthesis) เพื่อให้ได้ผังวงจร (Schematic Diagram) และนำเอาไปสร้างเป็นวงจรจริงๆ ได้ โดยหากนำเอาไปสร้างเป็น ASIC (Application Specific Integrated Circuit) หรือนำไปเบิร์น (Burn) ลงเอฟพีจีเอ (FPGA : Field Programmable Gate Array) เพื่อนำไปใช้งานจริงๆ ต่อไป

ในการศึกษาภาษา VHDL และอัลกอริธึมที่ช่วยในการออกแบบวงจรรวมนั้น ผู้จัดทำยอมรับว่ายังเป็นเรื่องใหม่และมีข้อผิดพลาดหลายประการ จึงมีความบกพร่องและไม่สมบูรณ์อยู่ทั้งในตัวรายงานและโครงงาน ซึ่งผู้จัดทำยินดีรับคำติชมและปรับปรุงแก้ไขให้ดีขึ้นต่อไป หวังว่าในปริญญานิพนธ์ฉบับนี้จะเป็นประโยชน์ต่อผู้ที่สนใจเพื่อเป็นแนวทางการศึกษาภาษา VHDL ต่อไป

บทที่ 2
วิธีการตัวประกอบปฐม

2.1 บทนำ

การเข้ารหัสและถอดรหัสเป็นขบวนการที่จำเป็นเพื่อเพิ่มความเชื่อถือได้ของระบบระบบส่งข้อมูล โดยขบวนการที่ใช้ในการเข้ารหัสและถอดรหัสนั้น มักจะใช้การแปลงเชิงเลขร่วมด้วยเสมอ ตัวอย่างเช่นการเข้ารหัสและถอดรหัสแบบรีด-โซโลมอนในเชิงความถี่ ทำโดยการนำข้อมูลที่ต้องการส่งมาเติมด้วยศูนย์ให้ครบตามขนาดของบล็อคแล้วนำไปทำการแปลงกลับ (InverseTransform) ผลที่ได้จะถูกส่งผ่านช่องสัญญาณไปยังด้านรับซึ่งจะทำการถอดรหัสข้อมูลโดยการแปลง ซึ่งข้อมูลที่ถูกต้องที่ได้จากระบบส่งสัญญาณนี้จะได้จากข้อมูลที่ได้จากการแปลงบวกด้วยข้อมูลที่ได้จากการแปลงชุดดังกล่าวซึ่งถูกนำไปผ่านขั้นตอนการหาตำแหน่งของข้อมูลที่ผิดพลาดโดยใช้วิธีของ Berlekamp Massy และขบวนการ Recursive extension ซึ่งก็เป็นอีกรูปแบบหนึ่งของการ แบ่งข้อมูล

เมื่อชุดของข้อมูลที่จะทำการเข้าและถอดรหัสมีขนาดใหญ่จะทำให้ใช้เวลาในการแปลงมากขึ้นมาก โดยเฉพาะบล็อคขนาดใหญ่เช่น รหัส RS(255,223) การคำนวณโดยตรงจะเสียเวลามาก การคำนวณแบบเร็วนั้นจะลดขนาดของชุดข้อมูลให้เล็กลงเสียก่อน โดยใช้วิธีการตัวประกอบปฐม (Prime Factor Algorithm, PFA) วิธีการนี้ชุดข้อมูลจะถูกแยกออกเป็นหลายมิติ ซึ่งขนาดในแต่ละมิติจะเล็กลงมาก การแปลงข้อมูลชุดเล็กๆ เหล่านี้ จะใช้วิธีการที่เหมาะสม ที่เรียกว่าการแปลงข้อมูลจำนวนน้อยๆ (Short Length Algorithm, SLA) ดังนั้นเมื่อนำเอาวิธีการตัวประกอบปฐมมาใช้ร่วมกับการแปลงข้อมูลจำนวนน้อยๆ จะช่วยให้ลดเวลาในการคำนวณลงไปได้มาก อีกทั้งโครงสร้างที่ได้จากการใช้วิธีการตัวประกอบปฐมนี้สามารถที่จะเลือกให้เหมาะสมกับการพัฒนาเป็นวงจรรวมที่มีประสิทธิภาพและความเร็วที่สูง เช่นเลือกการทำงานแบบท่อส่ง (pipeline)

ในบทนี้จะนำเสนอเนื้อหาตามลำดับดังนี้ ในหัวข้อที่ 2.2 จะกล่าวถึงการแปลงในสนามจำกัด หัวข้อที่ 2.3 กล่าวถึงวิธีการตัวประกอบปฐม หัวข้อที่ 2.4 แสดงการใช้วิธีการตัวประกอบปฐมกับการแปลงในสนามจำกัดขนาด 15 จุดในสนามจำกัด $GF(2^4)$ หัวข้อที่ 2.5 แสดงวิธีการแปลงข้อมูลจำนวนน้อยๆ และผังการคำนวณต่างๆ

2.2 การแปลงในสนามจำกัด (Finite Field Transform)

การแปลงในสนามจำกัด ถูกใช้มากในการประมวลผลสัญญาณเชิงเลข

(Digital Signal Processing) โดยเฉพาะการเข้ารหัสเพื่อควบคุมความผิดพลาด

การแปลงในสนามจำกัดมีรูปสมการที่ใกล้เคียงกับการแปลงในสนามจำนวนเชิงซ้อน และแสดงได้โดย

$$A_k = \sum_{n=0}^{N-1} a_n \alpha^{\langle nk \rangle_N} \qquad (2.1)$$

โดยที่ α = รากที่ N ของ 1 ($\alpha^N=1$)

n = 0,1,2,...,N-1

k = 0,1,2,...,N-1

เมื่อ N คือจำนวนข้อมูลที่จะทำการแปลงในหนึ่งบล็อค โดยการกระทำทางคณิตศาสตร์ต่างๆ จะทำในสนามจำกัด นอกจากการคำนวณหาค่า nk ซึ่งจะใช้การคำนวณในสนามตัวเลขจำนวนเต็มแต่ต้องทำการหารเต็มหน่วย (modulo) ด้วย N

2.3 วิธีการตัวประกอบปฐม (Prime Factor Algorithm , PFA)

การลดเวลาในการแปลงนั้นจะใช้วิธีลดจำนวนการคูณที่ซ้ำลงโดยใช้วิธีการแปลงข้อมูลจำนวนน้อยๆ หรือโดยการแปลงรูปแบบทางคณิตศาสตร์เช่นวิธีของวิโนแกรด (winograd) แต่สำหรับข้อมูลซึ่งมีขนาดโหญ่วิธีดังกล่าวก็ยังต้องใช้การคำนวณค่อนข้างมากซึ่งวิธีการที่จะช่วยให้ลดการคำนวณลงได้ก็คือการเปลี่ยนโครงสร้างของการแปลงจากหนึ่งมิติไปเป็นหลายมิติโดยใช้การจับคู่ดัชนี (index mapping) ซึ่งวิธีการนี้ถูกนำไปใช้ในหลายรูปแบบเช่นวิธีของ Cooley-Tukey หรือวิธีของ Good-Thomas ซึ่งจะทำการแปลงจาก 1 มิติในเป็น 2 มิติ และวิธีเปิดตารางซึ่งวิธีที่กล่าวมาแล้วนั้นเป็นวิธีที่ไม่มีรูปแบบและยากที่จะขยายผลไปสู่ n มิติ (แต่สามารถทำได้โดยการทำซ้ำซึ่งเป็นขบวนการที่ยุ่งยาก) การจับคู่ดัชนีแบบมีรูปแบบเป็นการจับคู่ที่มีรูปแบบที่แน่นอนและเหมาะสำหรับข้อมูลที่มีขนาดใหญ่โดยเฉพาะการจับคู่รูปแบบเชิงเส้นซึ่งจะกล่าวถึงต่อไปแต่ทั้งนี้การจับคู่ทุกวิธีจะต้องเป็นการจับคู่แบบหนึ่งต่อหนึ่งเท่านั้น

พึงสังเกตุว่าวิธีของ Cooley-tukey เหมาะสมและนิยมใช้กับบล็อคข้อมูลขนาด 2^m เมื่อ m คือ จำนวนเต็มใดๆ อย่างไรก็ตาม เนื่องจากรหัสรีด-โซโลมอนจะมีขนาดของบล็อคเป็น (2^m-1) ใน $GF(2^m)$ การพัฒนาขั้นตอนวิธีจึงมีข้อจำกัดดังแสดงต่อไปนี้

ให้ A และ M เป็นจำนวนเต็มใดๆ และ (A,M) = D จากวิธีของ Euclid จะได้ว่า

$$x'A + y'M = D \tag{2.2}$$

สำหรับบางค่าของ x' และ y'
พิจารณาสมการที่มีตัวแปร x เป็นตัวแปรไม่รู้ค่า

$$\therefore \quad xA \equiv C \bmod M \tag{2.3}$$

ซึ่งเท่ากับ

$$xA + yM = C \tag{2.4}$$

เนื่องจาก D/A และ D/M จึงทำให้ D/(xA+yM) ทุกค่า x และ y ดังนั้นสมการที่ (2.4) จะเป็นจริงก็ต่อเมื่อ D/C นั่นคือ $C=C_1D$ ดังนั้นจากสมการที่ (2.2) ทำให้เราสามารถเขียนได้ว่า $C=(C_1x')A+(C_1y')M$ ซึ่งเมื่อเปรียบเทียบกับสมการที่ (2.4) จะทำให้ได้ว่า $x=C_1x'$ จะเห็นได้ว่าถ้า D=1 ซึ่งหมายถึง (A,M)=1 จะทำให้สมการที่ (2.3) มีคำตอบเพียงหนึ่งเดียวและสมการที่ (2.4) เป็นจริงทุกค่า x,y และ C เป็นจำนวนเต็มใดๆ แต่ถ้า D=1 และ C=1 เราจะเรียก x ว่าส่วนกลับ (inverse) ของ A ซึ่งเขียนเป็น A^{-1} จะเห็นว่า A จะมีส่วนกลับก็ต่อเมื่อ (A,M)=1 เท่านั้น และเขียนได้ว่า $\langle AA^{-1}\rangle_M=1$ และในทำนองเดียวกัน สำหรับ $(A_1,A_2,...,A_h)=D$ โดยวิธีของ Euclid จะได้ว่า

$$\sum_{r=1}^{h} x_r' A_r = D \tag{2.5}$$

ดังนั้น

$$\sum_{r=1}^{h} x_r A_r = C \tag{2.6}$$

จะเป็นจริงทุกค่า X_r และ C เป็นจำนวนเต็มใดๆ และมีคำตอบพียงหนึ่งเดียวถ้า D=1

ให้ x(n) คืออันดับอินพุทและ Y(k) คืออันดับเอาท์พุทโดย n และ k เป็นดัชนีของอันดับ และ $0 \le n \le N-1$, $0 \le k \le N-1$ และ N เป็นผลคูณของจำนวนปฐมซึ่งแสดงได้ดังนี้

$$N = \prod_{r=1}^{h} N_r \qquad (2.7)$$

โดย $(N_i,N_j)=1$ เมื่อ $0<i<h$, $0<j<h$ และ $i \neq j$

พิจารณาสมการ

$$\left(n_1, n_2, \ldots, n_h, N_i\right) = D \qquad ; 0<i<h \text{ , } 0<n_i<N_i-1 \qquad (2.8)$$

เนื่องจาก N_i เป็นจำนวนปฐม ดังนั้นค่า D ในสมการที่ (2.8) จึงเท่ากับ 1 เมื่อเทียบเคียงกับสมการที่ (2.3) จึงได้ว่า

$$n = \left(\sum_{r=1}^{h} x_r n_r\right) \bmod N_i \qquad (2.9)$$

เป็นจริงทุกค่าจำนวนเต็ม x_r และ n ใดๆ ถ้าเราให้ x_r เป็นค่าใดๆ ก็ได้เพื่อให้สมการที่ (2.9) เป็นจริง ถ้าเราให้ $x_r = M_r$ โดย $M_r = N/N_r = N_1N_2...N_{r-1}N_{r+1}...N_h$ ซึ่งจะได้ว่า $M_j = 0 \bmod N_i$; $i \neq j$

ดังนั้น

$$\begin{aligned} n &= \left(\sum_{r=1}^{h} x_r n_r\right) \bmod N_i \\ &= \left(M_i n_i\right) \bmod N_i \qquad ;1<i<h \qquad (2.10) \end{aligned}$$

จากทฤษฎีจำนวนถ้า $a = b \bmod N_1$, $a = b \bmod N_2$, ... , $a = b \bmod N_r$ เมื่อ $N=N_1...N_i...N_h$ โดย N_i เป็นจำนวนปฐมซึ่งกันและกัน จะได้ $a = b \bmod N$

ดังนั้นสมการที่ (2.10) จึงเขียนได้เป็น

$$n = \left(\sum_{r=1}^{h} M_r n_r\right) \bmod N \qquad ;M_r=N/N_r \qquad (2.11)$$

จะเห็นว่าสมการที่ (2.11) จะเป็นการหาค่า n_i ที่แทนค่า n จึงเรียกว่าการจับคู่ดัชนีอินพุทและโดยที่ $(N_i,N_j)=1$, $i \neq j$ จึงเรียกว่าวิธีการตัวประกอบปฐม

ในการจับคู่ดัชนีเอาท์พุทเรามี k_i เป็นจำนวนเต็มโดยที่ $0 \leq k_i \leq N_i-1$, $0 \leq k \leq N-1$ เราจะหาค่า k ซึ่งสอดคล้องตามสมการ

$$k = k_i \bmod N_i \quad ; 0 \leq i \leq h \, , 0 \leq k_i \leq N_i-1 \qquad (2.12)$$

เมื่อ $N=N_1 \ldots N_i \ldots N_h$ โดย N_i เป็นจำนวนปฐมซึ่งกันและกัน พิจารณาสมการ

$$\left(k_1, k_2, \ldots, k_h, N_i\right) = 1 \quad ; 0 \leq i \leq h$$

เช่นเดียวกับสมการที่ (2.9) ดังนั้น

$$k = \left(\sum_{r=1}^{h} x_r k_r \right) \bmod N_i$$

ถ้า

$$x_i = 0 \bmod N_j \quad ; \; i \neq j$$
$$= 1 \bmod N_j \quad ; \; i = j$$

จะได้ $k = k_i \bmod N_i$ ซึ่งเท่ากับสมการที่ (2.12) จะเห็นว่า

$$\left\langle M_i M_i^{-1} \right\rangle_{N_j} = 0 \quad ; \; i \neq j$$
$$= 1 \quad ; \; i = j$$

ดังนั้น $x_i = M_i M_i^{-1}$ ซึ่งเช่นเดียวกับสมการที่ (2.10) ทำให้ได้ว่า

$$k = \left(\sum_{r=1}^{h} M_r M_r^{-1} k_r \right) \bmod N \qquad (2.13)$$

จะเห็นได้ว่าสมการที่ (2.11) และสมการที่ (2.12) จะทำการจับคู่ n และ k ไปสู่กลุ่ม n_r และ k_r โดยเป็นการจับคู่แบบหนึ่งต่อหนึ่ง ดังนั้นจะเห็นได้ว่าอันดับ 1 มิติ จะถูกเปลี่ยนไปเป็นอันดับใน h มิติ

ในทางปฏิบัติเราจะแยกอันดับออกมาทีละ 1 มิติ โดยให้ N^i แทนการจับคู่ดัชนีอินพุทของมิติที่ I และ K^i แทนการจับคู่ดัชนีเอาท์พุทมิติที่ i ซึ่งจากสมการที่ (2.11) จะได้ว่า

$$N^1 = n = \left\langle \sum_{r=1}^{h} M_r n_r \right\rangle_N = \left\langle M_1 n_1 + \sum_{r=2}^{h} M_r n_r \right\rangle_N \tag{2.14}$$

หลังจากที่ a_n ในมิติที่ 1 ถูกแยกออก และถูกแปลงโดยใช้วิธีการแปลงข้อมูลจำนวนน้อยๆ แล้วก็จะถูกใส่กลับไปไว้ในอันดับอินพุทโดยใช้สมการที่ (2.13) ดังนี้

$$K^1 = k = \left\langle M_1 M_1^{-1} n_1 + \sum_{r=2}^{h} M_r n_r \right\rangle_N \tag{2.15}$$

จากสมการที่ (2.14) และสมการที่ (2.15) ซึ่งแสดงการจับคู่ดัชนีอินพุทและเอาท์พุทของมิติที่ 1 จะเห็นได้ว่าถ้า เรากำหนดให้ $n_2, n_3, \ldots, n_h$ มีค่าคงที่ เราจะได้กลุ่มของดัชนี n และ k จากสมการที่ (2.14) และสมการที่ (2.15) เป็นกลุ่มเดียวกัน แต่อาจจะมีการสลับตำแหน่งกัน และจากสมการที่ (2.14) และสมการที่ (2.15) สามารถเขียนการจับคู่ดัชนีอินพุทและเอาท์พุทของมิติต่างๆ ให้อยู่ในรูปทั่วไปได้เป็น

$$N^i = \left\langle \sum_{r=1}^{i-1} M_r M_r^{-1} k_r + M_i n_i + \sum_{r=i+1}^{h} M_r n_r \right\rangle_N \tag{2.16}$$

$$K^i = \left\langle \sum_{r=1}^{i-1} M_r M_r^{-1} k_r + M_i M_i^{-1} k_i + \sum_{r=i+1}^{h} M_r n_r \right\rangle_N \tag{2.17}$$

จากสมการที่ (2.16) และสมการที่ (2.17) จะเห็นได้ว่าเราสามารถจะเลือกที่จะกระทำกับมิติใดๆ ก่อนหรือหลังได้ และเมื่อทำจนครบทุกมิติแล้วจะได้เท่ากับสมการที่ (2.13)

พิจารณาสมการที่ (2.16) และสมการที่ (2.17) จะเห็นว่าเมื่อเราคำนวณจับคู่ดัชนีอินพุทจากสมการที่ (2.16) .โดยให้ $n_r = 0,1,2,...,N_r-1$ แล้วจึงทำการแปลงโดยใช้วิธีการแปลงข้อมูลจำนวนน้อยๆ 1 ครั้ง และทำการจับคู่ดัชนีเอาท์พุทโดยใช้สมการที่ (2.17) ซึ่งเราจะต้องทำในขั้นตอนนี้เป็นจำนวนทั้งหมด M_r ครั้ง เราจึงจะทำวิธีการแปลงข้อมูลจำนวนน้อยๆ ครบทั้งอันดับอินพุท ซึ่งจะเห็นได้ว่าสมการที่ (2.16) และสมการที่ (2.17) ไม่เหมาะสมในการใช้งานจริง ดังนั้นเราจะลดรูปสมการที่ (2.16) และสมการที่ (2.17) ให้เหลือเพียง 2 ตัวแปร

เนื่องจาก $M_j = 0 \bmod N_i \ ; i \neq j$ ดังนั้น

$$\sum_{j=1}^{i-1} M_j M_j^{-1} k_j + \sum_{j=i+1}^{h} M_j n_j = 0 \bmod N_i \qquad (2.18)$$

ซึ่งจะได้ว่า

$$\sum_{j=1}^{i-1} M_j M_j^{-1} k_j + \sum_{j=i+1}^{h} M_j n_j = N_i n' \bmod N \qquad (2.19)$$

สำหรับบางค่าของ n'

ซึ่งจะเห็นว่าค่า n' ที่ทำให้สมการที่ (2.19) เป็นจริงคือ $0 \leq n' < M_r$ ดังนั้นสมการที่ (216) และสมการที่ (2.17) สามารถเขียนใหม่ได้เป็น

$$N^r = \left\langle N_i n' + M_r n_r \right\rangle_N$$
$$K^r = \left\langle N_i n' + M_r M_r^{-1} n_r \right\rangle_N \qquad (2.20)$$

โดยที่ $n' = 0,1,...,M_r-1$

2.4 การใช้วิธีการตัวประกอบปฐมกับการแปลงในสนามจำกัดขนาด 15 จุด ในสนามจำกัด $GF(2^4)$

การแปลงในสนามจำกัดขนาด 15 จุดมีสมการดังนี้คือ

$$A_k = \sum_{n=0}^{14} a_n \alpha^{\langle nk \rangle_{15}} \tag{2.21}$$

โดย n = 0,1,2,...,14

k = 0,1,2,...,14

$\alpha = 2$

จะเห็นได้ว่า N = 15 เมื่อเราใช้วิธีการตัวประกอบปฐมกับ N = 15 จะได้ $N_1 = 3$, $N_2 = 5$ และได้ $M_1 = 5$, $M_2 = 3$, $M_1^{-1} = 2$, $M_2^{-1} = 2$ ซึ่งเมื่อแทนค่าลงในสมการที่ (2.11) และสมการที่ (2.13) จะได้

$$n = \langle 5n_1 + 3n_2 \rangle_{15}$$

$$k = \langle 10k_1 + 6k_2 \rangle_{15} \tag{2.22}$$

แทนค่าจากสมการที่ (2.22) ลงในสมการที่ (2.21) จะได้

$$A_{(k_1,k_2)} = \sum_{n=0}^{14} a_{(n_1,n_2)} \alpha^{\langle 5n_1+3n_2 \rangle \langle 10k_1+6k_2 \rangle_{15}} \tag{2.23}$$

เนื่องจาก

$$\alpha^{\langle 3 \cdot 10 n_2 k_1 \rangle_{15}} , \alpha^{\langle 5 \cdot 6 n_1 k_2 \rangle_{15}} = 1$$

และ

$$\alpha^{\langle 5 \cdot 10 n_1 k_1 \rangle_{15}} = \alpha^{\langle 5 n_1 k_1 \rangle_{15}}$$

$$\alpha^{\langle 3 \cdot 6 n_2 k_2 \rangle_{15}} = \alpha^{\langle 3 n_2 k_2 \rangle_{15}}$$

ดังนั้นสมการที่ (2.23) จึงได้เป็น

$$A_{(k_1,k_2)} = \sum_{n_2=0}^{4} \sum_{n_1}^{2} a_{(n_1,n_2)} \alpha^{\langle 5 n_1 k_1 \rangle_{15}} \alpha^{\langle 3 n_2 k_2 \rangle_{15}} \tag{2.24}$$

จากค่า N , M_r และ M_r^{-1} ที่คำนวณได้นำไปแทนค่าในสมการที่ (2.20) จะได้ว่า

$$N^1 = \langle 3n' + 5n_1 \rangle_{15}$$

$$K^1 = \langle 3n' + 10k_1 \rangle_{15}$$

$$N^2 = \langle 5n' + 3n_2 \rangle_{15}$$

$$K^1 = \langle 5n' + 6k_2 \rangle_{15} \qquad (2.25)$$

2.5 วิธีการแปลงข้อมูลจำนวนน้อย ๆ (Short Length Algorithm, SLA)

ในวิธีการแปลงข้อมูลจำนวนน้อยๆนี้เป็นการลดการคำนวณซ้ำซึ่งจะอาศัยหลักการใดๆก็ได้ในที่นี้จะใช้การแทนสัมประสิทธิ์ให้อยู่ในรูปเมทริกซ์เพื่อให้สะดวกในการพิจารณาตัวประกอบร่วม ซึ่งจะแบ่งออกเป็น 2 หัวข้อสำหรับการแปลงขนาด 3 จุดและ 5 จุด

2.5.1 การแปลงในสนามจำกัดขนาด 3 จุด

การแปลงในสนามจำกัดขนาด 3 จุด ใน $GF(2^4)$ จะเขียนได้เป็น

$$A_k = \sum_{n=0}^{2} a_n \beta^{\langle nk \rangle_3} \quad ; \text{k=0,1,2 และ } \beta = \alpha^5 \qquad (2.26)$$

ซึ่งจะได้ว่า

$$A_0 = a_0 + a_1 + a_2$$

$$A_1 = a_0 + \beta a_1 + \beta^2 a_2$$

$$A_2 = a_0 + \beta^2 a_1 + \beta a_2$$

ซึ่ง $\beta^2 = \beta + 1$ จึงเขียนได้ว่า

$$A_0 = a_0 + a_1 + a_2$$

$$A_1 = a_0 + \beta a_1 + (\beta + 1)a_2$$

$$A_2 = a_0 + (\beta + 1)a_1 + \beta a_2$$

ซึ่งสามารถเขียนในรูปเมทริกซ์ได้เป็น

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \beta & \beta+1 \\ 1 & \beta+1 & \beta \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \beta \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

เมื่อพิจารณาตัวประกอบร่วมโดยพิจารณาจากที่มีจำนวนร่วมมากก่อนจึงสามารถเขียนได้ว่า

$$A_0 = a_0 + (a_1 + a_2)$$

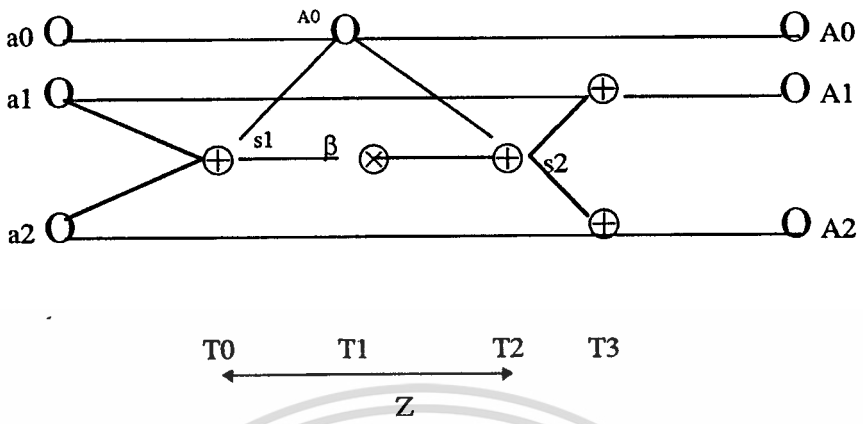
$$A_1 = A_0 + \beta(a_1 + a_2) + a_1$$

$$A_2 = A_0 + \beta(a_1 + a_2) + a_2$$

ซึ่งสามารถเขียนวิธีการแปลงข้อมูลจำนวนน้อยๆ ได้ดังนี้

$$s_1 = a_1 + a_2 \quad , A_0 = s_1 + a_0 \quad , m_1 = \beta s_1$$
$$s_2 = A_0 + m_1 \quad , A_1 = s_2 + a_1 \quad , A_2 = s_2 + a_2$$

ซึ่งนำมาเขียนผังการคำนวณสำหรับการแปลง 3 จุดได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงผังการคำนวณของวิธีการแปลงข้อมูลจำนวนน้อยๆ สำหรับการแปลง 3 จุด

2.5.2 การแปลงในสนามจำกัดขนาด 5 จุด

การแปลงในสนามจำกัดขนาด 5 จุด ใน $GF(2^4)$ จะเขียนได้เป็น

$$A_k = \sum_{n=0}^{4} a_n \beta^{\langle nk \rangle_5} \quad ;k=0,1,2,3,4 \text{ และ } \beta = \alpha^3 \qquad (2.26)$$

ซึ่งจะได้ว่า

$$A_0 = a_0 + a_1 + a_2 + a_3 + a_4$$

$$A_1 = a_0 + \beta a_1 + \beta^2 a_2 + \beta^3 a_3 + \beta^4 a_4$$

$$A_2 = a_0 + \beta^2 a_1 + \beta^4 a_2 + \beta a_3 + \beta^3 a_4$$

$$A_3 = a_0 + \beta^3 a_1 + \beta a_2 + \beta^4 a_3 + \beta^2 a_4$$

$$A_1 = a_0 + \beta^4 a_1 + \beta^3 a_2 + \beta^2 a_3 + \beta a_4$$

ซึ่ง $\beta^2 = 1 + \beta + \beta^3 + \beta^4$ จึงเขียนเป็นเมทริกซ์ได้ดังนี้

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \beta & 1+\beta+\beta^3+\beta^4 & \beta & \beta \\ 1 & 1+\beta+\beta^3+\beta^4 & \beta & \beta & \beta \\ 1 & \beta & \beta & \beta & 1+\beta+\beta^3+\beta^4 \\ 1 & \beta & \beta & 1+\beta+\beta^3+\beta^4 & \beta \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$= (\beta^3+\beta^4) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + (\beta+\beta^3) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$+(\beta+\beta^4) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + (\beta+\beta^4) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$+(1+\beta^3) \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

เมื่อพิจารณาตัวประกอบร่วมจากที่มีจำนวนร่วมมากก่อนจึงทำให้เขียนวิธีการแปลงข้อมูลจำนวนน้อยๆ ได้เป็น

$$s_1 = a_2 + a_3 \quad , s_2 = a_1 + a_4 \quad , s_3 = a_1 + a_3$$

$$s_4 = a_2 + a_4 \quad , s_5 = s_1 + s_2 \quad , A_0 = s_5 + a_0$$

$$m_1 = (1+\beta^3)s_5 \quad , m_2 = (\beta^3+\beta^4)s_1 \quad , m_3 = (\beta+\beta^3)s_2$$

$$m_4 = (\beta+\beta^4)s_3 \quad , m_5 = (\beta+\beta^4)s_4 \quad , s_6 = A_0 + m_1$$

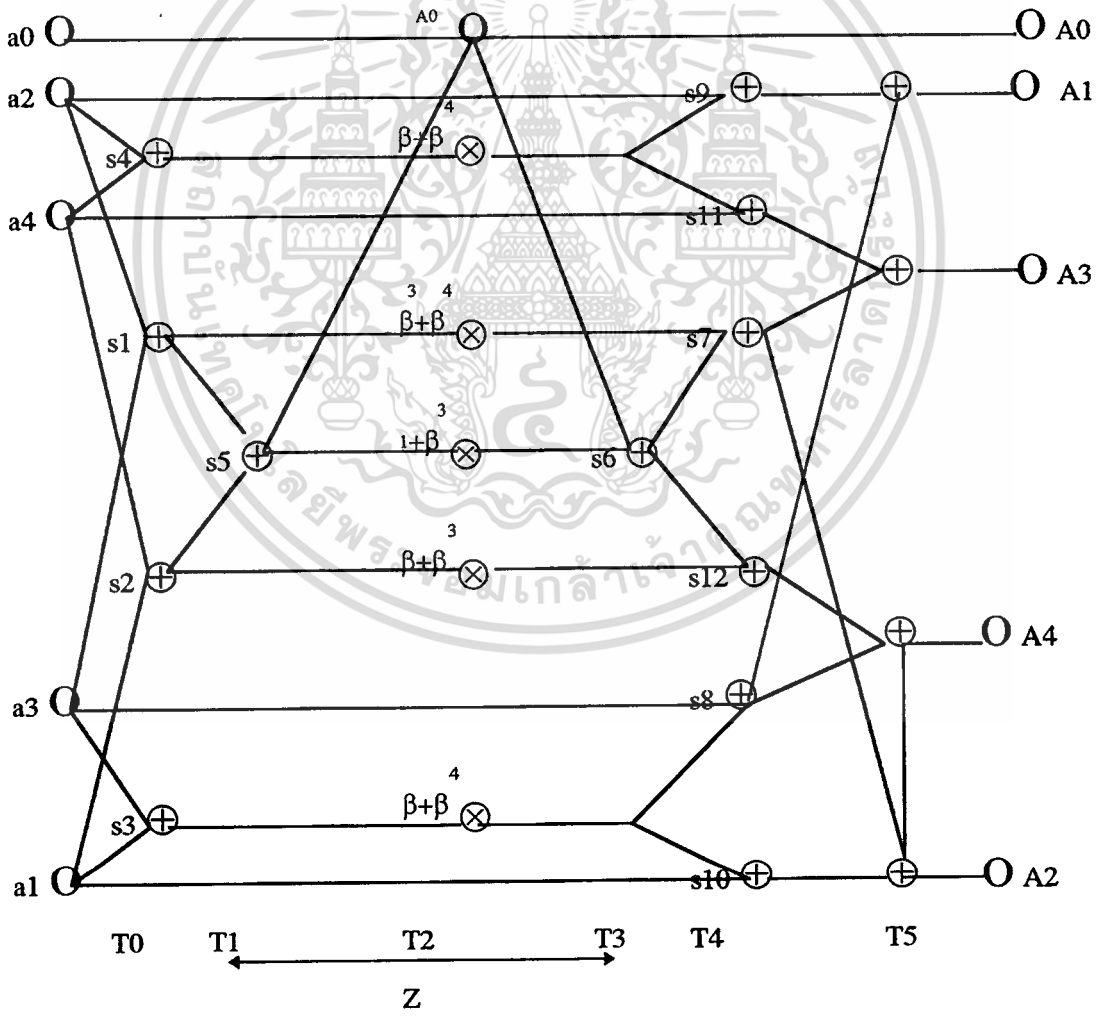
$$s_7 = s_6 + m_2 \quad , s_8 = s_6 + m_3 \quad , s_9 = m_5 + a_2$$

$$s_{10} = m_4 + a_1 \quad , s_{11} = m_5 + a_4 \quad , s_{12} = m_4 + a_3$$

$$A_1 = s_8 + s_9 \quad , A_2 = s_7 + s_{10} \quad , A_3 = s_7 + s_{11}$$

$$A_4 = s_8 + s_{12}$$

ซึ่งนำมาเขียนผังการคำนวณสำหรับการแปลง 5 จุดได้ดังรูปที่ 2.2



รูปที่ 2.2 แสดงผังการคำนวณของวิธีการแปลงข้อมูลจำนวนน้อยๆ สำหรับการแปลง 5 จุด

บทที่ 3

ภาษาวีเอชดีแอล (VHDL)

วีเอชดีแอล (VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) VHDL เป็นภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์ในลักษณะของข้อความ เป็นภาษาที่ถูกนำมาใช้เป็นประโยชน์ในการออกแบบอุปกรณ์ฮาร์ดแวร์ทั้งในระดับเกทจนถึงระบบดิจิตอลที่ซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่เหมาะในการนำมาใช้ในการเลียนแบบการทำงานของอุปกรณ์อีกทั้งยังมีความยืดหยุ่นและไม่ถูกจำกัดโดยความสามารถทางเทคโนโลยีใดๆ ทำให้ประหยัดเวลาและค่าใช้จ่ายในการออก-แบบ จึงได้มีการนำมาใช้กันอย่างกว้างขวางในวงการอุตสาหกรรม รูปแบบของภาษาวีเอชดีแอลประกอบด้วย 2 ส่วนใหญ่ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล (Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Languege) การโปรแกรมด้วยภาษาวีเอชดีแอลสามารถเขียนได้ทั้งสองรูปแบบรวมกัน นอกจากนี้ตัวภาษายังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อยเข้าด้วยกันเพื่อให้เป็นระบบใหญ่ได้ และสามารถกำหนดรูปแบบไวยากรณ์ (Syntax) อีกทั้งยังมีการตรวจสอบความหมายของตัวภาษาว่าจะซิมูเลท (Simulate) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดยวีเอชดีแอลต้องผ่านการซิมูเลทเพื่อตรวจสอบการทำงาน ฉะนั้นในการคอมไพล์ (Compile) จะมีการตรวจสอบทั้งวงจรและซิมูเลชั่นซีแมนติค (Semantic) อย่างไรก็ตามแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบและกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานโดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมด

3.1 แนะนำ VHDL (Introduction to VHDL)

ในช่วงฤดูร้อนของปี 1981 สถาบันเพื่อการป้องกัน(The Institute for Defence Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่ง เพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้น เรียกว่า VHDL (VHSIC HardwareDescription Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Circuit) ต่อมาในปี 1985

,IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านของการกำหนดพฤติกรรมการทำงานของวงจร ทำให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งวงจรของดิจิตอลทั่วๆ ไปและในระบบที่แตกต่างออกไปเช่น พฤติกรรมการทำงานของระบบเรดาร์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ได้ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้จะสามารถถูกใช้ได้ตลอดในทุกๆ ระดับขั้นของการออกแบบที่ต่างกันได้ นั่นคือ ในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงระดับที่ต่ำกว่า (Lower hardware level) สามารถใช้ภาษาเดียวกันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานร่วมกันได้เป็นอย่างดี

3.1.1 ข้อกำหนด (VHDL Requirement)

ในเอกสารของ DoD (Department of Defense Requirements for Hardware Description Language) ซึ่งออกมาในเดือนมกราคมปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

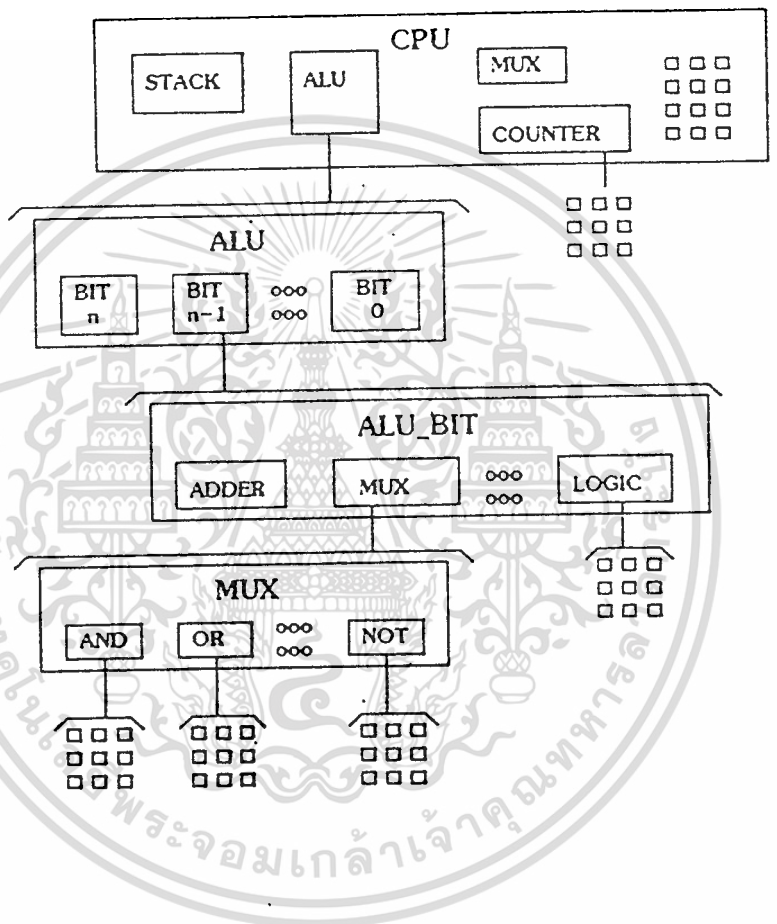
3.1.1.1 ลักษณะทั่วไป (General Features)

เอกสารของ DoD กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์คือ ระบบจนถึงระดับเกทอีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลจริงๆ ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งใน VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้ว ความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่ง องค์ประกอบ เกทหรือวงจรต่างๆจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อมๆ กัน)

3.1.1.2 สนับสนุนการออกแบบแบบลำดับขั้น (Support for Design Hierarchy)

การออกแบบแบบลำดับขั้น เป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบที่มีหลายๆ ระดับ ในการออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบก็สามารถกำหนดได้ด้วยตัวเองหรือกำหนดถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้อง

ถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ดังรูปที่ 3.1



รูปที่ 3.1 แสดงตัวอย่างการออกแบบแบบลำดับขั้น

3.1.1.3 ไลบรารี (Libraly Support)

VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารีหรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารี หลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไปใช้ได้ด้วย

3.1.1.4 ลำดับคำสั่ง (Sequential Statement)

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วย โครงสร้างแบบ case ,if-then-else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

3.1.1.5 การกำหนดคุณสมบัติ (Generic Design)

นอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน สิ่งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ภาษาสำหรับการออกแบบที่ดีควรจะสามารถให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพ เวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

3.1.1.6 ชนิดของข้อมูล (Type Declaration and Usage)

VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (Enumerate Type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเองก็ได้

3.1.1.7 โปรแกรมย่อย (Use of Subprograms)

ความสามารถในการใช้ฟังก์ชั่นและโพรซีเจอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก (Logic) การกำหนดตัวกระทำต่างๆ ทั้งเก่าและใหม่หรืออะไรก็ตามได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

3.1.1.8 การควบคุมเวลา (Timing Control)

VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกท หรือการหน่วงเวลาก็สามารถกระทำได้ โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ (Event) นอกจากนี้ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้อีกด้วย

3.1.1.9 การกำหนดแบบโครงสร้าง (Structural Specification)

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกๆ ระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมที่เกิดจากองค์ประกอบย่อยที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่งเช่นกัน

3.2 ความสามารถของภาษาวีเอชดีแอล (Capability)

- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิปกับผู้ออกแบบ (CAD Tools)

- ใช้เป็นสื่อกลางในการแลกเปลี่ยนสื่อสารระหว่างซีเออี (CAE) และซีเอดีทูล (CAD Tools) เช่น ตัวภาษาซ้อสโค้ด (Source code) ของ VHDL สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และ ซิมูเลเตอร์ (Simulater) ได้หลายตัวแตกต่างกัน

- ภาษา VHDL สนับสนุนการออกแบบ แบบท็อปดาวน์ (Top Down Design) และแบบบัททอมอัพ (Buttom Up Design) หรือผสมกันทั้งสองแบบ

- ตัวภาษา VHDL เป็นแบบทั่วไป (Generic) ไม่อิงเทคโนโลยีอันใดอันหนึ่ง ในขณะเดียวกันก็สามารถสนับสนุนหลายๆ เทคโนโลยี

- ตัวภาษา VHDL สามารถอ่านและทำความเข้าใจได้โดยมนุษย์

- สนับสนุนการออกแบบทั้งระบบซิงโครนัส (Synchronous) และอะซิงโครนัส (Asynchronous)

- ภาษา VHDL เป็นมาตรฐานรับรองโดย IEEE และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษา VHDL สามารถเคลื่อนย้ายไปยังระบบใดๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้

- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับซอฟแวร์)

- ภาษา VHDL สนับสนุนการเขียนถึง 3 รูปแบบ ได้แก่ แบบบีเฮฟวีเออร์ (Behavioral Style) ,แบบสทรักเทอรัล (Structural Style) ,แบบดาต้าโฟลว์ (Data Flow) หรือสามารถเขียนรวมกันได้ทั้งสามรูปแบบ

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

- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลา เช่น Propagation Delay ,Min-Max Delay ,Setup ,Holding Time สามารถอธิบายได้โดยตัวภาษา

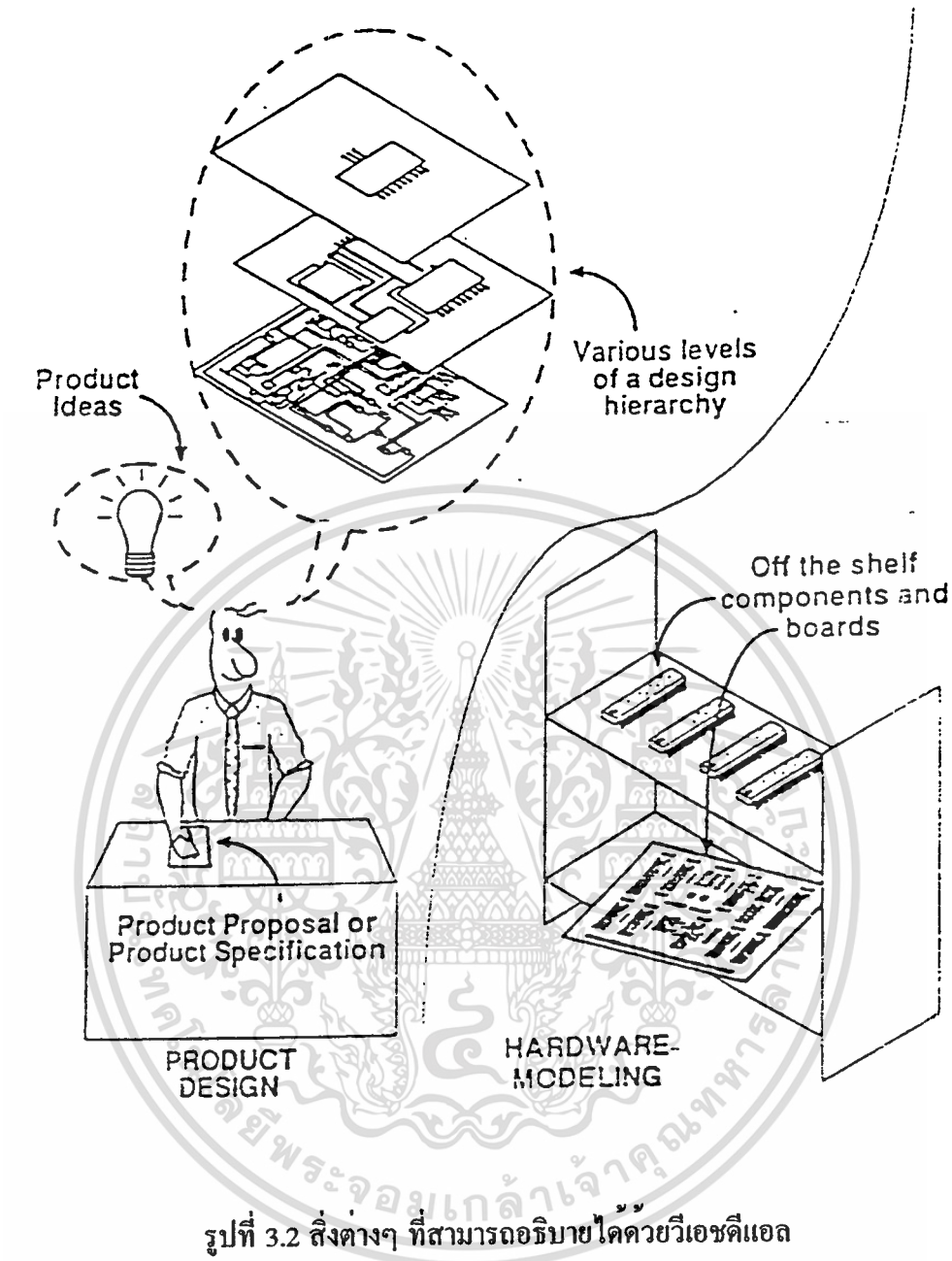
- ภาษา VHDL เป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลายๆ แห่ง ฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจถึงแม้ว่าจะมาจากแหล่งต่างๆ

- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะว่าตัวแปรภาษาได้ตรวตสอบไวยากรณ์ทางด้านซิมูเลชั่นซีแมนติคไว้ด้วย

3.3 หลักการสร้างโมเดลโดยใช้ภาษาวีเอชดีแอล (General VHDL Modelling Principles)

วีเอชดีแอลเป็นภาษาที่ใช้สำหรับอธิบายการทำงานของฮาร์ดแวร์ในรูปแบบฟอร์มที่อ่านเข้าใจได้ ซึ่งจะช่วยในการสร้างและออกแบบวงจรรวมดิจิตอล และส่วนประกอบต่างๆ อาจใช้อธิบายระบบทั้งระบบหรืออธิบายเพียงบางส่วน ซึ่งอยู่ในรูปของ Component Block จากนั้นก็ทำการจำลองการทำงาน (Simulate) โดยที่รูปแบบนั้นยังไม่ได้สร้างขึ้นจริง หรือเพียงแต่อยู่ในรูปของคำอธิบายเท่านั้น (Textual Format) หลังจากจำลองการทำงานจนได้ตามที่ต้องการจึงนำไปทำการ Synthesis เพื่อให้ได้วงจรเกทเลเวลต่อไป ประโยชน์จริงของการใช้วีเอชดีแอลเป็น Design Tools แทนการสร้างต้นแบบ (Prototype) ขึ้นมาจริง คือเราสามารถอธิบาย Product Idea , Product Proposal ,Product Specification เป็นลักษณะในรูปของ Text จากนั้นก็นำคอมไพล์เพื่อดู Timing การทำงานแล้วทำการแก้ไข (Refine) จนกว่าจะได้ Specification ตามต้องการ เมื่อ Product ได้ผลตามที่ต้องการแล้วจึงนำไปสู่การสังเคราะห์ (Synthesis) เพื่อให้ได้เกทเลเวล Schematic เพื่อนำไปสร้างเป็นต้นแบบจริงต่อไป ซึ่งต้นแบบที่สร้างนั้นทำงานได้จริงเพราะ ได้ทำการ Simulate เรียบร้อยแล้ว เป็นการลดเวลาและค่าใช้จ่ายในการสร้างต้นแบบได้มาก

ตัวภาษา VHDL สนับสนุนหลักการต่างๆ ให้เขียนแก้ไขและบำรุงรักษาวงจรดิจิตอลที่มีความซับซ้อนให้เป็นไปอย่างรวดเร็วและมีประสิทธิภาพโดยมีหลักการดังนี้



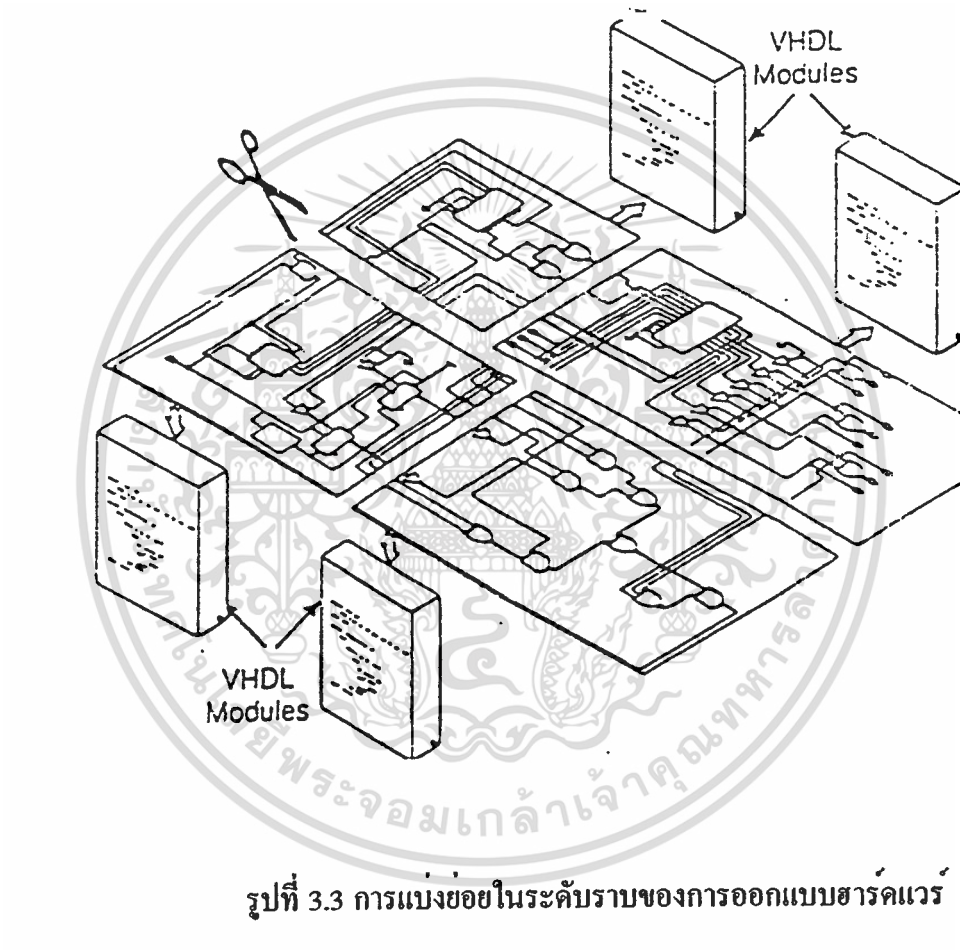
รูปที่ 3.2 สิ่งต่างๆ ที่สามารถอธิบายได้ด้วยวีเอชดีแอล

3.3.1 Top Down Design

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน เช่น ASIC (Application Specific Integrated Circuit) วิศวกรหรือผู้ออกแบบมักจะมองรูปแบบให้อยู่ในรูปของ Block Diagram เสียก่อน ก่อนที่จะย่อยรูปแบบให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายการทำงานของแต่ละ Block ,วิเคราะห์การทำงาน (Analyze) ,จัดการแก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามที่ต้องการก่อนที่จะทำการออกแบบให้ละเอียดลึกลงในขั้นตอนต่อไป การแก้ไขในขั้นตอนนี้จะทำให้ลดค่าใช้จ่ายกว่าการแก้ไขในช่วงของการพัฒนาในระดับสร้างซิลิกอนชิป (Silicon Chip)

3.3.2 Modularity

Modularity คือ หลักการในการแยกส่วน (Partitioning) ฮาร์ดแวร์ ออกเป็นส่วนย่อยเล็กๆ ลงไป ซึ่งปกติการทำงานของฮาร์ดแวร์ใหญ่ๆ ต้องประกอบด้วยฮาร์ดแวร์ย่อยๆ ลงไป ดังรูปที่ 3.3

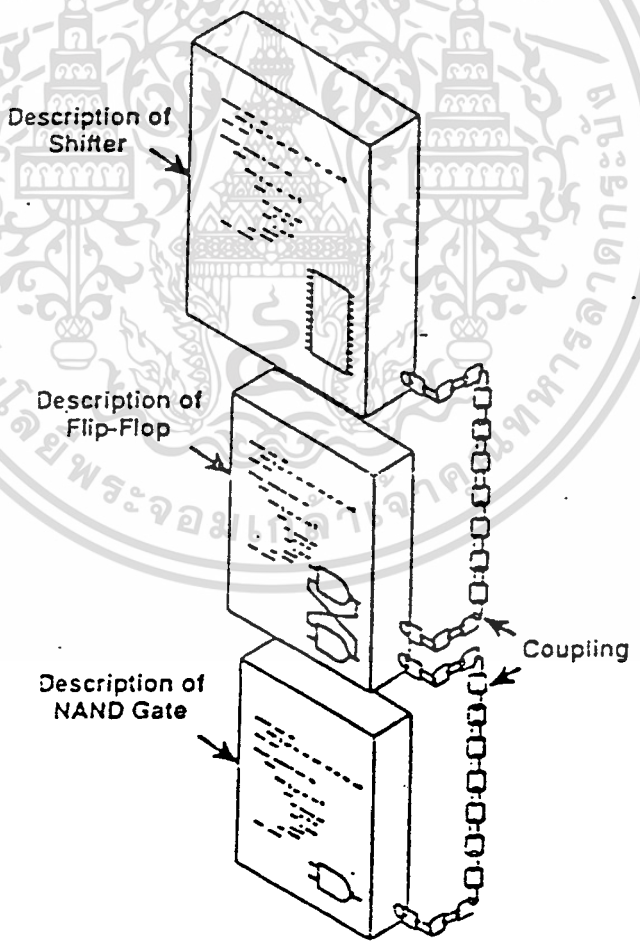


รูปที่ 3.3 การแบ่งย่อยในระดับราบของการออกแบบฮาร์ดแวร์

จากรูปซึ่งแสดงวงจรทั้งหมดในรูปเดียว (Flatten Design) หลังจากนั้นตัดเป็นส่วนย่อยๆ เล็กลงมา เมื่อเราออกแบบโดยใช้ภาษา VHDL หน้าที่การทำงานของแต่ละส่วนต้องสามารถอธิบายได้โดย Module ของ Code (คล้าย Function หรือ Procedure) ซึ่งแสดงการทำงานของส่วนย่อยนั้นอย่างชัดเจน ซึ่งการแยกรูปใหญ่ๆ ออกเป็นส่วนย่อยๆ นี้ทำให้ง่ายต่อการจัดการและง่ายต่อการทำความเข้าใจ

รูปที่ 3.4 แสดง Hierarchy Method โดยการแยกส่วนรูปแบบออกเป็นส่วนย่อยๆ ส่วนบนสุดอธิบายการทำงานของ Shifter ส่วนล่างๆลงมาคือการแยกส่วนของ Shifter ออกเป็นฟลิปฟลอป

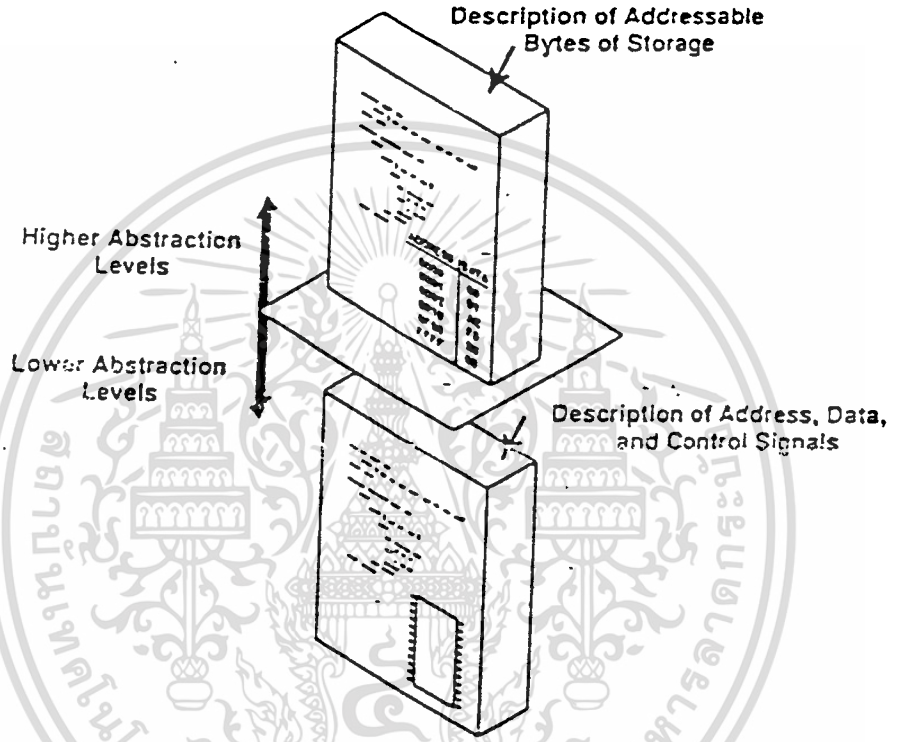
จากฟลิปฟลอปแยกเป็น NAND เกท ภายใน Shifter ได้อธิบายการทำงานโดยใช้การต่อกันของฟลิปฟลอป ในระดับต่ำลงมา ฟลิปฟลอปก็เกิดจากการใช้ NAND เกท ต่อกัน 2 ตัวในระดับต่ำลงมาอีกก็เป็น NAND เกท ซึ่งมีการอธิบายการทำงานอยู่ภายใน ซึ่งแต่ละ module จะมีคำอธิบายการทำงานในตัวของมันเองอยู่แล้วคำอธิบายภายในแต่ละ module มีไว้เพื่อให้สามารถให้ใช้ฟลิปฟลอป module ได้ ส่วนฟลิปฟลอป module ก็อธิบายการเชื่อมต่อไว้อย่างดีทำให้สามารถเชื่อมต่อกับ NAND เกท ในระดับล่างสุดได้ ประโยชน์อย่างหนึ่งของการแยกส่วนฟลิปฟลอป และ NAND เกท ออกจากกัน เนื่องจากว่าทำให้ง่ายในการที่จะใช้ NAND เกทตัวนี้ ในรูปแบบไฮเลเวล (High Level) ตัวอื่นๆ ทำให้นำออกไปใช้งานได้อีก และลดความซับซ้อนในการใช้อุปกรณ์ส่ง เป็นการง่ายที่จะแก้ไขการทำงานของ Shifter โดยปราศจากการแก้ไขฟลิปฟลอป และ NAND เกท จากประโยชน์ที่ได้จากการทำ Modularity นี้ ทำให้รูปแบบที่เราออกแบบง่ายต่อการเข้าใจและแก้ไขได้เสมอ



รูปที่ 3.4 การแบ่งแบบ Hierarchy ของ VHDL Shifter Description

3.3.3 Abstraction

คำนิยามของรูปแบบจะอธิบายการทำงานของตัวรูปแบบมากกว่าที่จะอธิบายถึงว่าพัฒนาตัวรูปแบบนั้นอย่างไร หลักการนี้มีความสำคัญอย่างใกล้ชิดกับหลักการของ Modularity ในรูปที่ 3.4 ฟลิปฟลอป เป็นนิยามในการใช้ NAND เกทและ Shifter เป็นนิยามในการใช้ฟลิปฟลอป



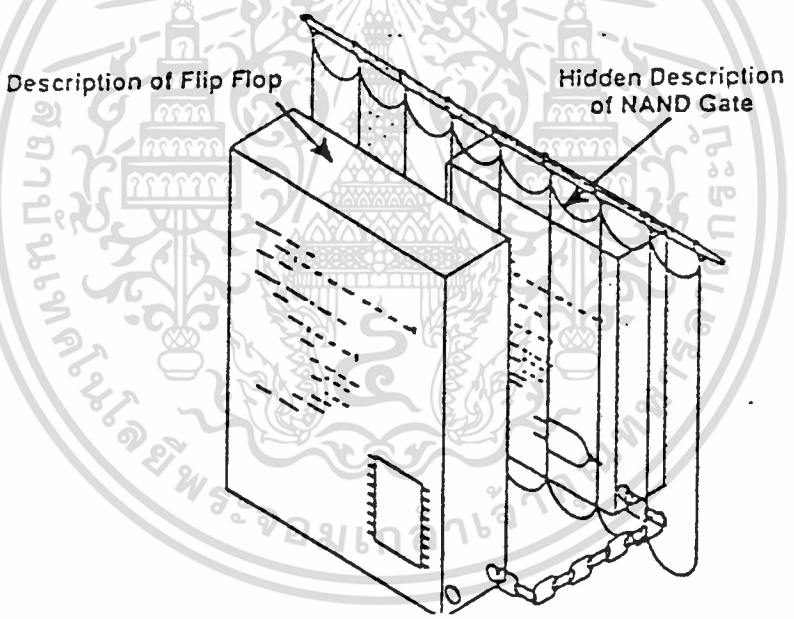
รูปที่ 3.5 Applying Abstraction to a ROM Description

รูปที่ 3.5 แสดงอีกวิธีการหนึ่งซึ่งแสดงถึงการอธิบายการทำงานของรูปแบบโดยใช้ VHDL ในหลายๆ ระดับของการนิยาม ROM (Read Only Memory) อธิบายโดยใช้ภาษาระดับสูง (High Level Language) แสดงถึงตำแหน่ง (Address) ต่างๆ ซึ่งเก็บข้อมูลไว้ในตำแหน่งนั้นๆ ที่ระดับนี้ไม่ต้องสนใจถึง Address Line ,Data Line หรือ Control Line เราสามารถเพ่งจุดสนใจไปที่ขนาดของข้อมูลโดยไม่ต้องคิดถึงสัญญาณควบคุมต่างๆ ภายในเพราะว่าส่วนนั้นจะถูกจัดการเองในระดับต่ำลงมา ในระดับล่างลงมาเราสามารถอธิบายการทำงานของสัญญาณแต่ละเส้นภายใน ROM ในการจัดการสัญญาณภายในทุกเส้นภายใน การที่จะอ่านข้อมูลหรือโปรแกรมข้อมูลใน ROM ถ้าเราต้องการเปลี่ยนค่าข้อมูลภายใน ROM เราควรแก้ไขในระดับที่สูงขึ้นมาจะทำให้ง่ายกว่า

ในการที่จะควบคุมสัญญาณภายใน จะเห็นว่าแต่ละระดับมีความเหมาะสมแตกต่างกันไป และในจุดนี้เองทำให้รูปแบบที่เราออกแบบง่ายต่อการแก้ไขโดยการใช้ประโยชน์ของ Abstraction

3.3.4 Information Hiding

เมื่อทำการเขียน VHDL Code ขึ้นมาเพื่ออธิบายการทำงานของฮาร์ดแวร์ตัวหนึ่ง บางครั้งเราอาจต้องการที่จะซ่อนรายละเอียดการพัฒนา module นั้นโดยไม่ต้องการให้ส่วน module อื่นๆ รู้การทำงานภายใน Information Hiding มีประโยชน์คือ ทำให้รูปแบบภาษา VHDL นั้นสามารถจัดการได้ง่ายและสามารถอ่านทำความเข้าใจได้ง่าย หลักการนี้จะใช้สนับสนุนหลักการ Abstraction คือ จะสนใจรายละเอียดในการใช้งานมากกว่าจะสนใจว่า รูปแบบนั้นจะถูกสร้างขึ้นมาอย่างไรบ้าง เป็นต้น การซ่อนรายละเอียดภายใน module ทำให้ความสนใจของผู้ออกแบบสนใจไปในส่วนที่สำคัญมากกว่า ในส่วนที่ไม่น่าสนใจจะซ่อนไว้และเข้าถึงไม่ได้ ดังรูปที่ 3.6



รูปที่ 3.6 การซ่อนรายละเอียดที่ไม่จำเป็นของระดับ NAND เกท

คนที่เขียนอธิบายการทำงานของฟลิปฟลอปไม่ต้องสนใจเลยว่า NAND เกท จะทำงานอย่างไร จะต่อกันภายในอย่างไร โดย NAND เกท สามารถเขียนขึ้นมาแล้วคอมไพล์ เก็บไว้ในไลบรารี ผู้ที่ออกแบบฟลิปฟลอประดับสูงขึ้นมา เพียงแต่ต้องรู้ว่า จะเชื่อมต่ออินพุท/เอาท์พุทของ NAND เกท มาใช้งานได้อย่างไร โดยไม่ต้องสนใจว่า NAND เกท จะถูกสร้างและพัฒนามา

อย่างไร ประโยชน์อีกอย่างหนึ่งคือ ป้องกันข้อมูลภายใน ในกรณีที่แจกจ่าย VHDL โมเดลไปยังที่อื่น ทำให้เราป้องกันทรัพย์สินทางปัญญาได้ในอีกระดับหนึ่ง

3.3.5 Uniforminity

Uniforminity เป็นหลักการอีกอย่างหนึ่งที่ช่วยในการอธิบายฮาร์ดแวร์ด้วยภาษา VHDL หมายถึง การสร้าง module ของรหัสในลักษณะคล้ายกัน โดยใช้ตัวภาษา VHDL Building Block ทำให้เกิดการเขียนรหัสที่ดีอย่างเช่น มีการใช้ย่อหน้า มีการใช้คำอธิบาย (Comment) เป็นต้น ทำให้การพัฒนา module ทำความเข้าใจง่าย

3.4 องค์ประกอบพื้นฐานใน VHDL (Basic Concept in VHDL)

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 3.7 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ **ENTITY** ตามด้วยชื่อขององค์ประกอบและคำ **IS** ภายในบรรยายถึงพอร์ตการติดต่อ อินพุท/เอาท์พุทพอร์ตขององค์ประกอบ ส่วนลักษณะกายภาพภายนอกอื่นๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า **ARCHITECTURE** ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นกับสัญญาณอินพุท/เอาท์พุท และพารามิเตอร์อื่นๆที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 3.7 การบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังจากคำว่า **BEGIN** เป็นต้นไป

```
ENTITY component_name IS
        input and output ports.
        physical and other parameters.
END component_name;
```

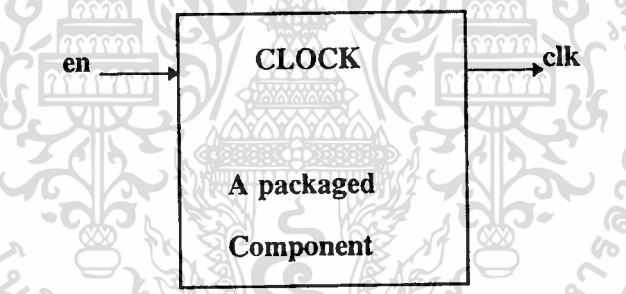
```
ARCHITECTURE identifier OF component_name IS
        declarations.
BEGIN
        specification of the functionality of the component
        in terms of its input lines and as influenced
```

```
        by physical and other parameters.
END identifier;
```

รูปที่ 3.7 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

3.4.1 การกำหนดการเชื่อมต่อ (Interface Description)

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 3.8 แสดงบล็อคไดอะแกรมและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดให้เป็นชื่อ clock_component ตามด้วยคำว่า **PORT** และชื่อของพอร์ตอยู่ในวงเล็บ **IN** และ **OUT** กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท **BIT** แสดงชนิดของข้อมูล




```
ENTITY clock_component IS
        PORT (en: IN BIT; ck: OUT BIT);
END clock_component;
```

รูปที่ 3.8 แสดงบล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_component

3.4.2 การกำหนดรูปแบบการบรรยาย (Architecture Description)

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 3.9 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุทและ ck เป็นเอาท์พุท **PROCESS** เป็นค่าเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในโพรเซสกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น “0” ถ้าสัญญาณ

en มีค่าเป็น “1” ค่าของ periodic จะถูกคอมพลีเมนต์ (Complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง **WAIT** กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS
BEGIN
        PROCESS
                VARIABLE periodic : BIT := “0 “;
        BEGIN
                IF en = “1” THEN
                        periodic := NOT periodic;
                END IF;
                ck <= periodic;
                WAIT FOR 1 US;
        END PROCESS;
END behavioral;
```

รูปที่ 3.9 แสดงการบรรยายเชิงพฤติกรรมของ clock_component

3.4.3 โปรแกรมย่อย (Subprograms)

การใช้ฟังก์ชั่นและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่น ถ้าเราให้ฟังก์ชั่นแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกจริงๆ ในขณะที่ถ้าเราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าหน่วงเวลา แล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์

รูปที่ 3.10 แสดงการใช้โพรซีเจอร์เพื่อเปลี่ยนข้อมูลชนิด 8 บิต เป็นค่าจำนวนเต็ม รูปที่ 3.11 แสดงการใช้ฟังก์ชั่นโดยกำหนดให้ X เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน

```
TYPE byte IS ARRAY ( 7 DOWN TO 0 ) OF BIT
...
PROCEDURE byte_to_integer (ib: IN byte; oi: OUT INTEGER) IS
        VARIABLE result : INTEGER :=0 ;
BEGIN
        FOR i IN 0 TO 7 LOOP
                IF ib(i) = "1" THEN
                        result := result + 2**i;
                END IF;
        END LOOP;
        oi := result;
END byte_to_integer;
```

รูปที่ 3.10 แสดงการใช้โพรซีเจอร์

```
FUNCTION f (a, b, c : BIT ) RETURN BIT IS
        VARIABLE  x : BIT;
BEGIN
        x := (( NOT a) AND ( NOT b) AND c);
        RETURN x ;
END f;
```

รูปที่ 3.11 แสดงการใช้ฟังก์ชั่น

3.4.4 โอเปอร์เรเตอร์ (VHDL Operators)

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟท์แวร์ทั่วไปดังรูปที่ 3.12

PREDEFINED OPERATORS

LOGICAL OPERATORS : NOT AND OR NAND NOR XOR

OPERAND TYPE : BIT BOOLEAN

RESULT TYPE : BIT BOOLEAN

RELATIONAL OPERATORS : = /= < <= > >=

OPERAND TYPE : any type

RESULT TYPE : Boolean

ARITHMETIC OPERATORS : + - * / ** MOD REM ABS

OPERAND TYPE : INTEGER REAL Physical

RESULT TYPE : INTEGER REAL Physical

CONCANTINATION OPERATOR : &

OPERAND TYPE : array of any type

RESULT TYPE : array of any type

รูปที่ 3.12 แสดงตัวกระทำใน VHDL

3.4.5 เวลาและความพร้อมเพรียง (Timing and Concurrency)

ในวงจรอิเล็คทรอนิค อุปกรณ์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (always active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยเสมอในทุกๆ เหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงาน ที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยาย (architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายๆ โพรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกๆ โพรเซสก็จะทำงานไปพร้อมๆ กันด้วย

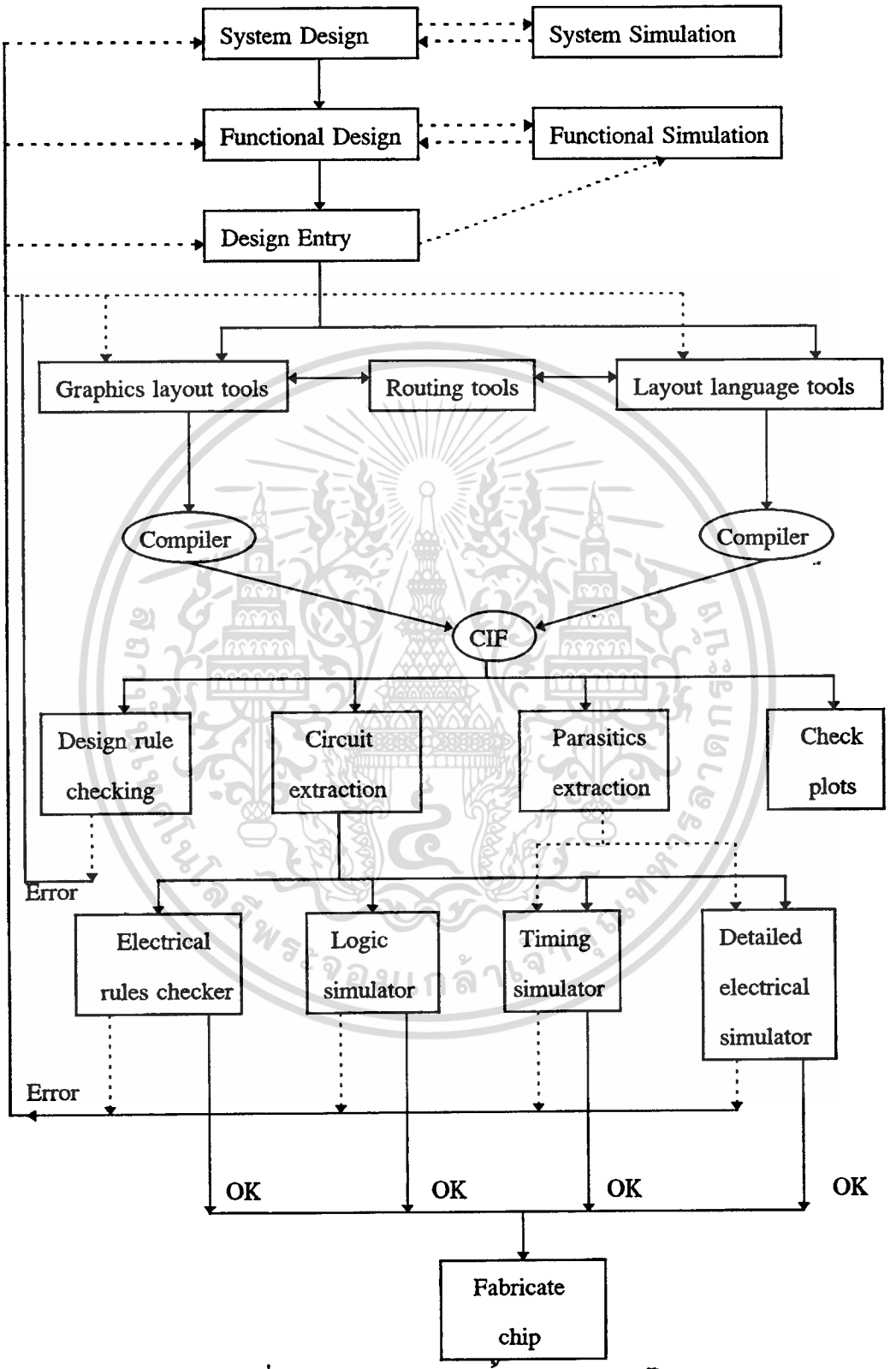
3.4.6 สัญญาณและตัวแปร (Signals and Variable)

สัญญาณที่เป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง **AFTER** เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น w <= AFTER 12 ns หมายถึง กำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 ns.

3.5 โครงสร้างของ VHDL

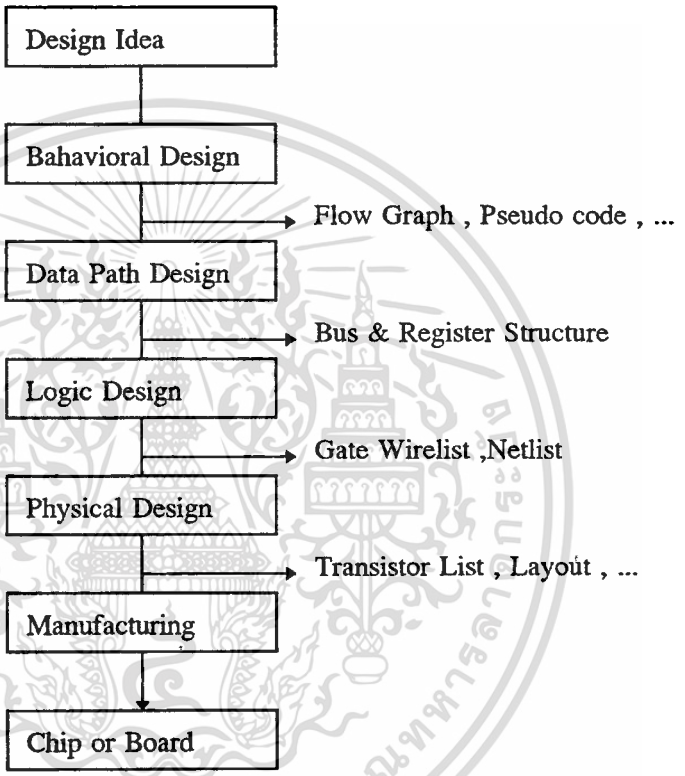
ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆ มากมายและในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและเข้ากระบวนการออกแบบในขั้นต่อไป

โครงสร้างการออกแบบ VHDL สามารถเขียนเป็นแผนผังได้ดังรูปที่ 3.13



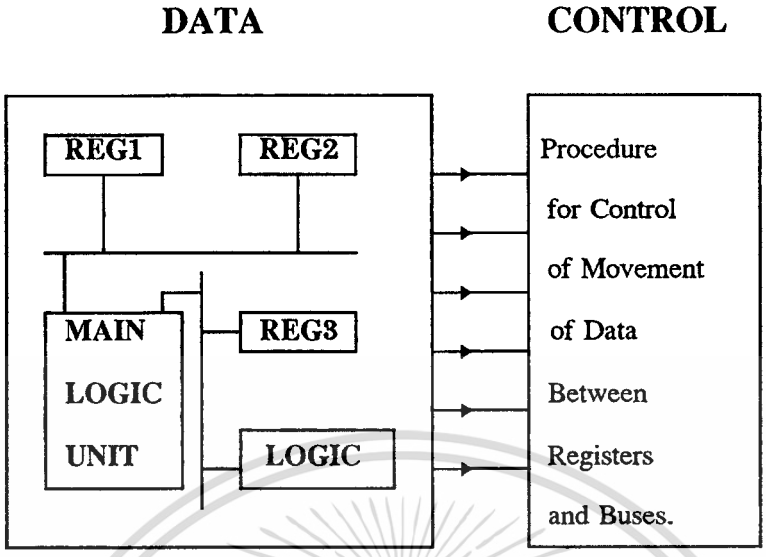
รูปที่ 3.13 รูปแสดงโครงสร้างการออกแบบ VHDL

รูปที่ 3.14 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วๆ ไป ขั้นแรกผู้ออกแบบกำหนดแนวความคิดในการออกแบบเสียก่อนและทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นภายในขั้นตอนนี้จึงมีความจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบ ซึ่งอาจจะเป็นผังงาน ผังแสดงแบบ หรือรหัสคำสั่งเทียม (Pseudo code) ก็ได้



รูปที่ 3.14 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์ (Register) และวงจรตรรก (Logic) ที่จำเป็นทั้งหมดเพื่อนำมาประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรตรรกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ ดังรูปที่ 3.15



รูปที่ 3.15 แสดงการออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิปฟลอปเป็นส่วนประกอบของอุปกรณ์แยกต่างๆ ได้แก่ รีจิสเตอร์เก็บข้อมูล วงจรตรรก และส่วนควบคุมฮาร์ดแวร์ ซึ่งในขั้นสุดท้ายจะได้ออกมาเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง

ต่อมา เป็นขั้นตอนของการเปลี่ยนเครือข่ายการโยงใยในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์ และเลย์เอาท์ (transistor list and layout) ในขั้นตอนนี้จะเกี่ยวข้องกับการจัดวางเกทและฟลิปฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซล ขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

3.6 การเขียนอธิบายในรูปแบบต่างๆ (Design Description Methods)

ภาษา VHDL เป็นวิธีการเขียนอธิบายการทำงานของฮาร์ดแวร์ในลักษณะของ Textual Format ซึ่งวิธีการเขียนอธิบายภาษา VHDL ในหลายๆ วิธี เพื่อที่จะอธิบาย Hardware Architecture มีดังนี้

3.6.1 การบรรยายเชิงพฤติกรรม (Behavioral Description of Hardware)

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรมเป็นการบรรยายในลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้น

สืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามาโดยไม่คำนึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในจะเป็นอย่างไร ในหัวข้อนี้จะแสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองรูปแบบอินพุทและเอาท์พุทในระดับพฤติกรรมแทนการใช้โมดูลฮาร์ดแวร์รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

3.6.1.1 โพรเซส (Process Statement)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อมๆ กันกับโพรเซสอื่นๆ ที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์การกำหนดค่าให้กับสัญญาณ (<=)

การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง **PROCESS** และจบด้วยคำสั่ง **END PROCESS** รูปที่ 3.16 แสดงส่วนประกอบของการบรรยายแบบโพรเซสซึ่งประกอบด้วยส่วนของการประกอบตัวแปรที่ต้องใช้และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ

รูปที่ 3.16 แสดงรูปแบบของการบรรยายแบบโพรเซส

3.6.1.2 การกำหนดตัวกระทำภายในโพรเซส (Declarative Part of a Process)

ตัวกระทำภายในโพรเซสมี 3 ชนิด คือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวกระทำทั้งสามชนิดนี้อาจมีการประกาศไว้ในโพรเซสใดก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซสสามารถจะทำได้โดยใช้สัญญาณ (Signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE

```
PROCESS
        FILE flush : TEXT IS IN "filename.dat";
        VARIABLE var : BIT;
        CONSTANT n : INTEGER := 0 ;
BEGIN
        ...
END PROCESS;
```

รูปที่ 3.17 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส

รูปที่ 3.17 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN ค่าเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น แต่ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้นๆ

3.6.1.3 การกำหนดการกระทำภายในโพรเซส (Statement Part of a process)

การกระทำใดๆ ภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ (sequential) เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการทำซ้ำได้เช่น IF-THEN-ELSE, CASE-WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 3.18 และ 3.19

```
ARCHITECTURE demo OF partial_process IS
        ...
BEGIN
        PROCESS
                ...
        BEGIN
                ...
                x <= '1';
                IF x='1' THEN
                        perform action_1
                ELSE
                        perform action_2
                END IF;
                ...
        END PROCESS;
END demo;
```

รูปที่ 3.18 แสดงเงื่อนไขการกระทำในโพรเซส

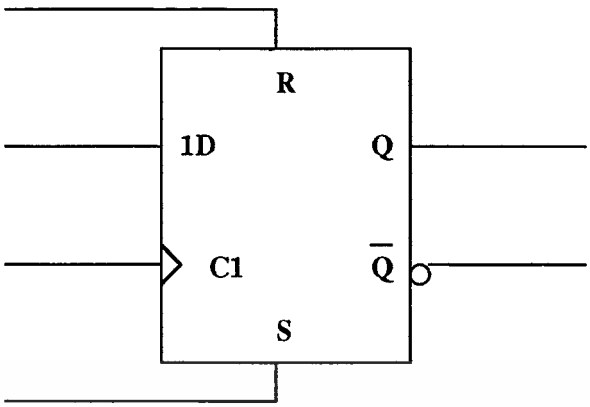
```
ARCHITECTURE demo OF partial_process IS
        ...
BEGIN
        PROCESS
        BEGIN
                ...
                x <= a AFTER 10 ns;
                y <= b AFTER 6 ns;
                ...
        END PROCESS;
END demo;
```

รูปที่ 3.19 แสดงการกระทำในโพรเซส

3.6.1.4 การกระตุ้นและยับยั้งการกระทำของโพรเซส (Sensitivity List)

การกระทำภายในโพรเซสจะเตรียมพร้อมและมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้โดยการกำหนดรายการของสัญญาณที่ต้องการให้โพรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้น เหตุการณ์ใดๆ ที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโพรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS

รูปที่ 3.20 แสดง (a) ตัวอย่างโมเดลและ (b) ตัวอย่างการบรรยายการเชื่อมต่อของ D-Flipflop รูปที่ 3.21 แสดงการบรรยายเชิงพฤติกรรมของ D-Flipflop (a) การใช้ตัวการะทำภายนอกโพรเซสและ (b) การใช้ตัวกระทำภายในโพรเซส โดยมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโพรเซส



(a)

```
ENTITY d-sr-flipflop IS
        GENERIC (sq-delay, rq_delay, cq_delay: TIME := 6 ns);
        PORT (d, set, rst, clk : IN BIT;  q, qb : OUT BIT);
END d_sr_flipflop;
```

(b)

รูปที่ 3.20 (a) ตัวอย่างโมเดล D-FlipFlop
(b) การบรรยายการเชื่อมต่อของ D-FlipFlop

```
ARCHITECTURE behavioral OF d_sr_flipflop IS
        SIGNAL state : BIT:= '0' ;
BEGIN
        dff : PROCESS (rst, set, clk)
                BEGIN
                    IF set = '1' THEN
                        state <= '1' AFTER sq_delay;
                    ELSIF rst = '1' THEN
                        state <= '0' AFTER rq_delay;
                    ELSIF  clk = '1' AND clk' EVENT THEN
                        state<= d AFTER  cq_delay;
                    END IF;
                END PROCESS dff;
```

```
        q<= state;
        qb<= NOT state;
END behavioral;
```

(a)

```
ARCHITECTURE average_delay_behavioral OF d_sr_flipflop IS
    BEGIN
    dff : PROCESS ( rst, set, clk )
        VARIABLE state : BIT := '0';
      BEGIN
        IF set = '1' THEN
            state := '1' ;
        ELSIF rst = '1' THEN
            state := '0' ;
        ELSIF clk = '1' AND clk'EVENT THEN
            state := d;
        END IF;
        q <= state AFTER (sq_delay + cq_delay)/3;
        qb <= NOT state AFTER ( sq_delay + rq_delay + cq_delay)/3;
    END PROCESS dff;
END behavioral;
```

(b)

รูปที่ 3.21 แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop

(a) การใช้ตัวกระทำภายนอกโพรเซส

(b) การใช้ตัวกระทำภายในโพรเซส

3.6.1.5 การหยุดรอ (Sequential Wait Statements)

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

```
WAIT FOR waiting_time;
WAIT ON waiting_sensitivity_list;
WAIT UNTIL waiting_condition;
WAIT;
```

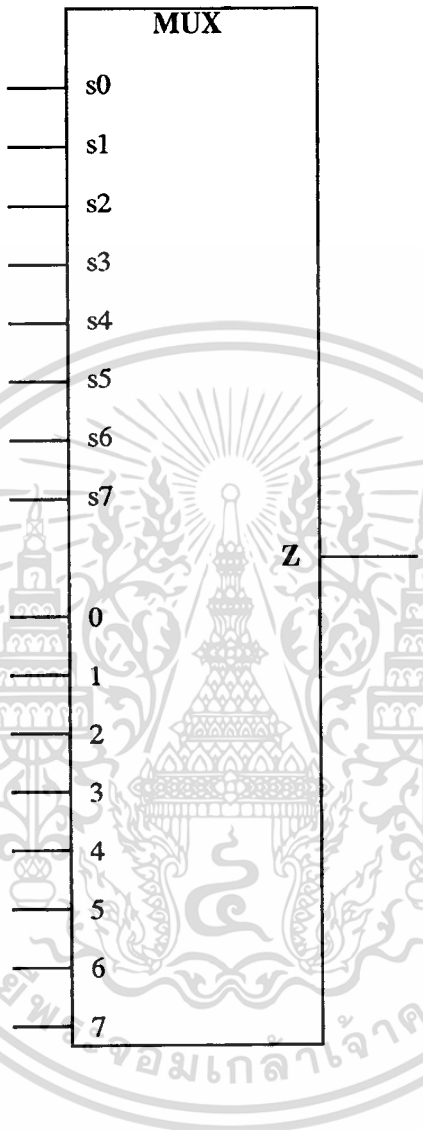
คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซสที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น **WAIT FOR** จะหยุดรอเป็นเวลาเท่ากับ waiting_time **WAIT ON** จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ waiting_sensitivity_list **WAIT UNTIL** จะหยุดรอจนกว่าเงื่อนไข waiting_condition เปลี่ยนจาก FALSE เป็น TRUE **WAIT** จะหยุดรอต่อไป

3.6.2 การบรรยายเชิงกระแสข้อมูล (Dataflow Description of Hardware)

การบรรยายเชิงกระแสข้อมูลเป็นการบรรยายถึงการเลื่อนไหลของข้อมูลผ่านรีจิสเตอร์และบัสของระบบ เป็นระดับขั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเคลื่อนไหวของข้อมูลได้แก่ conditional , selected และ guarded ในหัวข้อนี้จะแสดงให้เห็นถึงลักษณะและรูปแบบของการบรรยายเชิงกระแสข้อมูล รวมถึงข้อกำหนดและเงื่อนไขต่างๆ พร้อมกันทั้งตัวอย่างประกอบ

3.6.2.1 การกำหนดเลือกข้อมูล (Multiplexing and Data Selection)

ในระบบดิจิตอล โครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัสและรีจิสเตอร์ รูปที่ 3.22 แสดงตัวอย่างโมเดลและการบรรยายเชิงกระแสข้อมูลของ 8-ต่อ-1 มัลติเพล็กเซอร์



(a)

```
ENTITY mux_8_to_1 IS
        PORT ( i7,i6,i5,i4,i3,i2,i1,i0 : IN BIT;
                s7,s6,s5,s4,s3,s2,s1,s0 : IN BIT;
                z : OUT BIT);
END mux_8_to_1;

ARCHITECTURE dataflow OF mux_8_to_1 IS
        SIGNAL set_lines : BIT_VECTOR (7 DOWNTO 0);
BEGIN
        set_lines <= s7 & s6 & s5 & s4 & s3 & s2 & s1 & s0;
        WITH set_lines SELECT
          z <=  '0' AFTER 3 ns WHEN "00000000",
                i7 AFTER 3 ns WHEN "10000000"
                i6 AFTER 3 ns WHEN "01000000"
                i5 AFTER 3 ns WHEN "00100000"
                i4 AFTER 3 ns WHEN "00010000"
                i3 AFTER 3 ns WHEN "00001000"
                i2 AFTER 3 ns WHEN "00000100"
                i1 AFTER 3 ns WHEN "00000010"
                i0 AFTER 3 ns WHEN OTHERS;
END dataflow;
```

(b)

รูปที่ 3.22 แสดง 8-ต่อ-1 มัลติเพล็กเซอร์

(a) โมเดล

(b) การบรรยายเชิงกระแสข้อมูล

set_lines เป็นสัญญาณที่กำหนดขึ้นมาเพื่อรวมสัญญาณเลือกทั้ง 8 อินพุท (s7 - s0) เข้าด้วยกันแล้วนำไปใช้เป็นสัญญาณในการกำหนดเลือกอินพุทตัวใดตัวหนึ่งให้กับเอาท์พุท Z ถ้าไม่มี

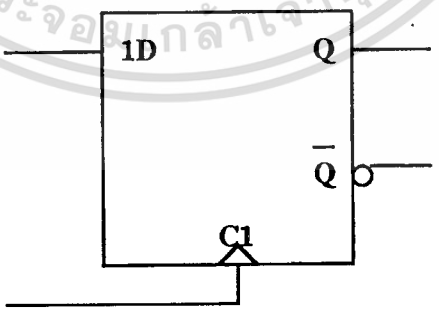
อินพุทใดมีค่าเป็น '1' ค่า '0' จะถูกส่งให้กับเอาท์พุท Z หลังจากเวลาผ่านไป 3 ns. ส่วนอินพุทอื่นๆ (i7-i0) จะถูกส่งให้ Z ขึ้นอยู่กับสัญญาณรูปใด (s7-s0) มีค่าเป็น '1'

3.6.2.2 การกำหนดการ์ด (Guarded Signal Assignments)

ในการบรรยายกระแสเชิงข้อมูลเรา สามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญญาณใดๆ ได้โดย ใช้คำสั่ง GUARDED ซึ่งมีรูปแบบในการเขียนดังนี้

```
target <= GUARDED waveforms or conditional waveforms or selected waveforms;
```

รูปที่ 3.23 แสดงตัวอย่างการใช้ GUARDED ในการบรรยายการทำงานของ d_flipflop การกำหนด GUARDED ทำได้โดยใช้รูปแบบของคำสั่ง BLOCK ตามด้วยเงื่อนไขภายในวงเล็บซึ่งผลลัพธ์ที่ได้จากวงเล็บนี้ก็คือ GUARDED นั่นเอง ในรูปเป็นการบรรยายการทำงานของ d_flipflop ที่มีการทำงานเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก '0' เป็น '1' หรือสัญญาณขาขึ้น จะเห็นว่า GUARDED ถูกใช้ในเงื่อนไขการกำหนดค่าให้กับ q และ qb นั่นคือ ถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น TRUE (GUARDED เป็น TRUE) ค่า d และ NOT d จะถูกส่งค่าให้กับ q และ qb หากเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (GUARDED เป็น FALSE) ค่า d และ NOT d จะไม่ถูกส่งให้กับ q และ qb



(a)

```
ENTITY d_flipflop IS
        GENERIC ( delay 1 : TIME := 4 ns; delay2 : TIME := 5 ns;
        PORT (d, c : IN BIT ; q, qb : OUT BIT);
END d_flipflop;
ARCHITECTURE guarding OF d_flipflop IS
BEGIN
        if : BLOCK ( c = '1' AND NOT c'STABLE )
                BEGIN
                        q <= GUARDED d AFTER delay1;
                        qb <= GUARDED NOT d AFTER delaly2;
                END  BLOCK if ;
END guarding;
```

(b)

รูปที่ 3.23 แสดงตัวอย่างการใช้ GUARDED

3.6.2.3 การเลือกสรรข้อมูล (Resolving Between Several Driving Values)

ในทางฮาร์ดแวร์มีหลายๆกรณีที่เรามีความจำเป็นต้องเชื่อมต่อหลายๆ เอาท์พุทไปยังอินพุทใดอินพุทหนึ่ง ซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกัน ทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทาง VHDL หมายถึง การกำหนดค่าจากหลายๆ สัญญาณให้กับสัญญาณเดียวซึ่งก็จะให้ผลลัพธ์ที่สับสนได้เหมือนกัน ในกรณีเช่นนี้เราสามารถแก้ไขได้โดย การกำหนดรูปแบบฟังก์ชั่นขึ้นมาเพื่อใช้เลือกสรรข้อมูล รูปที่ 3.24 แสดงตัวอย่างฟังก์ชั่นเลือกสรรข้อมูลแบบ AND ซึ่งกำหนดให้สัญญาณที่เข้ามาที่โหนดเดียวกันจะถูกนำมารวมกันในลักษณะของการ AND กันเสียก่อน ซึ่งสามารถแก้ปัญหาเรื่องความสับสนของข้อมูลได้

```
ARCHITECTURE wired_and OF y_circuit IS
    FUNCTION anding (drivers : BIT_VECTOR) RETURN BIT IS
        VARIABLE accumulate : BIT := '1' ;
    BEGIN
        FOR  i  IN drivers'RANGE LOOP
            accumulate := accumualte AND driver(i);
        END LOOP;
        RETURN accumualte;
    END anding;
    SIGNAL circuit_node : anding BIT;
BEGIN
    circuit_node <= a;
    circuit_node <= b;
    circuit_node <= c;
    circuit_node <= d;
    z <= circuit_node ;
END wired_and;
```

รูปที่ 3.24 แสดงการใช้ฟังก์ชั่น anding กับการเลือกสรรข้อมูล

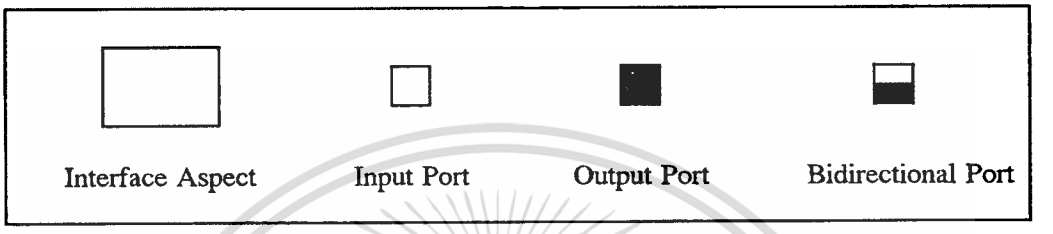
3.6.3 การบรรยายเชิงโครงสร้าง (Structure Specification of Hardware)

การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบนั้นและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่างๆ ด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด VHDL ได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ 1) ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี 2) การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้ 3) กลไกในการเชื่อมต่อระหว่างอุปกรณ์ 4) โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำๆ กัน

3.6.3.1 การกำหนดไลบรารี (Parts Libraly)

ในหัวข้อนี้จะแสดงการออกแบบเกทพื้นฐาน 3 ชนิดคือ INVERTER, NAND เกทชนิด 2 อินพุทและ 3 อินพุท ซึ่งเกทพื้นฐานเหล่านี้จะถูกเก็บไว้ในไลบรารีและนำมาใช้เป็นส่วนประกอบเพื่อออกแบบเป็นวงจรที่ใหญ่ขึ้นต่อไป

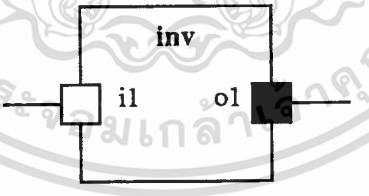
ในการออกแบบได้กำหนดสัญลักษณ์ที่เกี่ยวข้องไว้ดังนี้



รูปที่ 3.25 แสดงสัญลักษณ์แทนตัวอุปกรณ์

ก) อินเวอร์เตอร์โมเดล (Inverter Model)

รูปที่ 3.26 (a) แสดงสัญลักษณ์ของ Inverter รูป (b) แสดงการบรรยายการเชื่อมต่อโดยกำหนดให้อุปกรณ์มีชื่อว่า **inv** และมีพอร์ตการติดต่อ **i1** เป็นอินพุทและ **o1** เป็นเอาท์พุทมีชนิดของข้อมูลที่ผ่านเข้าออกเป็น **BIT** เราสามารถกำหนดพอร์ตเป็นแบบสองทิศทาง (Bidirectional) ได้โดยการกำหนดโหมดเป็น **INOUT** รูป (c) แสดงการบรรยายการทำงานของอุปกรณ์



(a)

```
ENTITY inv IS
        PORT ( i1 : IN BIT; o1 : OUT BIT );
END inv;
```

(b)

```
ARCHITECTURE single_delay OF inv IS
BEGIN
        o1 <= NOT  i1 AFTER 4 ns;
END single_delay;
```

(c)

รูปที่ 3.26 แสดง Inverter Model

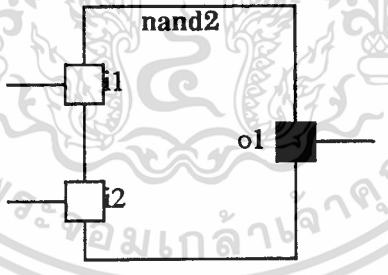
(a) แสดงสัญลักษณ์ของ Inverter

(b) แสดงการบรรยายการเชื่อม

(c) แสดงการบรรยายการทำงาน

ข) NAND เกทโมเดล (NAND Gate Model)

รูปที่ 3.27 แสดงสัญลักษณ์และการบรรยายของ NAND เกทชนิด 2 อินพุท



(a)

```
ENTITY nand2 IS
        PORT (i1,i2 : IN BIT; o1 : OUT BIT);
END nand2;
```

(b)

```
ARCHITECTURE single_delay OF nand2 IS
BEGIN
        o1 <= i1 NAND i2 AFTER 5 ns;
END single_delay;
```

(c)

รูปที่ 3.27 แสดง NAND2 Model

(a) แสดงสัญลักษณ์ของ NAND2

(b) แสดงการบรรยายการเชื่อม

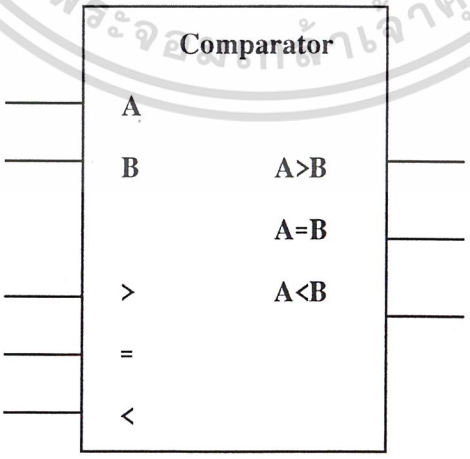
(c) แสดงการบรรยายการทำงาน

3.6.3.2 การเชื่อมต่ออุปกรณ์พื้นฐาน (Wiring of Primitive)

เมื่อเราได้ออกแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบโดยใช้อินเวอร์เตอร์และ NAND เกท

ตัวอย่างการออกแบบวงจรเปรียบเทียบ (Logic Design of Comparator)

วงจรเปรียบเทียบทีละบิต (Bit-comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุท สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 อินพุทดังรูปที่ 3.28



รูปที่ 3.28 แสดงวงจรเปรียบเทียบทีละบิต

เอาท์พุท A>B มีค่าเป็น '1' เมื่ออินพุท A มีค่ามากกว่า B (AB = 10) หรือถ้า A เท่ากับ B และอินพุท > มีค่าเป็น '1' เอาท์พุท A=B มีค่าเป็น '1' เมื่ออินพุท A เท่ากับ B และอินพุท = มีค่าเป็น '1' ส่วนเอาท์พุท A<B จะตรงข้ามกับเอาท์พุท A>B นั่นคือ มีค่าเป็น '1' เมื่ออินพุท A มีค่าน้อยกว่า B (AB = 01) หรือถ้า A เท่ากับ B และอินพุท < มีค่าเป็น '1'

ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต (VHDL Description of Bit-Comparator)

รูปที่ 3.29 แสดงการบรรยายการเชื่อมต่อของวงจรเปรียบเทียบทีละบิต เครื่องหมาย"--" ใช้แทนข้อความอธิบาย (comment)

```
ENTITY bit_comparator IS
  PORT ( a, b                  -- data inputs
         gt,                   -- previous greater than
         eq,                   -- previous equal
         It : IN BIT;          -- previous less than
         a_gt_b,               -- greater
         a_eq_b,               -- equal
         a_It_b : OUT BIT); -- less than
END bit_comparator;
```

รูปที่ 3.29 แสดงการบรรยายการเชื่อมต่อ

```
ARCHITECTURE gate_level OF bit_comparator IS
    COMPONENT n1 PORT (i1: IN BIT; o1: OUT BIT; END COMPONENT;
    COMPONENT n1 PORT (i1,i2: IN BIT; o1: OUT BIT; END COMPONENT;
    COMPONENT n1 PORT (i1,i2,i3: IN BIT; o1: OUT BIT; END COMPONENT;

    FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
    FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
    FOR ALL : n3 USE ENTITY WORK.nand3 (single_delay);
```

```
    SIGNAL im1,im2,im3,im4,im5,im6,im7,im8,im9,im10 : BIT;
BEGIN
    -- a_gt_b output
    go : n1 PORT MAP (A, im1);
    g1 : n1 PORT MAP (A, im2);
    g2 : n2 PORT MAP (A, im2, im3);
    g3 : n2 PORT MAP (A, gt, im4);
    g4 : n2 PORT MAP (im2, gt, im5);
    g5 : n3 PORT MAP (im3, im4, im5, gt, a_gt_b);
    -- a_gt_b output
    g6 : n3 PORT MAP (im1, im2, eq, im6);
    g7 : n3 PORT MAP (a, b, eq, im7);
    g8 : n2 PORT MAP (im6, im7, a_eq_b);
    -- a_lt_b output
    g9 : n2 PORT MAP (im1, im8);
    g10 : n2 PORT MAP (im1, lt, im9);
    g11 : n2 PORT MAP (b, lt, im10);
    g12 : n3 PORT MAP (im8, im9, im10, a_iT_b);
END gate_level;
```

รูปที่ 3.30 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต

บทที่ 4
โครงสร้างทางฮาร์ดแวร์ของวงจรแปลงแบบตัวประกอบปฐม

4.1 บทนำ

การเข้าและถอดรหัสเพื่อใช้ในการตรวจหาและ แก้ไขข้อมูลที่ผิดพลาด โดยใช้รหัสแบบรีด-โซโลมอน จะสามารถทำได้ทั้งเชิงเวลาและเชิงความถี่ โดยความยุ่งยากของการทำงานจะอยู่ที่การถอดรหัสซึ่งจะนิยมทำในเชิงความถี่ เนื่องจากในการถอดรหัสในเชิงความถี่นี้จะต้องใช้การแปลงและการแปลงกลับเป็นส่วนสำคัญ นอกเหนือไปจากการคำนวณอื่นๆ การแปลงและการแปลงกลับจะต้องใช้การคำนวณเป็นจำนวนมากจึงมีการนำวิธีการตัวประกอบปฐมมาใช้เพื่อลดจำนวนการคำนวณลงโดยใช้ร่วมกับวิธีการแปลงข้อมูลจำนวนน้อยๆ ในแง่ของการนำวิธีการดังกล่าวไปใช้งานจริง จะต้องพิจารณาเวลาที่ใช้ในการแปลงเป็นสำคัญจึงจำเป็นจะต้องมีการวิเคราะห์โครงสร้างการแปลงดังกล่าวในทางฮาร์ดแวร์

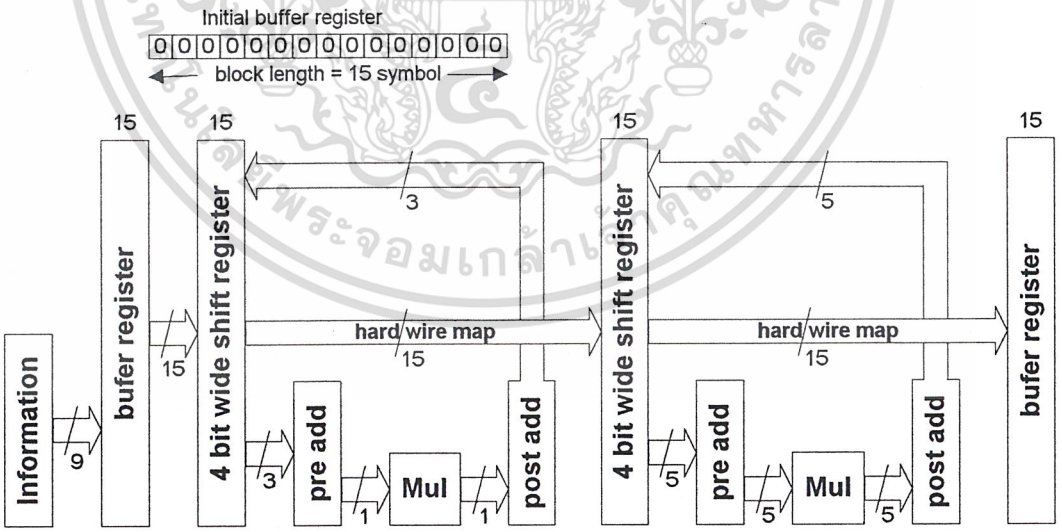
ในการแปลงและการแปลงกลับจะมีโครงสร้างการทำงานเหมือนกัน โดยจะมีการเปลี่ยนแปลงค่าสัมประสิทธิ์ที่ใช้เท่านั้น ซึ่งค่าสัมประสิทธิ์จะขึ้นอยู่กับ prime polynomial โดย prime polynomial ที่ใช้ในการออกแบบนี้คือ x^4+x+1 ซึ่งจะทำให้ขนาดของวงจรคูณแบบขนานชนิดตัวคูณคงที่มีขนาดเล็กที่สุด ในการพิจารณาเลือกใช้วงจรแปลงจะพิจารณาจำนวนคาบเวลาที่ใช้ในการแปลงและขนาดของวงจรแปลงเป็นสำคัญ รวมทั้งความสามารถในการปรับเปลี่ยนความเร็วของวงจรแปลงด้วย เนื่องจากต้องให้มีความเร็วในการแปลงสอดคล้องกับการทำงานในส่วนอื่นๆ ของการเข้าและถอดรหัสในขนาดที่เหมาะสม ในการพิจารณาเปรียบเทียบจะมีการนำ 2.0 micron standard cell (MDA 20) ของ motorola มาใช้อ้างอิงเพื่อให้เห็นถึงความแตกต่างและข้อจำกัดของฮาร์ดแวร์แต่ละแบบได้อย่างชัดเจนยิ่งขึ้น

4.2 การออกแบบโครงสร้างของวงจรที่ใช้ในการแปลง

การออกแบบโครงสร้างของวงจรที่ใช้ในการแปลง จะใช้วิธีการออกแบบโดยตรงจากขั้นตอนวิธี คือนำโครงสร้างการคำนวณจากบทที่ 2 มาออกแบบซึ่งวงจรแปลงจะแบ่งออกเป็น 2 ขั้น ดังแสดงในรูปที่ 4.1 จะเห็นว่าในการแปลงแต่ละขั้นจะประกอบในด้วยส่วนป้อนข้อมูล, ส่วนคำนวณ และส่วนรวบรวมข้อมูล โดยส่วนป้อนข้อมูลและรวบรวมข้อมูลจะใช้โครงสร้างเป็นบัฟเฟอร์เลื่อน (ออกแบบด้วย 2 อินพุท D F/F เนื่องจากต้องมีการถ่ายเทข้อมูลเข้าแบบขนานอีกด้วย) ซึ่งเป็นโครงสร้างที่ทำให้การป้อนข้อมูลให้กับส่วนคำนวณทำได้ง่าย อีกทั้งยังทำสามารถใช้

ตัวป้อนและรวบรวมข้อมูลร่วมกันได้เพื่อเป็นการลดขนาดของวงจรรวมเนื่องจากบัฟเฟอร์เลื่อนที่ใช้เป็นส่วนป้อนหรือรวบรวมข้อมูลนี้มีขนาดถึงประมาณ 510 เกท (2 อินพุท D F/F มีขนาด 8.5 เกท) สำหรับข้อมูลจำนวน 15 จุด

ส่วนป้อนข้อมูลจะป้อนข้อมูลให้กับส่วนคำนวณในแบบขนานและจะมีการเลื่อนข้อมูลชุดใหม่ให้มาอยู่ทางด้านล่างของส่วนป้อนข้อมูลเพื่อป้อนให้กับส่วนคำนวณในรอบการคำนวณถัดไป ซึ่งต้องใช้เวลาในการเลื่อนเท่ากับจำนวนข้อมูลที่ต้องใช้ในส่วนคำนวณ การเลื่อนจะทำแบบไม่ป้อนกลับเนื่องจากข้อมูลที่ถูกนำไปคำนวณแล้วจะไม่ถูกนำมาใช้ใหม่อีกทำให้เราสามารถจะเก็บผลลัพท์ที่ได้จากการคำนวณซึ่งมีขนาดเท่ากับข้อมูลที่นำมาใช้คำนวณลงที่ด้านบนของส่วนป้อนข้อมูลได้ จะเห็นว่าการเลื่อนทำไปอย่างต่อเนื่องทำให้เวลาที่สามารถใช้ได้ในส่วนคำนวณนับเป็นคาบเวลาจะเท่ากับจำนวนของข้อมูลที่ใช้ในส่วนคำนวณนั้นๆ นั่นเอง ซึ่งเท่ากับ 3 และ 5 คาบเวลาสำหรับการแปลง 3 และ 5 จุด ตามลำดับ จะเห็นได้ว่าเมื่อเราทำการเลื่อนข้อมูลจนครบ 15 คาบเวลา ก็จะเป็นการป้อนข้อมูลให้ส่วนคำนวณทำการคำนวณครบทั้ง 15 จุด ดังนั้นวงจรแปลงที่มีโครงสร้างแบบนี้จะมีความเร็วในการแปลง 15 คาบเวลาและมี latency time เป็น 30 คาบเวลา



รูปที่ 4.1 แสดงโครงสร้างวงจรแปลงที่ออกแบบโดยตรงจากขั้นตอนวิธี

เมื่อพิจารณาในส่วนคำนวณการแปลง 3 จุด จะเห็นว่าเวลาที่ใช้ในการทำ pre-add เป็น $2\tau_{xor(2)}$ เนื่องจากประกอบด้วยการบวก 2 ขั้น และในการคูณจะใช้เวลาเท่ากับ) $3\tau_{xor(4)}$ เมื่อใช้วงจรคูณแบบขนานชนิดตัวคูณคงที่ เนื่องจาก $4\tau_{xor(2)}<3\tau_{xor(4)}<2*4\tau_{xor(2)}$ ดังนั้นเราสามารถใช้คาบเวลานาฬิกาเป็น $4\tau_{xor(2)}$ ซึ่งจะทำให้ใช้เวลาในการ pre-add และ post-add รวมกันเป็น 1 คาบเวลา และการคูณใช้ 2 คาบ เวลารวมเป็น 3 คาบเวลาเท่ากับที่กำหนดไว้สำหรับการแปลง 3 จุด ในการแปลง 5 จุดจะใช้เวลาในการบวกเป็น $3\tau_{xor(2)}$ สำหรับการ pre-add และ post-add และใช้เวลาในการคูณเป็น $3\tau_{xor(4)}$ ดังนั้นจากคาบเวลาที่กำหนดเป็น $4\tau_{xor(2)}$ ทำให้เราใช้เวลาในส่วนการแปลง 5 จุดนี้เป็น 4 คาบเวลา

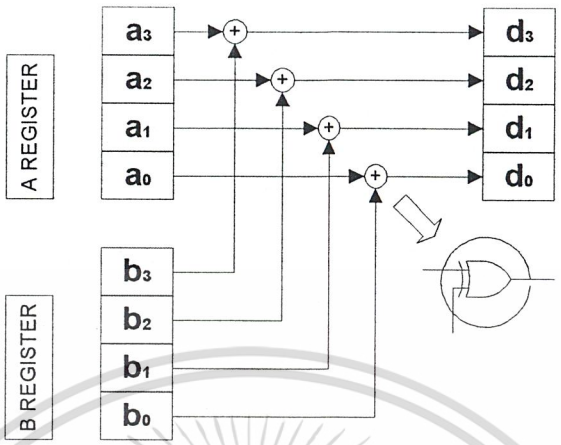
4.3 ส่วนคำนวณและอุปกรณ์ที่ใช้ในการออกแบบวงจรแปลง

ส่วนคำนวณที่ใช้ในการแปลงคือ ส่วนการบวกและส่วนการคูณซึ่งแบ่งออกเป็นส่วนคำนวณแบบขนานและแบบอนุกรม เนื่องจากข้อมูลที่ใช้ในการแปลงในสนามจำกัด $GF(2^4)$ จะแทนด้วยข้อมูลฐานสองขนาด 4 บิท ดังนั้นวงจรบวกแบบอนุกรมจะใช้เวลามากกว่าวงจรบวกแบบขนานอยู่ 4 เท่า ในขณะที่มีขนาดเล็กกว่าประมาณ 4 เท่าเช่นกัน ส่วนวงจรคูณนั้นนอกจากจะมีทั้งแบบขนานและแบบอนุกรมแล้วแต่ละแบบยังมีทั้งชนิดตัวคูณคงที่และตัวคูณทั่วไป โดยวงจรคูณแบบอนุกรมจะมีขนาดไม่น้อยไปกว่าวงจรคูณแบบขนานเท่าใดนักในขณะที่ต้องใช้เวลาในการคำนวณมากกว่ามาก จึงไม่เหมาะที่จะนำมาใช้ในการออกแบบ

ในการออกแบบส่วนคำนวณที่มีชนิดของข้อมูลเป็นทั้งแบบขนานและอนุกรมนั้นจะต้องมีการเพิ่มบัฟเฟอร์เพื่อเปลี่ยนชนิดของข้อมูลซึ่งทำให้เกิดความยุ่งยากในการออกแบบและยังต้องเพิ่มขนาดของวงจรในส่วนบัฟเฟอร์อีกด้วยดังนั้นในการออกแบบที่พิจารณาต่อไปจึงจะใช้เฉพาะวงจรคูณและวงจรบวกแบบขนานเท่านั้น

4.3.1 การบวกแบบขนาน

การบวกในสนามจำกัด $GF(2^4)$ สามารถใช้ Exclusive OR (XOR) เกทสร้างได้ เช่น $C = A \oplus B$ (อย่างไรก็ตามในปริญญานิพนธ์นี้จะใช้เครื่องหมายบวก (+) แทนเครื่องหมายการ XOR ($\oplus$)) โดยข้อมูลทั้ง 4 บิทจะถูกส่งไปบวกพร้อมกันจึงได้ผลลัพธ์ใน 1 คาบเวลา และมีเวลาในการหน่วง เท่ากับ τ_{XOR} นาโนวินาที แต่ต้องใช้ XOR เกท 4 ตัว จะมีโครงสร้างดังแสดงในรูปที่ 4.2



รูปที่ 4.2 โครงสร้างทางฮาร์ดแวร์ของวงจรบวกแบบขนาน

4.3.2 การคูณแบบขนานชนิดตัวคูณทั่วไป

ให้สมาชิกของสนามจำกัดแสดงโดย

$$A(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3$$

และ

$$B(x) = b_0 + b_1x^1 + b_2x^2 + b_3x^3$$

เมื่อ

$$D(x) = A(x)B(x) \bmod p(x)$$

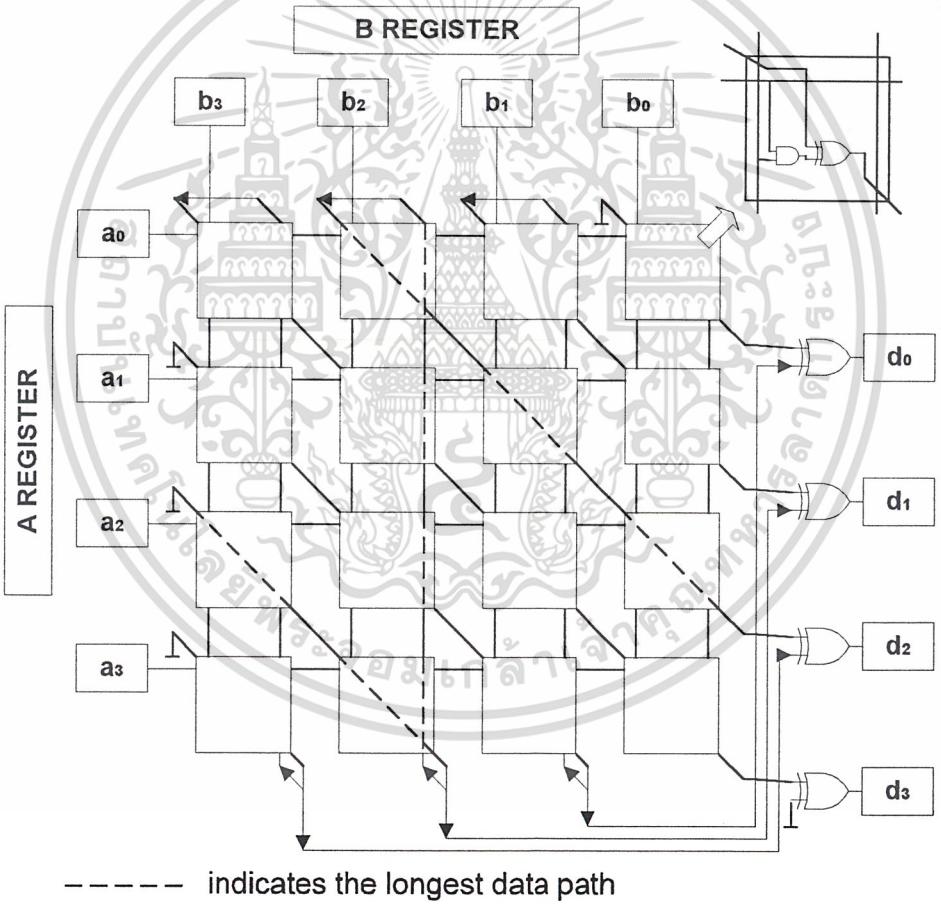
สัมประสิทธิ์ของ $D(x)$ คือ

$$d_k = \sum_{i=1}^{3} a_i b_{k-i} \qquad ; 0 \le \mathrm{k} \le 3$$

ซึ่ง $D(x)$ จะมีผลลัพธ์ดังนี้

$$\begin{aligned} D(x) = &\left(a_1b_3 + a_2b_2 + a_3b_1 + a_0b_0\right) \\ &+\left(a_1b_3 + a_2b_2 + a_3b_1 + a_0b_1 + a_1b_0 + a_2b_3 + a_3b_2\right)x \\ &+\left(a_2b_3 + a_3b_2 + a_0b_2 + a_1b_1 + a_2b_0 + a_3b_3\right)x^2 \\ &+\left(a_3b_3 + a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0\right)x^3 \end{aligned} \qquad (4.1)$$

สมการที่ (4.1) สามารถนำไปออกแบบเป็นฮาร์ดแวร์ได้ดังรูปที่ 4.3 โดยจะแบ่งออกเป็นเซล 16 เซลที่เหมือนกัน ในแต่ละเซลประกอบด้วย 2 อินพุท AND เกท 16 ตัว และ 2 อินพุท XOR เกท 20 ตัว โดยหน่วยความจำ A และ B จะป้อนข้อมูลอินพุทแบบขนานให้ A(x) และ B(x) ตามลำดับ และผลลัพธ์จะถูกหน่วงประมาณ $5\tau_{cell} + \tau_{XOR}$ เมื่อ τ_{cell} คือเวลาที่ถูกหน่วงใน 1 เซล และ τ_{XOR} คือเวลาที่ถูกหน่วงจาก 2 อินพุท XOR เกท ซึ่งเวลาที่ถูกหน่วงใน 1 เซล จะเท่ากับเวลาที่ถูกหน่วงจาก 2 อินพุท XOR เกท เพราะฉนั้นเวลาที่ถูกหน่วงทั้งหมดคือ $6\tau_{XOR}$ แต่ผลลัพธ์จะได้ภายใน 1 คาบเวลา



รูปที่ 4.3 โครงสร้างทางฮาร์ดแวร์ของวงจรคูณแบบขนานชนิดตัวคูณทั่วไป

4.3.3 การแปลงในสนามจำกัดขนาด 3 จุด

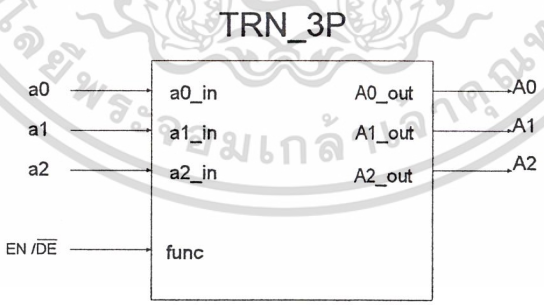
จากสมการที่ (2.26) คือสมการแปลงในสนามจำกัดขนาด 3 จุด

$$A_k = \sum_{n=0}^{2} a_n \beta^{\langle nk \rangle_3} \qquad ; \text{k=0,1,2 และ } \beta = \alpha^5$$

สามารถนำมาพิจารณาจำนวนร่วมมากได้ดังนี้

$$s_1 = a_1 + a_2 \quad , A_0 = s_1 + a_0 \quad , m_1 = \beta s_1$$
$$s_2 = A_0 + m_1 \quad , A_1 = s_2 + a_1 \quad , A_2 = s_2 + a_2$$

และจากผังการคำนวณของวิธีการแปลงข้อมูลจำนวนน้อยๆสำหรับการแปลง 3 จุดในรูปที่ 2.1 สามารถนำมาออกแบบเป็นฮาร์ดแวร์ของการแปลงในสนามจำกัดขนาด 3 จุดได้ดังรูป 4.4 จะมีอินพุทคือ ขา a0,a1,a2 ทำหน้าที่นำข้อมูลเข้าไปทำการแปลง 3 จุด และขา func ทำหน้าที่เลือกค่าคงที่ β สำหรับการเข้ารหัสหรือการถอดรหัส ที่จะนำไปทำการคูณในวงจร โดยภายในวงจรนี้จะนำเอา วงจรบวกและวงจรคูณมาทำการเชื่อมต่อกันตามผังการคำนวณสำหรับการแปลง 3 จุด และจะได้ค่าเอาท์พุทออกมาที่ขา A0,A1,A2



รูปที่ 4.4 บล็อคไดอะแกรมของการแปลงในสนามจำกัดขนาด 3 จุด

4.3.4 การแปลงในสนามจำกัดขนาด 5 จุด

จากสมการที่ (2.26) คือสมการแปลงในสนามจำกัดขนาด 5 จุด

$$A_k = \sum_{n=0}^{4} a_n \beta^{\langle nk \rangle_5} \qquad ; \text{k=0,1,2,3,4 และ } \beta = \alpha^3$$

สามารถนำมาพิจารณาจำนวนร่วมมากได้ดังนี้

$$s_1 = a_2 + a_3 \quad , s_2 = a_1 + a_4 \quad , s_3 = a_1 + a_3$$

$$s_4 = a_2 + a_4 \quad , s_5 = s_1 + s_2 \quad , A_0 = s_5 + a_0$$

$$m_1 = (1+\beta^3)s_5 \quad , m_2 = (\beta^3 + \beta^4)s_1 \quad , m_3 = (\beta + \beta^3)s_2$$

$$m_4 = (\beta + \beta^4)s_3 \quad , m_5 = (\beta + \beta^4)s_4 \quad , s_6 = A_0 + m_1$$

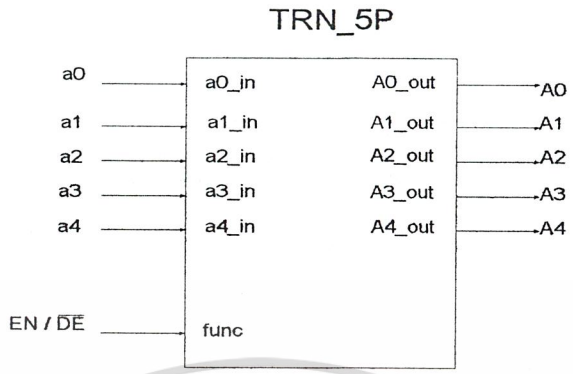
$$s_7 = s_6 + m_2 \quad , s_8 = s_6 + m_3 \quad , s_9 = m_5 + a_2$$

$$s_{10} = m_4 + a_1 \quad , s_{11} = m_5 + a_4 \quad , s_{12} = m_4 + a_3$$

$$A_1 = s_8 + s_9 \quad , A_2 = s_7 + s_{10} \quad , A_3 = s_7 + s_{11}$$

$$A_4 = s_8 + s_{12}$$

และจากผังการคำนวณของวิธีการแปลงข้อมูลจำนวนน้อยๆสำหรับการแปลง 5 จุดในรูปที่ 2.2 สามารถนำมาออกแบบเป็นฮาร์ดแวร์ของการแปลงในสนามจำกัดขนาด 5 จุดได้ดังรูป 4.5 จะมีอินพุทคือ ขา a0,a1,a2,a3,a4 ทำหน้าที่นำข้อมูลเข้าไปทำการแปลง 5 จุด และขา func ทำหน้าที่เลือกค่าคงที่ β สำหรับการเข้ารหัสหรือการถอดรหัส ที่จะนำไปทำการคูณในวงจร โดยภายในวงจรนี้จะนำเอาวงจรบวกและวงจรคูณมาทำการเชื่อมต่อกันตามผังการคำนวณสำหรับการแปลง 5 จุดและจะได้ค่าเอาท์พุทออกมาที่ขา A0,A1,A2,A3,A4



รูปที่ 4.5 บล็อคไดอะแกรมของการแปลงในสนามจำกัดขนาด 5 จุด

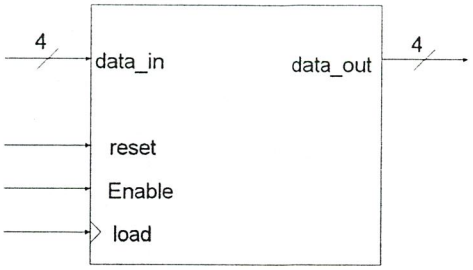
4.4 ส่วนรีจีสเตอร์เซลและอุปกรณ์เลือกอินพุท

ส่วนรีจีสเตอร์เซลซึ่งจะต้องทำหน้าที่ทั้งรับข้อมูลอินพุท,รับข้อมูลจากรีจีสเตอร์เซลอื่นเพื่อให้เป็นรีจีสเตอร์เลื่อน,รับข้อมูลจากเอาท์พุทของการแปลงในสนามจำกัดขนาด 3 จุด และรับข้อมูลของการแปลงในสนามจำกัดขนาด 5 จุด ดังนั้นจึงต้องมีอุปกรณ์สำหรับเลือกอินพุทที่จะมาเข้าขาอินพุทของรีจีสเตอร์เซล (มัลติเพล็กเซอร์) ซึ่งจะถูกควบคุมการทำงานด้วยสเตจแมทชีน

4.4.1 รีจีสเตอร์เซล

รีจีสเตอร์เซลทำหน้าที่เก็บข้อมูลเพื่อนำไปทำการแปลงและทำการการแปลงกลับ เมื่อทำการแปลงในสนามจำกัดขนาด 3 จุด และทำการแปลงในสนามจำกัดขนาด 5 จุดเสร็จสิ้นก็จะทำหน้าที่เก็บค่าเอาท์พุทที่ได้จากการแปลง และทำหน้าที่เป็นชิพรีจีสเตอร์เพื่อป้อนอินพุทค่าใหม่ให้วงจรแปลงในสนามจำกัดขนาด 3 จุดและวงจรแปลงในสนามจำกัดขนาด 5 จุด

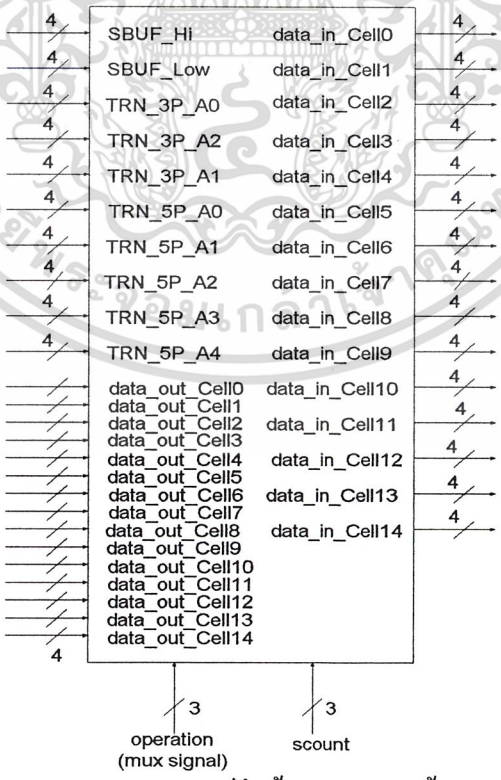
รีจีสเตอร์เซล 1 เซลจะประกอบด้วยบล็อกไดอะแกรมดังรูปที่ 4.6 โดยมีขารับและส่งข้อมูลขนาด 4 บิทคือขา data_in และขา data_out ตามลำดับ ก่อนที่จะทำการเก็บข้อมูลเพื่อนำไปทำการแปลงและทำการแปลงกลับรีจีสเตอร์เซลจะถูกรีเซ็ต(reset) เพื่อให้ขา data_out เป็นศูนย์ทั้ง 4 บิทโดยจะมีสัญญาณเข้ามาที่ขา reset และถ้ามีสัญญาณที่ขา Enable เป็นหนึ่ง สัญญาณที่ขา load เป็นขาขึ้นข้อมูลที่ขา data_out ก็จะเปลี่ยนเป็นข้อมูลที่มารออยู่ที่ขา data_in โดยรีจีสเตอร์เซลจะมีเซลทั้งหมด 15 เซล แต่ละเซลมีข้อมูล 4 บิท เพราะฉะนั้นจึงมี D F/F ทั้งหมด 60 ตัว



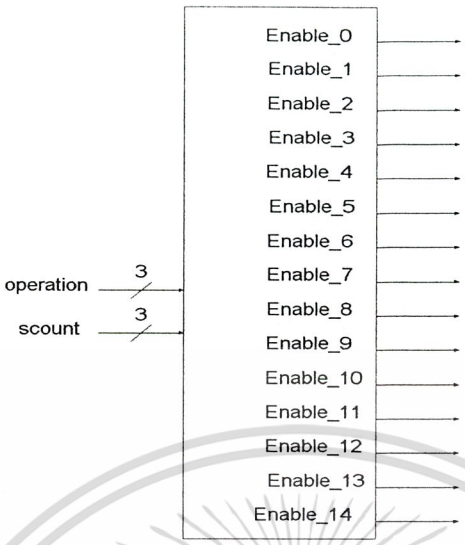
รูปที่ 4.6 บล็อคไดอะแกรมของรีจีสเตอร์เซล 1 เซล

4.4.2 อุปกรณ์เลือกอินพุท (มัลติเพล็กเซอร์)

เป็นอุปกรณ์ที่ใช้เลือกอินพุทเข้ารีจีสเตอร์ โดยจะมี 2 ส่วนคือ ส่วนของมัลติเพล็กเซอร์ที่ใช้เลือกอินพุทเข้าขา data_in ของรีจีสเตอร์ และ ส่วนของมัลติเพล็กเซอร์ที่ใช้เลือกอินพุทเข้าขา enable ของรีจีสเตอร์ โดยการเลือกอินพุทของมัลติเพล็กเซอร์ทั้งสองส่วนนี้เพื่อใช้รับข้อมูลอินพุท,รับข้อมูลจากรีจีสเตอร์เซลอื่นเพื่อให้เป็นรีจีสเตอร์เลื่อน,รับข้อมูลจากเอาท์พุทของการแปลงในสนามจำกัดขนาด 3 จุด และรับข้อมูลของการแปลงในสนามจำกัดขนาด 5 จุด โดยจะถูกเลือกตามสัญญาณ operation และ scount ที่ส่งมาจากสเตจแมทชีน



(a) บล็อคไดอะแกรมของมัลติเพล็กเซอร์ที่ใช้เลือกอินพุทเข้าขา data_in ของรีจีสเตอร์



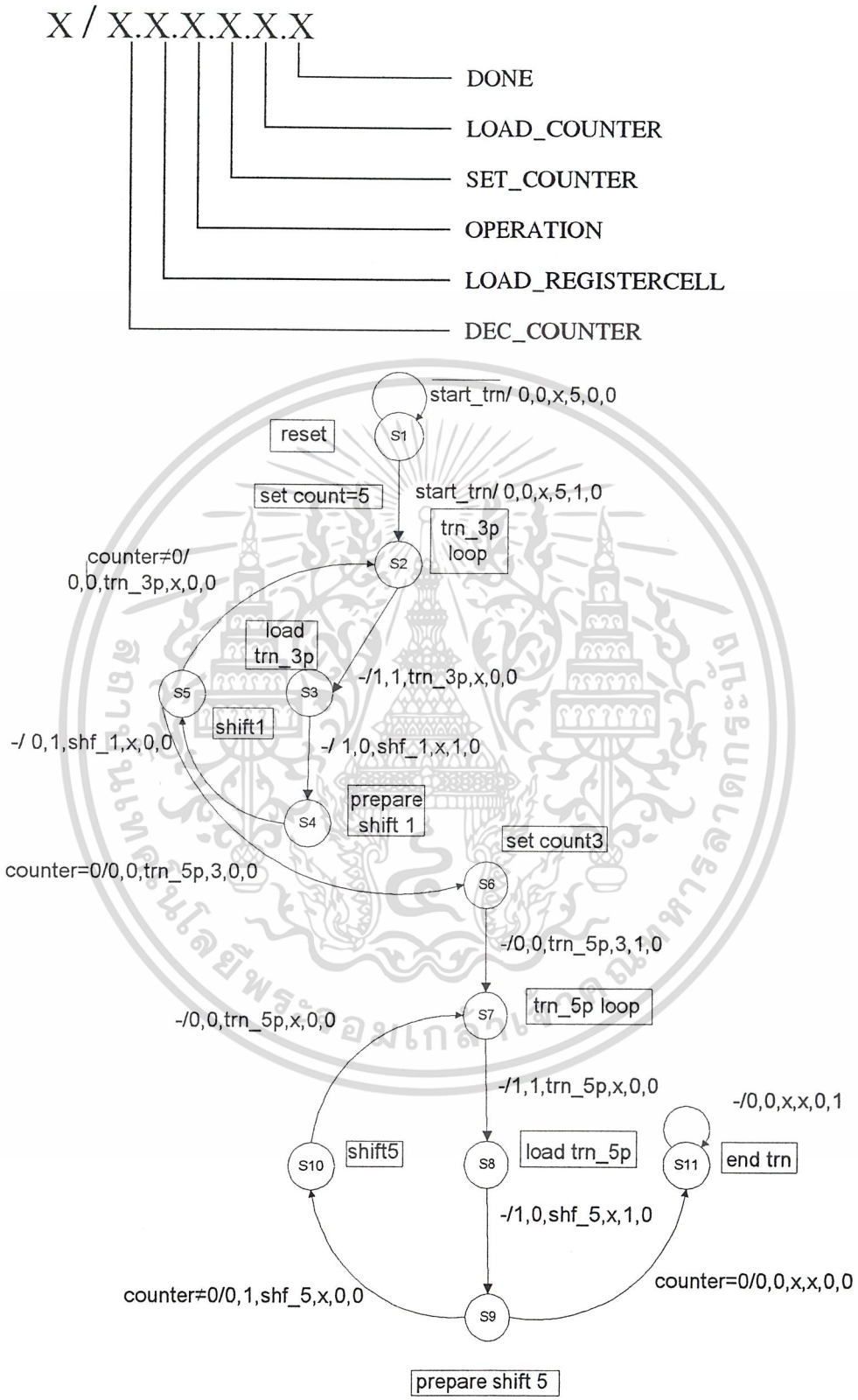
(b) บล็อคไดอะแกรมของมัลติเพล็กเซอร์ที่ใช้เลือกอินพุทเข้าขา enable ของรีจีสเตอร์

รูปที่ 4.7 บล็อกไดอะแกรมของอุปกรณ์เลือกอินพุท

4.4.3 สเตจแมทชีนของการแปลงและแปลงกลับ

สเตจแมทชีนของการแปลงและแปลงกลับจะทำหน้าที่ควบคุมการป้อนอินพุทเข้ารีจีสเตอร์เซลในส่วนของการแปลงและการแปลงกลับ โดยจะเริ่มทำงานเมื่อมีอินพุท start_trn เป็นหนึ่ง ในการทำงานจะทำการแปลงในสนามจำกัดขนาด 3 จุด แล้วจะทำการชิพไป 1 เซลเพื่อเปลี่ยนอินพุทในการทำการแปลงในสนามจำกัดขนาด 3 จุด เมื่อทำการแปลงในสนามจำกัดขนาด 3 จุด ครบ 5 ครั้ง ค่าของ counter จะกลายเป็นศูนย์ ทำให้เปลี่ยนสเตจไปทำการแปลงในสนามจำกัดขนาด 5 จุด และชิพรีจีสเตอร์เซลไปทีละ 5 เซลเพื่อเปลี่ยนอินพุทในการทำการแปลงในสนามจำกัดขนาด 5 จุด เมื่อทำการแปลงในสนามจำกัดขนาด 5 จุด ครบ 3 ครั้งค่าของ counter จะกลายเป็นศูนย์ ทำให้สเตจแมทชีนเปลี่ยนเป็นสเตจจบการทำงาน

ในการเปลี่ยนสเตจแต่ละครั้งเอาท์พุทของสเตจแมทชีนจะเปลี่ยนไปดังรูปที่ 4.8 จะแสดงสเตจแมทชีนของการแปลงและการแปลงกลับ โดยแสดงถึงการเปลี่ยนสเตจเมื่อมีอินพุทต่างๆ และค่าเอาท์พุทที่สเตจต่างๆ โดยมีเอาท์พุทคือ SET_COUNTER , DEC_COUNTER และ LOAD_COUNTER ทำการนับให้ counter วนลูปครบ 3 และ 5 ครั้ง , เอาท์พุท LOAD_CELL จะไปเป็นสัญญาณบอกให้รีจีสเตอร์เซลโหลดข้อมูลเข้าเซล , เอาท์พุท OPERATION เป็นสัญญาณเลือกอินพุทเข้ารีจีสเตอร์เซลโดยมัลติเพล็กเซอร์ และเอาท์พุท DONE เป็นสัญญาณบอกว่าการทำการแปลงเสร็จแล้ว โดยตำแหน่งของเอาท์พุทจะแทนสัญญาณเอาท์พุทต่างๆ ดังนี้



รูปที่ 4.8 สเตจแมทชีนของการแปลงและแปลงกลับ

4.5 ส่วนการรับส่งข้อมูลกับพอร์ตอนุกรม

ส่วนการรับส่งข้อมูลกับพอร์ตอนุกรมจะประกอบด้วยการรับข้อมูลแบบอนุกรมจากสายส่งแบบอนุกรมแล้วส่งข้อมูลแบบขนานให้กับชิพเพื่อนำไปทำการแปลงหรือแปลงกลับ เมื่อทำการแปลงหรือแปลงกลับเสร็จแล้วก็จะนำข้อมูลเอาท์พุทซึ่งเป็นแบบขนานผ่านเข้าไปในอุปกรณ์แปลงเป็นข้อมูลแบบอนุกรมแล้วส่งไปในสายส่งแบบอนุกรมต่อไป

ในส่วนการรับข้อมูลกับพอร์ตอนุกรมจะประกอบด้วยบัฟเฟอร์รีจีสเตอร์ทำการรับข้อมูลแบบอนุกรมมาเก็บไว้ทีละ 1 บิท เมื่อนำมาเก็บครบ 8 บิทก็จะส่งข้อมูลแบบขนานทั้ง 8 บิทให้กับมัลติเพล็กเซอร์เพื่อนำข้อมูลไปทำการแปลงหรือแปลงกลับต่อไป เมื่อทำการแปลงหรือแปลงกลับเสร็จสิ้นแล้วก็จะส่งข้อมูลกลับพอร์ตอนุกรมโดยจะประกอบด้วยอุปกรณ์ที่นำเอาข้อมูลที่ผ่านการแปลงและการแปลงกลับแล้วจากรีจีสเตอร์เซลมาทีละ 2 เซล เพื่อให้กลายเป็นข้อมูลขนาด 8 บิท แล้วนำไปใส่ในบัฟเฟอร์รีจีสเตอร์ แล้วข้อมูลจะถูกส่งแบบอนุกรมทีละ 1 บิท จนครบ 8 บิท ในการส่งและรับข้อมูลกับพอร์ตอนุกรมจะต้องใช้บัฟเฟอร์รีจีสเตอร์ตัวเดียวกัน ดังนั้นจึงต้องสร้างสเตจแมทชีนขึ้นมาคอยควบคุมบัฟเฟอร์รีจีสเตอร์ให้ส่งข้อมูลและรับข้อมูลจากพอร์ตอนุกรมไม่พร้อมกัน และในการแปลงและการแปลงกลับจะต้องใช้ข้อมูลจำนวน 15 เซล แต่การรับหรือส่งข้อมูลกับพอร์ตอนุกรมสามารถทำได้เพียงครั้งละ 8 บิท จึงต้องทำการรับหรือส่งข้อมูล 8 ครั้งจึงได้ข้อมูลเพียงพอที่จะนำมาทำการแปลงหรือแปลงกลับ ดังนั้นจึงต้องสร้างสเตจแมทชีนขึ้นมาควบคุมให้ทำการรับหรือส่งข้อมูล 8 ครั้ง

4.5.1 อุปกรณ์เลือกเอาท์พุท

อุปกรณ์เลือกเอาท์พุทจะทำหน้าที่เลือกข้อมูลที่ทำการแปลงหรือแปลงกลับเสร็จสิ้นแล้วจากรีจีสเตอร์เซลมาทีละ 2 เซลโดยจะเลือกข้อมูลตามลำดับของสมการที่ (2.25) คือสมการการจับคู่ดัชนีเอาท์พุท

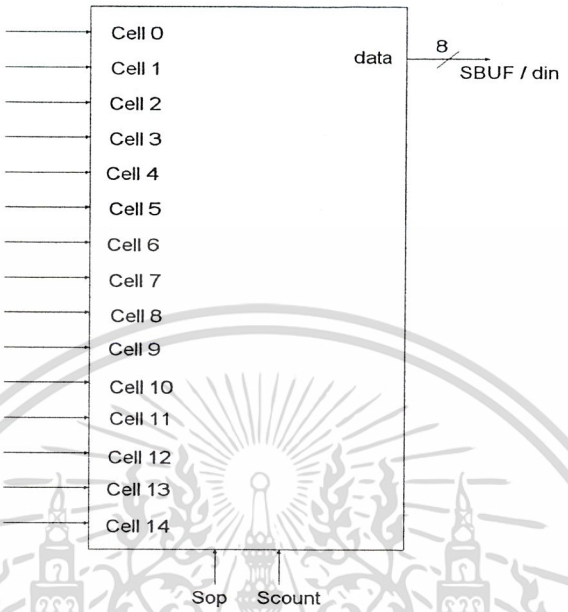
$$K^1 = \langle 3n' + 10k_1 \rangle_{15}$$

หรือถ้าจะทำการส่งข้อมูลที่รับมาก่อนการแปลงหรือแปลงกลับก็จะเลือกข้อมูลตามสมการการจับคู่ดัชนีอินพุทดังสมการ

$$N^1 = \langle 3n' + 5n_1 \rangle_{15}$$

โดยในอุปกรณ์เลือกเอาท์พุทตามรูปที่ 4.9 ก็จะมีอินพุทที่มาจากเอาท์พุทของรีจีสเตอร์เซลทั้ง 15 เซล และถูกควบคุมการเลือกโดยสัญญาณอินพุท Sop และ Scount ที่มาจากสเตจแมทชีน

และเอาท์พุทขา data จะมีขนาด 8 บิท และจะถูกส่งไปยังบัฟเฟอร์รีจีสเตอร์ต่อไป

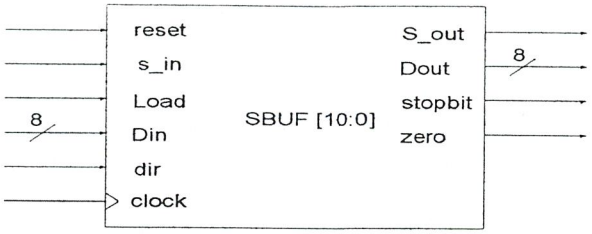


รูปที่ 4.9 บล็อกไดอะแกรมของอุปกรณ์เลือกเอาท์พุท

4.5.2 บัฟเฟอร์รีจีสเตอร์

บัฟเฟอร์รีจีสเตอร์เป็นอุปกรณ์ที่ทำหน้าที่รับหรือส่งข้อมูลขนาด 8 บิทระหว่างตัวชิพกับพอร์ตอนุกรม โครงสร้างของบัฟเฟอร์รีจีสเตอร์จะประกอบด้วย D F/F อยู่ 11 ตัว เนื่องจากการส่งข้อมูลแบบอนุกรมนั้นนอกจากจะมีข้อมูลขนาด 8 บิทแล้วยังมีสตาร์ทบิทและสตอปบิทซึ่งเป็นโปรโตคอลในการรับและส่งข้อมูลแบบขนานอีกด้วย

บัฟเฟอร์รีจีสเตอร์จะมีบล็อกไดอะแกรมดังรูปที่ 4.10 ซึ่งจะมีขาอินพุทต่างๆดังนี้ ขา reset จะทำการรีเซ็ตบัฟเฟอร์รีจีสเตอร์ก่อนที่จะทำการรับข้อมูล ขา s_in คือขาที่รับข้อมูลจากพอร์ตอนุกรมเข้าสู่บัฟเฟอร์รีจีสเตอร์ ขา Din คือขาที่รับข้อมูลแบบขนานขนาด 8 บิทจากชิพเข้าบัฟเฟอร์รีจีสเตอร์โดยจะทำการรับเมื่อมีสัญญาณจากสเตจแมทชีนเข้ามาที่ขา Load เมื่อสัญญาณเข้าที่ขา clock เป็นขาขึ้นจะทำการโหลดข้อมูลเข้าบัฟเฟอร์รีจีสเตอร์ โดยข้อมูลจะถูกกำหนดโดยสัญญาณที่ขา dir ว่าจะให้รับหรือส่งข้อมูล โดยสัญญาณ clock และ load จะถูกกำหนดจากสเตจแมทชีน และจะมีขาเอาท์พุทดังนี้ s_out และขา Dout คือขาที่ใช้ส่งข้อมูลในบัฟเฟอร์รีจีสเตอร์ไปยังพอร์ตอนุกรมและรีจีสเตอร์เซลตามลำดับ ขา stopbit คือขาที่เชื่อมต่อกับบิทที่ 10 ของบัฟเฟอร์รีจีสเตอร์ และขา zero คือขาแสดงว่าบิทที่ 10 ถึงบิทที่ 1 ของบัฟเฟอร์รีจีสเตอร์มีค่าเป็นศูนย์ทั้งหมดหรือไม่



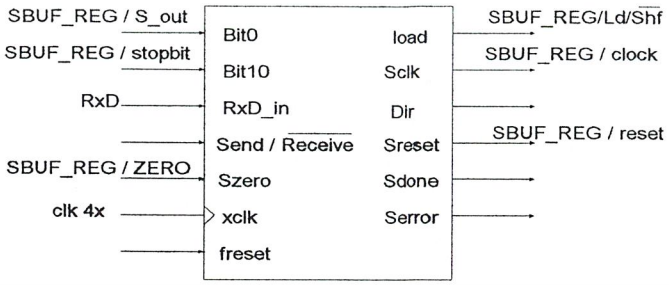
รูปที่ 4.10 บล็อคไดอะแกรมของบัฟเฟอร์รีจีสเตอร์

4.5.3 สเตจแมทชีนควบคุมการรับส่งของชิพ

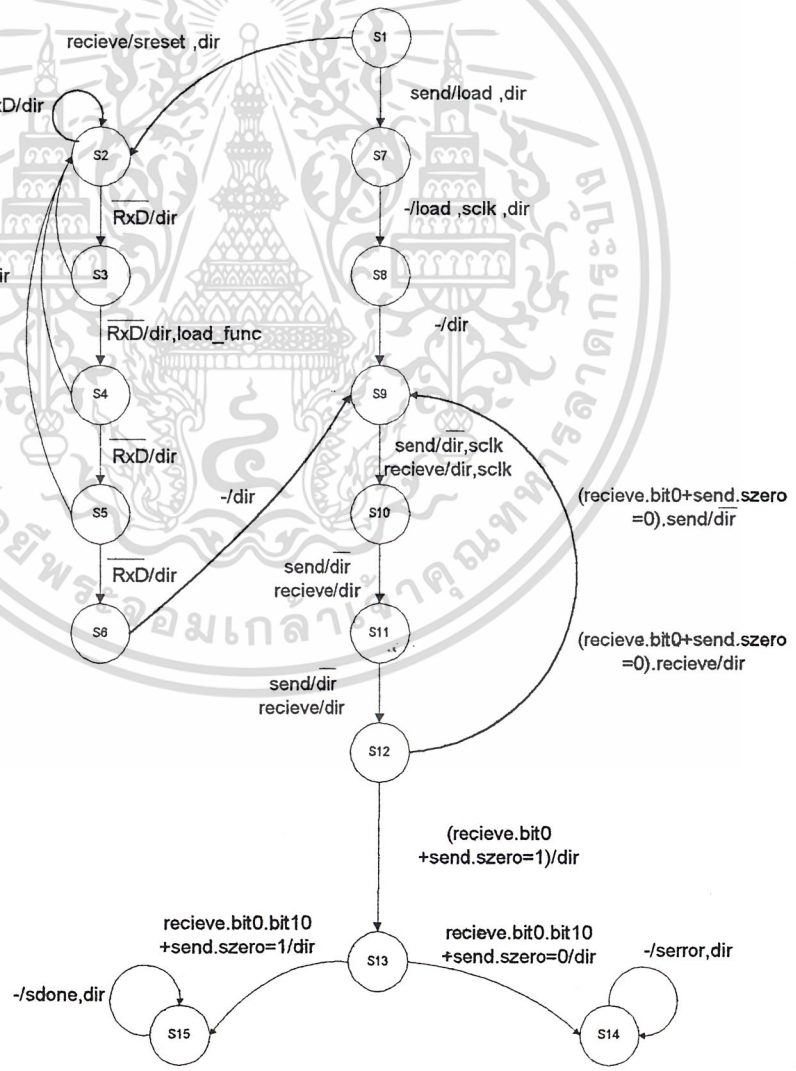
สเตจแมทชีนควบคุมการรับส่งของชิพจะทำงานร่วมกับบัฟเฟอร์รีจีสเตอร์ในการรับหรือส่งข้อมูลระหว่างชิพกับพอร์ตอนุกรม และจะเริ่มทำงานเมื่อสัญญาณ freset เป็นขาลง และถ้าขา Send/Recieve มีค่าเป็น 1 สเตจแมทชีนจะเปลี่ยนเป็นการส่งข้อมูลของชิพ และถ้าขา Send/Recieve มีค่าเป็น 0 สเตจแมทชีนจะเปลี่ยนเป็นการรับข้อมูล เนื่องจากเป็นการรับและส่งสัญญาณกับพอร์ตอนุกรมซึ่งถ้ากำหนดให้มี Buad rate ในการส่งเป็น 9600 Hz จะต้องใช้สัญญาณ clock ในการเปลี่ยนสเตจที่มีความถี่เป็น 4 เท่าของ Buad rate ดังนั้นจึงต้องสร้าง clock ใหม่จาก clock ของชิพเดิม ป้อนเข้าขา xclk ของสเตจแมทชีน

ในการรับข้อมูลจากพอร์ตอนุกรมสเตจแมทชีนจะต้องส่งสัญญาณไปทำการรีเซ็ตบัฟเฟอร์รีจีสเตอร์ ให้มีค่าเป็น “010000000” ก่อนและให้ dir เป็น 1 เพื่อกำหนดให้บัฟเฟอร์รีจีสเตอร์รับข้อมูล แล้วสเตจแมทชีนจะมาวนลูปรอที่สเตจ S2 จนกว่าจะมีสัญญาณที่ขา RxD ที่เป็น 0 เข้ามา (สตาร์ทบิท) จึงจะวนลูปรับข้อมูลที่สเตจ S9,S10,S11,S12 จากพอร์ตอนุกรมโดยบิท 9 ของบัฟเฟอร์รีจีสเตอร์จะชิพมาจนกระทั่งมาอยู่ที่บิท 0 จึงสิ้นสุดการรับข้อมูล

ในการส่งข้อมูลจากพอร์ตอนุกรมสเตจแมทชีนจะต้องส่งสัญญาณ load และ sclk เพื่อโหลดข้อมูลแบบขนานที่จะทำการส่งเข้าบัฟเฟอร์รีจีสเตอร์ และกำหนดบิทอื่นๆของบัฟเฟอร์รีจีสเตอร์ เป็น “1”& Din & “01” โดยมีบิท 1 เป็นสตาร์ทบิท และบิท 10 เป็นสตอปบิท และกำหนดสัญญาณ dir ให้มีค่าเป็น 0 เพื่อป้อนค่า 0 เข้าไปในบัฟเฟอร์รีจีสเตอร์ แล้วสเตจแมทชีนจะทำการวนลูปส่งข้อมูล ในการวนลูปผ่าน 4 สเตจคือสเตจ S9,S10,S11,S12 จะส่งข้อมูลออกไป 1 บิท โดยทำการชิพบัฟเฟอร์รีจีสเตอร์ไป 1 บิท เมื่อชิพจน บิท 1 ถึงบิท 10 ในบัฟเฟอร์รีจีสเตอร์มีค่าเป็น 0 ทั้งหมดจึงสิ้นสุดการส่งข้อมูล และรูปที่ 4.11 คือบล็อคแสดงอินพุทเอาท์พุทของสเตจแมทชีนควบคุมการรับส่งของชิพ โดยค่าเอาท์พุทต่างๆของสเตทแมทชีนจะแสดงในรูปที่ 4.12



รูปที่ 4.11 บล็อคแสดงอินพุทเอาท์พุทของสเตจแมทชีนควบคุมการรับส่งของชิพ

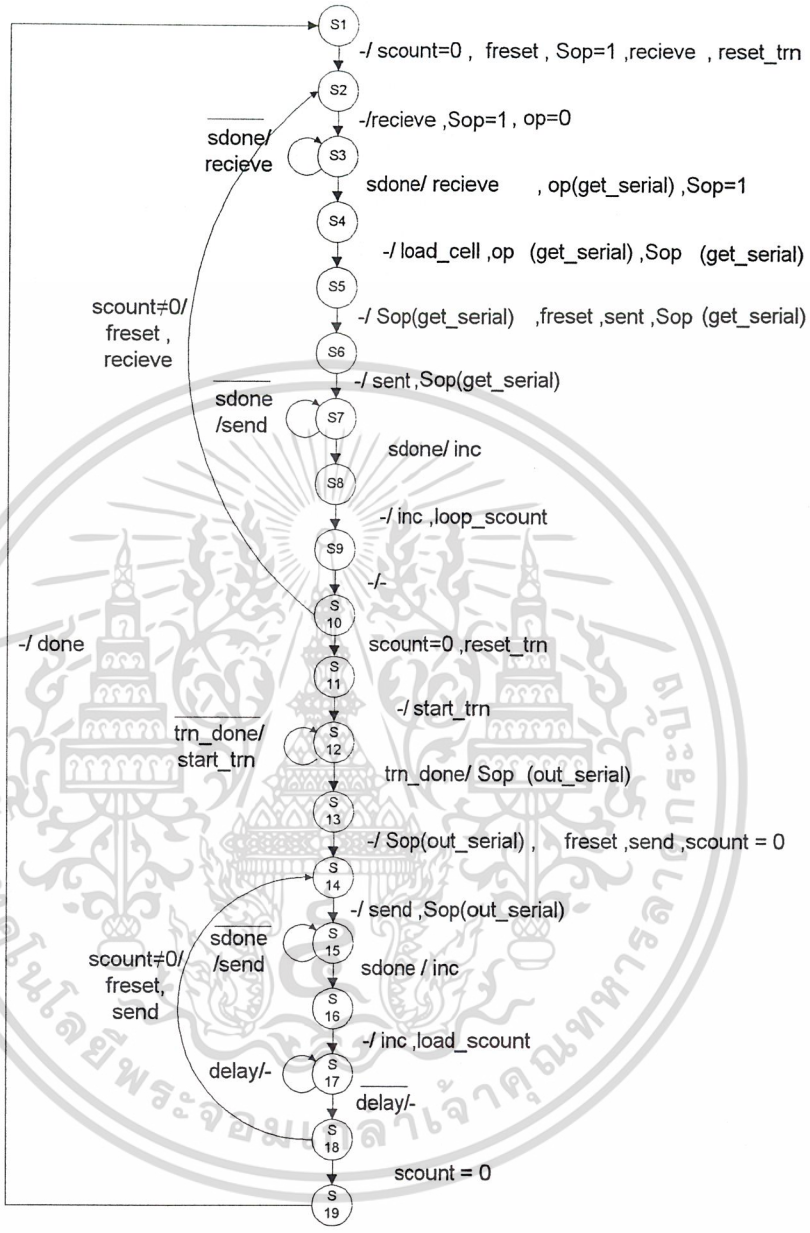


รูปที่ 4.12 สเตจแมทชีนควบคุมการรับส่งของชิพ

4.5.4 สเตจแมทชีนควบคุมการทำงานของชิพ

สเตจแมทชีนควบคุมการทำงานของชิพทำหน้าที่ควบคุมการรับข้อมูลทั้งหมดที่จะนำมาทำการแปลงหรือแปลงกลับ เมื่อทำการแปลงหรือแปลงกลับเสร็จสิ้นแล้ว ก็จะทำการส่งข้อมูลทั้งหมดที่ผ่านการแปลงหรือแปลงกลับออกจากชิพจึงเสร็จสิ้นการทำงาน 1 รอบ

โดยเมื่อมีสัญญาณรีเซ็ตเข้ามาสเตจจะเปลี่ยนเป็น S1 แล้วส่งสัญญาณไปทำการรีเซ็ตสเตจแมทชีนของการแปลงและการแปลงกลับและสเตจแมทชีนควบคุมการรับส่งของชิพ และจะรีเซ็ต scount ให้เป็น 0 แล้วจึงมาวนลูปที่สเตจ S3 รอให้สเตจแมทชีนควบคุมการทำงานของชิพทำงานจนเสร็จ ก็จะรับข้อมูลได้ 8 บิทซึ่งนำมาเก็บไว้ที่รีจีสเตอร์เซลได้ 2 เซล และจะเพิ่มค่า scount ขึ้น 1 ค่าและจะวนลูปผ่านสเตจ S3,S4,S5,S6,S7,S8,S9,S10,S2 มารับข้อมูลใหม่ และเพิ่มค่า scount ขึ้นอีก 1 จนกระทั่ง scount มีค่าเป็น 0 ใหม่ก็จะทำการรับข้อมูลครบทั้ง 15 เซล แล้วจะทำการแปลงหรือแปลงกลับ โดยส่งสัญญาณ Start_trn ไปที่สเตจแมทชีนของการแปลงและการแปลงกลับ และวนลูปรอที่สเตจ S12 จนกระทั่งทำการแปลงหรือการแปลงกลับเสร็จ สัญญาณ Trn_done จะกลายเป็น 1 แล้วส่งสัญญาณไปทำการรีเซ็ตสเตจแมทชีนควบคุมการรับส่งของชิพและจะรีเซ็ต scount ให้เป็น 0 แล้วจึงมาวนลูปที่สเตจ S15 รอให้สเตจแมทชีนควบคุมการทำงานของชิพทำงานจนเสร็จ ก็จะส่งข้อมูลได้ 8 บิท และจะเพิ่มค่า scount ขึ้น 1 ค่า และจะวนลูปผ่านสเตจ S16,S17,S18,S14,S15 มาส่งข้อมูลใหม่ และเพิ่มค่า scount ขึ้นอีก 1 จนกระทั่ง scount มีค่าเป็น 0 ใหม่ก็จะทำการส่งข้อมูลครบทั้ง 15 เซล แล้วสเตจแมทชีนจะส่งสัญญาณ Done แล้ววนกลับไปที่สเตจ S1 ใหม่เพื่อทำการแปลงข้อมูลรอบต่อไป โดยจะมีการทำงานของสเตจแมทชีนดังรูปที่ 4.13



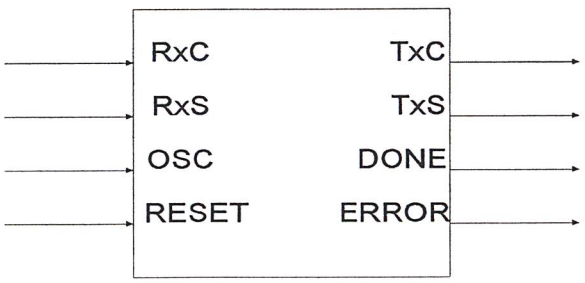
รูปที่ 4.13 สเตจแมทชีนควบคุมการทำงานของชิพ

4.6 ส่วนสัญญาณเข้าและสัญญาณออกของชิพ

จากโครงสร้างทางฮาร์ดแวร์ของวงจรแปลงสามารถนำมาประกอบกันเป็นชิพได้โดยในชิพ 1 ตัวจะสามารถทำได้ทั้งการแปลงและการแปลงกลับโดยการเปลี่ยนสัมประสิทธิ์การแปลงในสนามจำกัดขนาด 3 จุด และเปลี่ยนสัมประสิทธิ์การแปลงในสนามจำกัดขนาด 5 จุดซึ่งจะถูกกำหนดโดยค่าคงที่ (β) ที่ต่างกัน

ตัวชิพของวงจรแปลงและแปลงกลับดังรูปที่ 4.14 จะมีขาต่างๆ ดังนี้

- ขา OSC จะเป็นขาที่นำสัญญาณออสซิเลเตอร์จากภายนอกเข้าไปทำการสร้างสัญญาณ CLOCK ภายในชิพ
- ขา RESET จะเป็นขาที่นำสัญญาณรีเซ็ตจากภายนอกเข้าไปทำการรีเซ็ตอุปกรณ์และสเตจแมทชีนต่างๆ ภายในชิพ
- ขา RxC จะเป็นขาที่รับข้อมูลที่จะทำการแปลงจากพอร์ตอนุกรมเข้ามาทำการแปลงในตัวชิพ
- ขา RxS จะเป็นขาที่รับข้อมูลอนุกรมที่ทำการแปลงเสร็จสิ้นแล้วจากชิพอีกตัวหนึ่ง เข้ามาทำการแปลงกลับในชิพ
- ขา TxC จะเป็นขาที่นำข้อมูลที่ถูกทำการแปลงกลับเสร็จสิ้นแล้ว ส่งไปให้พอร์ตอนุกรม โดยในการส่งข้อมูลจะใช้การส่งแบบอนุกรม
- ขา TxS จะเป็นขาที่นำข้อมูลที่ถูกทำการแปลงเสร็จสิ้นแล้ว ส่งไปให้ชิพอีกตัวหนึ่งทำการแปลงกลับ โดยในการส่งข้อมูลจะใช้การส่งแบบอนุกรม
- ขา DONE จะเป็นขาที่นำสัญญาณ DONE จากสเตจแมทชีนควบคุมการทำงานของชิพ ส่งออกไปภายนอกเพื่อระบุว่าได้ทำการแปลงหรือแปลงกลับเสร็จสิ้นแล้ว 1 รอบ
- ขา ERROR จะเป็นขาที่นำสัญญาณ SERROR จากสเตจแมทชีนควบคุมการรับส่งของชิพ ส่งออกไปภายนอกเพื่อแจ้งรู้ว่าการรับข้อมูลผิดพลาด



รูปที่ 4.14 บล็อกไดอะแกรมของชิพ

บทที่ 5
ขั้นตอนการออกแบบและการทดสอบ

5.1 ขั้นตอนการออกแบบ

ขั้นตอนที่ใช้ในการออกแบบเริ่มจากการศึกษาวิธีการแปลงกลับและการแปลงในสนามจำกัด แล้วนำมาออกแบบเป็นโครงสร้างทางฮาร์ดแวร์เพื่อนำไปเขียนเป็นซอสโคด (Source Code) ภาษาวีเอชดีแอลแล้วนำไปตรวจสอบไวยากรณ์และจำลองการทำงานโดยโปรแกรมวี-ซีสเต็ม (V-System) เมื่อการทำงานถูกต้องแล้วจะนำไปสังเคราะห์เป็นวงจรระดับเกทโดยโปรแกรมกาลิเลโอ (Galilao) จากนั้นจึงนำแฟ้มข้อมูลที่ได้จากการสังเคราะห์มารวมกันเป็นแฟ้มเดียวเพื่อนำมาแยกส่วนและเชื่อมต่ออุปกรณ์ต่างๆ (Place & Route) ภายในวงจรรวมโดยใช้โปรแกรมไซลิงซ์ (Xilinx) แล้วจะทำการจำลองการทำงานของวงจรรวมโดยเพิ่มสัญญาณหน่วงเข้าไปในแฟ้มข้อมูลที่ทำการแยกส่วนและเชื่อมต่ออุปกรณ์ต่างๆเสร็จสิ้นแล้ว และนำไปแปลงเป็นภาษาวีเอชดีแอลอีกครั้งเพื่อใช้โปรแกรมวี-ซีสเต็มจำลองการทำงาน

5.1.1 ความคิดริเริ่มและวิเคราะห์ระบบที่จะออกแบบ

ในขั้นนี้เราจะต้องทำการกำหนดสิ่งต่างๆ ที่จะนำไปใช้ในขั้นต่อไปดังนี้

- กำหนดคุณสมบัติของระบบเชิงตัวเลข เช่นระบบนี้ให้ผลลัพธ์อะไรบ้าง เป็นต้น
- เขียนแผนผังการติดต่อของระบบย่อย (System Block Diagram) ที่ต้องใช้
- กำหนดสัญญาณที่ใช้ในการติดต่อ เช่น ชื่อ , ลักษณะที่ใช้ติดต่อ , จำนวน เป็นต้น
- เขียนอธิบายการทำงานอย่างคร่าวๆ

5.1.2 การออกแบบตัวบรรยายพฤติกรรม

จากขั้นตอนที่ผ่านมาเรานำเอาข้อกำหนดต่างๆ ที่ได้มาทำการออกแบบในขั้นนี้โดย

- เขียนประกาศเอ็นติตี้ (Entity Declaration) จากการกำหนดสัญญาณที่ใช้ในการติดต่อ
- เขียนผังการทำงาน (Flow Chart) โดยละเอียดในแต่ละส่วนของการทำงาน
- เขียนฐานเวลา (Timing Diagram) จากผังการทำงาน โดยดูความสัมพันธ์ระหว่างสัญญาณต่างๆ ว่าสัญญาณใดเมื่อเปลี่ยนแปลงจะมีผลทำให้สัญญาณอื่นๆ มีการเปลี่ยนแปลง และเปลี่ยนแปลงไปอย่างไร
- เขียนผังสถานะ (State Diagram) จากฐานเวลา

- เขียนตารางการโอนถ่ายสถานะ (Transition Table) จากผังสถานะ
- ทำการกำหนดการเปลี่ยนแปลงของสัญญาณที่เกิดขึ้น ณ.สถานะนั้นๆ
- เขียนประกาศสัญญาณที่ถูกใช้ภายในสถาปัตยกรรมของระบบ (Architecture Declaration)
- เขียนซอสโคด (Source Code) ภาษาวีเอชดีแอล โดยจะต้องเขียนตามหลักของ VHDL Synthesis Guide เพื่อที่จะสามารถนำมาสังเคราะห์เป็นวงจรระดับเกทได้
- นำซอสโคดที่ใด้ไปตรวจสอบไวยากรณ์ (Compile) โดยใช้โปรแกรมวี-ซีสเต็ม (V-System)
- นำซอสโคดไปจำลองการทำงาน (Simulation) ในขั้นแรกโดยใช้โปรแกรมวี-ซีสเต็ม (V-System)

5.1.3 การสังเคราะห์และการออกแบบทางกายภาพ

การสังเคราะห์จะเป็นการนำซอสโคดภาษาวีเอชดีแอลที่เขียนขึ้นมาทำการสังเคราะห์ (Synthesis) ให้เป็นวงจรระดับเกทโดยใช้โปรแกรมกาลิเลโอ (Galilao) โดยในโครงงานนี้จะเลือกใช้อุปกรณ์ (Component) ที่นิยามในรูปแบบฟังก์ชัน (Library) จากเทคโนโลยี XC4000 ของ XILINX เมื่อทำการสังเคราะห์แล้วจะได้แฟ้มข้อมูลที่บรรยายอุปกรณ์และการเชื่อมต่อระหว่างอุปกรณ์ (XNF file) และจำนวนของส่วนคอมบิเนชัน (Combination) และส่วนซีเควนเชียล (Sequential) ที่ใช้ไปสามารถดูได้จากแฟ้มข้อมูล (LOG file) เมื่อทำการสังเคราะห์ซอสโคดเสร็จสิ้นหมดแล้ว ก็จะนำ XNF file ทั้งหมดมารวมกันเป็นแฟ้มข้อมูลเดียว (XFF file) และทำการเปลี่ยนเป็นแฟ้มข้อมูลที่บรรยายทางกายภาพ (LCA file) โดยการวางและเชื่อมต่ออุปกรณ์ต่างๆ ที่จะนำมาสร้างเป็นชิพ ด้วยโปรแกรมไซลิงซ์ (Xilinx)

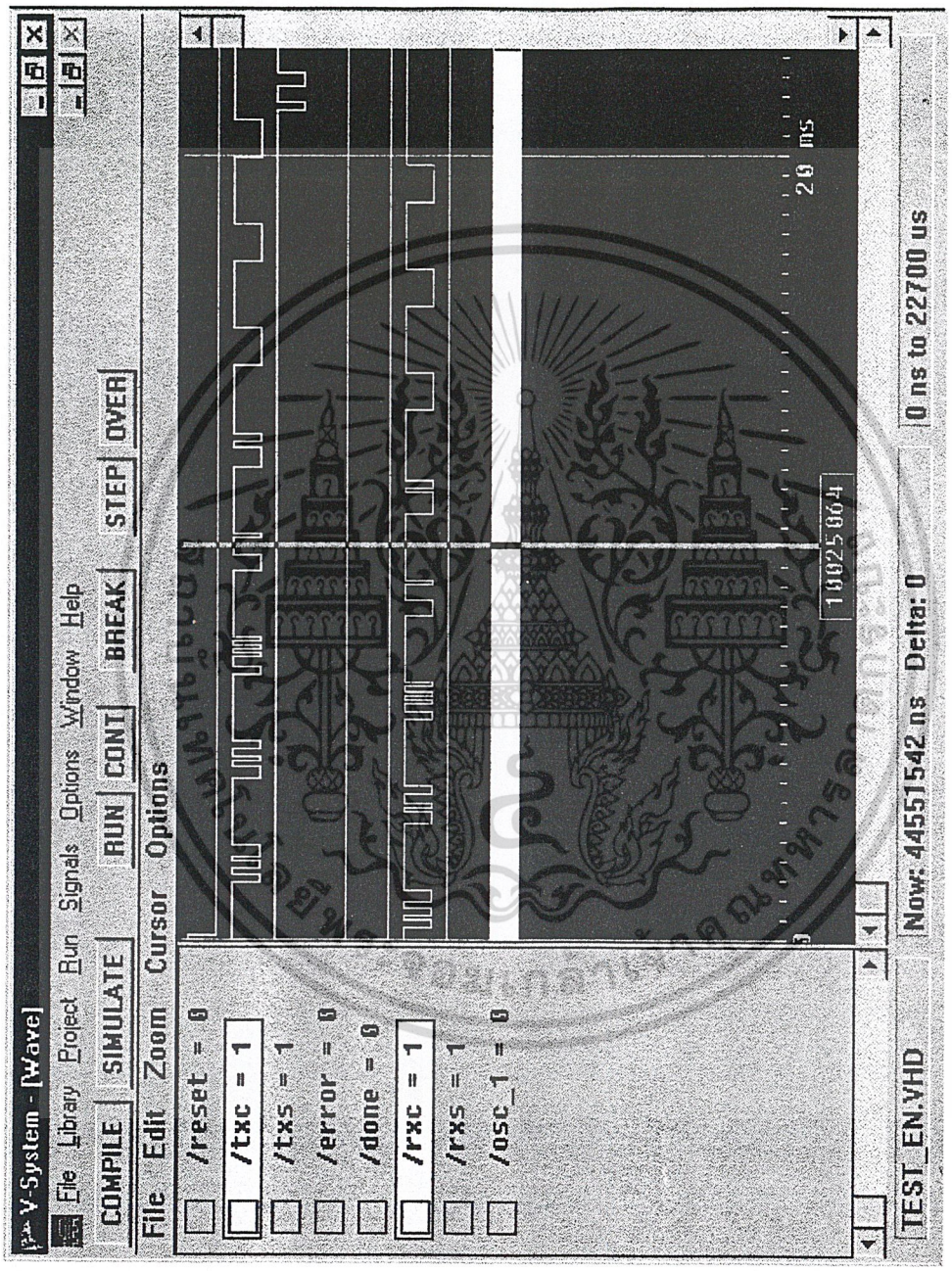
5.2 การทดสอบและผลการทดสอบ

การทดสอบจะสามารถทำโดยการจำลองการทำงาน (Simulation) ของชิพ โดยทำการเพิ่มสัญญาณหน่วงเข้าไปที่ LCA file จากนั้นจะทำการเปลี่ยน LCA file เป็น XNF file โดยคำสั่ง XDelay และ LCA2XNF ตามลำดับ ด้วยโปรแกรมไซลิงซ์ แล้วจะทำการเปลี่ยน XNF file กับไปเป็นแฟ้มข้อมูลที่บรรยายโดยภาษาวีเอชดีแอลและแฟ้มข้อมูลของสัญญาณหน่วง (SDF file) โดยการทำ Back Annotation ด้วยโปรแกรมกาลิเลโอ เพื่อนำมาทำการจำลองการทำงานโดยโปรแกรมวี-ซีสเต็มต่อไป

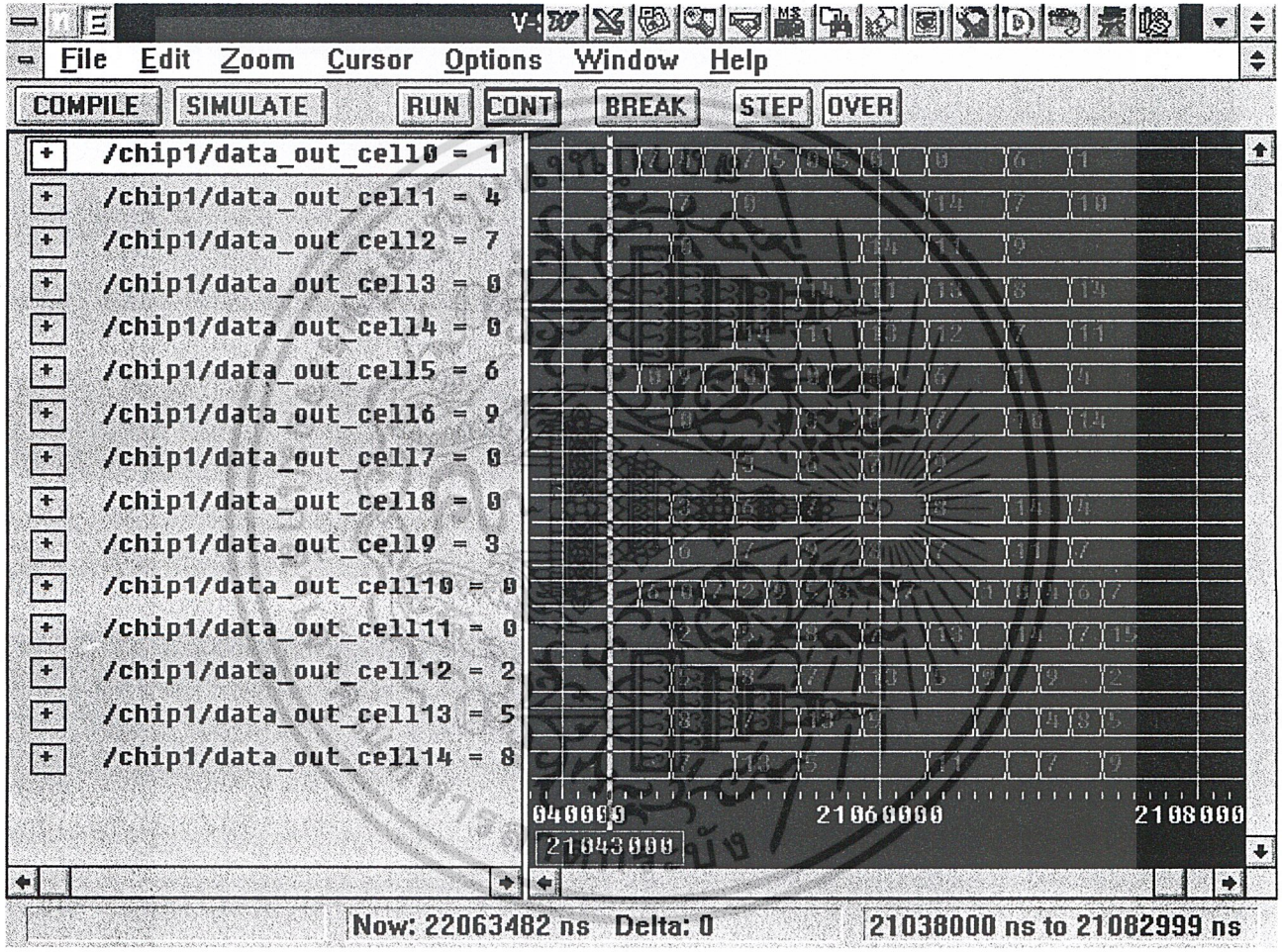
ในการจำลองการทำงานของวงจรที่มีสัญญาณหน่วงจะใช้โปรแกรมวี-ซีสเต็มที่มี Library ของ XC4000 จำลองการทำงานโดยเขียนภาษาวีเอชดีแอลป้อนข้อมูลอินพุท (Test Bench) ทั้งส่วนการเข้ารหัสและถอดรหัสแล้วดูข้อมูลเอาท์พุทที่ผ่านการเข้ารหัสหรือถอดรหัสแล้วว่าถูกต้องหรือไม่ โดยเปรียบเทียบกับข้อมูลที่ผ่านการเข้ารหัสหรือถอดรหัสแล้วที่ได้จากการคำนวณโดยโปรแกรมภาษาปาสคาล (Pascal) ผลการทดลองจะได้ตามตารางที่ 5.1

ตารางที่ 5.1 ผลการทดลองที่ได้โปรแกรมวี-ซีสเต็มและโปรแกรมภาษาปาสคาล

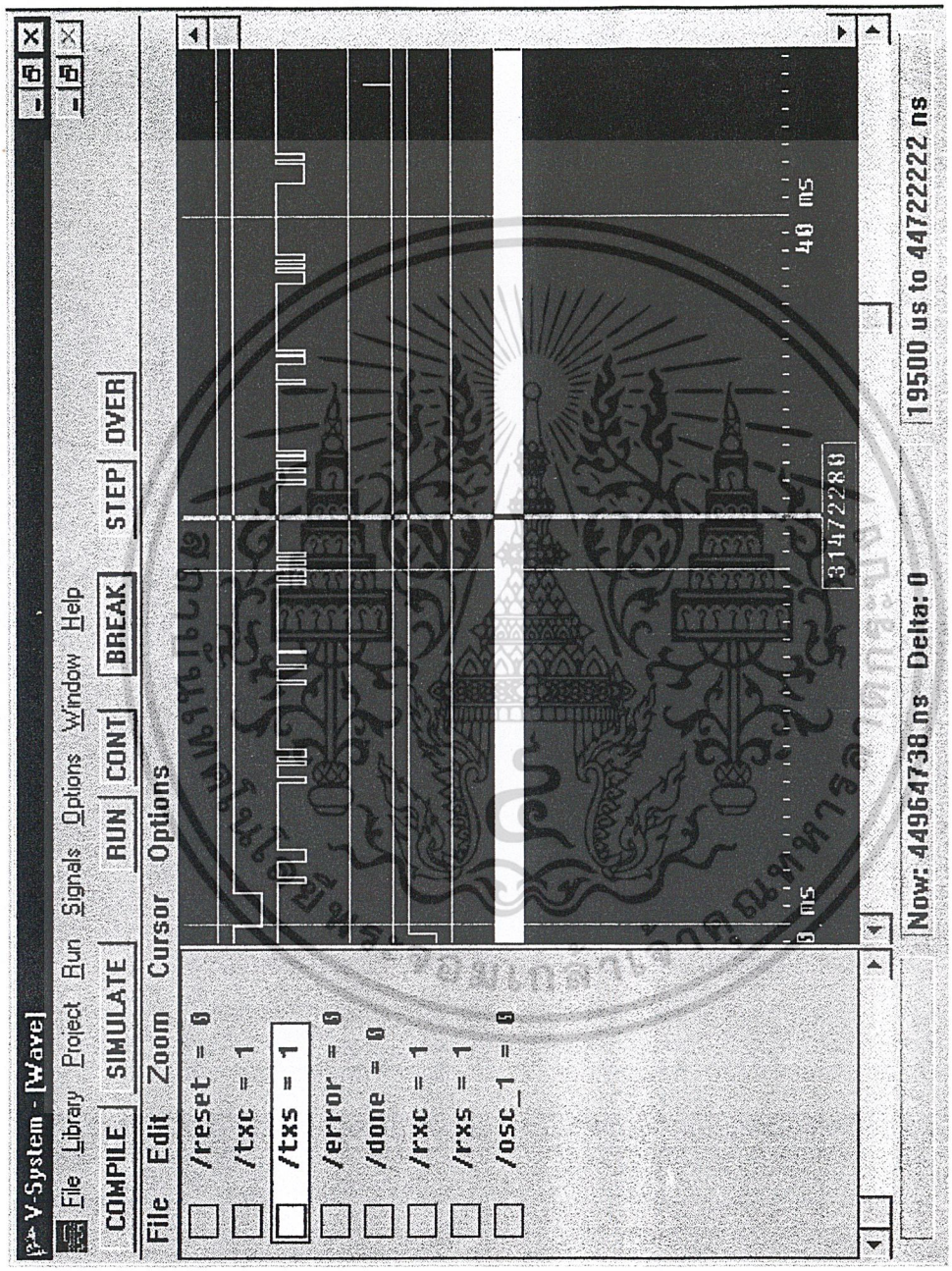
เซลที่	ข้อมูลการเข้ารหัส				ข้อมูลการถอดรหัส			
	วี-ซีสเต็ม		ปาสคาล		วี-ซีสเต็ม		ปาสคาล	
	อินพุท	เอาท์พุท	อินพุท	เอาท์พุท	อินพุท	เอาท์พุท	อินพุท	เอาท์พุท
0	1	1	1	1	1	1	1	1
1	2	14	2	14	14	2	14	2
2	3	2	3	2	2	3	2	3
3	4	14	4	14	14	4	14	4
4	5	7	5	7	7	5	7	5
5	6	7	6	7	7	6	7	6
6	7	10	7	10	10	7	10	7
7	8	9	8	9	9	8	9	8
8	9	5	9	5	5	9	5	9
9	0	11	0	11	11	0	11	0
10	0	4	0	4	4	0	4	0
11	0	15	0	15	15	0	15	0
12	0	9	0	9	9	0	9	0
13	0	4	0	4	4	0	4	0
14	0	9	0	9	9	0	9	0



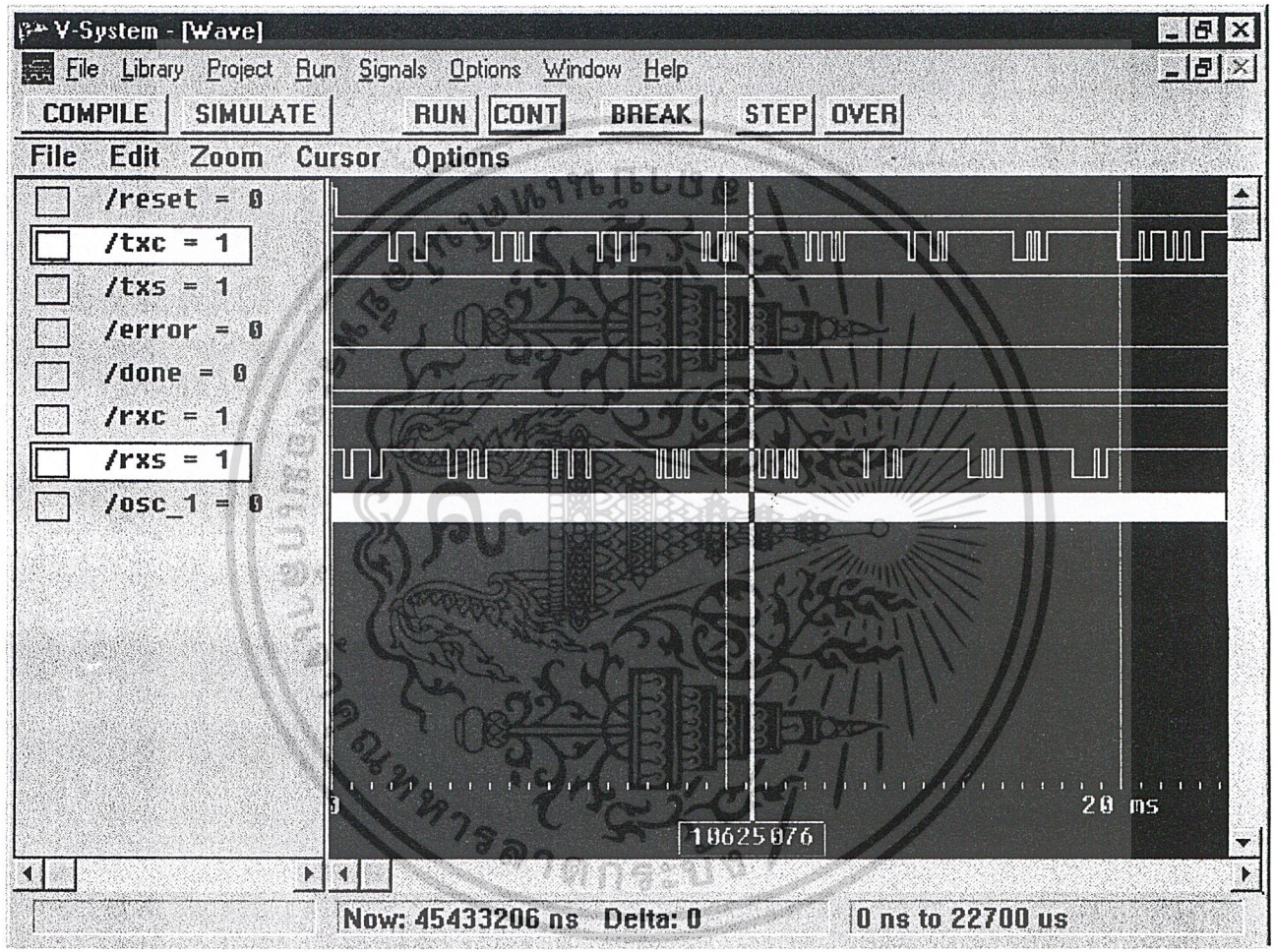
รูปที่ 5.1 การจำลองการทำงานของอินพุทส่วนเบรทัส



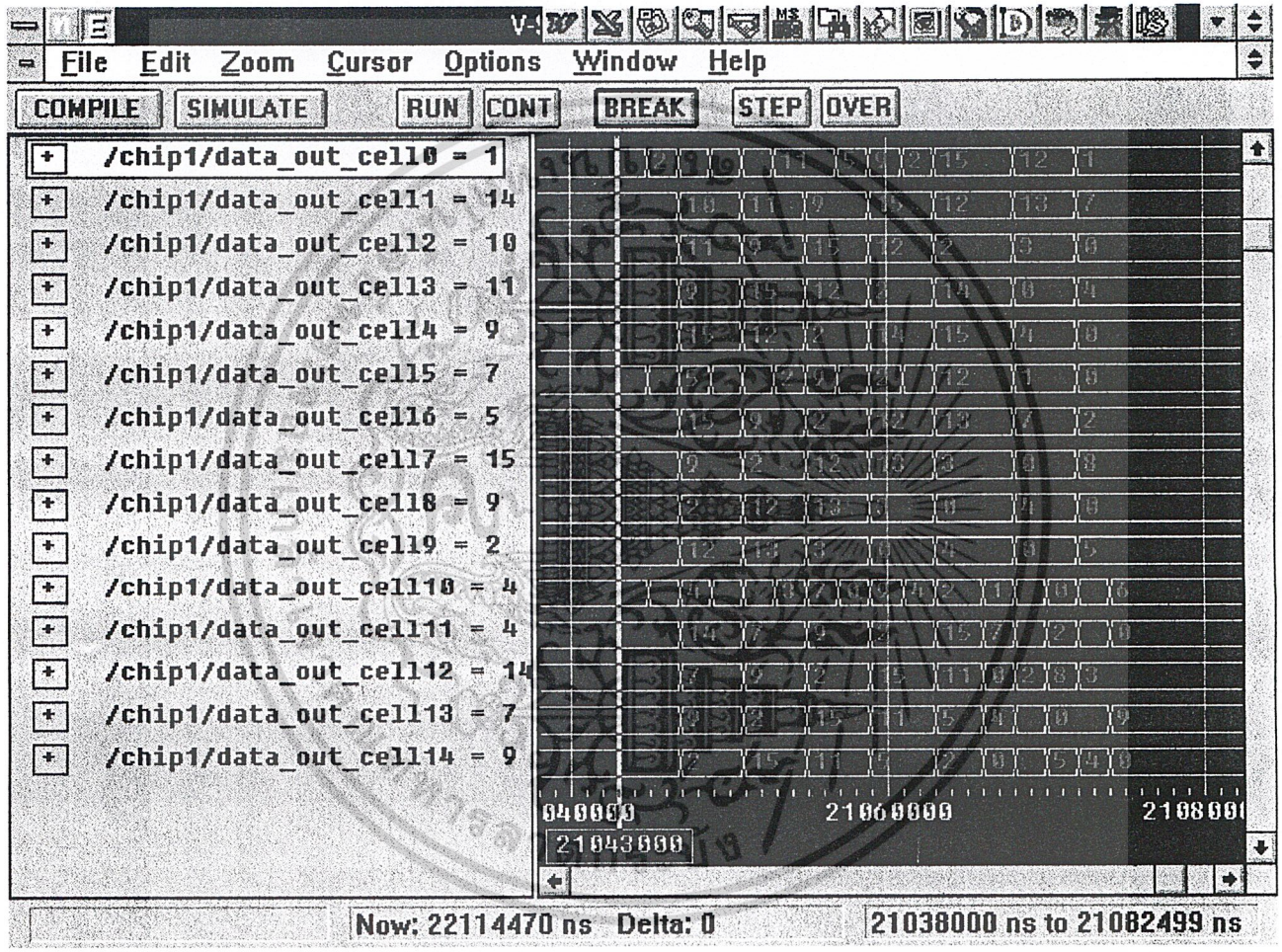
รูปที่ 5.2 การจำลองการทำงานของเอาท์พุทที่รีจีสเตอร์เซลส่วนเข้ารหัส



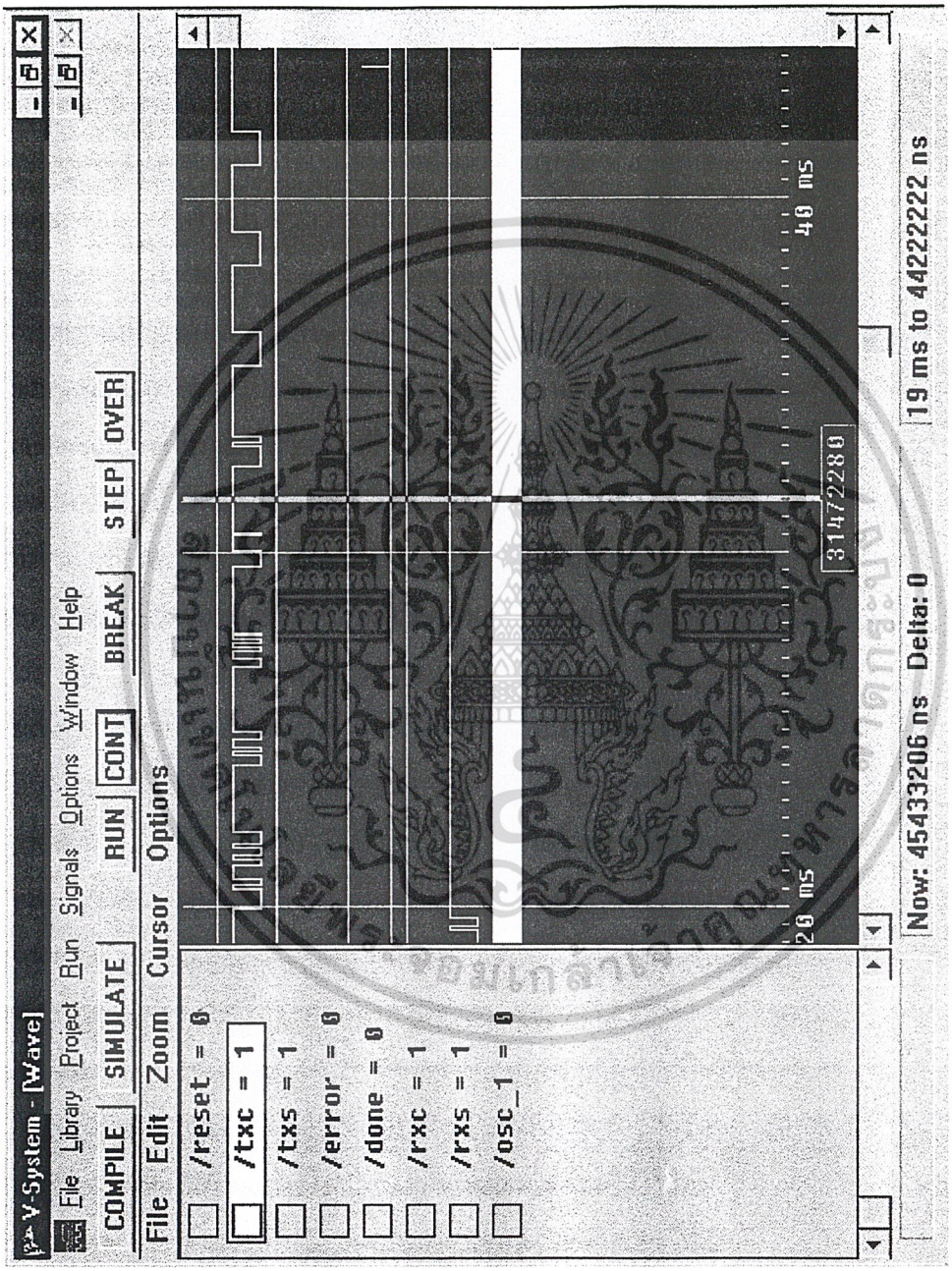
รูปที่ 5.3 การจำลองการทำงานของเอาพุทส่วนเยารหัส



รูปที่ 5.4 การจำลองการทำงานของอินพุทส่วนถอดรหัส



รูปที่ 5.5 การจำลองการทำงานของเอาท์พุทที่รีจีสเตอร์เซลส่วนถอดรหัส



รูปที่ 5.6 การจำลองการทำงานของเอาท์พุทส่วนถอดรหัส

บทที่ 6
สรุปและวิจารณ์

1. การแปลงขนาด 15 จุดใน สนามจำกัด GF(2^4) จะเห็นได้ว่า มีการบวก 255 ครั้งและมีการคูณ 255 ครั้งเช่นกัน จึงทำให้ต้องใช้เวลาในการคำนวณมาก วิธีการตัวประกอบปฐมจะเป็นการแยกข้อมูลจำนวน 15 จุด ออกเป็นส่วนๆ โดยแต่ละส่วนจะมีขนาดเป็นจำนวนปฐม ซึ่งเมื่อคูณกันทั้งหมดแล้วจะมีค่าเท่ากับ 15 ซึ่งแยกข้อมูลออกได้เป็นขนาด 3 และ 5 จุด ข้อมูลทั้ง 15 จุด จะนำมาผ่านการแปลง 3 และ 5 จุด โดยการแปลงแต่ละขั้นจะทำจนทั่วข้อมูลทั้ง 15 จุด และในระหว่างการแปลงแต่ละขั้นจะมีการสลับตำแหน่งข้อมูล (Mapping) ซึ่งเป็นไปตามวิธีการตัวประกอบปฐม ในการสลับข้อมูลนี้จะไม่มีการคำนวณเกิดขึ้นเลยเมื่อออกแบบเป็นวงจรรวม เนื่องจากใช้การเชื่อมต่อแบบคงที่ (hardwire) ดังนั้นจะเห็นได้ว่า เมื่อใช้วิธีการตัวประกอบปฐมจะใช้การบวกและการคูณเพียง 120 ครั้ง ซึ่งลดลงจากเดิมประมาณ 2 เท่า และยังจะเห็นได้ว่าการทำงานแบ่งออกเป็น 2 ส่วน จึงทำให้โครงสร้างการคำนวณที่ได้จากวิธีการตัวประกอบปฐมจะมีลักษณะเป็นแบบท่อส่ง (Pipeline) และเมื่อนำวิธีการแปลงข้อมูลจำนวนน้อยๆ มาใช้ร่วมด้วยจะช่วยลดจำนวนการคูณและการบวกลงเหลือ 20 ครั้งและ 76 ครั้งตามลำดับ

2. รูปแบบหลักในการบรรยายการทำงานของระบบซึ่ง VHDL ได้จัดเตรียมไว้แบ่งเป็น 3 ระดับคือ ระดับพฤติกรรม ระดับข้อมูลและระดับโครงสร้าง

การบรรยายในระดับพฤติกรรม (Behavioral Level) ช่วยอำนวยประโยชน์ในการออกแบบเป็นขั้นตอนเช่น การออกแบบพอร์ตติดต่อ (I/O Port) ที่จำเป็นใช้อธิบายการทำงานของวงจรในระดับสูง สิ่งที่สนใจเป็นเพียงสัญญาณที่อินพุทและเอาท์พุทเท่านั้น การออกแบบในระดับนี้จะช่วยตรวจสอบและแก้ไขโครงสร้างของวงจรทั้งในเรื่องของพอร์ต จำนวนอุปกรณ์ที่จำเป็นสำหรับการนำมาเชื่อมต่อให้เป็นวงจรที่สมบูรณ์ การประสานงานของแต่ละอุปกรณ์เมื่อรวมกันเป็นระบบ

การบรรยายในระดับกระแสข้อมูล (Dataflow Level) เป็นระดับกลางที่อยู่ระหว่างระดับพฤติกรรมและระดับโครงสร้าง การออกแบบในระดับนี้จะใช้แทนการทำงานของอุปกรณ์ได้ดี ประโยชน์ที่ได้คือ อธิบายการเลื่อนไหลของสัญญาณข้อมูลต่างๆระหว่างลอจิกและรีจิสเตอร์โดยผ่านตัวกลางคือบัสและช่วยในการตรวจสอบและแก้ไขการประสานงานระหว่างลอจิกและรีจิสเตอร์ในลักษณะของการปฏิบัติงานพร้อมกัน (concurrent) ซึ่งเป็นการเลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์

การบรรยายในระดับโครงสร้าง (Structure Level) เป็นระดับล่างสุดเมื่อการออกแบบในระดับบนทั้งสองถูกต้องแล้ว การออกแบบในระดับนี้เป็นการออกแบบในระดับเกทใช้อธิบายการทำงานที่ประสานกันของเกทต่างๆ เพื่อให้ได้สัญญาณออกมาตามต้องการ เมื่อถึงขั้นนี้แล้วการตรวจสอบและแก้ไขจะเป็นเพียงการดูแลความถูกต้องในเชิงตรรกเท่านั้น

3. ปัญหาและอุปสรรคในการทำงานที่สำคัญที่สุดคือ การขาดพื้นฐานความรู้และไม่มีประสบการณ์การทำงานการใช้ภาษา VHDL วิธีการและอัลกอริธึมต่างๆ ที่นำมาช่วยให้การออกแบบวงจรรวมให้มีความสามารถ, ประสิทธิภาพและความเร็วในการทำงานสูงขึ้น ทำให้ต้องเสียเวลาส่วนหนึ่งไปกับการศึกษา ดังนั้นแนวทางแก้ไขที่ดีที่สุดคือ การเร่งศึกษาให้เข้าใจโดยใช้เวลาที่น้อยที่สุดซึ่งจะทำให้มีเวลาในการออกแบบโปรแกรมมากขึ้น

หนังสืออ้างอิง

1. Richard E.Blahut , " Theory and Practice of Error Control Codes " , Addison-Wesley Publishing Company, 1984
2. สมศักดิ์ ชุมช่วย , " Fast Transform Techniques for RS Codes " , วิศวสารลาดกระบัง, ปีที่ 12, ฉบับที่ 1/มิถุนายน, หน้า 32-41, 2538
3. สมศักดิ์ ชุมช่วย และพงศธร หมายดี , " วิธีการตัวประกอบปฐมเพื่อเพิ่มความเร็วของการแปลงในสนามจำกัด " , วิศวสารลาดกระบัง, ปีที่ 13, ฉบับที่ 1/กรกฎาคม, หน้า 62-71 , 2538
4. สมศักดิ์ ชุมช่วย , " On the Implementation of Finite Field Operations " ,วิศวสารลาดกระบัง ปีที่ 11, ฉบับที่ 1/มิถุนายน, หน้า 7-17, 2537
5. Jayaram Bhasker , " A VHDL Priner " ,Prentice Hall, 1992
6. บวร ปภัสราทร , ประเสริฐ คันธมานนท์ และสุเมธ อังคะศิริกุล, " เทคโนโลยีการออกแบบวงจรรวม " , ศูนย์เทคโนโลยีอิเล็คทรอนิคส์และคอมพิวเตอร์แห่งชาติ, กระทรวงวิทยาศาสตร์เทคโนโลยีและการพลังงาน, 79 หน้า, 2533

ภาคผนวก

1. สัญลักษณ์ที่ใช้ในปริญญานิพนธ์

$\tau_{INV(X)}$ = propagation delay of inverter gate (fan out = x)

$\tau_{AND(X)}$ = propagation delay of and gate (fan out = x)

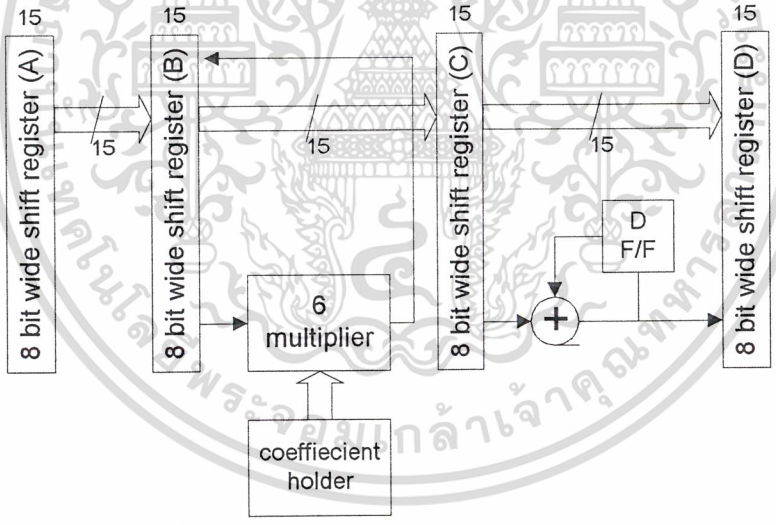
$\tau_{XOR(X)}$ = propagation delay of exclusive or gate (fan out = x)

$\langle c \rangle_p$ หมายถึงเศษเหลือ (residue) ของ C หารเต็มหน่วยด้วย P

(s,t) หมายถึง ตัวหารร่วมมากของ s และ t

$\equiv$ แสดงถึงการคอนครูเร็นซ์ (congruences)

2. วงจรแปลงขนาด 15 จุดในสนามจำกัด $GF(2^4)$ ซึ่งไม่ใช้วิธีตัวประกอบปฐม



รูปแสดงวงจรแปลงขนาด 15 จุดใน $GF(2^4)$ ที่ไม่ใช้วิธีตัวประกอบปฐม

วงจรแปลงดังรูป จะทำการแปลงโดยจะได้ข้อมูลที่ผ่านการแปลง 1 ตัวต่อรอบการคำนวณ รอบการคำนวณเริ่มจากข้อมูลจากบัฟเฟอร์ A ถูกถ่ายเทเข้าบัฟเฟอร์ B ซึ่งจะทำหน้าที่ป้อนข้อมูลให้กับชุดวงจรคูณทำการคูณข้อมูลทั้ง 15 ตัวด้วยตัวคูณที่ป้อนให้โดยอุปกรณ์เก็บตัวคูณ เมื่อทำการคูณเสร็จแล้วข้อมูลทั้ง 15 ตัวจะถูกถ่ายเทไปยังบัฟเฟอร์ C ซึ่งจะทำการคำนวณในขั้นตอนการบวกเพื่อให้ได้ผลลัพท์ 1 ตัวจากการแปลงซึ่งจะนำในเก็บไว้ที่ บัฟเฟอร์ D และจะมีการเลื่อนขึ้น

เพื่อรอรับผลจากการคำนวณในรอบต่อไป จะเห็นว่าการคำนวณการแปลงจะทำเช่นนี้ไปจนครบ 15 รอบจึงจะได้ผลลัพท์จากการแปลงทั้ง 15 ตัว การแปลงจากโครงสร้างนี้จะมีคาบเวลาสัญญาณนาฬิกาเป็น $\tau_{INV(2)}$ และจะใช้เวลาในการแปลงเท่ากับ $15 \times 15\tau_{INV(2)} = 515.25$ nS

3. ตารางแสดงค่า GF (2^4)

Exponential Notation	Polynomial Notation	Binary Notation	Decimal Notation
0	0	0000	0
α^0	1	0001	1
α^1	Z	0010	2
α^2	Z^2	0100	4
α^3	Z^3	1000	8
α^4	Z + 1	0011	3
α^5	$Z^2 + Z$	0110	6
α^6	$Z^3 + Z^2$	1100	12
α^7	$Z^3 + Z + 1$	1011	11
α^8	$Z^2 + 1$	0101	5
α^9	$Z^3 + Z$	1010	10
α^{10}	$Z^2 + Z + 1$	0111	7
α^{11}	$Z^3 + Z^2 + Z$	1110	14
α^{12}	$Z^3 + Z^2 + Z + 1$	1111	15
α^{13}	$Z^3 + Z^2 + 1$	1101	13
α^{14}	$Z^3 + 1$	1001	9

4. Field Programmable Gate Array (FPGA)

รู้จักกับ Programmable Gate Array

ความก้าวหน้าของอุตสาหกรรมอิเลคทรอนิคส์ในปัจจุบัน ทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆมากมาย ซึ่งทำให้เกิดการลดราคาค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาด ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้น เห็นได้ชัดจากเทคโนโลยีไมโครโปรเซสเซอร์และหน่วยความจำ ทุกๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างระหว่าง วงจรรวม VLSI และ IC standard มากขึ้น ในการพัฒนาเพิ่มความหนาแน่นและจำนวน Logic Function ที่เหมาะสม นักออกแบบอุปกรณ์ทางด้านดิจิตอล ได้พิจารณาให้ได้ขนาดมากๆ และการผลิตวงจรรวม ASIC

(Application Specific Integrated Circuit)

Field Programmable Gate Arrays (FPGA) ฯลฯ ASIC chip ที่มีปริมาณความหนาแน่นของ gate สูง สามารถจะกำหนดฟังก์ชันการทำงานได้ตามความต้องการของผู้ใช้ (user) FPGA ได้รวบรวมข้อดีทั้งหมดของการทำ CUSTOM VLSI มารวมไว้ทั้งหมด ได้แก่ การออกปบบการผลิต, ลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรรวมเป็นอย่างมาก นักออกแบบเพียงแต่กำหนดฟังก์ชันการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้ FPGA สามารถออกแบบและทดสอบภายในเวลาเพียง 2-3 วันเท่านั้น ตรงข้ามกับการออกแบบโดยใช้ CUSTOM gate array ซึ่งใช้เวลาหลายอาทิตย์ การเปลี่ยนแปลงแก้ไขแบบ ก็เช่นเดียวกัน จากผลประโยชน์ของ FPGA ซึ่งที่ได้กล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกแบบผลิตภัณฑ์ ลดค่า NRE (Nonrecurring engineering cost) ลงไปด้วย

Field Programmable Gate Array (FPGA)

คล้ายๆกับ Gate array ทั่วไป FPGA ไม่ต้องมีค่าใช้จ่ายคงที่ และ ไม่มีค่า Fabrication เพราะว่า FPGA ทุกค่ามีโครงสรางที่เหมือนกัน ค่าใช้จ่ายในการผลิตมีลักษณะคล้ายๆกับการผลิต IC standard ที่มีปริมาณมากๆ โดยทั่วไป

Programmable Logic Device (PLD)

มีลักษณะ PLDs โดยส่วนใหญ่จะใช้แทนที่ SSt/MSI device ได้ประมาณ 5-10 ตัวในวงจร จะเหมาะที่สุดเมื่อใช้ทำ ASIC ที่มีเกทจำนวน 200-300 เกท ภายใน PLD จะประกอบด้วยวงจรของ AND-OR Gate เป็นพื้นฐานข้อจำกัดของตัว PLD ก็คือจำนวนฟลิปฟลอป (flip-flop) , จำนวนของ อินพุท เอาท์พุท และการเชื่อมต่อภายใน

Standard Cell & Custom IC

การผลิตแบบนี้ต้องการหน้ากาก Mask สำหรับทุก Layer ในการผลิตทำให้ต้องมีค่าใช้จ่ายพิเศษ และมีความล่าช้าในการพัฒนา แต่ผลดีคือให้ต้นทุนต่อหน่วยต่ำที่สุดเหมาะสำหรับการผลิตให้จำนวนมากๆ standard cell ให้ประโยชน์ในการ high level building block

Gate Array

เราใช้ gate array สร้างวงจรลอจิกโดยการเชื่อมต่อทรานซิสเตอร์และ gate ต่างๆ เข้าด้วยกันจนได้ฟังก์ชั่นการทำงานตามที่ต้องการซึ่งเป็นขั้นตอนสุดท้ายของกระบวนการผลิต gate array มีความหนาแน่นของ gate ประมาณ 100,000 เกท หรือมากกว่านั้น ใช้ประโยชน์ประมาณ 80-90 % สำหรับอุปกรณ์เล็กๆ และ 40-60 % สำหรับวงจรใหญ่ ลักษณะไม่เหมือน IC Standard products ,gate-array มีค่าใช้จ่าย fixed cost และ product cost การใช้งาน gate array จะประหยัดที่สุดเมื่อจำนวนการผลิตสูงมากจนทำให้ค่าใช้จ่ายคงที่หายไป

Programmable Gate Array Architecture

Logic Cell Array (LCA) เป็นสิ่งประดิษฐ์ของบริษัท XILINX มีสถาปัตยกรรมภายในคล้ายๆ gate array โดยทั่วไป ภายในมีลักษณะเป็นเมตริกซ์ของ logic block และล้อมรอบไปด้วย I/O Interface block การเชื่อมต่อระหว่าง CLB และ IOB ทำได้โดยผ่าน Channel ที่ว่างพาดผ่านระหว่าง row และcolumn มีการทำงานเหมือนกับMicroprocessor ตัว LCA จะทำงานได้ดีต้องใช้ Program-driven logic device หน้าที่ของ CLB ,IOB แต่ละตัว,การเชื่อมต่อภายใน (interconnection) ถูกกำหนดไว้ใน Configuration program หรือเก็บไว้ใน EPROM ภายใน LCA โปรแกรมจะถูก load เข้าสู่ LCA เมื่อมีการจ่ายไฟ (power-up) ,โดยทางคำสั่ง (command) ซึ่งเป็นส่วนหนึ่งของการทำ Systam initialization

ประสิทธิภาพของ LCA กำหนดโดยความเป็น Logic ส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ ความเร็วของ System clock rate ถูกกำหนดด้วย toggle flip-flop สำหรับการประยุกต์ใช้โดยทั่วไปจะอยู่ที่ประมาณ 1/3 ถึง 1/2 ของ maximum toggle gate

1. Configuration logic block (CLB)

หัวใจหลักภายใน LCA คือเมตริกซ์ของ CLB หลายๆ ตัว CLB แต่ละตัวประกอบด้วย Programmable combination logic และ Storage register ส่วนในวงจร Combinatorial logic section สามารถใช้สร้างวงจรทางด้านBoolean Function ของ Input ส่วน Register รับค่าจากส่วน Combination หรือโดยตรงจาก CLB Output สามารถขับวงจร Combination Logic โดยตรงผ่านทาง Feedback Path

2. Input/output block (IOB)

เป็นส่วนติดต่อกับวงจรภายนอกของ LCA ได้สร้างมาจาก Programmable input/output device (IOBs) IOB แต่ละตัวสามารถโปรแกรมได้อย่างอิสระโดยจะให้เป็น input/output แบบ 3 state หรือ I/O แบบสองทิศทางก็ได้โดย Input สามารถโปรแกรมให้อยู่ได้ทั้งระดับสัญญาณ TTL และ CMOS threshold ของ IOB แต่ละตัวมี flip-flop สามารถใช้เป็น Buffer สำหรับ input หรือ output

3. Interconnect

ความยืดหยุ่นของการใช้ LCA มาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม resource ต่างๆ ที่อยู่ภายในเข้าด้วยกัน การที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายใน chip เหมือนกับ gate array ทั่วๆ ไป การเชื่อมต่อภายใน LCA ประกอบด้วย network 2 ทิศทางคือทาง vertical และ horizontal หรือ row และ column ซึ่งวางอยู่ระหว่าง CLB programmable switch จะทำการเชื่อมต่อ input และ output ของ IOBs และ CLBs ที่จุดต่อร่วมระหว่าง row และ column สามารถ switch signal จาก path ไปยังส่วนอื่นๆ เส้น long line จะลากผ่านตลอด chip เพื่อแก้ปัญหาเรื่อง critical signal ด้วย minimum relay หรือ skew ที่เกิดขึ้น

Logic cell array family

1. XC3000

1.1 คุณลักษณะ (Features)

1.1.1 ประสิทธิภาพสูงใช้งานได้ที่ความถี่ 70 ,100-125 Mhz ขึ้นไป

1.1.2 เป็น Generation ของ FPGA

* I/O function
* Digital logic function
* Interconnection

1.1.3 มีสถาปัตยกรรมภายในที่ยืดหยุ่น

* มีจำนวน Gate ภายในจำนวน 2,000 ถึง 3,000 เกท
* เพิ่มความสามารถพิเศษของ I/O และ Register
* มีค่า fan-out สูง
* มี 3 state bus ภายใน
* ทำงานกับสัญญาณ TTL และ CMOS
* มี Oscillator amplifier ในตัว

1.1.4 เป็นผลิตภัณฑ์มาตรฐาน

* ใช้พลังงานต่ำ ,เป็น CMOS ,ใช้เทคโนโลยี Static Memory

* ประสิทธิภาพเทียบเท่าการใช้ TTL SSI/MIS

* ผ่านการทดสอบจากโรงงานแล้ว 100 %

* สามารถเลือก Configuration Mode ได้

* มี Development Software ช่วยในการพัฒนา (XACT development software) สมบูรณ์แบบ

* สามารถ capture ผังวงจรได้

* มี Automatic place/route

* ทำการจำลองวงจรและหาฐานเวลาได้

* มีตัวที่ใช้แก้ไขในการออกแบบได้

* มี libraly และ user macros

* ทำการคำนวณหาค่าของฐานเวลาได้

* มี XACT In-Circuit Verifier

* มีมาตรฐานของแฟ้มข้อมูลของตัว PROM

1.2 รายละเอียด (Description)

LCA ตระกูล XC3000 Logic cell array (LCA) family ประกอบไปด้วย

วงจรลอจิกที่มีความหนาแน่นและประสิทธิภาพสูง ความยืดหยุ่นและการโปรแกรมได้ user array architecture เกิดขึ้นมาจาก configuration program ที่อยู่ภายในและส่วนประกอบที่สามารถปรับเปลี่ยนเพื่อให้ได้วงจรตามที่ต้องการได้ 3 แบบ คือ IOB ,array of CLB ,เชื่อมต่อภายในตามความต้องการของผู้ใช้ logic function และการเชื่อมต่อภายในถูกกำหนดด้วยโปรแกรมที่อยู่ภายในหน่วยความจำ ภายในตัว LCA โปรแกรมนี้ load เข้าสู่ Memory ได้ในหลายๆ รูปแบบตามความเหมาะสมที่ใช้งาน โปรแกรมถูกบรรจุอยู่ใน EPROM หรือ ROM ภายนอกหรืออยู่บน floppy-disk , harddisk ก็ได้ ภายใน chip มีส่วนพิเศษที่ช่วยในการ load โปรแกรมแบบอัตโนมัติ เมื่อจ่ายไฟเข้าระบบ (power-up) LCA ตระกูล XC3000 นี้มีหลายแบบให้เลือกตามความเหมาะสมตามขนาด, อุณหภูมิ,รูปแบบของตัวถัง,อุณหภูมิใช้งาน เป็นต้น

2. XC4000

2.1 คุณลักษณะ

2.1.1 เป็น Generation ที่ 3 ของ FPGA

* มี flip-flop เป็นจำนวนมาก

* ในการผลิตฟังก์ชั่นของการทำงานมีความยืดหยุ่นสูง
* มี Ultra-fast RAM ในตัว
* ใช้กับงานที่ต้องการความเร็วสูง
* มี wide edge decoder
* Interconnect line เป็นแบบ Hierachy
* สามารถใช้ 3-state bus ภายในได้
* มีการกระจายพลังงานของสัญญาณต่ำ

2.1.2 มีสถาปัตยกรรมภายในที่ยืดหยุ่น

* มี logic blocks และ I/O blocks ที่โปรแกรมได้
* มี Interconnect และ wide decoder ที่โปรแกรมได้

2.2 รายละเอียด

LCA ตระกูล XC4000 ประกอบด้วยวงจรลอจิกที่มีความหนาแน่นสูง ,มีความยืดหยุ่น ,การโปรแกรมสถาปัตยกรรมของ CLBs ที่ต่อกันภายในมีประสิทธิภาพมากในวิธีการแบบ Hierachy

โครงสร้างภายใน FPGA

วงจรรวมชนิดนี้ผลิตโดยบริษัท Xilinx ซึ่งเป็นบริษัทที่ร่วมกันทำการค้นคว้ากับบริษัท MMI สร้างเป็นอะเรย์ที่ประกอบด้วยเกทจำนวน 600-25,000 เกท ดังแสดงในตารางที่ 1 การที่ต้องขนาดของวงจรรวมเป็นจำนวนเกท เพราะจะได้รู้ว่าขนาดของวงจรรวมที่ออกแบบไว้สามารถโปรแกรมลงบนวงจรรวม FPGA ได้หรือไม่ เช่น วงจรรวมสำหรับการเข้ารหัสถอดรหัสแบบรีดโซโลมอนนี้ มีจำนวนเกทประมาณ เกทเพราะฉะนั้นใช้ไอซี FPGA ตระกูล XC4000 เบอร์ 4010 ส่วนการประมาณจำนวนเกทเมื่อเทียบกับเกทพื้นฐานเป็นดังตารางที่ 2

FPGA มีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกทอะเรย์ (Gate array) มาก สามารถโปรแกรมและลด configuration static ram ภายในโดยใช้กระแสไฟฟ้า ซึ่งจะทำการโปรแกรมได้โดยดึง

ข้อมูลฐานสิบหกมาจากภายนอก เช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD ,PAL ที่มี RPROM อยู่ภายในตัว ภายใน FPGA จัดเรียงเป็น ลอจิกเซลล์ ล้อมรอบภายนอกด้วยอินพุทเอาท์พุทเซลล์ FPGA ตัวแรกที่ผลิตโดย Xilinx คือ XC2064 (2000 Family) ประกอบด้วยเซลล์

เรียงเป็นเมตริกซ์มีจำนวนเซลล์ 64 เซลล์ หลังจากนั้นผลิตตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างภายใน CLB ซับซ้อนขึ้น สามารถจุจำนวนเกทได้สูงและดีขึ้น แต่ละเซลล์เรียกว่า CLB (Configurable Logic Block)

2.1.3 ทำกระบวนการ Sub-micron ชนิด CMOS ได้

* มี logic และ interconnect ที่มีความเร็วสูง
* ใช้กำลังงานต่ำ

2.1.4 คุณสมบัติทาง system oriented

* รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary scan logic
* สามารถโปรแกรมค่า output slew rate ได้
* สามารถโปรแกรมให้ output มี pull-up หรือ pull-down resister ได้
* ให้กระแส output ได้ตั้งแต่ 12-24 mA (แล้วแต่รุ่น)

2.1.5 ทำการ Config โดยการโหลดเอาแฟ้มข้อมูล Binary

* ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
* มี mode ในการโปรแกรมให้เลือก 6 modes

2.1.6 มีโปรแกรม XACT Development System ที่ทำงานบน PC 386/486 ,NEC PC ,Apollo ,Sun-4 และ Hewlett-Packard 700 series

* สามารถติดต่อกับโปรแกรมอื่นๆ ได้ เช่น VIEWlogic , Mentor Graphics และ Orcad
* มี Automatic Place and Routing ที่ครบ
* มี Interactive Design Editor ที่ใช้สำหรับการทำ optimization
* มี 288 macros ,34 hard macrons ,RAM/ROM compiler

5. โปรแกรมภาษาวีเอชดีแอล

5.1 โปรแกรมบรรยายพฤติกรรมของวงจรบวก

```
--------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity ADDNEW is
        port(A,B:in std_logic_VECTOR(3 downto 0);
             C:out std_logic_VECTOR(3 downto 0));
end ADDNEW;

architecture STRUC of ADDNEW is
begin
        C <= A xor B;
end STRUC;
--------------------------------------------------------------------------------------------------------
```

5.2 โปรแกรมบรรยายพฤติกรรมของส่วนประกอบวงจรคูณ

```
--------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity MULNEW is
        port(X,Y,CARRY:in STD_LOGIC;Z:out STD_LOGIC);
end MULNEW;

architecture STRUC of MULNEW is
begin
        Z <= (X and Y) xor CARRY;
end STRUC;
--------------------------------------------------------------------------------------------------------
```

5.3 โปรแกรมบรรยายโครงสร้างของวงจรคูณ

```
--------------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity MUL4BIT is
        port(A,B:in STD_LOGIC_VECTOR(3 downto 0);
             C:out STD_LOGIC_VECTOR(3 downto 0));
end MUL4BIT;

architecture STRUC of MUL4BIT is
        component MULNEW
              port(X,Y,CARRY:in STD_LOGIC; Z:out STD_LOGIC);
        end component;
        signal AS:STD_LOGIC_VECTOR(1 TO 16);
        signal AS_CONST:STD_LOGIC:='0';
begin
        MUL1:MULNEW port map (A(0),B(3),AS(13),AS(1));
        MUL2:MULNEW port map (A(0),B(2),AS(14),AS(2));
        MUL3:MULNEW port map (A(0),B(1),AS(15),AS(3));
        MUL4:MULNEW port map (A(0),B(0),AS_CONST,AS(4));
        MUL5:MULNEW port map (A(1),B(3),AS_CONST,AS(5));
        MUL6:MULNEW port map (A(1),B(2),AS(1),AS(6));
        MUL7:MULNEW port map (A(1),B(1),AS(2),AS(7));
        MUL8:MULNEW port map (A(1),B(0),AS(3),AS(8));
        MUL9:MULNEW port map (A(2),B(3),AS_CONST,AS(9));
        MUL10:MULNEW port map (A(2),B(2),AS(5),AS(10));
        MUL11:MULNEW port map (A(2),B(1),AS(6),AS(11));
        MUL12:MULNEW port map (A(2),B(0),AS(7),AS(12));
        MUL13:MULNEW port map (A(3),B(3),AS_CONST,AS(13));
        MUL14:MULNEW port map (A(3),B(2),AS(9),AS(14));
```

```
        MUL15:MULNEW port map (A(3),B(1),AS(10),AS(15));
        MUL16:MULNEW port map (A(3),B(0),AS(11),AS(16));
        C(0) <= AS(4) xor AS(15);
        C(1) <= AS(8) xor AS(14);
        C(2) <= AS(12) xor AS(13);
        C(3) <= AS(16) xor AS_CONST;
end STRUC;

configuration CONFIG_MUL4BIT of MUL4BIT is
        for STRUC
                for all:MULNEW use entity work.MULNEW(STRUC);
                end for;
        end for;
end CONFIG_MUL4BIT;
```

5.4 โปรแกรมบรรยายโครงสร้างของวงจรแปลงขนาด 3 จุด

```
library IEEE;
use IEEE.std_logic_1164.all;
entity TRN_3P is
        port(A0_IN_3P,A1_IN_3P,A2_IN_3P:in STD_LOGIC_VECTOR(3 downto 0);
           FUNC:in STD_LOGIC;
           A0_OUT_3P,A1_OUT_3P,A2_OUT_3P:out STD_LOGIC_VECTOR(3 downto 0));
end TRN_3P;

architecture ACH_TRN_3P of TRN_3P is
        component ADDNEW
                port(A,B:in STD_LOGIC_VECTOR(3 downto 0);
```

```
                C:out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component MUL4BIT
        port(A,B:in STD_LOGIC_VECTOR(3 downto 0);
            C:out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    signal  S1,S2,M1,A0_BUF:STD_LOGIC_VECTOR(3 downto 0);
    signal  CONST          :STD_LOGIC_VECTOR(1 to 4);
    constant BELTA_ENCODE  :STD_LOGIC_VECTOR(1 to 4):="0111";
    constant BELTA_DECODE  :STD_LOGIC_VECTOR(1 to 4):="0110";
begin
    TRANFORM_3P:process(FUNC)
    begin
        if FUNC='1' then CONST<=BELTA_ENCODE;
        else CONST<=BELTA_DECODE;
        end if;
    end process;
    ADD1:ADDNEW port map (A1_IN_3P,A2_IN_3P,S1);
    ADD2:ADDNEW port map (S1,A0_IN_3P,A0_BUF);
    MUL1:MUL4BIT port map (CONST,S1,M1);
    ADD3:ADDNEW port map (M1,A0_BUF,S2);
    ADD4:ADDNEW port map (A1_IN_3P,S2,A1_OUT_3P);
    ADD5:ADDNEW port map (A2_IN_3P,S2,A2_OUT_3P);
    A0_OUT_3P <= A0_BUF;
end ACH_TRN_3P;

configuration CONFIG_TRN_3P of TRN_3P is
    for ACH_TRN_3P
    for all:ADDNEW use entity work.ADDNEW(STRUC);
    end for;
```

```
        for all:MUL4BIT use entity work.MUL4BIT(STRUC);
        end for;
        end for;
end CONFIG_TRN_3P;
```

5.5 โปรแกรมบรรยายโครงสร้างของวงจรแปลงขนาด 5 จุด

```
library IEEE;
use IEEE.std_logic_1164.all;
entity TRN_5P is
        port(A0_IN,A1_IN,A2_IN,A3_IN,A4_IN:in STD_LOGIC_VECTOR(3 downto 0);
             FUNC:in STD_LOGIC;
             A0_OUT,A1_OUT,A2_OUT,A3_OUT,A4_OUT:out STD_LOGIC_VECTOR(3
             downto 0));
end TRN_5P;

architecture ACH_TRN_5P of TRN_5P is
        component ADDNEW
              port(A,B:in STD_LOGIC_VECTOR(3 downto 0);
                   C:out STD_LOGIC_VECTOR(3 downto 0));
        end component;
        component MUL4BIT
              port(A,B:in STD_LOGIC_VECTOR(3 downto 0);
                   C:out STD_LOGIC_VECTOR(3 downto 0));
        end component;
    signal S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13:STD_LOGIC_VECTOR(3 downto 0);
    signal M1,M2,M3,M4,M5:STD_LOGIC_VECTOR(3 downto 0);
    type CON is ARRAY(1 to 4) of STD_LOGIC_VECTOR(3 downto 0);
```

```
    signal CONST:CON;
    constant BELTA_ENCODE:CON:=("1101","0111","0100","0011");
    constant BELTA_DECODE:CON:=("1011","0111","0101","0010");
begin   TRANFORM_5P:process(FUNC)
        begin
            if FUNC='1' then CONST<=BELTA_ENCODE;
            else CONST<=BELTA_DECODE;        end if;
      end process;
      ADD1:ADDNEW port map (A2_IN,A3_IN,S1);
      ADD2:ADDNEW port map (A1_IN,A4_IN,S2);
      ADD3:ADDNEW port map (A1_IN,A3_IN,S3);
      ADD4:ADDNEW port map (A2_IN,A4_IN,S4);
      ADD5:ADDNEW port map (S1,S2,S5);
      MUL1:MUL4BIT port map (CONST(1),S5,M1);
      MUL2:MUL4BIT port map (CONST(3),S1,M2);
      MUL3:MUL4BIT port map (CONST(4),S2,M3);
      MUL4:MUL4BIT port map (CONST(2),S3,M4);
      MUL5:MUL4BIT port map (CONST(2),S4,M5);
      ADD6:ADDNEW port map (S13,M1,S6);
      ADD7:ADDNEW port map (S6,M2,S7);
      ADD8:ADDNEW port map (S6,M3,S8);
      ADD9:ADDNEW port map (M5,A2_IN,S9);
      ADD10:ADDNEW port map (M4,A1_IN,S10);
      ADD11:ADDNEW port map (M5,A4_IN,S11);
      ADD12:ADDNEW port map (M4,A3_IN,S12);
      ADD13:ADDNEW port map (S5,A0_IN,S13);
      ADD14:ADDNEW port map (S8,S9,A1_OUT);
      ADD15:ADDNEW port map (S7,S10,A2_OUT);
      ADD16:ADDNEW port map (S7,S11,A3_OUT);
      ADD17:ADDNEW port map (S8,S12,A4_OUT);
```

```
        A0_OUT <= S13;
end ACH_TRN_5P;
configuration CONFIG_TRN_5P of TRN_5P is
        for ACH_TRN_5P
        for all:ADDNEW use entity work.ADDNEW(STRUC);
        end for;
        for all:MUL4BIT use entity work.MUL4BIT(STRUC);
        end for;
        end for;
end CONFIG_TRN_5P;
```

5.6 โปรแกรมบรรยายพฤติกรรมของวงจรกำเนิด Clock

```
library IEEE;
library EXEMPLAR;
use IEEE.std_logic_1164.all;
use EXEMPLAR.EXEMPLAR_1164.all;
entity GEN is
        port(RESET,OSC:in STD_LOGIC;
            CLK:out STD_LOGIC);
end GEN;

architecture RTL of GEN is
        signal COUNTER:STD_LOGIC_VECTOR(3 downto 0);
begin
        CLK <= COUNTER(3);
        CLOCK_GENERATOR:
        process(RESET,OSC)
```

```
    begin
        if RESET = '1' then
            COUNTER <= "0000";
        elsif OSC'EVENT and OSC = '1' then
            if COUNTER /= "1011" then
                COUNTER <= COUNTER+"1";
            else
                COUNTER <= "0000";
            end if;
        end if;
    end process;
end RTL;
```

5.7 โปรแกรมบรรยายพฤติกรรมของวงจรกำเนิด Clock ความถี่ 4 เท่าของ Baud rate

---

```
library EXEMPLAR;
library IEEE;
use EXEMPLAR.EXEMPLAR_1164.all;
use IEEE.std_logic_1164.all;
entity XCLOCK is
    port ( CLOCK,RESET: in STD_LOGIC;
        XCLK: out STD_LOGIC);
end XCLOCK;

architecture ACH_XCLOCK of XCLOCK is
    signal COUNTER: STD_LOGIC_VECTOR(7 DOWNTO 0);
begin
    CLOCK_GENERTOR:
```

```
    process(RESET,CLOCK)
    begin
        if RESET = '1' then
            COUNTER <= "11100111";
            XCLK    <= '0';
        elsif CLOCK'EVENT and CLOCK = '1' then
            if COUNTER /="00000000" then
                COUNTER <= COUNTER+"1";
                XCLK <= '0';
            else
                COUNTER <= "11100111";
                XCLK <= '1';
            end if;
        end if;
    end process;
end ACH_XCLOCK;
```

5.8 โปรแกรมบรรยายพฤติกรรมของวงจรรีจิสเตอร์เซล

```
library IEEE;
use IEEE.std_logic_1164.all;
entity REG_CELL is
    port(DATA_IN:in STD_LOGIC_VECTOR(59 downto 0);
        LOAD:in STD_LOGIC;
        ENABLE:in STD_LOGIC_VECTOR(14 downto 0);
        DATA_OUT:out STD_LOGIC_VECTOR(59 downto 0));
end REG_CELL;
```

```
architecture ACH_REG of REG_CELL is
begin
        REG:
        process(DATA_IN,LOAD)
        begin
                if LOAD'EVENT and LOAD = '1' then
                        for i in 0 to 14 loop
                                if Enable(i)= '1' then
                                DATA_OUT(59-(4*i)) <= DATA_IN(59-(4*i));
                                DATA_OUT(58-(4*i)) <= DATA_IN(58-(4*i));
                                DATA_OUT(57-(4*i)) <= DATA_IN(57-(4*i));
                                DATA_OUT(56-(4*i)) <= DATA_IN(56-(4*i));
                                end if;
                        end loop;
                end if;
        end process;
end ACH_REG;
```

5.9 โปรแกรมบรรยายพฤติกรรมของวงจรอินพุทมัลติเพล็กเซอร์

```
library IEEE;
use IEEE.std_logic_1164.all;
entity MUX_IN is
        port(SBUF_HI,SBUF_LOW:in STD_LOGIC_VECTOR(3 downto 0);
            OPERATION,SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
            TRN_3P_A0,TRN_3P_A1,TRN_3P_A2:in STD_LOGIC_VECTOR(3 downto 0);
            TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4
            :in STD_LOGIC_VECTOR(3 downto 0);
```

```
        DATA_OUT_CELL0,DATA_OUT_CELL1,DATA_OUT_CELL2,
        DATA_OUT_CELL3, DATA_OUT_CELL4,DATA_OUT_CELL5,
        DATA_OUT_CELL6,DATA_OUT_CELL7,DATA_OUT_CELL8,
        DATA_OUT_CELL9,DATA_OUT_CELL10,DATA_OUT_CELL11,
        DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14
        :in STD_LOGIC_VECTOR(3 downto 0);
        DATA_IN_CELL0,DATA_IN_CELL1,DATA_IN_CELL2,DATA_IN_CELL3,
        DATA_IN_CELL4,DATA_IN_CELL5,DATA_IN_CELL6,DATA_IN_CELL7,
        DATA_IN_CELL8,DATA_IN_CELL9,DATA_IN_CELL10,DATA_IN_CELL11,
        DATA_IN_CELL12,DATA_IN_CELL13,DATA_IN_CELL14
        :out STD_LOGIC_VECTOR(3 downto 0));
end MUX_IN;

architecture ACH_MUX_IN of MUX_IN is
begin
    MUX:  process(SBUF_HI,SBUF_LOW,OPERATION,SCOUNT,
          TRN_3P_A0,TRN_3P_A1,TRN_3P_A2,
          TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4,
          DATA_OUT_CELL0,DATA_OUT_CELL1,DATA_OUT_CELL2,
          DATA_OUT_CELL3,DATA_OUT_CELL4,DATA_OUT_CELL5,
          DATA_OUT_CELL6,DATA_OUT_CELL7,DATA_OUT_CELL8,
          DATA_OUT_CELL9,DATA_OUT_CELL10,DATA_OUT_CELL11,
          DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14)
    begin
        case OPERATION is
        when "001" =>
            case SCOUNT is
            when "000" =>
                DATA_IN_CELL0 <= SBUF_HI;
                DATA_IN_CELL1 <= "0000";
```

```
            DATA_IN_CELL2 <= "0000";
            DATA_IN_CELL3 <= "0000";
            DATA_IN_CELL4 <= "0000";
            DATA_IN_CELL5 <= "0000";
            DATA_IN_CELL6 <= "0000";
            DATA_IN_CELL7 <= "0000";
            DATA_IN_CELL8 <= "0000";
            DATA_IN_CELL9 <= "0000";
            DATA_IN_CELL10 <= "0000";
            DATA_IN_CELL11 <= "0000";
            DATA_IN_CELL12 <= SBUF_LOW;
            DATA_IN_CELL13 <= "0000";
            DATA_IN_CELL14 <= "0000";
          when "001" =>
            DATA_IN_CELL0 <= "0000";
            DATA_IN_CELL1 <= SBUF_LOW;
            DATA_IN_CELL2 <= "0000";
            DATA_IN_CELL3 <= "0000";
            DATA_IN_CELL4 <= "0000";
            DATA_IN_CELL5 <= "0000";
            DATA_IN_CELL6 <= "0000";
            DATA_IN_CELL7 <= "0000";
            DATA_IN_CELL8 <= "0000";
            DATA_IN_CELL9 <= SBUF_HI;
            DATA_IN_CELL10 <= "0000";
            DATA_IN_CELL11 <= "0000";
            DATA_IN_CELL12 <= "0000";
            DATA_IN_CELL13 <= "0000";
            DATA_IN_CELL14 <= "0000";
          when "010" =>
```

```
DATA_IN_CELL0 <= "0000";
DATA_IN_CELL1 <= "0000";
DATA_IN_CELL2 <= "0000";
DATA_IN_CELL3 <= "0000";
DATA_IN_CELL4 <= "0000";
DATA_IN_CELL5 <= SBUF_LOW;
DATA_IN_CELL6 <= "0000";
DATA_IN_CELL7 <= "0000";
DATA_IN_CELL8 <= "0000";
DATA_IN_CELL9 <= "0000";
DATA_IN_CELL10 <= "0000";
DATA_IN_CELL11 <= "0000";
DATA_IN_CELL12 <= "0000";
DATA_IN_CELL13 <= SBUF_HI;
DATA_IN_CELL14 <= "0000";
when "011" =>
DATA_IN_CELL0 <= "0000";
DATA_IN_CELL1 <= "0000";
DATA_IN_CELL2 <= SBUF_HI;
DATA_IN_CELL3 <= "0000";
DATA_IN_CELL4 <= "0000";
DATA_IN_CELL5 <= "0000";
DATA_IN_CELL6 <= "0000";
DATA_IN_CELL7 <= "0000";
DATA_IN_CELL8 <= "0000";
DATA_IN_CELL9 <= "0000";
DATA_IN_CELL10 <= "0000";
DATA_IN_CELL11 <= "0000";
DATA_IN_CELL12 <= "0000";
DATA_IN_CELL13 <= "0000";
```

```
        DATA_IN_CELL14 <= SBUF_LOW;
    when "100" =>
        DATA_IN_CELL0 <= "0000";
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= SBUF_LOW;
        DATA_IN_CELL4 <= "0000";
        DATA_IN_CELL5 <= "0000";
        DATA_IN_CELL6 <= SBUF_HI;
        DATA_IN_CELL7 <= "0000";
        DATA_IN_CELL8 <= "0000";
        DATA_IN_CELL9 <= "0000";
        DATA_IN_CELL10 <= "0000";
        DATA_IN_CELL11 <= "0000";
        DATA_IN_CELL12 <= "0000";
        DATA_IN_CELL13 <= "0000";
        DATA_IN_CELL14 <= "0000";
    when "101" =>
        DATA_IN_CELL0 <= "0000";
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= "0000";
        DATA_IN_CELL4 <= "0000";
        DATA_IN_CELL5 <= "0000";
        DATA_IN_CELL6 <= "0000";
        DATA_IN_CELL7 <= SBUF_LOW;
        DATA_IN_CELL8 <= "0000";
        DATA_IN_CELL9 <= "0000";
        DATA_IN_CELL10 <= SBUF_HI;
        DATA_IN_CELL11 <= "0000";
```

```
        DATA_IN_CELL12 <= "0000";
        DATA_IN_CELL13 <= "0000";
        DATA_IN_CELL14 <= "0000";
    when "110" =>
        DATA_IN_CELL0 <= "0000";
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= "0000";
        DATA_IN_CELL4 <= SBUF_HI;
        DATA_IN_CELL5 <= "0000";
        DATA_IN_CELL6 <= "0000";
        DATA_IN_CELL7 <= "0000";
        DATA_IN_CELL8 <= "0000";
        DATA_IN_CELL9 <= "0000";
        DATA_IN_CELL10 <= "0000";
        DATA_IN_CELL11 <= SBUF_LOW;
        DATA_IN_CELL12 <= "0000";
        DATA_IN_CELL13 <= "0000";
        DATA_IN_CELL14 <= "0000";
    when others =>
        DATA_IN_CELL0 <= "0000";
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= "0000";
        DATA_IN_CELL4 <= "0000";
        DATA_IN_CELL5 <= "0000";
        DATA_IN_CELL6 <= "0000";
        DATA_IN_CELL7 <= "0000";
        DATA_IN_CELL8 <= SBUF_HI;
        DATA_IN_CELL9 <= "0000";
```

```
        DATA_IN_CELL10 <= "0000";
        DATA_IN_CELL11 <= "0000";
        DATA_IN_CELL12 <= "0000";
        DATA_IN_CELL13 <= "0000";
        DATA_IN_CELL14 <= "0000";
    end case;
when "010" =>
        DATA_IN_CELL0 <= TRN_3P_A0;
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= "0000";
        DATA_IN_CELL4 <= "0000";
        DATA_IN_CELL5 <= TRN_3P_A1;
        DATA_IN_CELL6 <= "0000";
        DATA_IN_CELL7 <= "0000";
        DATA_IN_CELL8 <= "0000";
        DATA_IN_CELL9 <= "0000";
        DATA_IN_CELL10 <= TRN_3P_A2;
        DATA_IN_CELL11 <= "0000";
        DATA_IN_CELL12 <= "0000";
        DATA_IN_CELL13 <= "0000";
        DATA_IN_CELL14 <= "0000";
 when "011" =>
        DATA_IN_CELL0 <= "0000";
        DATA_IN_CELL1 <= "0000";
        DATA_IN_CELL2 <= "0000";
        DATA_IN_CELL3 <= "0000";
        DATA_IN_CELL4 <= "0000";
        DATA_IN_CELL5 <= "0000";
        DATA_IN_CELL6 <= "0000";
```

```
        DATA_IN_CELL7 <= "0000";
        DATA_IN_CELL8 <= "0000";
        DATA_IN_CELL9 <= "0000";
        DATA_IN_CELL10 <= TRN_5P_A0;
        DATA_IN_CELL11 <= TRN_5P_A1;
        DATA_IN_CELL12 <= TRN_5P_A2;
        DATA_IN_CELL13 <= TRN_5P_A3;
        DATA_IN_CELL14 <= TRN_5P_A4;
    when "100" =>
        DATA_IN_CELL0 <= DATA_OUT_CELL1;
        DATA_IN_CELL1 <= DATA_OUT_CELL2;
        DATA_IN_CELL2 <= DATA_OUT_CELL3;
        DATA_IN_CELL3 <= DATA_OUT_CELL4;
        DATA_IN_CELL4 <= DATA_OUT_CELL5;
        DATA_IN_CELL5 <= DATA_OUT_CELL6;
        DATA_IN_CELL6 <= DATA_OUT_CELL7;
        DATA_IN_CELL7 <= DATA_OUT_CELL8;
        DATA_IN_CELL8 <= DATA_OUT_CELL9;
        DATA_IN_CELL9 <= DATA_OUT_CELL10;
        DATA_IN_CELL10 <=DATA_OUT_CELL11;
        DATA_IN_CELL11 <=DATA_OUT_CELL12;
        DATA_IN_CELL12 <=DATA_OUT_CELL13;
        DATA_IN_CELL13 <=DATA_OUT_CELL14;
        DATA_IN_CELL14 <=DATA_OUT_CELL0;
    when "101" =>
        DATA_IN_CELL0 <= DATA_OUT_CELL5;
        DATA_IN_CELL1 <= DATA_OUT_CELL6;
        DATA_IN_CELL2 <= DATA_OUT_CELL7;
        DATA_IN_CELL3 <= DATA_OUT_CELL8;
        DATA_IN_CELL4 <= DATA_OUT_CELL9;
```

```
                DATA_IN_CELL5 <= DATA_OUT_CELL10;
                DATA_IN_CELL6 <= DATA_OUT_CELL11;
                DATA_IN_CELL7 <= DATA_OUT_CELL12;
                DATA_IN_CELL8 <= DATA_OUT_CELL13;
                DATA_IN_CELL9 <= DATA_OUT_CELL14;
                DATA_IN_CELL10 <=DATA_OUT_CELL0;
                DATA_IN_CELL11 <=DATA_OUT_CELL1;
                DATA_IN_CELL12 <=DATA_OUT_CELL2;
                DATA_IN_CELL13 <=DATA_OUT_CELL3;
                DATA_IN_CELL14 <=DATA_OUT_CELL4;
            when others =>
                DATA_IN_CELL0 <= "0000";
                DATA_IN_CELL1 <= "0000";
                DATA_IN_CELL2 <= "0000";
                DATA_IN_CELL3 <= "0000";
                DATA_IN_CELL4 <= "0000";
                DATA_IN_CELL5 <= "0000";
                DATA_IN_CELL6 <= "0000";
                DATA_IN_CELL7 <= "0000";
                DATA_IN_CELL8 <= "0000";
                DATA_IN_CELL9 <= "0000";
                DATA_IN_CELL10 <= "0000";
                DATA_IN_CELL11 <= "0000";
                DATA_IN_CELL12 <= "0000";
                DATA_IN_CELL13 <= "0000";
                DATA_IN_CELL14 <= "0000";
        end case;
    end process;
end ACH_MUX_IN;
```

5.10 โปรแกรมบรรยายพฤติกรรมของวงจรเซลมัลติเพล็กเซอร์

```
--------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity MUX_ADDR is
        port(OPERATION,SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
            ENABLE_CELL0,ENABLE_CELL1,ENABLE_CELL2,ENABLE_CELL3,
            ENABLE_CELL4,ENABLE_CELL5,ENABLE_CELL6,ENABLE_CELL7,
            ENABLE_CELL8,ENABLE_CELL9,ENABLE_CELL10,ENABLE_CELL11,
            ENABLE_CELL12,ENABLE_CELL13,ENABLE_CELL14:out STD_LOGIC);
end MUX_ADDR;

architecture ACH_MUX_CELL_ADDRESS of MUX_ADDR is
begin
        MUX:
        process(OPERATION,SCOUNT)
        begin
                case OPERATION is
                when "001" =>
                    case SCOUNT is
                    when "000" =>
                        ENABLE_CELL0 <= '1';
                        ENABLE_CELL1 <= '0';
                        ENABLE_CELL2 <= '0';
                        ENABLE_CELL3 <= '0';
                        ENABLE_CELL4 <= '0';
                        ENABLE_CELL5 <= '0';
                        ENABLE_CELL6 <= '0';
                        ENABLE_CELL7 <= '0';
                        ENABLE_CELL8 <= '0';
```

```
        ENABLE_CELL9 <= '0';
        ENABLE_CELL10 <= '0';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '1';
        ENABLE_CELL13 <= '0';
        ENABLE_CELL14 <= '0';
    when "001" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '1';
        ENABLE_CELL2 <= '0';
        ENABLE_CELL3 <= '0';
        ENABLE_CELL4 <= '0';
        ENABLE_CELL5 <= '0';
        ENABLE_CELL6 <= '0';
        ENABLE_CELL7 <= '0';
        ENABLE_CELL8 <= '0';
        ENABLE_CELL9 <= '1';
        ENABLE_CELL10 <= '0';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '0';
        ENABLE_CELL13 <= '0';
        ENABLE_CELL14 <= '0';
    when "010" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '0';
        ENABLE_CELL2 <= '0';
        ENABLE_CELL3 <= '0';
        ENABLE_CELL4 <= '0';
        ENABLE_CELL5 <= '1';
        ENABLE_CELL6 <= '0';
```

```
        ENABLE_CELL7 <= '0';
        ENABLE_CELL8 <= '0';
        ENABLE_CELL9 <= '0';
        ENABLE_CELL10 <= '0';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '0';
        ENABLE_CELL13 <= '1';
        ENABLE_CELL14 <= '0';
    when "011" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '0';
        ENABLE_CELL2 <= '1';
        ENABLE_CELL3 <= '0';
        ENABLE_CELL4 <= '0';
        ENABLE_CELL5 <= '0';
        ENABLE_CELL6 <= '0';
        ENABLE_CELL7 <= '0';
        ENABLE_CELL8 <= '0';
        ENABLE_CELL9 <= '0';
        ENABLE_CELL10 <= '0';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '0';
        ENABLE_CELL13 <= '0';
        ENABLE_CELL14 <= '1';
    when "100" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '0';
        ENABLE_CELL2 <= '0';
        ENABLE_CELL3 <= '1';
        ENABLE_CELL4 <= '0';
```

```
        ENABLE_CELL5 <= '0';
        ENABLE_CELL6 <= '1';
        ENABLE_CELL7 <= '0';
        ENABLE_CELL8 <= '0';
        ENABLE_CELL9 <= '0';
        ENABLE_CELL10 <= '0';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '0';
        ENABLE_CELL13 <= '0';
        ENABLE_CELL14 <= '0';
      when "101" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '0';
        ENABLE_CELL2 <= '0';
        ENABLE_CELL3 <= '0';
        ENABLE_CELL4 <= '0';
        ENABLE_CELL5 <= '0';
        ENABLE_CELL6 <= '0';
        ENABLE_CELL7 <= '1';
        ENABLE_CELL8 <= '0';
        ENABLE_CELL9 <= '0';
        ENABLE_CELL10 <= '1';
        ENABLE_CELL11 <= '0';
        ENABLE_CELL12 <= '0';
        ENABLE_CELL13 <= '0';
        ENABLE_CELL14 <= '0';
      when "110" =>
        ENABLE_CELL0 <= '0';
        ENABLE_CELL1 <= '0';
        ENABLE_CELL2 <= '0';
```

```
            ENABLE_CELL3 <= '0';
            ENABLE_CELL4 <= '1';
            ENABLE_CELL5 <= '0';
            ENABLE_CELL6 <= '0';
            ENABLE_CELL7 <= '0';
            ENABLE_CELL8 <= '0';
            ENABLE_CELL9 <= '0';
            ENABLE_CELL10 <= '0';
            ENABLE_CELL11 <= '1';
            ENABLE_CELL12 <= '0';
            ENABLE_CELL13 <= '0';
            ENABLE_CELL14 <= '0';
          when others =>
            ENABLE_CELL0 <= '0';
            ENABLE_CELL1 <= '0';
            ENABLE_CELL2 <= '0';
            ENABLE_CELL3 <= '0';
            ENABLE_CELL4 <= '0';
            ENABLE_CELL5 <= '0';
            ENABLE_CELL6 <= '0';
            ENABLE_CELL7 <= '0';
            ENABLE_CELL8 <= '1';
            ENABLE_CELL9 <= '0';
            ENABLE_CELL10 <= '0';
            ENABLE_CELL11 <= '0';
            ENABLE_CELL12 <= '0';
            ENABLE_CELL13 <= '0';
            ENABLE_CELL14 <= '0';
        end case;
      when "010" =>
```

```
            ENABLE_CELL0 <= '1';
            ENABLE_CELL1 <= '0';
            ENABLE_CELL2 <= '0';
            ENABLE_CELL3 <= '0';
            ENABLE_CELL4 <= '0';
            ENABLE_CELL5 <= '1';
            ENABLE_CELL6 <= '0';
            ENABLE_CELL7 <= '0';
            ENABLE_CELL8 <= '0';
            ENABLE_CELL9 <= '0';
            ENABLE_CELL10 <= '1';
            ENABLE_CELL11 <= '0';
            ENABLE_CELL12 <= '0';
            ENABLE_CELL13 <= '0';
            ENABLE_CELL14 <= '0';
        when "011" =>
            ENABLE_CELL0 <= '0';
            ENABLE_CELL1 <= '0';
            ENABLE_CELL2 <= '0';
            ENABLE_CELL3 <= '0';
            ENABLE_CELL4 <= '0';
            ENABLE_CELL5 <= '0';
            ENABLE_CELL6 <= '0';
            ENABLE_CELL7 <= '0';
            ENABLE_CELL8 <= '0';
            ENABLE_CELL9 <= '0';
            ENABLE_CELL10 <= '1';
            ENABLE_CELL11 <= '1';
            ENABLE_CELL12 <= '1';
            ENABLE_CELL13 <= '1';
```

```
            ENABLE_CELL14 <= '1';
        when "100" =>
            ENABLE_CELL0 <= '1';
            ENABLE_CELL1 <= '1';
            ENABLE_CELL2 <= '1';
            ENABLE_CELL3 <= '1';
            ENABLE_CELL4 <= '1';
            ENABLE_CELL5 <= '1';
            ENABLE_CELL6 <= '1';
            ENABLE_CELL7 <= '1';
            ENABLE_CELL8 <= '1';
            ENABLE_CELL9 <= '1';
            ENABLE_CELL10 <= '1';
            ENABLE_CELL11 <= '1';
            ENABLE_CELL12 <= '1';
            ENABLE_CELL13 <= '1';
            ENABLE_CELL14 <= '1';
        when "101" =>
            ENABLE_CELL0 <= '1';
            ENABLE_CELL1 <= '1';
            ENABLE_CELL2 <= '1';
            ENABLE_CELL3 <= '1';
            ENABLE_CELL4 <= '1';
            ENABLE_CELL5 <= '1';
            ENABLE_CELL6 <= '1';
            ENABLE_CELL7 <= '1';
            ENABLE_CELL8 <= '1';
            ENABLE_CELL9 <= '1';
            ENABLE_CELL10 <= '1';
            ENABLE_CELL11 <= '1';
```

```
                ENABLE_CELL12 <= '1';
                ENABLE_CELL13 <= '1';
                ENABLE_CELL14 <= '1';
            when others =>
                ENABLE_CELL0 <= '0';
                ENABLE_CELL1 <= '0';
                ENABLE_CELL2 <= '0';
                ENABLE_CELL3 <= '0';
                ENABLE_CELL4 <= '0';
                ENABLE_CELL5 <= '0';
                ENABLE_CELL6 <= '0';
                ENABLE_CELL7 <= '0';
                ENABLE_CELL8 <= '0';
                ENABLE_CELL9 <= '0';
                ENABLE_CELL10 <= '0';
                ENABLE_CELL11 <= '0';
                ENABLE_CELL12 <= '0';
                ENABLE_CELL13 <= '0';
                ENABLE_CELL14 <= '0';
        end case;
    end process;
end ACH_MUX_CELL_ADDRESS;
-------------------------------------------------------------------------------------------------------
```

5.11 โปรแกรมบรรยายพฤติกรรมของสเตจแมทชีนของการแปลง

```
-------------------------------------------------------------------------------------------------------
library IEEE;
library EXEMPLAR;
use IEEE.std_logic_1164.all;
```

```
use EXEMPLAR.EXEMPLAR_1164.all;
entity STA_MACH is
        port(RESET,START_TRN,CLK:in STD_LOGIC;
            OPERATION:out STD_LOGIC_VECTOR(2 downto 0);
            LOAD_CELL,DONE:out STD_LOGIC);
end STA_MACH;

architecture ACH_STATE_MACHINE of STA_MACH is
        type STATE_TYPE is (S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11);
        signal CURRENT_STATE,NEXT_STATE:STATE_TYPE;
        signal COUNTER:STD_LOGIC_VECTOR(2 downto 0);
        signal DEC_COUNTER,SET_COUNTER,LOAD_COUNTER:STD_LOGIC;
        signal COUNTER_INITIAL,COUNTER_IN:STD_LOGIC_VECTOR(2 downto 0);
begin
        SYNCH:process(RESET,CLK)
        begin
                if RESET = '1' then
                        CURRENT_STATE <= S1;
                elsif CLK'EVENT and CLK = '1' then
                        CURRENT_STATE <= NEXT_STATE;
                end if;
        end process;

        COMBINATION:
        process(CURRENT_STATE,START_TRN,COUNTER)
        begin
                case CURRENT_STATE is
                when S1 =>
                        if START_TRN = '0' then
                                DEC_COUNTER  <= '0';
```

```
                LOAD_CELL    <= '0';
                OPERATION    <= "000";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S1;
        else
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "010";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '1';
                DONE         <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S2 =>
                DEC_COUNTER  <= '1';
                LOAD_CELL    <= '1';
                OPERATION    <= "010";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S3;
when S3 =>
                DEC_COUNTER  <= '1';
                LOAD_CELL    <= '0';
                OPERATION    <= "100";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '1';
                DONE         <= '0';
```

```
                NEXT_STATE   <=  S4;
when S4 =>
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '1';
                OPERATION    <= "100";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S5;
when S5 =>
        if COUNTER /= "000" then
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "010";
                SET_COUNTER  <= '0';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S2;
        else
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "011";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S6;
        end if;

when S6 =>
                DEC_COUNTER  <= '0';
```

```
                LOAD_CELL    <= '0';
                OPERATION    <= "011";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '1';
                DONE         <= '0';
                NEXT_STATE   <=  S7;
        when S7 =>
                DEC_COUNTER  <= '1';
                LOAD_CELL    <= '1';
                OPERATION    <= "011";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S8;
        when S8 =>
                DEC_COUNTER  <= '1';
                LOAD_CELL    <= '0';
                OPERATION    <= "101";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '1';
                DONE         <= '0';
                NEXT_STATE   <=  S9;
        when S9 =>
            if COUNTER /= "000" then
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '1';
                OPERATION    <= "101";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
```

```
                NEXT_STATE  <=  S10;
            else
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "000";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S11;
            end if;
        when S10 =>
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "011";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S7;
        when S11 =>
                DEC_COUNTER  <= '0';
                LOAD_CELL    <= '0';
                OPERATION    <= "000";
                SET_COUNTER  <= '1';
                LOAD_COUNTER <= '0';
                DONE         <= '1';
                NEXT_STATE   <=  S11;
        end case;
    end process;

    COUNT:
```

```
    process(DEC_COUNTER,SET_COUNTER,LOAD_COUNTER,
            COUNTER_INITIAL,COUNTER)
     begin
            if SET_COUNTER = '0' then
                  COUNTER_INITIAL <= "101";
            else
                  COUNTER_INITIAL <= "011";
            end if;
            if DEC_COUNTER = '0' then
                  COUNTER_IN <= COUNTER_INITIAL;
            else
                  COUNTER_IN <= COUNTER-"1";
            end if;
            if LOAD_COUNTER'EVENT and LOAD_COUNTER = '1' then
                  COUNTER <= COUNTER_IN;
            end if;
    end process;
end ACH_STATE_MACHINE;
```

5.12 โปรแกรมบรรยายพฤติกรรมของวงจรเลือกเอาท์พุท

```
library IEEE;
use IEEE.std_logic_1164.all;
entity OUT_MAP is
      port(CELL_0,CELL_1,CELL_2,CELL_3,CELL_4,CELL_5,CELL_6,
          CELL_7,CELL_8,CELL_9,CELL_10,CELL_11,CELL_12,CELL_13,
          CELL_14:in STD_LOGIC_VECTOR(3 downto 0);
          Sop:in STD_LOGIC;
```

```
        Scount:in STD_LOGIC_VECTOR(2 downto 0);
        DATA:out STD_LOGIC_VECTOR(7 downto 0));
end OUT_MAP;

architecture ACH_OUT_MAP of OUT_MAP is
begin
    MAPPING:
    process(Sop,Scount,
        CELL_0,CELL_1,CELL_2,CELL_3,CELL_4,CELL_5,
        CELL_6,CELL_7,CELL_8,CELL_9,CELL_10,CELL_11,
        CELL_12,CELL_13,CELL_14)
    begin
    if Sop = '0' then
        case Scount is
        when "000" =>
            DATA <= CELL_0 & CELL_6;
        when "001" =>
            DATA <= CELL_12 & CELL_3;
        when "010" =>
            DATA <= CELL_9 & CELL_10;
        when "011" =>
            DATA <= CELL_1 & CELL_7;
        when "100" =>
            DATA <= CELL_13 & CELL_4;
        when "101" =>
            DATA <= CELL_5 & CELL_11;
        when "110" =>
            DATA <= CELL_2 & CELL_8;
        when others =>
            DATA <= CELL_14 & "0000";
```

```
        end case;
    else
        case Scount is
        when "000" =>
            DATA <= CELL_0 & CELL_12;
        when "001" =>
            DATA <= CELL_9 & CELL_1;
        when "010" =>
            DATA <= CELL_13 & CELL_5;
        when "011" =>
            DATA <= CELL_2 & CELL_14;
        when "100" =>
            DATA <= CELL_6 & CELL_3;
        when "101" =>
            DATA <= CELL_10 & CELL_7;
        when "110" =>
            DATA <= CELL_4 & CELL_11;
        when others =>
            DATA <= CELL_8 & "0000";
        end case;
    end if;
    end process;
end ACH_OUT_MAP;
```

5.13 โปรแกรมบรรยายพฤติกรรมของวงจรบัฟเฟอร์รีจีสเตอร์

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity  SBUF is
        port ( RESET,RxD,LOAD,DIR,CLOCK : in STD_LOGIC;
               DIN : in STD_LOGIC_VECTOR( 7 downto 0);
               S_OUT, STOPBIT, ZERO : out STD_LOGIC;
               DOUT : out STD_LOGIC_VECTOR(7 downto 0));
end SBUF;

architecture ACH_SBUF_REG of SBUF is
        signal SBUF : STD_LOGIC_VECTOR( 10 downto 0 );
        signal SHF : STD_LOGIC;
begin
        STOPBIT <= SBUF(10);
        S_OUT   <= SBUF(0);
        DOUT    <= SBUF( 9 downto 2 );
        ZERO    <= '1' when SBUF( 10 downto 1) <= "0000000000" else '0';
        SHF     <= RxD when DIR = '1' else '0';
        SBUF_GEN:process( CLOCK,RESET)
        begin
                if RESET = '1' then
                        SBUF <= "01000000000" ;
                elsif CLOCK'EVENT and CLOCK = '1' then
                        if LOAD = '1' then
                                SBUF <= '1' & DIN & "01";
                        else
                                SBUF <= SHF & SBUF ( 10 downto 1 );
                        end if;
                end if;
        end process;
end ACH_SBUF_REG;
```

```
        if SEND_RECEIVE = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '1';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S2;
        else
                LOAD <= '1';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S7;
        end if;
when S2 =>
        if RxD = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S3;
        else
```

5.14 โปรแกรมบรรยายพฤติกรรมของสเตจแมทชีนควมคุมการรับส่งข้อมูลของชิพ

```
--------------------------------------------------------------------------------------------------------------
library IEEE;
library EXEMPLAR;
use IEEE.std_logic_1164.all;
use EXEMPLAR.EXEMPLAR_1164.all;
entity STA_SBUF is
        port(FRESET,SEND_RECEIVE,XCLK,RxD,BIT_0,BIT_10,SZERO:in STD_LOGIC;
            LOAD,SCLK,DIR,RESET_SBUF,SDONE,SERROR,LOAD_FUNC:out
STD_LOGIC);
end STA_SBUF;

architecture ACH_STATE_MACHINE_SBUF of STA_SBUF is
        type STATE_TYPE is (S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15);
        signal CURRENT_STATE,NEXT_STATE:STATE_TYPE;
begin
        SYNCH:process(FRESET,XCLK)
        begin
                if FRESET = '1' then
                        CURRENT_STATE <= S1;
                elsif XCLK'EVENT and XCLK = '1' then
                        CURRENT_STATE <= NEXT_STATE;
                end if;
        end process;

        COMBINATION:
        process(CURRENT_STATE,SEND_RECEIVE,RxD,BIT_0,BIT_10,SZERO)
        begin
                case CURRENT_STATE is
                when S1 =>
```

```
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S3 =>
        if RxD = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '1';
                NEXT_STATE   <=  S4;
        else
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S4 =>
```

```
        if RxD = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S5;
        else
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S5 =>
        if RxD = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S6;
        else
```

```
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S6 =>
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S9;
when S7 =>
                LOAD <= '1';
                SCLK <= '1';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S8;
when S8 =>
                LOAD <= '0';
                SCLK <= '0';
```

```
                DIR  <= '0';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S9;
when S9 =>
        if   SEND_RECEIVE = '0' then
                        LOAD <= '0';
                        SCLK <= '1';
                        DIR  <= '1';
                        RESET_SBUF <= '0';
                        SDONE <= '0';
                        SERROR <= '0';
                        LOAD_FUNC <= '0';
                        NEXT_STATE   <=  S10;
        else
                        LOAD <= '0';
                        SCLK <= '1';
                        DIR  <= '0';
                        RESET_SBUF <= '0';
                        SDONE <= '0';
                        SERROR <= '0';
                        LOAD_FUNC <= '0';
                        NEXT_STATE   <=  S10;
        end if;
when S10 =>
        if   SEND_RECEIVE = '0' then
                        LOAD <= '0';
                        SCLK <= '0';
```

```
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S11;
        else
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '0';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S11;
        end if;
when S11 =>
        if   SEND_RECEIVE = '0' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S12;
        else
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '0';
```

```
                    RESET_SBUF <= '0';
                    SDONE <= '0';
                    SERROR <= '0';
                    LOAD_FUNC <= '0';
                    NEXT_STATE   <=  S12;
            end if;
        when S12 =>
            if (((not SEND_RECEIVE) and BIT_0) or (SEND_RECEIVE and
SZERO))= '1' then
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S13;
            elsif   SEND_RECEIVE = '0' then
                    LOAD <= '0';
                    SCLK <= '0';
                    DIR  <= '1';
                    RESET_SBUF <= '0';
                    SDONE <= '0';
                    SERROR <= '0';
                    LOAD_FUNC <= '0';
                    NEXT_STATE   <=  S9;
                else
                    LOAD <= '0';
                    SCLK <= '0';
                    DIR  <= '0';
```

```
                    RESET_SBUF <= '0';
                    SDONE <= '0';
                    SERROR <= '0';
                    LOAD_FUNC <= '0';
                    NEXT_STATE   <=  S9;
            end if;
        when S13 =>
            if (((not SEND_RECEIVE) and BIT_0 and BIT_10) or
(SEND_RECEIVE and SZERO))= '0' then
                    LOAD <= '0';
                    SCLK <= '0';
                    DIR  <= '1';
                    RESET_SBUF <= '0';
                    SDONE <= '0';
                    SERROR <= '1';
                    LOAD_FUNC <= '0';
                    NEXT_STATE   <=  S14;
            else
                    LOAD <= '0';
                    SCLK <= '0';
                    DIR  <= '1';
                    RESET_SBUF <= '0';
                    SDONE <= '0';
                    SERROR <= '0';
                    LOAD_FUNC <= '0';
                    NEXT_STATE   <=  S15;
            end if;
        when S14 =>
                    LOAD <= '0';
                    SCLK <= '0';
```

```
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '0';
                SERROR <= '1';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S14;
        when S15 =>
                LOAD <= '0';
                SCLK <= '0';
                DIR  <= '1';
                RESET_SBUF <= '0';
                SDONE <= '1';
                SERROR <= '0';
                LOAD_FUNC <= '0';
                NEXT_STATE   <=  S15;
        end case;
    end process;
end ACH_STATE_MACHINE_SBUF;
```

5.15 โปรแกรมบรรยายพฤติกรรมของวงจรนับ

```
library IEEE;
library EXEMPLAR;
use IEEE.std_logic_1164.all;
use EXEMPLAR.EXEMPLAR_1164.all;
entity SCOUNT is
    port(INC,LOAD,RESET:in STD_LOGIC;
        SCOUNT:out STD_LOGIC_VECTOR(2 downto 0));
```

```
end SCOUNT;

architecture ACH_SCOUNT of SCOUNT is
        signal DATA,SCOUNT_BUF:STD_LOGIC_VECTOR(2 downto 0);
begin
        COUNTER:
        process(INC,LOAD,RESET,SCOUNT_BUF)
        begin
                if INC = '1' then
                        DATA <= SCOUNT_BUF + "1";
                else
                        DATA <= SCOUNT_BUF;
                end if;
                if RESET = '1' then
                        SCOUNT_BUF <= "000";
                elsif LOAD'EVENT and LOAD = '1' then
                        SCOUNT_BUF <= DATA;
                end if;
        end process;
        SCOUNT <= SCOUNT_BUF;
end ACH_SCOUNT;
```

5.16 โปรแกรมบรรยายพฤติกรรมของสเตจแมทชีนควบคุมการทำงานของชิพ

```
library IEEE;
library EXEMPLAR;
use IEEE.std_logic_1164.all;
use EXEMPLAR.EXEMPLAR_1164.all;
entity STA_IO is
```

```
        port(RESET,SDONE,TRN_DONE,CLOCK,XCLK:in STD_LOGIC;
            SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
            SEND_RECEIVE,FRESET,Sop,LOAD_CELL,INC,RESET_SCOUNT,
            LOAD_SCOUNT,RESET_TRN,START_TRN,DONE:out STD_LOGIC;
            OPERATION:out STD_LOGIC_VECTOR(2 downto 0));

end STA_IO;

architecture ACH_STATE_MACHINE_IO of STA_IO is
        type STATE_TYPE is (S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16,S17,S18,S19);
        signal CURRENT_STATE,NEXT_STATE:STATE_TYPE;
        signal DELAY:STD_LOGIC;
        signal COUNT_XCLK:STD_LOGIC_VECTOR(5 DOWNTO 0);
begin
        DELAY1:process(XCLK,CURRENT_STATE)
        begin
            if CURRENT_STATE /= S17 then
                COUNT_XCLK <= "000000";
                DELAY <= '1';
            elsif XCLK'EVENT and XCLK = '1' then
                if COUNT_XCLK /= "111111" then
                    COUNT_XCLK <= COUNT_XCLK + "1";
                    DELAY <= '1';
                else
                    COUNT_XCLK <= "000000";
                    DELAY <= '0';
                end if;
            end if;
        end process;

        SYNCH:process(RESET,CLOCK)
        begin
            if RESET = '1' then
                CURRENT_STATE <= S1;
```

```
        elsif CLOCK'EVENT and CLOCK = '1' then
                CURRENT_STATE <= NEXT_STATE;
        end if;
end process;

COMBINATION:
process(CURRENT_STATE,SDONE,TRN_DONE,SCOUNT,DELAY)
begin
        case CURRENT_STATE is
        when S1 =>
                SEND_RECEIVE <= '0';
                FRESET       <= '1';
                Sop          <= '1';
                RESET_SCOUNT <= '1';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S2;
        when S2 =>
                SEND_RECEIVE <= '0';
                FRESET       <= '0';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
```

```
            NEXT_STATE   <=  S3;
when S3 =>
        if SDONE = '0' then
            SEND_RECEIVE <= '0';
            FRESET       <= '0';
            Sop          <= '1';
            RESET_SCOUNT <= '0';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "001";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '1';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S3;
        else
            SEND_RECEIVE <= '0';
            FRESET       <= '0';
            Sop          <= '1';
            RESET_SCOUNT <= '0';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "001";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '1';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S4;
        end if;
when S4 =>
            SEND_RECEIVE <= '0';
            FRESET       <= '0';
            Sop          <= '1';
            RESET_SCOUNT <= '0';
```

```
                INC             <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION     <= "001";
                LOAD_CELL     <= '1';
                RESET_TRN     <= '1';
                START_TRN     <= '0';
                DONE          <= '0';
                NEXT_STATE    <=  S5;
when S5 =>
                SEND_RECEIVE <= '1';
                FRESET        <= '1';
                Sop           <= '1';
                RESET_SCOUNT <= '0';
                INC           <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION     <= "001";
                LOAD_CELL     <= '0';
                RESET_TRN     <= '1';
                START_TRN     <= '0';
                DONE          <= '0';
                NEXT_STATE    <=  S6;
when S6 =>
                SEND_RECEIVE <= '1';
                FRESET        <= '0';
                Sop           <= '1';
                RESET_SCOUNT <= '0';
                INC           <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION     <= "001";
                LOAD_CELL     <= '0';
                RESET_TRN     <= '1';
                START_TRN     <= '0';
                DONE          <= '0';
                NEXT_STATE    <=  S7;
```

```
when S7 =>
        if SDONE = '0' then
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <= S7;
        else
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '1';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <= S8;
        end if;
when S8 =>
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '1';
```

```
                LOAD_SCOUNT  <= '1';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S9;
when S9 =>
                SEND_RECEIVE <= '0';
                FRESET       <= '0';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S10;
when S10 =>
        if SCOUNT = "000" then
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '1';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S11;
```

```
        else
                SEND_RECEIVE <= '0';
                FRESET       <= '1';
                Sop          <= '1';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "001";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '1';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S2;
        end if;
when S11 =>
                SEND_RECEIVE <= '0';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '1';
                DONE         <= '0';
                NEXT_STATE   <=  S12;
when S12 =>
        if TRN_DONE = '0' then
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '0';
                INC          <= '0';
```

```
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "000";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '0';
            START_TRN    <= '1';
            DONE         <= '0';
            NEXT_STATE   <=  S12;
        else
            SEND_RECEIVE <= '1';
            FRESET       <= '0';
            Sop          <= '0';
            RESET_SCOUNT <= '0';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "000";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '0';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S13;
        end if;
when S13 =>
            SEND_RECEIVE <= '1';
            FRESET       <= '1';
            Sop          <= '0';
            RESET_SCOUNT <= '1';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "000";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '0';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S14;
```

```
when S14 =>
            SEND_RECEIVE <= '1';
            FRESET       <= '0';
            Sop          <= '0';
            RESET_SCOUNT <= '0';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "000";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '0';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S15;
when S15 =>
    if SDONE = '0' then
            SEND_RECEIVE <= '1';
            FRESET       <= '0';
            Sop          <= '0';
            RESET_SCOUNT <= '0';
            INC          <= '0';
            LOAD_SCOUNT  <= '0';
            OPERATION    <= "000";
            LOAD_CELL    <= '0';
            RESET_TRN    <= '0';
            START_TRN    <= '0';
            DONE         <= '0';
            NEXT_STATE   <=  S15;
    else
            SEND_RECEIVE <= '1';
            FRESET       <= '0';
            Sop          <= '0';
            RESET_SCOUNT <= '0';
            INC          <= '1';
            LOAD_SCOUNT  <= '0';
```

```
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <= S16;
        end if;
when S16 =>
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '0';
                INC          <= '1';
                LOAD_SCOUNT  <= '1';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <= S17;
when S17 =>
        if DELAY = '1' then
                SEND_RECEIVE <= '1';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <= S17;
```

```
else
        SEND_RECEIVE <= '1';
        FRESET       <= '0';
        Sop          <= '0';
        RESET_SCOUNT <= '0';
        INC          <= '0';
        LOAD_SCOUNT  <= '0';
        OPERATION    <= "000";
        LOAD_CELL    <= '0';
        RESET_TRN    <= '0';
        START_TRN    <= '0';
        DONE         <= '0';
        NEXT_STATE   <=  S18;
    end if;
when S18 =>
    if SCOUNT = "000" then
        SEND_RECEIVE <= '0';
        FRESET       <= '0';
        Sop          <= '0';
        RESET_SCOUNT <= '0';
        INC          <= '0';
        LOAD_SCOUNT  <= '0';
        OPERATION    <= "000";
        LOAD_CELL    <= '0';
        RESET_TRN    <= '0';
        START_TRN    <= '0';
        DONE         <= '0';
        NEXT_STATE   <=  S19;
    else
        SEND_RECEIVE <= '1';
        FRESET       <= '1';
        Sop          <= '0';
        RESET_SCOUNT <= '0';
        INC          <= '0';
```

```
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '0';
                DONE         <= '0';
                NEXT_STATE   <=  S14;
            end if;
        when S19 =>
                SEND_RECEIVE <= '0';
                FRESET       <= '0';
                Sop          <= '0';
                RESET_SCOUNT <= '0';
                INC          <= '0';
                LOAD_SCOUNT  <= '0';
                OPERATION    <= "000";
                LOAD_CELL    <= '0';
                RESET_TRN    <= '0';
                START_TRN    <= '0';
                DONE         <= '1';
                NEXT_STATE   <=  S1;
        end case;
    end process;
end ACH_STATE_MACHINE_IO;
```

5.17 โปรแกรมบรรยายพฤติกรรมของวงจรเลือกการทำงานของชิพ

```
library IEEE;
use IEEE.std_logic_1164.all;
entity IO_PIN is
        port(RxC,RxS,TxD,LOAD_FUNC,TRN_DONE:in STD_LOGIC;
```

```
        TxC,TxS,RxD,FUNC:out STD_LOGIC);
end IO_PIN;

architecture ACH_IO_PIN of IO_PIN is
        signal RxC_RxS:STD_LOGIC_VECTOR(1 downto 0);
        signal FUNC_IN,FUNC_DATA:STD_LOGIC;
begin
        RxC_RxS <= RxC & RxS;

        FUNC1:
        process(RxC_RxS,LOAD_FUNC,TRN_DONE)
        begin
                case RxC_RxS is
                when "01" =>
                        FUNC_DATA <= '1';
                when "10" =>
                        FUNC_DATA <= '0';
                when others =>
                        FUNC_DATA <= '-';
                end case;

                if LOAD_FUNC'EVENT and LOAD_FUNC = '1' then
                        if TRN_DONE = '0' then
                                FUNC_IN <= FUNC_DATA;
                        end if;
                end if;
        end process;

        IO:process(RxC,RxS,TxD,FUNC_IN,TRN_DONE)
        begin
```

```
            if TRN_DONE = '0' then
                RxD <= not (RxC xor RxS);
                TxC <= TxD;
                TxS <= '1';
            elsif FUNC_IN = '1' then
                RxD <= '1';
                TxC <= '1';
                TxS <= TxD;
            else
                RxD <= '1';
                TxC <= TxD;
                TxS <= '1';
            end if;
        end process;
        FUNC <= FUNC_IN;
end ACH_IO_PIN;
```

5.18 โปรแกรมบรรยายโครงสร้างของชิพ

```
library IEEE;
use IEEE.std_logic_1164.all;
entity CHIP is
        port(RESET,OSC,RxC,RxS:in STD_LOGIC;
            TxC,TxS,ERROR,DONE:out STD_LOGIC);
end CHIP;

architecture ACH_CHIP of CHIP is
        component GEN
```

```
        port(RESET,OSC:in STD_LOGIC; CLK:out STD_LOGIC);
end component;
component IO_PIN
        port(RxC,RxS,TxD,LOAD_FUNC,TRN_DONE:in STD_LOGIC;
            TxC,TxS,RxD,FUNC:out STD_LOGIC);
end component;
component MUX_ADDR
        port(OPERATION,SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
            ENABLE_CELL0,ENABLE_CELL1,ENABLE_CELL2,ENABLE_CELL3,
            ENABLE_CELL4,ENABLE_CELL5,ENABLE_CELL6,ENABLE_CELL7,
            ENABLE_CELL8,ENABLE_CELL9,ENABLE_CELL10,ENABLE_CELL11
            ,ENABLE_CELL12,ENABLE_CELL13,ENABLE_CELL14
            :out STD_LOGIC);
end component;
component MUX_IN
        port(SBUF_HI,SBUF_LOW:in STD_LOGIC_VECTOR(3 downto 0);
            OPERATION,SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
            TRN_3P_A0,TRN_3P_A1,TRN_3P_A2
            :in STD_LOGIC_VECTOR(3 downto 0);
            TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4
            :in STD_LOGIC_VECTOR(3 downto 0);
            DATA_OUT_CELL0,DATA_OUT_CELL1,DATA_OUT_CELL2,
            DATA_OUT_CELL3,DATA_OUT_CELL4,DATA_OUT_CELL5,
            DATA_OUT_CELL6,DATA_OUT_CELL7,DATA_OUT_CELL8,
            DATA_OUT_CELL9,DATA_OUT_CELL10,DATA_OUT_CELL11,
            DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14
            :in STD_LOGIC_VECTOR(3 downto 0);
            DATA_IN_CELL0,DATA_IN_CELL1,DATA_IN_CELL2,
            DATA_IN_CELL3,DATA_IN_CELL4,DATA_IN_CELL5,
            DATA_IN_CELL6,DATA_IN_CELL7,DATA_IN_CELL8,
```

```
            DATA_IN_CELL9,DATA_IN_CELL10,DATA_IN_CELL11,
            DATA_IN_CELL12,DATA_IN_CELL13,DATA_IN_CELL14
            :out STD_LOGIC_VECTOR(3 downto 0));
end component;
component OUT_MAP
        port(CELL_0,CELL_1,CELL_2,CELL_3,CELL_4,CELL_5,CELL_6,CELL_7,
            CELL_8,CELL_9,CELL_10,CELL_11,CELL_12,CELL_13,CELL_14
            :in STD_LOGIC_VECTOR(3 downto 0);
            Sop:in STD_LOGIC; Scount:in STD_LOGIC_VECTOR(2 downto 0);
            DATA:out STD_LOGIC_VECTOR(7 downto 0));
end component;
component REG_CELL
        port(DATA_IN:in STD_LOGIC_VECTOR(59 downto 0);
            LOAD:in STD_LOGIC;
            ENABLE:in STD_LOGIC_VECTOR(14 downto 0);
            DATA_OUT:out STD_LOGIC_VECTOR(59 downto 0));
end component;
component SBUF
        port(RESET,RxD,LOAD,DIR,CLOCK:in STD_LOGIC;
            DIN:in STD_LOGIC_VECTOR(7 downto 0);
            S_OUT,STOPBIT,ZERO:out STD_LOGIC;
            DOUT:out STD_LOGIC_VECTOR(7 downto 0));
end component;
component SCOUNT
        port(INC,LOAD,RESET:in STD_LOGIC;
            SCOUNT:out STD_LOGIC_VECTOR(2 downto 0));
end component;
component STA_IO
        port(RESET,SDONE,TRN_DONE,CLOCK,XCLK:in STD_LOGIC;
            SCOUNT:in STD_LOGIC_VECTOR(2 downto 0);
```

```
        SEND_RECEIVE,FRESET,Sop,LOAD_CELL,INC,RESET_SCOUNT,
        LOAD_SCOUNT,RESET_TRN,START_TRN,DONE:out STD_LOGIC;
        OPERATION:out STD_LOGIC_VECTOR(2 downto 0));
end component;
component STA_MACH
    port(RESET,START_TRN,CLK:in STD_LOGIC;
        OPERATION:out STD_LOGIC_VECTOR(2 downto 0);
        LOAD_CELL,DONE:out STD_LOGIC);
end component;
component STA_SBUF
    port(FRESET,SEND_RECEIVE,XCLK,RxD,BIT_0,BIT_10,SZERO
        :in STD_LOGIC;
        LOAD,SCLK,DIR,RESET_SBUF,SDONE,SERROR,LOAD_FUNC
        :out STD_LOGIC);
end component;
component TRN_3P
    port(A0_IN_3P,A1_IN_3P,A2_IN_3P:in STD_LOGIC_VECTOR(3 downto 0);
        FUNC:in STD_LOGIC;
        A0_OUT_3P,A1_OUT_3P,A2_OUT_3P
        :out STD_LOGIC_VECTOR(3 downto 0));
end component;
component TRN_5P
    port(A0_IN,A1_IN,A2_IN,A3_IN,A4_IN
        :in STD_LOGIC_VECTOR(3 downto 0);FUNC:in STD_LOGIC;
        A0_OUT,A1_OUT,A2_OUT,A3_OUT,A4_OUT
        :out STD_LOGIC_VECTOR(3 downto 0));
end component;
component XCLOCK
    port(CLOCK,RESET: in STD_LOGIC;
        XCLK: out STD_LOGIC);
```

```
end component;

signal CLOCK,XCLK,RxD,TxD,TRN_DONE,Sop,LOAD_CELL,
       LOAD_CELL_STA_IO,LOAD_CELL_STA_MACH,
       RESET_SBUF,LOAD_SBUF,DIR,SCLK,S_OUT,STOPBIT,ZERO,
       INC_SCOUNT,LOAD_SCOUNT,RESET_SCOUNT,SDONE,
       SEND_RECEIVE,FRESET,RESET_TRN,START_TRN,LOAD_FUNC,FUNC,
       ENABLE_CELL0,ENABLE_CELL1,ENABLE_CELL2,ENABLE_CELL3,
       ENABLE_CELL4,ENABLE_CELL5,ENABLE_CELL6,ENABLE_CELL7,
       ENABLE_CELL8,ENABLE_CELL9,ENABLE_CELL10,ENABLE_CELL11,
       ENABLE_CELL12,ENABLE_CELL13,ENABLE_CELL14:STD_LOGIC;

 signal SBUF_HI,SBUF_LOW,
      TRN_3P_A0,TRN_3P_A1,TRN_3P_A2,
      TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4,
      DATA_OUT_CELL0,DATA_OUT_CELL1,DATA_OUT_CELL2,
      DATA_OUT_CELL3,DATA_OUT_CELL4,DATA_OUT_CELL5,
      DATA_OUT_CELL6,DATA_OUT_CELL7,DATA_OUT_CELL8,
      DATA_OUT_CELL9,DATA_OUT_CELL10,DATA_OUT_CELL11,
      DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14,
      DATA_IN_CELL0,DATA_IN_CELL1,DATA_IN_CELL2,DATA_IN_CELL3,
      DATA_IN_CELL4,DATA_IN_CELL5,DATA_IN_CELL6,DATA_IN_CELL7,
      DATA_IN_CELL8,DATA_IN_CELL9,DATA_IN_CELL10,DATA_IN_CELL11,
      DATA_IN_CELL12,DATA_IN_CELL13,DATA_IN_CELL14
      :STD_LOGIC_VECTOR(3 downto 0);

signal OPERATION,OPERATION_STA_IO,OPERATION_STA_MACH,
       SCOUNT_BUF:STD_LOGIC_VECTOR(2 downto 0);
signal DATA_IN,DATA_OUT:STD_LOGIC_VECTOR(7 downto 0);
signal DATA_IN_BUF,DATA_OUT_BUF:STD_LOGIC_VECTOR(59 downto 0);
```

```
    signal ENABLE:STD_LOGIC_VECTOR(14 downto 0);

begin
    TxD <= DIR or S_OUT;
    OPERATION <= OPERATION_STA_IO or OPERATION_STA_MACH;
    LOAD_CELL <= LOAD_CELL_STA_IO or LOAD_CELL_STA_MACH;
    SBUF_HI  <= DATA_OUT(7 downto 4);
    SBUF_LOW <= DATA_OUT(3 downto 0);
  DATA_IN_BUF <= DATA_IN_CELL14 & DATA_IN_CELL13 &DATA_IN_CELL12
        &DATA_IN_CELL11 & DATA_IN_CELL10 & DATA_IN_CELL9 &
        DATA_IN_CELL8 & DATA_IN_CELL7 & DATA_IN_CELL6 &
        DATA_IN_CELL5 & DATA_IN_CELL4 & DATA_IN_CELL3 &
        DATA_IN_CELL2 & DATA_IN_CELL1 & DATA_IN_CELL0;
  ENABLE <= ENABLE_CELL0&ENABLE_CELL1&ENABLE_CELL2&
        ENABLE_CELL3&ENABLE_CELL4&ENABLE_CELL5&
        ENABLE_CELL6&ENABLE_CELL7&ENABLE_CELL8&
        ENABLE_CELL9&ENABLE_CELL10&ENABLE_CELL11&
        ENABLE_CELL12&ENABLE_CELL13&ENABLE_CELL14;
  DATA_OUT_CELL14 <= DATA_OUT_BUF(59 downto 56);
  DATA_OUT_CELL13 <= DATA_OUT_BUF(55 downto 52);
  DATA_OUT_CELL12 <= DATA_OUT_BUF(51 downto 48);
  DATA_OUT_CELL11 <= DATA_OUT_BUF(47 downto 44);
  DATA_OUT_CELL10 <= DATA_OUT_BUF(43 downto 40);
  DATA_OUT_CELL9 <= DATA_OUT_BUF(39 downto 36);
  DATA_OUT_CELL8 <= DATA_OUT_BUF(35 downto 32);
  DATA_OUT_CELL7 <= DATA_OUT_BUF(31 downto 28);
  DATA_OUT_CELL6 <= DATA_OUT_BUF(27 downto 24);
  DATA_OUT_CELL5 <= DATA_OUT_BUF(23 downto 20);
  DATA_OUT_CELL4 <= DATA_OUT_BUF(19 downto 16);
  DATA_OUT_CELL3 <= DATA_OUT_BUF(15 downto 12);
```

```
DATA_OUT_CELL2 <= DATA_OUT_BUF(11 downto 8);
DATA_OUT_CELL1 <= DATA_OUT_BUF(7 downto 4);
DATA_OUT_CELL0 <= DATA_OUT_BUF(3 downto 0);

GEN1:GEN port map (RESET,OSC,CLOCK);
XCLOCK1:XCLOCK port map (CLOCK,RESET,XCLK);
IO_PIN1:IO_PIN port map (RxC,RxS,TxD,LOAD_FUNC,TRN_DONE,
        TxC,TxS,RxD,FUNC);
MUX_ADDR1:MUX_ADDR port map(OPERATION,SCOUNT_BUF,
        ENABLE_CELL0,ENABLE_CELL1,ENABLE_CELL2,ENABLE_CELL3,
        ENABLE_CELL4,ENABLE_CELL5,ENABLE_CELL6,ENABLE_CELL7,
        ENABLE_CELL8,ENABLE_CELL9,ENABLE_CELL10,ENABLE_CELL11,
        ENABLE_CELL12,ENABLE_CELL13,ENABLE_CELL14);
MUX_IN1:MUX_IN port map (SBUF_HI,SBUF_LOW,OPERATION,SCOUNT_BUF,
        TRN_3P_A0,TRN_3P_A1,TRN_3P_A2,
        TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4,
        DATA_OUT_CELL0,DATA_OUT_CELL1,DATA_OUT_CELL2,
        DATA_OUT_CELL3,DATA_OUT_CELL4,DATA_OUT_CELL5,
        DATA_OUT_CELL6,DATA_OUT_CELL7,DATA_OUT_CELL8,
        DATA_OUT_CELL9,DATA_OUT_CELL10,DATA_OUT_CELL11,
        DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14,
        DATA_IN_CELL0,DATA_IN_CELL1,DATA_IN_CELL2,
        DATA_IN_CELL3,DATA_IN_CELL4,DATA_IN_CELL5,
        DATA_IN_CELL6,DATA_IN_CELL7,DATA_IN_CELL8,
        DATA_IN_CELL9,DATA_IN_CELL10,DATA_IN_CELL11,
        DATA_IN_CELL12,DATA_IN_CELL13,DATA_IN_CELL14);
OUT_MAP1:OUT_MAP port map (DATA_OUT_CELL0,DATA_OUT_CELL1,
        DATA_OUT_CELL2,DATA_OUT_CELL3,DATA_OUT_CELL4,
        DATA_OUT_CELL5,DATA_OUT_CELL6,DATA_OUT_CELL7,
        DATA_OUT_CELL8,DATA_OUT_CELL9,DATA_OUT_CELL10,
```

```
        DATA_OUT_CELL11,DATA_OUT_CELL12,DATA_OUT_CELL13,
        DATA_OUT_CELL14,Sop,SCOUNT_BUF,DATA_IN);
    REG_CEL1:REG_CELL port map (DATA_IN_BUF,LOAD_CELL,ENABLE,
        DATA_OUT_BUF);
    SBUF1:SBUF port map (RESET_SBUF,RxD,LOAD_SBUF,DIR,SCLK,DATA_IN,
        S_OUT,STOPBIT,ZERO,DATA_OUT);
    SCOUNT1:SCOUNT port map (INC_SCOUNT,LOAD_SCOUNT,RESET_SCOUNT,
        SCOUNT_BUF);
    STA_IO1:STA_IO port map (RESET,SDONE,TRN_DONE,CLOCK,XCLK,
        SCOUNT_BUF,SEND_RECEIVE,FRESET,Sop,LOAD_CELL_STA_IO,
        INC_SCOUNT,RESET_SCOUNT,LOAD_SCOUNT,
        RESET_TRN,START_TRN,DONE,OPERATION_STA_IO);
    STA_MACH1:STA_MACH port map (RESET_TRN,START_TRN,CLOCK,
        OPERATION_STA_MACH,LOAD_CELL_STA_MACH,TRN_DONE);
    STA_SBUF1:STA_SBUF port map (FRESET,SEND_RECEIVE,XCLK,RxD,S_OUT,
        STOPBIT,ZERO,LOAD_SBUF,SCLK,DIR,RESET_SBUF,SDONE,ERROR,
        LOAD_FUNC);
    TRN_3P1:TRN_3P port map (DATA_OUT_CELL0,DATA_OUT_CELL5,
        DATA_OUT_CELL10,
        FUNC,TRN_3P_A0,TRN_3P_A1,TRN_3P_A2);
    TRN_5P1:TRN_5P port map (DATA_OUT_CELL10,DATA_OUT_CELL11,
        DATA_OUT_CELL12,DATA_OUT_CELL13,DATA_OUT_CELL14,FUNC,
        TRN_5P_A0,TRN_5P_A1,TRN_5P_A2,TRN_5P_A3,TRN_5P_A4);

end ACH_CHIP;

configuration CONFIG_CHIP of CHIP is
    for ACH_CHIP
    for all:GEN use entity work.GEN(RTL);
    end for;
```

```
        for all:IO_PIN use entity work.IO_PIN(ACH_IO_PIN);
        end for;
        for all:MUX_ADDR use entity work.MUX_ADDR(ACH_MUX_CELL_ADDRESS);
        end for;
        for all:MUX_IN use entity work.MUX_IN(ACH_MUX_IN);
        end for;
        for all:OUT_MAP use entity work.OUT_MAP(ACH_OUT_MAP);
        end for;
        for all:REG_CELL use entity work.REG_CELL(ACH_REG);
        end for;
        for all:SBUF use entity work.SBUF(ACH_SBUF_REG);
        end for;
        for all:SCOUNT use entity work.SCOUNT(ACH_SCOUNT);
        end for;
        for all:STA_IO use entity work.STA_IO(ACH_STATE_MACHINE_IO);
        end for;
        for all:STA_MACH use entity work.STA_MACH(ACH_STATE_MACHINE);
        end for;
        for all:STA_SBUF use entity work.STA_SBUF(ACH_STATE_MACHINE_SBUF);
        end for;
        for all:TRN_3P use entity work.TRN_3P(ACH_TRN_3P);
        end for;
        for all:TRN_5P use entity work.TRN_5P(ACH_TRN_5P);
        end for;
        for all:XCLOCK use entity work.XCLOCK(ACH_XCLOCK);
        end for;
        end for;
end CONFIG_CHIP;
```

---

6. โปรแกรมภาษาปาสคาล

```
program TEST_PROJECT;
type CELL_TYPE = array[0..14] of BYTE;

var CELL,CELL_IN,CELL_3P,CELL_5P,CELL_OUT:CELL_TYPE;
    EN_DE,i,j:INTEGER;

const CON_0:BYTE=1; CON_1:BYTE=2; CON_2:BYTE=4; CON_3:BYTE=8;

procedure MUL(A,B:BYTE; var C:BYTE);
        var A_0,A_1,A_2,A_3,B_0,B_1,B_2,B_3:BYTE;
begin
    A_0 := (A and CON_0);
    A_1 := ((A and CON_1) shr 1);
    A_2 := ((A and CON_2) shr 2);
    A_3 := ((A and CON_3) shr 3);
    B_0 := (B and CON_0);
    B_1 := ((B and CON_1) shr 1);
    B_2 := ((B and CON_2) shr 2);
    B_3 := ((B and CON_3) shr 3);
    C := (((A_1 and B_3) xor (A_2 and B_2) xor (A_3 and B_1) xor (A_0 and B_0)) xor
        (((A_1 and B_3) xor (A_2 and B_2) xor (A_3 and B_1) xor (A_0 and B_1) xor
        (A_1 and B_0) xor (A_2 and B_3) xor (A_3 and B_2)) shl 1) xor
        (((A_2 and B_3) xor (A_3 and B_2) xor (A_0 and B_2) xor (A_1 and B_1) xor (A_2 and B_0)
        xor (A_3 and B_3)) shl 2) xor
        (((A_3 and B_3) xor (A_0 and B_3) xor (A_1 and B_2) xor (A_2 and B_1) xor
        (A_3 and B_0)) shl 3));
end;

procedure TRN_3P(A0,A1,A2:BYTE; var C0,C1,C2:BYTE);
        var S1,S2,M1,BELTA:BYTE;
begin
    if EN_DE = 1 then
      BELTA := 7
    else
```

```
        BELTA := 6;
    S1 := A1 xor A2;
    C0 := S1 xor A0;
    MUL(BELTA,S1,M1);
    writeln('belta = ',BELTA);
    writeln('s1 = ',S1);
    writeln('M = ',M1);
    readln;
    S2 := C0 xor M1;
    C1 := S2 xor A1;
    C2 := S2 xor A2;
end;

procedure TRN_5P(A0,A1,A2,A3,A4:BYTE; var C0,C1,C2,C3,C4:BYTE);
        var S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,M1,M2,M3,M4,M5,
            BELTA1,BELTA2,BELTA3,BELTA4:BYTE;
begin
    if EN_DE = 1 then
    begin
        BELTA1 := 13; BELTA2 := 4; BELTA3 := 3; BELTA4 := 7;
    end
    else
    begin
        BELTA1 := 11; BELTA2 := 5; BELTA3 := 2; BELTA4 := 7;
    end;
    S1 := A2 xor A3;
    S2 := A1 xor A4;
    S3 := A1 xor A3;
    S4 := A2 xor A4;
    S5 := S1 xor S2;
    C0 := S5 xor A0;
    MUL(BELTA1,S5,M1);
    MUL(BELTA2,S1,M2);
    MUL(BELTA3,S2,M3);
    MUL(BELTA4,S3,M4);
    MUL(BELTA4,S4,M5);
```

```
    S6 := C0 xor M1;
    S7 := S6 xor M2;
    S8 := S6 xor M3;
    S9 := M5 xor A2;
    S10 := M4 xor A1;
    S11 := M5 xor A4;
    S12 := M4 xor A3;
    C1 := S8 xor S9;
    C2 := S7 xor S10;
    C3 := S7 xor S11;
    C4 := S8 xor S12;
end;

begin
    write('Enter 1 with Encode, 0 with Decode: '); readln(EN_DE);

    write('Enter Data of Cell0: '); readln(CELL[0]);
    write('Enter Data of Cell1: '); readln(CELL[1]);
    write('Enter Data of Cell2: '); readln(CELL[2]);
    write('Enter Data of Cell3: '); readln(CELL[3]);
    write('Enter Data of Cell4: '); readln(CELL[4]);
    write('Enter Data of Cell5: '); readln(CELL[5]);
    write('Enter Data of Cell6: '); readln(CELL[6]);
    write('Enter Data of Cell7: '); readln(CELL[7]);
    write('Enter Data of Cell8: '); readln(CELL[8]);
    write('Enter Data of Cell9: '); readln(CELL[9]);
    write('Enter Data of Cell10: '); readln(CELL[10]);
    write('Enter Data of Cell11: '); readln(CELL[11]);
    write('Enter Data of Cell12: '); readln(CELL[12]);
    write('Enter Data of Cell13: '); readln(CELL[13]);
    write('Enter Data of Cell14: '); readln(CELL[14]);

    for i:=0 to 14 do
    begin
        CELL_IN[i] := CELL[i];
    end;
```

```
TRN_3P(CELL[0],CELL[5],CELL[10],CELL_3P[0],CELL_3P[5],CELL_3P[10]);
TRN_3P(CELL[3],CELL[8],CELL[13],CELL_3P[3],CELL_3P[8],CELL_3P[13]);
TRN_3P(CELL[6],CELL[11],CELL[1],CELL_3P[6],CELL_3P[11],CELL_3P[1]);
TRN_3P(CELL[9],CELL[14],CELL[4],CELL_3P[9],CELL_3P[14],CELL_3P[4]);
TRN_3P(CELL[12],CELL[2],CELL[7],CELL_3P[12],CELL_3P[2],CELL_3P[7]);

TRN_5P(CELL_3P[0],CELL_3P[3],CELL_3P[6],CELL_3P[9],CELL_3P[12],
        CELL_5P[0],CELL_5P[3],CELL_5P[6],CELL_5P[9],CELL_5P[12]);
TRN_5P(CELL_3P[5],CELL_3P[8],CELL_3P[11],CELL_3P[14],CELL_3P[2],
        CELL_5P[5],CELL_5P[8],CELL_5P[11],CELL_5P[14],CELL_5P[2]);
TRN_5P(CELL_3P[10],CELL_3P[13],CELL_3P[1],CELL_3P[4],CELL_3P[7],
        CELL_5P[10],CELL_5P[13],CELL_5P[1],CELL_5P[4],CELL_5P[7]);

CELL_OUT[0] := CELL_5P[0];
CELL_OUT[1] := CELL_5P[8];
CELL_OUT[2] := CELL_5P[1];
CELL_OUT[3] := CELL_5P[9];
CELL_OUT[4] := CELL_5P[2];
CELL_OUT[5] := CELL_5P[10];
CELL_OUT[6] := CELL_5P[3];
CELL_OUT[7] := CELL_5P[11];
CELL_OUT[8] := CELL_5P[4];
CELL_OUT[9] := CELL_5P[12];
CELL_OUT[10] := CELL_5P[5];
CELL_OUT[11] := CELL_5P[13];
CELL_OUT[12] := CELL_5P[6];
CELL_OUT[13] := CELL_5P[14];
CELL_OUT[14] := CELL_5P[7];

writeln('CELL_IN0 = ',CELL_IN[0],'     CELL_OUT0 = ',CELL_OUT[0]);
writeln('CELL_IN1 = ',CELL_IN[1],'     CELL_OUT1 = ',CELL_OUT[1]);
writeln('CELL_IN2 = ',CELL_IN[2],'     CELL_OUT2 = ',CELL_OUT[2]);
writeln('CELL_IN3 = ',CELL_IN[3],'     CELL_OUT3 = ',CELL_OUT[3]);
writeln('CELL_IN4 = ',CELL_IN[4],'     CELL_OUT4 = ',CELL_OUT[4]);
writeln('CELL_IN5 = ',CELL_IN[5],'     CELL_OUT5 = ',CELL_OUT[5]);
writeln('CELL_IN6 = ',CELL_IN[6],'     CELL_OUT6 = ',CELL_OUT[6]);
```

```
    writeln('CELL_IN7 = ',CELL_IN[7],'      CELL_OUT7 = ',CELL_OUT[7]);
    writeln('CELL_IN8 = ',CELL_IN[8].'      CELL_OUT8 = ',CELL_OUT[8]);
    writeln('CELL_IN9 = ',CELL_IN[9],'      CELL_OUT9 = ',CELL_OUT[9]);
    writeln('CELL_IN10 = ',CELL_IN[10],'    CELL_OUT10 = ',CELL_OUT[10]);
    writeln('CELL_IN11 = ',CELL_IN[11],'    CELL_OUT11 = ',CELL_OUT[11]);
    writeln('CELLIN_12 = ',CELL_IN[12],'    CELL_OUT12 = ',CELL_OUT[12]);
    writeln('CELLIN_13 = ',CELL_IN[13],'    CELL_OUT13 = ',CELL_OUT[13]);
    writeln('CELLIN_14 = ',CELL_IN[14],'    CELL_OUT14 = ',CELL_OUT[14]);
    readln;
end.
```

กิตติกรรมประกาศ

รายงานฉบับนี้สำเร็จเป็นรูปเล่มขึ้นมาได้นั้นด้วยความร่วมมือและการช่วยเหลือจากอาจารย์ทุกท่านและเพื่อนๆหลายคน ทางคณะผู้จัดทำรายงานฉบับนี้ขอขอบคุณทุกท่าน โดยเฉพาะอย่างยิ่ง ดร. สมศักดิ์ ชุมช่วย ผู้ซึ่งให้คำปรึกษาตลอดจนชี้แนะแนวทางในการแก้ไขปัญหาในด้านการออกแบบโปรแกรม ทฤษฎี ให้รายงานฉบับนี้ได้สำเร็จลุล่วงตามเป้าหมาย