การออกแบบและจำลองการสร้าง Trunk โดยใช้วิธี CDMA

Design and Implementation of CDMA Trunk Demonstrator

โดย

นายชาญวุฒิ อัศวศิริศิลป์

นายชินภัทร เศรษฐิน

นายปริญญา อู่ไพจิตร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การออกแบบและจำลองการสร้าง Trunk โดยใช้วิธี CDMA

Design and Implementation of CDMA Trunk Demonstrator

โดย

นายชาญวุฒิ อัศวศิริศิลป์ 44010115

นายชินภัทร เศรษฐิน 44010117

นายปริญญา อู่ไพจิตร 44010294

อาจารย์ที่ปรึกษา

รศ.ดร. กอบชัย เดชหาญ

อาจารย์ ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การออกแบบและจำลองการสร้าง Trunk โดยใช้วิธี CDMA**

Design and Implementation of CDMA Trunk Demonstrator

ผู้จัดทำ

1.นายชาญวุฒิ อัศวศิริศิลป์ 44010115

2.นายชินภัทร เศรษฐิน 44010117

3.นายปริญญา อู่ไพจิตร 44010294

_____ อาจารย์ที่ปรึกษา

(**รศ.ดร. กอบชัย เดชหาญ**)

_____ อาจารย์ที่ปรึกษา

(**อาจารย์ ศรวัฒน์ ชิวปรีชา**)

การออกแบบและจำลองการสร้าง Trunk
โดยใช้วิธี CDMA

Design and Implementation of CDMA Trunk Demonstrator

โดย นายชินภัทร เศรษฐิน 44010117
นายชาญวุฒิ อัศวศิริศิลป์ 44010115
นายปริญญา อู่ไพจิตร 44010294

อาจารย์ที่ปรึกษา รศ.ดร.กอบชัย เดชหาญ
อ.ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ มีจุดประสงค์เพื่อทำการทดลองการออกแบบ และจำลองการสร้าง Trunk โดยนำวิธีการเข้ารหัสแบบ CDMA มารองรับการใช้งานจากหลายผู้ใช้

Abstract

The aim of this project is to demonstrate the design and implementation process of digital trunk. Code division multiple access is issued to provide the multiple access function

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาของปริญญานิพนธ์

ในปัจจุบันโลกเราได้มีการพัฒนาทางด้านการสื่อสาร อย่างรวดเร็วซึ่งการสื่อสารนับเป็นปัจจัยที่สำคัญอย่างหนึ่งต่อการพัฒนาของประเทศทุกๆประเทศ โดยเมื่อประชากรเพิ่มขึ้นก็ย่อมต้องมีการติดต่อสื่อสารกันมากขึ้นไม่ว่าจะเป็นในทางด้านธุรกิจ การศึกษาและอีกหลายด้าน ตลอดจนปัจจุบันได้นำไปใช้ทางด้านการแพทย์แล้ว

โครงข่ายการสื่อสารแบบดิจิตอลนั้นเป็นโครงข่าย ที่ถูกพัฒนาขึ้นมาอย่างต่อเนื่องตลอดโดยเฉพาะการสื่อสารด้านโทรศัพท์ ซึ่งจะมีการเชื่อมโยงระหว่างชุมสายโทรศัพท์เข้าด้วยกัน ข้อดีของสัญญาณดิจิตอล (Digital Signal) คือ สามารถทำการสร้างสัญญาณใหม่ขึ้นมาทดแทนสัญญาณเดิมได้ โดยสถานีทวนสัญญาณ (Repeater) ในระยะ 20 ปีที่ผ่านมา โครงข่ายสื่อสารข้อมูลสาธารณะได้พัฒนาไปอย่างรวดเร็ว ด้วยเทคโนโลยีการสื่อสารข้อมูลดิจิตอล สัญญาณอนาลอกจะถูกแปลงให้เป็นสัญญาณดิจิตอลและสื่อสารผ่านโครงข่ายดิจิตอล (Integrated Digital Network) ก่อนที่จะแปลงกลับมาเป็นสัญญาณอนาลอกดังเดิม ดังนั้นการสื่อสารระบบดิจิตอลจึงเป็นเทคโนโลยีที่ถูกพัฒนาให้มีประสิทธิภาพมากขึ้นเรื่อย ๆ ด้วยคุณภาพที่ดีขึ้นและค่าใช้จ่ายที่ต่ำลงกว่าเดิม จึงเป็นอีกเหตุผลหนึ่งที่ทำให้มีการพัฒนารูปแบบใหม่ ๆ ในการให้บริการไม่ว่าจะเป็นเทคโนโลยีแบบไร้สาย หรือแม้กระทั่งโครงข่ายบริการสื่อสารร่วมระบบดิจิตอล (Integrated Services Digital Network)

วิทยานิพนธ์ฉบับนี้จะกล่าวถึงการออกแบบและทำการทดลองสร้าง Trunk ที่เป็นระบบ CDMA (Code Division Multiple Access) ขึ้นมาโดยจะนำความรู้จะระบบการสื่อสารแบบ CDMA มาประยุกต์ใช้ร่วมกับการมัลติเพล็กซ์ (Multiplex) แบบ TDM (Time Division Multiplexing) ซึ่งอาศัยโครงข่ายการสื่อสารทางด้านฮาร์ดแวร์เพื่อใช้ในการส่งข้อมูลในรูปแบบดิจิตอล เพื่อเป็นพื้นฐานในการสร้างโครงข่ายขนาดใหญ่ต่อไป

1.2 วัตถุประสงค์ของวิทยานิพนธ์

วัตถุประสงค์ในการทำวิทยานิพนธ์ฉบับนี้ ผู้เขียนมีวัตถุประสงค์หลักในการออกแบบ และ ทดสอบเพื่อให้แน่ใจว่า สามารถทำการนำโครงข่ายที่ออกแบบขึ้นมาใหม่ นี้มาเชื่อมต่อกับระบบภายในแบบดั้งเดิมได้ โดยจะมีต้นแบบการออกแบบมาจากระบบ E1 2.048Mbps ซึ่งมีสัญญาณข้อมูลข่าวสารเป็นแบบ 30 + 2 ช่องสัญญาณ โดยจะทำการวิเคราะห์และออกแบบโดยการนำเอาเทคนิค CDMA และ TDM เข้ามาใช้ในการแก้ไขปัญหา โดยกำหนดวัตถุประสงค์หลักไว้ดังต่อไปนี้

1.2.1 ศึกษาคุณสมบัติและหลักการทำงานของการมัลติเพล็กซ์แบบแบ่งเวลา (TDM) ในระบบ PCM-32 E1 digital carrier (2.048Mbps)

1.2.2 ศึกษาคุณสมบัติและหลักการทำงานของการเข้ารหัสแบบ CDMA เพื่อที่จะนำมาใช้แทนระบบการมัลติเพล็กซ์แบบแบ่งเวลา

1.2.3 ศึกษาคุณสมบัติและหลักการทำงานของการทำการเข้ารหัสช่องสัญญาณ (Channel coding) แบบต่าง ๆ และเพื่อที่นำมาทำการเลือกใช้ให้เหมาะสม เพื่อทำการเพิ่มประสิทธิภาพในการส่ง

1.2.4 ศึกษาและออกแบบเฟรม (Frame) ของ CDMA trunk ที่ทำการออกแบบขึ้นมาโดยวิเคราะห์จากปัจจัยต่าง ๆ โดยศึกษาจากระบบเดิมที่มีการใช้อยู่แล้ว

1.2.5 ศึกษาชนิดของรหัสออร์ธอโกนอล (Orthogonal code) ที่จะนำมาใช้ในระบบ

1.2.6 เปรียบเทียบประสิทธิภาพของระบบที่มีอยู่แล้วเทียบกับระบบใหม่

1.3 สมมติฐานของการศึกษา

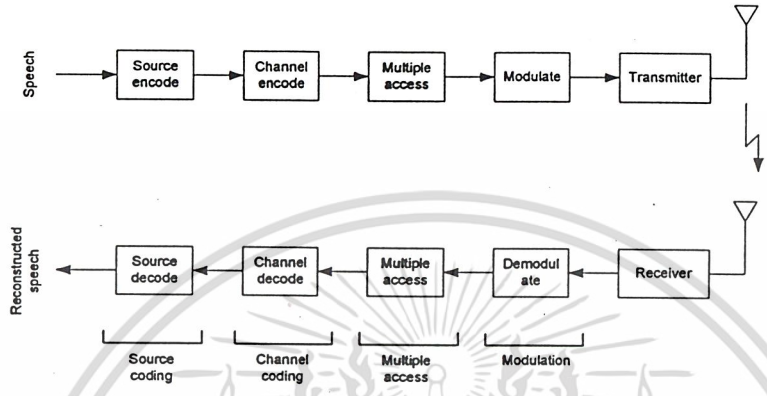
ในการออกแบบโครงข่ายการเชื่อมต่อ (Trunk) ในปริญญานิพนธ์นี้จะอาศัยหลักการและทฤษฎีของระบบการเชื่อต่อโครงข่ายเดิมในระบบ E1 เดิมที่ใช้ 2.048 Mbps โดยใช้เทคนิค TDM มาประยุกต์โดยการนำเทคนิค CDMA เข้ามาแทนที่ในการทำงาน โดยการทดลองออกแบบพิจารณาปัจจัยต่างๆที่มีผลต่อระบบโดยโปรแกรม MATLAB และ ทำเป็นอุปกรณ์โดยใช้ FPGA และ ภาษา VHDL เข้ามาช่วยสำหรับการออกแบบเป็นฮาร์ดแวร์

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการทดลอง

แนวความคิดที่ใช้ในการออกแบบและทดลองครั้งนี้ เกิดจากความต้องที่จะใช้งานโครงข่ายมีมากขึ้นทุกวัน ในการทดลองนี้จะทำการนำจุดเด่นของเทคนิค CDMA เข้ามาประยุกต์ใช้กับระบบโครงข่ายการเชื่อมต่อเพื่อที่จะทำการเพิ่มความปลอดภัยของระบบ และเพิ่มประสิทธิภาพ

บทที่ 2
ทฤษฎีและหลักการ

2.1 การสื่อสารแบบดิจิตอล



รูปที่ 2.1 ส่วนประกอบพื้นฐานโดยทั่วไปของระบบการสื่อสาร

จากรูปที่ 2.1 เป็นการแสดงบล็อกไดอะแกรมของระบบการสื่อสารแบบดิจิตอลชนิดหนึ่ง ในขั้นตอนแรกแหล่งกำเนิดข้อมูล เช่น เสียงของมนุษย์ จะถูกเปลี่ยนให้เป็นสัญญาณดิจิตอลด้วยตัวที่ทำหน้าที่เข้ารหัสแหล่งข้อมูล (Source encode function) แล้วนำผ่านตัวเข้ารหัสช่องสัญญาณ (Channel encode) เพื่อเข้ารหัสให้กับข้อมูลที่เป็นดิจิตอลที่ส่งมา โดยมีจุดมุ่งหมายเพื่อที่จะลดผลกระทบต่าง ๆ ที่อาจเกิดกับช่องสัญญาณ ต่อมานำข้อมูลที่ได้ไปจัดเรียงด้วยตัวที่ทำหน้าที่เข้าถึงแบบหลายทาง (Multiple access function) ทำให้ผู้ใช้มากกว่าหนึ่งรายก็จะสามารถใช้สเปกตรัมร่วมกันได้ ส่วนต่อมาคือส่วนที่ทำหน้าที่มอดูเลต (Modulated) ทำหน้าที่เปลี่ยนข้อมูลจากข้อมูลเบสแบนด์ (Baseband) ไปเป็นสัญญาณที่มีลักษณะอยู่ในช่วงที่สามารถส่งผ่านได้ (Bandpass (RF) waveform) ซึ่งจะถูกส่งโดยตัวส่ง (Transmitter) ที่ทางด้านรับ สัญญาณที่ส่งผ่านมาจะถูกรับด้วยตัวรับ (Receiver) ในขั้นแรกสัญญาณจะถูกดีมอดูเลต (Demodulated) จากสัญญาณ RF ไปเป็น สัญญาณเบสแบนด์ ส่งต่อไปยังส่วนที่ทำหน้าที่เข้าถึงแบบหลายทาง ทำการแยกแยะสัญญาณของผู้ใช้ที่ใช้สเปกตรัมร่วมกันแต่ละรายออกจากกัน หลังจากนั้นตัวที่ทำการถอดรหัสช่องสัญญาณ (Channel decode) และแก้ไขข้อผิดพลาดให้ถูกต้อง หลังจากนั้นตัวที่ทำหน้าที่ถอดรหัสแหล่งกำเนิด (Source decode) ทำหน้าที่แปลงสัญญาณเบสแบนด์กลับไปเป็นสัญญาณเสียงดังเดิม

โดยในรายงานนี้ เราจะทำการพูดถึงแต่ในส่วนที่เป็น การทำให้สามารถเข้าถึงแบบหลายทางได้โดยใช้เทคนิคสเปรดสเปกตรัมแบบไดเรคซีเควนซ์ (Direct sequence spread spectrum) เข้ามาช่วย

2.2 เทคนิคสเพรดสเปกตรัม (Spread Spectrum: SS)

ในระบบการสื่อสารสิ่งแรกที่มีความสัมพันธ์กับการทำงานของ ระบบการสื่อสารก็ คือประสิทธิภาพของแบนด์วิดท์ (Bandwidth) และกำลังงาน อย่างไรก็ตามในการประยุกต์ใช้งานก็ต้องคำนึงถึงความสามารถในการต่อต้านสัญญาณรบกวน (Noise) การขจัดการแทรกสอด ความสามารถในการใช้ช่องสัญญาณร่วมกัน (Multiple access) ความสามารถที่เครื่องรับเครื่องอื่นไม่สามารถรับสัญญาณได้ (Low probability of intercept) ซึ่งสิ่งต่าง ๆ เหล่านี้มีความสำคัญในการนำไปใช้ในทางการทหาร ระบบการสื่อสารที่ทำงานได้ดีตามคุณสมบัติที่กล่าวมาแล้วนั้น คือการใช้เทคนิคของสเพรดสเปรกตรัม เนื่องจากว่าแบนด์วิดท์ที่ใช้ในการส่งสัญญาณจะมีค่ามากกว่าแบนด์วิดท์ที่น้อยที่สุดที่ต้องการในการส่งสัญญาณข้อมูลข่าวสารมาก ๆ

เดิมทีในช่วงปลาย 1940s ในวงการทหารได้มีการนำเอาเทคนิคสเพรดสเปกตรัม เข้ามาใช้โดยที่เทคนิคสเพรดสเปกตรัม มีข้อเด่น คือ สามารถป้องกันสัญญาณรบกวนเพื่อที่ต้องการจะทำการ Jamming ได้ดี ซึ่งสัญญาณจะถูกส่งไปหลบอยู่ภายใน Background noise ซึ่งระบบนี้พึ่งเริ่มมีการนำมาใช้เพื่อการพาณิชย์ได้ไม่นานโดยที่เราสามารถจำแนกระบบสเพรดสเปกตรัมออกได้เป็น 3 ชนิดแบบพื้นฐานคือ

2.2.1 ไดเรคซีเควนซ์ (Direct Sequence Spread Spectrum: DSSS)

เป็นวิธีการส่งแบบที่ นำสัญญาณคลื่นพาห์ (Carrier) มาทำการมอดูเลตกับรหัสที่เป็นสัญญาณดิจิตอล ซึ่งความเร็วต่อบิตของรหัส (Bit rate) จะมีค่าสูงกว่าความเร็วข้อมูลข่าวสารมาก ๆ ซึ่งระบบนี้มีชื่อเรียกย่ออีกชื่อหนึ่งว่า Psudonoise (PN)

2.2.2 Frequency Hopping Spread Spectrum (FHSS)

ความถี่ของสัญญาณคลื่นพาห์จะถูกเปลี่ยนไปมาแบบ Discrete ตามลำดับของรหัสสัญญาณ และในบางครั้งนั้นรหัสที่ถูกทำการเลือกขึ้นมาใช้นั้นเพื่อที่จะป้องกันการไปรบกวนหรือถูกรบกวนจากระบบอื่นที่ไม่ใช่สเพรดสเปรกตรัม ในระบบ FHSS นั้นความถี่ของสัญญาณจะคงที่อยู่ในช่วงเวลาหนึ่งซึ่งเราเรียกค่าเวลานั้นว่า Time chip (T_c) ซึ่งการ Hopping ในระบบ FHSS สามารถทำได้อยู่ 2 วิธีคือ

1. ระบบ Fast-Hop ความเร็วที่ใช้ในการ Hop นั้นจะเกิดขึ้นด้วยความเร็วที่มีมากกว่าบิตเรตของข้อมูลข่าวสาร
2. ระบบ Slow-Hop ความถี่ที่ใช้ในการ Hop นั้นจะมีความเร็วที่น้อยกว่าบิตเรตของข้อมูลข่าวสาร

2.2.3 Time-Hopped (TH) system

โดยเวลาในการส่ง (Transmission time) จะถูกแบ่งออกเป็นช่วงเรียกว่า "เฟรม" ซึ่งแต่ละเฟรมจะถูกแบ่งออกเป็น Time slot ซึ่งจะมีเพียง Time slot เดียวเท่านั้นที่จะถูกทำการมอดูเลตกับ ข้อมูลข่าวสาร

ระบบที่จะสามารถกำหนดได้ว่าเป็นระบบสเพรดสเปรกตรัมจะมีลักษณะดังต่อไปนี้ คือ

1. แบนด์วิดท์ที่ใช้ในการส่งสัญญาณ จะต้องมากกว่าแบนวิดท์ที่น้อยที่สุดที่ต้องการในการส่งสัญญาณข้อมูลข่าวสารอยู่มาก ๆ
2. การกระจายแบนด์วิดท์ของสัญญาณ จะทำได้โดยการมอดูเลตสัญญาณข้อมูลข่าวสารสเพรดดิง (Spreading Signal) หรือเรียกอีกอย่างหนึ่งว่าสัญญาณรหัสและสัญญาณรหัสจะต้องไม่ขึ้นกับสัญญาณข้อมูลข่าวสาร
3. ที่เครื่องรับจะสามารถทำการดึงสัญญาณกลับคืนมาโดยการใช้คอร์รีเลท (Correlate) ของสัญญาณที่รับได้กับสัญญาณรหัสที่ใช้ในการกระจายข้อมูลข่าวสารทางเครื่องส่ง

การมอดูเลตแบบเอฟเอ็ม (Frequency Modulation: FM) พีซีเอ็ม (Pulse Code Modulation: PCM) เป็นการกระจายสัญญาณข้อมูลข่าวสารจริงแต่ก็ไม่เหมาะสมกับระบบสเพรดสเปกตรัม เพราะว่าไม่ได้สอดคล้องกับเงื่อนไขที่กล่าวมาข้างต้น

2.3 ทฤษฎีและหลักการของความจุช่องสัญญาณ

เทคโนโลยีของการสเพรดสเปกตรัมขึ้นอยู่กับทฤษฎีความจุของช่องสัญญาณ (Theory of channel capacity) ในช่องสัญญาณการส่งส่วนใด ๆ เราสามารถอธิบายได้ด้วยทฤษฎีของ C.E. Shannon

$$C = W \log_2(1 + S/N) \tag{2.1}$$

เมื่อ C = ความจุของช่องสัญญาณ (บิต/วินาที)

W = ช่วงความกว้างของแบนด์วิดท์ (เฮิรตซ์)

N = กำลังงานของสัญญาณรบกวน (วัตต์)

S = กำลังงานของสัญญาณข้อมูล (วัตต์)

สมการนี้แสดงใช้เห็นประสิทธิภาพของ ช่องสัญญาณที่สามารถส่งข้อมูลได้โดยไม่เกิดความผิดพลาดโดยกำหนดค่าอัตราส่วนของสัญญาณต่อสัญญาณรบกวน (Signal-to-noise ratio: SNR, S/N) ในช่องสัญญาณและขนาดของแบนด์วิดท์

เมื่อจัดรูปสมการใหม่จะได้ว่า

$$N/S = 1.44W/C \approx W/C \tag{2.2}$$

และ

$$W = NC/S \tag{2.3}$$

ซึ่งจากสมการที่ (2.1) นั้นจะพบว่าความสัมพันธ์ระหว่างความสามารถของความจุช่องสัญญาณ โดยปราศจากความผิดพลาด (Error free) โดยกำหนดค่า SNR และแบนด์วิดท์ของ Channel capacity จะมีค่าเพิ่มขึ้นเมื่อทำการเพิ่มค่าแบนด์วิดท์ กำลังในการส่ง หรือ เพิ่มทั้งคู่

<u>ตัวอย่าง</u> : เช่น ถ้าต้องการสื่อสารโดยที่สัญญาณรบกวนมากกว่าสัญญาณข้อมูล 100 เท่า และอัตราการส่งผ่านข้อมูลเท่ากับ 9.6 kbps แล้วจะต้องส่งข้อมูลด้วยแบนด์วิดท์เท่าใด

$$W = \frac{100 \times 9.67741 \times 10^3}{1.44} = 672.04236 \text{ kHz} \qquad (2.4)$$

จาก Shannon model ใช้กับ Channel ที่ base band แต่อย่างไรก็ตาม จากสมการที่ 2.1 จะสามารถใช้กับ Radio Frequency ได้เช่นเดียวกัน โดยที่กำหนดว่า ฟิวเตอร์ (Filter) ของความถี่ Intermediate Frequency (IF) เป็นแบบ ideal (Flat) band-pass responded โดยที่แบนด์วิดท์ที่มีค่าน้อยที่สุดคือ 2*BW โดยเราจะสมมติว่า ค่า channel noise Additive White Gaussian noise (AWGN) ซึ่งโดยส่วนมากแล้ว AWGN จะถูกนำไปใช้ในรูปแบบของ RF Channel

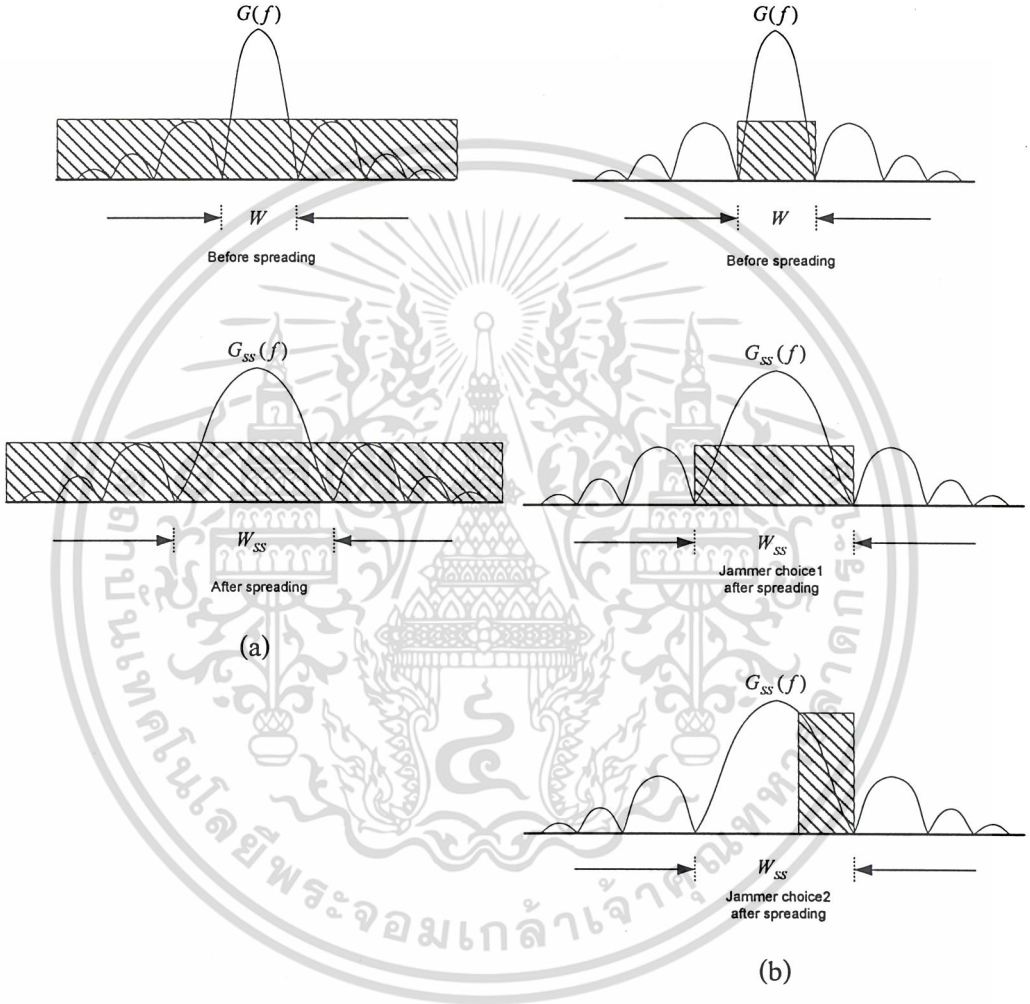
ซึ่งสมมติฐานนี้เป็นจริง เนื่องจากสัญญาณรบกวนจะถูกสร้างด้วยการทะลักของอิเลกตรอน (electron) แบบสุ่ม (Random Electric Fluctuations) โดยที่ทฤษฎี central limit ได้ให้ข้อสมมติฐานแก่เรา คือ ผลลัพธ์ของวงจรกรองความถี่แบบ IF จะมีลักษณะการกระจายตัวแบบ Gaussion และจะมีความถี่ที่เป็นอิสระต่อกันด้วย ซึ่งในทางปฏิบัติแล้วข้อจำกัดที่เกิดขึ้นกับระบบคือ สัญญาณรบกวนที่เกิดจากอุณหภูมิ (Termal noise) โดยปกติแล้วในระบบมือถือแบบอนาลอกจะมีค่า SNR ประมาณ 17 dB หรืออาจจะมากกว่านี้ แต่ว่าในระบบ CDMA นั้นระบบสามารถทำงานได้แม้ในสภาวะที่ SNR ต่ำกว่า

2.4 ประโยชน์ของระบบสเพรดสเปรกตรัม (Interference Suppression Benefits)

รูปแบบสมการคณิตศาสตร์ที่ได้อธิบายสัญญาณรบกวนขาว (White Gaussian Noise) ว่ามีการกระจายกำลังงานที่ไม่จำกัดอย่างสม่ำเสมอตลอดทุกความถี่และการสื่อสารที่มีประสิทธิ ภาพอาจจะเกิดขึ้นได้โดยการแทรกสอดของสัญญาณรบกวนที่มีกำลังงานไม่จำกัด เพราะสัญญาณรบกวนที่มีกำลังงานจำกัดเท่านั้นจึงจะสามารถแทรกสอดกับสัญญาณสเพรดสเปรกตรัมได้ สำหรับสัญญาณที่มีแบนด์วิดท์แคบ ๆ สัญญาณรบกวนในแบนด์วิดท์ของสัญญาณสามารถลดลงได้ แนวความคิดของการต่อต้านสัญญาณรบกวนในระบบสเพรดสเปกตรัมเป็นดังนี้คือ พิจารณาสัญญาณที่ใช้ส่ง ภายในจะมีกลุ่มเล็ก ๆ ของสัญญาณข้อมูลข่าวสารที่ใช้อยู่ ณ ที่เวลาใด ๆ เราสมมติว่าสัญญาณรบกวนไม่สามารถที่จะทราบกลุ่มของสัญญาณที่ใช้งานอยู่ในขณะนั้น สัญญาณที่มีแบนด์วิดท์ (W) และมีช่วงเวลา (T) กลุ่มของสัญญาณสามารถประมาณได้ว่ามีแบนด์วิดท์เป็น $2WT$ โดยให้การผิดพลาดของระบบเป็นฟังก์ชันของ N_0 เท่านั้น การต่อต้านสัญญาณรบกวนขาวโดยการกระจายแบนด์วิดท์ที่มากกว่า $2WT$ ไม่ได้ทำให้ระบบมีประสิทธิภาพที่ดีขึ้น อย่างไรก็ตามสัญญาณรบกวน

โดยสัญญาณรบกวนที่มีกำลังงานจำกัดและความไม่แน่นอนของกลุ่มสัญญาณ ซึ่งจะทำให้ทางเลือกของสัญญาณรบกวนถูกกำจัดดังต่อไปนี้

1. สัญญาณรบกวนจะแทรกตัวเข้าไปในสัญญาณสเพรดสเปกตรัมทั้งหมดของระบบด้วยค่าของกำลังงานที่เท่ากัน ซึ่งจะทำให้แต่ละกลุ่มสัญญาณมีค่ากำลังงานที่น้อยลง
2. สัญญาณรบกวนจะแทรกตัวเข้าไปในสัญญาณสเพรดสเปกตรัมได้เพียงเล็กน้อย



รูปที่ 2.2 การแสดงการแผ่กระจายของสเปกตรัมโดย
(a) คือการกระจายเสปกตรัมของสัญญาณที่มีสัญญาณรบกวนขาว
(b) คือการกระจายเสปกตรัมของสัญญาณรบกวนที่แทรกสอดเข้ามา

จากรูปที่ 2.2 เป็นการเปรียบเทียบการกระจายของสเปกตรัมที่แสดงด้วยสัญญาณรบกวนขาว และสัญญาณรบกวนที่แทรกสอดเข้ามา โดยกำหนดให้ความหนาแน่นกำลังงานเชิงสเปกตรัม (Power Spectrum Density: PSD) ของสัญญาณก่อนกระจายเป็น $G(f)$ และความหนาแน่นกำลังงานเชิงสเปกตรัมหลังการกระจายแล้วเป็น $G_{ss}(f)$ ซึ่งจะพิจารณาในแกนความถี่ ในรูปที่ 2.2(a) ชี้ให้เห็นว่าความหนาแน่นกำลังเชิง

สเปกตรัมแถบข้างเดียวของสัญญาณรบกวนขาว (N_0) ซึ่งเป็นผลของการกระจายแบนด์วิดท์ของสัญญาณจาก W ไปเป็น W_{ss} นั้นไม่มีการเปลี่ยนแปลง โดยที่ค่าของกำลังงานเฉลี่ยของสัญญาณรบกวนขาวจะมีค่าไม่จำกัด ดังนั้นการกระจายของสัญญาณไม่ได้ทำให้ประสิทธิภาพดีขึ้น ในรูปที่ 2.2(b) แสดงให้เห็นถึงรูปของสัญญาณรบกวนที่เครื่องรับ (ที่มีค่าจำกัด) J และมีความหนาแน่นกำลังเชิงสเปกตรัม $J = J / W$ โดยที่ W เป็นแบนด์วิดท์ที่ไม่ได้ถูกกระจายและถูกรบกวน เมื่อแบนด์วิดท์ของสัญญาณถูกกระจายมีความเป็นไปได้ที่สัญญาณรบกวนจะทำให้เกิด 2 กรณีดังต่อไปนี้

กรณีที่ 1 จะมีผลทำให้สเปกตรัมกำลังงาน (Power spectrum) ของสัญญาณรบกวน J_0 ถูกทำให้ลดลง W / W_{ss} ซึ่งผลของความหนาแน่นกำลังเชิงสเปกตรัม จะเป็น $J_0 = J / W_{ss}$ ซึ่งเรียกว่า Broadband Jammer Noise Spectrum Density

กรณีที่ 2 จะมีผลทำให้จำนวนของกลุ่มสัญญาณที่ถูกรบกวนนั้นมีค่าน้อยลง อย่างไรก็ตามสัญญาณรบกวนที่สามารถที่จะเพิ่มความหนาแน่นจาก J_0 เป็น J_0 / ρ โดยที่ $0 \le \rho \le 1$ โดยที่ ρ เป็นสัดส่วนโดยตรงกับทั้งแบนด์วิดท์ของสัญญาณข้อมูลที่กระจายอยู่และ แบนด์วิดท์ที่ถูกสัญญาณรบกวนเข้าแทรกสอดเข้ามา

สัญญาณรบกวนอาจจะมิได้เกิดขึ้นจากการกระทำที่ตั้งใจเท่านั้น บางครั้งสัญญาณรบกวนอาจเกิดจากปรากฏการณ์ทางธรรมชาติ และบางครั้งเกิดจากการแทรกสอดภายในตัวเอง (Internal Interferearance Symbol) ซึ่งเกิดจากคลื่นเดินทางมาจากหลาย ๆ เส้นทาง

2.5 สรุปข้อดีของการเทคนิคสเพรดสเปกตรัม

2.5.1 การเลือกที่อยู่ (Selective address) ผู้ใช้ทุกรายมีรหัสสุ่มเทียม (Pseudo Random Code) เพื่อที่จะทำการดีสเพรด (Despread) สัญญาณ ดังนั้นกลุ่มผู้ใช้ประเภทเดี่ยว หรือ กลุ่มอาจถูกกำหนดที่อยู่ด้วยรหัสเดียวกันในขณะที่ผู้ใช้รายอื่นถูกกำหนดด้วยรหัสที่แตกต่างกันออกไป

2.5.2 การเข้าถึงแบบหลายทางแบบแยกความแตกต่างทางรหัส (Code Division Multiple Access) เมื่อใช้รหัสสุ่มเทียมได้ถูกต้องจะทำให้ ค่าครอสคอรีเลชั่นที่ต่ำ (Low Cross-correlation) เพื่อที่จะให้มีการรบกวนหรือแทรกสอดเล็กน้อยระหว่างผู้ใช้ และ ค่าออโต้คอร์รีเลชั่นที่สูง (High auto-correlation) เพื่อที่จะเป็การง่ายต่อการทำให้เกิดซิงโครไนเซชั่นกัน (Synchronization)ได้ง่ายขึ้น ผู้ใช้หลาย ๆ รายสามารถใช้ความถี่เดียวกันโดยไม่เกิดการรบกวนกันกับผู้ใช้งานรายอื่น ๆ

2.5.3 ความหนาแน่นของสเปกตรัมกำลังงานต่ำ (Low Power Spectrum Density: PSD) กำลังงานรวมของสัญญาณข้อมูลถูกขยายให้แบนด์วิดท์กว้างขึ้น ซึ่งจะทำให้กำลังงานที่ส่งออกไปในบริเวณที่แบนด์วิดท์แคบจะต่ำ นี่เป็นลักษณะที่สำคัญของการซ่อนสัญญาณ (Signal hiding) และ ลดการรบกวนที่อาจเกิดกับผู้ใช้รายอื่น ๆ

2.5.4 การป้องกันข้อมูล (Message Protection) ระบบจะมีความสามารถในการป้องกันอยู่ในระดับที่ค่อนข้างสูงสำหรับการสื่อสาร เพราะสัญญาณสเพรดสเปกตรัม (Spread spectrum signal)ไม่สามารถดีสเพรดได้ ถ้าหากไม่รู้ถึงรหัสที่ใช้ในการทำการสเพรดสัญญาณและรหัสที่ใช้ต้องมีการซิงโครไนซ์ที่แม่นยำเพื่อที่จะสามารถกู้ข้อมูลเดิมกลับมาได้

2.5.5 ความสามารถในการบอกระยะห่างได้อย่างละเอียด (High Resolution Ranging) โดยที่อุปกรณ์บอกตำแหน่งที่ความละเอียดสูงและนำเทคนิคสเพรดสเปกตรัมมาประยุกต์ใช้ เช่น GPS โดยที่ความเอียดของการวัดจะขึ้นอยู่กับอัตราการเข้ารหัส และความยาวของรหัสที่ใช้

2.5.6 การรบกวน (Interference Rejection) ในระบบการสื่อสารทางทหาร ทางด้านฝ่ายตรงข้ามจะเป็นไปได้ยากที่จะพยายามทำการรบกวนการส่งสัญญาณ ถ้าสัญญาณรบกวนเป็นสัญญาณคลื่นที่คงที่ (Constant Wave: CW) แล้วการดีสเพรดที่ด้านรับอาจจะมีผลกระทบจากสัญญาณรบกวนบนสัญญาณข้อมูลข่าวสารที่ถูกเพรดซึ่งจะมีผลเพียงเล็กน้อยในการตัดสินสัญญาณข้อมูลข่าวสารที่ทำการดีสเพรดออกมา

2.6 รูปแบบของสเพรดสเปกตรัมที่กำจัดการแทรกสอด (Model for Spread-Spectrum Interference Rejection)

ระบบสเพรดสเปกตรัมที่มีการกำจัดการแทรกสอด ที่ตัวมอดูเลต (Modulator) สัญญาณข้อมูลข่าวสาร $x(t)$ ที่มีอัตราของ bit rate = R bit/sec ถูกคูณด้วยสัญญาณรหัส $g(t)$ ที่มีอัตราเร็วของสัญญาณรหัส R_p chip/sec สมมติว่าแบนด์วิดท์ของการส่งสำหรับ $x(t)$ และ $g(t)$ เป็น R Hz และ R_p Hz ตามลำดับ การคูณในทางเวลาสามารถเปลี่ยนเป็น ในทางความถี่ดังนี้

$$x(t)g(t) \leftrightarrow x(\omega)g(\omega) \tag{2.5}$$

ดังนั้นถ้าสัญญาณมีแบนด์วิดท์แคบ ๆ เมื่อเทียบกับสัญญาณที่ถูกกระจายแล้ว ผลของการคูณสัญญาณ $x(t)g(t)$ เราสามารถจะประมาณแบนด์วิดท์ว่าเท่ากับแบนด์วิดท์ของสัญญาณที่กระจายแล้วและที่ตัวดีมอดูเลต (Demodulator) สัญญาณที่รับได้จะถูกคูณด้วยแบบจำลองของสัญญาณรหัส ที่มีการซิงโครไนซ์ ซึ่งจะทำให้มีการรวมสัญญาณที่กระจายอยู่วงจรกรองความถี่ที่มีแบนด์วิดท์ R ก็จะกำจัดความถี่ที่สูงกว่า R ออกไป ถ้ามีสัญญาณที่ไม่ต้องการปรากฏที่เครื่องรับการคูณด้วย $g(t)$ จะเป็นการกระจายสัญญาณที่ไม่ต้องการออกไป และในทำนองเดียวกันการคูณด้วย $g(t)$ ที่เครื่องส่งก็เป็นการกระจายสัญญาณข้อมูลข่าวสารเช่นกัน พิจารณาผลของสัญญาณรบกวนที่พยายามแทรกตัวเข้ามาอยู่ในแบนด์วิดท์ของสัญญาณข้อมูลข่าวสาร การคูณสัญญาณที่ได้รับกับสัญญาณรหัสจะทำให้สัญญาณที่รบกวนถูกกระจาย ไปที่แบนด์วิดท์ของสัญญาณที่ถูกกระจาย

สาระสำคัญของความสามารถในการกำจัดการแทรกสอดของสัญญาณอื่นๆ มีดังนี้

1. การคูณโดยสัญญาณรหัสครั้งแรก จะเป็นการกระจายแบนด์วิดท์ของสัญญาณ
2. การคูณโดยสัญญาณรหัสครั้งที่สองและตามด้วยวงจรกรองความถี่ จะทำให้ได้สัญญาณ ข่าวสารกลับมา
3. สัญญาณข่าวสารจะได้โดยการคูณครั้งที่สอง แต่สัญญาณแทรกสอดจะได้โดยการคูณครั้งแรก

2.7 การเข้าถึงแบบหลายทิศทาง (Multiple Access Using Spread Spectrum)

วิธีดั้งเดิมของการกระจายสัญญาณในทางเวลา (เช่น Time Division Multiple Access: TDMA) หรือในทางความถี่ (เช่น Frequency Division Multiple Access: FDMA) เป็นวิธีที่ใช้กันโดยทั่วไปเพื่อให้แน่ใจได้ว่าสัญญาณที่ได้นั้นมีลักษณะเป็นออร์ธอโกนอล (Orthogonal) และ ไม่รบกวนกัน (Noninterfering) เทคนิค CDMA หรือ Code Division Multiple Access อาศัยการเข้ารหัสเป็นกุญแจสำคัญในด้านของผู้ใช้ระบบมือถือ อาจจะมองประโยชน์ของการเข้ารหัสเป็นเรื่องของความปลอดภัย การป้องกันการลักลอบจูน เป็นต้น แต่ในความเป็นจริง การใช้รหัสนี้ให้ประโยชน์มากมายหลายประการ มิใช่แค่เรื่องความปลอดภัยอย่างเดียว หากวิเคราะห์ลึกลงไปในทางเทคนิคจะพบสิ่งที่น่าสนใจหลายประการที่ทำให้ CDMA เป็นเทคโนโลยีที่มีอนาคตและเป็นพื้นฐานสำหรับมือถือในยุค 3G ดังนั้นในช่วงต่อไปจะขอนำเอารหัสที่ใช้ในระบบ CDMA มาแนะนำให้ได้ทราบกัน โดยขออ้างอิงจากมาตรฐาน IS-95A ซึ่งเป็นมาตรฐานที่มีการใช้ในงานอยู่จริงในปัจจุบัน ในระบบ CDMA นั้น ผู้ใช้งานที่แตกต่างกันเข้าครอบครองช่วงความถี่เดียวกันในช่วงเวลาเดียวกัน แต่สามารถแยกแยะผู้ใช้แต่ละรายโดยการใช้รหัสที่มีลักษณะเป็นออร์ธอโกนอลระบุไปที่ผู้ใช้แต่ละราย ค่าจริงของรหัส 2 ชุด x และ y จะเรียกว่าเป็นออร์ธอโกนอลกันได้ถ้าค่าครอสคอรีเลชั่น (Cross correlation) $R_{xy}(0)$ ของรหัสชุด x และชุด y เท่ากับ ศูนย์บนช่วงคาบเวลา T จะได้ว่า

$$R_{xy}(0) = \int_0^T x(t)y(t)dt \tag{2.6}$$

ในเชิงของเวลาที่ไม่ต่อเนื่องรหัส x และ y เป็นออร์ธอโกนอลกัน ถ้าค่าที่ได้จากการทำครอสโปรดัก $R_{xy}(0)$ มีค่าเท่ากับ ศูนย์ การทำ ครอสโปรดัก อธิบายได้ด้วยสมการข้างล่างนี้ คือ

$$R_{xy}(0) = x^T y = \sum_{i=1}^{I} x_i y_i \tag{2.7}$$

โดยที่

$$x^T = [x_1 \;\; x_2 \;\; \ldots \; x_I] \tag{2.8}$$

$$y^T = [y_1 \;\; y_2 \;\; \ldots \; y_I] \tag{2.9}$$

<u>หมายเหตุ</u> : ตัวอักษร T แสดงถึง การสับเปลี่ยนตำแหน่งกันระหว่างแถวกับหลัก (T: Transpose)

ตัวอย่างเช่น รหัส x และ y ทั้ง 2 ชุดข้างล่างนี้เป็น ออร์ธอโกนอล กัน

$$x^T = [\,-1\ \ -1\ \ 1\ \ 1\,] \tag{2.10}$$

$$y^T = [\,-1\ \ 1\ \ 1\ \ -1\,] \tag{2.11}$$

เพราะ ค่าที่ได้จากการทำครอสคอรีเลชั่นระหว่างรหัสชุด x กับชุด y มีค่าเท่ากับ ศูนย์ โดยมีวิธีทำดังนี้

$$R_{xy}(0) = x^T y = (-1)(-1) + (-1)(1) + (1)(1) + (1)(-1) = 0 \tag{2.12}$$

ข้อบังคับของรหัสที่จะนำมาใช้ในระบบการเข้าถึงข้อมูลแบบหลายทาง เราต้องพิจารณาเพิ่มอีก 2 คุณสมบัติ การที่จะบวกค่าต่างตามสมการที่ (2.7) แล้วให้ได้ผลลัพธ์ที่มีค่าเท่ากับ ศูนย์ ตามคุณสมบัติของการทำครอสคอรีเลชั่นนั้น รหัสแต่ละชุดในกลุ่มของรหัสที่เป็นออร์ธอโกนอลกัน จะต้องมีจำนวนสมาชิกที่เป็น 1 เท่ากับจำนวนสมาชิกที่เป็น -1 คุณสมบัติข้อที่สอง คือ การเข้ารหัสที่มีรูปแบบเฉพาะเจาะจงในลักษณะการสุ่มเทียมแบบธรรมชาติ (Pseudorandom nature) คุณสมบัติข้อที่สามคือ การทำดอทโปรดัก (หรือการคูณกัน) ในแต่ละค่าของรหัสแต่ละชุดนั้น ต้องให้ผลลัพธ์เท่ากับ 1 ตามที่ได้กล่าวแล้วข้อบังคับต่าง ๆ เหล่านี้มีผลต่อความยาวของรหัส และ การทำดอทโปรดักอธิบายได้ด้วยค่าที่ได้จากการคูณชุดของรหัสเข้ากับรหัสของตัวมันเองและ การรวมค่าของแต่เทอมเข้าด้วยกัน นั่นก็คือการทำดอทโปรดักของรหัส x คือ

$$R_{xx}(0) = x^T x = \sum_{i=1}^{I} x_i x_i \tag{2.13}$$

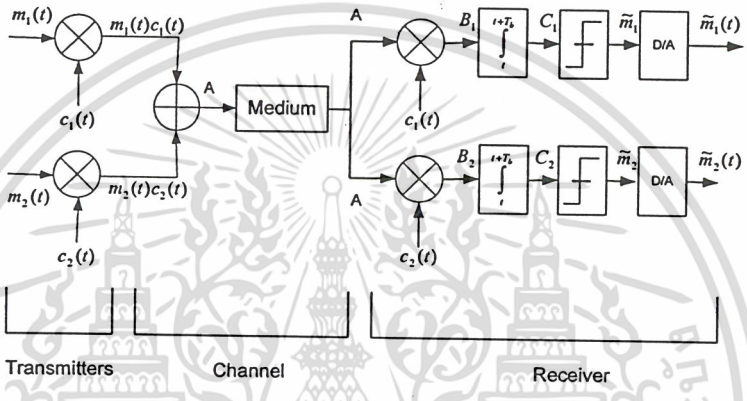
รหัสที่เป็นออร์ธอโกนอลกัน 2 ชุด ในตัวอย่างก่อนหน้านี้ เป็นการแก้ปัญหาให้กับเงื่อนไขของคุณสมบัติข้อที่สอง และ ข้อที่สาม รหัสทั้งชุด x และชุด y มีจำนวนของ 1 เท่ากับจำนวน -1 แล้วค่าขนาดของการดอทโปรดักที่ได้คือ

$$(x^T x)/4 = (-1)(-1) + (-1)(-1) + (1)(1) + (1)(1) = 4/4 = 1$$
$$(y^T y)/4 = (-1)(-1) + (1)(1) + (1)(1) + (-1)(-1) = 4/4 = 1$$

หมายเหตุ : อับดับของแต่ละรหัสคือ 4

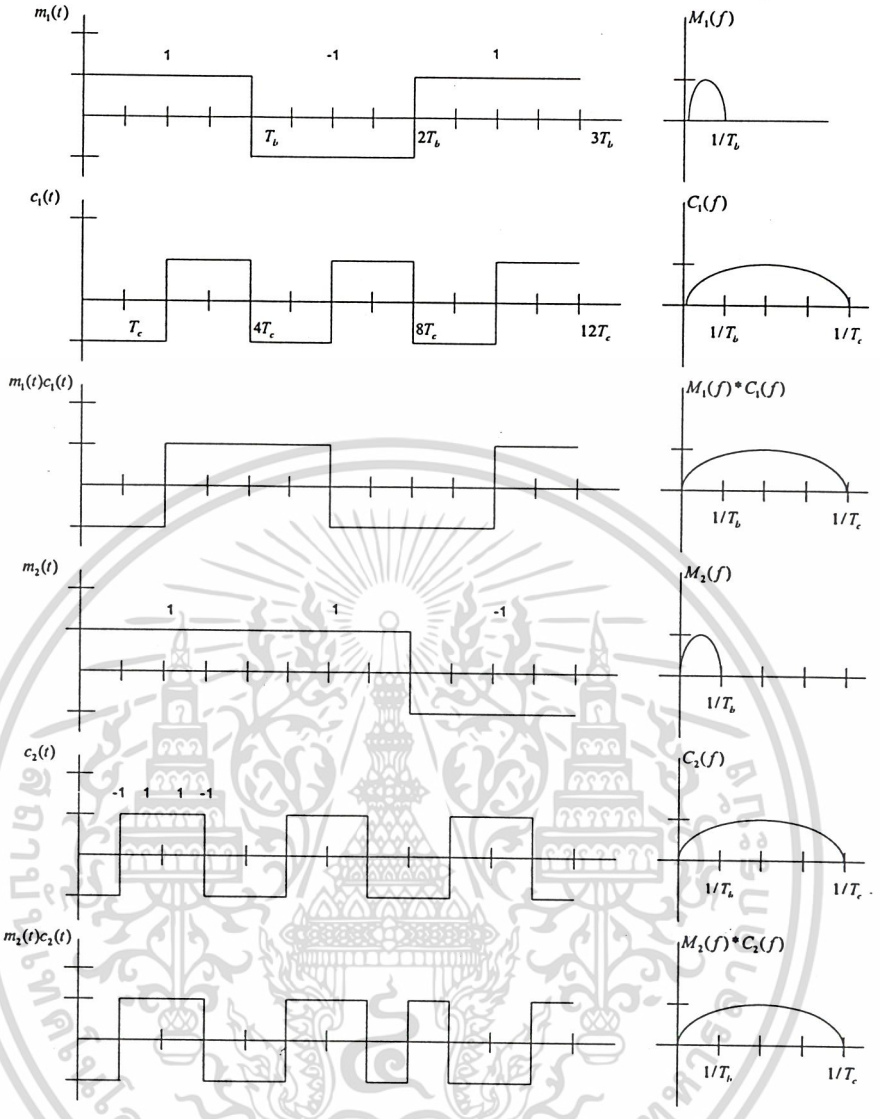
ดังนั้นเราสามารถสรุปคุณสมบัติของกลุ่มรหัสที่เป็น ออร์ธอโกนอลกัน เพื่อที่จะใช้ในระบบ DS-SS multiple access

1. ผลลัพธ์ที่ได้จากการทำครอสคอรีเลชั่น ควรที่จะเท่ากับ 0 หรือ มีค่าน้อยมาก ๆ
2. ในรหัสแต่ละชุดควรมีจำนวนของ 1 เท่ากับจำนวนของ -1 หรือ มีจำนวนของ 1 แตกต่างจากจำนวนของ -1 เพียงตัวเดียว
3. ผลลัพธ์ที่ได้จากการทำดอทโปรดักในรหัสแต่ละชุด ควรมีค่าเท่ากับ 1



รูปที่ 2.3 แสดงบล็อกไดอะแกรมของระบบ DS-SS multiple access ขั้นพื้นฐาน

จากรูปที่ 2.3 แสดงตัวอย่างโครงสร้างขั้นพื้นฐานของระบบ DS-SS multiple access ที่นิยมใช้กับการสื่อสารระบบดิจิตอล (Digital communication) จากรูปผู้ใช้สองคนส่งสัญญาณไปพร้อมกันด้วยข้อมูลที่ต่างกันสองชุด $m_1(t)$ และ $m_2(t)$ ในย่านความถี่เดียวกันที่เวลาเดียวกัน ผู้ใช้ทั้งสองคนถูกแบ่งแยกออกจากกันด้วยการคูณกับรหัสที่ต่างกัน $c_1(t)$ และ $c_2(t)$ ซึ่งเป็นออร์ธอโกนอลกัน เช่นเดียวกับรหัส x และ y ในเทอมของเวลาที่ต่อเนื่องซึ่งได้เคยกล่าวถึงมาแล้ว ข้อมูล $m_1(t)$ ถูกคูณด้วยรหัส $c_1(t)$ และข้อมูล $m_2(t)$ ถูกคูณด้วยรหัส $c_2(t)$ แล้วผลที่ได้ถูกนำมารวมด้วยกันที่ตัวบวกแล้วจึงถูกส่งผ่านไปตามช่องสัญญาณ ในกรณีนี้เป็นการสมมติว่ามีการซิงโคไนเซชั่นที่สมบูรณ์แบบของรหัสที่ด้านรับ และไม่สนใจความผิดพลาดเกิดขึ้นในช่องสัญญาณ สัญญาณที่กู้คืนกลับมา $\tilde{m}_1(t)$ และ $\tilde{m}_2(t)$ จะตรงกับข้อมูลต้นทาง $m_1(t)$ และ $m_2(t)$ ที่ส่งมา ในตัวอย่างนี้จะสนใจการส่งข้อมูลสองชุดที่แตกต่างกัน โดยสมมติให้ $m_1(t)$ คือ (+1, -1, +1) และ $m_2(t)$ คือ (+1, +1, -1)



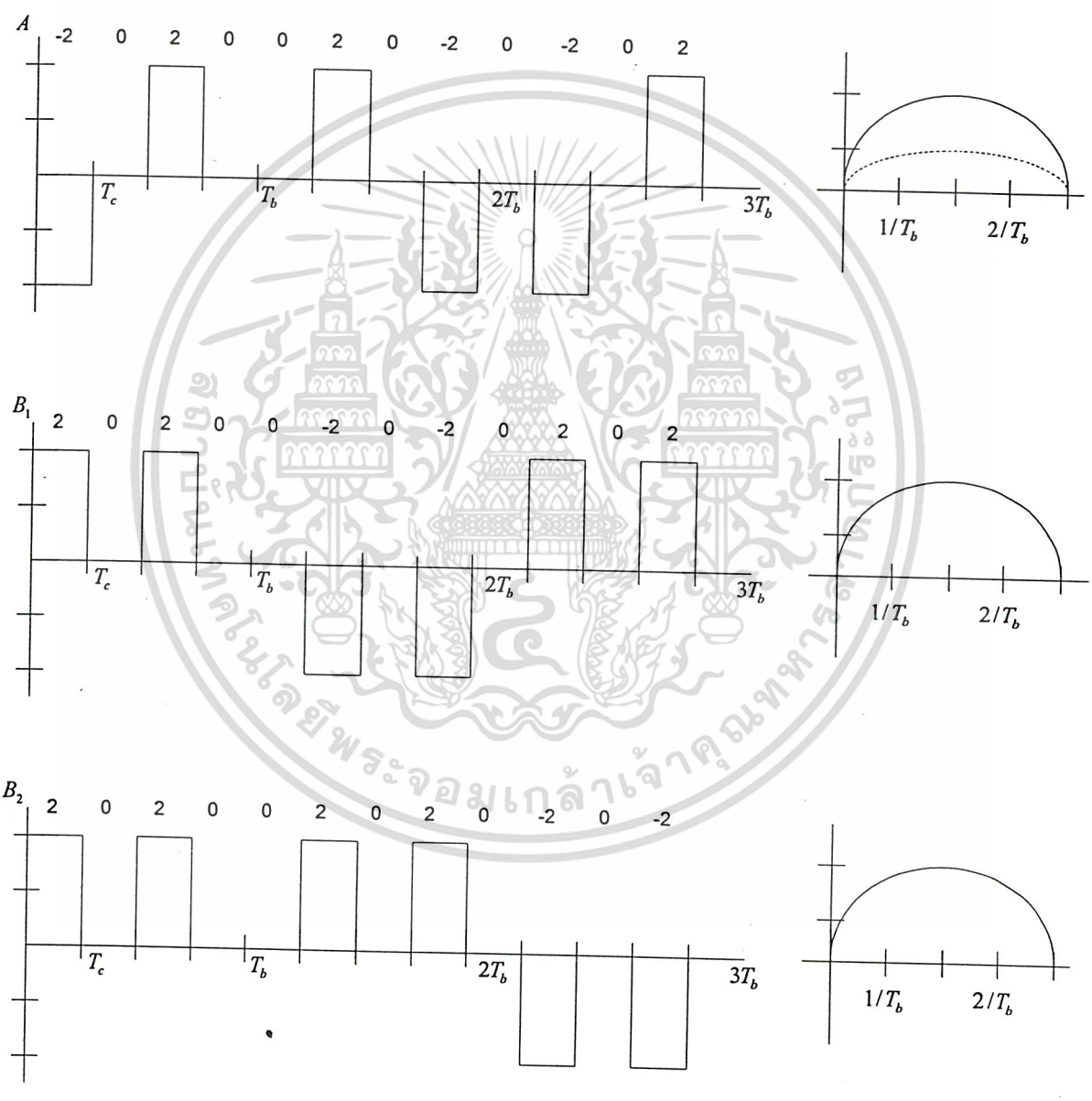
รูปที่ 2.4 แสดงรูปสัญญาณในแกนเวลาและสเปกตรัมในแกนความถี่ของข้อมูล $m_1(t)$ และ $m_2(t)$ รหัส $c_1(t)$ และ $c_2(t)$ และข้อมูลที่ทำการสเพรดแล้ว $m_1(t)\ c_1(t)$ และ $m_2(t)\ c_2(t)$

รูปที่ 2.4 แสดง รูปร่างของสัญญาณ (Waveform) และ สเปกตรัม (Spectrum) ของข้อมูล $m_1(t)$ และ $m_2(t)$ รหัส $c_1(t)$ และ $c_2(t)$ และข้อมูลที่ทำการสเพรดแล้ว $m_1(t)\ c_1(t)$ และ $m_2(t)\ c_2(t)$ โดยในที่เราไม่สนใจรายละเอียดของการคำนวณสเปกตรัมจากรูปสัญญาณในแกนเวลาเหล่านี้ ซึ่งเพียงพอที่จะใช้อธิบายสำหรับจุดประสงค์ในที่นี้ได้ โดยแบนด์วิดท์ของสัญญาณดิจิตอลแบบสุ่มถูกจำกัดอยู่ที่ 1/T โดย T คือ ช่วงเวลา 1 บิต ของสัญญาณดิจิตอลแบบสุ่ม เราสามารถบอกความแตกต่างระหว่าง T_b กับ T_c ได้ โดย T_b คือช่วงเวลา 1 บิตของบิตข้อมูล และ T_c คือ ช่วงเวลาการชิฟ (Chip) ของการรัน (Run) รหัส ในตัวอย่างนี้ ชิฟเรต (Chip rate, $1/T_c$) ของรหัสคิดเป็น 4 เท่าของบิตเรต (bit rate, $1/T_b$: อัตราการเลื่อนบิตข้อมูล) ดังนั้นแบนด์วิดท์ (Effective bandwidth) จะขยายขึ้นเป็น 4 เท่า บางครั้งเรียก ส่วนประกอบการขยายแบนด์วิดท์ว่า Processing

Gain หรือ W/R โดยที่ W คือ แบนด์วิดท์สุดท้าย (Final bandwidth) ของสัญญาณที่ถูกสเพรด และ R คือ แบนด์วิดท์ของสัญญาณเดิม (Baseband message)

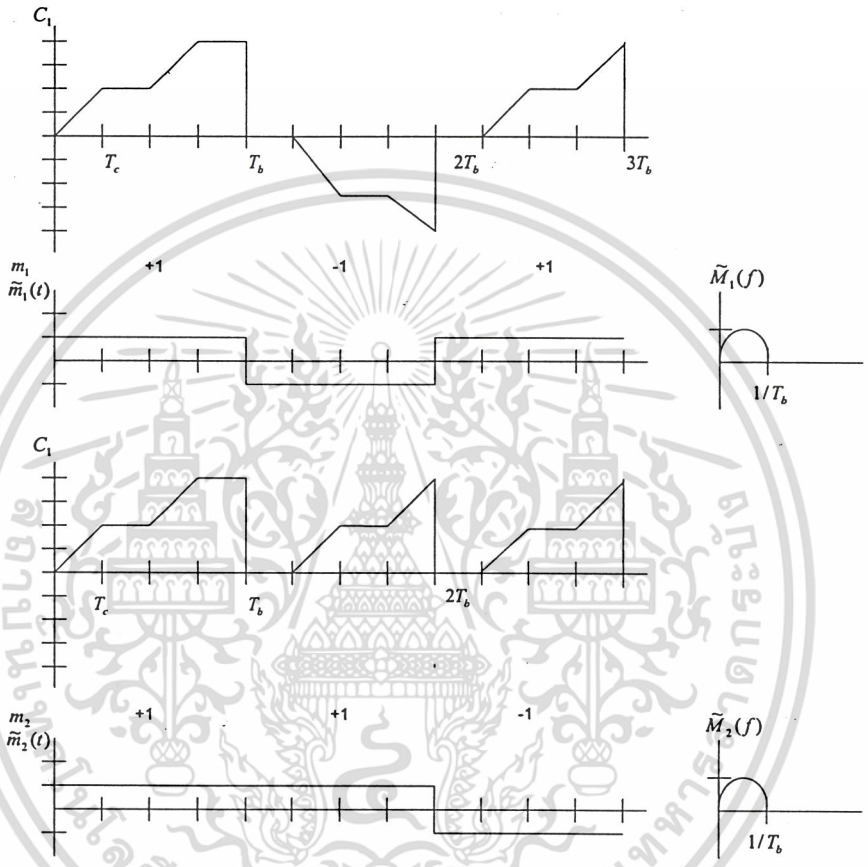
<u>หมายเหตุ</u> : ในตัวอย่างนี้ W เท่ากับ $1/T_c$, R เท่ากับ $1/T_b$ และ Process gain (W/R) เท่ากับ 4 หรือ 6 dB

หลังจากการสเพรดดิง (Spreading) ด้วยรหัสเป็นออร์ธอโกนอลแล้ว ข้อมูลที่ทำการสเพรดแล้ว $m_1(t)c_1(t)$ และ $m_2(t)c_2(t)$ ในขณะนี้ครอบครองแบนด์วิดท์กว้างกว่าสัญญาณเดิม



รูปที่ 2.5 แสดงรูปสัญญาณในแกนเวลาและ
สเปกตรัมในแกนความถี่ของสัญญาณ ณ จุดต่างๆกัน ที่ด้านรับ

รูปที่ 2.5 แสดงรูปสัญญาณ ณ จุดต่าง ๆ กันที่ทางด้านรับ สัญญาณที่จุด A คือสัญญาณที่ได้จากการรวมสัญญาณที่ทำการสเพรดแล้ว 2 สัญญาณเข้าด้วยกัน สเปกตรัมที่จุด A ประกอบด้วย 2 สัญญาณที่แตกต่างกัน ในการกู้คืนสัญญาณ 2 สัญญาณแยกกันออกจากสเปกตรัมที่รวมกันมาโดยนำ สัญญาณที่จุด A ถูกนำมาคูณด้วยรหัสที่ได้จากจุด B_1 และ B_2



รูปที่ 2.6 แสดงรูปสัญญาณที่ได้จากตัวอินทิเกรตและการตัดสินระดับของสัญญาณ

รูปที่ 2.6 แสดงสัญญาณที่จุด C_1 และ C_2 มี $\tilde{m}_1$ และ $\tilde{m}_2$ เป็นสัญญาณที่ได้จากการตัดสินระดับของสัญญาณ และสัญญาณที่กู้คืน $\tilde{m}_1(t)$ และ $\tilde{m}_2(t)$ ตัวอินทิเกรตทำให้กำลังของสัญญาณเพิ่มขึ้นบนช่วงเวลา T_b ของสัญญาณเดิม และตัวตัดสินระดับของสัญญาณทำการตัดสินสัญญาณที่ออกจากตัวอินทิเกรต ว่าแต่ละบิตเป็น +1หรือ -1 ถ้าเอาท์พุทที่ได้จากตัวอินทิเกรตมีค่ามากกว่า 0 ให้ตัดสินว่าเป็น +1 และถ้าเอาท์พุทที่ได้จากตัวอินทิเกรตมีค่าน้อยกว่า 0 ให้ตัดสินว่าเป็น -1 ตัวแปลงสัญญาณดิจิตอลเป็นอานาล็อก (Digital to Analog converter; D/A) จะทำการแปลงผลที่ได้จากตัดสินระดับสัญญาณเป็น $\tilde{m}_1(t)$ และ $\tilde{m}_2(t)$ จากตัวอย่างนี้เป็นตัวอย่างในอุดมคติ กล่าวคือ จะได้สัญญาณ $\tilde{m}_1(t)$ และ $\tilde{m}_2(t)$ ที่กู้กลับคืนมาตรงตามสัญญาณต้นแบบ $m_1(t)$ และ $m_2(t)$ ที่ส่งมา

ตัวอย่างนี้แสดงโครงสร้างของระบบ DS-SS multiple access ขั้นพื้นฐาน เราได้ทำการแสดงการใช้เทคนิค DS-SS สัญญาณที่แตกต่างกันสามารถส่งไปในช่องสัญญาณเดียวกันและใช้ความถี่เดียวกันที่เวลา

เดียวกัน และสัญญาณสามารถกู้กลับคืนมาได้อย่างสมบูรณ์ที่ด้านรับ แต่อย่างไรก็ตาม มีปรากฏการณ์ที่เกิดขึ้นจริงบนโลกนี้มากมายโดยเฉพาะอย่างยิ่งในสภาพแวดล้อมของการสื่อสารในระบบ โทรศัพท์มือถือ ซึ่งจะอาจทำให้ประสิทธิภาพของระบบ DS-SS multiple access ลดลง มีอยู่ 2 ปัญหาที่เกิดขึ้นได้แก่ ปัญหาความใกล้-ไกล (Near - far problem) และปัญหา Partial correlation

ในระบบการสื่อสารโทรศัพท์มือถือนั้นผู้ใช้แต่ละรายอยู่ในภูมิลำเนาที่ต่างกัน แต่ใช้ความถี่เดียวกันส่งโดยใช้เทคนิค DS-SS ผู้ใช้งานบางรายที่อยู่ใกล้กับสถานีฐาน (Base station) มากกว่าผู้ใช้รายอื่นๆ ความแรงของสัญญาณจะสูงกว่าผู้ใช้รายอื่นที่อยู่ไกลออกไป เหตุผลเพราะผู้ใช้งานทุกคนที่ส่งในช่วงความถี่เดียวกันนั้นอาจเกิดการรบกวนกัน เนื่องจากสัญญาณของผู้ใช้ที่อยู่ข้างเคียงมีกำลังงานสูงกว่า จึงทำให้ประสิทธิภาพโดยรวมของระบบลดลง การควบคุมกำลังงานเพื่อให้กำลังงานที่รับได้ที่สถานีฐานมีความเท่าเทียมกันสำหรับผู้ใช้งานทุกคน จากตัวอย่างก่อนหน้านี้ เป็นการสมมติว่าเป็นการควบคุมกำลังงานที่สมบูรณ์แบบ กล่าวคือ $m_1(t)c_1(t)$ และ $m_2(t)c_2(t)$ ทั้งคู่มีขนาดของสัญญาณที่เท่ากัน (เช่น มีขนาดอยู่ในช่วง +1 ถึง -1)

ปัญหาที่สองคือ Partial correlation ปัญหานี้จะเกิดขึ้นเมื่อ ตัวส่งไม่ซิงโครไนซ์กัน แม้ว่าตัวส่งจะซิงโครไนซ์กัน แต่ก็เกิดปัญหาเรื่องความล่าช้าจากการแพร่กระจายคลื่น (Propagation delay) ก็ยังคงมีอยู่ซึ่งจะมีประจำอยู่แล้วในช่องสัญญาณของโทรศัพท์มือถือ ตัวอย่างเช่น รหัสที่ได้กล่าวในสมการที่ 2.10 และ 2.11 ซึ่งเป็นออร์ธอโกนอลกัน

x_i	-1	-1	1	1	-1	-1	1	1
y_i	-1	1	1	-1	-1	1	1	-1

ถ้า y_i เกิดล่าช้า (Delaly) ไป 1 ชิฟ ทำให้เกิดปัญหาความล่าช้าจากการแพร่กระจายคลื่นในช่องสัญญาณของโทรศัพท์มือถือ และจะได้ว่า

x_i	-1	-1	1	1	-1	-1	1	1
y_{i-1}	1	1	-1	-1	1	1	-1	-1

และสามารถตรวจสอบรหัสทั้งสองชุดนี้ว่ายังคงเป็นออร์ธอโกนอลได้ คือ ถ้ารหัสไม่เป็น ออร์ธอโกนอล กันแล้วนั้น การซิงโคไนเซชั่น หรือ การไม่เท่าเทียมกันของช่องสัญญาณ แล้วสัญญาณที่เข้าถึงแบบหลายทางในช่วงความถี่เดียวกันไม่สามารถแบ่งแยกออกจากกัน โดยรหัสที่เป็นออร์ธอโกนอล ผลที่ได้คือเกิดการแทรกสอด (Correlation crosstalk) และเกิดการรบกวนซึ่งกันและกัน ปัจจัยสำคัญที่เพิ่มเข้ามาและต้องคำนึงคือ

$$\int_0^{T-\tau} x(t)y(t+\tau)dt = 0 \qquad (2.14)$$

$$\int_{T-\tau}^{T} x(t)y(t+\tau-T)dt = 0 \qquad (2.15)$$

ดังนั้นการทำออร์ธอโกนอลโดยทั่วไป (Simple orthogonality) ระหว่างรหัส 2 ชุด ไม่เพียงพอ จำเป็นต้องมีค่าที่ได้จากการทำคอรีเลชั่นของรหัสแต่ละตัวที่ เท่ากับ ศูนย์ หรือมีค่าน้อยมาก ๆ ด้วย

ในระบบ CDMA สัญญาณแบนด์แคบ (Narrowband signal) ของผู้ใช้แต่ละรายจะถูกทำการสเพรดเพื่อให้ช่วงแบนด์วิดท์กว้างขึ้น การส่งข้อมูลที่ใช้ช่วงแบนด์วิดท์ที่กว้างขึ้นนี้จะดีกว่าช่วงแบนด์วิดท์ที่แคบ ๆ สัญญาณแบนด์แคบของผู้ใช้แต่ละรายจะถูกทำการสเพรดโดยการกำหนดรหัสของแต่ละแบนด์วิดท์ให้ต่างกัน และรหัสแต่ละชุดจะเป็นออร์ธอโกนอลกับรหัสอีกชุดหนึ่ง การส่งไปตามช่องสัญญาณของผู้ที่ใช้งานพร้อมกันได้สำเร็จโดยใช้กลุ่มของรหัสที่เป็นออร์ธอโกนอล นำทุกสัญญาณที่สเพรดให้ช่วงแบนด์วิดท์กว้างขึ้นนี้ มารวมเข้าด้วยกันเป็นสัญญาณองค์ประกอบร่วม (Composite signal) สัญญาณองค์ประกอบร่วมนี้จะถูกส่งออกอากาศไปโดยใช้ช่วงความถี่เดียวกัน ทางด้านรับสามารถแยกความแตกต่างของผู้ใช้แต่ละรายได้ด้วยการใช้รหัสที่ได้คัดลอกไว้ ทางด้านรับจะทำการคัดเลือกผู้ใช้ที่ต้องการออกจากสัญญาณองค์ประกอบร่วมโดยตรวจสอบความสัมพันธ์กับรหัสเดิม ส่วนผู้ใช้คนอื่นที่มีรหัสไม่ตรงกับรหัสที่ทางด้านรับต้องการก็จะถูกปฏิเสธไป

2.8 การทำ Multiple access โดยการใช้ PN code (Pseudo Random Binary Sequences)

Pseudo Random Binary Sequences (PRBS) หรือที่รู้จักในชื่อของ Pseudo Random Number หรือ Pseudo Noise code (PN code) คือ ชุดรหัส (Code sequence) ซึ่งปรากฏในรูปการสุ่มกลุ่มบุคคลที่สาม ผู้ซึ่งไม่รู้มาก่อนเลยว่าวิธีการสร้างรหัสทำอย่างไร รหัสไม่ได้เป็นการสุ่ม มันจะมีลักษณะเป็นคาบ ๆ และวนกลับมาสร้างใหม่หลังจากสิ้นสุดความยาวของรหัสในแต่ละครั้ง

ชุดของรหัสถูกนำมาใช้ในการสเพรดสเปกตรัม สำหรับสัญญาณข้อมูลไปบนแบนด์วิดท์ที่กว้างขึ้นอย่างมากและยังช่วยในการเพิ่มการรักษาความปลอดภัยให้แก่ข้อมูล อย่างไรก็ตามการรักษาความปลอดภัยยังขึ้นอยู่กับคุณสมบัติของรหัส และคุณสมบัติของระบบที่การรักษาความปลอดภัยเป็นสิ่งสำคัญที่สุด ตัวอย่างเช่น ระบบการทหารข้อมูลมักจะถูกทำการเข้ารหัสก่อนที่มันจะถูกใช้

คุณสมบัติที่มีประโยชน์ของ PRBS มีดังนี้

1. การป้องกันการรบกวน (Interference Protection) – การเข้ารหัสจะทำให้ Processing gain มากขึ้น โดยมีผลทำให้แบนด์วิดท์ต้องกว้างขึ้นตามไปด้วย
2. การรักษาความปลอดภัยข้อมูล – การเข้ารหัสทำให้สามารถป้องกันสัญญาณจากการดักฟังหรือการถูกขโมย โดยความสามารถในการป้องกันนี้จะขึ้นอยู่กับระบบการรักษาความปลอดภัยของรหัส
3. การลดผลกระทบที่เกิดจากสัญญาณรบกวน (Noise-Effect Reduction) – การทำ Error-detection และ Error-correction ทำให้สามารถลดผลที่เกิดจากสัญญาณรบกวน และ การรบกวน (Interference)

2.8.1 Maximal Linear Sequences

Maximal Linear Sequences หรือ m-sequence คือ รหัสที่ยาวที่สุดที่สามารถสร้างจาก ชิฟรีจีสเตอร์ (Shift register) หรือ อุปกรณ์หน่วงเวลา (Delay element) เพื่อให้ได้ความยาวของรหัสตามที่ต้องการ ความยาวของรหัสที่มากที่สุดเท่ากับ $2^n - 1$ ชิฟ โดย n คือ จำนวนสเตจ (stage) ของ ชิฟรีจีสเตอร์

ตัวกำเนิดชุดรหัส ประกอบขึ้นด้วย ชิฟรีจีสเตอร์ และป้อนกลับจากชิฟรีจีสเตอร์ 2 สเตจ (Stage) หรือมากกว่า โดยบวกเข้าด้วยกันแบบ 2-modulo ไม่เพียงแต่การรวมกันของจุดป้อนกลับที่จะสร้าง Maximal sequence ได้เท่านั้นแต่ยังมีอีกหลายจุดหรือหลายตำแหน่งที่สามารถนำมาใช้ทำเป็นจุดป้อนกลับ ได้อีก

คุณสมบัติที่น่าสนใจของ Maximal sequence มีดังนี้

1. มีจำนวนของบิต 1 ในแถวลำดับ ใกล้เคียงกับจำนวนของบิต 0 ในการ ชิฟ 1 ครั้ง ตัวอย่างเช่น ในการชิฟ 511 ครั้ง มีบิต 1 อยู่ 256 ตัว และมีบิต 0 อยู่ 255 ตัว คุณสมบัติข้อนี้เป็นทำให้ปัญหาเรื่อง DC component ในการเข้ารหัสนั้นถูกกำจัดไป
2. สถิติการกระจายของบิต 1 และบิต 0 จะเท่ากันเสมอ
3. การบวกแบบ 2 โมดูโล สำหรับ Maximal code ด้วยการที่ตัวมันเองบวกกับตัวมันเองที่เฟสชิฟ จะได้ค่าๆหนึ่งที่ไม่เท่ากับสองข้างแรกที่นำมาบวกกัน
4. ทุก ๆ สถานะ ที่เป็นไปได้ของ ชิฟรีจีสเตอร์ n สเตจ ที่ส่งออกมาใน 1 ครั้งของการ generate 1 cycle ยกเว้นกรณีที่ให้ทุกสถานะเป็นศูนย์หมด และต้องหลีกเลี่ยงกรณีที่ทุกสเตจเป็นศูนย์ โดยที่อาจจะทำการ ล็อคค่าในแต่ละ สถานะ ไว้ถาวร

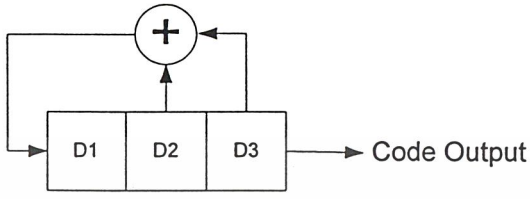
2.8.2 Code Sequence Generator Configuration

ตัวกำเนิดชุดรหัสแบบลิเนียร์ (Linear code sequence) ถูกสร้างมาจาก 3 ส่วน ได้แก่ จำนวนของอุปกรณ์หน่วงเวลา เส้นทางที่ใช้ในการต่อย้อนกลับ ตัวบวกกันแบบลิเนียร์ (Linear combining element) โดยในส่วนของอุปกรณ์หน่วงเวลา ปกติจะใช้ฟลิปฟลอป (Flip-Flop) มาทำหน้าที่เป็นตัวชิฟรีจีสเตอร์และในส่วนของตัวบวกกันแบบลิเนียร์ ใช้ตัวบวกแบบ 2 โมดูโล หรือที่รู้จักในชื่อของ Exclusive-or gates (XOR gates)

มี 2 วิธีในการสร้าง code sequence ได้แก่

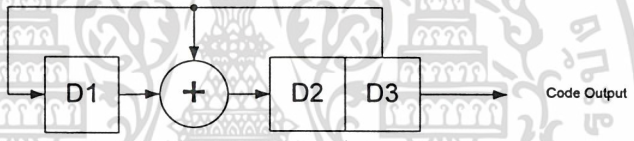
1. การสร้างโค้ดแบบไม่ซับซ้อน (Simple type code sequence generator)
2. การสร้างโค้ดแบบโมดูล่า (Modular type code sequence generator)

การสร้างโค้ดแบบไม่ซับซ้อนนั้น สามารถทำได้โดยการนำเอาท์พุทจาก สเตจ ที่ n และอีก 1 สเตจ หรือมากกว่า 1 สเตจ ที่อยู่ระหว่างกลาง นำทั้งสองส่วนมารวมเขาด้วยกันโดยใช้ตัวบวกแบบ 2 โมดูโล ก่อนที่จะนำเอาท์พุทที่ได้นั้นไปทำการป้อนกลับไปยังอินพุตของสเตจ แรก ดังรูปที่ 2.7 แสดง การสร้างโค้ดแบบไม่ซับซ้อน



รูปที่ 2.7 แสดงบล็อกไดอะแกรมของ วงจรสร้างโค้ดแบบไม่ซับซ้อน

การสร้างโค้ดแบบไม่ซับซ้อน ทำได้โดยจัดวางตัวบวกไว้ระหว่าง สถานะ(Stage) ของการสร้างโค้ดแบบโมดูล่า ถูกสร้างจากตัวอุปกรณ์หน่วงเวลา และ ตัวบวกแบบ 2 โมดูโล ซึ่งใช้จำนวนอุปกรณ์เท่ากับแบบการสร้างโค้ดแบบไม่ซับซ้อน แสดงดังรูปที่ 2.8 เป็นการสร้างโค้ดแบบโมดูล่า



รูปที่ 2.8 แสดงบล็อกไดอะแกรมของ ตัวสร้างโค้ดแบบโมดูล่า

ข้อได้เปรียบของ ตัวสร้างโค้ดแบบโมดูล่า คือ ค่าของการหน่วงเวลา (Delay time) อันเนื่องมาจากผลของเส้นทางการป้อนกลับจะมีค่าลดลง นั่นก็หมายความว่าการเข้ารหัสจะเร็วขึ้นเท่าที่เป็นไปได้ ซึ่ง

2.8.3 การเลือกค่าโค๊ดที่เป็นเชิงเส้น (Choosing a linear code)

ปัญหาที่ยากที่สุดในการที่เราจะทำการสร้างโค๊ดขึ้นมาเห็นจะได้แก่ (1) การหาจุดโลจิกที่จะต้องทำการป้อนกลับเพื่อที่จะทำให้เกิดโค้ดที่มีความยาวตามที่เราต้องการ (2) ทำการตรวจสอบชุดโค๊ดที่ได้จากลำดับของตัวสร้างโค๊ดหลังจากทำการสร้างขึ้นมาว่ามันสามารถทำงานได้อย่างถูกต้องหรือไม่ ซึ่งเป็นการยากอย่างแน่นอนที่เราจะทำการสร้างลำดับของชิฟรีจีสเตอร์ เพื่อที่จะทำการให้กำเนิดโค๊ดที่มีความยาวตามต้องการ อย่างไรก็ตามมันมีความเป็นไปได้ที่เราจะทำการหาจุดป้อนกลับที่จะให้ได้ความยาวของโค๊ดที่มากสุดได้โดยการทดลองไปเรื่อยๆจนกว่าจะเจอลำดับที่ทำให้เกิดชุดโค๊ดยาวที่สุด

ซึ่งจากตารางที่2.1 นั้นเราสามารถจะทราบถึงจุดป้อนกลับที่จะสามารถทำให้เกิดความยาวโค๊ดที่สูงสุดได้ ซึ่งจะเป็นการสะดวกอย่างมากในการที่จะออกแบบเพื่อที่จะทำการสร้างโค๊ดให้มีความยาวสูงที่สุดตามที่ต้องการ

Number of Stages	Code Length	Maximal Taps
2	3	[2, 1]
3	7	[3, 1]
4	15	[4, 1]
5	31	[5, 2] [5, 4, 3, 2] [5, 4, 2,1]
6	63	[6, 1] [6, 5, 2, 1] [6, 5, 3, 2]
7	127	[7, 1] [7, 3] [7, 3, 2, 1] [7, 4, 3, 2] [7, 6, 4, 2] [7, 6, 3, 1] [7, 6, 5, 2] [7, 6, 5, 4, 2, 1] [7, 5, 4, 3, 2, 1]
8	255	[8, 4, 3, 2] [8, 6, 5, 3] [8, 6, 5, 2] [8, 5, 3, 1] [8, 6, 5, 1] [8, 7, 6, 1] [8, 7, 6, 5, 2, 1] [8, 6, 4, 3, 2, 1]
9	511	[9, 4] [9, 6, 4, 3] [9, 8, 5, 4] [9, 8, 4, 1] [9, 5, 3, 2] [9, 8, 6, 5] [9, 8, 7, 2] [9, 6, 5, 4, 2, 1] [9, 7, 6, 4, 3, 1] [9, 8, 7, 6, 5, 3]
10	1023	[10, 3] [10, 8, 3, 2] [10, 4, 3, 1] [10, 8, 5, 1] [10, 8, 5, 4] [10, 9, 4, 1] [10, 8, 4, 3] [10, 5, 3, 2] [10, 5, 2, 1] [10, 9, 4, 2]
11	2047	[11, 1] [11, 8, 5, 2] [11, 7, 3, 2] [11, 5, 3, 5] [11, 10, 3, 2] [11, 6, 5, 1] [11, 5, 3, 1] [11, 9, 4, 1] [11, 8, 6, 2] [11, 9, 8, 3]
12	4095	[12, 6, 4, 1] [12, 9, 3, 2] [12, 11, 10, 5, 2, 1] [12, 11, 6, 4, 2, 1] [12, 11, 9, 7, 6, 5] [12, 11, 9, 5, 3, 1] [12, 11, 9, 8, 7, 4] [12, 11, 9, 7, 6, 5] [12, 9, 8, 3, 2, 1] [12, 10, 9, 8, 6, 2]
13	8191	[13, 4, 3, 1] [13, 10, 9, 7, 5, 4] [13, 11, 8, 7, 4, 1] [13, 12, 8, 7, 6, 5] [13, 9, 8, 7, 5, 1] [13, 12, 6, 5, 4, 3] [13, 12, 11, 9, 5, 3] [13, 12, 11, 5, 2, 1] [13, 12, 9, 8, 4, 2] [13, 8, 7, 4, 3, 2]

14	16383	[14, 12, 2, 1] [14, 13, 4, 2] [14, 13, 11, 9]
		[14, 10, 6, 1] [14, 11, 6, 1] [14, 12, 11, 1]
		[14, 6, 4, 2] [14, 11, 9, 6, 5, 2]
		[14, 13, 6, 5, 3, 1] [14, 13, 12, 8, 4, 1]
		[14, 8, 7, 6, 4, 2] [14, 10, 6, 5, 4, 1]
		[14, 13, 12, 7, 6, 3] [14, 13, 11, 10, 8, 3]
15	32767	[15, 13, 10, 9] [15, 13, 10, 1] [15, 14, 9, 2]
		[15, 1] [15, 9, 4, 1] [15, 12, 3, 1] [15, 10, 5, 4]
		[15, 10, 5, 4, 3, 2] [15, 11, 7, 6, 2, 1]
		[15, 7, 6, 3, 2, 1] [15, 10, 9, 8, 5, 3]
		[15, 12, 5, 4, 3, 2] [15, 10, 9, 7, 5, 3]
		[15, 13, 12, 10] [15, 13, 10, 2] [15, 12, 9, 1]
		[15, 14, 12, 2] [15, 13, 9, 6] [15, 7, 4, 1]
		[15, 4] [15, 13, 7, 4]
16	65535	[16, 12, 3, 1] [16, 12, 9, 6] [16, 9, 4, 3]
		[16, 12, 7, 2] [16, 10, 7, 6] [16, 15, 7, 2]
		[16, 9, 5, 2] [16, 13, 9, 6] [16, 15, 4, 2]
		[16, 15, 9, 4]
17	131071	[17, 3] [17, 3, 2, 1] [17, 7, 4, 3]
		[17, 16, 3, 1] [17, 12, 6, 3, 2, 1]
		[17,8,7,6,4,3] [17,11,8,6,4,2]
		[17,9,8,6,4,1] [17,16,14,10,3,2]
		[17, 12, 11, 8, 5, 2]
18	262143	[18,17] [18,10,7,5] [18,13,11,9,8.7,6,3]
		[18, 17, 16, 15, 10, 9, 8, 7]
		[18, 15, 12, 11, 9, 8, 7, 6]
19	524287	[19, 5, 2, 1] [19, 13, 8, 5, 4, 3]
		[19,12,10,9,7,3] [19,17,15,14,13,12,6,1]
		[19, 17, 15, 14, 13, 9, 8, 4, 2, 1]
		[19,16,13,11,19,9,4,1] [19,9,8,7,6,3]
		[19, 16, 15, 13, 12, 9, 5, 4, 2, 1]
		[19, 18, 15, 14, 11, 10, 8, 5, 3, 2]
		[19, 18, 17, 16, 12, 7, 6, 5, 3, 1]

20	1048575	[20, 3] [20, 9, 5, 3] [20, 19, 4, 3]
		[20,11,8,6,3,2] [20,17,14,10,7,4,3,2]
21	2097151	[21, 2] [21, 14, 7, 2] [21, 13, 5, 2]
		[21,14,7,6,3,2] [21,8,7,4,3,2]
		[21,10,6,4,3,2] [21, 15, 10,9,5,4,3,2]
		[21,14,12,7,6,4,3,2] [21,20,19,18,5,4,3,2]
22	4194303	[22, 1] [22, 9, 5, 1] [22, 20, 18, 16,6,4,2, 1]
		[22,19,16,13,10,7,4,1] [22,17,9,7,2,1]
		[22,17,13,12,8,7,2,1] [22,14,13,12,7,3,2,1]
23	8388607	[23, 5] [23, 17, 11, 5] [23, 5, 4, 1]
		[23, 12, 5, 4] [23, 21, 7, 5] [23, 16, 13, 6, 5, 3]
		[23,11,10,7,6,5] [23,15,10,9,7,5,4,3]
		[23, 17, 11, 9,8,5,4, 1] [23, 18, 16, 13, 11, 8, 5, 2]
24	16777215	[24, 7, 2, 1] [24, 4, 3, 1]
		[24,22,20,18,16,14,11,9,8,7,5,4]
		[24,21,19,18,17,16,15,14,13,10,9,5,4,1]
25	33554431	[25, 3] [25, 3, 2, 1] [25, 20, 5, 3] [25, 12, 5, 4]
		[25, 17, 10, 3, 2, 1] [25, 23, 21, 19, 9, 7, 5, 3]
		[25, 18, 12, 11, 6, 5, 4] [25, 20, 16, 11,5,3,2, 1]
		[25, 12, 11, 8, 7, 6, 4, 3]
26	67108863	[26,6,2,1] [26, 22, 21,16,12,11,10,8,5,4,3,1]
27	134217727	[27,5,2,1] [27,18,11,10,9,5,4,3]
28	268435455	[28,3] [28,13,11,9,5,3] [28,22,11,10,4,3]
		[28, 24, 20, 16, 12, 8, 4, 3, 2, 1]
29	536870911	[29, 2] [29, 20, 11, 2] [29, 13, 7, 2]
		[29,21,5,2] [29,26,5,2] [29,19,16,6,3,2]
		[29, 18, 14, 6, 3, 2]
30	1073741823	[30, 23, 2, 1] [30, 6, 4, 1]
		[30, 24, 20, 16, 14, 13, 11, 7, 2, 1]
31	2147483647	[31, 29, 21, 17] [31, 28, 19, 15] [31, 3]
		[31, 3, 2, 1] [31, 13, 8, 3] [31, 21, 12, 3, 2, 1]
		[31,20,18,7,5,3] [31,30,29,25] [31, 28, 24,10]
		[31,20,15,5,4,3] [31,16,8,4,3,2]

32	4294967295	[32, 22, 2, 1] [32, 7, 5, 3, 2, 1]
		[32, 28, 19, 18, 16, 14, 11, 10,9,6,5, 1]
		[33, 13] [33, 22, 13, 11] [33, 26, 14, 10]
33	8589934591	[33, 6, 4, 1] [33, 22, 16, 13, 11, 8]

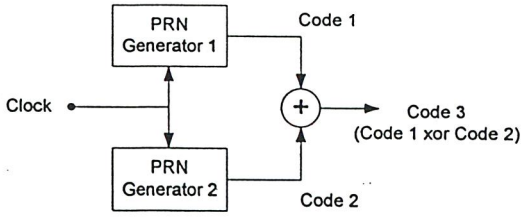
ตารางที่ 2.1 แสดงรายละเอียดจุดป้อนกลับที่ถูกต้องสำหรับตัวกำเนิด 2 สเตจ จนถึง 33 สเตจ

สำหรับทุกเซตของการทำเชื่อมต่อแบบป้อนกลับ หรือการทำ แท็ป (Tap) ซึ่งจะมีกลุ่มเซตเสมือน ของการทำแท็ป ที่จะให้กำเนิดรหัสที่มีลักษณะเฉพาะ (Identical code) แต่ว่าจะเปลี่ยนแปลงไปตามเวลา รูปแบบเสมือนของ ป้อนกลับของเซต [n, r,...,p] คือ [n, n-r, ..., n-p] ตัวอย่างเช่น รูปแบบเสมือนของเซต [3,1] คือ [3,3-1] ซึ่งจะเท่ากับ [3,2] ซึ่งตัวกำเนิดชุดรหัสในรูปที่ 2.7 ใช้ แท็ปป้อนกลับแบบ [3,2] ซึ่งจะเป็นการสร้างรหัสที่ยาวที่สุด (Maximal length code sequence generator)

2.8.4 Composite Codes

แม้ว่า รหัสแบบลิเนียร์ที่ยาวที่สุด (Maximal linear code) ไม่สามารถทำให้ประสิทธิภาพของมันดีขึ้นในระบบสเพรดสเปกตรัม Composite code ถูกสร้างขึ้นด้วยการรวมกันของรหัสแบบลิเนียร์ที่ยาวที่สุด ทำให้ได้เปรียบกว่า ตัวอย่างของ Composite code เช่น JPL ranging code และ Gold code JPL ranging code มีคุณสมบัติการทำคอรีเลชั่นในลักษณะพิเศษสำหรับการซิงโครไนเซชั่น ด้วยเหตุที่ Gold code มีคุณสมบัติการคอรีเลชั่นที่ดี ซึ่งเป็นสิ่งที่จำเป็นสำหรับการเข้าถึงแบบหลายทางแต่อย่างไรก็ตาม Composite code ไม่สามารถให้ค่าที่สูงสุดได้

Gold code สามารถสร้างได้โดย การบวกแบบ 2 โมดูโล เป็นคู่ ๆ ของรหัสแบบลิเนียร์ที่ยาวที่สุด Gold code ที่ถูกสร้างเป็น ความยาวที่เหมือนกันกับรหัสฐานสองตัว แต่มันก็ไม่สามารถให้ค่าที่สูงสุดได้ ทุก ๆ ครั้งของการเปลี่ยนเฟสระหว่างรหัสฐานสองตัวที่ให้ Gold code ที่แตกต่างกัน ตัวกำเนิดรหัส Gold code ผลิตได้ 2^{n-1} Gold code บวกกับ r maximal code โดยที่ r คือ จำนวนของตัวกำเนิดรหัสฐาน (Number of base code generator) และ n คือ ความยาวของแหล่งกำเนิด (Generator length) รูปที่ 2.9 แสดงตัวอย่างรูปแบบของตัวกำเนิด Gold code

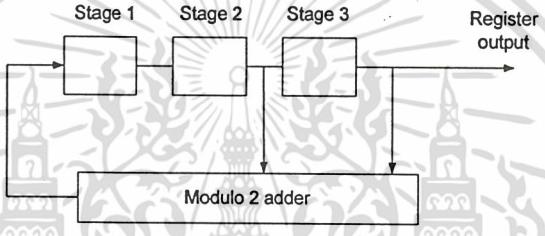


รูปที่ 2.9 แสดงตัวอย่างรูปแบบของตัวกำเนิดชุดรหัส
Gold code (Typical gold code sequence generator configuration)

ตัวกำเนิด Gold code สามารถผลิตรหัสที่มีความแตกต่างกันเป็นจำนวนมากได้อย่างดี โดยที่คุณสมบัติจะขึ้นกับการคอรีเลชั่นซึ่งเป็นประโยชน์อย่างมากสำหรับระบบ CDMA

2.9 การสร้างรหัส PN code (Generation of PN code)

ชุดรหัส PN code สามารถสร้างได้โดยการป้อนกลับชิฟรีจีสเตอร์ ดังตัวอย่างที่แสดงใน รูปที่ 2.10 เป็นการใช้ชิฟรีจีสเตอร์ 3 ตัว แต่ละบิตจะถูกชิฟหรือเลื่อนไปยังสเตจต่าง ๆ ของชิฟรีจีสเตอร์ เอาท์พุทของสเตจสุดท้ายและสเตจที่อยู่ระหว่างกลางอีก 1 สเตจ จะถูกนำมารวมเข้าด้วยกันและป้อนกลับไปยังอินพุทของสเตจแรก ชิฟรีจีสเตอร์จะเริ่มต้นการทำงานด้วยค่าเริ่มต้นที่ได้กำหนดไว้ และเมื่อชิฟรีจีสเตอร์ได้รับสัญญาณนาฬิกา และบิตก็จะถูกเลื่อนผ่านไปยังสเตจต่าง ๆ วิธีนี้มีความต่อเนื่องของชิฟรีจีสเตอร์ที่จะสร้างบิตเอาท์พุทที่จะป้อนไปเป็นบิตอินพุทของสเตจแรก

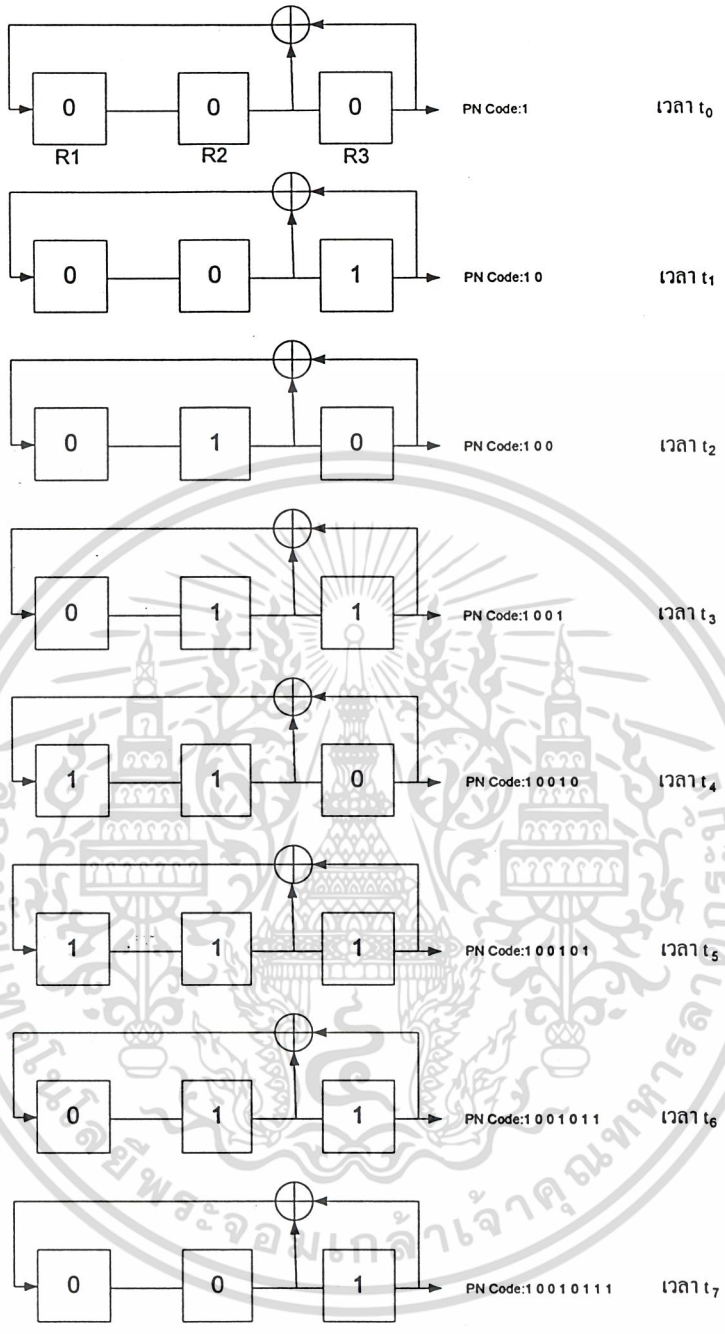


รูปที่ 2.10 แสดงตัวอย่างการสร้าง PN code จากชิฟรีจีสเตอร์

บิตเอาท์พุทของสเตจสุดท้ายก็คือ PN code ตัวอย่างต่อไปนี้แสดงการสร้างรหัสโดยใช้ชิฟรีจีสเตอร์ที่แสดงในรูปที่ 2.10 กำหนดให้ใช้ค่าเริ่มต้นในชิฟรีจีสเตอร์เป็น [1, 0, 1] เอาท์พุทของสเตจที่ 3 เป็นเอาท์พุทของชิฟรีจีสเตอร์ เมื่อป้อนสัญญาณนาฬิกาเข้าไปบิตเอาท์พุทของชิฟรีจีสเตอร์ก็จะเลื่อนออกมา โดยผลลัพท์ที่ได้แสดงไว้ใน ตารางที่ 2.2

	Output	Output	Output	Output
Shift	Stage 1	Stage 2	Stage 3	Register
0	1	0	1	1
1	1	1	0	0
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	1	0	0	0
6	0	1	0	0
7	1	0	1	1

ตารางที่ 2.2 ผลที่ได้จากบิตเอาท์พุทของชิฟรีจีสเตอร์ในรูปที่ 2.10 ที่กำหนดค่าเริ่มต้นในชิฟรีจีสเตอร์เป็น [1, 0, 1]



รูปที่ 2.11 การทำงานของชิฟรีจีสเตอร์เพื่อสร้าง PN code

จากรูปที่ 2.11 จะพบว่าเมื่อปล่อยให้วงจรทำงานตามจังหวะของสัญญาณ Clock ไปเรื่อย ๆ ค่าใน ชิฟรีจีสเตอร์จะมีการชิฟเปลี่ยนค่าไปเช่นกัน ทำให้เกิดเป็นรหัส PN Code ต่าง ๆ กันไปตามเวลา ซึ่งในการชิฟในครั้งที่ 7 ค่าในชิฟรีจีสเตอร์จะกลับไปเป็นค่าเริ่มต้นที่ได้กำหนดไว้ และในการชิฟครั้งต่อไปก็จะได้ผลออกมาเช่นเดิม ดังนั้นความยาวช่วงคาบของ PN code มีค่าเท่ากับ 7 และรูปแบบของ PN code ที่ออกมาจากชิฟรีจีสเตอร์คือ

$$P = [1\,0\,1\,1\,1\,0\,0] \qquad (2.16)$$

รหัสที่ได้จากวิธีนี้เรียกว่ารหัสที่ความยาวมากที่สุดของชิฟรีจีสเตอร์ (Maximal-length shift register code) และความยาว L ของความยาวมากสุดของรหัส (Maximal-length code) หาได้จาก

$$L = 2^N - 1 \tag{2.17}$$

โดยที่ N คือ จำนวนสเตจ จากตัวอย่างนี้ ได้ว่า N = 3 และ L = 7 รูปแบบของ PN code หาได้จาก ค่าที่ป้อนกลับและค่าเริ่มต้นของชิฟรีจีสเตอร์ที่ได้กำหนดไว้ ตัวอย่างเช่นถ้ากำหนดค่าเริ่มต้นในชิฟรีจีสเตอร์เป็น [0, 0, 0] แล้วค่าเอาท์พุทของชิฟรีจีสเตอร์ที่ได้จะเป็นศูนย์ทั้งหมด และรหัสที่ได้จะไม่เกิด Maximal length

PN code 7 ชุด ที่สร้างขึ้นนี้แทนด้วย p_n ; n = 0, 1, 2, …., n และเปลี่ยน 0 เป็น -1

$$p_0 = [+1\ -1\ +1\ +1\ +1\ -1\ -1]$$
$$p_1 = [-1\ +1\ -1\ +1\ +1\ +1\ -1]$$
$$p_2 = [-1\ -1\ +1\ -1\ +1\ +1\ +1]$$
$$p_3 = [+1\ -1\ -1\ +1\ -1\ +1\ +1]$$
$$p_4 = [+1\ +1\ -1\ -1\ +1\ -1\ +1]$$
$$p_5 = [+1\ +1\ +1\ -1\ -1\ +1\ -1]$$
$$p_6 = [-1\ +1\ +1\ +1\ -1\ -1\ +1]$$

ดังนั้นสามารถสรุปคุณสมบัติของรหัสที่จะนำมาใช้ในระบบ DS-SS multiple access ได้ดังนี้

1. การทำ Cross-correlation ควรจะให้ค่าเท่ากัน 0 หรือมีค่าน้อยมาก ๆ
2. แต่ละรหัสในแต่ละชุดรหัส มีจำนวนของ +1 และ -1 เท่ากัน หรือ มีจำนวนของ +1 ต่างจากจำนวนของ -1 1 ตัว
3. ค่าที่ได้จากการทำดอทโปรดักแต่ละรหัสควรมีค่าเท่ากับ 1

เพราะว่า ความยาวสูงสุดของ PN code มีค่าเป็นจำนวนคี่ และรหัสที่แสดงในข้างต้นมี +1อยู่ 4 ตัว และมี -1 อยู่ 3 ตัว รหัสชุดนี้ตรงคุณสมบัติข้อที่ 2

2.10 Channelization using PN codes

ต่อนี้เป็นการแสดงให้เห็นว่า PN code สามารถใช้กับการทำ Multiple Accessได้อย่างไร โดยใช้ตัวอย่างเดิมอีกครั้ง สมมติให้ผู้ใช้งานทั้งสามคนที่มีคุณสมบัติเหมือนกัน แต่ส่งข้อมูลที่ต่างกันสามชุดโดยเรียงตามลำดับ

$$m_1 = [+1\ -1\ +1]$$
$$m_2 = [+1\ +1\ -1]$$
$$m_3 = [-1\ +1\ +1]$$

ผู้ใช้แต่ละรายถูกกำหนดด้วย PN code โดยเรียงกันตามลำดับดังนี้

$$p_0 = [+1\ -1\ +1\ +1\ +1\ -1\ -1]$$
$$p_3 = [+1\ -1\ -1\ +1\ -1\ +1\ +1]$$
$$p_6 = [-1\ +1\ +1\ +1\ -1\ -1\ +1]$$

ข้อมูลชุดที่ 1 ถูกกำหนดด้วย PN code 0 ข้อมูลชุดที่ 2 ถูกกำหนดด้วย PN code 3 และข้อมูลชุดที่ 3 ถูกกำหนดด้วย PN code 6 ข้อมูลแต่ละชุดถูกแบ่งแยกออกจากกันด้วย PN code ของตัวมันเอง

หมายเหตุ ในที่นี้กำหนดให้ ชิฟเรตของ PN code เป็น 7 เท่าของบิตเรตของข้อมูล ทำให้ Processing gain เป็น 7 เท่า

<u>ข้อมูลชุดที่ 1</u>

$m_1(t)$	+1	-1	+1
$m_1(t)$	+1 +1 +1 +1 +1 +1 +1	-1 -1 -1 -1 -1 -1 -1	+1 +1 +1 +1 +1 +1 +1
$p_0(t)$	+1 -1 +1 +1 +1 -1 -1	+1 -1 +1 +1 +1 -1 -1	+1 -1 +1 +1 +1 -1 -1
$m_1(t)\ p_0(t)$	+1 -1 +1 +1 +1 -1 -1	-1 +1 -1 -1 -1 +1 +1	+1 -1 +1 +1 +1 -1 -1

<u>หมายเหตุ</u> $m_1(t)\ p_0(t)$ คือ สัญญาณสเพรดสเปกตรัมของข้อมูลชุดที่ 1

<u>ข้อมูลชุดที่ 2</u>

$m_2(t)$	+1	+1	-1
$m_2(t)$	+1 +1 +1 +1 +1 +1 +1	+1 +1 +1 +1 +1 +1 +1	-1 -1 -1 -1 -1 -1 -1
$p_3(t)$	+1 -1 -1 +1 -1 +1 +1	+1 -1 -1 +1 -1 +1 +1	+1 -1 -1 +1 -1 +1 +1
$m_2(t)\ p_3(t)$	+1 -1 -1 +1 -1 +1 +1	+1 -1 -1 +1 -1 +1 +1	-1 +1 +1 -1 +1 -1 -1

ข้อมูลชุดที่ 3

$m_3(t)$ -1 +1 +1

$m_3(t)$ -1 -1 -1 -1 -1 -1 -1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1

$p_6(t)$ -1 +1 +1 +1 -1 -1 +1 -1 +1 +1 +1 -1 -1 +1 -1 +1 +1 +1 -1 -1 +1

$m_3(t)\ p_6(t)$ +1 -1 -1 -1 +1 +1 -1 -1 +1 +1 +1 -1 -1 +1 -1 +1 +1 +1 -1 -1 +1

ทำการแปลงสัญญาณสเพรดสเปกตรัมของข้อมูลทั้งสามชุด $m_1(t)\ p_0(t)$, $m_2(t)\ p_3(t)$ และ $m_3(t)\ p_6(t)$ ให้อยู่ในรูปของสัญญาณองค์ประกอบร่วม $C(t)$ ดังสมการ

$$C(t) = m_1(t)\ p_0(t) + m_2(t)\ p_3(t) + m_3(t)\ p_6(t)$$

ผลลัพธ์ของ $C(t)$ ที่ได้คือ

$$C(t) = +3\ -3\ -1\ +1\ +1\ +1\ -1\ -1\ +1\ -1\ +1\ -3\ +1\ +3\ -1\ +1\ +3\ +1\ +1\ -3\ -1$$

$C(t)$ คือ สัญญาณองค์ประกอบร่วมที่ถูกส่งในช่วงความถี่คลื่นวิทยุ (RF band) ถ้าไม่สนใจความผิดพลาดที่เกิดขึ้นระหว่างกระบวนการส่ง และทางด้านรับตรวจจับสัญญาณองค์ประกอบร่วม $C(t)$ เพื่อที่จะคัดเลือกข้อมูล $m_1(t)$, $m_2(t)$ และ $m_3(t)$ ออกจากสัญญาณองค์ประกอบร่วม $C(t)$ ทางด้านรับจะทำการคูณ $C(t)$ ด้วย PN code ของข้อมูลแต่ละชุดที่ได้กำหนดไว้

$C(t)\ p_0(t)$ +3 +3 -1 +1 +1 -1 +1 -1 -1 -1 +1 -3 -1 -3 -1 -1 +3 +1 +1 +3 +1

$C(t)\ p_3(t)$ +3 +3 +1 +1 -1 +1 -1 -1 -1 +1 +1 +3 +1 +3 -1 -1 -3 +1 -1 -3 -1

$C(t)\ p_6(t)$ -3 -3 -1 +1 -1 -1 -1 +1 +1 -1 +1 +3 -1 +3 +1 +1 +3 +1 -1 +3 -1

หลังจากนั้นทางด้านรับจะทำการรวมค่าในแต่ละคาบของบิตที่ส่งมาได้ผลลัพธ์คือ $M_1(t)$, $M_2(t)$ และ $M_3(t)$

$C(t)\ p_0(t)$ +3 +3 -1 +1 +1 -1 +1 -1 -1 -1 +1 -3 -1 -3 +3 +3 -1 +1 +1 -1 +1

$M_1(t)$ +7 -9 +7

$C(t)\ p_3(t)$ +3 +3 +1 +1 -1 +1 -1 -1 -1 +1 +1 +3 +1 +3 -1 -1 -3 +1 -1 -3 -1

$M_2(t)$ +7 +7 -9

$C(t)\ p_6(t)$ -3 -3 -1 +1 -1 -1 -1 +1 +1 -1 +1 +3 -1 +3 +1 +1 +3 +1 -1 +3 -1

$M_3(t)$ -9 +7 +7

การตัดสินระดับของสัญญาณ $M_1(t)$, $M_2(t)$ และ $M_3(t)$ กฎการตัดสินที่ใช้คือ

$$\tilde{m}(t) = 1 \qquad \text{ถ้า } M(t) > 0$$

$$\tilde{m}(t) = -1 \qquad \text{ถ้า } M(t) < 0$$

หลังจากใช้กฎการตัดสินข้างต้นแล้วจะได้ว่า

$\tilde{m}_1(t)$ 1 -1 1 -

$\tilde{m}_2(t)$ 1 1 -1

$\tilde{m}_3(t)$ -1 1 1

2.11 Concluding Remark

เราสามารถอธิบายการออโต้คอร์รีเลชั่น (Autocorrelation) บนช่วงเวลาที่ไม่ต่อเนื่องของรหัส x ได้ตามสมการ

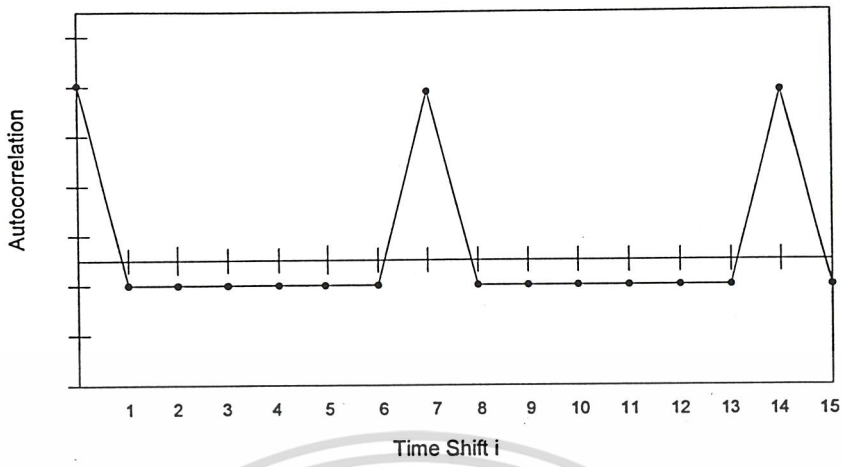
$$R_x(i) = \sum_{j=0}^{j-1} x_j x_{j-1} \tag{2.18}$$

แต่ละครั้งที่ทำการชิฟ i ครั้ง เราสามารถคำนวณการรวมผลที่ได้จากค่าของ x_j และรูปแบบที่ถูกชิฟของมันเอง x_{j-1} ตารางที่ 2.3 แสดงค่าที่ได้จากการทำ ออโต้คอร์รีเลชั่น $R_{p_0}(i)$ ของ PN code, p_0

ในรูปที่ 2.12 แสดงให้เห็นการทำออโต้คอร์รีเลชั่นจะให้ค่าสูงสุดทุก ๆ ครั้งที่ทำการชิฟเป็นครั้งที่ 7 ส่วนการชิฟในครั้งอื่นจะให้ค่าที่ต่ำที่สุดซึ่งมีค่าเท่ากับ -1 คุณสมบัติการออโต้คอร์รีเลชั่น แสดงในรูปที่2.12

i	$p_{0,j-i}$							$R_{p_0}(i)$
0	+1	-1	+1	+1	+1	-1	-1	+7
1	-1	+1	-1	+1	+1	+1	-1	-1
2	-1	-1	+1	-1	+1	+1	+1	-1
3	+1	-1	-1	+1	-1	+1	+1	-1
4	+1	+1	-1	-1	+1	-1	+1	-1
5	+1	+1	+1	-1	-1	+1	-1	-1
6	+1	+1	+1	+1	-1	-1	+1	-1
7	+1	-1	+1	+1	+1	-1	-1	+7
8	-1	+1	-1	+1	+1	+1	-1	-1
9	-1	-1	+1	-1	+1	+1	+1	-1
10	+1	-1	-1	+1	-1	+1	+1	-1
11	+1	+1	-1	-1	+1	-1	+1	-1
12	+1	+1	+1	-1	-1	+1	-1	-1
13	-1	+1	+1	+1	-1	-1	+1	-1
14	+1	-1	+1	+1	+1	-1	-1	+7
15	-1	+1	-1	+1	+1	+1	-1	-1

ตารางที่ 2.3 แสดงค่าที่ได้จากการทำ ออโต้คอร์รีเลชั่น $R_{p_0}(i)$ ของ PN code , p_0



รูปที่2.12 แสดงกราฟที่บันทึกค่าที่ได้จากการทำออโต้คอร์รีเลชั่น $R_{p_0}(i)$ ของ PN code, p_0

รูปที่ 2.12 เป็นรูปที่มีความสำคัญมากเพราะช่วยทำให้เข้าใจเรื่องการซิงโคไนเซชั่นของรหัส PN code ทางด้านรับ ผลของการคอรีเลชั่นที่สูงที่สุด (High correlation) จะปรากฏขึ้นเมื่อรหัสนั้นตรงกัน (เช่น เมื่อการชิฟครั้งที่ i เท่ากับ 0, Time shift = 0) แต่ถ้ารหัสไม่ตรงกันผลของการคอรีเลชั่นจะต่ำ

ในทางปฏิบัติ ทางด้านรับจะควบคุมสำเนาของรหัส PN code เดิม (Original copy PN code) ทางด้านรับต้องการที่จะได้ชุดรหัส $p_{0,j-i}$ ที่ใช้ตัดสินเฟส มีเพียงเฉพาะด้านรับเท่านั้นจะที่ต้อง "เลื่อน" (slide) ชุดของรหัสที่เข้ามาและคำนวณการออโต้คอร์รีเลชั่น เมื่อการทำออโต้คอร์รีเลชั่นได้ค่าสูงสุดแล้วรหัสทั้งสองชุดเป็นตัวเดียวกัน ในระบบ IS-95 CDMA สถานีโทรศัพท์มือถือจะทำการจัดการกับการออโต้คอร์รีเลชั่น โดยการรับ Pilot channel ที่ไม่ได้ถูกมอดูเลท รูปแบบนี้ยังสามารถนำมาใช้ได้ในกรณีที่ ความยาวของรหัสที่ทำการสเพรดเท่ากับช่วงคาบของข้อมูล

2.12 Autocorrelation and Cross-Correlation

ออโต้คอร์รีเลชั่น (Autocorrelation) คือ จำนวนของส่วนที่เหมือนกันระหว่างรหัสกับเฟสที่ชิฟของตัวมันเอง ออโต้คอร์รีเลชั่นเป็นการแสดง จำนวนของ Agreement ลบกับ Disagreement บนความยาวของรหัส (Length of the code) สำหรับการ

ออโต้คอร์รีเลชั่นเป็นตัวที่บรรยายถึง จำนวนของส่วนที่เหมือนกันระหว่าง รหัสกับเฟสที่ชิฟของตัวมันเอง ออโต้คอร์รีเลชั่นเป็นการแสดงจำนวนของ Agreement ลบกับ Disagreement บนความยาวของรหัสสำหรับการชิฟเฟสทุกแบบที่เป็นไปได้ การพล็อตกราฟออโต้คอร์รีเลชั่นของค่า Maximal code มีค่าเป็น -1 สำหรับทุกจุดที่มีการชิฟ ยกเว้นจุดที่ชิฟ 0 เมื่อรหัสเป็นแบบซิงโครไนเซชั่น ที่จุดนี้มีค่าเท่ากับ 2^{n-1} รูปแบบของ ค่าออโต้คอร์รีเลชั่นที่สูงนี้เป็นสิ่งที่จำเป็นสำหรับการซิงโครไนเซชั่นของรหัสที่ด้านรับ

ครอสคอรีเลชั่น (Cross-Correlation) คือ การวัดส่วนที่เหมือนกันระหว่างรหัส 2 รหัสที่แตกต่างกัน และเป็นการคำนวณในส่วนที่เหมือนกันกับออโต้คอร์รีเลชั่น การเกิดค่าครอสคอร์รีเลชั่นนับว่าของรหัสเป็นสิ่งที่สำคัญในระบบสเพรดสเปกตรัม เพื่อที่จะป้องกันการรบกวนที่ด้านรับระหว่างผู้ใช้งานในระบบการเข้าถึงแบบหลายทาง

2.13 ระบบเลขฐานสองแบบคิดเครื่องหมาย (Signed 2's Complement Number)

การทำงานของไมโครคอมพิวเตอร์จะใช้เลขฐานสองเท่านั้น ถ้าหากการคำนวณเลขฐานสองที่คิดเครื่องหมายจะต้องมี 1 บิตเพื่อเป็นตัวบ่งบอกเครื่องหมายของตัวเลขว่าเป็นเลขลบหรือเลขบวก ซึ่งบิตนี้จะอยู่ซ้ายสุด (MSB) ส่วนบิตที่เหลืออยู่จะบอกขนาด เช่น ถ้าเป็นเลขลบบิตนี้จะมีค่าเป็น "1" ส่วนที่เหลือขนาดจะถูกเก็บอยู่ในหน่วยความจำในรูปของ 2's คอมพลีเมนท์ฐานสอง ถ้าต้องการทราบขนาดจริงๆ ให้ทำเป็น 2's คอมพลีเมนท์อีกครั้งหนึ่งก็จะได้ขนาดที่แท้จริง กรณีเดียวกัน ถ้าเป็นเลขบวก เครื่องหมายจะมีค่าเป็น "0" ส่วนที่เหลือจะเป็นขนาดที่แท้จริง เลขระบบนี้เรียกว่า "ระบบเลขฐานสองแบบคิดเครื่องหมาย" (Signed 2's Complement)

<u>ตัวอย่าง</u> จงแปลง 00101101 ในระบบเครื่องหมายฐานลบสองเป็นเลขฐานสิบ

<u>วิธีทำ</u>

	0	0	1	0	1	1	0	1
บิตเครื่องหมาย		64	32	16	8	4	2	1

ขนาดจริงคือ 32 + 8 + 4 + 1 = 45

<u>ตัวอย่าง</u> ตัวบวกสูงสุดที่ 8 บิต ของเครื่องหมายเลขฐานสอง คือ 01111111

<u>วิธีทำ</u>

	0	1	1	1	1	1	1	1
บิตเครื่องหมาย		64	32	16	8	4	2	1

ขนาดจริงคือ 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127

<u>ตัวอย่าง</u> จงแปลง 10010011 ในเครื่องหมายเลขฐานสองเป็นเลขฐานสิบ

<u>วิธีทำ</u>

1 0 0 1 0 0 1 1

บิตเครื่องหมาย → ขนาดจริงต้องทำเป็น 2's Complement

Signed 2's Complement = 1 0 0 1 0 0 1 1

True Magnitude = 0 1 1 0 1 1 0 1

0	1	1	0	1	1	0	1
128	64	32	16	8	4	2	1

64 + 32 + 8 + 4 + 2 + 1 = 109

ตัวเลขจะเป็นลบ คำตอบคือ -109

<u>ตัวอย่าง</u> จงแปลง 11110000 ในเครื่องหมายเลขฐานสองเป็นเลขฐานสิบ

<u>วิธีทำ</u> 1 1 1 1 0 0 0 0

บิตเครื่องหมาย ↑ ค่าจริง →

Signed 2's Complement = 1 0 0 0 0 0 0 0

True Magnitude = 1 0 0 0 0 0 0 0

1	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

ตัวเลขจะเป็นลบ คำตอบ คือ -128

เพราะฉะนั้นเลขฐานสองที่คิดเครื่องหมายขนาด 8 บิต สามารถแทนด้วยเลขฐานสิบได้ทั้งหมด 256 ค่า ตั้งแต่ -128 ถึง 127

การบวกลบเลขฐานสองแบบคิดเครื่องหมายสามารถทำการบวกลบได้โดยตรง บิตเครื่องหมายที่เกิดขึ้น จะมีค่าเป็นจริงตามขนาดที่ทำการบวกลบ คือ ถ้าตัวตั้งและตัวบวกมีเครื่องหมายเหมือนกัน บิตเครื่องหมายจะเหมือนทั้งตัวตั้งตัวบวก เครื่องหมายต่างกัน บิตเครื่องหมายจะเหมือนบิตที่มีขนาดสูงสุด กรณีอื่นแสดงว่าผลลัพธ์ที่ได้จากการบวกลบเกิดโอเวอร์โฟล์ว (Overflow) แสดงว่าจำนวนบิตของขนาดไม่สามารถรองรับเก็บข้อมูลได้หมด ยกตัวอย่างเช่น การบวกลบเลขที่คิดเครื่องหมายขนาด 8 บิต สามารถแทนได้เลขฐานสิบได้เท่ากับ -128 ถึง +127 หากเป็นการบวกเลขที่คิดเครื่องหมายขนาด 16 บิต สามารถแทนได้เลขฐานสิบได้เท่ากับ -32768 ถึง +32767

<u>ตัวอย่าง</u> การบวกเลข 8 บิตที่คิดเครื่องหมาย Signed 2's Complement

01011001 + 10101101

<u>วิธีทำ</u>

0 1 0 1 1 0 0 1 (+88)

+ 1 0 1 0 1 1 0 1 (-83)

1 0 0 0 0 0 1 1 0 (+6)

↑ Ignore Overflow

<u>ตัวอย่าง</u> การบวกเลข 8 บิตที่คิดเครื่องหมาย Signed 2's Complement

1 0 1 1 0 0 1 + 1 0 1 0 1 1 0 1

<u>วิธีทำ</u>

1 1 0 1 1 0 0 1 (-39)

+ 1 0 1 0 1 1 0 1 (-83)

1 1 0 0 0 0 1 1 0 (-122)

Ignore Overflow ↑ ↑ Signbit — Magnitude bits ↓

Signed 2's Complement = 1 0 0 0 0 1 1 0

True Magnitude = 0 1 1 1 1 0 1 0

1	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

ตัวเลขจะเป็นลบ คำตอบคือ - (64 + 32 + 16 + 8 +2) = -122

ตรวจสอบ -39 + (-83) = -122

ในแต่ละตัวอย่างของการตรวจเช็คเลขฐานสองที่คิดเครื่องหมายแบบ 2's คอมพลีเมนท์ขนาด 8 บิต สามารถแทนเลขขนาดเลขที่คิดเครื่องหมายได้ตั้งแต่ -128 ถึง +127 เท่านั้น (บิตเครื่องหมาย 1 บิต และบิตขนาด 7 บิต) ถ้าหากได้ค่าน้อยหรือมากกว่าจะเกิดโอเวอร์โฟลว์ หมายถึง จำนวนบิตไม่พอเพียงสำหรับการเก็บขนาด ข้อควรสังเกตเมื่อมีการทดจากบิตที่ 7 ไปสู่บิตเครื่องหมายและบิตโอเวอว์โฟลว์ สามารถหาคำตอบที่ถูกต้องได้ดังนี้

1. ถ้าตัวทดจากคอลัมน์ที่ 7 ไปที่คอลัมน์ของบิตเครื่องหมายและมีโอเวอร์โฟลว์ คำตอบถูกต้อง
2. ถ้าตัวทดจากคอลัมน์ที่ 7 ไปที่คอลัมน์ของบิตเครื่องหมายและไม่มีโอเวอร์โฟลว์ คำตอบถูกต้อง
3. ถ้าไม่มีตัวทดจากคอลัมน์ที่ 7 ไปที่คอลัมน์ของบิตเครื่องหมายและไม่มีโอเวอร์โฟลว์ คำตอบผิด

<u>ตัวอย่าง</u> การลบเลข 8 บิตที่คิดเครื่องหมาย Signed 2's Complement

10000101 – 01111111

<u>วิธีทำ</u> ตัวตั้งยกมาได้เลย ส่วนตัวลบทำเป็น 2's Complement ก่อนแล้วนำไปบวกกับตัวตั้ง

1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 (-123)

- <u>0 1 1 1 1 1 1 1 1</u> 2's + <u>1 0 0 0 0 0 0 0 1</u> -(+127)

1 0 0 0 0 0 0 1 1 0 (250)

ผลลัพธ์คือ +6 แต่ควรเป็น -250 ซึ่งไม่มีตัวทดที่บิตเครื่องหมาย แต่มีเวอร์โฟลว์ เพราะผลต่างไม่ถูกต้อง ความผิดพลาดเกิดขึ้นเพราะคำตอบที่แท้จริงมากเกิน 8 บิต ของระบบฐานลบสอง

<u>ตัวอย่าง</u> การลบเลข 8 บิตที่คิดเครื่องหมาย Signed 2's Complement ของตัวเลข 01111110 + 00111101

<u>วิธีทำ</u>

0 1 1 1 1 1 1 0 = +126

+ 0 0 1 1 1 1 0 1 = +61

0 1 0 1 1 1 0 1 1 ไม่เท่ากับ 187

↑ ไม่มีโอเวอโฟลว์

ผลลัพธ์จะเป็นค่าลบ ตัวคอลัมน์ที่ 7 ที่บิตเครื่องหมายมีการเปลี่ยนเป็น 1 ไม่มีโอเวอโฟลว์เกิดขึ้น เนื่องจากผลลบที่เกิดขึ้นไม่ถูกต้องเพราะ +187 มีค่ามากกว่า 8 บิต

2.14 Counter shift register

เป็น 2 class ที่สำคัญของ sequential circuits อาจจะอธิบายแบบง่ายๆได้คือ

- counter คือ วงจรที่ทำการนับ pulse ซึ่ง base counter จะสามารถพัฒนาให้ใช้ร่วมกับฟังก์ชันต่างๆได้ เช่น Synchronous และ Asynchronous

- ชิฟรีจีสเตอร์(Shift register) คือ วงจรที่ทำการเก็บและเลื่อน Data ซึ่งเราสามารถใช้ในการทำการส่งข้อมูลแบบ Serial, Serial/parallel conversion

<u>หมายเหตุ</u> : โดยทั้ง counter, register จะสามารถสร้างได้โดยใช้ Circuit Schematic, VHDL, LPM

2.14.1 หลักการพื้นฐานของ Digital Counter

- Counter : เป็นวงจร digital sequential ซึ่งมีลำดับของ output เป็นรูปแบบที่ซ้ำกันลามารถคาดเดาได้โดยทำงานตาม clock pulse
- Recycle : เพื่อทำการ Transition จาก state สุดท้ายของลำดับการนับไปยัง state แรก
- Count sequence : ลำดับขั้นของ output state ในแต่ละ counter
- State diagram : เป็น diagramที่โชว์ลำดับขั้นของวงจร state sequential
- Modulus: จำนวนครั้งของ state ที่ผ่านมาถึง counter
- Modulo – n (or mod-n) : counter ที่มีค่า โมดูโล (Modulo) n ค่า
- Up counter : counter ที่มี ascending sequence
- Down counter : counter ที่มี decending sequence

ลำดับ Output ของ counter จะสามารถบอกได้จากค่า modulus ของมัน (คือจำนวน state มันเองซึ่ง counter ทำการทำงาน)

Up counter ที่มีค่า modulus 12 จะทำการนับ 12 states จาก

0000 1011 (0 ถึง 11 ในเลขฐาน 10) และจะกลับไปเริ่มที่ 0000 ใหม่ และทำการวน loop อีกครั้ง

Down counter ที่มีค่า modulus 12 จะทำการนับจาก

1011 0000 และจะไปเริ่มที่ค่า 1011

**(ซึ่งทั้ง 2 แบบจะถูกเรียกว่า Modulus-12 หรือ Mod-12)

2.14.2 แผนภาพลำดับสถานะ (State Diagram)

สถานะของ Counter จะถูกแทนโดย State diagram ซึ่งจะเป็นตัวเปรียบเทียบ Mod-12 Up counter กับสัญญาณนาฬิกาซึ่งในแต่ละ Counter state จะประกอบด้วยวงกลมที่มีค่า Binary อยู่ ขบวนการจะเป็นไปตาม

จะพบว่า หน้าปัดนาฬิกา กับ Counter state diagram จะทำงานเหมือนกันกล่าวคือ จะทำงานเป็นระบบปิด หากทำงานจนครบรอบแล้วจะเริ่มต้นใหม่อีกครั้งหนึ่ง

2.14.3 Number of Bits and Maximum Modulus

- Maximum modulus (m_{max})

  ค่าที่มากที่สุดที่สามารถแสดงได้ด้วย n-bit ของ Counter ($m_{max} = 2^n$)

- Full-sequence counter

  Counter ซึ่งค่า modulus จะมีค่าเท่ากับ Maximum modulus ($m = 2^n$ สำหรับ n-bit counter)

- Binary counter

  Counter ที่ให้สร้าง Binary count sequence

- Truncated-sequence counter

  Counter ซึ่งค่า modulus มีค่าน้อยกว่าค่า Maximum modulus ($m < 2^n$)

2.14.4 Counter-Sequence Table & Timing Diagram

เราสามารถแสดงลำดับการนับ(Count sequence) โดยวิธี Count sequence table (คือการนับค่าของ counter state มาเรียงตามลำดับของการนับ) หรือ โดยวิธี Timing diagram

เราสามารถหาค่า Timing diagram จากตารางเหล่านี้ ซึ่งจะพบว่าการเปลี่ยน state ของ counter แต่ละตัวจะขึ้นอยู่กับ clock pulse จะเห็นได้จากตัวอย่างของ Mod-16 Up counter

Q_0 - จะเรียง state ในแต่ละ clock pulse

Q_1 - จะเรียง state ในทุกๆ 2 clock pulse

Q_2 - จะเรียง state ในทุกๆ 4 clock pulse

Q_3 - จะเรียง state ในทุกๆ 8 clock pulse

2.14.5 Synchronous Counter

ความหมาย :

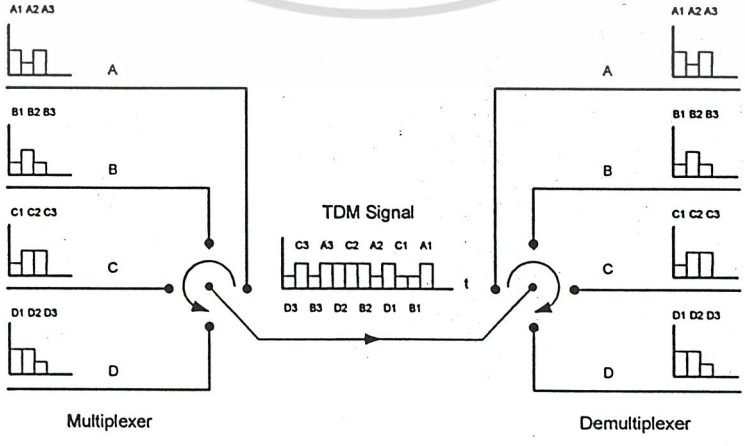
- Synchronous Counter : เป็น Counter ที่ flip-flop ใช้สัญญาณนาฬิการ่วมกันเพื่อความเป็นจังหวะพร้อมกัน (synchronization)
- Present State : state ปัจจุบันของ flip-flop outputs ในระบบ synchronous sequential circuit
- Next State : state ถัดไปของ flip-flop ใน synchronous sequential circuit หลังจากได้รับ clock pulse
- Memory section : set ของ flip-flop ใน synchronous circuit ที่มี state ปัจจุบันอยู่
- Control section : เป็นส่วน combination logic ของ synchronous circuit ที่เป็นตัวตัดสิน state ถัดไปของวงจร

2.15 การมัลติเพล็กซ์แบบแบ่งเวลา

การแปลงสัญญาณอนาลอกให้อยู่ในรูปดิจิตอล ทำให้สามารถที่จะรวมสัญญาณจากหลายๆแหล่งเข้าด้วยกันได้โดยง่ายเทคนิควิธีที่ใช้กัน คือ การมัลติเพล็กซ์ข้อมูลเข้าด้วยกันในกรอบของเวลาเรียกวิธีการมัลติเพล็กซ์แบบนี้ว่า TDM (Time Division Multiplexing) ซึ่งในที่นี้จะกล่าวถึงหลักการเบื้องต้น วิธีการซิงโครไนซ์ กระบวนการตัดสินใจในการควบคุมสถานะการซิงโครไนซ์ และการเลือกใช้เฟรม อไลเมนต์ เวอร์ด์

2.15.1 หลักการเบื้องต้นของการมัลติเพล็กซ์แบบแบ่งเวลา

ระบบ TDM เป็นระบบสื่อสารดิจิตอลที่ใช้ส่งข้อมูลจากหลายๆ ช่องข้อมูล ให้เป็นสัญญาณดิจิตอลเดียวโดยใช้วิธีการแบ่งช่วงเวลาที่แน่นอนค่าหนึ่งให้แต่ละช่องข้อมูลเรียงลำดับกันไป ดังรูปที่ 2.13 เรียกช่องเวลาที่กำหนดให้แต่ละช่องข้อมูลว่า “ช่องเวลา” (Time Slot) และถ้าแต่ละช่องเวลาประกอบด้วยข้อมูลเพียง 1 บิต เรียกว่าการมัลติเพล็กซ์ข้อมูลแบบ บิตอินเตอลีฟ (Bit Interleaved) แต่ถ้าประกอบด้วยกลุ่มของบิตหรือเวอร์ด เรียกว่า เวอร์ดอินเตอลีฟ (Word Interleaved)



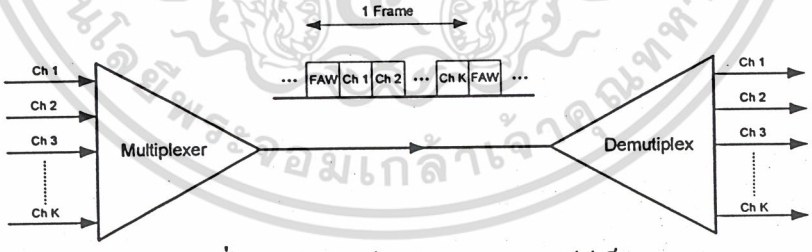
รูปที่ 2.13 ภาพสมมูลแสดงตัวส่ง (Multiplexer) และตัวรับ (Demultiplexer) ระบบทีดีเอ็ม

การสื่อสารระบบทีดีเอ็มสามารถแบ่งออกเป็น 2 ระบบใหญ่ๆ คือ

1. ซิงโครนัส ทีดีเอ็ม (Synchronous TDM) เป็นระบบที่ใช้ แหล่งกำเนิดข้อมูล และตัวมัลติเพล็กซ์ข้อมูลทำงานที่สัญญาณนาฬิการเดียวกัน ทำให้การส่งและรับข้อมูลระหว่างแหล่งข้อมูล และตัวมัลติเพล็กซ์เป็นไปในจังหวะเดียวกัน หรือกล่าวได้ว่า ทุกๆครั้งที่ตัวมัลติเพล็กซ์เข้ามาอ่านข้อมูลแหล่งข้อมูล

2. อะซิงโครนัสทีดีเอ็ม (Asynchronous TDM) เป็นระบบทีแหล่งกำเนิดข้อมูลและตัวมัลติเพล็กซ์ทำงานที่สัญญาณนาฬิกาต่างกันทำให้การส่ง และรับข้อมูลระหว่างแหล่งข้อมูลและตัวมัลติเพล็กซ์ไม่เป็นไปในจังหวะเดียวกันเช่น จังหวะที่ตัวมัลติเพล็กซ์เข้ามาอ่านข้อมูลแหล่งข้อมูลอาจจะไม่พร้อมที่จะให้ข้อมูลเป็นผลให้ข้อมูลในช่องข้อมูลนั้นๆ มักจะไม่ค่อยมีความต่อเนื่องจึงทำให้การสื่อสารในระบบนี้จะต้องมีกระบวนการ จัสติฟิเคชั่น (Justification) หรือ พัลซ์สตัฟฟิ่ง (Pulse Stuffing) เพื่อทำการซิงโครไนซ์สัญญาณนาฬิการของแหล่งข้อมูลกับสัญญาณนาฬิการของตัวมัลติเพล็กซ์ การสื่อสารแบบนี้จึงมีความยุ่งยากซับซ้อนกว่าแบบซิงโครนัสทีดีเอ็ม แต่เป็นแบบที่มีความยืดหยุ่นมากกว่าในการเชื่อมต่อแหล่งข้อมูลที่มีอัตราข้อมูลต่างๆ กัน

ในการสื่อสารระบบทีดีเอ็มสิ่งหนึ่งที่สำคัญที่สุดคือ การซิงโครไนซ์ระหว่างตัวมัลติเพล็กซ์ และตัวดีมัลติเพล็กซ์ จากรูป 2.13 จะเห็นว่าการที่ตัวมัลติเพล็กซ์สามารถแยกข้อมูลที่รวมกันมาเป็นสัญญาณทีดีเอ็มไปยังช่องข้อมูลที่ถูกต้องของมันนั้น จำเป็นอย่างยิ่งที่ตัวดีมัลติเพล็กซ์จะต้องอยู่ในสภาพซิงโครไนซ์กับตัวมัลติเพล็กว์ พิจารณากันอย่างง่ายๆ หมายความว่า สวิตซ์ทางตัวส่งและตัวรับต้องอยู่ในตำแหน่งเดียวกันและหมุนไปพร้อมๆกัน วิธีการที่ใช้กันในทางปฏิบัติคือ การกำหนดช่องเวลาขึ้น 1 ช่อง (หรือ 2 ช่อง) ซึ่งบรรจุเวอร์ดที่มีลักษณะจำเพาะ ช่องเวลานี้กำหนดขึ้นเพื่อเป็นช่องเวลาอ้างอิงเพื่อใช้ระบุตำแหน่งของช่องเวลาอื่นๆ ดังนั้นถ้าตัวดีมัลติเพล็กซ์ สามารถตรวจสอบพบเวอร์ดนี้ได้ตัวดีมัลติเพล็กซ์ก็สามารถทราบตำแหน่งของช่องเวลาอื่นๆ ทำให้สามารถแยกแยะข้อมูลในช่องเวลาต่างๆออกมาได้ เรียกเวอร์ดที่บรรจุอยู่ในช่องเวลาอ้างอิงนี้ว่าเฟรมอไลเมนต์เวอร์ด FAW (Frame Alignment word) ดังรูปที่ 2.14



รูปที่ 2.14 โครงสร้างเฟรมของระบบทีดีเอ็ม

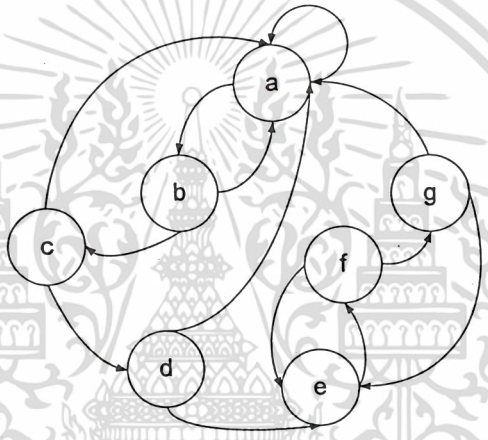
สำหรับอัตราความเร็วสามารถคำนวณได้คือ ถ้าให้ระบบมีช่องข้อมูลที่ต้องการส่ง K ช่องและเฟรมอไลเมนต์เวอร์ด FAW 1 ช่องเวลา ดังนั้นใน 1 เฟรมข้อมูลประกอบด้วยช่องเวลา K+1 ช่องเวลา และกำหนดให้ข้อมูลในแต่ละช่องข้อมูลได้จากการสุ่มสัญญาณอนาลอก ด้วยความถี่ f_s ความละเอียด m บิต ต่อการสุ่ม 1 ครั้ง ดังนั้นอัตราความเร็วบิตของสัญญาณมัลติเพล็กซ์ F_o เป็น

$$F_o = mf_s(K+1) \tag{2.19}$$

2.15.2 การซิงโครไนซ์เฟรมข้อมูล

เนื่องจากข้อมูลที่ส่งมาในช่องข้อมูลต่าง ๆ นั้น มีลักษณะเป็น แรนดอม คือ อาจมีรูปแบบข้อมูลเป็นลักษณะใดๆก็ได้ ดังนั้นจึงเป็นไปได้ที่จะเกิดเวอร์ดข้อมูลที่มีลักษณะเหมือนกับ FAW ได้เรียกเวอร์ดข้อมูลที่มีลักษณะเหมือน FAW นี้ว่า FAW เทียม ประกอบกับข้อมูลที่ส่งจากต้นทางไปถึงปลายทางมีโอกาสผิดพลาดได้เสมอ ดังนั้นจึงจำเป็นต้องมีกระบวนการ หรือขั้นตอนที่ใช้ในการตัดสินใจว่า FAW ที่ตรวจพบนั้นเป็น FAW ที่แท้จริงหรือไม่ และเมื่อใดที่จะถือว่าระบบอยู่ในสภาวะการซิงโครไนซ์ แล้วกระบวนการที่ใช้ในการตัดสินใจและดำเนินการในสิ่งเหล่านี้เรียกว่า การอไลเมนต์เฟรม (Frame Alignment)

จากเหตุผลที่กล่าวข้างต้นนั้น การอไลเมนต์เฟรมจึงจำเป็นต้องกำหนดสถานะต่างๆ ในการทำงานดังรูป 2.15



รูปที่ 2.15 ไดอะแกรมแสดงสถานะอไลเมนต์

สถานะ a เป็นสถานะอไลเมนต์เฟรมสมบูรณ์ระบบอยู่ในสภาพล็อก (Synchronize)

สถานะ b,c,d เป็นสถานะอไลเมนต์เฟรมชั่วคราว ระบบจะอยู่ในโหมดการตรวจสอบ (Check Mode)

สถานะ e เป็นสถานะของการไม่อไลเมนต์เฟรม ระบบจะอยู่ในโหมดการค้นหา FAW (Search Mode)

สถานะ f,g เป็นสถานะรอ ระบบจะอยู่ในโหมดการค้นหาและการตรวจสอบ (Search/Check Mode)

FAW

จากไดอะแกรมในรูปที่ 2.15 สามารถแบ่งโหมดการทำงานเป็น 2 โหมด ใหญ่ๆ คือ

1. โหมดซิงโครไนซ์ ประกอบด้วย 4 สถานะย่อย คือ

สถานะ a เป็นสถานะที่ระบบอยู่ในสภาพการอไลเมนต์สมบูรณ์

สถานะ b เป็นสถานะที่ระบบตรวจไม่พบ FAW ในเฟรมที่ n

สถานะ c เป็นสถานะที่ระบบตรวจไม่พบ FAW ในเฟรมที่ n +1

สถานะ d เป็นสถานะที่ระบบตรวจไม่พบ FAW ในเฟรมที่ n +2

กล่าวโดยสรุป สำหรับในโหมดซิงโครไนซ์ คือ เมื่อระบบอยู่ในสภาพการซิงโครไนซ์แล้ว (สถานะ a) ถ้ามีการตรวจไม่พบ FAW ในจุดที่กำหนดไว้ 4 เฟรมติดต่อกัน ระบบจะเข้าสู่โหมดค้นหา (สถานะ e) แต่ถ้ามีการตรวจพบ FAW เพียงเฟรมใดเฟรมหนึ่งระบบก็จะกลับเข้าสู่สถานะ a ใหม่ การกำหนดให้มีสถานะ b,c,d ทำให้ เถียรภาพของระบบมีความมั่นคงมากขึ้น ผลของความผิดพลาดของข้อมูลนี้ จะมีผลต่อสภาพการซิงโครไนซ์ลดลง และยังทำให้ระบบไม่หลุดจากสภาพการซิงโครไนซ์ง่ายๆได้

2. โหมดค้นหา ประกอบขึ้นจาก 3 สถานะย่อย คือ

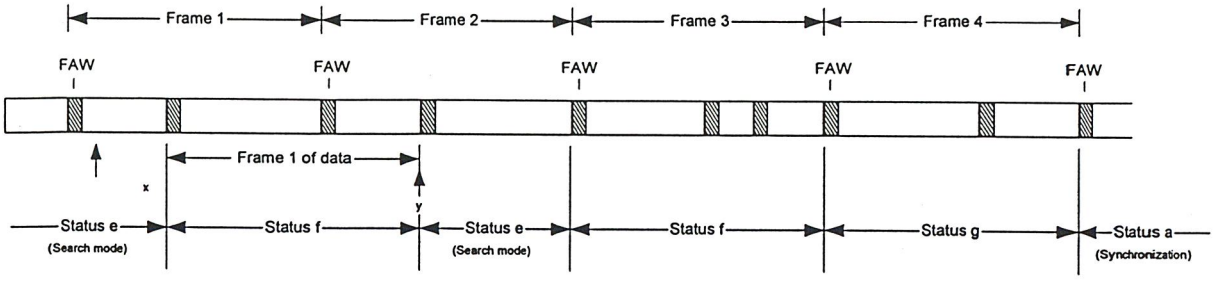
สถานะ e เป็นสถานะที่ระบบอยู่ในสภาพค้นหา FAW

สถานะ f เป็นสถานะที่ระบบตรวจพบ FAW ในเฟรมที่ 0

สถานะ g เป็นสถานะที่ระบบตรวจพบ FAW ในเฟรมที่ 1

กล่าวโดยสรุปสำหรับในโหมดค้นหาคือ เมื่อระบบอยู่ในสภาพการค้นหา (สถานะ e) ถ้ามีการตรวจพบ FAW ติดต่อกัน 3 เฟรม ระบบจะเข้าสู่โหมดการซิงโครไนซ์ เป็นการถือได้ว่า FAW ที่พบนั้นเป็น FAW ที่แท้จริง แต่ถ้ามีเฟรมใดเฟรมหนึ่งตรวจแล้วไม่พบ FAW ระบบจะกลับเข้าสู่สถานะ e เพื่อค้นหา FAW ใหม่ ซึ่งแสดงว่า FAW ตัวแรกที่พบนั้นไม่ใช่ FAW ที่แท้จริง

ความจริงแล้วเทคนิคของการอไลเมนต์เฟรมมีหลายวิธีเช่น การอไลเมนต์เฟรมแบบอนุกรม (Serial Frame Alignment) หรือว่า การอไลเมนต์แบบขนาน (Parallel Frame Alignment) ซึ่งจะอไลเมนต์เฟรมได้รวดเร็วกว่าแบบอนุกรม แต่วิธีการอไลเมนต์แบบอนุกรมเป็นวิธีการที่ง่ายที่สุดและใช้กันมากที่สุด ดังรูปที่ 3.15 I1, I2, I3 และ I4 เป็น FAW เทียม จุด x เป็นจุดเริ่มต้นการทำงาน และกำหนดให้สภาวะเริ่มต้นของตัวดีมัลติเพล็กซ์อยู่ที่สถานะ e คือ สถานะการอไลเมนต์เฟรม (เช่นตอนเปิดเครื่อง) วิธีการอไลเมนต์เฟรมใช้หลักการที่ว่า FAW แท้จริงจะปรากฏที่ตำแหน่งเดิมของทุก ๆ เฟรม ส่วน FAW เทียมจะไม่ปรากฏที่ตำแหน่งใด ๆ อย่างถาวร ดังนั้นเมื่อตรวจพบ FAW ที่ตำแหน่งเดิมของทุกๆเฟรม ส่วน FAW เทียมจะไม่ปรากฏที่ตำแหน่งใด ๆ อย่างถาวรดังนั้นเมื่อตรวจพบ FAW ที่ตำแหน่งใด ๆ แล้ว ระบบจะต้องไปตรวจสอบที่อีกครั้งที่ตำแหน่งเดิมของเฟรมต่อไป กระบวนการอไลเมนต์เฟรมจะเริ่มที่จุด x โดยการเริ่มตรวจสอบ F บิตแรก ถ้าไม่ตรงกับ FAW ที่ตั้งค่าไว้ก็จะทำการตรวจสอบเวอร์ดต่อไป โดยการเลื่อนไปอีก 1 บิต จากเวอร์ดหลังสุดที่ทดสอบ และกระบวนการจะเป็นเช่นนี้ไปเรื่อย ๆ จนกว่าจะพบเวอร์ดที่เหมือน FAW



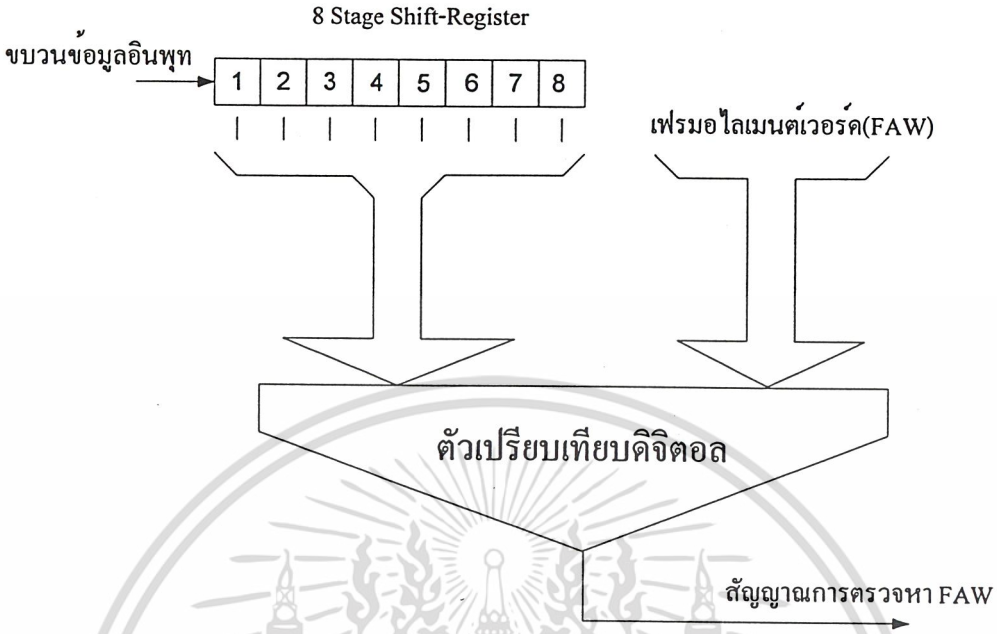
รูปที่ 2.16 ตัวอย่างการซิงโครไนซ์เฟรม

จากจุด x เป็นต้นไปเวอร์ดแรกที่เหมือน FAW คือ I1 ซึ่งเป็น FAW เทียม เมื่อระบบพบ I1 ระบบจะเปลี่ยนสถานะจาก e เป็นสถานะ f (ดูรูปที่ 2.15 ประกอบ) จากนั้นกระโดดไปตรวจสอบ FAW อีกครั้งที่จุด y เพื่อตรวจสอบว่า I1 ที่ตรวจพบนั้นเป็น FAW ที่แท้จริงหรือไม่และช่วงระหว่างจุด x และ y จะไม่มรการตรวจสอบใด ๆ ทั้งสิ้น จากจุด y เมื่อตรวจสอบแล้วปรากฏว่าไม่พบ FAW จึงสรุปได้ว่า I1 ที่พบนั้น ไม่ใช่ FAW ที่แท้จริง และระบบก็จะเปลี่ยนสถานะจาก f กลับมาที่ e ใหม่เพื่อทำการเริ่มต้นหา FAW ใหม่

จากจุด y FAW ที่พบตัวต่อไปคือ FAW ของเฟรม 3 ซึ่งเป็น FAW ที่แท้จริงระบบจะเปลี่ยนสถานะจาก e ไปสถานะ f ใหม่ เช่นเดียวกันระบบจะกระโดดไปตรวจสอบอีกครั้งในเฟรมต่อไปซึ่งจะพบ FAW ของเฟรม 4 และ5 ทำให้สถานะของระบบเปลี่ยนจากสถานะ f ไป g และเข้าสู่สภาพซิงโครไนซ์ในสถานะ a ตามลำดับ จะเห็นว่าเพียงเริ่มต้นพบ FAW ที่แท้จริงเท่านั้นระบบก็จะเข้าสู่สถานะการซิงโครไนซ์ในที่สุด ข้อสังเกตุ คือ เพียงแค่ระบบค้นพบ FAW ที่แท้จริงเท่านั้น FAW เทียมที่เกิดขึ้นภายในเฟรม (I2, I3, I4) จะไม่มีผลต่อระบบ

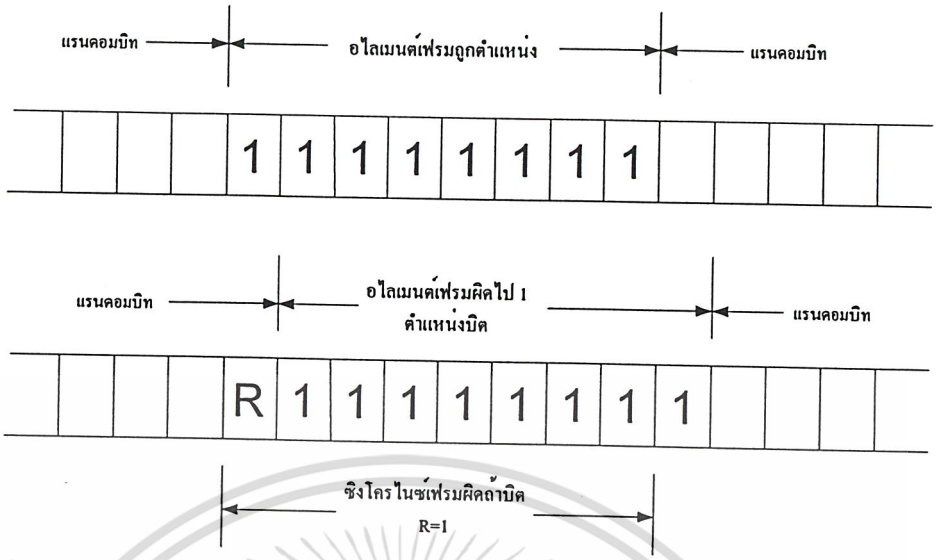
2.15.3 เฟรมอไลเมนต์เวอร์ด

ปัญหาประการหนึ่งของการอไลเมนต์เฟรมคือ ยิ่งปรากฏมี FAW เทียมมากก็ยิ่งทำให้ระบบใช้เวลามากขึ้นในการซิงโครไนซ์เฟรม เป็นผลให้สมรรถนะของระบบซิงโครไนซ์เฟรมต่ำลง นั่นหมายถึง สมรรถนะโดยรวมของระบบต่ำลงด้วย ดังนั้นการเลือกใช้รหัส FAW ที่เหมาะสมนั้นจึงสามารถเพิ่มสมรรถนะของระบบการซิงโครไนซ์แฟรมได้ สาเหตุเนื่องจากมีรหัสบางค่าที่สามารถถูกเลียนแบบได้ง่าย ดังนั้นจึงมีโอกาสพบ FAW เทียมได้มาก



รูปที่ 2.17 แสดงไดเอแกรมการตรวจหาเฟรมอไลเมนต์เวอร์ด (FAW)

โดยปกติแล้วการตรวจหา FAW จะทำโดยการเลื่อนขบวนข้อมูลที่รับได้เข้าสู่ชิฟรีจิสเตอร์ขนาด 8 สเตจ (กรณี FAW เป็นขนาด 8 บิต) โดนเลื่อนเข้าทีละ 1 บิต ข้อมูลและนำเอาท์พุทจากทุกๆสเตจของชิฟรีจิสเตอร์เข้าสู่วงจรเปรียบเทียบเพื่อเปรียบเทียบเฟสกับรหัส FAW ที่ได้กำหนดไว้แล้วดัง รูป 2.17 เมื่อใดก็ตามที่เวอร์ดใดๆ ที่มีรูปเหมือน FAW ถูกเลื่อนมาปรากฏบนรีจิสเตอร์ทั้ง 8 ตัว วงจรเปรียบเทียบจะกำเนิดสัญญาณตรวจหา FAW ออกมาทันที เพื่อแสดงให้เห็นผลของการเลือกรหัส FAW ที่ไม่เหมาะสม จะกำหนดให้ขบวนข้อมูลที่รับได้มีลักษณะเป็นแรนดอม และใช้รหัส FAW เป็น 11111111 เนื่องจากข้อมูลมีลักษณะเป็นแรนดอม ดังนั้นโอกาสที่ข้อมูลที่ปรากฏบนชิฟรีจิสเตอร์จะมีรหัสเหมือน FAW จะเป็น 0.5^F โดย F เป็นจำนวนบิตของ FAW เวอร์ด พิจารณาเมื่อ 7 บิตแรกของ FAW เวอร์ด ถูกเลื่อนเข้ามา ในชิฟรีจิสเตอร์ดังรูป 2.18 พบว่าในสภาพเช่นนี้โอกาสที่ FAW จะถูกเลียนแบบจะมีค่าถึง 0.5 แทนที่จะเป็น 0.5^8 เนื่องจาก 7 บิตแรกที่ถูกเลื่อนเข้าไปในชิฟรีจิสเตอร์เหมือนกับ 7 บิตหลังของรหัส FAW ที่ถูกเลื่อนบิต วิธีที่จะลดผลปรากฏการนี้ คือ การเลือกรหัสที่เกิดการเลียนแบบตัวเองได้น้อยที่สุด คุณสมบัติเชิงคณิตศาสตร์ที่แสดงความสัมพันธ์นี้ คือ ค่าออโต้คอร์รีเลชัน ซึ่งมีนิยามดังนี้



รูปที่ 2.18 การซิงโครไนเฟรมผิดเนื่องจากการเลือก FAW ที่ไม่เหมาะสม

เมื่อกำหนดให้ลำดับของรหัส FAW เป็น $x_1, x_2, x_3, ..., x_m$ ดังนั้นค่าออโต้คอร์รีเลชั่น $R_x(k)$ จะได้

$$R_x(k) = \sum_{i=1}^{m=k} x_1 \cdot x_{i+k} \quad , x_i = \pm 1 \tag{2.20}$$

โดยแทน ลอจิก 1 ด้วยค่า +1

ลอจิก 0 ด้วยค่า -1

รหัสที่มีคุณสมบัติจะต้องมีค่า $R_x(k)$ น้อยที่สุดเมื่อ $k \neq 0$ เหมาะสำหรับการซิงโครไนซ์ควรจะมีเงื่อนไขค่าออโต้คอร์รีเลชั่นดังนี้

$$|R_x(k)| \leq 1 \quad , k \neq 0 \tag{2.21}$$

เรียกรหัสที่มีคุณสมบัติตามสมการ (2.21) นี้ว่า รหัสบาร์เกอร์ (Barker Code) แต่อย่างไรก็ดีรหัสที่มีคุณสมบัติที่ดีสำหรับการซิงโครไนซ์ไม่จำเป็นที่จะต้องมีคุณสมบัติตามสมการ (2.21) เสมอไป ตัวอย่างรหัสบาร์เกอร์ เช่น 110, 1110010, 11100010010 ซึ่งค่าคอมพลีเมนต์ ค่าสะท้อน (การกลับรหัสจากทางซ้ายมายังทางขวา) หรือคอมพลีเมนต์ของค่าสะท้อนของรหัสบาร์เกอร์ล้วนมีคุณสมบัติตามสมการ (2.21)

2.16 มาตรฐานของ CCITT International E1

รูปแบบเฟรมของ CCITT International ถูกนำมาใช้ในหลายประเทศ (ยุโรป. อเมริกากลางและใต้ ฯลฯ) ประสิทธิภาพนี้ทำงานที่ 2.048 Mbps

เฟรมรูปแบบนี้ มีการกำหนดไว้ใน CCITT คือ G.704 และมี G.732 จะเป็นภาคผนวกของ G.704

G.704 : การใช้โครงสร้าง frame แบบ Synchronous และ Hierarchical Level ระดับ Primary และ Secondary

G.732 : ลักษณะพิเศษของ Primary PCM Multiplex Equipment ที่ทำงานที่ 2048 KBPS

2.16.1 โครงสร้างของ G.732/G.704

เฟรมในแบบมาตรฐานมี 32 timeslots แต่ละ timeslot จะประกอบด้วย 8-bit byte ถ้าเป็น Multiframe จะประกอบด้วย 16 frames ตั้งแต่หมายเลข 0 ถึง 15 ส่วน Timeslot มีตั้งแต่หมายเลข 0 ถึง 31 Timeslot หมายเลข 0 ใช้สำหรับ

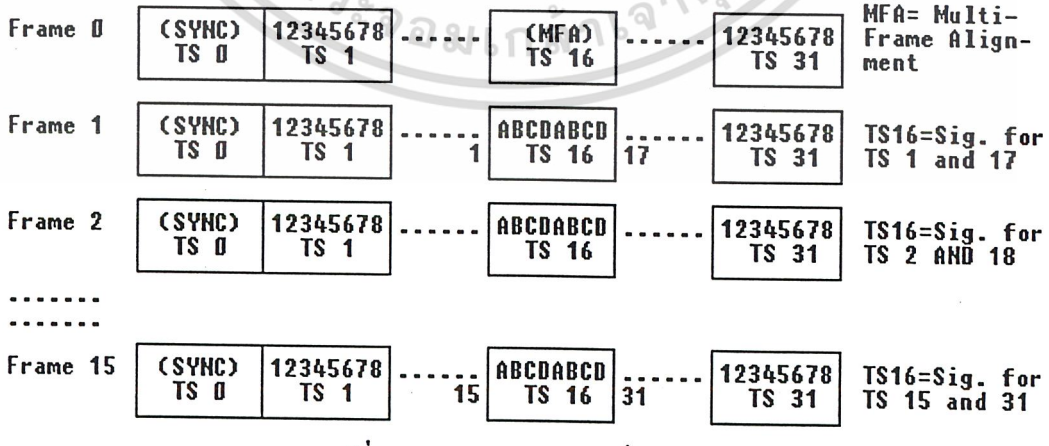
- Synchronization
- Alarm Transport (แจ้งเตือนการส่ง)
- International Carrier use (ใช้เป็น carrier แบบ International)

Timeslot 16 อาจจะถูกใช้เป็นสื่อส่งข้อมูลสัญญาณที่มีการใช้ช่องสัญญาณร่วม (Channel Associated Signaling (CAS))

**<u>ข้อสังเกต</u> : G.732 จะไม่ถูกกำหนดให้อยู่ในสถานะ Signaling จะมีเพียงการส่งของสถานะเท่านั้นที่ผ่านเฟรม G.732

อย่างไรก็ตาม G.704 จะรับรองข้อกำหนดของช่องสัญญาณทั่วไป และยอมให้มีการส่งแบบ TRANSPARENT End-To-End ของ timeslot 16 อีกด้วย

<u>เฟรมที่ 0 ถึง 15 :</u>



รูปที่ 2.19 รูปแบบของเฟรมที่ 0 ถึง 15

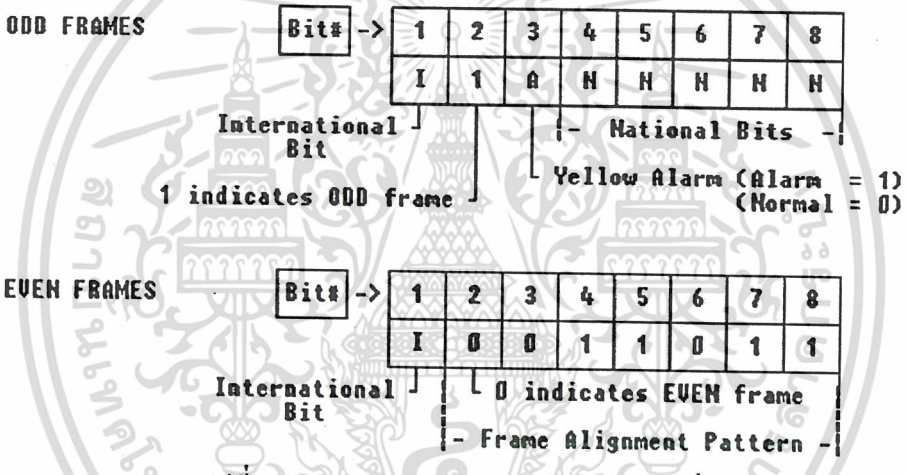
2.16.2 การทำงานของ Timeslot 0

ในแอพพลิเคชั่นปกติทั่วไป ไม่มีคุณสมบัติ ERROR CHECKING ในข้อกำหนดของ G.704 ซึ่งในกรณีเช่นนี้ timeslot 0 จะจัดเตรียมความสามารถในการส่งบิตแบบ International และ National

อย่างไรก็ตาม G.704 มีการทำงานแบบ Optional (เลือกใช้หรือไม่ก็ได้) เมื่อตัว CRC-4 checksum สามารถถูกนำมาใช้สร้าง Error Detection สำหรับเฟรม 0 ถึง 7 และเฟรม 8 ถึง 15 ได้ เมื่อรูปแบบ Optional นี้ถูกใช้ International bit ของ even frame (เฟรมเลขคู่) จะถูกแทนที่ด้วย CRC-4 bit ส่วน National bit จะถูกตั้งชื่อใหม่ว่า Spare bits

2.16.3 การทำงานแบบปกติบน Timeslot 0 (ไม่มี CRC-4)

ความหมายของข้อมูลจะถูกเก็บอยู่ใน Timeslot 0 byte ขึ้นอยู่กับว่าหมายเลขของเฟรมเป็นเลขคู่หรือเลขคี่ :

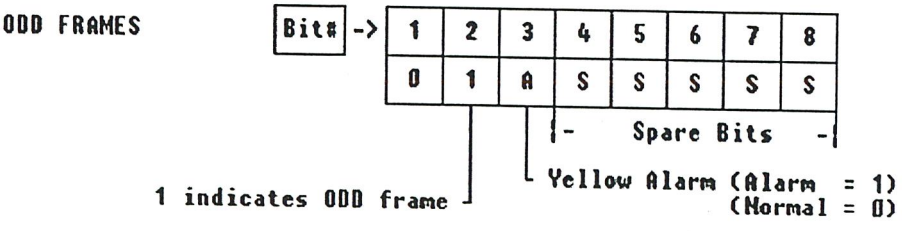


รูปที่ 2.20 รูปแบบของเฟรมเลขคู่และเฟรมเลขคี่

2.16.4 การทำงานแบบปกติบน Timeslot 0 (มี CRC-4)

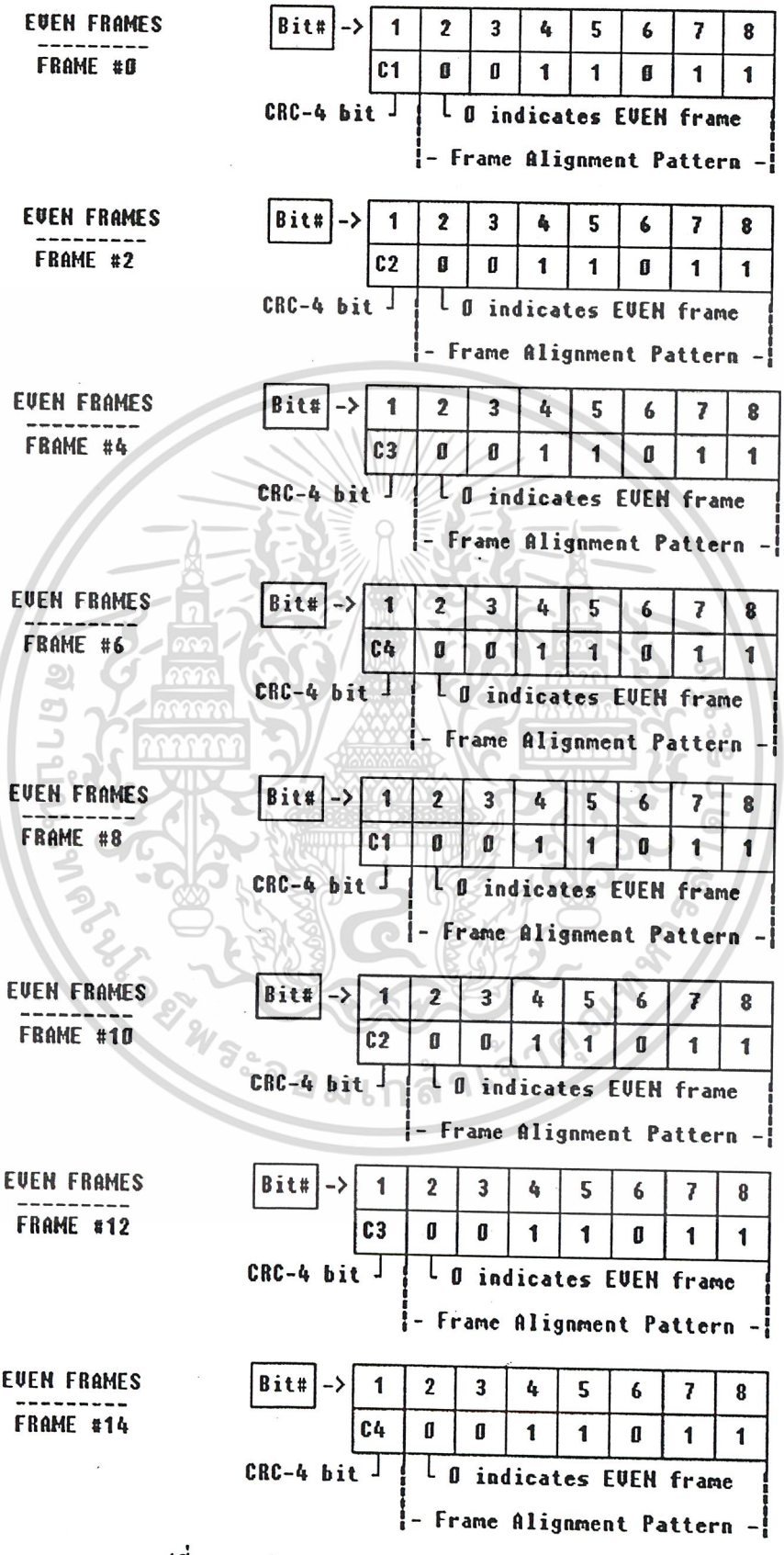
ดังที่กล่าวไว้ข้างต้นว่า มีการใช้ตัว 4-bit Cyclic Redundancy Check (CRC) CRC bits ใน frame 0, 2, 4 และ 6 จะสร้าง error detection สำหรับการส่ง/รับของ SUB-MULTIFRAME (frame 0 ถึง 7) ก่อนหน้า ส่วน CRC bits ใน frame 8, 10, 12 และ 14 จะสร้าง error detection สำหรับการส่ง/รับของ SUB-MULTIFRAME (frame 8 ถึง 15) ก่อนหน้า :

<u>Timeslot 0 ใน frame เลขคี่</u> :



รูปที่ 2.21 รูปแบบของ Timeslot 0 ใน frame เลขคี่

Timeslot 0 ใน frame เลขคู่ :

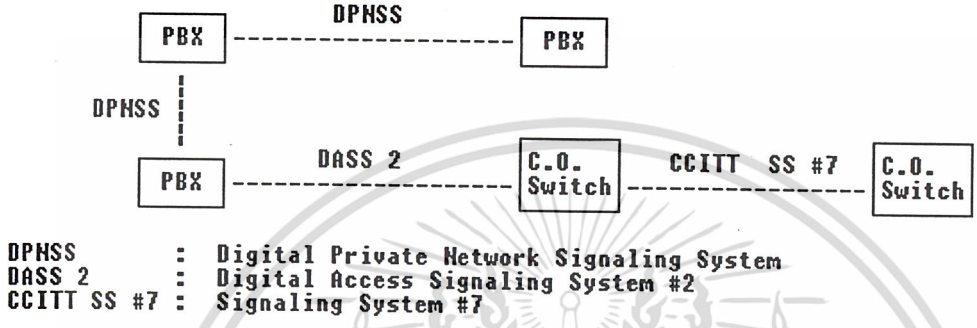


รูปที่ 2.22 รูปแบบของ Timeslot 0 ใน frame เลขคู่

2.16.5 การทำงานของ Timeslot 16

ดังที่กล่าวไว้ใน CCITT G.704 ตัว Timeslot 16 อาจจะส่ง CAS หรือไม่ก็ได้ ดังที่ได้มีการกล่าวถึงในส่วนที่ 2 ของหัวข้อนี้

เมื่อ CCS ถูกนำมาใช้ โปรโตคอล (protocol) ของสัญญาณดิจิตอลก็จะถูกใช้ระหว่างอุปกรณ์เชื่อมต่อระหว่างกัน ซึ่งสามารถเป็นได้ทั้ง SS7, ISDN, or PBX/Switch Proprietary protocols จะเห็นได้จาก diagram ถัดไป (เป็น diagram ของ UK hierarchical application) :



รูปที่2.23 ตัวอย่างการเชื่อมต่อระหว่างอุปกรณ์

เมื่อ CAS ถูกนำมาใช้ ITU ซึ่งเป็นตัวสนับสนุน G.732 จะถูกกำหนดให้เป็นแบบ MultiFrame สำหรับสัญญาณ E1 :

Frame 0: MultiFrame Alignment Signal

เมื่อ Timeslot 16 ของ E1 frame มีจุดประสงค์เพื่อถูกใช้สำหรับ Channel Associated Signaling ตัว Frame 0 จะเก็บข้อมูลที่ถูกใช้โดยตัวรับเพื่อพิจารณาเฟรมถัดไปที่กำลังมา ซึ่งลักษณะนี้เป็นลักษณะเฉพาะของ Timeslot 0 ตัว Frame 0 จะถูกเรียกเป็น MultiFrame Alignment Signal (MFAS)

	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Frame 0	0	0	0	0	X	Y	X	X

X = Spare Bits จะถูก set เป็น 1 ถ้าไม่มีการใช้

Y = Yellow Alarm (ค่า Loss ของ MultiFrame Alignment Signal)

(0 = ปกติ | 1 = ค่า Loss ของ MFAS)

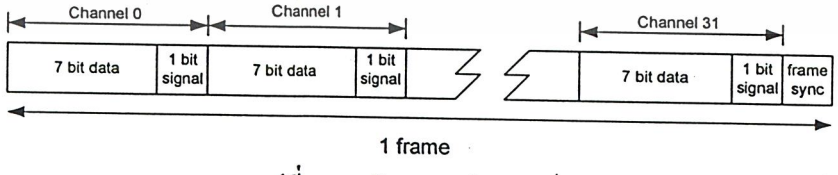
Frames 1-15: Signaling Bits

เมื่อ Channel Associated Signaling ถูกใช้ Timeslot 16 of Frames 1-15 จะถูกใช้เพื่อรับส่งสถานะของ A,B,C และ D signaling bits. มีข้อสังเกตว่า :

- สถานะ ABCD ของ 0000 ใช้ไม่ได้. ถ้าบิตทั้งหมดใน Timeslot 16 เป็น 0 ค่า Loss ของ MultiFrame Alignment Signal สามารถถูกสมมติขึ้นมาได้

- เมื่อ A-Bit ถูกใช้เพียง signaling (ไม่มีค่า BCD bits) BCD bits ควรจะถูกกำหนดค่าตายตัวเป็น B=1, C=0, D=1 (101)

```
        |Bit 1 |Bit 2 |Bit 3 |Bit 4 |Bit 5 |Bit 6 |Bit 7 |Bit 8
Frame 1 |------|------|------|------|------|------|------|------
        |Ch 1-A|Ch 1-B|Ch 1-C|Ch 1-D|Ch16-A|Ch16-B|Ch16-C|Ch16-D

        |Bit 1 |Bit 2 |Bit 3 |Bit 4 |Bit 5 |Bit 6 |Bit 7 |Bit 8
Frame 2 |------|------|------|------|------|------|------|------
        |Ch 2-A|Ch 2-B|Ch 2-C|Ch 2-D|Ch17-A|Ch17-B|Ch17-C|Ch17-D

        |Bit 1 |Bit 2 |Bit 3 |Bit 4 |Bit 5 |Bit 6 |Bit 7 |Bit 8
Frame 3 |------|------|------|------|------|------|------|-----
        |Ch 3-A|Ch 3-B|Ch 3-C|Ch 3-D|Ch18-A|Ch18-B|Ch18-C|Ch18-D

.....

        |Bit 1 |Bit 2 |Bit 3 |Bit 4 |Bit 5 |Bit 6 |Bit 7 |Bit 8
Frame 15|------|------|------|------|------|------|------|-----
        |Ch15-A|Ch15-B|Ch15-C|Ch15-D|Ch30-A|Ch30-B|Ch30-C|Ch30-D
```



รูปที่ 2.24 ลักษณะเฟรมของ E1

Europe & Australia		America & Japan	
Channels	Mbps	Channels	Mbps
30	2.048	24	1.544
120	8.448	48	3.152
480	34.368	96	6.312
1920	139.364	672	44.736
7680	565	4032	247.176

ตารางที่ 2.4 เปรียบเทียบความจำนวนช่องสัญญาณและความเร็วในการส่งของมาตรฐานตระกูล E1 และ T1

มาตรฐานอื่นๆ ในตระกูล E1

- E1 = 2.048 Mbps,
- E2 = 4 × E1 = 8.448 Mbps
- E3 = 4 × E2 = 16 × E1 = 34.368 Mbps
- E4 = 4 × E3 = 64 × E1 = 139.264 Mbps
- E5 = 4 × E4 = 256 × E1 = 565.148 Mbps.

2.17 ทฤษฎีโครงข่ายเชื่อมต่อภายใน

โครงข่ายเชื่อมโยงภายในสามารถจะนำไปประยุกต์ใช้งานได้อย่างกว้างขวางในการสื่อสารข้อมูลระบบดิจิตอล ไม่ว่าจะเป็นการประมวลผลแบบขนาน (Parallel Processing) ของการประมวลผลโปรเซสเซอร์ การนำมาสร้างเป็นโครงข่ายเชื่อมโยงภายในด้วยตัวเอง (Self-Routing Switching) การนำไปใช้งานกับโครงข่ายสื่อสารร่วมดิจิตอลแถบกว้าง (Broadband Integrated Service Digital Network) หรือ B-ISDN การนำไปใช้งานสำหรับโครงข่ายที่ต้องการถ่ายโอนข้อมูลด้วยความเร็วสูง

2.17.1 โปรโตคอลของการเชื่อมโยง

การเชื่อมต่อสื่อสารกับโครงข่ายเชื่อมต่อภายใน จะต้องมีกฎระเบียบและวิธีปฏิบัติที่เรียกว่าโปรโตคอลของโครงข่าย โดยยึดตามคุณสมบัติการเชื่อมต่อตามแบบจำลองของ OSI โปรโตคอลที่เกี่ยวข้องของโครงข่ายจะมีด้วยกัน 3 ชั้น คือ โปรโตคอลชั้นกายภาพ โปรโตคอลชั้นเชื่อมโยงข้อมูลและโปรโตคอลชั้นโครงข่าย ในที่นี้จะพิจารณาเฉพาะ 2 ชั้นแรกเท่านั้น

1. โปรโตคอลชั้นกายภาพ (Physical Layer Protocol) กล่าวถึง กฎระเบียบและวิธีการเชื่อมต่ออุปกรณ์สื่อสารเข้ากับโครงข่าย แสดงถึงคุณสมบัติที่แท้จริงของการเชื่อมต่อ เช่น ไลน์โค้ดดิ้ง สัญญาณที่ใช้ในการติดต่อใช้รหัสอะไร ระดับแรงดับแรงดันไฟฟ้าที่ใช้แทนบิต “0” และบิต “1” ลักษณะของเต้าเสียบหรือตัวเชื่อมต่อจำนวนเข็มและหน้าที่ของแต่ละเข็มของเต้าเสียบ จังหวะในการรับส่งข้อมูล เป็นต้น

2. โปรโตคอลชั้นเชื่อมโยงข้อมูล (Data Layer Protocol) การติดต่อสื่อสารระหว่างอุปกรณ์สื่อสารปลายทางกับโครงข่าย นอกจากจะใช้การเชื่อมต่อทางกายภาพแล้ว ยังต้องมีกฎระเบียบและวิธีปฏิบัติเพื่อให้อุปกรณ์สื่อสารปลายทาง และโครงข่ายสามารถตีความหมายของชุดข้อมูลได้ รวมทั้งสามารถบอกได้ว่าข้อมูลที่ได้รับนั้น มีความผิดพลาดเกิดขึ้นในการส่งหรือไม่ โปรโตคอลในชั้นเชื่อมโยงข้อมูลนี้มีอยู่ด้วยกันหลายตัวแต่ละตัวก็มีหลักการพื้นฐานที่เหมือนกันจะแตกต่างไปบ้างในรายละเอียดปลีกย่อยเท่านั้น

ในการประยุกต์ใช้งานส่วนนี้จะพิจารณาเฉพาะ 2 ชั้นเท่านั้น โดยใช้โปรโตคอลทั้งสองในการเชื่อมโยงโครงข่าย การควบคุมในการรับส่งข้อมูลในรูปแบบของชุดข้อมูล การกำหนดรูปแบบของชุดข้อมูล การตรวจสอบข้อผิดพลาดในการรับส่งข้อมูล และการควบคุมการเชื่อมต่อชุดข้อมูลผ่านโครงข่าย ให้เป็นไปตามข้อกำหนดของ CCITT

ในการออกแบบโครงข่ายเชื่อมต่อภายในนั้น จะพิจารณาจากโทโปโลยีที่ใช้งาน และมีข้อกำหนดในการพิจารณาที่เหมาะสมทางด้านสถาปัตยกรรมของโครงข่าย ดังนี้ 1) โหมดการทำงาน (Operation Mode) 2) การควบคุม (Control Strategy) 3) วิธีการสวิทชิ่ง (Switching Methodology) และ 4) โทโปโลยี (Topology)

2.17.2 โหมดการทำงาน

สามารถที่จะแบ่งโหมดการทำงานของการสื่อสารโดยทั่วไปออกเป็น 3 ประเภท คือ

2.17.1. ซิงโครนัสโหมด (Synchronous Mode) เหมาะสำหรับการสื่อสารที่ต้องถ่ายเทข้อมูลหรือการกระจายข้อมูลที่เข้าจังหวะกระทำไปพร้อมๆกันไป โดยสามารถที่จะใช้ส่งผ่านข้อมูลได้ด้วยความเร็วสูงได้

2.17.2. อะซิงโครนัสโหมด (Asynchronous Mode) เหมาะสำหรับใช้ในการสื่อสารที่มีการติดต่อที่เปลี่ยนแปลงอยู่เสมอ แต่จะมีข้อด้อยตรงที่ความสามารถส่งถ่ายข้อมูลได้ไม่ดีนัก

2.17.3. ระบบผสม (Combination Mode) เป็นระบบสื่อสารที่ใช้ทั้งระบบซิงโคนัส และอะซิงโคนัสรวมกัน

โครงข่ายเชื่อมต่อภายในประกอบด้วยวงจรสวิตซ์จำนวนมากเชื่อมต่อกัน ตามรูปแบบการเชื่อมโยงโครงข่าย “อินเตอคอนเนคชั่นลิงค์” จะเชื่อมโยงวงจรสวิตซ์ในลักษณะต่างกันตามโทโปโลยีและการควบคุม “อินเตอคอนเนคชั่นฟังก์ชัน” ควบคุมวงจรสวิตซ์โดยการควบคุมจากศูนย์กลาง (Centralized Control) การควบคุมวงจรสวิตซ์แต่ละตัวจะกระทำโดยหน่วยควบคุมกลาง

2.18 พื้นฐานการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก (D/A)

หน้าที่ของวงจรแปลง D/A คือ รับกลุ่มของบิทจากคอมพิวเตอร์หรืออุปกรณ์ดิจิตอลอื่นๆและแปลงรูปรูปแบบบิทนั้นไปเป็นระดับแรงดันอนาลอก โดยทั่วไปรูปแบบบิทมักจะเป็นไบนารี่ (binary) เอาต์พุตของ D/A จะมีระดับที่แตกต่างกันสำหรับแต่ละดิจิตอลอินพุตที่จ่ายให้ นอกจากนี้เอาพุต์พุตของ D/A อาจจะเป็นแรงดันหรือกระแสก็ได้ขึ้นอยู่กับโครงสร้างภายในจของอุปกรณ์

<u>ความละเอียดและเอาต์พุตเต็มสเกลของ D/A</u>

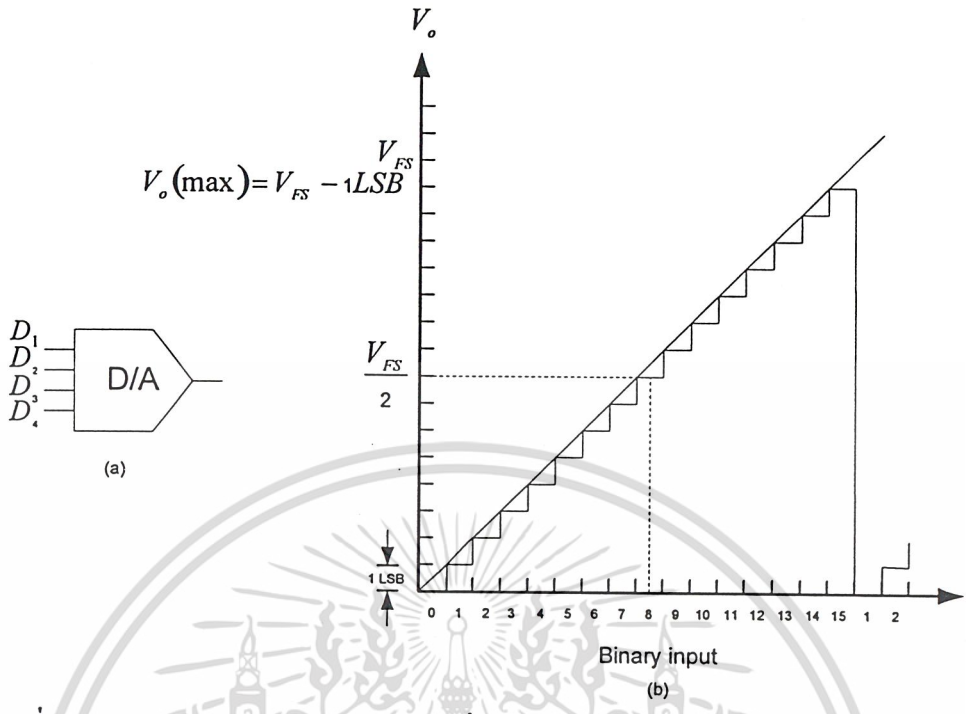
จำนวนระดับเอาต์พุตที่แตกต่างกันที่สามารถผลิตได้โดย D/A นั้นจะสัมพันธ์กับจำนวนของอินพุตบิทที่วงจรมี โดยจะมีความสัมพันธ์ดังต่อไปนี้

$$N = 2^n \tag{2.22}$$

เมื่อ N เป็นจำนวนของระดับเอาต์พุตที่แตกต่างกันที่ D/A สามารถผลิตได้ และ n คือจำนวนอินพุตบิทที่วงจรมี ซึ่งค่า N ของ D/A ถูกใช้เพื่อกำหนดความละเอียด (resolution) ของ D/A ตัวนั้นๆ อินพุตของ D/A ที่มาก (n มาก) จะให้ความละเอียดที่สูง ความละเอียดอาจถูกแสดงได้เป็น 1 ใน N ส่วน (ค่า N ถูกนิยามไว้ดังสมการที่ 2.22) ยกตัวอย่างเช่น D/A 10 บิทก้จะมีความละเอียด 1 ใน 1024 ส่วน หรือความละเอียดยังอาจถูกแสดงได้เป็นค่าเปอร์เซ็นต์อีกด้วยดังสมาการต่อไปนี้ ซึ่งเป็นความสัมพันธ์ระหว่างความละเอียดและจำนวนอินพุตของ D/A ที่มี

$$\text{Percent resolution} = \frac{1}{2^n} \times 100\% \tag{2.23}$$

ใช้สมการที่ (2.23) กับ D/A 10 บิทที่ได้ยกตัวอย่างไปแล้วข้างต้นจะพบว่า D/A นี้มีความละเอียด 0.098% ซึ่งหมายความถึงการเปลี่ยนแปลงน้อยที่สุดที่เป็นไปได้ของเอาต์พุต D/A ที่สามารถเกิดขึ้นได้ โดยการเปลี่ยนแปลงของอินพุตไบนารี่ จะเป็น 0.098% ของค่าเอาต์พุตเต็มสเกล (scale)



รูปที่ 2.25 บิท D/A converter (a) สัญลักษณ์ทั่วไปของ D/A และ (b) transfer characteristic

เอาต์พุตเต็มเสกล คือแรงดัน (V_{FS}) หรือกระแส (I_{FS}) ซึ่งถูกผลิตขึ้นที่เอาต์พุตของ D/A ที่ถูกสมมติว่ามีความละเอียดเป็นอนันต์ (จำนวนอินพุตเป็นอนันต์) โดยทุกๆ อินพุตของ D/A ถูกจ่ายด้วยไบนารี่ "1" เนื่องจาก D/A ตามความเป็นจริงแล้วไม่สามารถมีจำนวนอินพุตเป็นอนันต์ได้ เอาต์พุตของมันจึงมีค่าไม่ถึงระดับเต็มสเกลของเอาต์พุตทางอุดมคติแน่นอน ตัวอย่างเช่น พิจารณา 4 บิท D/A ในรูปที่ 2.25(a) ถ้าเราจ่ายไบนารี่ 4 บิทอย่างเป็นลำดับ (เริ่มต้นที่ทุกบิทเป็นศูนย์) จากน้อยไปหามากให้กับอินพุทจะได้กราฟถ่ายโอน (transfer curve) ที่เป็นขั้นบันไดดังรูปที่ 2.25(b) สังเกตว่ามีแรงดันเอาต์พุต 16 ระดับหรือ 16 ขั้นที่แตกต่าง เพื่อให้มีเอาต์พุตที่ต็มสเกลเราต้องการลำดับขั้นถึง 17 ขั้น (ซึ่งไม่ใช่ค่าที่เป็นกำลังของ 2)

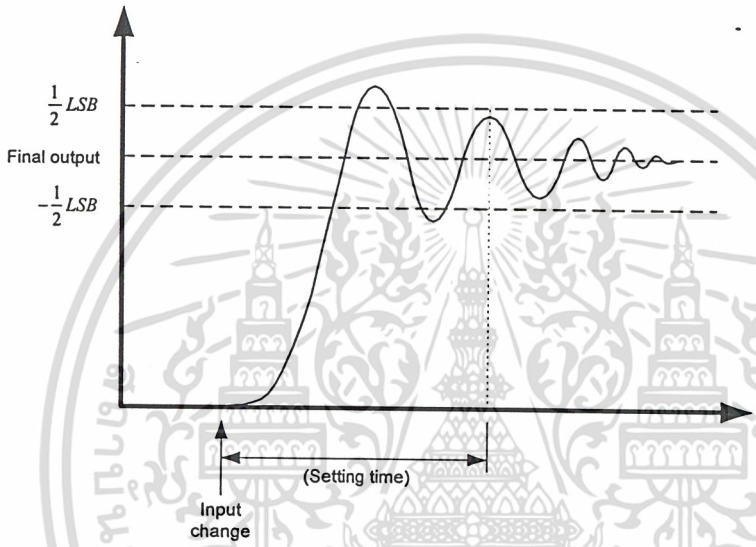
ดังนั้นเราจึงสามารถเห็นได้จากรูปว่าค่า V_o มากที่สุดจะน้อยกว่า V_{FS} อยู่หนึ่งขั้นการเปลี่ยนแปลงน้อยที่สุดที่เอาต์พุตซึ่งสามารถเกิดขึ้นได้จากการเปลี่ยนแปลงอินพุตจะมีค่าเป็น 1 LAB หรือ 1 ขั้น การเปลี่ยนแปลงนี้ใช้คำว่า 1 LSB เพราะว่ามันเกิดขึ้นเมื่อบิทที่มีความสำคัญต่ำสุด (LSB) ของอินพุตไบนารี่เปลี่ยนสถานะ

ในอุดมคติ ขนาดเอาต์พุตสำหรับแต่ละขั้นจะเท่ากัน และถูกกำหนดโดยจำนวนขั้น(ความละเอียด) และแรงดันต็มสเกล ผ่านความสัมพันธ์ต่อไปนี้

$$\text{step size} = 1\ \text{LSB} = \frac{V_{FS}}{2^n} \tag{2.24}$$

ที่ n เป็นจำนวนของอินพุตไบนารี่ของ D/A และ V_{FS} เป็นแรงดันเต็มสเกลของ D/A เทียบเคียงในทางอุดมคติ

โดยการใช้สมการที่ (2.24) จึงเป็นไปได้ที่จะกำหนดแรงดันเอาต์พุตของ D/A (หรือกระแส) สำหรับอินพุตไบนารี่ใดๆ ที่กำหนดให้ ยกตัวอย่างเช่น 4 บิท D/A มี V_{FS} =10 V ดังนั้น D/A จะมี step size เท่ากับ 0.625 V ถ้าข้อมูลไบนารี่เป็น 1000 แล้วเราจะได้แรงดันเอาต์พุตของ D/A คือ (8)(0.6245 V) = 5V จากตัวอย่างข้างต้น D/A จะมีแรงดันเอาต์พุตเป็น $V_{FS}/2$ เมื่อ MSB บิทของอินพุตไบนารี่เป็น High และอินพุตบิทที่เหลือเป็น Low นอกจากนี้ยังสังเกตได้อีกว่าเมื่อค่าเชิงตัวเลขของอินพุตไบนารี่ถูกเพิ่มหรือลด แรงดันเอาต์พุตก็จะเพิ่มหรือลดตามสัดส่วนโดยตรง ความละเอียดของ D/A อาจถูกใช้เป็นเครื่องแสดงโดยทั่วไปถึงความถูกต้องแม่นยำของ D/Aได้



รูปที่ 2.26 การกำหนดความแน่นอนของ D/A settling time

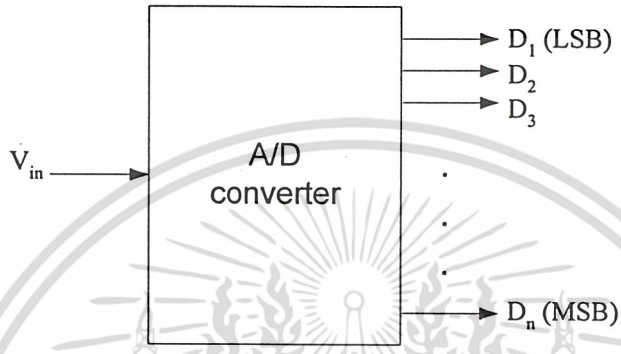
Settling Time

D/A ในทางอุดมคติจะมีการตอบสนองทีทันทีต่ออินพุต แต่ D/A ในทางปฏิบัติแล้วต้องการเวลาระยะหนึ่งเพื่อนเปลี่ยนสถานะเอาต์พุต และทำให้ค่าสุดท้ายของเอาต์พุตสงบนิ่งในระดับหนึ่ง ช่วงเวลานี้ถูกระบุว่าเป็น Settling Time ของ converter ปกติโดยทั่วไป Settling Time ถูกนิยามว่าเป็นเวลาที่ต้องการสำหรับ converter ภายใต้เงื่อนไขที่เลวร้าย Settling Time สูงสุดจะเกิดขึ้นเมื่อเอาต์พุตของ converter ทำการเปลี่ยนแปลงจากค่าต่ำ

เพื่อให้เอาต์พุตมาถึงในช่วง $\pm\frac{1}{2}LSB$ (หรือ 99.5% ของระดับเอาต์พุตสุดท้าย) ไปยังค่าสูงสุกหรือในทางตรงกันข้ามจากค่าสูงสุดไปยังค่าต่ำสุด รูปที่ 2.26 แสดงภาพรวมของ Settling Time ซึ่งค่าสุดท้ายถูกสมมติว่าเป็น $\pm\frac{1}{2}LSB$ Settling Time นั้นจะเป็นตัวจำกัดอัตราที่ D/A สามารถทำการแปลงอย่างต่อเนื่องได้ ยกตัวอย่างเช่น พิจารณา D/A converter ที่มี Settling Time ไม่เกิน 1 ms ค่าของไบนารี่อินพุตจะต้องไม่เปลี่ยนสถานะที่อัตรามากกว่าทุกๆ 1 ms เพื่อให้ได้รับเอาต์พุตที่ถูกต้องภายใต้การเปลี่ยนแปลงที่เป็นไปได้ทั้งหมดของอินพุท Word

2.19 พื้นฐานการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล (A/D)

สัญลักษณ์บล็อกไดอะแกรมของ A/D แสดงได้ดังรูป 2.27 หน้าที่ของ A/D คือสุ่มค่าระดับสัญญาณอนาล็อก (ปกติเป็นแรงดัน) และผลิตข้อมูลไบนารี่เป็นตัวแทนของระดับที่เอาต์พุตของมันและ เช่นเดียวกันกับ D/A จำนวนของบิทที่เอาต์พุตของ A/D เป็นตัวกำหนดความละเอียด (resolution) และความเที่ยง (accuracy) ของ A/D

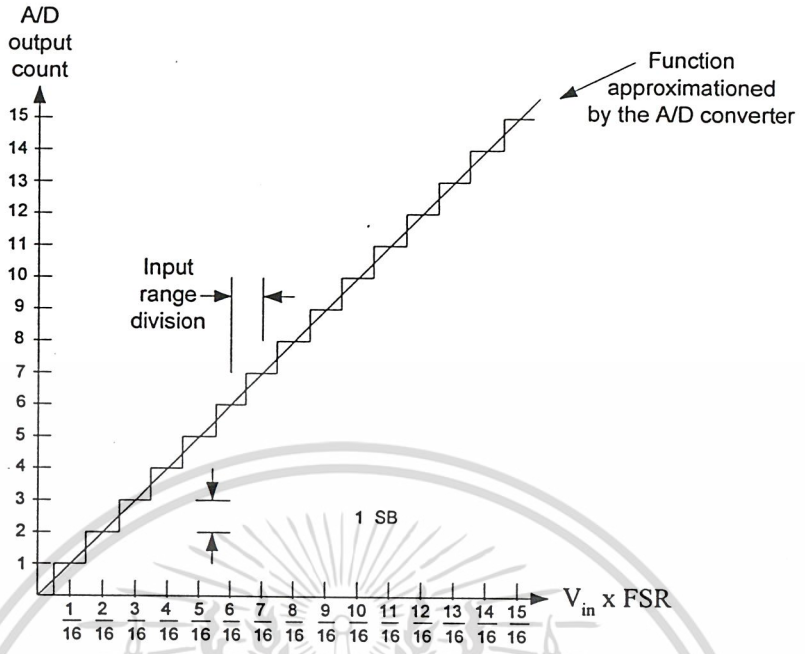


รูปที่ 2.27 สัญลักษณ์สำหรับ A/D converter n-bit

Full – Scale Range

ในรูป 2.280 เป็นกราฟถ่ายโอนคุณลักษณะ (Transfer Characteristics) ของ 4 บิท A/D ซึ่ง 4 บิท A/D สามารถจำแนก (resolved) สัญญาณอินพุทอนาล็อกออกเป็น 1 ใน 16 เอาต์พุต code word อินพุทของ A/D นั้นเหมาะสมเฉพาะเพียงที่ย่านจำกัดเท่านั้น ยกตัวอย่างเช่น จาก 0 ถึง 10V ช่วงของอินพุตที่สามารถถูกจำแนกได้จะถูกเรียกว่า Full Scale Range (FSR) ย่านอินพุตของ A/D ถูกแบ่งออกได้เป็น 2^n ส่วน เมื่อ n คือจำนวนบิทของเอาต์พุต code word การแบ่งย่านอินพุตของ A/D คล้ายคลึงกับความละเอียดของ D/A ดังแสดงไว้ในสมการที่ (2.24) ดังนั้นเราจะได้

$$\text{Input range division} = 1\ \text{LSB} = \frac{FSR}{2^n} \qquad (2.25)$$



รูปที่ 2.28 กราฟถ่ายโอนคุณลักษณะสำหรับ A/D converter 4 bit ทางอุดมคติ

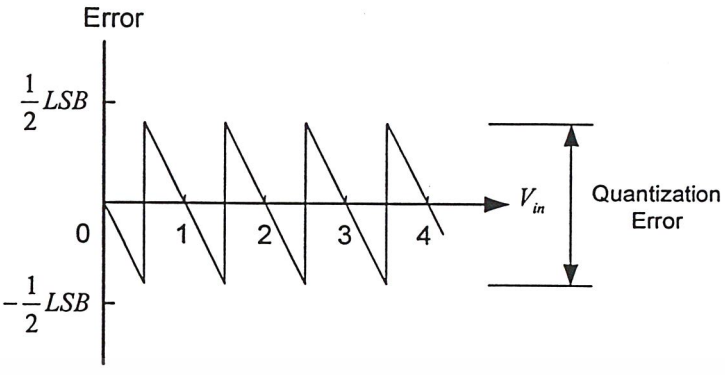
เอาต์พุตของ A/D ปกติแล้วจะเป็นรูปแบบไบนารี่ มาตรฐาน 8-4-2-1 มีอยู่ข้อหนึ่งที่ A/D คล้ายกับ D/A คือแรงดันเอาต์พุตเต็มสเกลไม่สามารถทำให้เกิดขึ้นจริงได้ แรงดันอินพุตที่เกิน FSR ของ A/D จะทำให้ A/D ไม่สามารถตัดสินได้ว่าแรงดันอยู่ที่ระดับใด นั่นเป็นเพราะว่าอินพุทเต็มสเกลที่จ่ายไปยัง A/D เกินจากเงื่อนไขที่ย่านจะรับได้

A/d Error Specification

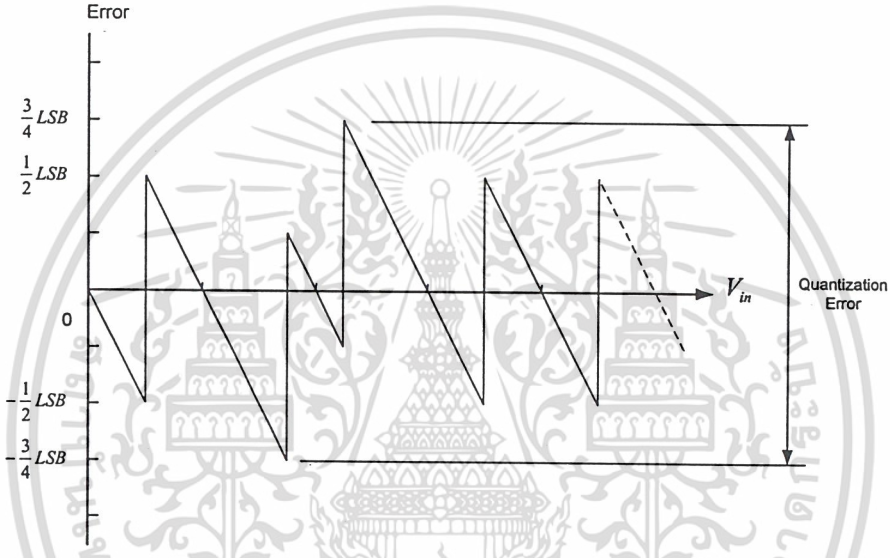
ถึงแม่ว่าสัญญาณอนาล็อกจะอยู่ภายในย่านที่จำกัด แต่ว่าสัญญาณอาจมีค่าใดๆ ก็ได้ที่แตกต่างกันอย่างไม่จำกัด A/D จะสุ่มสัญญาณอนาล็อกอินพุทมาจัดระบบให้อยู่ในช่วงพิสัยที่จำกัดและผลิตเอาต์พุตเป็น code word ซึ่งเป็นตัวแทนของอินพุทนั้น การทำแบบนี้เป็นไปได้ที่ตัวมันเองจะทำให้เกิดเอาต์พุตผิดพลาด (error) ได้

Quantization error

มีความไม่แน่นอนเกิดขึ้นเสมอเกี่ยวกับค่าจริงของอินพุท เพราะว่าจำนวนบิทที่เอาต์พุตของ A/D นั้นมีจำกัด ซึ่งความไม่แน่นอนที่เกี่ยวข้องกับการแปลงนี้ถูกเรียกว่า Quantization error หรือ Quantization noise A/D ใดๆ จะแสดง Quantization error ต่ำสุดออกมาอย่างน้อย $\pm\frac{1}{2}LSB$ รูป 2.29(a) แสดงการ plot ของ Quantization error เทียบกับแรงดันอินพุทสำหรับ A/D ด้วยความเที่ยงตรง $\pm\frac{1}{2}LSB$



(a)



(b)

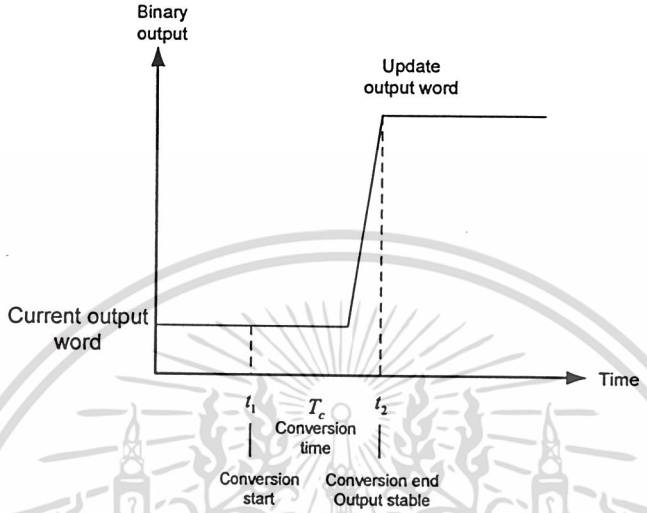
รูปที่2.29 การพล็อตค่าความผิดพลาดของ A/D converter โดย (a) converter อุดมคติ (b) converter ที่มีความแม่นยำ $\pm\frac{3}{4}LSB$

มันค่อนข้างยากที่จะผลิต A/D ความละเอียดสูงและมี Quantization error เพียง $\pm\frac{1}{2}LSB$ แสดง error plot ของ A/D ที่มี Quantization error เพียง $\pm\frac{3}{4}LSB$ โดยทั่วไปตำแหน่งของ error peak มักมีค่าสุ่ม ดังนั้นเราจึงต้องสมมติว่าสำหรับอินพุทใดๆที่กำหนดให้เอาต์พุต code word ของมันจะเกิดผิดพลาดขึ้นอย่างมาก $\pm\frac{3}{4}LSB$ A/D บางตัวอาจแสดง Quantization error มากถึง $\pm LSB$ ในการใช้งานด้านการประมวลผลสัญญาณ บางครั้งต้องการแสดง Quantization noise ในรูปแบบของเดซิเบล (dB) มากกว่า ซึ่งเรียกว่าอัตราส่วน Signal to Quantization Noise Ratio (SQNR) นิยามได้ดังนี้

$$SQNR = 20\log\frac{FSR(V)}{1/LSB(V)} \tag{2.26}$$

Conversion Time

A/D ในทางปฏิบัติแล้วต้องการเวลาระยะหนึ่งในการแปลงข้อมูล ช่วงเวลานี้ถูกเรียกว่า Conversion Time (T_c) ซึ่งแสดงไว้ในรูปที่ 2.30 ไม่เสมอไปว่า A/D จะต้องมี Conversion Time ที่คงที่ แต่ส่วนใหญ่จะระบุมาเป็นเงื่อนไขการทำงานที่ต้องการเวลามากที่สุดแทน



รูปที่ 2.30 เวลาการแปลงที่ต้องการของ A/D

2.20 ภาษาวีเอชดีแอล

ความซับซ้อนและขนาดของระบบดิจิตอลในปัจจุบันได้เพิ่มมากขึ้นทุกขณะ ส่งผลให้มีการนำคอมพิวเตอร์เพื่อช่วยในการออกแบบ มาใช้ในกระบวนการออกแบบฮาร์ดแวร์เพิ่มขึ้น อีกทั้งอุปกรณ์และวิธีการออกแบบใหม่ๆ ก็ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบมากขึ้นด้วย สำหรับภาษาบรรยายอุปกรณ์ฮาร์ดแวร์ เอชดีแอล (HDL: Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่องเพื่อช่วยในการปรับปรุงขบวนการออกแบบระบบดิจิตอลให้เป็นไปอย่างมีประสิทธิภาพ

2.20.1 ประวัติความเป็นมาของภาษาวีเอชดีแอล

วีเอชดีแอล ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และสามารถเขียนได้หลายรูปแบบด้วยเหตุผลนี้จึงทำให้ภาษาวีเอชดีแอล เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกรได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชั่นการทำงานของวงจรอย่างสังเขป โดยไม่คำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนี้ วีเอชดีแอล ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มีความซับซ้อนได้ทั้งหมด ดังนั้น วีเอชดีแอล จึงเป็นภาษาที่น่าสนใจใน

การศึกษาและนำไปใช้งานเป็นอย่างยิ่ง สำหรับมาตรฐานของภาษาที่ใช้บรรยายพฤติกรรมวงจร หรือฮาร์ดแวร์ สามารถสรุปได้ดังนี้

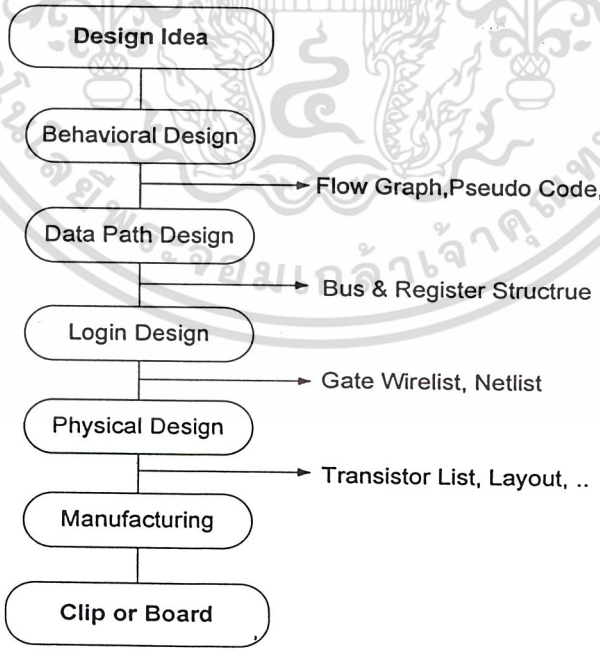
- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่น เดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรมภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า ภาษาโปรแกรมระดับสูง

2.20.2 การออกแบบระบบดิจิตอล

ในการออกแบบระบบดิจิตอล เริ่มตั้งแต่การกำเนิดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆ มากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์ในแต่ละขั้นก่อนเข้าสู่กระบวนการออกแบบในขั้นต่อไป

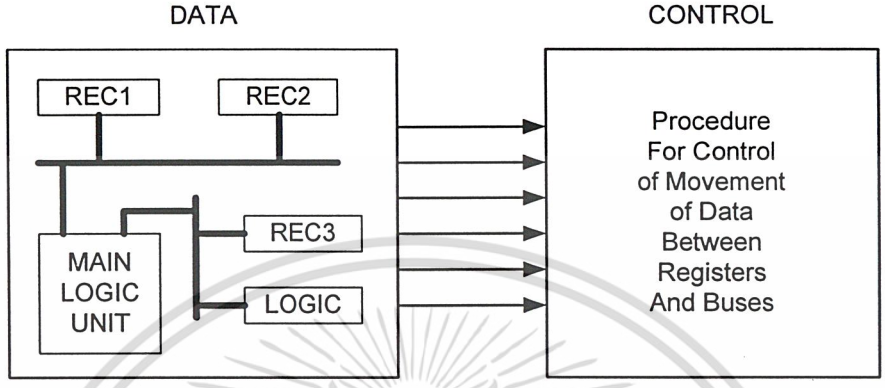
รูปที่ 2.31 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบดิจิตอลทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบ แล้วทำการพัฒนาให้สามารถนำไปใช้ได้อย่างสมบูรณ์ ซึ่งภายในขั้นตอนนี้ผู้ออกแบบจำเป็นต้องสร้างรูปแบบระบบเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงานแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo code) ก็ได้



รูปที่ 2.31 แสดงขั้นตอนการออกแบบระบบดิจิตอล

ขั้นตอนต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์และวงจรลอจิกที่จำเป็นทั้งหมด เพื่อนำมาประกอบเป็นระบบที่สมบูรณ์ โดยแต่ละองค์ประกอบ

สามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรลอจิกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.32



รูปที่ 2.32 การออกแบบระบบเส้นทางข้อมูล

ขั้นตอนถัดมาเป็นการออกแบบวงจรลอจิกซึ่งจะเกี่ยวข้องกับการนำเกทดิจิตอลพื้นฐาน และ ฟลิปฟลอป (Flip – Flop) มาประกอบเป็นอุปกรณ์ย่อยต่าง ๆ เช่นรีจิสเตอร์เก็บข้อมูล บัสวงจรลอจิก และส่วนควบคุมฮาร์ดแวร์ ซึ่งผลลัพธ์ที่ได้ในขั้นตอนนี้จะเป็นเครือข่ายของการโยงใยระหว่างเกทและฟลิปฟลอปนั่นเอง

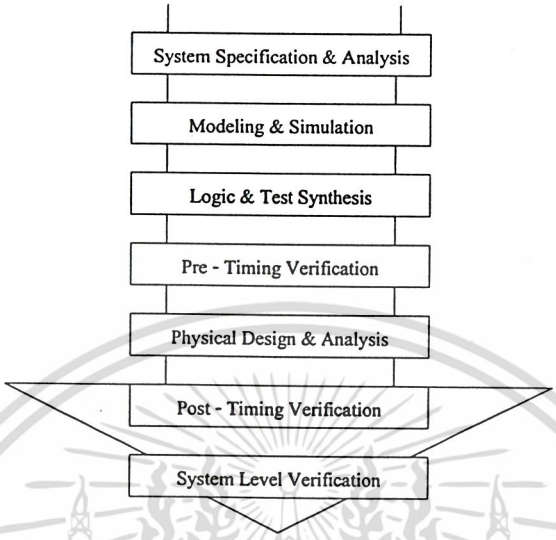
การออกแบบในขั้นตอนถัดไปเป็นการเปลี่ยนเครือข่ายการโยงใยที่ได้จากขั้นตอนที่แล้ว ให้เป็นลำดับของทรานซิสเตอร์ (Transistor List) และ โครงงาน (Layout) ซึ่งขั้นตอนนี้จะเกี่ยวข้องกันโดยตรงกับการจัดวางทรานซิสเตอร์หรือไลบรารีเซลล์เพื่อแทนเกทและฟลิปฟลอปต่าง ๆ

และในขั้นตอนสุดท้ายจะเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด

2.20.3 การออกแบบจากบนลงล่าง (Top-Down Design)

ในการพัฒนาวงจรรวมดิจิตอลขนาดใหญ่ที่มีความซับซ้อน วิศวกรหรือผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของ บล็อกไดอะแกรมก่อนที่ทำวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา วีเอชดีแอลนั้นอนุญาตให้อธิบายและวิเคราะห์การทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการนอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่างนั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวางวงจรด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบก่อน แล้วจึงทำการจำลองการทำงาน และตรวจสอบความถูกต้อง วีเอชดีแอล กับหลักการออกแบบ

จากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.33 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.33 แสดงถึงขั้นตอนการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย เนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1) ความต้องการของระบบและการวิเคราะห์ คือ การสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2) รูปแบบและการจำลองการทำงาน คือ การเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษาวีเอชดีแอล หรือ ภาษา เอชดีแอล อื่นๆ สำหรับใช้ในการบรรยายพฤติกรรมการทำงาน พร้อมทั้งทำการจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3) ลอจิกและการทดสอบการสังเคราะห์ คือ หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมรายละเอียดลงมาเป็นลำดับขั้นที่สอง จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจรจริง หรือทำการสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่จะเขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของโครงข่ายการเชื่อมต่อ ที่สามารถนำไปผลิตอุปกรณ์อื่นได้

4) การตรวจสอบเวลาก่อนการออกแบบ คือ หลังจากได้ทำการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือโครงข่ายการเชื่อมต่อแล้ว ข้อมูลนี้จะถูกนำไปใช้สำหรับการจำลองการทำงานในเรื่องความถูกต้องของฟังก์ชัน พร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาใช้ในการประกอบในการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ทางอิเล็กทรอนิกส์ทุกชิ้นจะต้องมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะ

เป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถที่จะทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

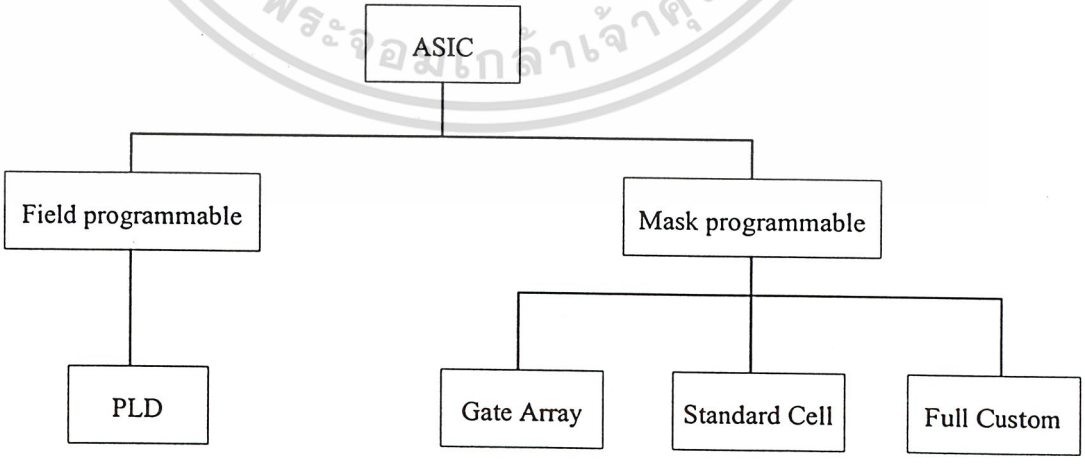
5) การออกแบบทางกายภาพและการวิเคราะห์ คือ ขั้นตอนในการผลิตเป็นวงจรจริง (Technology and device mapping) โดยจะนำข้อมูลที่ได้จากการสังเคราะห์ มาใช้ในการผลิตเป็นวงจรรวม ซึ่งอาจ จะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวม เอซิก (ASIC)

6) การตรวจสอบเวลาหลังการออกแบบ คือ การทำการตรวจสอบการทำงานด้วยตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

7) การตรวจสอบระบบ คือ การนำวงจรที่ออกแบบไว้ประกอบเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบที่สมบูรณ์ แล้วทำการทดสอบการทำงานทั้งระบบรวมกับอุปกรณ์อื่นๆ อีกครั้งเพื่อควบคุมคุณภาพของผลิตภัณฑ์

2.21 เอฟพีจีเอ

เทคโนโลยีความก้าวหน้าของอุตสาหกรรมอิเล็กทรอนิกส์ในปัจจุบัน ทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆ ซึ่งทำให้ลดค่าใช้จ่ายต่างๆได้มาก ในขณะเดียวกันก็มีการเพิ่มประสิทธิภาพและระดับความเชื่อถือได้ของวงจรรวมที่สูงขึ้นเห็นได้ชัดจากเทคโนโลยีไมโครโปรเซสเซอร์ และหน่วยความจำปัจจุบัน ทุกๆครั้งที่มีการพัฒนาขึ้นทำให้เกิดช่องว่างระหว่างวงจรรวมและไอซีมาตรฐานมากขึ้น นักออกแบบอุปกรณ์ทางด้านดิจิตอล ได้พิจารณาถึงการผลิตให้มีขนาดมากขึ้นและการผลิตวงจรรวมเอซิก (ASIC: Application Specific Integrated Circuit) ซึ่งวงจรรวมจะแบ่งตามสร้างออกเป็น 2 กลุ่ม คือ ฟิลด์โปรแกรมเมเบิล (Field programmable) และ แมสโปรแกรมเมเบิล (Mask programmable) ดังแสดงในรูปที่ 2.34



รูปที่ 2.34 แสดงผังการแบ่งกลุ่มของวงจรรวมเอซิก

2.21.1 การออกแบบวงจรเชิงเลขด้วยชิพอุปกรณ์เอฟพีจีเอ

ชิพอุปกรณ์เอฟพีจีเอ เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ เอฟพีจีเอ มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ ในการทำชิพอุปกรณ์ ซึ่งเป็นวิธีการออกแบบ ไอซี (IC : Integrated Circuit) แบบ เซมิคัสตัม (Semi custom) อีกวิธีหนึ่ง เมื่อเทียบกับการทำ เอซิก แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำชิพอุปกรณ์เอฟพีจีเอ จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในชิพอุปกรณ์เอฟพีจีเอ จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำชิพอุปกรณ์เอฟพีจีเอ ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำชิพอุปกรณ์ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำเอซิก มาก และการตรวจสอบหรือแก้ไขการออกแบบที่ทำได้สะดวก

การทำชิพอุปกรณ์เอฟพีจีเอ ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตชิพอุปกรณ์เอฟพีจีเอ ได้เพิ่มความสามารถของชิพอุปกรณ์เอฟพีจีเอ โดยเพิ่มจำนวนองค์ประกอบภายใน หรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ พีพีอาร์ (PPR: Partitioning Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย

สำหรับตัวชิพอุปกรณ์เอฟพีจีเอนั้นมีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นชิพอุปกรณ์เอฟพีจีเอ ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นชิพอุปกรณ์เอฟพีจีเอ สามารถนำไปประยุกต์ใช้งานได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP: Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น

2.21.2 ปัจจัยที่ทำให้การออกแบบชิพอุปกรณ์เอฟพีจีเอ ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพเพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึงภาษาแอสแซมบลี้ (Assembly) ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจรหรือเอชดีแอล เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มันจากนั้นตัวซอฟต์แวร์จะทำการสังเคราะห์ ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของคอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้โดยไม่จำเป็นต้องถอดมโปรแกรมข้างนอก และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสียค่าใช้จ่ายเพิ่มแต่อย่างใด

บทที่ 3
การคำนวณและการสร้าง

3.1 การการออกแบบคำนวณค่าปัจจัยต่างๆที่มีต่อระบบ จากโปรแกรม MATLab

3.1.1 การคำนวนค่าของ PNcode

3.1.1.1 การออกแบบ PN code แบบ m-sequence โดยนำโค้ดจาก [10] โดยเรียกค่าจากฟังก์ชัน mseq(stg, taps, inidata, n) เป็นฟังก์ชันที่ใช้สร้าง M-sequence ซึ่งมีรายละเอียดดังต่อไปนี้

stg คือ จำนวนสเตจทั้งหมดของชิฟรีจิสเตอร์

taps คือ เมตริกซ์ที่เก็บค่าที่เป็นตำแหน่งสเตจของชิฟรีจิสเตอร์ที่มีการย้อนกลับ

inidata คือ เมตริกซ์ที่เก็บค่าสถานะเริ่มต้นของชิฟรีจิสเตอร์ที่มีการย้อนกลับ

n คือ จำนวนเอาท์พุท M-sequence n ชุด

mout คือ เมตริกซ์เก็บค่าเอาท์พุท M-sequence

เริ่มต้นด้วยการพิจารณาว่ามีจำนวนอินพุทของฟังก์ชันน้อยกว่า 4 ตัวหรือไม่ ถ้าเป็นจริงจะกำหนดให้ n = 1 หรือก็คือ ให้มีจำนวนเอาท์พุท M-sequence ที่ออกมา 1 ชุด แต่ถ้าเป็นเท็จก็จะข้ามไปทำขั้นตอนถัดไป ขั้นตอนต่อมาก็คือ กำหนดให้เมตริกซ์ mout เป็นเมตริกซ์ศูนย์ขนาด $n\times (2^{stg}-1)$ และกำหนดเมตริกซ์ fops ในตำแหน่งที่มีค่าเทากับ taps โดยให้ตำแหน่งนั้นมีค่าเท่ากับ 1 กระบวนการต่อไป คือ ใช้การวนลูปด้วยคำสั่ง for เพื่อเก็บค่าเอาท์พุท M-sequence ที่ได้ลงในเมตริกซ์ mout โดยวนลูปตั้งแต่ ii = 1 หรือ $2^{stg}-1$ ซึ่งคิดเป็นจำนวนรอบเท่ากับ $2^{stg}-1$ รอบ ซึ่งมีขั้นตอนดังต่อไปนี้

- เก็บค่าเอาท์พุท M-sequence ทีได้ลงในแถวที่ 1 ของเมตริกซ์ mout โดยเก็บรอบละ 1 ตำแหน่ง ตั้งแต่หลักที่ 1 ถึงหลักที่ $2^{stg}-1$
- คำนวณค่าที่ป้อนกลับ ซึ่งรวมทุกสเตจที่มีการป้อนกลับแบบ 2 – modulo โดยเก็บค่าในตัวแปร num
- ทำการเลื่อนค่าในชิฟรีจิสเตอร์ไป 1 สเตจ
- นำผลลัพธ์ที่คำนวณจากสเตจการป้อนกลับมาป้อนเป็นอินพุทให้กับชิฟรีจิสเตอร์สเตจแรก

กระบวนการสุดท้าย คือ ทำการตรวจสอบค่า n โดยถ้าค่า n > 1 เป็นจริงจะทำการวนลูปโดยใช้คำสั่ง for ตั้งแต่ ii = 2 ถึง n คิดเป็นจำนวนรอบได้ n -1 รอบ เพื่อเก็บค่าเอาท์พุท M-sequence ลงในเมตริกซ์ mout ให้ครบ n ชุด ซึ่งเอาท์พุทแต่ละชุดตั้งแต่ชุดที่ 2 จะได้มาจากการเลื่อนค่า M-sequence ชุดก่อนไปทางขวา 1 ครั้งและเมื่อทำกระบวนการวนลูปนี้เสร็จสิ้นก็จะเป็นการสิ้นสุดกระบวนการทั้งหมดในฟังก์ชันนี้ซึ่งจะได้ M-sequence จำนวน n ชุดเก็บในเมตริกซ์ mout ขนาด $n\times 2^{stg}-1$ โดยแต่ละค่าในแถวของเมตริกซ์ mout ก็คือ M-sequence 1 ชุด

ส่วนในกรณีที่ n > 1 เป็นเท็จ กล่าวคือ n = 1 กระบวนการทั้งหมดก็จะเสร็จสิ้นลงโดยได้เมตริกซ์ mout ขนาด $n \times 2^{m}$-1 ซึ่งก็คือ M-sequence 1 ชุด นั่นเอง

โดยที่ค่า PNcode แบบ m-sequence ที่ใช้นั้นเป็นแบบ maximal length แบบ 31-modulo มีทั้งหมด 5 สเต็จของชิฟรีจิสเตอร์โดยค่าเริ่มต้นมีค่าเป็น 1 ทั้งหมดต้องการค่าทั้งหมด 31 ชุดทำการป้อนกลับที่จุด 2, 5 จะต้องทำการเรียกคำสั่ง mseq โดยป้อนค่าตัวแปรคือ

```
mPNcode = mseq(5, [2 5], [1 1 1 1 1], 31);
```

3.1.1.2 การออกแบบ PN code แบบ goldsequence โดยนำโค้ดจาก [10] โดยเรียกค่าจากฟังก์ชันฟังก์ชัน goldseq (m1, m2, n) เป็นฟังก์ชันในการสร้าง Gold Sequence โดย

m1	คือ M-sequence ชุดที่ 1
m2	คือ M-sequence ชุดที่ 2
n	คือ จำนวนเอาท์พุท Gold Sequence n ชุด
gout	คือ Gold Sequence

เริ่มต้นด้วยการพิจารณาจำนวนอินพุทของฟังก์ชันกล่าว คือ ถ้าจำนวนอินพุทของฟังก์ชันน้อยกว่า 3 ตัวเป็นจริง ก็จะกำหนดให้ n = 1 แล้วจึงทำขั้นตอนถัดไป แต่ถ้าเป็นเท็จ ก็จะข้ามไปทำขั้นตอนถัดไปทันที ซึ่งขั้นตอนถัดมาคือกำหนดให้เมตริกซ์ gout เป็นเมตริกซ์ศูนย์ที่มีขนาด n×(ความยาวของชุดรหัส M-sequence ชุดที่ 1) ขั้นตอนสุดท้าย คือ ทำการวนลูป for ตั้งแต่ ii = 1 ถึง n เพื่อเก็บค่าเอาท์พุท Gold Sequence ลงในเมตริกซ์ gout ทีละแถวตั้งแต่แถวที่ 1 ถึง n ซึ่งมีขั้นตอนดังนี้

- นำ M-sequence ชุดที่ 1 มา exclusive-or กับ M-sequence ชุดที่ 2
- ทำการเลื่อนค่า M-sequence ชุดที่ 2 ไปทางขวา 1 ตำแหน่ง

ดังนั้นในการวนลูปในรอบถัดไปก็เท่ากับว่า M-sequence ชุดที่ 1 ทำการ exclusive-or กับ M-sequence ชุดที่ 2 ในรอบก่อนหน้าที่ถูกเลื่อนตำแหน่งไปแล้ว 1 ตำแหน่งนั่นเองเมื่อสิ้นสุดกระบวนการทั้งหมดก็จะได้เมตริกซ์ gout ขนาด n×(ความยาวของชุดรหัส M-sequence ชุดที่ 1) ที่เก็บค่า Gold Sequence จำนวน n ชุด ซึ่งค่าใน 1 แถวของเมตริกซ์ gout ก็คือ Gold Sequence 1 ชุด

โดยที่ค่า PNcode แบบ gold-sequence ที่ใช้นั้นจะทำการสร้างจาก m-sequence ที่เป็น maximal length 2 ตัวนำมาทำการ xor กันโดยทำการเรียกฟังก์ชั่น goldseq() ขึ้นมา

```
m1 = mseq(5, [2 5], [1 1 1 1 1]);
m2 = mseq(5, [2 3 4 5], [1 1 1 1 1]);
gPNcode = goldseq(m1, m2, 31);
```

3.1.2 ฟังก์ชั่นการชิฟค่า

โดยนำโค้ดจาก [10] โดยเรียกค่าจาก ฟังก์ชัน Shift(inregi, shiftr, shiftu) เป็นฟังก์ชันที่ใช้ในการเลื่อนตำแหน่งค่าในเมตริกซ์ โดย

inregi คือ เมตริกซ์หรือเวกเตอร์ที่ต้องการเลื่อนตำแหน่ง

shiftr คือ จำนวนครั้งในการเลื่อนตำแหน่งวนไปทางขวามือ

shiftu คือ จำนวนครั้งในการเลื่อนตำแหน่งวนไปทางด้านบน

outregi คือ เมตริกซ์เอาท์พุทที่ได้จากการเลื่อนตำแหน่งเรียบร้อยแล้ว

เริ่มต้นด้วยการกำหนดตัวแปร h ให้มีค่าเท่ากับจำนวนแถวของเมตริกซ์ inergi และตัวแปร v มีค่าเท่ากับจำนวนหลักของเมตริกซ์ของ inergi ขั้นตอนถัดมากำหนดเมตริกซ์ outregi เท่ากับเมตริกซ์ inergi กำหนดค่า shiftr เท่ากับค่าเศษที่เหลือจากการหารกันระหว่างค่า shiftr กับ u และกำหนดค่า shiftu เท่ากับค่าเศษที่เหลือจากการหารกันระหว่างค่า shiftu กับ h กระบวนการถัดมาทำการพิจารณาเงื่อนไขต่าง ๆ ดังต่อไปนี้

เงื่อนไขแรกคือ ถ้า shiftr > 0 เป็นจริง จะทำการเลื่อนค่าในแต่ละตำแหน่งวนไปทางขวามือเป็นจำนวน shiftr ครั้ง แต่ถ้าเป็นเท็จจะพิจารณาเงื่อนไขต่อไปที่ว่า ถ้าค่า shiftr < 0 เป็นจริงจะทำการเลื่อนค่าในแต่ละตำแหน่งวนไปทางซ้ายมือเป็นจำนวน | shiftr | ครั้ง และเมื่อพิจารณาเงื่อนไขต่าง ๆ ข้างต้นเสร็จแล้วก็จะทำการกำหนดค่าในเมตริกซ์ inergi ให้เท่ากับเมตริกซ์ outregi ซึ่งเป็นเมตริกซ์ที่ได้ทำการเลื่อนตำแหน่งตามแนวนอนมาเรียบร้อยแล้วหรืออาจจะเป็นกรณีที่ ค่า shiftr = 0

เงื่อนไขต่อมาคือ ถ้า shiftu > 0 เป็นจริงจะทำการเลื่อนค่าในแต่ละตำแหน่งวนขึ้นไปทางด้านบนตามแนวตั้งเป็นจำนวน shiftu ครั้งแต่ถ้าเป็นเท็จก็จะพิจารณาเงื่อนไขที่ว่า ถ้าค่า shiftu < 0 จะทำการเลื่อนค่าในแต่ละตำแหน่งวนลงด้านล่างเป็นจำนวน | shiftu | ครั้ง และเมื่อพิจารณาเงื่อนไขต่าง ๆ ข้างต้นหมดแล้วก็จะได้เมตริกซ์ outergi ที่ได้ถูกทำการเลื่อนตำแหน่งตามที่ต้องการเรียบร้อยแล้ว

<u>หมายเหตุ</u> : ถ้ากำหนด shiftr = 0 หมายความว่า ไม่มีการเลื่อนตำแหน่งในแนวนอน

<u>หมายเหตุ</u> : ถ้ากำหนด shiftu = 0 หมายความว่า ไม่มีการเลื่อนตำแหน่งในแนวตั้ง

3.1.3 ฟังก์ชั่นตรวจค่าออโตร์คอร์ลีเลชั่น

โดยนำโค้ดจาก [10] โดยเรียกค่าจาก ฟังก์ชัน autocorr(indata, tn) เป็นฟังกชันในการคำนวณค่าออโต้คอร์รีเลชันของชุดรหัส โดย

indata คือ ชุดรหัสอินพุท หรือ ชุดรหัสที่ต้องการหาค่าออโต้คอร์รีเลชัน

tn คือ จำนวนคาบ

out คือ ค่าออโต้คอร์รีเลชันที่คำนวณได้

เริ่มต้นด้วยการพิจารณาจำนวณอินพุทของฟังก์ชันกล่าวคือ ถ้าจำนวนอินพุทของฟังก์ชันน้อยกว่า 2 เป็นจริงก็จะกำหนดให้ค่า tn = 1 แล้วจึงทำขั้นตอนถัดไป แต่ถ้าเป็นเท็จก็จะข้ามไปทำขั้นตอนถัดไปทันที ขั้นตอนต่อมา คือ กำหนดตัวแปร ln ให้มีค่าเท่ากับความยาวของ indata และกำหนดให้เมตริกซ์ out เป็นเมตริกซ์ศูนย์ขนาด $1\times(\ln\times tn)$

และขั้นตนสุดท้าย คือ ทำการวนลูป for ตั้งแต่ ii = 0 ถึง $(\ln\times tn) - 1$ เพื่อเก็บค่าออโต้คอร์รีเลชันที่คำนวณในเมตริกซ์ gout โดยคำนวณมาจากผลรวมของผลคูณระหว่าง ชุดรหัสอินพุทกับชุดรหัสอินพุทที่ถูกเลื่อนตำแหน่งไปทางขวา ii ตำแหน่ง เมื่อสิ้นสุดกระบนการทั้งหมดก็จะได้ค่าออโต้คอร์รีเลชันทั้งหมด $\ln\times tn$ ค่าที่เกิดจากชุดรหัสดังกล่าวโดยเก็บอยู่ในเมตริกซ์ out

เช่น stem(autocorr(m1)); % เป็นการพลอตค่าออกมาให้ดู

3.1.4 ฟังก์ชั่นตรวจค่าครอสคอร์ลีเลชั่น

โดยนำโค้ดจาก [10] โดยเรียกค่าจาก ฟังก์ชัน crosscorr(indata 1, indata 2, tn) เป็นฟังกชันในการคำนวณค่า crosscorrelation ระหว่างอินพุท 2 ชุด โดย

indata 1 คือ ชุดรหัสอินพุทชุดที่ 1

indata 2 คือ ชุดรหัสอินพุทชุดที่ 2

tn คือ จำนวนคาบ

out คือ ค่าออโต้คอร์รีเลชันที่คำนวณได้

เริ่มต้นด้วยการพิจารณาจำนวณอินพุทของฟังก์ชันกล่าวคือ ถ้าจำนวนอินพุทของฟังก์ชันน้อยกว่า 3 ตัว เป็นจริงก็จะกำหนดให้ค่า tn = 1 แล้วจึงทำขั้นตอนถัดไป แต่ถ้าเป็นเท็จก็จะข้ามไปทำขั้นตอนถัดไปทันที ขั้นตอนต่อมา คือ กำหนดตัวแปร ln ให้มีค่าเท่ากับความยาวของ indata และกำหนดให้เมตริกซ์ out เป็นเมตริกซ์ศูนย์ขนาด $1\times(\ln\times tn)$

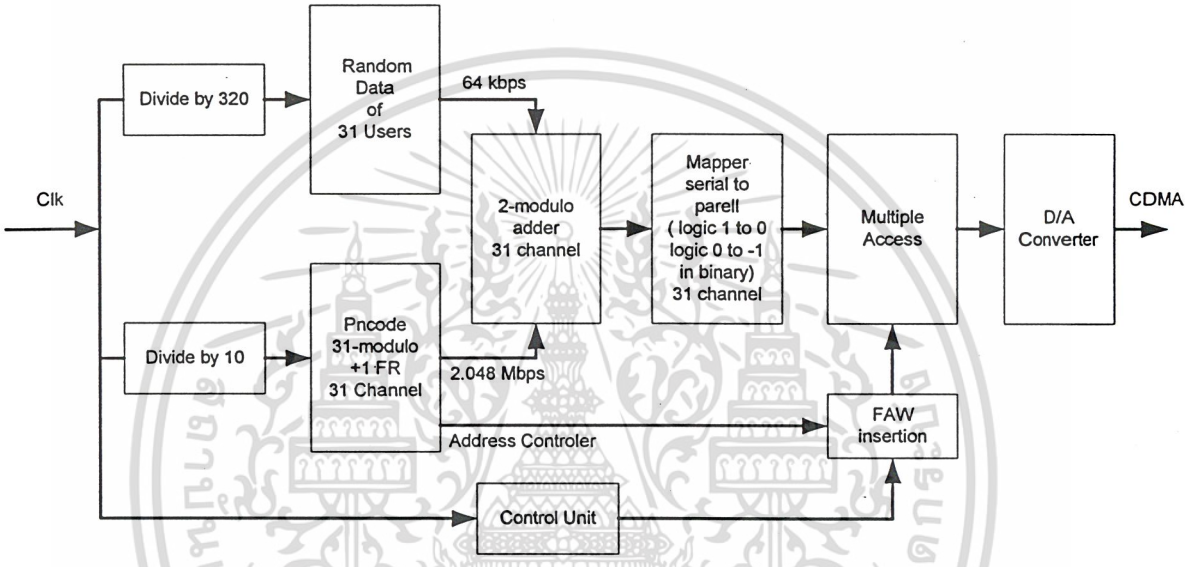
และขั้นตนสุดท้าย คือ ทำการวนลูป for ตั้งแต่ ii = 0 ถึง $(\ln\times tn) - 1$ เพื่อเก็บค่า crosscorrelation ที่คำนวณในเมตริกซ์ gout โดยคำนวณมาจากผลรวมของผลคูณระหว่าง รหัสชุดแรกกับรหัสชุดที่ 2 ที่ถูกเลื่อนตำแหน่งไปทางขวา ii ตำแหน่ง เมื่อสิ้นสุดกระบนการทั้งหมดก็จะได้ค่าcrosscorrelation ทั้งหมด $\ln\times tn$ ค่า ที่เกิดจากรหัส 2 ชุดดังกล่าวโดยเก็บอยู่ในเมตริกซ์ out

เช่น stem(crosscorr(m1,m2)); % เป็นการพลอตค่าออกมาให้ดู

3.2 การออกแบบวงจรและการสร้างด้วยภาษา VHDL

3.2.1 การออกแบบวงจรภาคส่ง

การออกแบบวงจรในด้านส่งนั้น ได้ออกแบบให้สัญญาณข่าวสารเป็นไปตามมาตรฐานของ PCM ที่ใช้ในระบบ E1 ซึ่งได้กำหนดความเร็วของ output ไว้ที่ 64 kbps และ ออกแบบชุดของ PN code ที่จะในมาใช้เข้ารหัสแบบ direct sequence แบบ 31-modulo ซึ่งจะสามารถรองรับผู้ใช้ได้พร้อมกันมากที่สุด 31 คน ซึ่งจะมีการใส่ FAW (Frame Alignment Word) ให้กับข้อมูลทุกๆบิต



รูปที่ 3.1 block diagram ทางด้านส่ง

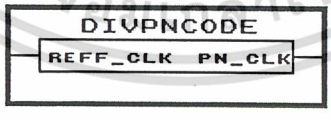
โดยที่ตามหลักการแล้วค่าความเร็วที่ใช้ในการสร้าง PN (chip rate) นั้นจะมีค่าเป็นจำนวน N-modulo เท่าของสัญญาณข้อมูลข่าวสารซึ่ง chip rate ต้องมีค่าเท่ากับ 64 kbps*(31+1) = 2.048 Mbps โดยการออกแบบวงจรที่จะทำการกำเนิดค่าข้อมูล กับค่า PN code ที่ใช้ในการทดลองในครั้งนี้จะได้มาจากการคำนวณค่าด้วยโปรแกรม MATLab และ จากโปรแกรม MAX+plus2 โดยเทคนิค LFSR(Linear Feedback Shift Register) จากนั้นนำค่าที่ได้มาทำการบรรยายการทำงานเป็น code ด้วยภาษา VHDL จากนั้นทำการ simulate และสังเคราะห์วงจรด้วยโปรแกรม MAX+plus2 ซึ่งจะแสดงรายละเอียดดังต่อไปนี้

3.2.1.1 วงจรหารความถี่

3.2.1.1.1 วงจรหารความถี่สำหรับใช้ในการสร้าง PN code จะทำการสร้างโดยใช้ VHDL code โดยออกแบบให้เป็น counter โดยนับค่าค่า 10 จากโครงสร้างภาษาโดยจะพบว่าจะเริ่มทำงานที่ขอบขาขึ้น โดยจะทำการนับค่า input clock ที่มีค่า 20.48 MHz จาก Crystal Oscillator โดยเมื่อนับค่า 5 ลูกแรกจะทำการส่งค่า logic 1 ออกมาและส่งค่า logic 0 ออกมาเมื่อนับค่าลูกที่ 6 ถึง 10 ก็จะเริ่มนับใหม่อย่างนี้ไปเรื่อยๆ โดยจะได้ความถี่ออกมาคือ 2.048 MHz

```
library ieee;
use ieee.std_logic_1164.all;

entity DivPNcode is
port(Reff_clk : in  std_logic;
       PN_clk  : out std_logic);
end;

architecture Clkosc of DivPNcode is
signal count:integer range 0 to 9;
begin
process(Reff_clk)
begin
     if Reff_clk'event and Reff_clk = '1' then
          if   count /= 9 then
               count <= count+1;
          else count <=0;
          end if;
          if count < 5 then
               PN_clk <= '1';
          else PN_clk <= '0';
          end if;
     end if;
end process;
end clkosc;
```

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.2 block หาร 40 ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

3.2.1.1.2 วงจรหารความถี่สำหรับใช้ในการสร้างข้อมูล จะทำการสร้างโดยใช้ VHDL code โดยออกแบบให้เป็น counter โดยนับค่าค่า 320 จากโครงสร้างภาษาโดยจะพบว่าจะเริ่มทำงานที่ขอบขาขึ้น โดยจะทำการนับค่า input clock ที่มีค่า 20.48 MHz จาก Crystal Oscillator โดยเมื่อนับค่า 160 ลูกแรกจะทำการส่งค่า logic 1 ออกมาและส่งค่า logic 0 ออกมาเมื่อนับค่าลูกที่ 161 ถึง 320 ก็จะเริ่มนับใหม่อย่างนี้ไปเรื่อยๆ โดยจะได้ความถี่ออกมาคือ 64 kHz

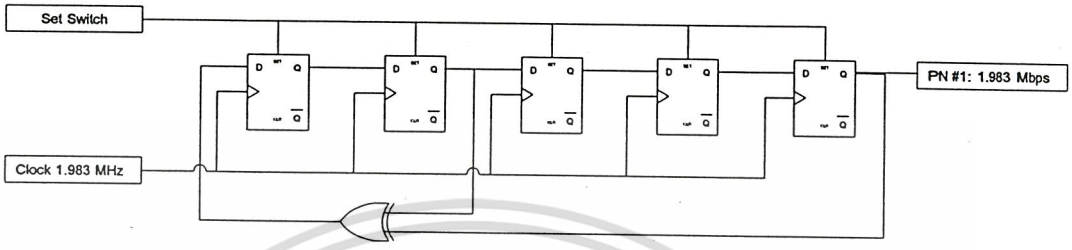
```
library ieee;
use ieee.std_logic_1164.all;

entity DivData is
      port(Reff_clk : in  std_logic;
           Data_clk : out std_logic);
end;

architecture clkosc of DivData is
signal count:integer range 0 to 319;
begin
process(Reff_clk)
begin
      if Reff_clk'event and Reff_clk = '1' then
            if   count /= 319 then
                 count <= count+1;
            else count <=0;
            end if;
            if count < 160 then
                 data_clk <= '1';
            else data_clk <= '0';
            end if;
      end if;
end process;
end clkosc;
```

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา

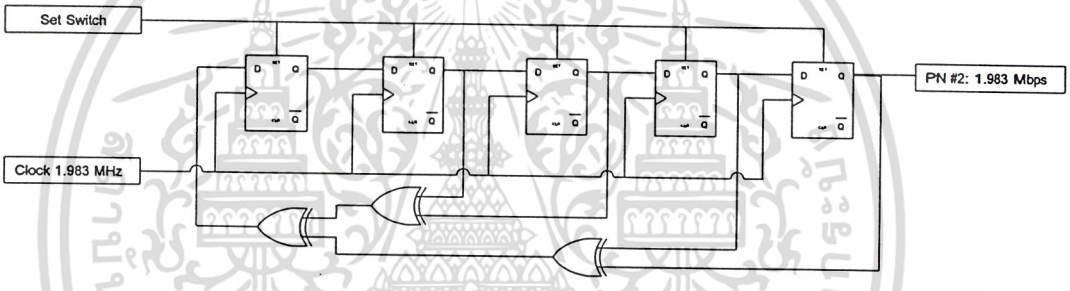
รูปที่ 3.3 block หาร 320 ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

3.2.1.2 วงจรสร้างสัญญาณแบบ Random

3.2.1.2.1 วงจรสร้างสัญญาณแบบ random ของสัญญาณ PN code โดยใช้ LFSR โดยใช้ทั้งหมด 5 state โดยทำการ feed back จาก state 5 กับ 2 และ state 5 4 3 และ 2 ตามลำดับดังเช่นในภาพที่ 3.4 และ 3.5 ตามลำดับ



รูปที่ 3.4 วงจรกำเนิดค่า PNcode แบบ m-sequence 31-modulo(1) ความเร็ว 1.983 Mbps



รูปที่ 3.5 วงจรกำเนิดค่า PNcode แบบ m-sequence 31-modulo(2) ความเร็ว 1.983 Mbps

และนำเอาทต์พุทมาทำการ XOR กันจะได้ผลออกมาเป็น PN code แบบ Gold-sequence ซึ่งเราจะทำการคำนวณจากโปรแกรม MATLab โดยฟังก์ชั่นดังต่อไปนี้

```
M1 = mseq(5, [5 2], [1 1 1 1 1]);
M2 = mseq(5, [5 4 3 2], [1 1 1 1 1]);
G1 = goldseq(M1,M2,31);
```

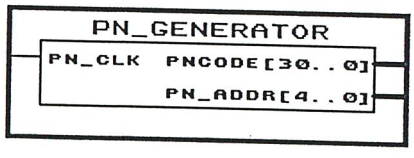
ซึ่งนำผลลัพธ์ที่ได้จากวงจรกำเนิด data แบบ 31-modulo มาจะได้เป็น Matrix ขนาด 31x31 ออกมานำไป Transpose แล้วนำมาบรรยายเป็นโค๊ดด้วยภาษา VHDL ซึ่งเพื่อนำไปทำการสร้างชิ้นงานซึ่งจะมีรายละเอียดดังต่อไปนี้

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity PN_Generator1 is
port (PN_Clk  : in  std_logic;
      PNCode  : out std_logic_vector(30 downto 0);
      PN_ADDR : out std_logic_vector(4 downto 0));
end;

ARCHITECTURE construction OF PN_Generator1 IS
signal count    : integer  range 0 to 123;
BEGIN
PNGenerater: Process(PN_Clk)
Begin
IF PN_Clk'event AND PN_Clk = '1' then
      IF  count /= 31 then
          count <= count+1;
      ELSE count <=0;
      END IF;

CASE count IS
   When  0 =>
      PNCode  <= "1111111111111111111111111111111";
      PN_ADDR <= "00000";

   When  1 =>
      PNCode  <= "0100011101010010111100110110000";
      PN_ADDR <= "00001";

   When  2 =>
      PNCode  <= "0010001110101001011110011011000";
      PN_ADDR <= "00010";

   When  3 =>
      PNCode  <= "0001000111010100101111001101100";
      PN_ADDR <= "00011";

   When  4 =>
      PNCode  <= "0000100011101010010111100110110";
      PN_ADDR <= "00100";

   When  5 =>
      PNCode  <= "0000010001110101001011110011011";
      PN_ADDR <= "00101";

   When  6 =>
      PNCode  <= "0111110111000101011010000110010";
      PN_ADDR <= "00110";

   When  7 =>
      PNCode  <= "0011111011100010101101000011001";
      PN_ADDR <= "00111";

   When  8 =>
      PNCode  <= "0110000010001110101001011110011";
      PN_ADDR <= "01000";

   When  9 =>
      PNCode  <= "1011000001000111010100101111001";
      PN_ADDR <= "01001";
```

```
When 10 =>
   PNCode  <= "00100111111011100010101101000011";
   PN_ADDR <= "01010";

When 11 =>
   PNCode  <= "01101100000100011101010010111110";
   PN_ADDR <= "01011";

When 12 =>
   PNCode  <= "11001001111101110001010110100000";
   PN_ADDR <= "01100";

When 13 =>
   PNCode  <= "01100100111111011100010101101000";
   PN_ADDR <= "01101";

When 14 =>
   PNCode  <= "11001101100000100011110101001011";
   PN_ADDR <= "01110";

When 15 =>
   PNCode  <= "00011001001111110111000101011010";
   PN_ADDR <= "01111";

When 16 =>
   PNCode  <= "00001100100111111011100010101101";
   PN_ADDR <= "10000";

When 17 =>
   PNCode  <= "10000110010011111101110001010110";
   PN_ADDR <= "10001";

When 18 =>
   PNCode  <= "01000011001001111110111000101011";
   PN_ADDR <= "10010";

When 19 =>
   PNCode  <= "01011110011011000001000111010110";
   PN_ADDR <= "10011";

When 20 =>
   PNCode  <= "11010000110010011111101110001010";
   PN_ADDR <= "10100";

When 21 =>
   PNCode  <= "10010111100110110000010001110100";
   PN_ADDR <= "10101";

When 22 =>
   PNCode  <= "10110100001100100111111011100010";
   PN_ADDR <= "10110";

When 23 =>
   PNCode  <= "10100101111001101100000100011100";
   PN_ADDR <= "10111";

When 24 =>
   PNCode  <= "01010010111100110110000001000111";
   PN_ADDR <= "11000";
```

```
When 25 =>
    PNCode  <= "10101001011110011011000001000011";
    PN_ADDR <= "11001";

When 26 =>
    PNCode  <= "00101011010000110010011111011110";
    PN_ADDR <= "11010";

When 27 =>
    PNCode  <= "11101010010111100110110000010000";
    PN_ADDR <= "11011";

When 28 =>
    PNCode  <= "01110101001011110011011000001000";
    PN_ADDR <= "11100";

When 29 =>
    PNCode  <= "11000101011010000110010011111101";
    PN_ADDR <= "11101";

When 30 =>
    PNCode  <= "11100010101101000011001001111110";
    PN_ADDR <= "11110";

When 31 =>
    PNCode  <= "01110001010110100001100100111111";
    PN_ADDR <= "11111";

When others =>

end case;
End IF;
END;
END;
```

โค๊ดข้างต้นเป็นการบรรยาการทำงานของตัว Gold-Sequence generator แบบ 31-modulo ซึ่งได้ผลมาจากโปรแกรม MATLab โดยทำการแทรก Fake Frame Alignment Word ซึ่งมีค่าเป็น 1 เข้าไปเป็นตัวแรกของทุกชุดโค๊ด โดยการทำงานนั้นโปรแกรมจะทำการสร้าง Address ขึ้นมาเพื่อเป็นตัวระบุว่าขณะนี้กำลังใช้รหัสตัวที่เท่าไรเพื่อที่จะนำไปทำการใส่ Frame Alignment word ที่แท้จริงหลังจากการทำ Multiple Access ซึ่งทำการสังเคราะห์ออกมาเป็นตัวอุปกรณ์ ในโปรแกรม MAX+plus II ได้ดังภาพ



รูปที่ 3.6 PN_Generator ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

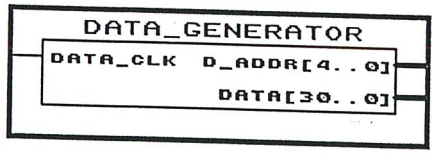
3.2.1.2.2 วงจรสร้างสัญญาณแบบ random ของสัญญาณข่าวสารซึ่งสัญญาณข่าวสารที่ได้จะได้มาจากการทำการใช้คำสั่ง ceil(2*rand(31,31)) ซึ่งเป็นการทำการสร้างตารางข้อมูลที่มีค่าโลจิกเป็น 0 กับ 1 ขนาด 31x31 จากนั้นนำมาบรรยายเป็นโค๊ดด้วยภาษา VHDL ซึ่งเพื่อนำไปทำการสร้างชิ้นงานซึ่งจะมีรายละเอียดดังต่อไปนี้

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity DATA_generator1 is
port (DATA_Clk : in  std_logic;
      D_ADDR   : out std_logic_vector(4  downto 0);
      DATA1    : out std_logic_vector(30 downto 0));
end;

ARCHITECTURE construction OF DATA_generator1 IS
      signal count : integer  range 0 to 123;
BEGIN
PNGenerater: Process(DATA_Clk)
Begin
IF DATA_Clk'event AND DATA_Clk = '1' THEN
      IF  count /= 31 then
          count <= count+1;
      ELSE count <=0;
      END IF;
Case count Is
   When 0 =>
      DATA1   <= "1100011010101000100101100100001";
      D_ADDR  <= "00000";

   When 1 =>
      DATA1   <= "0101010101010000000110100110100";
      D_ADDR  <= "00001";

   When  2 =>
      DATA1   <= "1111100010001001001001001001101";
      D_ADDR  <= "00010";

   When  3 =>
      DATA1   <= "1100010101100111110000011010111";
      D_ADDR  <= "00011";

   When  4 =>
      DATA1   <= "1110110100010001111110101111110";
      D_ADDR  <= "00100";

   When  5 =>
      DATA1   <= "1000011010101111111001110101000";
      D_ADDR  <= "00101";

   When  6 =>
      DATA1   <= "1000011110011100110101100110000";
      D_ADDR  <= "00110";

   When  7 =>
      DATA1   <= "1010111011101111001010101110101";
      D_ADDR  <= "00111";
```

```
When  8 =>
   DATA1   <= "00100001010000111000111010111101";
   D_ADDR  <= "01000";

When  9 =>
   DATA1   <= "00011110101000101111011000110010";
   D_ADDR  <= "01001";

When 10 =>
   DATA1   <= "10111110101101010011010011011011";
   D_ADDR  <= "01010";

When 11 =>
   DATA1   <= "01010000000010001111100111001100";
   D_ADDR  <= "01011";

When 12 =>
   DATA1   <= "01010111101001000001111100010110";
   D_ADDR  <= "01100";

When 13 =>
   DATA1   <= "01100010100101001001110000010100";
   D_ADDR  <= "01101";

When 14 =>
   DATA1   <= "01001010101001001111110011111100";
   D_ADDR  <= "01110";

When 15 =>
   DATA1   <= "00111100111111100010100101111010";
   D_ADDR  <= "01111";

When 16 =>
   DATA1   <= "01110101110110001111110010110101";
   D_ADDR  <= "10000";

When 17 =>
   DATA1   <= "10000010011111111010100110000001";
   D_ADDR  <= "10001";

When 18 =>
   DATA1   <= "11000110011111111011011010001001";
   D_ADDR  <= "10010";

When 19 =>
   DATA1   <= "11001000001100101010100011010010";
   D_ADDR  <= "10011";

When 20 =>
   DATA1   <= "00010110000001101101000100000010";
   D_ADDR  <= "10100";

When 21 =>
   DATA1   <= "10111010010111011001011011011111";
   D_ADDR  <= "10101";

When 22 =>
   DATA1   <= "11000000100110111110110010010111";
   D_ADDR  <= "10110";
```

```
When 23 =>
   DATA1   <= "10110111001000001000111010011101";
   D_ADDR  <= "10111";

When 24 =>
   DATA1   <= "01000101111110111011001110111000";
   D_ADDR  <= "11000";

When 25 =>
   DATA1   <= "00111101000011000000111110100000";
   D_ADDR  <= "11001";

When 26 =>
   DATA1   <= "01001010101100011101111000010010";
   D_ADDR  <= "11010";

When 27 =>
   DATA1   <= "10101100101000111111111001010100";
   D_ADDR  <= "11011";

When 28 =>
   DATA1   <= "01110100100111000111010110110101";
   D_ADDR  <= "11100";

When 29 =>
   DATA1   <= "01101110110111111111111011011110";
   D_ADDR  <= "11101";

When 30 =>
   DATA1   <= "01010001111001010000010010011011";
   D_ADDR  <= "11110";

When 31 =>
   DATA1 <= "11010111010110100010111111100010";
   D_ADDR  <= "11111";

When others =>

end case;
End IF;
END;
END;
```

โค๊ดข้างต้นเป็นการบรรยายการทำงานของตัว data generator แบบ 31-user ซึ่งได้ผลมาจากโปรแกรม MATLab โดยการทำงานนั้นโปรแกรมจะทำการสร้าง Address ขึ้นมาเพื่อเป็นตัวระบุว่าขณะนี้เป็นสัญญาณตัวที่เท่าไรเพื่อที่จะง่ายต่อการตรวจสอบ ซึ่งทำการสังเคราะห์ออกมาเป็นตัวอุปกรณ์ ในโปรแกรม MAX+plus II ได้ดังภาพ

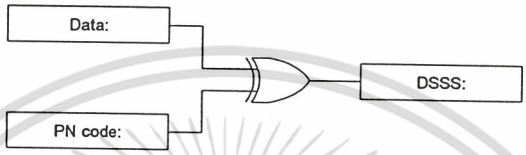


รูปที่ 3.7 block data ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

3.2.1.3 วงจรเข้ารหัสสัญญาณแบบสเพรดสเปกตรัม

3.2.1.3.1 หลักการทำงานวงจรเข้ารหัสสัญญาณแบบสเพรดสเปกตรัม

ในส่วนของวงจรนี้จะนำสัญญาณข้อมูลมาทำการเข้ารหัสกับสัญญาณ PN code โดยมีหลักการคือเราจะทำการเลือกค่า PN code 1 (จำนวน n ค่าของ modulo) ชุดต่อข้อมูล 1 bit โดยในการทดลองครั้งนี้เราจะพบว่าเราเลือกค่า PN เป็นแบบ 31-modulo ดังนั้นเราจะต้องใช้อัตราในการเข้า code [31, 1] โดยในระบบ Direct Sequence Spread Spectrum โดยจะนำ Data มาทำการเข้ารหัสกับ PN code โดยใช้การวงจร Exclusive-OR



รูปที่ 3.8 วงการเข้ารหัสแบบสเพรดสเปกตรัม

ดังในตัวอย่างเช่นหาก กำหนดให้ค่า data ของ user ทั้งสามเป็น

D0 = [0 0 1], D1 = [0 1 0], D2 = [1 0 1]

จากหัวข้อที่ 3.2.1.2.1 จะได้ค่า PN code ออกทำการเลือกค่ามา 3 ชุดจะเป็นได้ดังนี้

PN01= [1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0]

PN13= [1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1]

PN31= [1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1]

ตัวอย่างที่1: นำ data กับ PN code มาทำการเข้ารหัสกัน

User #1:

D0 = [0 0 1]

PN01= [1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0]

ข้อมูลบิตที่ 1	D0(0)	=	0
	D0(0)	=	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PN code ชุดที่ 1	PN01	=	1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0
ทำการ XOR	DSSS0 (0)	=	1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0
ข้อมูลบิตที่ 2	D0(1)	=	0
	D0(1)	=	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PN code ชุดที่ 1	PN01	=	1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0
ทำการ XOR	DSSS0 (1)	=	1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

ข้อมูลบิตที่ 3 D0(2) = 1

D0(2) = 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 1 1 1 1 1 1 1

PN code ชุดที่ 1 PN01 = 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

ทำการ XOR DSSS0 (2) = 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1

User #2:

D1 = [0 1 0]

PN13= [1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1]

ข้อมูลบิตที่ 1 D1 (0) = 0

D1 (0) = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PN code ชุดที่ 2 PN13 = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

ทำการ XOR DSSS1 (0) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

ข้อมูลบิตที่ 2 D1 (1) = 1

D1 (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 1 1 1 1 1 1 1 1

PN code ชุดที่ 2 PN13 = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

ทำการ XOR DSSS1 (1) = 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1 0

ข้อมูลบิตที่ 3 D1 (2) = 0

D1 (2) = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PN code ชุดที่ 2 PN13 = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

ทำการ XOR DSSS1 (2) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

User #3:

D2 = [1 0 1]

PN31= [1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1]

ข้อมูลบิตที่ 1 D2 (0) = 1

D2 (0) = 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 1 1 1 1 1 1 1

PN code ชุดที่ 3 PN31 = 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

ทำการ XOR DSSS2 (0) = 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0

ข้อมูลบิตที่ 2 D1 (1) = 0

D1 (1) = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PN code ชุดที่ 3 PN31 = 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

ทำการ XOR DSSS2 (1) = 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

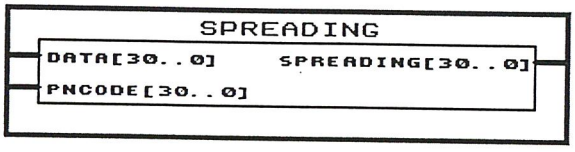
ข้อมูลบิตที่ 3	D2(2)	=	1
	D2 (2)	=	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
PN code ชุดที่ 3	PN31	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
ทำการ XOR	DSSS2 (2)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0

3.2.1.3.2 การออกแบบวงจรเข้ารหัสสัญญาณแบบสเพรดสเปกตรัม

เราจะสามารถทำการออกแบบวงที่ทำการสเพรดสัญญาณแบบ 31 ช่องสัญญาณได้โดย ทำการบรรยายด้วยโค้ด VHDL

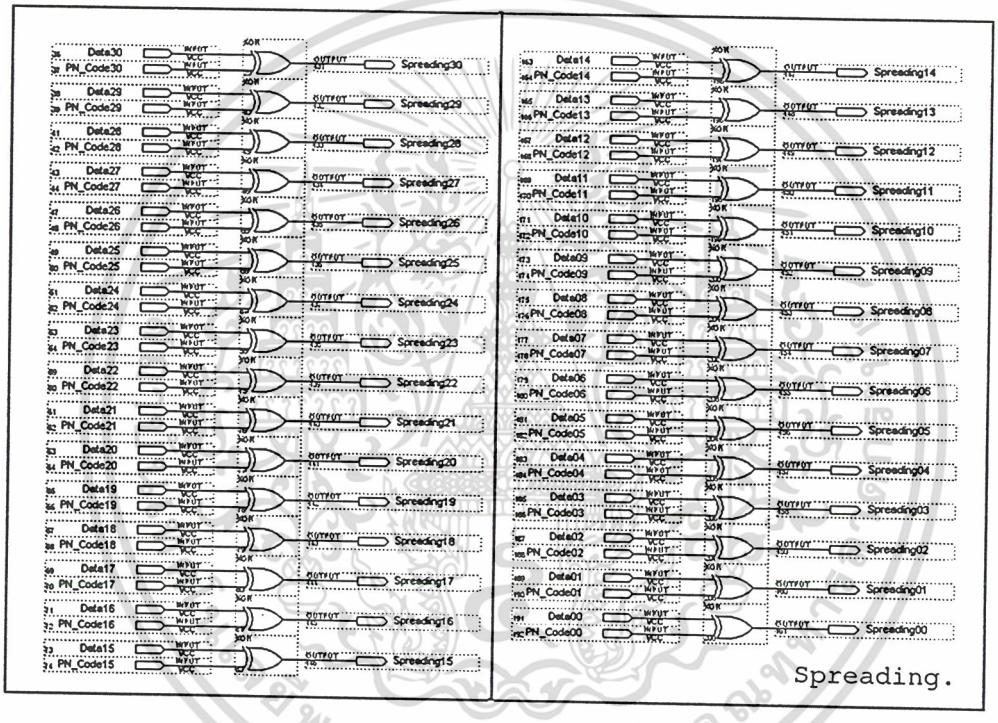
```
library ieee;
use  ieee.std_logic_1164.all;

entity spreading is
  port (       data : in  std_logic_vector (30 downto 0);
             pncode: in  std_logic_vector (30 downto 0);
          spreading : out std_logic_vector (30 downto 0));
end;

ARCHITECTURE spread OF spreading IS
BEGIN
     spreading(30) <= data(30) xor pncode(30);
     spreading(29) <= data(29) xor pncode(29);
     spreading(28) <= data(28) xor pncode(28);
     spreading(27) <= data(27) xor pncode(27);
     spreading(26) <= data(26) xor pncode(26);
     spreading(25) <= data(25) xor pncode(25);
     spreading(24) <= data(24) xor pncode(24);
     spreading(23) <= data(23) xor pncode(23);
     spreading(22) <= data(22) xor pncode(22);
     spreading(21) <= data(21) xor pncode(21);
     spreading(20) <= data(20) xor pncode(20);
     spreading(19) <= data(19) xor pncode(19);
     spreading(18) <= data(18) xor pncode(18);
     spreading(17) <= data(17) xor pncode(17);
     spreading(16) <= data(16) xor pncode(16);
     spreading(15) <= data(15) xor pncode(15);
     spreading(14) <= data(14) xor pncode(14);
     spreading(13) <= data(13) xor pncode(13);
     spreading(12) <= data(12) xor pncode(12);
     spreading(11) <= data(11) xor pncode(11);
     spreading(10) <= data(10) xor pncode(10);
     spreading(9)  <= data(9)  xor pncode(9);
     spreading(8)  <= data(8)  xor pncode(8);
     spreading(7)  <= data(7)  xor pncode(7);
     spreading(6)  <= data(6)  xor pncode(6);
     spreading(5)  <= data(5)  xor pncode(5);
     spreading(4)  <= data(4)  xor pncode(4);
     spreading(3)  <= data(3)  xor pncode(3);
     spreading(2)  <= data(2)  xor pncode(2);
     spreading(1)  <= data(1)  xor pncode(1);
     spreading(0)  <= data(0)  xor pncode(0);
END spread;
```

ซึ่งเป็นการทำการบวกแบบ 2-modulo ในอัตราส่วน 31:1 (PNcode:data) จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.9 block Spreading ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

ซึ่งภายในจะประกอบไปด้วย XOR จำนวน 31 ตัวตามวงจรดังต่อไปนี้



รูปที่ 3.10 วงจรภายในของตัว Spreading ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

3.2.1.4 วงจรการทำ Multiple Access

3.2.1.4.1 การทำการวิเคราะห์กระบวนการสร้าง

จากทฤษฎีในการทำ Multiple access นั้น เราจะต้องนำสัญญาณจากข้อที่ 3.2.1.3 มาทำการบวกกันในแต่ละบิต แต่จะเกิดปัญหาขึ้นเนื่องจากสัญญาณ digital ที่ออกจากตัว FPGA นั้นจะมีแค่ 2 ระดับคือ logic 1(+5V), logic 0(0V) จึงทำการแก้ปัญหาโดยทำการแปลงค่าและเพิ่มจำนวนบิต

- พิจารณา mapping ค่า logic ให้เป็นค่า voltage เสมือน
- พิจารณาจำนวนบิตที่เพิ่มขึ้นเนื่องจากในแต่ละ user

3.2.1.4.1.1 การพิจารณาค่า logic ให้เป็นสัญญาณ voltage เสมือน

จำเป็นต้องทำเนื่องจากทางด้านรับนั้นเราไม่สามารถทำการกู้วงจรกลับได้ด้วยการทำ Exclusive-OR หลังจากการทำ Serial to Parallel ได้ทางด้านรับจึงต้องใช้ operation การคูณแทน โดยการทำ mapping ขั้นแรกนั้นเราจะทำการหาความสัมพันธ์ระหว่างการคูณกับการทำ Exclusive-OR เนื่องจากการนำค่า 2 ค่ามากระทำกันจะมีทั้งหมดแค่ 4 กรณี

D1	D2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

ตารางที่ 3.1 ตารางค่าความจริงของ 2 input ทำการ XOR กัน

โดยเราจะทำการสมมติให้ logic 1 -> 1 V

logic 0 -> -1 V

ลองทำการเขียนค่าตารางความจริงดูจะได้ว่า

D1	D2	multiply
-1	-1	1V
-1	1	-1V
1	-1	-1V
1	1	1V

ตารางที่ 3.2 ตารางค่าความจริงของ 2 input ของ voltage เสมือนทำการคูณกัน

จะพบว่าค่าความสัมพันธ์ที่ได้จะไม่เป็นไปตามวงจร Exclusive-OR ดังนั้นลองทำอีกวิธีหนึ่ง

โดยเราจะทำการสมมติให้ logic 1 -> -1

logic 0 -> 1

ทำการเขียนค่าตารางความจริงดูจะได้ว่า

D1	D2	multiply
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1

ตารางที่ 3.3 ตารางค่าความจริงของ 2 อินพุท ของ โวลท์เตจ เสมือนทำการคูณกัน

จะพบว่าค่าความสัมพันธ์ที่ได้จะเป็นไปตามวงจร Exclusive-OR

3.2.1.4.1.2 พิจารณาจำนวนบิตที่เพิ่มขึ้นเนื่องจากแต่ละ user

จากข้อที่ 3.1.5.1 เราจะพบว่าค่า logic 1 -> -1V, logic 0 -> 1V แต่เราจะพบว่าจากหัวข้อที่ 3.2.1.5 ว่าวงจร FPGA ที่เราใช้นั้นจะมีระดับสัญญาณเพียง 2 ระดับคือ 5V และ 0V แต่เราจำเป็นจะต้องใช้ค่า ที่เป็น logic ในการแทนค่าสัญญาณ 1,-1 ด้วยวิธีการ 2' complement โดยจำนวนบิตที่ใช้นั้นเราจะต้องทำการพิจารณาจากค่าสูงสุดต่ำสุดของการรวมค่าของแต่ละบิตในแต่ละ user โดยความเป็นไปได้ที่เกิดขึ้นในระบบของเราที่ออกแบบให้รองรับ user ได้ทั้งหมด 31 user

- กรณีแต่ละ user ส่งค่าสูงสุดคือ 1 ซึ่งจะให้ค่าสูงสุดรวมของระบบคือ 31
- กรณีแต่ละ user ส่งค่าต่ำสุดคือ -1 ซึ่งจะให้ค่าต่ำสุดรวมของระบบคือ -31

จากนั้นเรามาพิจารณาค่าที่ได้จาก 2 ข้อดังกล่าวข้างต้นหากค่าสูงสุดที่ระบบต้องใช้คือ 31 ลองทำการแปลงเป็นเลขฐานสองดูจะได้ว่า 31 => 1 1 1 1 1 ใช้เลข 5 บิตแต่ว่าเรามีค่าต่ำสุดคือ -31 ดังนั้นจึงต้องทำการเพิ่มค่าเข้าไปอีก 1 บิตเพื่อให้เป็น sign bit ในรูปแบบของ 2'complement

- ซึ่งจะได้ค่า 31 => 0 1 1 1 1 1
- และจะได้ค่า -31 => 1 0 0 0 0 1

แสดงว่าหากทำการใช้สัญญาณ binary แบบ 6 bit ในการแทนค่าเป็นสัญญาณ voltage เสมือนนั้นเป็นที่ยอมรับได้ ในการออกแบบวงจรที่จะทำการ map ค่านั้นต้องเป็นไปตามคุณสมบัติตามข้อ 3.2.1.5.1, 3.2.1.5.2 และ ควรจะมีความซับน้อยที่สุดในการทำวงจร

3.2.1.4.2 กระบวนการสร้างแปลงระดับสัญญาณเสมือน

ออกแบบวงจรควรจะทำให้ซับซ้อนน้อยที่สุด ดังนั้นการทำการ Map ค่าควรจะทำตั้งแต่ในส่วนที่สัญญาณข้อมูลข่าวสารผ่านการเข้า code ด้วยสัญญาณ PN ออกมา (DSSS) ตามรูปที่ 3.1 ซึ่งจะมีค่า output ที่ออกมาเพียง 2 ค่าคือ logic 0 กับ logic 1 ซึ่งจะทำให้การ map ค่าในส่วนนี้จะทำให้วงจรที่ได้ออกมามีขนาดเล็กที่สุดโดยจะทำการแปลง

Logic 0 -> 0 0 0 0 0 1

Logic 1 -> 1 1 1 1 1 1

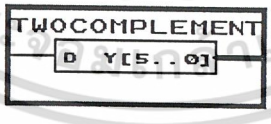
โดยจะออกแบบให้เป็นการแปลงแบบ serial to parallel ด้วยเหตุผล 2 อย่างคือ

- สามารถใช้รวมกันกับอุปกรณ์สำเร็จรูปตัวอื่นๆที่มีอยู่ในโปรแกรม
- ไม่จำเป็นต้องเพิ่มความเร็วในการส่งให้มากขึ้น

จะทำการออกแบบโดยการบรรยายเป็น vhdl code ตามข้อความด้านล่าง

```
LIBRARY    ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Twocomplement IS
    PORT(
        d       : in Std_Logic;
        Y       : out Std_Logic_vector(5 downto 0));
    END Mapping;

    ARCHITECTURE Mapper OF Mapping IS
    BEGIN
                Y <=   "111111" WHEN ('d=1') ELSE
                       "000001" WHEN ('d=0') ELSE
                       "000000";
    END Mapper;
```

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.11 block การ map ค่าที่ได้จากการทำการ synthesis จากโปรแกรม MAX+plus II

ตัวอย่างที่ 2: ดังเช่นในตัวอย่างจากการเข้า code ในตัวอย่างที่ 1 เราจะได้ว่า

User #1: data = [0 0 1]

DSSS0 (0) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

DSSS0 (1) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

DSSS0 (2) = 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1

User #2: data = [0 1 0]

DSSS1 (0) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

DSSS1 (1) = 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0

DSSS1 (2) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

User #3: data = [1 0 1]

DSSS2 (0) = 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0

DSSS2 (1) = 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

DSSS2 (2) = 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0

นำมาทำการแปลงเป็นค่า voltage เสมือนที่แสดงอยู่ในรูป binary 6 บิต (ตามแนวตั้ง) จะได้ว่า:

User #1: data = [0 0 1]

DSSS0(0) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

MSB D00(5) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

.D00(4) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D00(3) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D00(2) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D00(1) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

LSB D00(0) = 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 1 1 1 1 1 1 1 1 1 1 1

DSSS0(1) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

MSB D01(5) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D01(4) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D01(3) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D01(2) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

D01(1) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

LSB D01(0) = 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 1 1 1 1 1 1 1 1 1 1 1

	DSSS0 (2)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
MSB	D02(5)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
	D02(4)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
	D02(3)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
	D02(2)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
	D02(1)	=	0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1
LSB	D02(0)	=	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 1 1 1 1 1 1 1 1

User #2: data = [0 1 0]

	DSSS1 (0)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
MSB	D10(5)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D10(4)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D10(3)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D10(2)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D10(1)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
LSB	D10(0)	=	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 1 1 1 1 1 1 1 1
	DSSS1 (1)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
MSB	D11(5)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
	D11(4)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
	D11(3)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
	D11(2)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
	D11(1)	=	0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
LSB	D11(0)	=	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 1 1 1 1 1 1 1 1
	DSSS1 (2)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
MSB	D12(5)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D12(4)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D12(3)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D12(2)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
	D12(1)	=	1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1
LSB	D12(0)	=	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 1 1 1 1 1 1 1 1

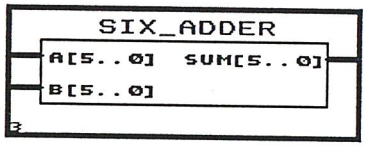
User #3: data = [1 0 1]

	DSSS2 (0)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
MSB	D20(5)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D20(4)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D20(3)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D20(2)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D20(1)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
LSB	D20(0)	=	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 1 1 1 1 1 1 1 1 1 1 1
	DSSS2 (1)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
MSB	D21(5)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
	D21(4)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
	D21(3)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
	D21(2)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
	D21(1)	=	1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
LSB	D21(0)	=	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 1 1 1 1 1 1 1 1 1 1 1
	DSSS2 (2)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
MSB	D22(5)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D22(4)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D22(3)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D22(2)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
	D22(1)	=	0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
LSB	D22(0)	=	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 1 1 1 1 1 1 1 1 1 1 1

โดยค่าที่ได้ออกมานั้นคือ Dxx(5) คือ line ของ data ที่เป็น MSB ซึ่งได้จากการ map ค่าจนถึงค่า Dxx(0) คือ line ของ data ที่เป็น LSB ซึ่งได้จากการ map ค่า

3.2.1.4.3 วงจรทำการบวกค่า

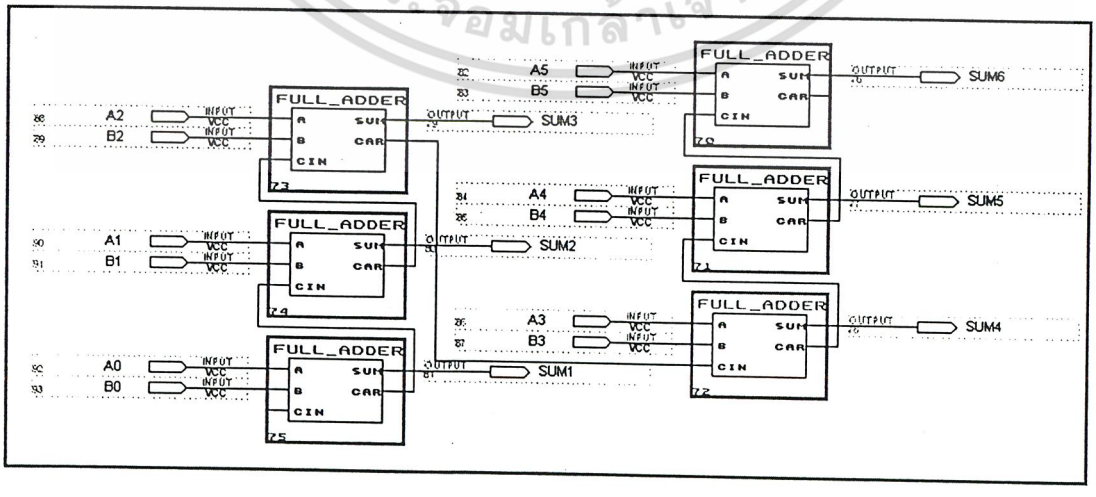
เมื่อผ่านการ Block function mapping แล้วเร็วจะนำสัญญาณที่ได้จากทั้งสาม user มารวมกันด้วยตัวบวกตามรูปที่ 3.11 โดยการบรรยายโค้ดด้วยภาษา VHDL



รูปที่ 3.12 block วงจรบวกขนาด 6 bit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity Six_adder is
      port(a,b          : in  std_logic_vector(5 downto 0);
           sum          : out std_logic_vector(5 downto 0));
end;
ARCHITECTURE structa OF Six_adder IS
SIGNAL s1,s2,s3,s4,s5,s6,s7 : std_logic;
component full_adder
port(a,b,cin      : in  std_logic;
     sum,car      : out std_logic);
end component;
begin
      s6 <= '0';
      A5 : full_adder  port map(a(5),b(5),s5,sum(5),s7);
      A4 : full_adder  port map(a(4),b(4),s4,sum(4),s5);
      A3 : full_adder  port map(a(3),b(3),s3,sum(3),s4);
      A2 : full_adder  port map(a(2),b(2),s2,sum(2),s3);
      A1 : full_adder  port map(a(1),b(1),s1,sum(1),s2);
      A0 : full_adder  port map(a(0),b(0),s6,sum(0),s1);
end;
```

ซึ่งภายในจะประกอบไปด้วย Full Adder จำนวน 6 ตัวตามวงจรดังต่อไปนี้



รูปที่ 3.13 ภายใน block วงจรบวกขนาด 6 bit

ตัวอย่างที่ 3: ดังเช่นในตัวอย่างจากการเข้า mapping ในตัวอย่างที่ 2 เราจะได้ว่า

เราจะนำ DS-CDMA = DSSS0+DSSS1+DSSS2

DS-CDMA(0) = DSSS0(0)+DSSS1(0)+DSSS2(0):

MSB DS-CDMA0(5) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
DS-CDMA0(4) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
DS-CDMA0(3) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
DS-CDMA0(2) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
DS-CDMA0(1) = 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1
LSB DS-CDMA0(0) = 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 1 1 1 1 1 1 1

DS-CDMA(1) = DSSS0(1)+DSSS1(1)+DSSS2(1):

MSB DS-CDMA1(5) = 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
DS-CDMA1(4) = 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
DS-CDMA1(3) = 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
DS-CDMA1(2) = 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
DS-CDMA1(1) = 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0
LSB DS-CDMA1(0) = 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 1 1 1 1 1 1 1

DS-CDMA(2) = DSSS0(2)+DSSS1(2)+DSSS2(2):

MSB DS-CDMA2(5) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
DS-CDMA2(4) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
DS-CDMA2(3) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
DS-CDMA2(2) = 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
DS-CDMA2(1) = 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 1
LSB DS-CDMA2(0) = 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 1 1 1 1 1 1 1

จะได้สัญญาณรวมของสัญญาณ DSSS หลาย user ซึ่งจะเป็นการส่งแบบ code division multiple access (CDMA) นั่นเอง

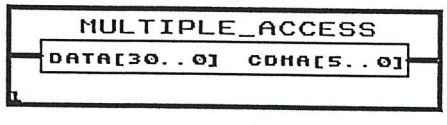
3.2.1.4.4 วงจรรวม Multiple Access

ในการส่งสัญญาณข้อมูลออกไปในช่องสัญญาณนั้น เราจะต้องทำการรวมข้อมูลในทุกช่องสัญญาณโดยการนำข้อมูลที่ได้จากการ Spreading ในข้อที่ 3.2.1.4 มาทำการผ่านวงจรในข้อ 3.2.1.5 และ 3.2.1.6 โดยการนำมารวมกันโดยใช้โค้ด vhdl ในการบรรยาย

```
Library ieee;
use     ieee.std_logic_1164.all;

entity multiple_access is
port(data : in  std_logic_vector(30 downto 0);
     CDMA : out std_logic_vector(5 downto 0));
end;

architecture rtl of multiple_access is
component Twocomplement
port(d    : in    std_logic;
     y    : out   std_logic_vector(5 downto 0));
end component;

component Six_adder is
port(a,b  : in  std_logic_vector(5 downto 0);
     sum  : out std_logic_vector(5 downto 0));
end component;

signal s1,s2,s3,s4,s5       : std_logic_vector(5 downto 0);
signal s6,s7,s8,s9,s10      : std_logic_vector(5 downto 0);
signal s11,s12,s13,s14,s15  : std_logic_vector(5 downto 0);
signal s16,s17,s18,s19,s20  : std_logic_vector(5 downto 0);
signal s21,s22,s23,s24,s25  : std_logic_vector(5 downto 0);
signal s26,s27,s28,s29,s30  : std_logic_vector(5 downto 0);
signal s31                  : std_logic_vector(5 downto 0);
signal k1,k2,k3,k4,k5       : std_logic_vector(5 downto 0);
signal k6,k7,k8,k9,k10      : std_logic_vector(5 downto 0);
signal k11,k12,k13,k14,k15  : std_logic_vector(5 downto 0);
signal k16                  : std_logic_vector(5 downto 0);
signal l1,l2,l3,l4,l5       : std_logic_vector(5 downto 0);
signal l6,l7,l8             : std_logic_vector(5 downto 0);
signal m1,m2,m3,m4          : std_logic_vector(5 downto 0);
signal n1,n2                : std_logic_vector(5 downto 0);
begin
      a1    : Six_adder       port map(n1,n2,CDMA);
      b1    : Six_adder       port map(m1,m2,n1);
      b2    : Six_adder       port map(m3,m4,n2);
      c1    : Six_adder       port map(l1,l2,m1);
      c2    : Six_adder       port map(l3,l4,m2);
      c3    : Six_adder       port map(l5,l6,m3);
      c4    : Six_adder       port map(l7,l8,m4);
      d1    : Six_adder       port map(k1,k2,l1);
      d2    : Six_adder       port map(k3,k4,l2);
      d3    : Six_adder       port map(k5,k6,l3);
      d4    : Six_adder       port map(k7,k8,l4);
      d5    : Six_adder       port map(k9,k10,l5);
      d6    : Six_adder       port map(k11,k12,l6);
      d7    : Six_adder       port map(k13,k14,l7);
      d8    : Six_adder       port map(k15,s31,l8);
      e1    : Six_adder       port map(s1,s2,k1);
      e2    : Six_adder       port map(s3,s4,k2);
```

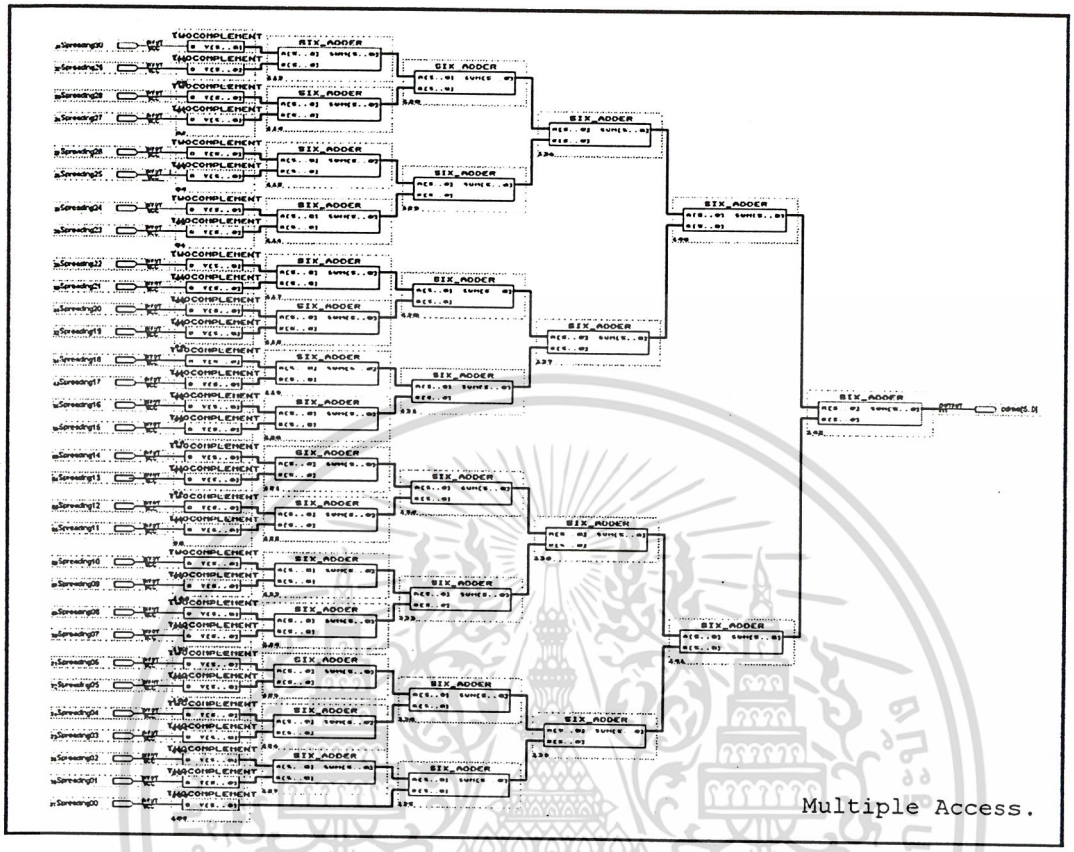
```
        e3    : Six_adder        port map(s5,s6,k3);
        e4    : Six_adder        port map(s7,s8,k4);
        e5    : Six_adder        port map(s9,s10,k5);
        e6    : Six_adder        port map(s11,s12,k6);
        e7    : Six_adder        port map(s13,s14,k7);
        e8    : Six_adder        port map(s15,s16,k8);
        e9    : Six_adder        port map(s17,s18,k9);
        e10   : Six_adder        port map(s19,s20,k10);
        e11   : Six_adder        port map(s2,s22,k11);
        e12   : Six_adder        port map(s23,s24,k12);
        e13   : Six_adder        port map(s25,s26,k13);
        e14   : Six_adder        port map(s27,s28,k14);
        e15   : Six_adder        port map(s29,s30,k15);
        f1    : Twocomplement    port map(data(0),s1);
        f2    : Twocomplement    port map(data(1),s2);
        f3    : Twocomplement    port map(data(2),s3);
        f4    : Twocomplement    port map(data(3),s4);
        f5    : Twocomplement    port map(data(4),s5);
        f6    : Twocomplement    port map(data(5),s6);
        f7    : Twocomplement    port map(data(6),s7);
        f8    : Twocomplement    port map(data(7),s8);
        f9    : Twocomplement    port map(data(8),s9);
        f10   : Twocomplement    port map(data(9),s10);
        f11   : Twocomplement    port map(data(10),s11);
        f12   : Twocomplement    port map(data(11),s12);
        f13   : Twocomplement    port map(data(12),s13);
        f14   : Twocomplement    port map(data(13),s14);
        f15   : Twocomplement    port map(data(14),s15);
        f16   : Twocomplement    port map(data(15),s16);
        f17   : Twocomplement    port map(data(16),s17);
        f18   : Twocomplement    port map(data(17),s18);
        f19   : Twocomplement    port map(data(18),s19);
        f20   : Twocomplement    port map(data(19),s20);
        f21   : Twocomplement    port map(data(20),s21);
        f22   : Twocomplement    port map(data(21),s22);
        f23   : Twocomplement    port map(data(22),s23);
        f24   : Twocomplement    port map(data(23),s24);
        f25   : Twocomplement    port map(data(24),s25);
        f26   : Twocomplement    port map(data(25),s26);
        f27   : Twocomplement    port map(data(26),s27);
        f28   : Twocomplement    port map(data(27),s28);
        f29   : Twocomplement    port map(data(28),s29);
        f30   : Twocomplement    port map(data(29),s30);
        f31   : Twocomplement    port map(data(30),s31);
end;
```

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.14 block การทำ multiple access

ซึ่งภายในจะประกอบไปด้วย Two Compliment จำนวน 31 Six_Adder 30 ตัวตามวงจรดังต่อไปนี้



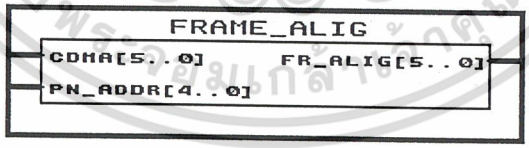
รูปที่ 3.15 ภายใน block วงจร Multiple Access

3.2.1.5 วงจรแทรก FAW

วงจรแทรก FAW จะทำงานเป็นจังหวะโดยสัมพันธ์กับสัญญาณ PN_Addr จากตัว PN_Generator โดยมีหลักการคือจะทำการทรอดแทรก FAW เฉพาะค่า PN_Addr ที่เป็น "00000" เท่านั้นโดยที่จะให้ค่าของ FAW เท่ากับ "011111" ซึ่งจะนำมาบรรยายการทำงานโดยภาษา VHDL ได้ดังต่อไปนี้

```
Library ieee;
use     ieee.std_logic_1164.all;
use     ieee.std_logic_unsigned.all;

entity frame_alig is
port(CDMA    : in  std_logic_vector(5 downto 0);
     PN_Addr : in  std_logic_vector(4 downto 0);
     FR_Alig : out std_logic_vector(5 downto 0));
end;

------------ Frame Aligning ----------------

ARCHITECTURE construction OF frame_alig IS
BEGIN
FrameAligning: Process(CDMA,PN_Addr)
Begin
case PN_Addr IS
     When "00000" =>
          FR_Alig <= "011111";
     When Others  =>
          FR_Alig <= CDMA;
     end case;
End process FrameAligning;
END construction;
```

จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรม MAX+PlusII จะได้ออกมาเป็นอุปกรณ์



รูปที่ 3.16 block การสอดแทรก Frame Aligment Word

ตัวอย่างที่ 4: จากวงจรที่ทำการสอดแทรก FAW จะได้ว่า

วงจรจะทำหน้าที่เปลี่ยน Fake-FAW ให้ไปอยู่ลักษณะที่เป็น FAW จริงซึ่งมีค่าเป็น

FR-CDMA(0) = FAW & DS-CDMA(0)

MSB	FR-CDMA0(5) =	0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	FR-CDMA0(4) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	FR-CDMA0(3) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	FR-CDMA0(2) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	FR-CDMA0(1) =	1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1
LSB	FR-CDMA0(0) =	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 1 1 1 1 1 1 1

FR-CDMA(1) = FAW & DS-CDMA(1)

MSB	FR-CDMA1(5) =	0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
	FR-CDMA1(4) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
	FR-CDMA1(3) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
	FR-CDMA1(2) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0
	FR-CDMA1(1) =	1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0
LSB	FR-CDMA1(0) =	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 1 1 1 1 1 1 1

FR-CDMA(2) = FAW & DS-CDMA(2)

MSB	FR-CDMA2(5) =	0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	FR-CDMA2(4) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	FR-CDMA2(3) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	FR-CDMA2(2) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	FR-CDMA2(1) =	1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1
LSB	FR-CDMA2(0) =	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 1 1 1 1 1 1 1

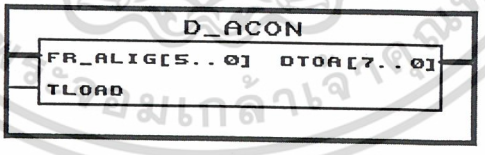
จะพบว่าตัวอย่างข้างต้นเป็นการใส่ FAW ลงไปแทนที่ค่าสัญญาณค่าเดิมที่มีอยู่เพื่อเป็นการบอกจุดเริ่มต้นของชุดข้อมูล

3.2.1.6 วงจรเตรียมค่าสัญญาณเพื่อที่จะทำการส่งไปยัง D/A converter

วงจรเตรียมค่าสัญญาณเพื่อที่จะทำการส่งไปยัง D/A converter นั้นจะมีลักษณะการทำงานคือจะทำการกลับค่า MSB และ ทำการเพิ่มบิตท้าย 2 บิตที่มีค่าเช่นเดียวกับค่า MSB ที่เปลี่ยนไปแล้ว เช่น "011111" จะกลายเป็น "11111111" ซึ่งจะถูกควบคุมจังหวะการทำงานโดยสัญญาณ Tload จากตัว Control Unit โดยจะสามารถบรรยายการทำงานโดยภาษา VHDL ได้คือ

```
library ieee;
use   ieee.std_logic_1164.all;
entity      D_Acon is
port (FR_Alig : in     std_logic_vector(5 downto 0);
      Tload   : in         std_logic;
      DtoA    : buffer std_logic_vector(7 downto 0));
end;

architecture rtl of D_Acon is
signal s1,s2   : std_logic_vector(5 downto 0);
begin
   s1(5)          <= not(FR_Alig(5));
   s1(4 downto 0) <= FR_Alig(4 downto 0);
process(Tload)
begin
  if Tload'event and Tload='1' then
     if s1(5)='1' then
        DtoA  <= s1&"11";
     else
        DtoA  <= s1&"00";
     end if;
  end if;
end process;
end;
```

จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรม MAX+PlusII จะได้ออกมาเป็นอุปกรณ์



รูปที่ 3.17 block วงจรปรับระดับสัญญาณเพื่อที่จะทำการส่งไปยัง D/A converter

ตัวอย่างที่ 5: จากวงจรเตรียมค่าสัญญาณเพื่อที่จะทำการส่งไปยัง D/A converter

วงจรจะทำหน้าที่เปลี่ยนค่าของบิตข้อมูลให้อยู่ในระดับที่เหมาะสมเพื่อที่จะทำเปลี่ยนค่าจาก Digital to Analog ให้เหมาะสมแก่การส่งออก

DA-CDMA(0) = FR-CDMA(0) +2 stuffing bits

MSB	DA-CDMA0(7) =	1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0
	DA-CDMA0(6) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	DA-CDMA0(5) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	DA-CDMA0(4) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1
	DA-CDMA0(3) =	1 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1
	DA-CDMA0(2) =	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 1 1 1 1 1 1 1 1
	DA-CDMA0(1) =	1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0
LSB	DA-CDMA0(0) =	1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0

DA-CDMA(1) = FR-CDMA(1) +2 stuffing bits

MSB	DA-CDMA1(7) =	1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1
	DA-CDMA1(6) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0
	DA-CDMA1(5) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0
	DA-CDMA1(4) =	1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0
	DA-CDMA1(3) =	1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0
	DA-CDMA1(2) =	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 1 1 1 1 1 1 1 1
	DA-CDMA1(1) =	1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1
LSB	DA-CDMA1(0) =	1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1

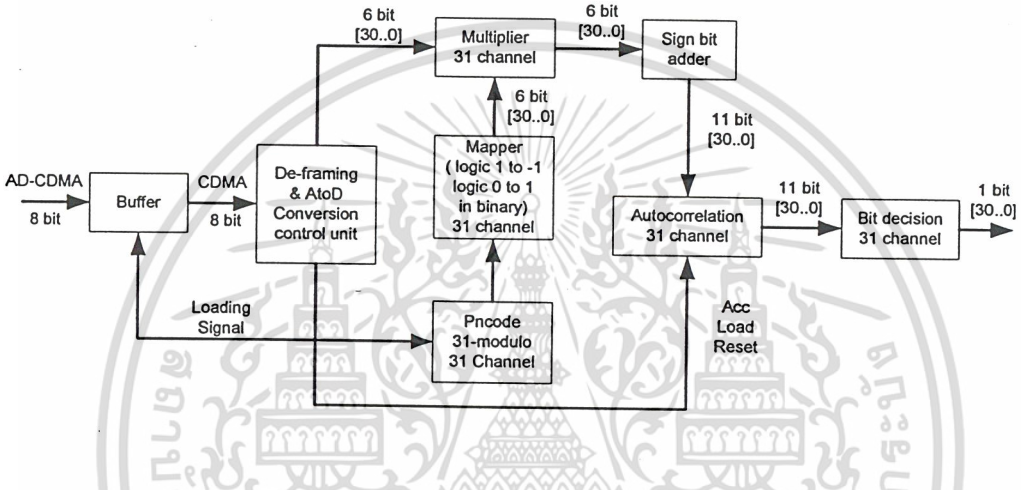
DA-CDMA(2) = FR-CDMA(2) +2 stuffing bits

MSB	DA-CDMA2(7) =	1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0
	DA-CDMA2(6) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	DA-CDMA2(5) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	DA-CDMA2(4) =	1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1
	DA-CDMA2(3) =	1 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1
	DA-CDMA2(2) =	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 1 1 1 1 1 1 1 1
	DA-CDMA2(1) =	1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0
LSB	DA-CDMA2(0) =	1 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0

จะพบว่าตัวอย่างข้างต้นเป็นการกลับบิตแรกของค่าทุกค่าและทำการแทรกบิต 2 บิตสุดท้ายที่มีค่าเช่นเดียวกับบิตแรกที่ถูกทำการเปลี่ยนแปลงแล้ว

3.2.2 การออกแบบวงจรภาครับ

การออกแบบวงจรในด้านรับนั้น ขั้นแรกเราจะต้องทำการถอด Frame Alignment Word ออกเสียก่อนจากนั้นทำการ เปลี่ยนค่าบิตแรกกลับและทำการตัดค่าสองบิตท้ายที่ถูกเพิ่มเข้ามาออก จากนั้นจะเริ่มเข้าสู่ขบวนการถอด PN code โดยการคูณดังนั้นเราจะต้องทำการแปลงชุดของค่า PN ให้เป็นค่า voltage เสมือนที่ถูกแสดงอยู่รูปของเลขฐานสองจำนวน 6 บิตแบบ 2 compliment เสียก่อนจากนั้นนำไปคูณกับสัญญาณรวมที่ถูกส่งมาจากทางด้านส่งแล้วทำการ บวกสะสมค่า (accumulate) แล้วส่งต่อไปยังวงจรตัดสินค่าทางโลจิก



รูปที่ 3.18 block diagram ทางด้านรับ

โดยจะมีสัญญาณมาควบคุมการทำงานอยู่ 4 ตัวคือ วงจรควบคุมการ sampling จาก A/D converter และ Buffer, สัญญาณนาฬิกาที่ทำการควบคุมการสร้างของสัญญาณ PN, สัญญาณนาฬิกาที่ทำการควบคุมการทำงานของวงจรควบคุม Control Unit, สัญญาณควบคุมการเลือกของ PN code ที่จะนำมาใช้โดยจะใช้ความถี่ต่างๆกัน

- สัญญาณนาฬิกาที่ทำการควบคุมการสร้างของสัญญาณ PN จะต้องมีความเร็วเท่าทางด้านส่งคือ 2.048 Mcps
- สัญญาณนาฬิกาที่ทำการควบคุมการทำงานของวงจรควบคุม Control Unit จะต้องทำงานอย่างน้อย 4 จังหวะคือเว้นค่าสำหรับ Delay, Acc, Load, Reset
- สัญญาณควบคุมการเลือกของ PN code จะใช้ความเร็วเท่ากับค่าของ chip rate เพื่อการ synchronization ในการสร้างสัญญาณ

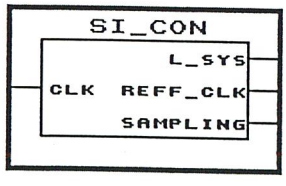
โดยรายละเอียดของส่วนต่างๆจะมีดังต่อไปนี้

3.2.2.1 วงจรควบคุมการ Sampling และ Load ค่าเข้า Buffer

3.2.2.1.1 วงจรควบคุมการ Sampling และ Load ค่านั้นจะเป็นตัวส่งสัญญาณไปทำการ Sampling ค่าจาก AtoD ออกมาและ ตัวสัญญาณที่ใช้ในการ Load ข้อมูลเข้าใน Buffer นั้นเพื่อต้องการค่าที่คงที่มาใช้ในการประมวณผลต่อไป โดยจะสามารถบรรยายการทำงานโดยภาษา VHDL ได้ดังต่อไปนี้

```
library ieee;
use   ieee.std_logic_1164.all;

entity SI_con is
port (Clk             : in  std_logic;
      l_sys,Reff_clk  : out std_logic;
      sampling        : out std_logic);
end;
architecture rtl of SI_con is
    Signal count:  integer range 0 to 9;
begin
process(Clk)
begin
      Reff_clk <= Clk;
 if Clk'event and Clk='1'  then
      if  count /= 9 then
          count <= count+1;
      else
          count <= 0;
      end if;
 end if;
 case count is
      when 1 =>
         l_sys    <= '0';
         sampling <= '1';
      when 2 =>
         l_sys    <= '1';
         sampling <= '0';
      when others =>
         l_sys    <= '0';
         sampling <= '0';
  end case;
end process;
end;
```

จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรม MAX+PlusII จะได้ออกมาเป็นอุปกรณ์

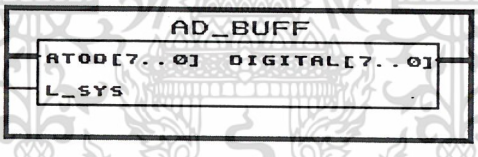


รูปที่ 3.19 ชุดควบคุมสัญญาณการ Sampling, Load ค่า

3.2.2.1.2 วงจร buffer จะทำหน้าที่เป็นตัวเก็บค่าสัญญาณและจะคงค่าสัญญาณไว้จนกว่าจะมีสัญญาณ สั่งให้ Load ค่าใหม่เข้าไป โดยจะสามารถบรรยายการทำงานโดยภาษา VHDL ได้ดังต่อไปนี้

```
library ieee;
use   ieee.std_logic_1164.all;

entity AD_Buff is
port (AtoD    : in     std_logic_vector(7 downto 0);
      l_sys   : in     std_logic;
      Digital : buffer std_logic_vector(7 downto 0));
end;

architecture rtl of AD_Buff is
begin
process(AtoD, l_sys)
   begin
     if l_sys'event and l_sys='1'  then
          Digital <= AtoD;
     end if;
   end process;
end;
```

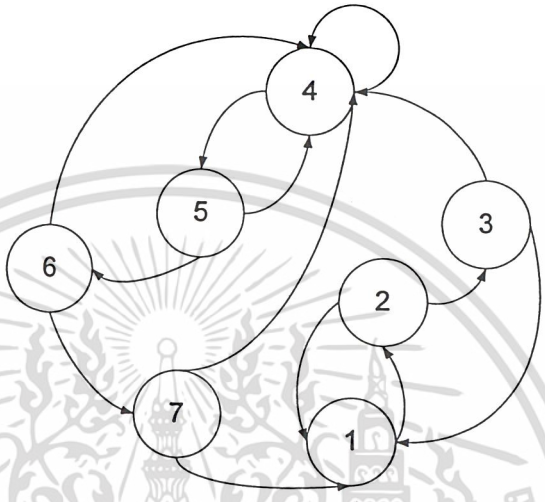
จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรม MAX+PlusII จะได้ออกมาเป็นอุปกรณ์



รูปที่ 3.20 ชุด Buffer ควบคุมโดยสัญญาณ L_sys

3.2.2.2 ตรวจสอบและถอด FAW ซึ่งทำการแปลงสัญญาณกลับเพื่อการคำนวณ และ วงจรชุดควบคุมรวม

3.2.2.2.1 วงจรตรวจสอบและถอดค่า FAW จะทำงานแบ่งออกเป็น 2 โหมด คือ Search mode และ lock mode ซึ่งจะแบ่งเป็น stage ได้ตามไดเอแกมดังต่อไปนี้



สถานะ 4	เป็นสถานะอไลเมนต์เฟรมสมบูรณ์ระบบอยู่ในสภาพล็อก (Synchronize)
สถานะ 5, 6, 7	เป็นสถานะอไลเมนต์เฟรมชั่วคราวระบบจะอยู่ในโหมดการตรวจสอบ (Check Mode)
สถานะ 1	เป็นสถานะของการไม่อไลเมนต์เฟรม ระบบจะอยู่ในโหมดการค้นหา FAW (Search Mode)
สถานะ 2,3	เป็นสถานะรอระบบจะอยู่ในโหมดการค้นหาและการตรวจ-สอบ (Search/Check Mode) FAW

โดยที่ในสภาวะ 1,2,3 เราจะไม่ให้มีสัญญาณออกมาเป็นการตรวจสอบ FAW ซึ่งเมื่อเข้า สถานะที่ 4 จะให้สัญญาณ output คือสัญญาณที่ถูกถอด FAW ออกแล้วและถูกทำการปรับระดับกลับ (คือ กลับค่า MSB และตัดสองบิตสุดท้ายที่เพิ่มเข้ามาออก

3.2.2.2.2 ชุดควบคุมรวม

จะทำหน้าที่คือเมื่อสถานะการทำงานของวงจรตรวจจับค่าอยู่ใน สถานะที่ 4 จะให้สัญญาณควบคุมออกมา 5 ตัวซึ่งประกอบไปด้วย

1. Start : สั่งให้วงจร กำเนิดสัญญาณ PN เริ่มทำงาน
2. PN_Clk : เป็นสัญญาณนาฬิกาของสัญญาณ PN
3. L_Acc : เป็นตัวควบคุมการบวกสะสมค่า
4. L_Buff : เมื่อบวกได้ครบตามความยาวที่กำหนดให้ส่งออก
5. Reset : ทำการเคลียร์ค่าก่อนจะรับข้อมูลชุดใหม่

ซึ่งเราสามารถบรรยายการทำงานด้วยภาษา VHDL ของหัวข้อที่ 3.2.2.2.1 และ 3.2.2.2.2 ได้ดังต่อไปนี้คือ

```
library     ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity gcontalter is
port(Digital  : in  std_logic_vector(7 downto 0);
     Reff_clk : in  std_logic;
     Data     : buffer std_logic_vector(5 downto 0);
     PN_Clk   : out std_logic;
     Start    : out std_logic;
     l_acc    : out std_logic;
     reset    : out std_logic;
     l_buff   : out std_logic;
     GC_S     : out std_logic_vector(2 downto 0));
end;

architecture rtl of gcontalter is
Signal state:  integer range 0 to 8;
Signal count:  integer range 0 to 320;
Signal count1: integer range 0 to 319;

Begin
Process(Digital,Reff_clk)
Begin
      If (Reff_clk'event and Reff_clk='1') then
      Case state Is
      when 0  =>  GC_S  <= "000";
            case Digital Is
            when "11110100"|"11110101"|"11110110"|
                 "11110111"|"11111000"|"11111001"|
                 "11111010"|"11111110"|"11111111" =>
                  count <= 0;
                  state <= 2;
            when others=>
                  count  <= 0;
                  l_acc  <= '0';
                  reset  <= '0';
                  l_buff <= '0';
            end case;
```

```
when 1 => GC_S <= "001";
      case Digital Is
      when "11110100"|"11110101"|"11110110"|
           "11110111"|"11111000"|"11111001"|
           "11111010"|"11111110"|"11111111" =>
            count <= 0;
            state <= 2;
      when others =>
            count  <= 0;
            l_acc  <= '0';
            reset  <= '0';
            l_buff <= '0';
      end case;

when 2 => GC_S <= "010";
      if  count /= 320 then
          count <= count+1;
      else
          count <= 1;
      end if;

      case count is
      when 320 =>
            case Digital Is
            when "11110100"|"11110101"|"11110110"|
                 "11110111"|"11111000"|"11111001"|
                 "11111010"|"11111110"|"11111111"=>
                  count <= 1;
                  state <= 3;
            when others =>
                  state <= 1;
            end case;

      when others =>
      end case;

when 3 => GC_S <= "011";
      if  count /= 320 then
          count <= count+1;
      else
          count <= 1;
      end if;

      case count is
      when 320 =>
            case Digital Is
            when "11110100"|"11110101"|"11110110"|
                 "11110111"|"11111000"|"11111001"|
                 "11111010"|"11111110"|"11111111"=>
                  count <= 1;
                  state <= 4;
            when others =>
                  state <= 1;
            end case;
            when others =>
            end case;
```

```
when 4 => GC_S  <= "100";
      if   count /= 320 then
           count <= count+1;
      else count <=1;
      end if;

   case count is
   when 1 =>
      PN_Clk <= '1';
      Data   <= "000000";

   when 11|21|31|41|51|61|71|81|91|101 =>
      Data(5) <= not(Digital(7));
      Data(4 downto 0) <= Digital(6 downto 2);
      PN_Clk  <= '1';

   when 111|121|131|141|151|161|171|181|191|201 =>
      Data(5)          <= not(Digital(7));
      Data(4 downto 0) <= Digital(6 downto 2);
      PN_Clk           <= '1';

   when 211|221|231|241|251|
        261|271|281|291|301|311 =>
      Data(5)          <= not(Digital(7));
      Data(4 downto 0) <= Digital(6 downto 2);
      PN_Clk           <= '1';

   when 6|16|26|36|46|56|66|76|86|96 =>
      l_acc  <= '1';
      l_buff <= '0';
      reset  <= '0';

   when 106|116|126|136|146|156|166|176|186|196 =>
      l_acc  <= '1';
      l_buff <= '0';
      reset  <= '0';

   when 206|216|226|236|246|256|266|276|
        286|296|306|316 =>
      l_acc  <= '1';
      l_buff <= '0';
      reset  <= '0';

   when 317 =>
      l_acc  <= '0';
      l_buff <= '1';
      reset  <= '0';
      start  <= '1';

   when 318 =>
      l_acc  <= '0';
      l_buff <= '0';
      reset  <= '1';
      start  <= '0';
```

```
when 320 =>
case Digital Is
   when "11110100"|"11110101"|"11110110"|
        "11110111"|"11111000"|"11111001"|
        "11111010"|"11111110"|"11111111" =>
         count <= 1;
         state <= 4;

   when others =>
         count <= 1;
         state <= 5;
   end case;

   when others =>
         l_acc  <= '0';
         l_buff <= '0';
         reset  <= '0';
         start  <= '0';
         PN_Clk <= '0';
         end case;

when 5 => GC_S <= "101";
    if   count /= 320 then
         count <= count+1;
    else count <=1;
    end if;

    case count is
    when 1 =>
         PN_Clk  <= '1';
         Data    <= "000000";

    when 11|21|31|41|51|61|71|81|91|101 =>
         Data(5)          <= not(Digital(7));
         Data(4 downto 0)<= Digital(6 downto 2);
         PN_Clk           <= '1';

    when 111|121|131|141|151|
         161|171|181|191|201 =>
         Data(5)          <= not(Digital(7));
         Data(4 downto 0)<= Digital(6 downto 2);
         PN_Clk           <= '1';

   when 211|221|231|241|251|261|
        271|281|291|301|311 =>
         Data(5)          <= not(Digital(7));
         Data(4 downto 0)<= Digital(6 downto 2);
         PN_Clk           <= '1';

   when 6|16|26|36|46|56|66|76|86|96 =>
         l_acc  <= '1';
         l_buff <= '0';
         reset  <= '0';

   when 106|116|126|136|146|
        156|166|176|186|196 =>
         l_acc  <= '1';
         l_buff <= '0';
         reset  <= '0';
```

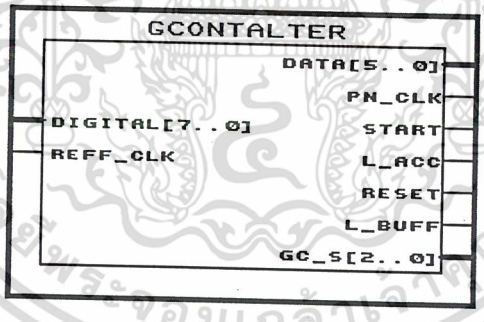
```
            when  206|216|226|236|246|256|
                  266|276|286|296|306|316 =>
                  l_acc  <= '1';
                  l_buff <= '0';
                  reset  <= '0';

            when 317 =>
                  l_acc  <= '0';
                  l_buff <= '1';
                  reset  <= '0';
                  start  <= '1';

            when 318 =>
                  l_acc  <= '0';
                  l_buff <= '0';
                  reset  <= '1';
                  start  <= '0';

            when 320 =>
            case Digital Is
                  when "11110100"|"11110101"|"11110110"|
                       "11110111"|"11111000"|"11111001"|
                       "11111010"|"11111110"|"11111111"=>
                        count <= 1;
                        state <= 4;
                  when others =>
                        count <= 1;
                        state <= 6;
            end case;

            when others =>
                  l_acc  <= '0';
                  l_buff <= '0';
                  reset  <= '0';
                  start  <= '0';
                  PN_Clk <= '0';
      end case;

      when 6 => GC_S  <= "110";
         if   count /= 320 then
              count <= count+1;
         else count <=1;
         end if;

         case count is
            when 1 =>
               PN_Clk   <= '1';
               Data     <= "000000";

            when 11|21|31|41|51|61|71|81|91|101 =>
               Data(5)          <= not(Digital(7));
               Data(4 downto 0) <= Digital(6 downto 2);
               PN_Clk           <= '1';

            when 111|121|131|141|151|
                 161|171|181|191|201 =>
               Data(5)          <= not(Digital(7));
               Data(4 downto 0) <= Digital(6 downto 2);
               PN_Clk           <= '1';
```

```
when 211|221|231|241|251|261|
     271|281|291|301|311 =>
   Data(5)          <= not(Digital(7));
   Data(4 downto 0) <= Digital(6 downto 2);
   PN_Clk           <= '1';

when 6|16|26|36|46|56|66|76|86|96 =>
      l_acc   <= '1';
      l_buff  <= '0';
      reset   <= '0';

when 106|116|126|136|146|
     156|166|176|186|196 =>
      l_acc  <= '1';
      l_buff <= '0';
      reset  <= '0';

when 206|216|226|236|246|256|
     266|276|286|296|306|316 =>
      l_acc  <= '1';
      l_buff <= '0';
      reset  <= '0';

when 317 =>
      l_acc  <= '0';
      l_buff <= '1';
      reset  <= '0';
      start  <= '1';

when 318 =>
      l_acc  <= '0';
      l_buff <= '0';
      reset  <= '1';
      start  <= '0';

when 320 =>
case Digital Is
      when "11110100"|"11110101"|"11110110"|
           "11110111"|"11111000"|"11111001"|
           "11111010"|"11111110"|"11111111"=>
            count <= 1;
            state <= 4;

      when others =>
            count <= 1;
            state <= 7;
end case;

when others =>
      l_acc  <= '0';
      l_buff <= '0';
      reset  <= '0';
      start  <= '0';
      PN_Clk <= '0';
end case;
```

```
when 7 => GC_S  <= "111";
   if   count /= 320 then
        count <= count+1;
   else count <=1;
   end if;

   case count is
      when 1 =>
         PN_Clk   <= '1';
         Data     <= "000000";

      when 11|21|31|41|51|61|71|81|91|101 =>
         Data(5)          <= not(Digital(7));
         Data(4 downto 0) <= Digital(6 downto 2);
         PN_Clk           <= '1';

      when 111|121|131|141|151|
           161|171|181|191|201 =>
         Data(5)          <= not(Digital(7));
         Data(4 downto 0) <= Digital(6 downto 2);
         PN_Clk           <= '1';

      when 211|221|231|241|251|261|
           271|281|291|301|311 =>
           Data(5)          <= not(Digital(7));
           Data(4 downto 0)<= Digital(6 downto 2);
           PN_Clk           <= '1';

      when 6|16|26|36|46|56|66|76|86|96 =>
           l_acc  <= '1';
           l_buff <= '0';
           reset  <= '0';

      when 106|116|126|136|146|
           156|166|176|186|196 =>
           l_acc  <= '1';
           l_buff <= '0';
           reset  <= '0';

      when 206|216|226|236|246|256|
           266|276|286|296|306|316 =>
           l_acc  <= '1';
           l_buff <= '0';
           reset  <= '0';

      when 317 =>
           l_acc  <= '0';
           l_buff <= '1';
           reset  <= '0';
           start  <= '1';

      when 318 =>
           l_acc  <= '0';
           l_buff <= '0';
           reset  <= '1';
           start  <= '0';
```

```
when 320 =>
case Digital Is
      when "11110100"|"11110101"|"11110110"|
           "11110111"|"11111000"|"11111001"|
           "11111010"|"11111110"|"11111111"=>
            count <= 1;
            state <= 4;

      when others =>
            count <= 1;
            state <= 1;
            Data  <= "000000";
end case;

when others =>
      l_acc  <= '0';
      l_buff <= '0';
      reset  <= '0';
      start  <= '0';
      PN_Clk <= '0';
end case;

When others=>
end case;
end if;
end process;
end;
```

จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรม MAX+PlusII จะได้ออกมาเป็นอุปกรณ์



รูปที่ 3.21 วงจร GCON ใช้ check FAW กับควบคุมสัญญาณ

ตัวอย่างที่ 6: จากวงจรที่ทำการตรวจจับและถอดค่า FAW

วงจรจะทำหน้าที่ตรวจจับและถอดค่า FAW ออก

DF-CDMA(0) = DA-CDMA(0) - 2 stuffing bits

MSB DF-CDMA0(5) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1

DF-CDMA0(4) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1

DF-CDMA0(3) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1

DF-CDMA0(2) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0 1

DF-CDMA0(1) = 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1

LSB DF-CDMA0(0) = 0 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 1 1 1 1 1 1

DF-CDMA(1) = DA-CDMA(1) - 2 stuffing bits

MSB DF-CDMA1(5) = 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0

DF-CDMA1(4) = 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0

DF-CDMA1(3) = 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0

DF-CDMA1(2) = 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 0

DF-CDMA1(1) = 0 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0

LSB DF-CDMA1(0) = 0 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 1 1 1 1 1 1

DF-CDMA(2) = DA-CDMA(2) - 2 stuffing bits

MSB DF-CDMA2(5) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1

DF-CDMA2(4) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1

DF-CDMA2(3) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1

DF-CDMA2(2) = 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1

DF-CDMA2(1) = 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1

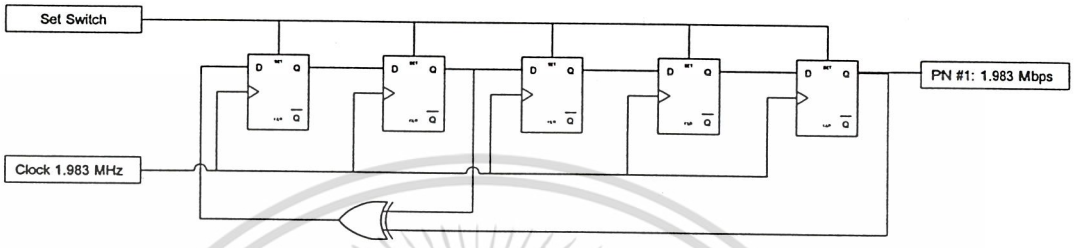
LSB DF-CDMA2(0) = 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

จะพบว่าตัวอย่างข้างต้นเป็นการถอด FAW ออกจากค่าสัญญาณค่าเดิมที่มีอยู่เพื่อเป็นการบอกจุดเริ่มต้นของชุดข้อมูล

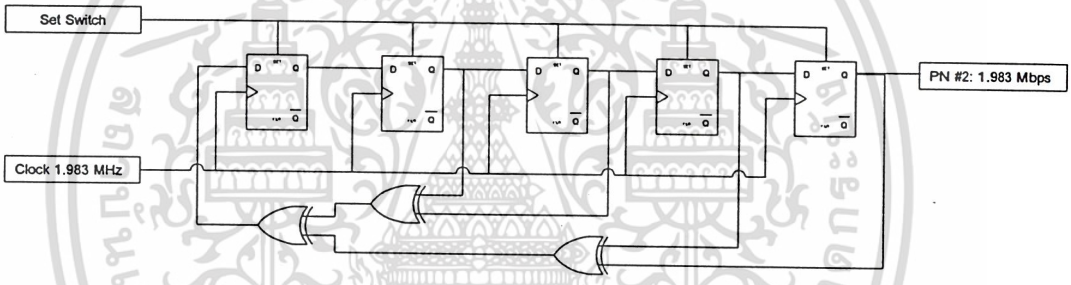
3.2.2.3 วงจรทำการกู้สัญญาณกลับ

3.2.2.3.1 วงจรกำเนิดสัญญาณ PN

วงจรสร้างสัญญาณแบบ random ของสัญญาณ PN code โดยใช้ LFSR โดยใช้ทั้งหมด 5 state โดยทำการ feed back จาก state 5 กับ 2 และ state 5 4 3 และ 2 ตามลำดับดังเช่น ในภาพที่ 3.4 และ 3.5 ตามลำดับ

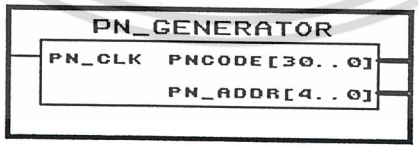


รูปที่ 3.22 วงจรกำเนิดค่า PNcode แบบ m-sequence 31-modulo(1) ความเร็ว 1.983 Mbps



รูปที่ 3.23 วงจรกำเนิดค่า PNcode แบบ m-sequence 31-modulo(2) ความเร็ว 1.983 Mbps

ซึ่งจะมีรูปแบบการทำงานเช่นเดียวกับ PNGenerator ทางด้านส่ง ซึ่งทำการสังเคราะห์ออกมาเป็นตัวอุปกรณ์ ในโปรแกรม MAX+plus II ได้ดังภาพ



รูปที่ 3.24 PN_Generator ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

3.2.2.3.2 การ map ค่า

จะต้องมีหลักการอย่างเดียวกับค่าในด้านส่งในข้อที่ 3.2.1.4.2

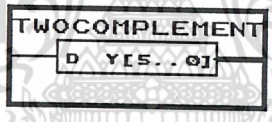
Logic 0 -> 0 0 0 0 0 1

Logic 1 -> 1 1 1 1 1 1

จะทำการออกแบบโดยการบรรยายเป็น vhdl code ตามข้อความด้านล่าง

```
LIBRARY     ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Twocomplement IS
PORT(
      d      : in Std_Logic;
      Y      : out Std_Logic_vector(5 downto 0));
END Mapping;

ARCHITECTURE Mapper OF Mapping IS
BEGIN
            Y <=  "111111" WHEN ('d=1') ELSE
                  "000001" WHEN ('d=0') ELSE
                  "000000";
END Mapper;
```

จากนั้นทำการ compile ในโปรแกรม MAX+plus จะได้ component ออกมา



รูปที่ 3.25 block การ map ค่าที่ได้จากการทำการ synthesis จากโปรแกรม MAX+plus II

ตัวอย่างที่ 7: นำสัญญาณ PN ที่ใช้ในตัวอย่างที่ 1 มาทำการ map ค่า ซึ่งได้แก่ให้เป็นค่า voltage เสมือนแสดงอยู่ในรูปไบนารี 6 บิต (ตามแนวตั้ง)

PN01= [1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0]

PN13= [1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1]

PN31= [1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1]

PN of User #1:

PN01 = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

MSB P01(5) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

P01(4) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

P01(3) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

P01(2) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

P01(1) = 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0

LSB P01(0) = 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 1 1 1 1 1 1 1

PN of User #2:

PN13 = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

MSB P13(5) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

P13(4) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

P13(3) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

P13(2) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

P13(1) = 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1

LSB P13(0) = 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 1 1 1 1 1 1 1

PN of User #3:

PN13 = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

MSB P31(5) = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

P31(4) = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

P31(3) = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

P31(2) = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

P31(1) = 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1

LSB P31(0) = 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 1 1 1 1 1 1 1

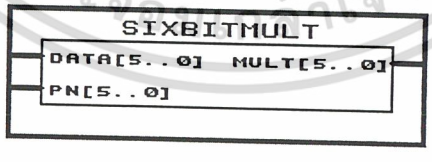
เราจะนำค่า PN ที่ถูกนำไป map ค่าให้เป็น voltage เสมือนที่อยู่ในรูปแบบของเลขฐานสองแบบ 2's compliment (ตามแนวตั้ง) เพื่อจะนำไปใช้ในการกู้สัญญาณออกมาในหัวข้อถัดไป

3.2.2.3.3 วงจรทำการคูณค่า

เมื่อทำการ map ค่าของ PN code เรียบร้อยแล้วเราจะนำสัญญาณที่เป็น multiple access มาที่การคูณเราจะนำสัญญาณ 6 bit มาคูณด้วยสัญญาณ 6 bit แล้วเอาผลลัพธ์เพียง 6 bit สุดท้ายเนื่องจากต้องการแค่การเปลี่ยนแปลงของขนาดซึ่งจะสามารถบรรยายการทำงานด้วยภาษา VHDL ได้ดังต่อไปนี้

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity sixbitmult is
port(data : in  std_logic_vector(5 downto 0);
       pn : in  std_logic_vector(5 downto 0);
     mult : out std_logic_vector(5 downto 0));
end;

ARCHITECTURE structa OF sixbitmult IS
begin
mult_proc:process(data,pn)
    variable sum  : std_logic_vector(11 downto 0);
    variable zero : std_logic_vector( 5 downto 0);
begin
      sum   := (others => '0');
      zero  := (others => '0');
   shift_add_loop : for i in data'range loop
       if pn(i)='1' then -- compute partial product
           sum := sum + (zero & data);
       end if;
       if i/=0 then
           sum := sum(10 downto 0) & '0';
       end if;
   end loop shift_add_loop;
      mult <= sum(5 downto 0);
end process mult_proc;
End;
```

ซึ่งเมื่อทำการสังเคราะห์ด้วยโปรแกรม MAX+PlusII ได้



รูปที่ 3.26 block วงจรคูณที่ได้จากการ Syntersis

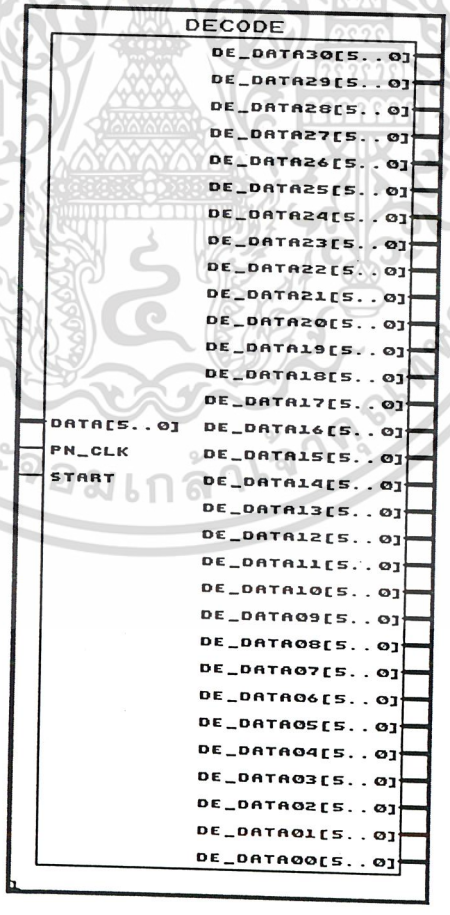
ซึ่งจะสามารถทำการรเขียนเป็นวงจรรวมด้วยการบรรยายการทำงานโดยภาษา VHDL ได้ดังต่อไปนี้

```
library     ieee;
use   ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Decode Is
port( DATA         : in  std_logic_vector(5 downto 0);
      PN_Clk       : in  std_logic;
      Start        : in  std_logic;
      De_Data30    : out std_logic_vector(5 downto 0);
      De_Data29    : out std_logic_vector(5 downto 0);
      De_Data28    : out std_logic_vector(5 downto 0);
      De_Data27    : out std_logic_vector(5 downto 0);
      De_Data26    : out std_logic_vector(5 downto 0);
      De_Data25    : out std_logic_vector(5 downto 0);
      De_Data24    : out std_logic_vector(5 downto 0);
      De_Data23    : out std_logic_vector(5 downto 0);
      De_Data22    : out std_logic_vector(5 downto 0);
      De_Data21    : out std_logic_vector(5 downto 0);
      De_Data20    : out std_logic_vector(5 downto 0);
      De_Data19    : out std_logic_vector(5 downto 0);
      De_Data18    : out std_logic_vector(5 downto 0);
      De_Data17    : out std_logic_vector(5 downto 0);
      De_Data16    : out std_logic_vector(5 downto 0);
      De_Data15    : out std_logic_vector(5 downto 0);
      De_Data14    : out std_logic_vector(5 downto 0);
      De_Data13    : out std_logic_vector(5 downto 0);
      De_Data12    : out std_logic_vector(5 downto 0);
      De_Data11    : out std_logic_vector(5 downto 0);
      De_Data10    : out std_logic_vector(5 downto 0);
      De_Data09    : out std_logic_vector(5 downto 0);
      De_Data08    : out std_logic_vector(5 downto 0);
      De_Data07    : out std_logic_vector(5 downto 0);
      De_Data06    : out std_logic_vector(5 downto 0);
      De_Data05    : out std_logic_vector(5 downto 0);
      De_Data04    : out std_logic_vector(5 downto 0);
      De_Data03    : out std_logic_vector(5 downto 0);
      De_Data02    : out std_logic_vector(5 downto 0);
      De_Data01    : out std_logic_vector(5 downto 0);
      De_Data00    : out std_logic_vector(5 downto 0));
end;

architecture rtl of Decode is
component sixbitmult
port( data   : in  std_logic_vector(5 downto 0);
      pn     : in  std_logic_vector(5 downto 0);
      mult   : out std_logic_vector(5 downto 0));
end component;
```

```
component g2com is
port( Start         : in   std_logic;
      RPN_Clk       : in   std_logic;
      PN30          : out  std_logic_vector(5 downto 0);
      PN29          : out  std_logic_vector(5 downto 0);
      PN28          : out  std_logic_vector(5 downto 0);
      PN27          : out  std_logic_vector(5 downto 0);
      PN26          : out  std_logic_vector(5 downto 0);
      PN25          : out  std_logic_vector(5 downto 0);
      PN24          : out  std_logic_vector(5 downto 0);
      PN23          : out  std_logic_vector(5 downto 0);
      PN22          : out  std_logic_vector(5 downto 0);
      PN21          : out  std_logic_vector(5 downto 0);
      PN20          : out  std_logic_vector(5 downto 0);
      PN19          : out  std_logic_vector(5 downto 0);
      PN18          : out  std_logic_vector(5 downto 0);
      PN17          : out  std_logic_vector(5 downto 0);
      PN16          : out  std_logic_vector(5 downto 0);
      PN15          : out  std_logic_vector(5 downto 0);
      PN14          : out  std_logic_vector(5 downto 0);
      PN13          : out  std_logic_vector(5 downto 0);
      PN12          : out  std_logic_vector(5 downto 0);
      PN11          : out  std_logic_vector(5 downto 0);
      PN10          : out  std_logic_vector(5 downto 0);
      PN09          : out  std_logic_vector(5 downto 0);
      PN08          : out  std_logic_vector(5 downto 0);
      PN07          : out  std_logic_vector(5 downto 0);
      PN06          : out  std_logic_vector(5 downto 0);
      PN05          : out  std_logic_vector(5 downto 0);
      PN04          : out  std_logic_vector(5 downto 0);
      PN03          : out  std_logic_vector(5 downto 0);
      PN02          : out  std_logic_vector(5 downto 0);
      PN01          : out  std_logic_vector(5 downto 0);
      PN00          : out  std_logic_vector(5 downto 0));
end component;

signal add                      : std_logic_vector(4 downto 0);
signal s30,s29,s28,s27,s26: std_logic_vector(5 downto 0);
signal s25,s24,s23,s22,s21: std_logic_vector(5 downto 0);
signal s20,s19,s18,s17,s16: std_logic_vector(5 downto 0);
signal s15,s14,s13,s12,s11: std_logic_vector(5 downto 0);
signal s10,s09,s08,s07,s06: std_logic_vector(5 downto 0);
signal s05,s04,s03,s02,s01: std_logic_vector(5 downto 0);
signal s00                      : std_logic_vector(5 downto 0);

begin
PR    : g2com port map(Start,PN_Clk,s30,s29,s28,s27,s26,
                       s25,s24,s23,s22,s21,s20,s19,s18,
                       s17,s16,s15,s14,s13,s12,s11,s10,
                       s09,s08,s07,s06,s05,s04,s03,s02,
                       s01,s00);

      T30 : sixbitmult  port map(DATA,s30,De_Data30);
      T29 : sixbitmult  port map(DATA,s29,De_Data29);
      T28 : sixbitmult  port map(DATA,s28,De_Data28);
      T27 : sixbitmult  port map(DATA,s27,De_Data27);
      T26 : sixbitmult  port map(DATA,s26,De_Data26);
      T25 : sixbitmult  port map(DATA,s25,De_Data25);
      T24 : sixbitmult  port map(DATA,s24,De_Data24);
      T23 : sixbitmult  port map(DATA,s23,De_Data23);
      T22 : sixbitmult  port map(DATA,s22,De_Data22);
```

```
    T21 : sixbitmult  port map(DATA,s21,De_Data21);
    T20 : sixbitmult  port map(DATA,s20,De_Data20);
    T19 : sixbitmult  port map(DATA,s19,De_Data19);
    T18 : sixbitmult  port map(DATA,s18,De_Data18);
    T17 : sixbitmult  port map(DATA,s17,De_Data17);
    T16 : sixbitmult  port map(DATA,s16,De_Data16);
    T15 : sixbitmult  port map(DATA,s15,De_Data15);
    T14 : sixbitmult  port map(DATA,s14,De_Data14);
    T13 : sixbitmult  port map(DATA,s13,De_Data13);
    T12 : sixbitmult  port map(DATA,s12,De_Data12);
    T11 : sixbitmult  port map(DATA,s11,De_Data11);
    T10 : sixbitmult  port map(DATA,s10,De_Data10);
    T09 : sixbitmult  port map(DATA,s09,De_Data09);
    T08 : sixbitmult  port map(DATA,s08,De_Data08);
    T07 : sixbitmult  port map(DATA,s07,De_Data07);
    T06 : sixbitmult  port map(DATA,s06,De_Data06);
    T05 : sixbitmult  port map(DATA,s05,De_Data05);
    T04 : sixbitmult  port map(DATA,s04,De_Data04);
    T03 : sixbitmult  port map(DATA,s03,De_Data03);
    T02 : sixbitmult  port map(DATA,s02,De_Data02);
    T01 : sixbitmult  port map(DATA,s01,De_Data01);
    T00 : sixbitmult  port map(DATA,s00,De_Data00);
end;
```

ซึ่งเมื่อทำการสังเคราะห์ด้วยโปรแกรม MAX+PlusII ได้



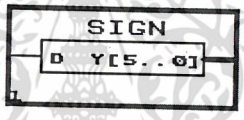
รูปที่ 3.27 วงจร DeCode รวม

3.2.2.3.3.1 วงจรเพิ่มขนาดบิตเครื่องหมาย

เพื่อที่จะได้ค่าเหมาะสมในการเข้าสู่วงจรบวกสะสมค่า สามารถบรรยายการทำงานโดยภาษา VHDL ได้ดังต่อไปนี้

```
LIBRARY     ieee;
USE ieee.std_logic_1164.ALL;
entity Sign is
port(
      d     : in  std_logic;
      y     : out std_logic_vector(5 downto 0));
end Sign;

architecture Mapper of Sign is
begin
y <=  "000000" when (d='0') else
      "111111" when (d='1');
end Mapper;
```

ซึ่งเมื่อทำการสังเคราะห์ด้วยโปรแกรม MAX+PlusII ได้



รูปที่ 3.28 block วงจรค่าเครื่องหมายที่ได้จากการ Syntersis

ตัวอย่างที่ 8: ดังเช่นในตัวอย่างจากการบวกในตัวอย่างที่ 7 และค่า PN code ที่ได้จากตัวอย่างที่ 4 จะได้ว่า

DF-CDMA*PN = CDPN

CDPN of User #1: DS-CDMA(0)*PN01 = CDPN0

ของสัญญาณบิตที่ 1:

MSB CDPN00(10) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(09) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(08) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(07) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(06) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(05) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

ค่าที่ได้จากวงวงจร Sign

CDPN00(04) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(03) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(02) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(01) = 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0

LSB CDPN00(00) = 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 1 1 1 1 1 1 1 1

MSB CDPN01(10) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(09) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(08) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(07) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(06) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(05) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

ค่าที่ได้จากวงวงจร Sign

CDPN01(04) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(03) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(02) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0

CDPN01(01) = 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0

LSB CDPN01(00) = 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 1 1 1 1 1 1 1 1

ของสัญญาณบิตที่ 3:

MSB CDPN02(10) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(09) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(08) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(07) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(06) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(05) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

} ค่าที่ได้จากวงวงจร Sign

CDPN02(04) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(03) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(02) = 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1

CDPN02(01) = 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0 1 1 1 1

LSB CDPN02(00) = 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 1 1 1 1 1 1 1

CDPN of User #2: DS-CDMA(1)*PN13 = CDPN1

ของสัญญาณบิตที่ 1:

MSB CDPN10(10) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(09) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(08) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(07) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(06) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(05) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN10(04) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(03) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(02) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(01) = 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1

LSB CDPN10(00) = 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 1 1 1 1 1 1 1

ของสัญญาณบิตที่ 2:

MSB CDPN11(10) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(09) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(08) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(07) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(06) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(05) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN11(04) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(03) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(02) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN11(01) = 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1
LSB CDPN11(00) = 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 1 1 1 1 1 1 1

ของสัญญาณบิตที่ 3:

MSB CDPN12(10) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(09) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(08) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(07) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(06) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(05) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0

} ค่าที่ได้จากวงวงจร Sign

CDPN12(04) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(03) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(02) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(01) = 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0
LSB CDPN12(00) = 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 1 1 1 1 1 1 1

CDPN of User #3: DS-CDMA(2)*PN31 = CDPN2

ของสัญญาณบิตที่ 1:

MSB CDPN20(10) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(09) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(08) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(07) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(06) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(05) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
} ค่าที่ได้จากวงวงจร Sign
CDPN20(04) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(03) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(02) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1
CDPN20(01) = 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1
LSB CDPN20(00) = 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 1 1 1 1 1 1 1

ของสัญญาณบิตที่ 2:

MSB CDPN21(10) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(09) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(08) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(07) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(06) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(05) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
} ค่าที่ได้จากวงวงจร Sign
CDPN21(04) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(03) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(02) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1
CDPN21(01) = 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1
LSB CDPN21(00) = 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 1 1 1 1 1 1 1

ของสัญญาณบิตที่ 3:

MSB CDPN22(10) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(09) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(08) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(07) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(06) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(05) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0

} ค่าที่ได้จากวงวงจร Sign

CDPN22(04) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(03) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(02) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0
CDPN22(01) = 0 1 0 1 1 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0
LSB CDPN22(00) = 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 1 1 1 1 1 1 1

3.2.2.4 การบวกสะสมค่า

การบวกสะสมค่านั้นจะถูกควบคุมจังหวะการบวก และการส่งผ่านค่าโดยวงจรควบคุม โดยที่วงจรบวกสะสมค่านั้นจะประกอบด้วย 3 วงจรย่อยๆ คือ

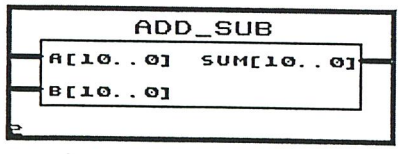
- วงจรบวกแบบ 2's compliment
- วงจรป้อนกลับที่ถูกควบคุมจังหวะโดยสัญญาณ control
- วงจร Buffer สำหรับส่งค่าออกไปยังส่วนตัดสินโลจิก

3.2.2.4.1 วงจรบวกแบบ 2's compliment

จะทำโดยการบรรยาย code โดยภาษา VHDL โดยมีใจความดังนี้ ในส่วนแรกจะเป็นการประกาศ library ที่ต้องการนำมาใช้ในส่วนในส่วนถัดมาคือส่วน entity คือ ส่วนที่เป็นโครงสร้างของอุปกรณ์โดยการประกาศจะให้มี port input อยู่ 2 port คือ a, b โดยจะประกาศเป็นชนิด vector (เป็น bus ขนาด 12 เส้น)และจะมี output port คือ sum โดยจะประการเป็นชนิด vector หรือเป็น bus ขนาด 12 เส้นนั้นเอง

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add_sub is
      port(   a : in std_logic_vector(10 downto 0);
              b : in std_logic_vector(10 downto 0);
            sum : out std_logic_vector(10 downto 0));
end add_sub;

architecture rtl of add_sub is
      signal temp : std_logic_vector(10 downto 0);
begin
      temp<=a;
      process(a,b)
            begin
                  sum<=b+temp;
      end process;
end rtl;
```

ในส่วนโครงสร้างการทำงาน(Architecture) นั้นจะทำการประกาศให้เป็นการทำงานแบบ process โดยที่การทำงานแบบ process ซึ่งจะเหมือนกับการเขียนโปรแกรมภาษาชั้นสูง คือ จะมีการเรียงลำดับการทำงานลงมาในแต่ละบรรทัดโดยการโหลดค่าใน port a จะถูกโหลดค่าเข้าใน signaling ชื่อ temp จากนั้นจะนำค่าจาก temp บวกค่าที่ได้จาก input port b แล้วส่งออกขา sum จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.29 block วงจรบวกลบค่า ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

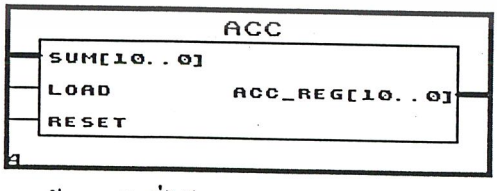
3.2.2.4.2 วงจรป้อนกลับที่ถูกควบคุมจังหวะโดยสัญญาณ control

การป้อนกลับนั้นจะถูกควบคุมโดยสัญญาณนาฬิกาจากตัว control unit เข้าที่ขา load โดยทำงานที่ขอบขาขึ้นโดยมีเงื่อนไขในการทำงานอีกอย่าง คือ ให้ตรวจสอบค่าจาก control unit ที่จะมาเข้าในขาสัญญาณ reset หากมีค่าเป็น 1 ให้ clear ค่าทั้งหมดที่จะออกจาก acc_reg หากไม่ใช่ให้ทำการ load ค่าจากขา sum ไปออกยังขา acc_reg

```
library ieee;
use   ieee.std_logic_1164.all;

entity      acc is
      port (       sum : in std_logic_vector (10 downto 0);
            load,reset : in std_logic;
               acc_reg : out std_logic_vector(10 downto 0));
end;

architecture rtl of acc is
begin
      process(load,reset)
      begin
            if    reset = '1' then
                  acc_reg <= (others=>'0');
            elsif load'event and load='1' then
                  acc_reg<=sum;
            end if;
      end process;
end;
```

จะทำโดยการบรรยาย code โดยภาษา VHDL โดยมีใจความดังนี้ ในส่วนแรกจะเป็นการประกาศ library ที่ต้องการนำมาใช้ในส่วนในส่วนถัดมาคือส่วน entity คือ ส่วนที่เป็นโครงสร้างของอุปกรณ์โดยการประกาศจะให้มี port input อยู่ 3 port คือ sum โดยจะประกาศเป็นชนิด vector (เป็น bus ขนาด 11 เส้น) และ input อีกสองตัวคือ load, reset จะเป็นตัวแปรชนิด std_logic และจะมี output port คือ acc_reg โดยจะประกาศเป็นชนิด vector (เป็น bus ขนาด 11 เส้นนั้นเอง)

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



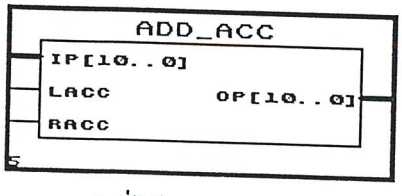
รูปที่ 3.30 block วงจรป้อนกลับที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

วงการบวกสะสมค่าจะต้องทำการนำวงจรในหัวข้อ 3.2.2.5.1 และ 3.2.2.5.2 มาทำการเชื่อมต่อกัน โดยการบรรยายด้วย VHDL code ดังนี้

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity add_acc is
  port(ip : in  std_logic_vector(10 downto 0);
       op : out std_logic_vector(10 downto 0);
       lacc, racc      : in std_logic);
end;

architecture rtl of add_acc is

component add_sub
   port(   a : in  std_logic_vector(10 downto 0);
           b : in  std_logic_vector(10 downto 0);
             sum : out std_logic_vector(10 downto 0));
end component;

component acc
          port (sum : in std_logic_vector (10 downto 0);
         load,reset : in std_logic;
          acc_reg : out std_logic_vector(10 downto 0));
end component;

signal   add_out:std_logic_vector(10 downto 0);
signal   add_in :std_logic_vector(10 downto 0);
signal   acc_out:std_logic_vector(10 downto 0);
signal   acc_in :std_logic_vector(10 downto 0);

begin

   add_in(10 downto 0)<=acc_out(10 downto 0);

   acc_in(10 downto 0)<=add_out(10 downto 0);

summing     : add_sub port map(ip,add_in,add_out);
accumulator : acc     port map(acc_in,lacc,racc,acc_out);

 op(10 downto 0)<=acc_out(10 downto 0);

end;
```

เป็นการประกาศเชื่อมต่อทางโครงสร้างเพื่อที่จะให้ได้อุปกรณ์ ที่มีคุณสมบัติรวมของทั้งสองส่วน ในหัวข้อ 3.2.2.5.1 และ 3.2.2.5.2

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.31 block วงจรบวกสะสมค่าที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

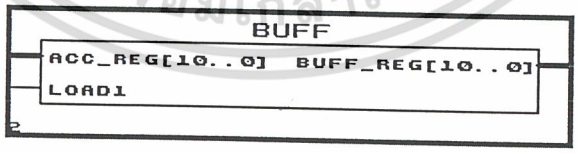
3.2.2.4.3 วงจร Buffer สำหรับส่งค่าออกไปยังส่วนตัดสินโลจิก

จะทำโดยการบรรยาย code โดยภาษา VHDL โดยมีใจความดังนี้ ในส่วนแรกจะเป็นการประกาศ library ที่ต้องการนำมาใช้ในส่วนในส่วนถัดมาคือส่วน entity คือ ส่วนที่เป็นโครงสร้างของอุปกรณ์โดยการประกาศจะให้มี port input อยู่ 2 port คือ acc_reg โดยจะประกาศเป็นชนิด vector (เป็น bus ขนาด 11 เส้น)และ load1 โดยจะประกาศเป็นชนิด std_logic โดยในส่วน output port คือ buff_reg โดยจะประกาศเป็นชนิด vector (เป็น bus ขนาด 11 เส้นนั้นเอง)

```
library ieee;
use   ieee.std_logic_1164.all;

entity buff is
      port (acc_reg  : in std_logic_vector(10 downto 0);
            load1    : in std_logic;
            buff_reg : buffer std_logic_vector(10 downto 0));
end;

architecture rtl of buff is
begin
      process(load1)
      begin
            if load1'event and load1='1'  then
                  buff_reg<=acc_reg;
            end if;
      end process;
end;
```

ในส่วนโครงสร้างการทำงาน(Architecture) นั้นจะทำการประกาศให้เป็นการทำงานแบบ process โดยที่การทำงานแบบ process การโหลดค่าเข้าใน buffer นั้นจะถูกควบคุมโดยสัญญาณนาฬิกาจาก control unit เข้าที่ขา load1 ซึ่งจะทำงานที่ขอบขาขึ้นแล้วจะทำการส่งต่อออกไปในวงจรตัดสินค่าโลจิก

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา



รูปที่ 3.32 block วงจร buffer ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

ตัวอย่างที่ 6: ดังเช่นในตัวอย่างจากการคูณในตัวอย่างที่ 5 เช่น

ทำการบวกค่าสะสมของในแต่ละ user จะได้ว่า

$$CDPNacc = \sum_{0}^{n-1} CDPN(n-1)$$

: โดยที่ n คือจำนวน state ของ LSFR ที่ใช้ในการสร้างสัญญาณ PN code ในที่นี้คือ 31

USER #1: ข้อมูลบิตที่ 1

MSB CDPN00(10) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(09) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(08) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(07) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(06) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(05) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

} ค่าที่ได้จากวงวงจร Sign

CDPN00(04) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(03) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(02) = 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0

CDPN00(01) = 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0

LSB CDPN00(00) = 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 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN00 ตามลำดับ

-1 1 -1 1 3 -1 3 1 3 3 1 1 1 3 -1 -1 -1 3 1 1 3 3 1 -1 1 1 -1 1 1 1 1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc00(10) = 0

CDPNacc00(09) = 0

CDPNacc00(08) = 0

CDPNacc00(07) = 0

CDPNacc00(06) = 0

CDPNacc00(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc00(04) = 1

CDPNacc00(03) = 1

CDPNacc00(02) = 1

CDPNacc00(01) = 1

LSB CDPNacc00(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc00 ตามลำดับจะมีค่าเท่ากับ 31

ข้อมูลบิตที่ 2 คือ

MSB CDPN01(10) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(09) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(08) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(07) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(06) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(05) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

} ค่าที่ได้จากวงวงจร Sign

CDPN01(04) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(03) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(02) = 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

CDPN01(01) = 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0

LSB CDPN01(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN01 ตามลำดับ

3 1 3 1 -1 3 -1 1 -1 -1 1 1 1 -1 3 3 3 -1 1 1 -1 -1 1 3 1 1 3 1 1 1 1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc01(10) = 0

CDPNacc01(09) = 0

CDPNacc01(08) = 0

CDPNacc01(07) = 0

CDPNacc01(06) = 0

CDPNacc01(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc01(04) = 1

CDPNacc01(03) = 1

CDPNacc01(02) = 1

CDPNacc01(01) = 1

LSB CDPNacc01(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc01 ตามลำดับจะมีค่าเท่ากับ 31

ข้อมูลบิตที่ 3 คือ

MSB CDPN02(10) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(09) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(08) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(07) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(06) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(05) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

} ค่าที่ได้จากวงวงจร Sign

CDPN02(04) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(03) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(02) = 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1

CDPN02(01) = 0 1 0 1 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0 1 1 1 1

LSB CDPN02(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN01 ตามลำดับ

-3 -1 -3 -1 1 -3 1 -1 1 1 -1 -1 -1 1 -3 -3 -3 1 -1 -1 1 1 -1 -3 -1 -1 -3 -1 -1 -1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc02(10) = 1

CDPNacc02(09) = 1

CDPNacc02(08) = 1

CDPNacc02(07) = 1

CDPNacc02(06) = 1

CDPNacc02(05) = 1

} ค่าที่ได้จากวงวงจร Sign

CDPNacc02(04) = 0

CDPNacc02(03) = 0

CDPNacc02(02) = 0

CDPNacc02(01) = 0

LSB CDPNacc02(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc02 ตามลำดับจะมีค่าเท่ากับ -31

USER #2:

ข้อมูลบิตที่ 1 คือ

MSB CDPN10(10) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(09) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(08) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(07) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(06) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(05) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN10(04) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(03) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(02) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN10(01) = 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1

LSB CDPN10(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN01 ตามลำดับ

1 1 1 1 3 1 3 1 3 3 -1 1 -1 3 1 1 1 3 -1 -1 3 3 1 1 -1 -1 1 -1 1 1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc10(10) = 0

CDPNacc10(09) = 0

CDPNacc10(08) = 0

CDPNacc10(07) = 0

CDPNacc10(06) = 0

CDPNacc10(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc10(04) = 1

CDPNacc10(03) = 1

CDPNacc10(02) = 1

CDPNacc10(01) = 1

LSB CDPNacc10(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc10 ตามลำดับจะมีค่าเท่ากับ 31

ข้อมูลบิตที่ 2 คือ

MSB CDPN11(10) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(09) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(08) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(07) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(06) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(05) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN11(04) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(03) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(02) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN11(01) = 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1

LSB CDPN11(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN11 ตามลำดับ

-3 1 -3 1 -1 -3 -1 1 -1 -1 -1 1 -1 -1 -3 -3 -3 -1 -1 -1 -1 -1 1 -3 -1 -1 -3 -1 1 1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc11(10) = 1

CDPNacc11(09) = 1

CDPNacc11(08) = 1

CDPNacc11(07) = 1

CDPNacc11(06) = 1

CDPNacc11(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc11(04) = 1

CDPNacc11(03) = 1

CDPNacc11(02) = 1

CDPNacc11(01) = 1

LSB CDPNacc11(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc11 ตามลำดับจะมีค่าเท่ากับ -33

ข้อมูลบิตที่ 3 คือ

MSB CDPN12(10) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(09) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(08) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(07) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(06) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(05) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0

} ค่าที่ได้จากวงวงจร Sign

CDPN12(04) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(03) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(02) = 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
CDPN12(01) = 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0
LSB CDPN12(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN11 ตามลำดับ

3 -1 3 -1 1 3 1 -1 1 1 1 -1 1 1 3 3 3 1 1 1 1 -1 3 1 1 3 1 -1 -1 1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc12(10) = 0
CDPNacc12(09) = 0
CDPNacc12(08) = 0
CDPNacc12(07) = 0
CDPNacc12(06) = 0
CDPNacc12(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc12(04) = 1
CDPNacc12(03) = 1
CDPNacc12(02) = 1
CDPNacc12(01) = 1
LSB CDPNacc12(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc12 ตามลำดับจะมีค่าเท่ากับ 33

USER #3:

ข้อมูลบิตที่ 1 คือ

MSB CDPN20(10) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(09) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(08) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(07) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(06) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(05) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN20(04) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(03) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(02) = 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1

CDPN20(01) = 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1

LSB CDPN20(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN20 ตามลำดับ

-1 1 -1 1 -3 -1 -3 1 -3 -3 -1 1 -1 -3 -1 -1 -1 -3 -1 -1 -3 -3 1 -1 -1 -1 -1 -1 1 1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc20(10) = 1

CDPNacc20(09) = 1

CDPNacc20(08) = 1

CDPNacc20(07) = 1

CDPNacc20(06) = 1

CDPNacc20(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc20(04) = 1

CDPNacc20(03) = 1

CDPNacc20(02) = 1

CDPNacc20(01) = 1

LSB CDPNacc20(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc20 ตามลำดับจะมีค่าเท่ากับ -33

ข้อมูลบิตที่ 2 คือ

MSB CDPN21(10) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(09) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(08) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(07) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(06) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(05) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

} ค่าที่ได้จากวงวงจร Sign

CDPN21(04) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(03) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(02) = 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1

CDPN21(01) = 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1

LSB CDPN21(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN21ตามลำดับ

3 1 3 1 1 3 1 1 1 1 -1 1 -1 1 3 3 3 1 -1 -1 1 1 1 3 -1 -1 3 -1 1 1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc21(10) = 0

CDPNacc21(09) = 0

CDPNacc21(08) = 0

CDPNacc21(07) = 0

CDPNacc21(06) = 0

CDPNacc21(05) = 0

} ค่าที่ได้จากวงวงจร Sign

CDPNacc21(04) = 1

CDPNacc21(03) = 1

CDPNacc21(02) = 1

CDPNacc21(01) = 1

LSB CDPNacc21(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc21 ตามลำดับจะมีค่าเท่ากับ 31

ข้อมูลบิตที่ 3 คือ

MSB CDPN22(10) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(09) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(08) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(07) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(06) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(05) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

} ค่าที่ได้จากวงวงจร Sign

CDPN22(04) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(03) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(02) = 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0

CDPN22(01) = 0 1 0 1 1 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0

LSB CDPN22(00) = 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 1 1 1 1 1 1 1

ซึ่งจะมีค่าเท่ากับ CDPN21ตามลำดับ

3 1 3 1 1 3 1 1 1 1 -1 1 -1 1 3 3 3 1 -1 -1 1 1 1 3 -1 -1 3 -1 1 1 -1

จะได้ค่าการบวกรวมออกมา คือ

MSB CDPNacc22(10) = 1

CDPNacc22(09) = 1

CDPNacc22(08) = 1

CDPNacc22(07) = 1

CDPNacc22(06) = 1

CDPNacc22(05) = 1

} ค่าที่ได้จากวงวงจร Sign

CDPNacc22(04) = 0

CDPNacc22(03) = 0

CDPNacc22(02) = 0

CDPNacc22(01) = 0

LSB CDPNacc22(00) = 1

ซึ่งจะมีค่าเท่ากับ CDPNacc22 ตามลำดับจะมีค่าเท่ากับ -31

3.2.2.5 วงจรตัดสินสัญญาณโลจิก

วงจรในส่วนนี้จะทำหน้าที่ในการตัดสินค่าในการทำการกู้สัญญาณกลับว่าจะออกเป็นสัญญาณ โลจิก 0 หรือ โลจิก 1 โดยจะแบ่งการทำงานออกเป็น 2 ส่วนคือ

- ส่วนตัดสินค่าว่าค่า voltage เสมือนมีค่ามากกว่า 0 หรือน้อยกว่า 0 หากว่ามากกว่า 0 ให้แสดงค่าเป็น +1 หากน้อยกว่า 0 จะแสดงค่าเป็น -1
- ส่วนที่ทำการแปลงค่าโลจิกกลับ สืบเนื่องมาจากการแมพค่าหัวข้อที่ 3.2.1.5.1.1

โดยจากหลักการของการบวกแบบ 2's compliment แล้วจะได้ว่าค่าของ MSB จะเป็นค่าของ บิตเครื่องหมาย (sign bit) ดังนั้นเราจะนำค่าของบิตเครื่องหมายออกมาได้ใช้เลยเสมือนการตัดสินค่า จากนั้นมาทำการแปลงค่ากลับโดยการบรรยายด้วย VHDL code ดังนี้

```
LIBRARY     ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Outta IS
      PORT(
            d     : IN  STD_LOGIC_VECTOR(11 downto 0);
            y11   : OUT STD_ULOGIC);
    END Outta;

ARCHITECTURE Mapper OF Outta IS
BEGIN
      y11 <= not d(11);
END Mapper;
```

ซึ่งจะมีหลักการทำงานของโปรแกรม คือ ทำการดึงค่าของ MSB ของสัญญาณ input มาตัวเดียว จากนั้นจะทำการ กลับค่าและส่งออกไป

จากนั้นทำการ compile ในโปรแกรม MAX+plus II โดยกด Ctrl+Shift+L จะได้ component ออกมา

รูปที่ 3.33 block วงจรตัดสินค่า logic ที่ได้จากการ synthesis จากโปรแกรม MAX+plus II

บทที่ 4
การทดลองและผลการทดลอง

4.1 ผลการทดลองจากโปรแกรม MATLAB

4.1.1 PN code

4.1.1.1 M-sequence ที่ได้จากการแท็ปที่เสตจที่ 2 และ 5 จากชิฟรีจีสเตอร์ ทั้งหมด 5 สเตจ หรือเขียนเป็นสมการได้ว่า $x^5 + x^2 + 1$

1111100110100100001010111011000
0111110011010010000101011101100
0011111001101001000010101110110
0001111100110100100001010111011
1000111110011010010000101011101
1100011111001101001000010101110
0110001111100110100100001010111
1011000111110011010010000101011
1101100011111001101001000010101
1110110001111100110100100001010
0111011000111110011010010000101
1011101100011111001101001000010
0101110110001111100110100100001
1010111011000111110011010010000
0101011101100011111001101001000
0010101110110001111100110100100
0001010111011000111110011010010
0000101011101100011111001101001
1000010101110110001111100110100
0100001010111011000111110011010
0010000101011101100011111001101
1001000010101110110001111100110
0100100001010111011000111110011
1010010000101011101100011111001
1101001000010101110110001111100
0110100100001010111011000111110
0011010010000101011101100011111
1001101001000010101110110001111
1100110100100001010111011000111
1110011010010000101011101100011
1111001101001000010101110110001

ตารางที่ 4.1 ค่าของชุด PN sequence 31-modulo [2 5]

4.1.1.2 M-sequence ที่ได้จากการแท็ปที่เสตจที่ 2, 3, 4 และ 5 จากชิฟรีจีสเตอร์ ทั้งหมด 5 สเตจ หรือเขียนเป็นสมการได้ว่า $x^5 + x^4 + x^3 + x^2 + 1$

1111100100110000101101010001110
0111110010011000010110101000111
1011111001001100001011010100011
1101111100100110000101101010001
1110111110010011000010110101000
0111011111001001100001011010100
0011101111100100110000101101010
0001110111110010011000010110101
1000111011111001001100001011010
0100011101111100100110000101101
1010001110111111001001100001011
0101000111011111100100110000101
1010100011101111110010011000010
1101010001110111111001001100001
0110101000111011111100100110000
1011010100011101111110010011000
0101101010001110111111100100110
0010110101000111011111100100110
0001011010100011101111110010011
0000101101010001110111111001001
1000010110101000111011111001001
1100001011010100011101111100100
0110000101101010001110111111001
0011000010110101000111011111100
1001100001011010100011101111100
0100110000101101010001110111110
0010011000010110101000111011111
1001001100001011010100011101111
1100100110000101101010001110111
1110010011000010110101000111011
1111001001100001011010100011101

ตารางที่ 4.2 ค่าของชุด PN sequence 31-modulo [2 3 4 5]

4.1.1.3 Gold-sequence ได้จากการนำเอาค่าของ 4.1.1.1 กับ 4.1.1.2 มาทำการ XOR กันจะได้ออกมาเป็นชุดของ Gold sequence

00000000100101001001111101010110
10000101001111000111000100111111
01000111111010000000011011111011
00100110100000100011111010001001
00010110001101110010000011100000
10001110011011011010111000011100
11000010010000001110100101100010
11100100010101100100101011011101
01110111010111010001101100000010
10111110110110000101100111110101
01011010000110100110011110011110
10101000011110110000110110100111
01010001010010111011100010111101
00101101111010011111100010001101O
10010011100111111110011110111001
01001100101110011101100111010000
10100011001010101101001010000000
11010100111000110101011100101000
11101111000001111001010111111110
11110010111110101111110100100101
01111100000011001100010000100001
00111011011100000101110001111100
10011000110011100001000001010100
11001001000100010011011001000001
01100001111111110101001010100100
10110101100010010110110011001100
11011111101100101000100000000111
01101010101011111011110100110111
00110000001000011000001101011111
00011101011001101111111111100011
00001011110001010100000110001010

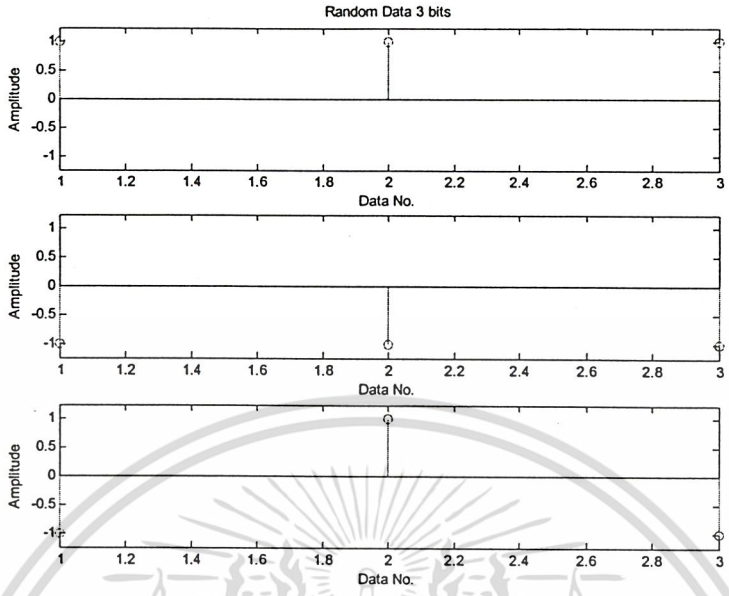
ตารางที่ 4.3 ค่าของชุด Gold sequence

4.1.2 ผลที่ได้จากการสุ่มค่าเพื่อสร้างชุดข้อมูลโดยใช้คำสั่ง ceil(2*rand(31,31))-1

0 0 0 1 1 1 1 1 0 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1
1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0 1 0
1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1
0 0 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 0 0
0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0
0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0
1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1
0 0 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 0 1
1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1
0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1
1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1
0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1
1 0 0 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 1
0 1 0 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 0 1 0 1
0 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1
0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1
0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 0 1
1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1
1 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 1 0
0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0
1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1
1 0 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0
0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 1 0 0 1
0 1 1 1 0 1 1 0 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 1 1 0
0 0 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 1
0 1 0 0 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 1 1
0 1 1 1 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 1

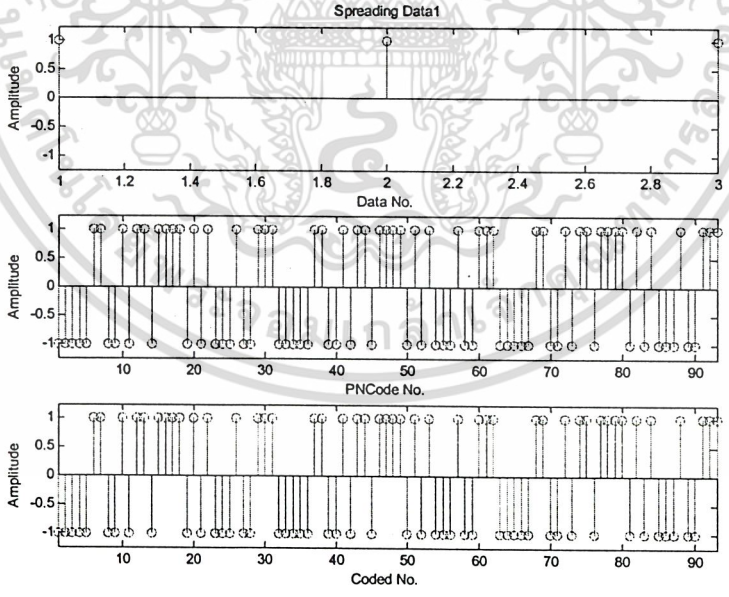
ตารางที่ 4.4 ค่าของชุด ceil(2*rand(31,31))-1

4.1.3 การสุ่มค่าของข้อมูล 3 ชุด



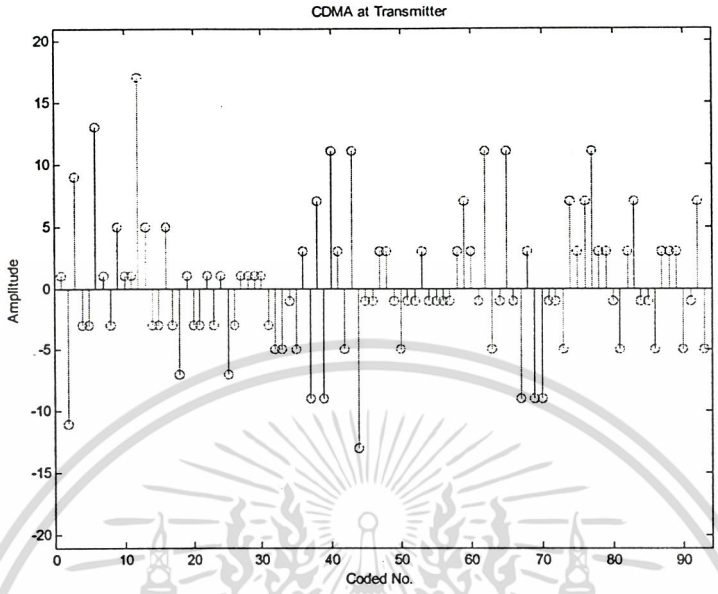
รูปที่ 4.1 แสดงผลของการสุ่มค่าของข้อมูล 3 ชุด

4.1.4 ค่าแอมปลิจูดของสัญญาณ Spreading Data ของข้อมูล 3 ชุด



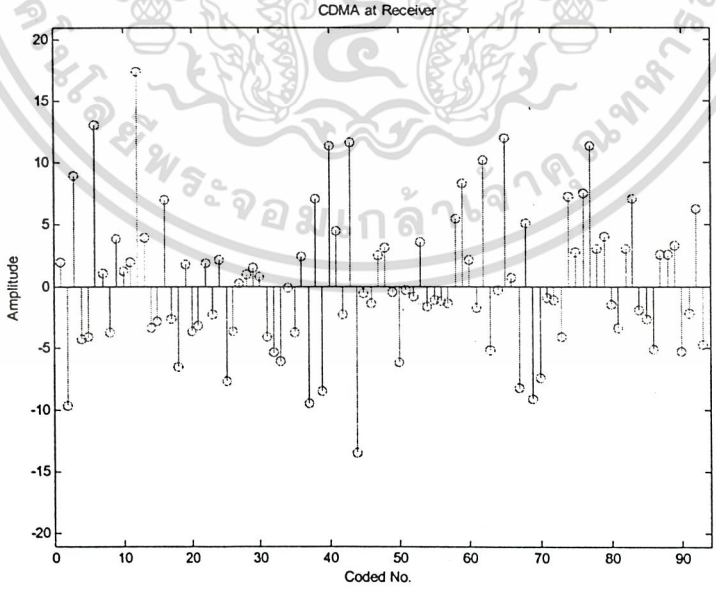
รูปที่ 4.2 แสดงผลจากการทำ Spreading Data 3 ชุด

4.1.5 ค่าแอมปลิจูดของสัญญาณ CDMA ที่ด้านส่ง



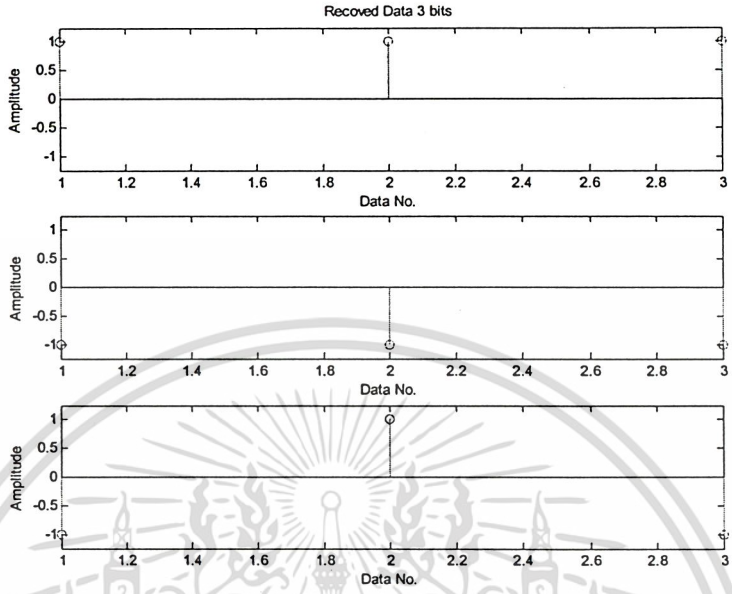
รูปที่ 4.3 แสดงค่าแอมปลิจูดของสัญญาณ CDMA ณ ด้านส่ง

4.1.6 ผลที่ได้จากการถอดรหัสสัญญาณ CDMA ที่ด้านรับได้จากด้านส่ง



รูปที่ 4.4 แสดงผลของสัญญาณที่ถูกรบกวนด้วย AGWN ที่ SNR=16 dB

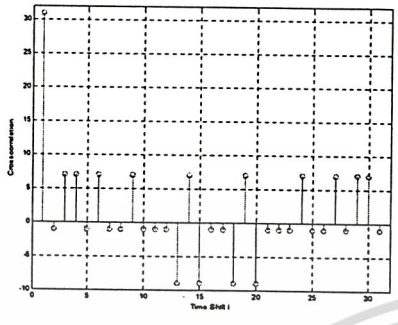
4.1.7 ผลที่ได้จากการทำ Data Recovery



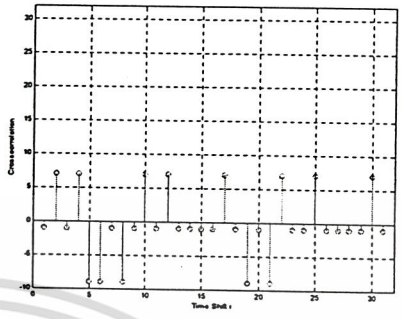
รูปที่ 4.5 แสดงผลที่ได้จากการทำ Data Recovery

4.1.8 ผลการพล็อตค่าครอสคอรีเลชั่น และค่าออร์โต้คอรีเลชั่น

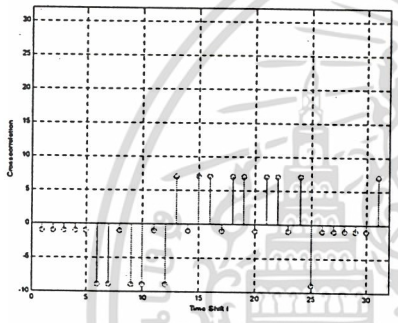
Gold sequence ชุดที่ 1 เทียบกับชุดข้อมูลอื่นๆ G(m,n) = goldseq(m1,m2,n);



รูปที่ 4.6 แสดงค่าออร์โต้คอรีเลชั่น G1

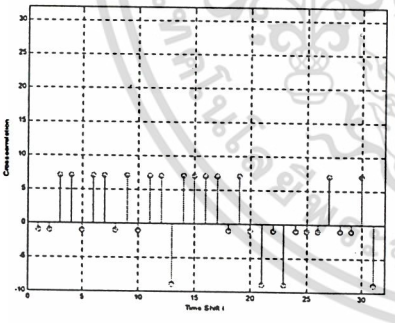


รูปที่ 4.7 แสดงค่าครอสคอรีเลชั่น G1,G2

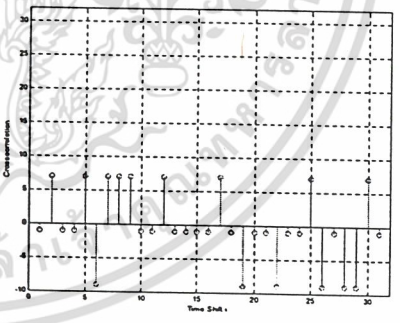


รูปที่ 4.8 แสดงค่าครอสคอรีเลชั่น G1,G3

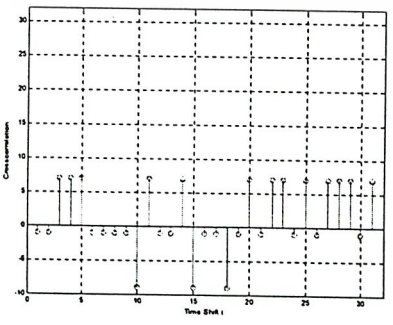
รูปที่ 4.9 แสดงค่าครอสคอรีเลชั่น G1,G4



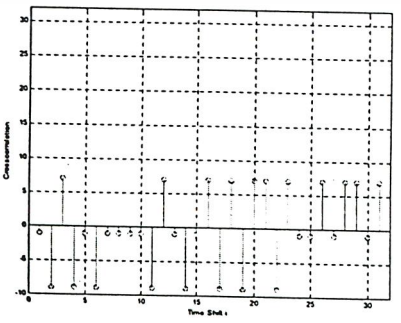
รูปที่ 4.10 แสดงค่าครอสคอรีเลชั่นG1,G5



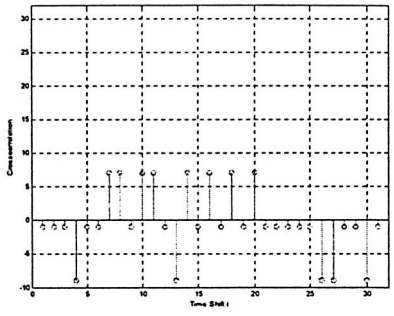
รูปที่ 4.11 แสดงค่าครอสคอรีเลชั่นG1,G6



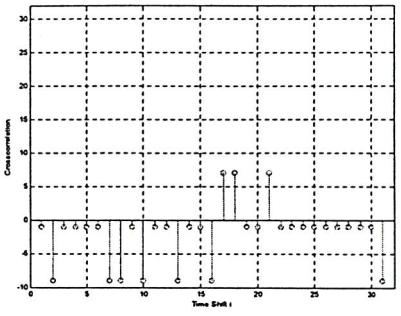
รูปที่ 4.12 แสดงค่าครอสคอรีเลชั่น G1,G7



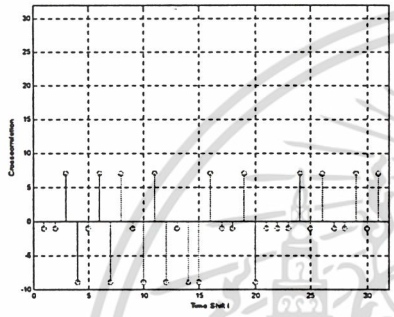
รูปที่ 4.13 แสดงค่าครอสคอรีเลชั่น G1,G8



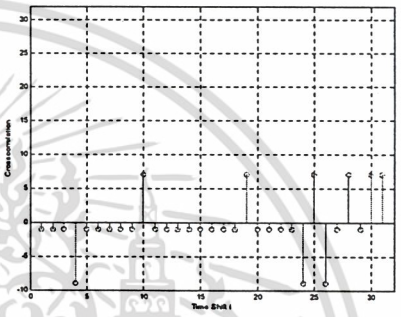
รูปที่ 4.14 แสดงค่าครอสคอรีเลชั่น G1,G09



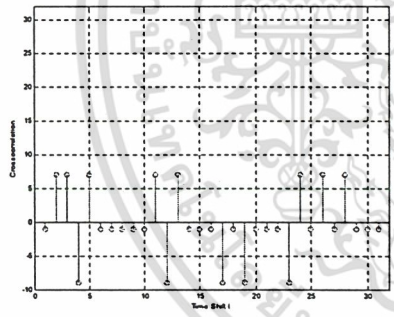
รูปที่ 4.15 แสดงค่าครอสคอรีเลชั่น G1,G10



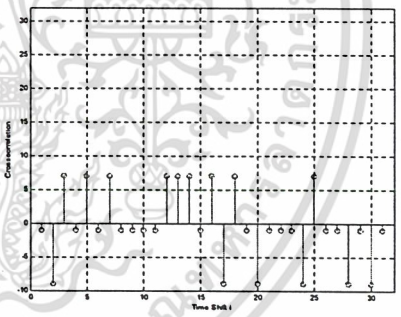
รูปที่ 4.16 แสดงค่าครอสคอรีเลชั่น G1,G11



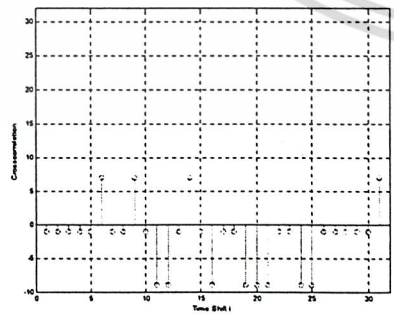
รูปที่ 4.17 แสดงค่าครอสคอรีเลชั่น G1,G12



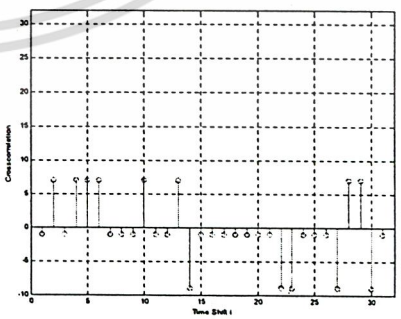
รูปที่ 4.18 แสดงค่าครอสคอรีเลชั่น G1,G13



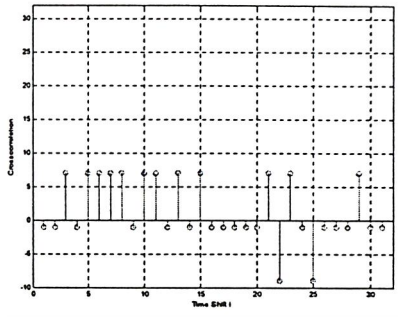
รูปที่ 4.19 แสดงค่าครอสคอรีเลชั่น G1,G14



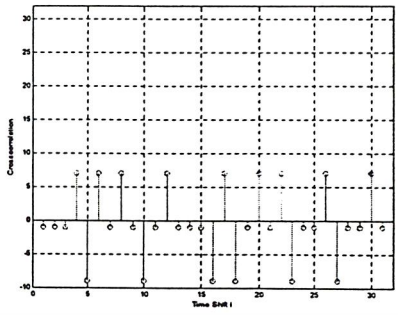
รูปที่ 4.20 แสดงค่าครอสคอรีเลชั่น G1,G15



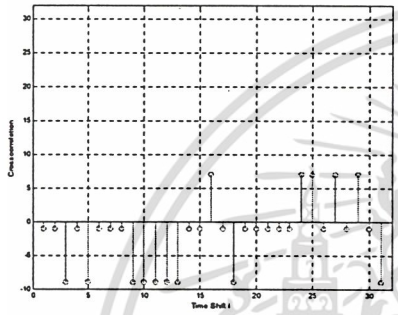
รูปที่ 4.21 แสดงค่าครอสคอรีเลชั่น G1,G16



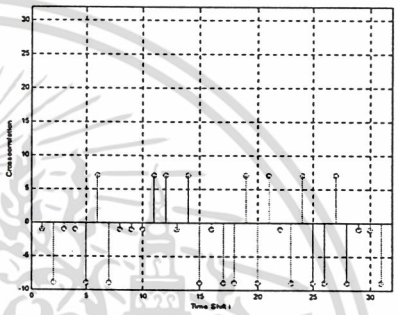
รูปที่ 4.22 แสดงค่าครอสคอรีเลชั่น G1,G17



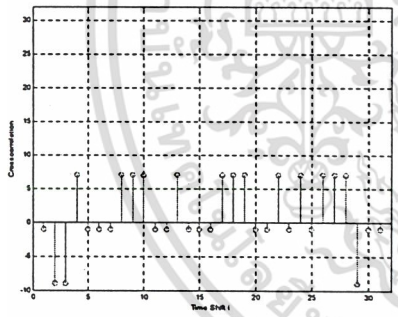
รูปที่ 4.23 แสดงค่าครอสคอรีเลชั่น G1,G18



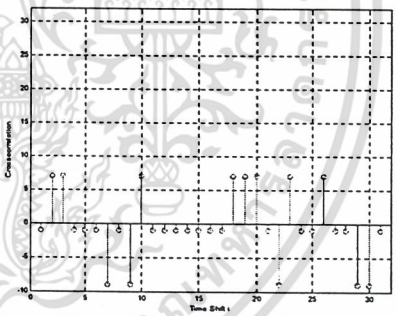
รูปที่ 4.24 แสดงค่าครอสคอรีเลชั่น G1,G19



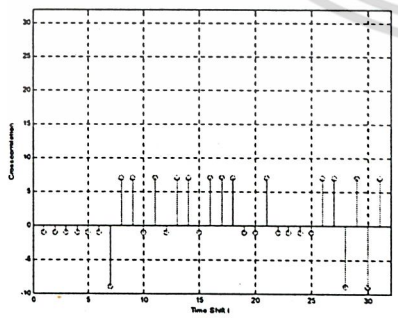
รูปที่ 4.25 แสดงค่าครอสคอรีเลชั่น G1,G20



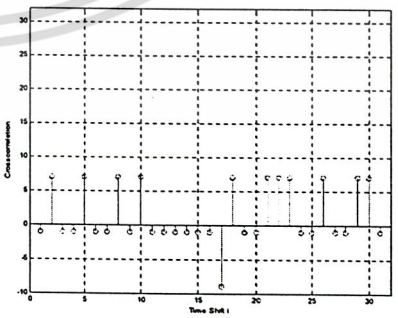
รูปที่ 4.26 แสดงครอสคอรีเลชั่น G1,G21



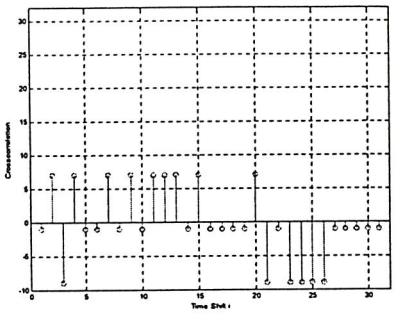
รูปที่ 4.27 แสดงค่าครอสคอรีเลชั่น G1,G22



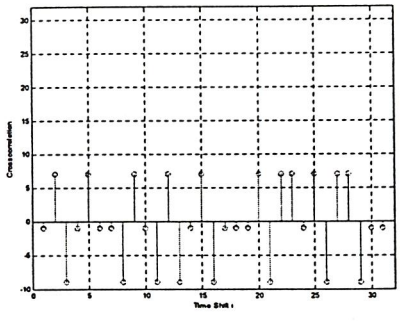
รูปที่ 4.28 แสดงค่าครอสคอรีเลชั่นน G1,G23



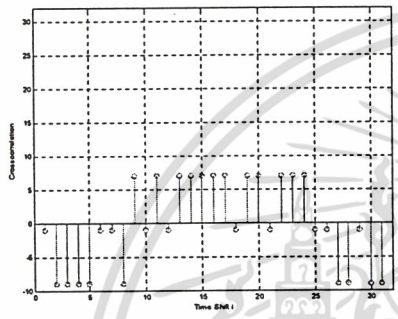
รูปที่ 4.29 แสดงครอสคอรีเลชั่น G1,G24



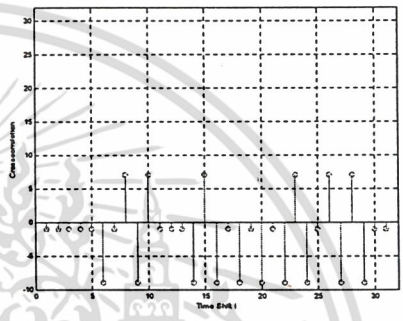
รูปที่ 4.30 แสดงค่าครอสคอรีเลชั่น G1,G25



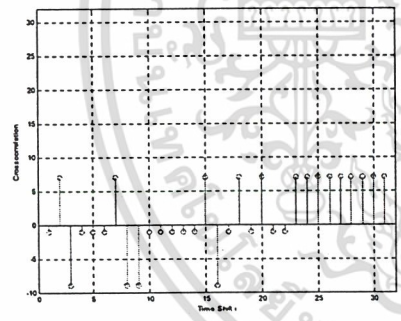
รูปที่ 4.31 แสดงค่าครอสคอรีเลชั่น G1,G26



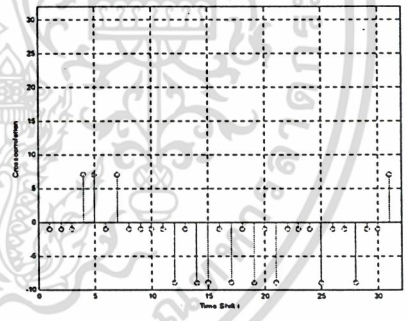
รูปที่ 4.32 แสดงค่าครอสคอรีเลชั่น G1,G27



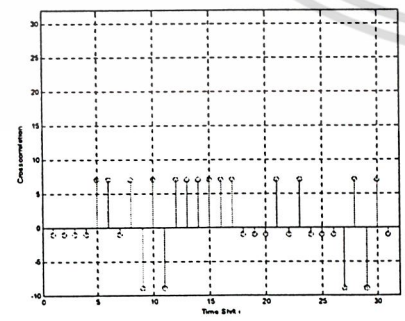
รูปที่ 4.33 แสดงค่าครอสคอรีเลชั่น G1,G28



รูปที่ 4.34 แสดงค่าครอสคอรีเลชั่น G1,G29

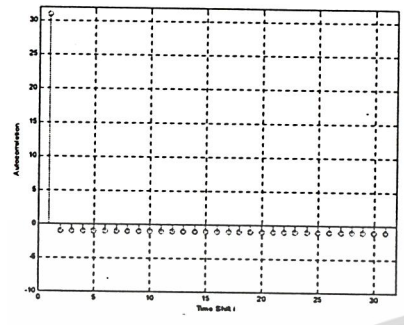


รูปที่ 4.35 แสดงค่าครอสคอรีเลชั่น G1,G30

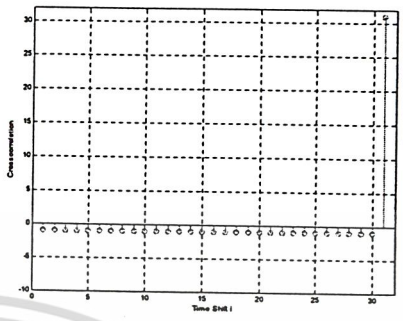


รูปที่ 4.36 แสดงค่าครอสคอรีเลชั่น G1,G31

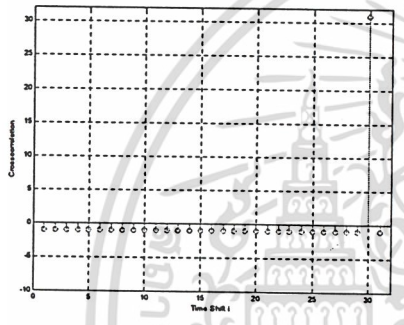
4.1.9 ผลการพล็อตค่าครอสคอรีเลชั่น และค่าออร์โต้คอรีเลชั่น m1 ชุดที่ 1 เทียบกับชุดข้อมูลอื่นๆ [2 5], [1 1 1 1 1]



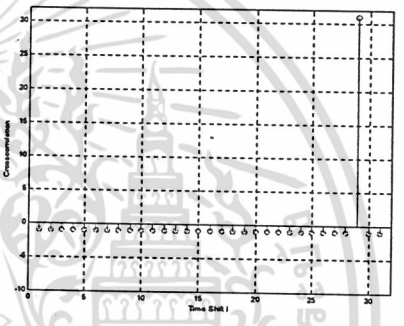
รูปที่ 4.37 แสดงค่าออร์โต้คอรีเลชั่น M1



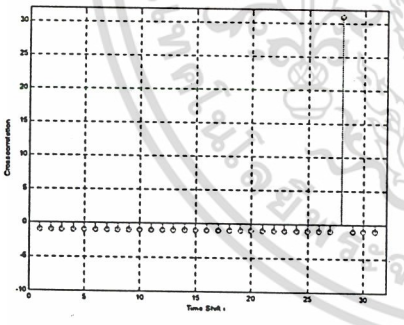
รูปที่ 4.38 แสดงค่าคอสคอรีเลชัน M1, M2



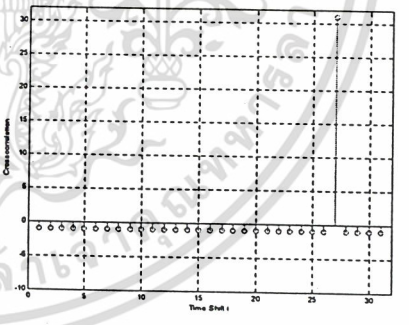
รูปที่ 4.39 แสดงค่าคอสคอรีเลชัน M1, M3



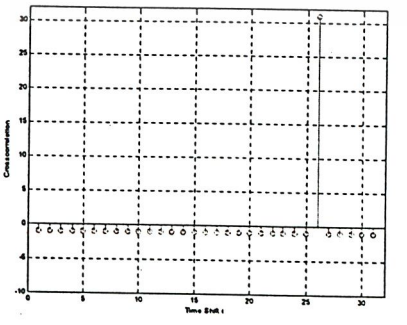
รูปที่ 4.40 แสดงค่าคอสคอรีเลชัน M1, M4



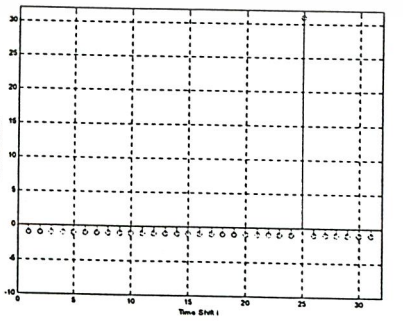
รูปที่ 4.41แสดงค่าคอสคอรีเลชัน M1, M5



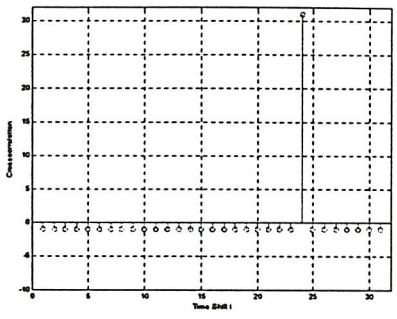
รูปที่ 4.42 แสดงค่าคอสคอรีเลชัน M1, M6



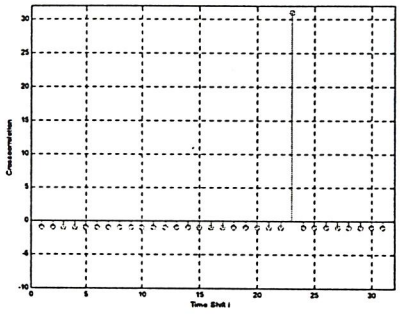
รูปที่ 4.43 แสดงค่าคอสคอรีเลชัน M1, M7



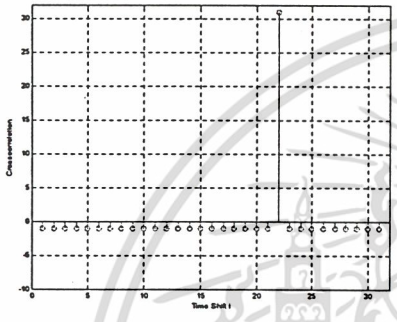
รูปที่ 4.44 แสดงค่าคอสคอรีเลชัน M1, M8



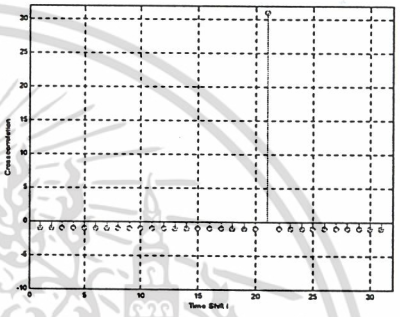
รูปที่ 4.45 แสดงค่าคอสคอรีเลชัน M1, M9



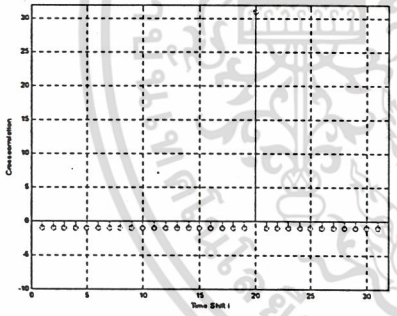
รูปที่ 4.46 แสดงค่าคอสคอรีเลชัน M1, M10



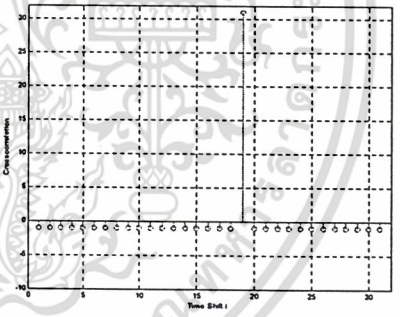
รูปที่ 4.47 แสดงค่าคอสคอรีเลชัน M1, M11



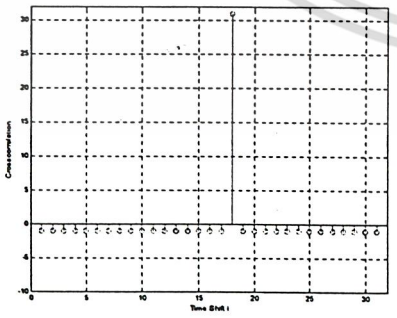
รูปที่ 4.48 แสดงค่าคอสคอรีเลชัน M1, M12



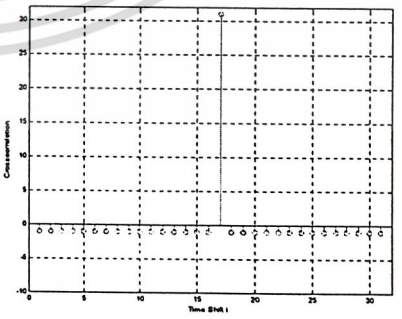
รูปที่ 4.49 แสดงค่าคอสคอรีเลชัน M1, M13



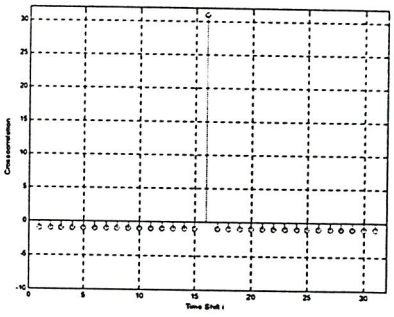
รูปที่ 4.50 แสดงค่าคอสคอรีเลชัน M1, M14



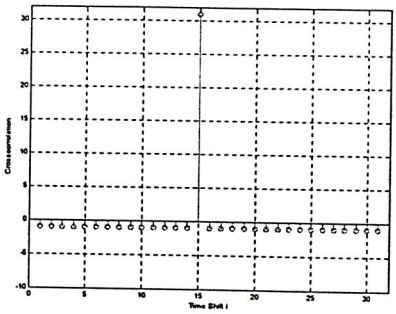
รูปที่ 4.51 แสดงค่าคอสคอรีเลชัน M1, M15



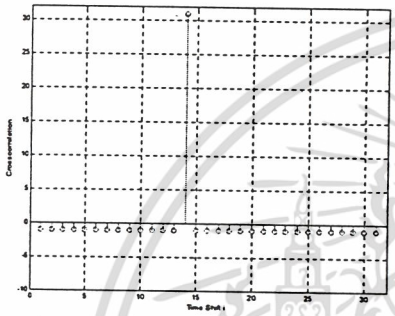
รูปที่ 4.52 แสดงค่าคอสคอรีเลชัน M1, M16



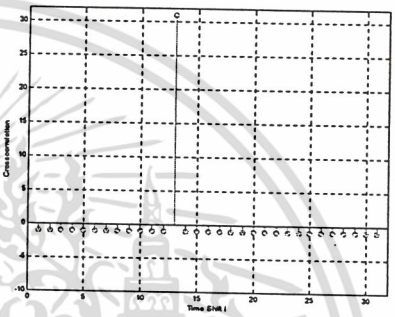
รูปที่ 4.53 แสดงค่าคอสคอรีเลชัน M1, M17



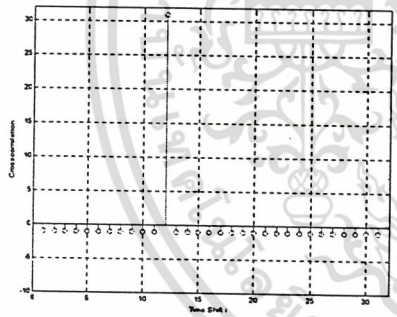
รูปที่ 4.54 แสดงค่าคอสคอรีเลชัน M1, M18



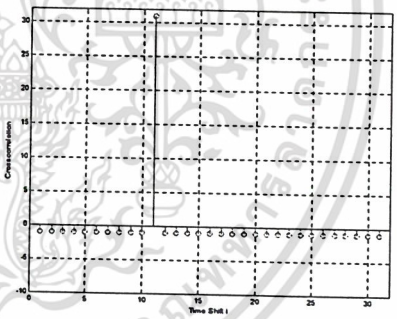
รูปที่ 4.55 แสดงค่าคอสคอรีเลชัน M1, M19



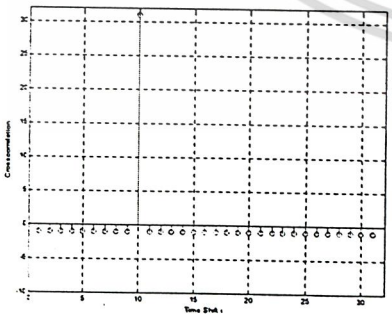
รูปที่ 4.56 แสดงค่าคอสคอรีเลชัน M1, M20



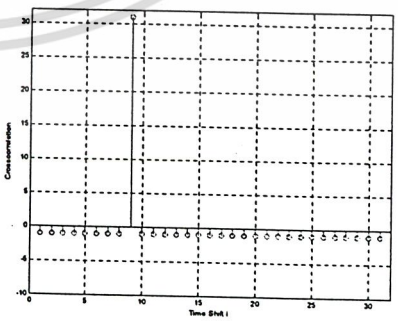
รูปที่ 4.57 แสดงค่าคอสคอรีเลชัน M1, M21



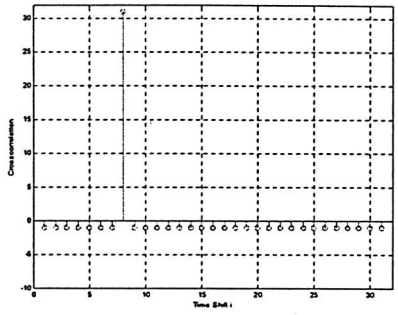
รูปที่ 4.58 แสดงค่าคอสคอรีเลชัน M1, M22



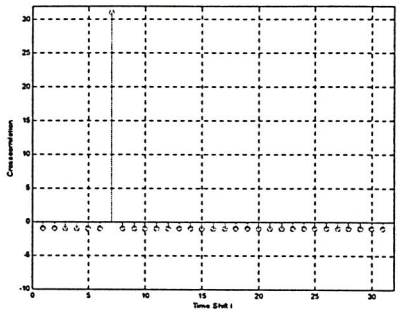
รูปที่ 4.59 แสดงค่าคอสคอรีเลชัน M1, M23



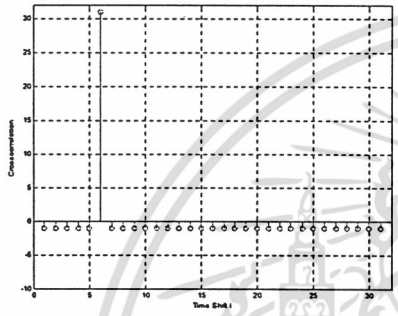
รูปที่ 4.60 แสดงค่าคอสคอรีเลชัน M1, M24



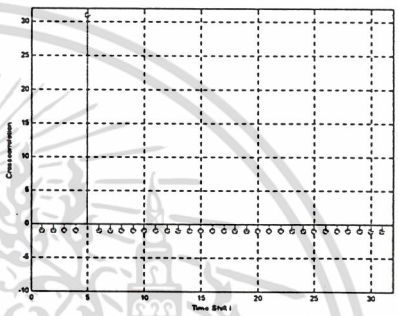
รูปที่ 4.61 แสดงค่าคอสคอรีเลชัน M1, M25



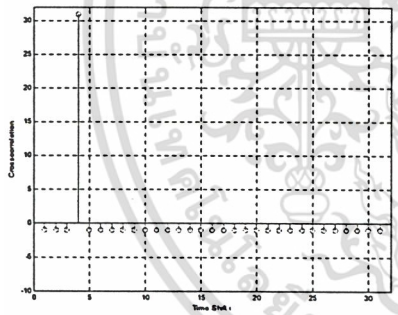
รูปที่ 4.62 แสดงค่าคอสคอรีเลชัน M1, M26



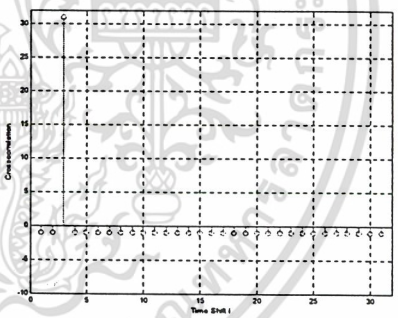
รูปที่ 4.63 แสดงค่าคอสคอรีเลชัน M1, M27



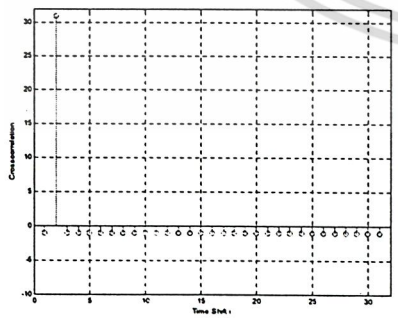
รูปที่ 4.64 แสดงค่าคอสคอรีเลชัน M1, M28



รูปที่ 4.65 แสดงค่าคอสคอรีเลชัน M1, M29



รูปที่ 4.66 แสดงค่าคอสคอรีเลชัน M1, M30

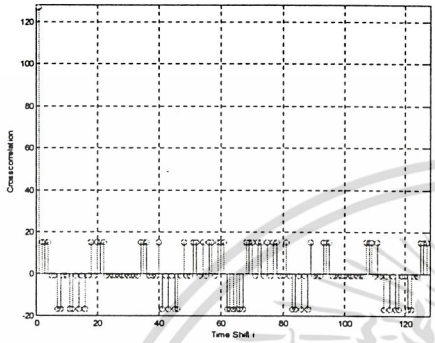


รูปที่ 4.67 แสดงค่าคอสคอรีเลชัน M1, M31

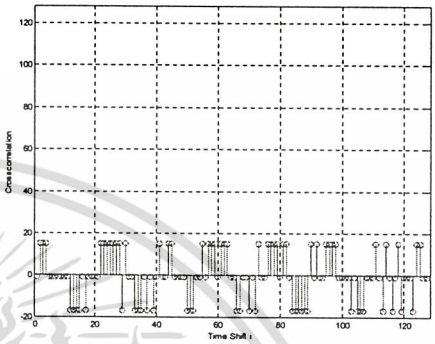
4.1.10 ผลการพล็อตค่าครอสคอรีเลชั่น และค่าออร์โต้คอรีเลชั่น

Gold sequence ชุดที่ 1 เทียบกับชุดข้อมูลอื่นๆ โดยใช้คำสั่งในที่ค่าเริ่มต้นดังนี้

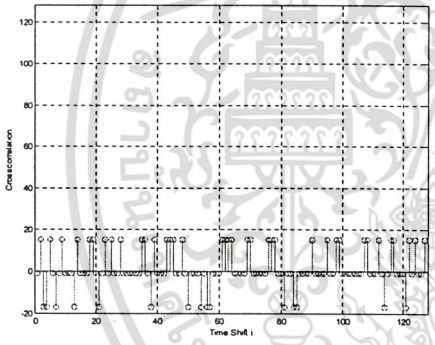
```
m1    = mseq(7,[7 6 5 4 2 1],[1 1 1 1 1 1 1]);
m2    = mseq(7,[7 5 4 3 2 1],[1 1 1 1 1 1 1]);
gs=goldseq(m1, m2, 127);
```



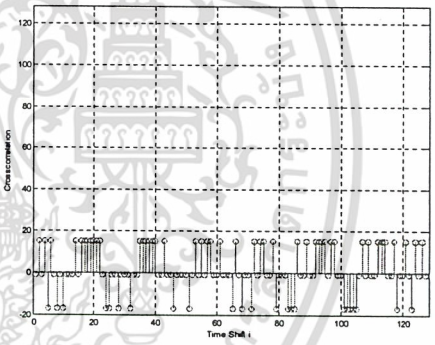
รูปที่ 4.68 แสดงค่าออร์โต้คอรีเลชั่น G1



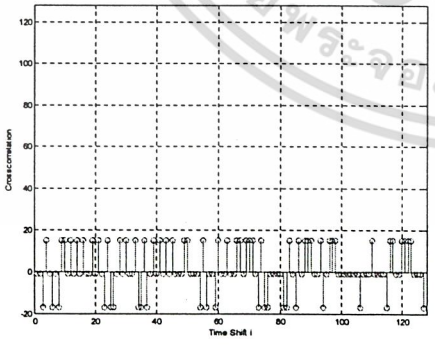
รูปที่ 4.69 แสดงค่าครอสคอรีเลชั่น G1,G2



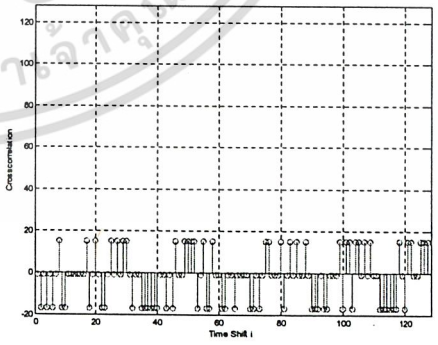
รูปที่ 4.70 แสดงค่าครอสคอรีเลชั่น G1,G3



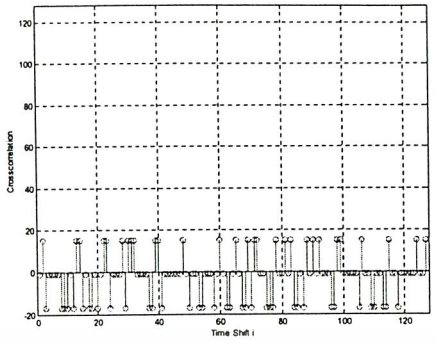
รูปที่ 4.71 แสดงค่าค่าครอสคอรีเลชั่น G1,G4



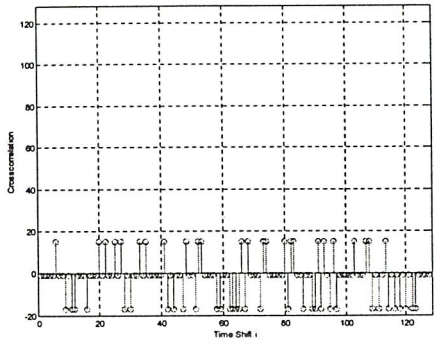
รูปที่ 4.72 แสดงค่าครอสคอรีเลชั่น G1,G5



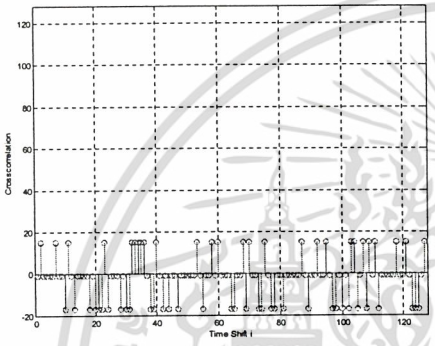
รูปที่ 4.73 แสดงค่าครอสคอรีเลชั่น G1,G6



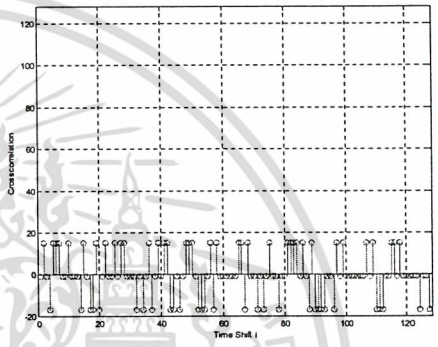
รูปที่ 4.74 แสดงค่าครอสคอรีเลชั่น G1,G7



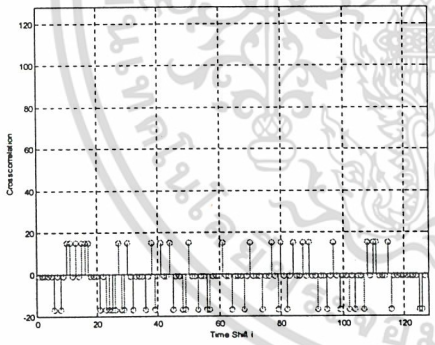
รูปที่ 4.75 แสดงค่าครอสคอรีเลชั่น G1,G8



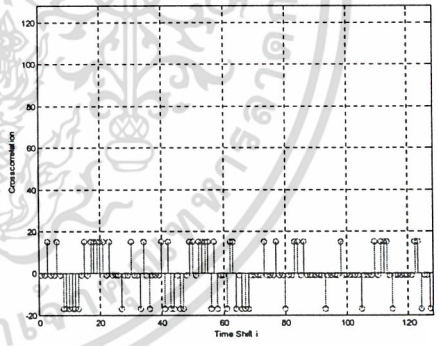
รูปที่ 4.76 แสดงค่าครอสคอรีเลชั่น G1,G9



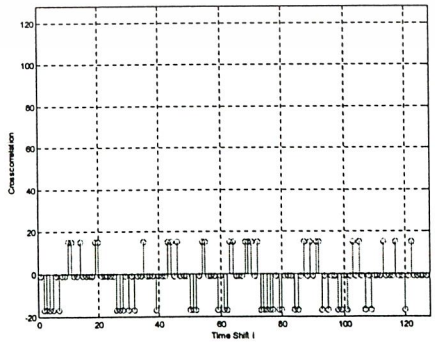
รูปที่ 4.77 แสดงค่าค่าครอสคอรีเลชั่น G1,G10



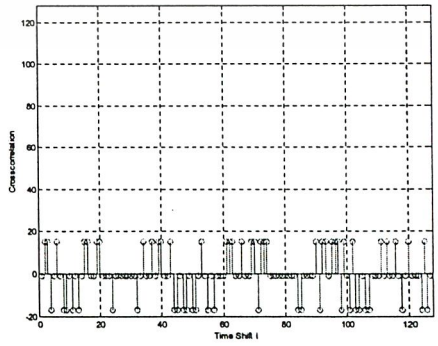
รูปที่ 4.78 แสดงค่าครอสคอรีเลชั่น G1,G11



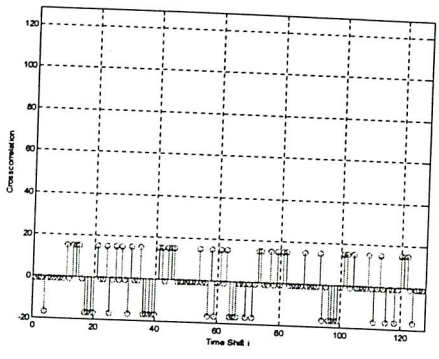
รูปที่ 4.79 แสดงค่าครอสคอรีเลชั่น G1,G12



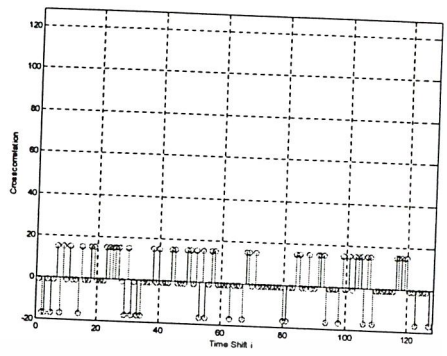
รูปที่ 4.80 แสดงค่าครอสคอรีเลชั่น G1,G13



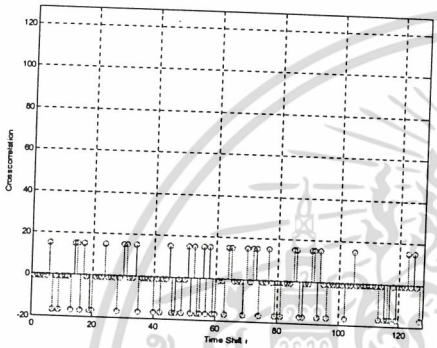
รูปที่ 4.81 แสดงค่าครอสคอรีเลชั่น G1,G14



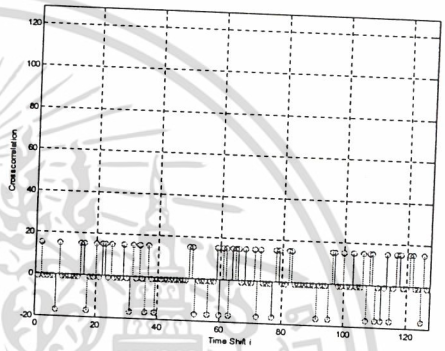
รูปที่ 4.82 แสดงค่าครอสคอรีเลชั่น G1,G15



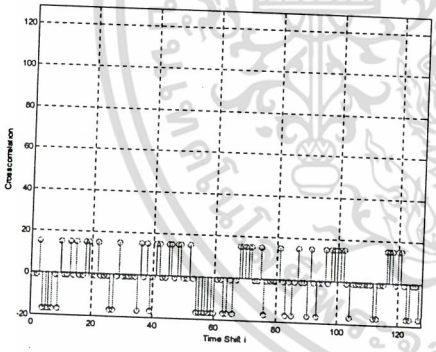
รูปที่ 4.83แสดงค่าครอสคอรีเลชั่น G1,G16



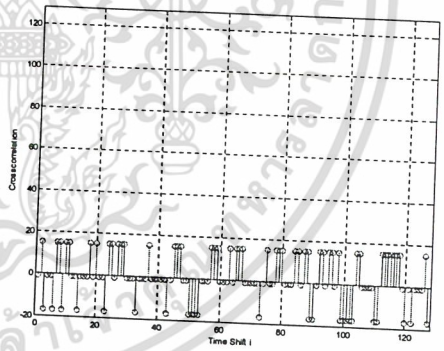
รูปที่ 4.84 แสดงค่าครอสคอรีเลชั่น G1,G17



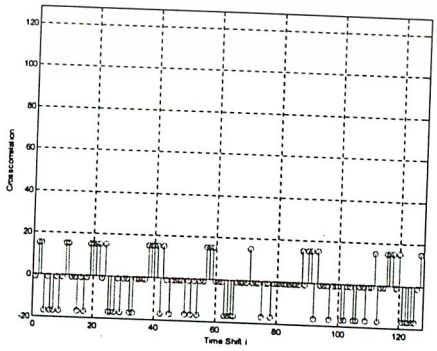
รูปที่ 4.85 แสดงค่าค่าครอสคอรีเลชั่น G1,G18



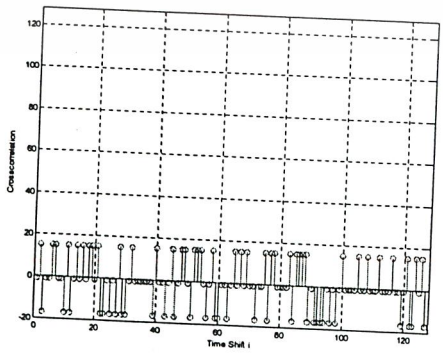
รูปที่ 4.86 แสดงค่าครอสคอรีเลชั่น G1,G19



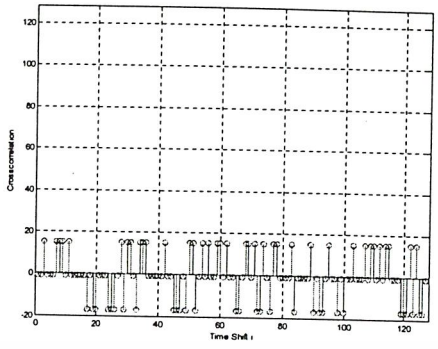
รูปที่ 4.87 แสดงค่าครอสคอรีเลชั่น G1,G20



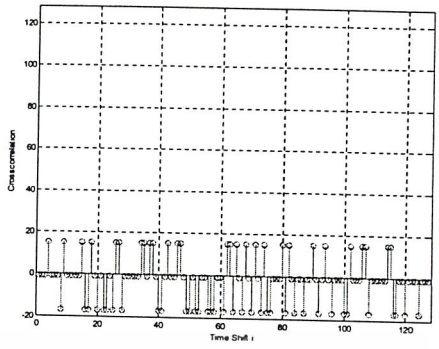
รูปที่ 4.88 แสดงค่าครอสคอรีเลชั่น G1,G21



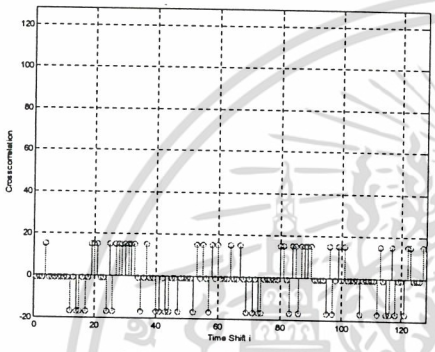
รูปที่ 4.89 แสดงค่าครอสคอรีเลชั่น G1,G22



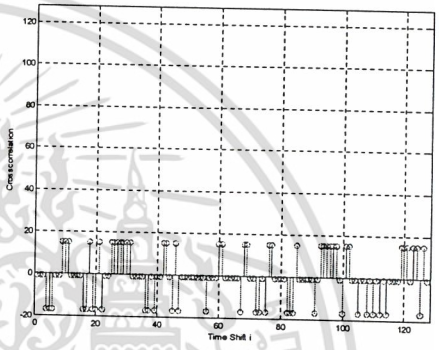
รูปที่ 4.90 แสดงค่าครอสคอรีเลชั่น G1,G23



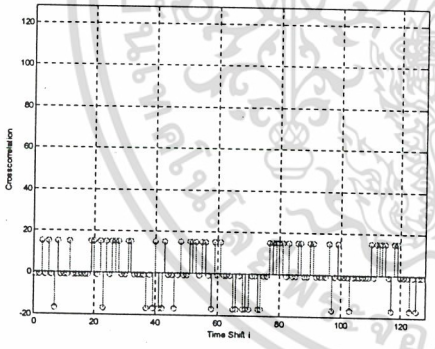
รูปที่ 4.91 แสดงค่าครอสคอรีเลชั่น G1,G24



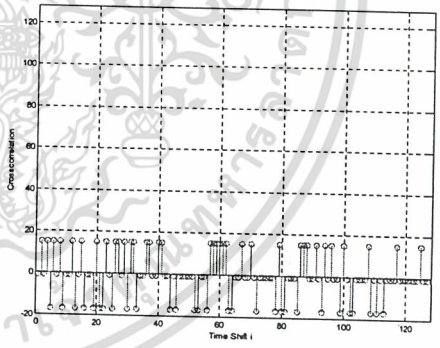
รูปที่ 4.92 แสดงค่าครอสคอรีเลชั่น G1,G25



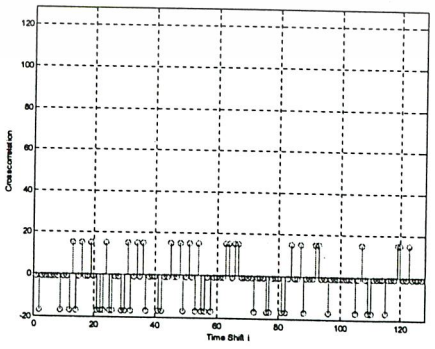
รูปที่ 4.93 แสดงค่าค่าครอสคอรีเลชั่น G1,G26



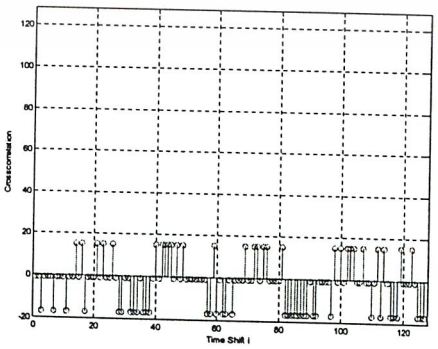
รูปที่ 4.94 แสดงค่าครอสคอรีเลชั่น G1,G27



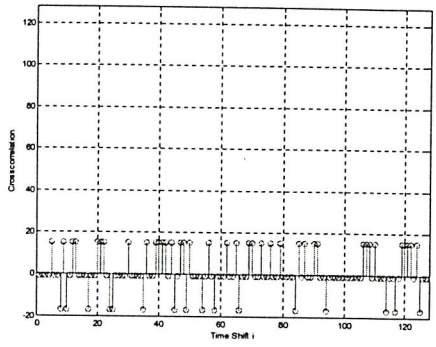
รูปที่ 4.95 แสดงค่าครอสคอรีเลชั่น G1,G28



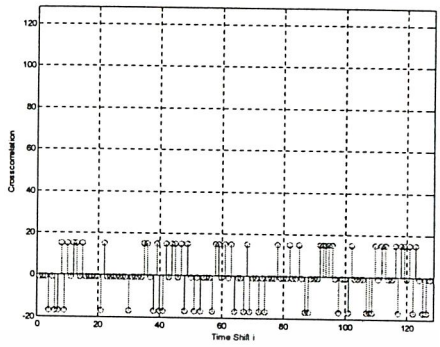
รูปที่ 4.96 แสดงค่าครอสคอรีเลชั่น G1,G29



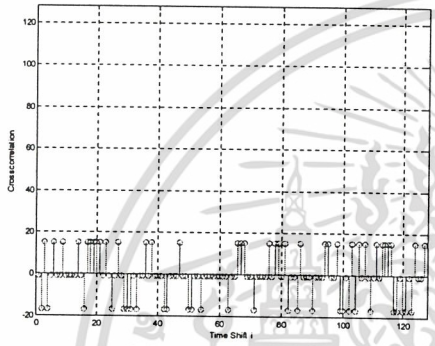
รูปที่ 4.97 แสดงค่าครอสคอรีเลชั่น G1,G30



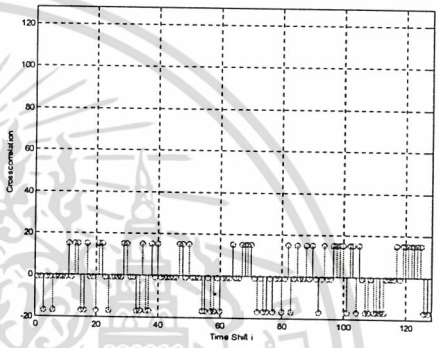
รูปที่ 4.98 แสดงค่าครอสคอรีเลชั่น G1,G31



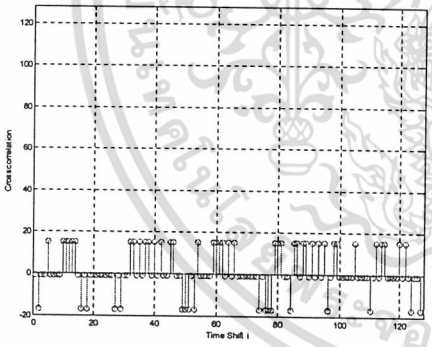
รูปที่ 4.99แสดงค่าครอสคอรีเลชั่น G1,G32



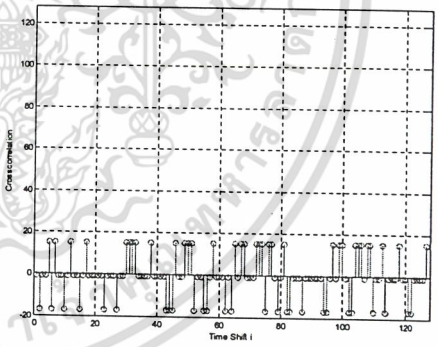
รูปที่ 4.100 แสดงค่าครอสคอรีเลชั่น G1,G33



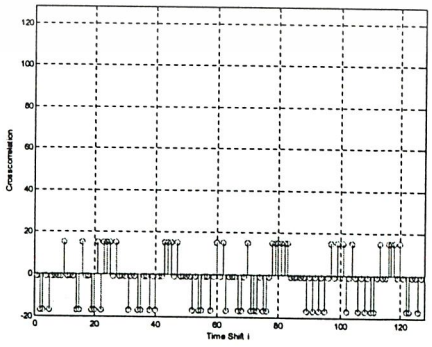
รูปที่ 4.101 แสดงค่าค่าครอสคอรีเลชั่น G1,G34



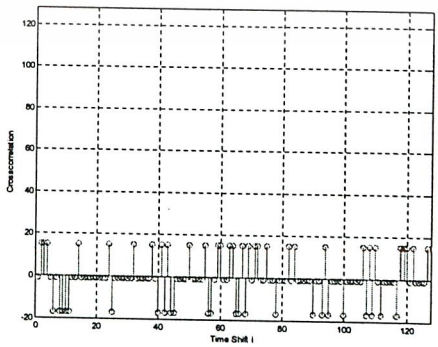
รูปที่ 4.102 แสดงค่าครอสคอรีเลชั่น G1,G35



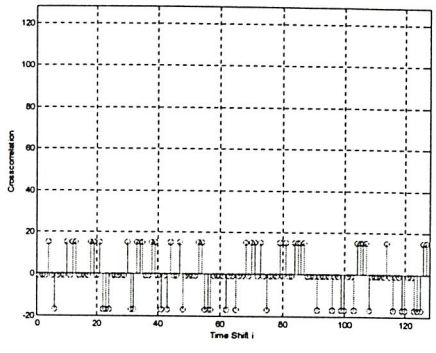
รูปที่ 4.103 แสดงค่าครอสคอรีเลชั่น G1,G136



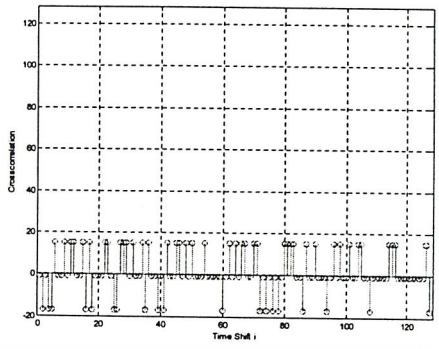
รูปที่ 4.104 แสดงค่าครอสคอรีเลชั่น G1,G37



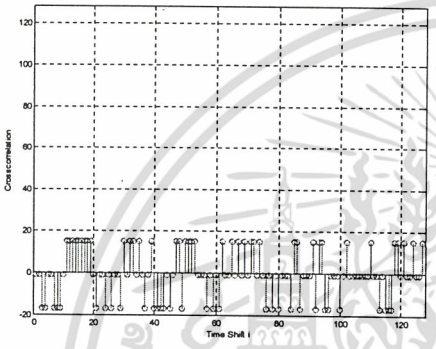
รูปที่ 4.105 แสดงค่าครอสคอรีเลชั่น G1,G38



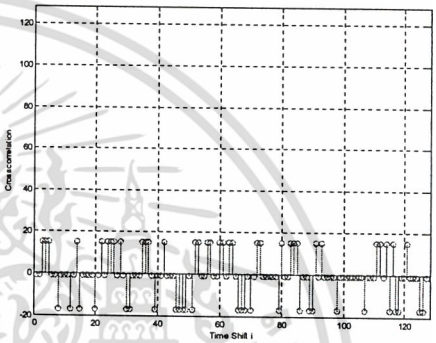
รูปที่ 4.106 แสดงค่าครอสคอรีเลชั่น G1,G39



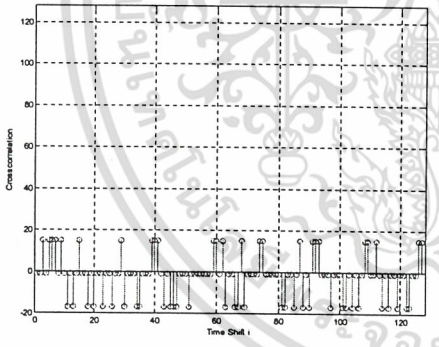
รูปที่ 4.107 แสดงค่าครอสคอรีเลชั่น G1,G40



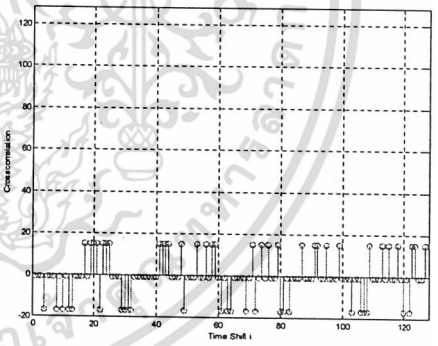
รูปที่ 4.108 แสดงค่าครอสคอรีเลชั่น G1,G41



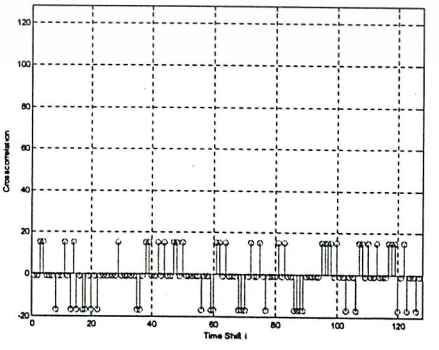
รูปที่ 4.109 แสดงค่าค่าครอสคอรีเลชั่น G1,G42



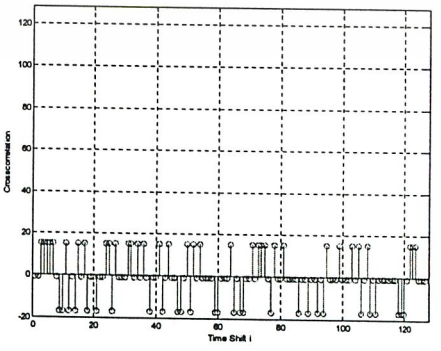
รูปที่ 4.110 แสดงค่าครอสคอรีเลชั่น G1,G43



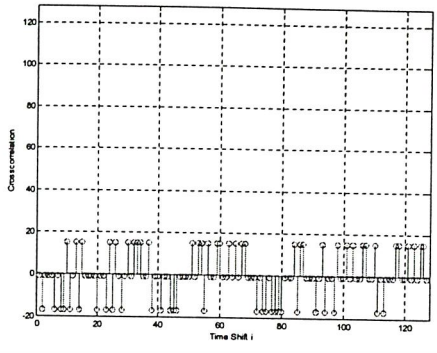
รูปที่ 4.111 แสดงค่าครอสคอรีเลชั่น G1,G44



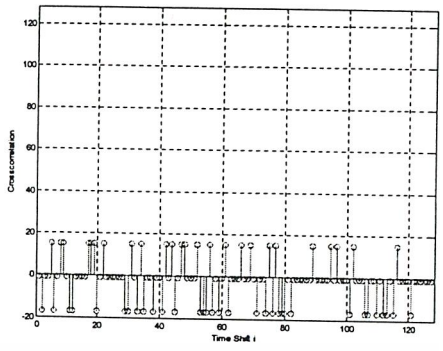
รูปที่ 4.112 แสดงค่าครอสคอรีเลชั่น G1,G45



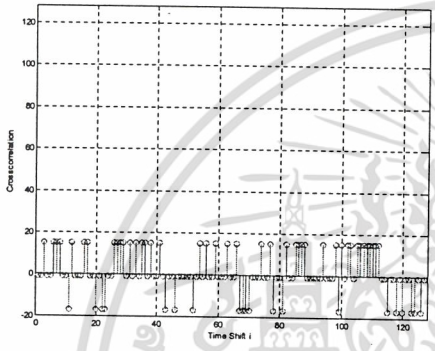
รูปที่ 4.113 แสดงค่าครอสคอรีเลชั่น G1,G46



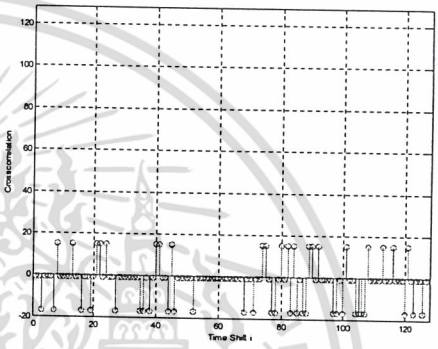
รูปที่ 4.114 แสดงค่าครอสคอรีเลชั่น G1,G47



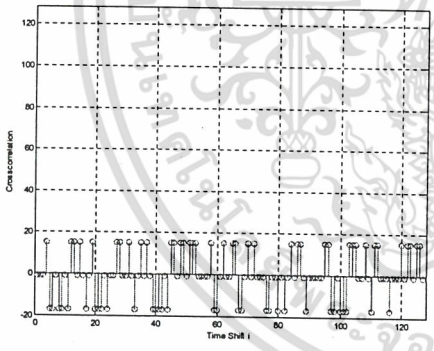
รูปที่ 4.115 แสดงค่าครอสคอรีเลชั่น G1,G48



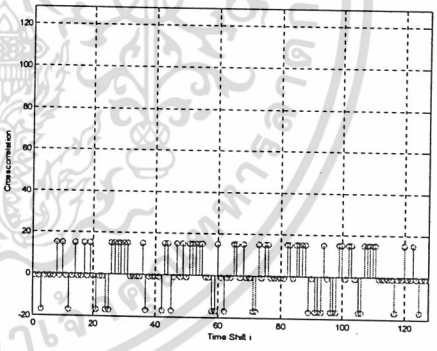
รูปที่ 4.116 แสดงค่าครอสคอรีเลชั่น G1,G49



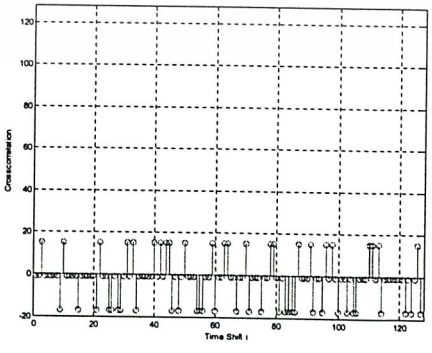
รูปที่ 4.117 แสดงค่าค่าครอสคอรีเลชั่น G1,G50



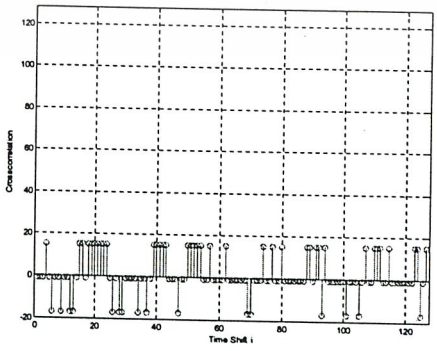
รูปที่ 4.118 แสดงค่าครอสคอรีเลชั่น G1,G51



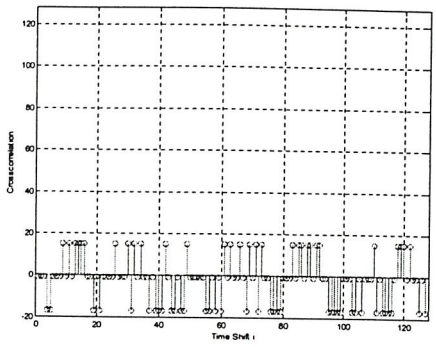
รูปที่ 4.119 แสดงค่าครอสคอรีเลชั่น G1,G52



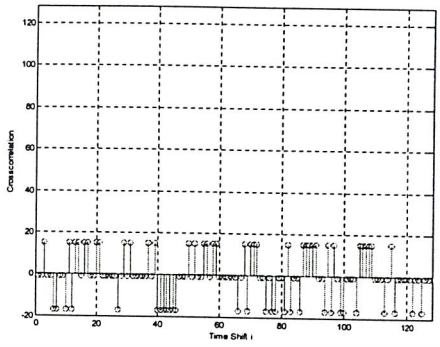
รูปที่ 4.120 แสดงค่าครอสคอรีเลชั่น G1,G54



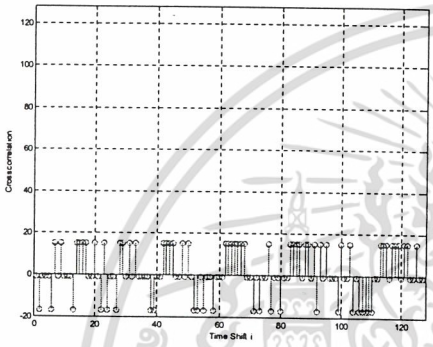
รูปที่ 4.121 แสดงค่าครอสคอรีเลชั่น G1,G54



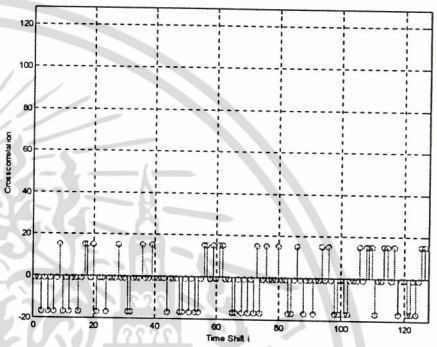
รูปที่ 4.122 แสดงค่าครอสคอรีเลชั่น G1,G55



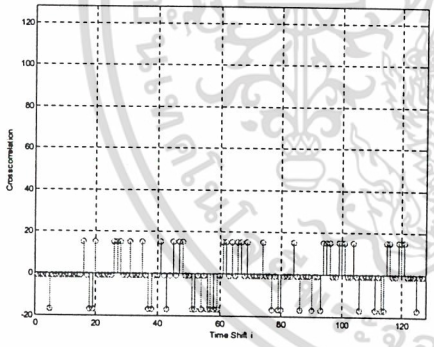
รูปที่ 4.123 แสดงค่าครอสคอรีเลชั่น G1,G56



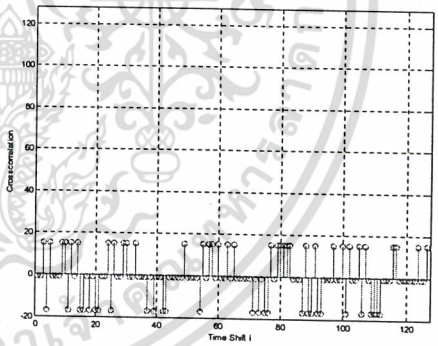
รูปที่ 4.124 แสดงค่าครอสคอรีเลชั่น G1,G57



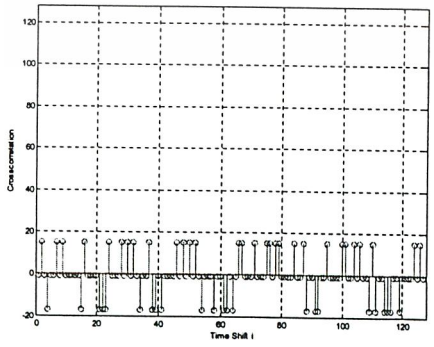
รูปที่ 4.125 แสดงค่าค่าครอสคอรีเลชั่น G1,G58



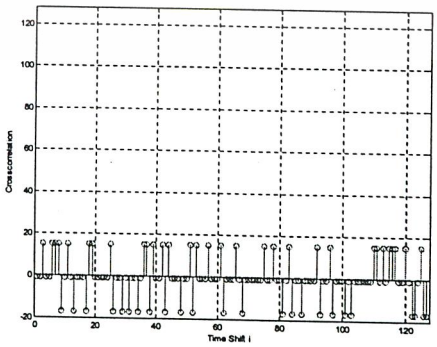
รูปที่ 4.126 แสดงค่าครอสคอรีเลชั่น G1,G59



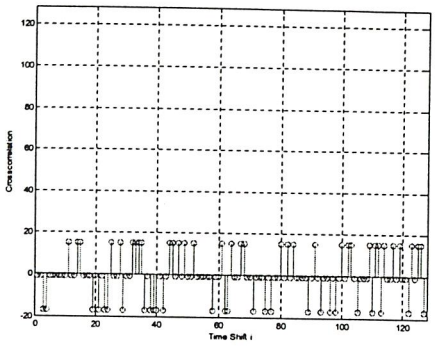
รูปที่ 4.127 แสดงค่าครอสคอรีเลชั่น G1,G60



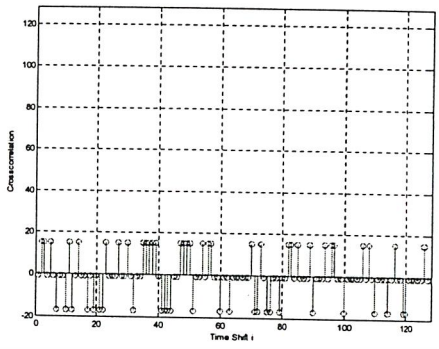
รูปที่ 4.128 แสดงค่าครอสคอรีเลชั่น G1,G61



รูปที่ 4.129 แสดงค่าครอสคอรีเลชั่น G1,G62

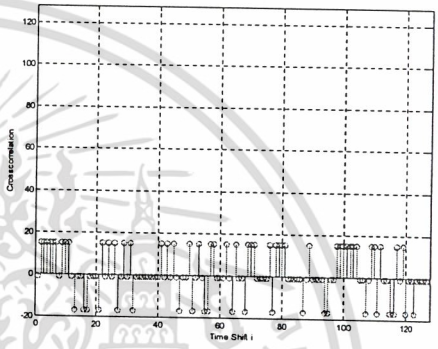


รูปที่ 4.130 แสดงค่าครอสคอรีเลชั่น G1,G63

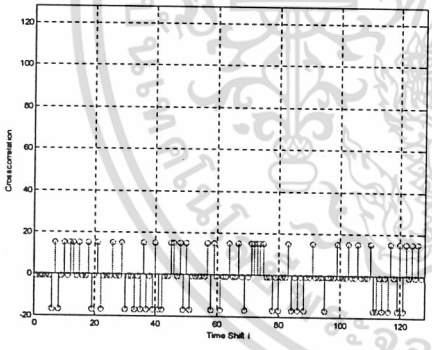


รูปที่ 4.131 แสดงค่าครอสคอรีเลชั่น G1,G64

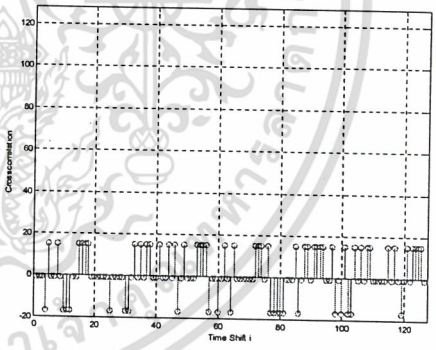
รูปที่ 4.132 แสดงค่าครอสคอรีเลชั่น G1,G65



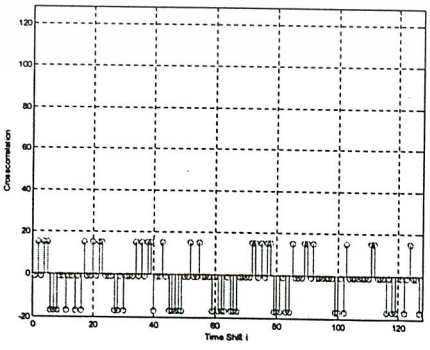
รูปที่ 4.133 แสดงค่าค่าครอสคอรีเลชั่น G1,G66



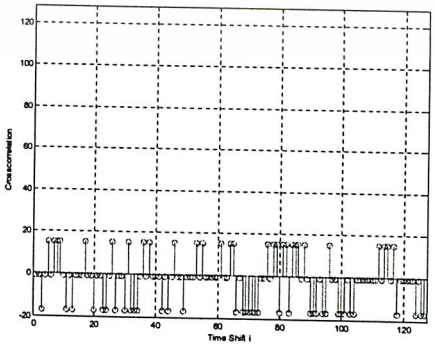
รูปที่ 4.134 แสดงค่าครอสคอรีเลชั่น G1,G67



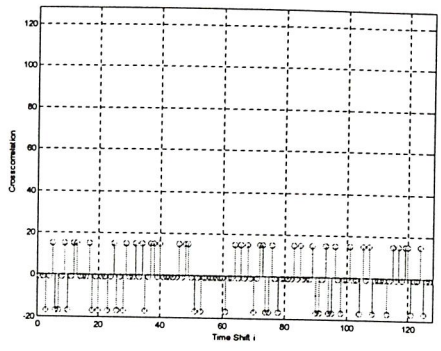
รูปที่ 4.135 แสดงค่าครอสคอรีเลชั่น G1,G68



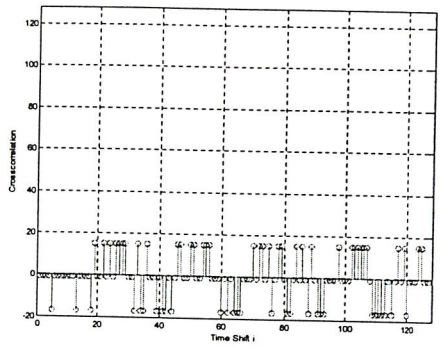
รูปที่ 4.136 แสดงค่าครอสคอรีเลชั่น G1,G69



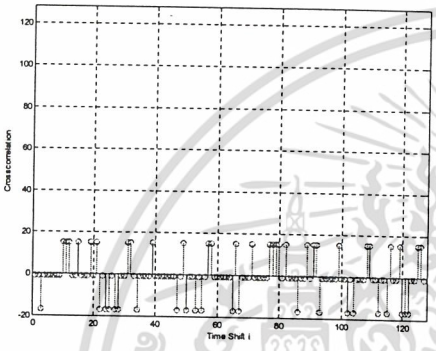
รูปที่ 4.137 แสดงค่าครอสคอรีเลชั่น G1,G70



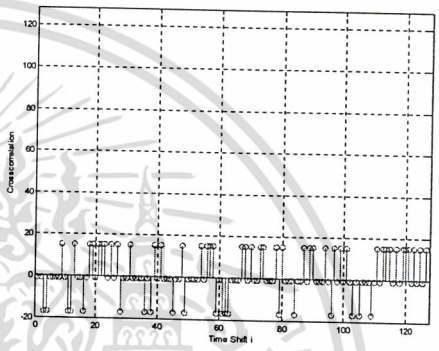
รูปที่ 4.138 แสดงค่าครอสคอรีเลชั่น G1,G71



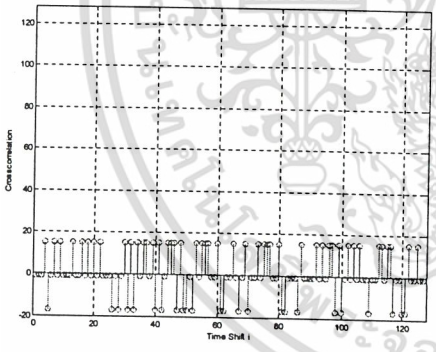
รูปที่ 4.139 แสดงค่าครอสคอรีเลชั่น G1,G72



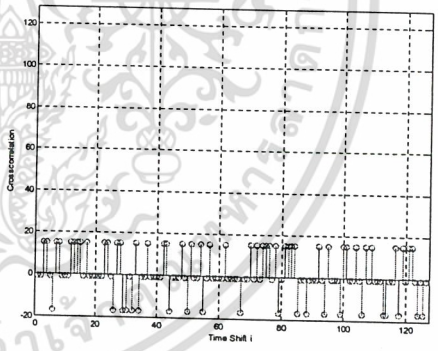
รูปที่ 4.140 แสดงค่าครอสคอรีเลชั่น G1,G73



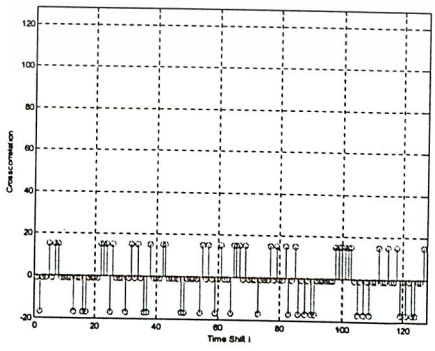
รูปที่ 4.141 แสดงค่าค่าครอสคอรีเลชั่น G1,G74



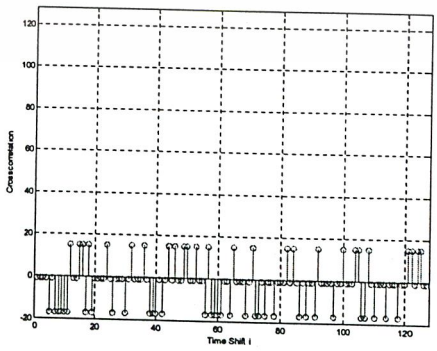
รูปที่ 4.142 แสดงค่าครอสคอรีเลชั่น G1,G75



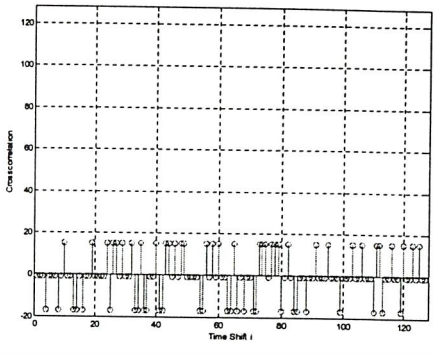
รูปที่ 4.143 แสดงค่าครอสคอรีเลชั่น G1,G76



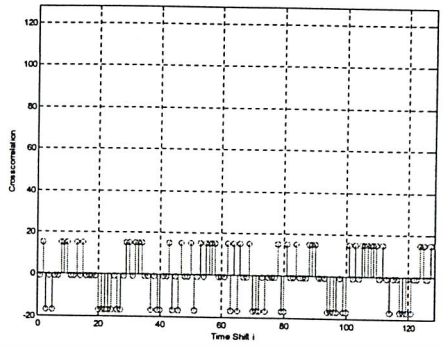
รูปที่ 4.144 แสดงค่าครอสคอรีเลชั่น G1,G77



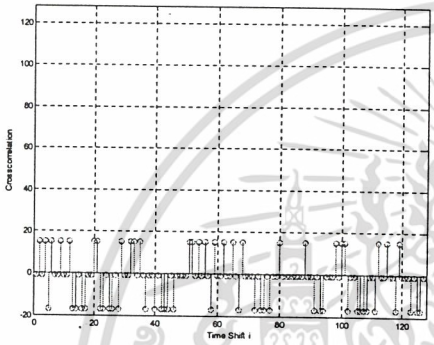
รูปที่ 4.145 แสดงค่าครอสคอรีเลชั่น G1,G78



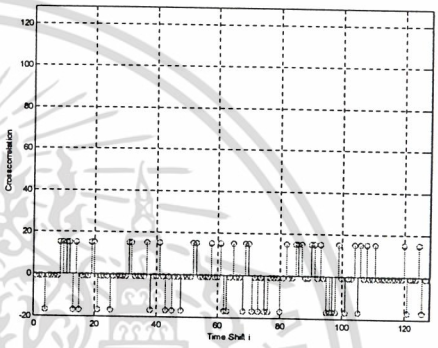
รูปที่ 4.146 แสดงค่าครอสคอรีเลชั่น G1,G79



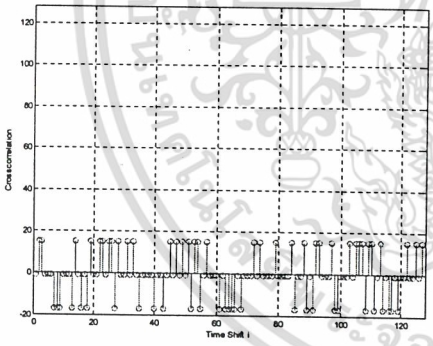
รูปที่ 4.147 แสดงค่าครอสคอรีเลชั่น G1,G80



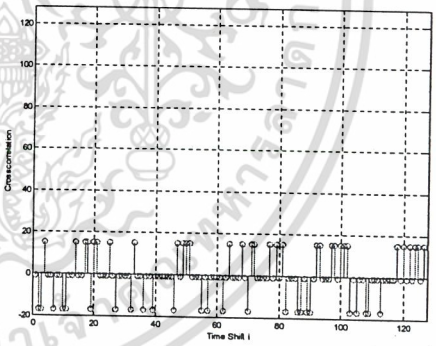
รูปที่ 4.148 แสดงค่าครอสคอรีเลชั่น G1,G81



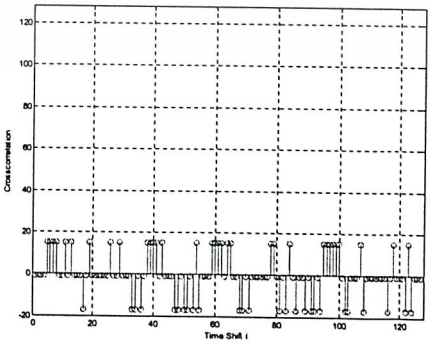
รูปที่ 4.149 แสดงค่าค่าครอสคอรีเลชั่น G1,G82



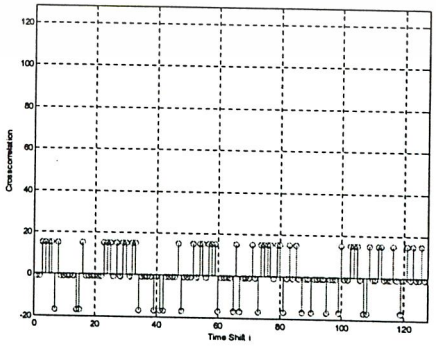
รูปที่ 4.150 แสดงค่าครอสคอรีเลชั่น G1,G83



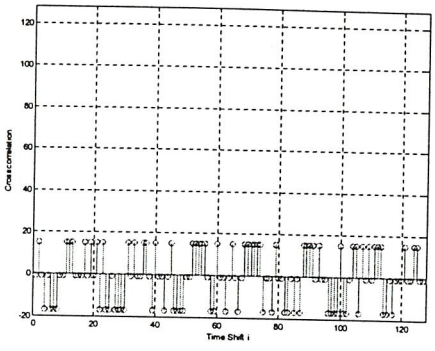
รูปที่ 4.151 แสดงค่าครอสคอรีเลชั่น G1,G84



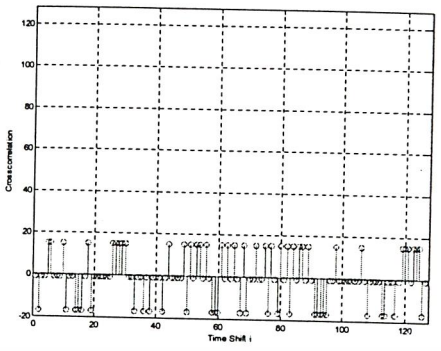
รูปที่ 4.152 แสดงค่าครอสคอรีเลชั่น G1,G85



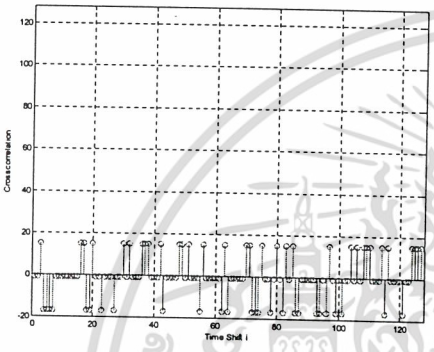
รูปที่ 4.153 แสดงค่าครอสคอรีเลชั่น G1,G86



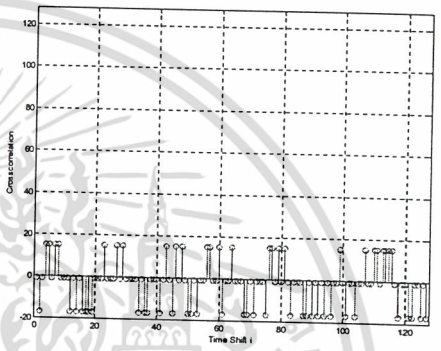
รูปที่ 4.154 แสดงค่าครอสคอรีเลชั่น G1,G87



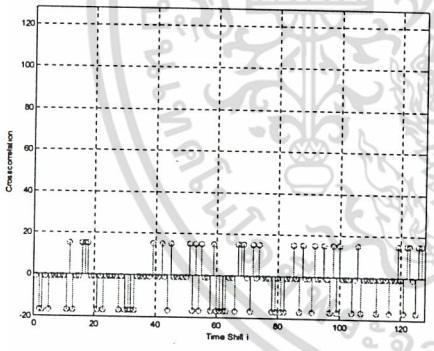
รูปที่ 4.155 แสดงค่าครอสคอรีเลชั่น G1,G88



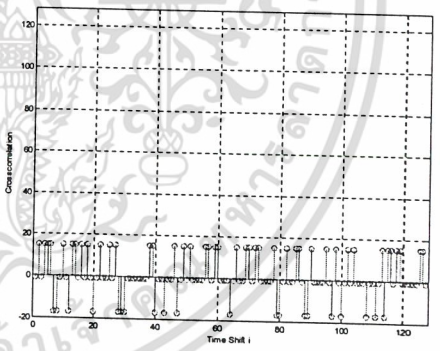
รูปที่ 4.156 แสดงค่าครอสคอรีเลชั่น G1,G89



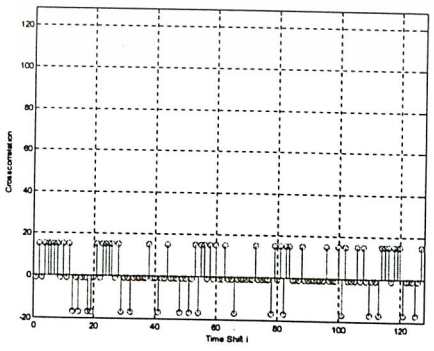
รูปที่ 4.157 แสดงค่าค่าครอสคอรีเลชั่น G1,G90



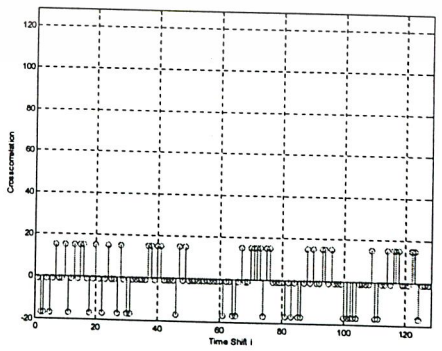
รูปที่ 4.158 แสดงค่าครอสคอรีเลชั่น G1,G91



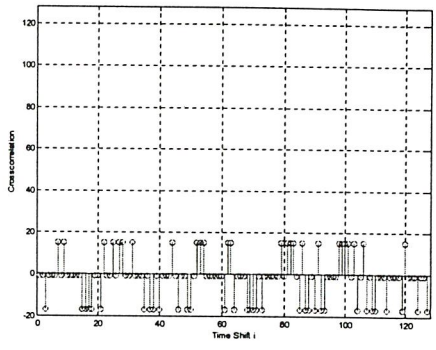
รูปที่ 4.159 แสดงค่าครอสคอรีเลชั่น G1,G92



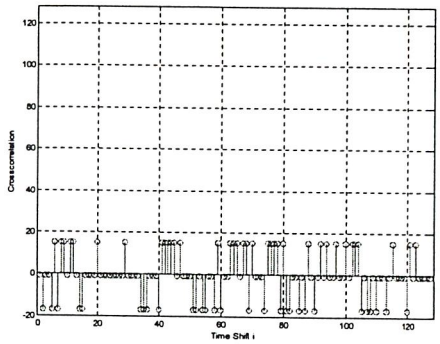
รูปที่ 4.160 แสดงค่าครอสคอรีเลชั่น G1,G93



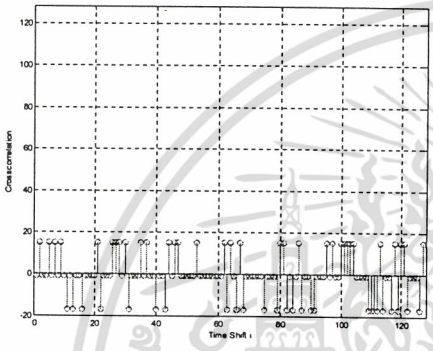
รูปที่ 4.161 แสดงค่าครอสคอรีเลชั่น G1,G94



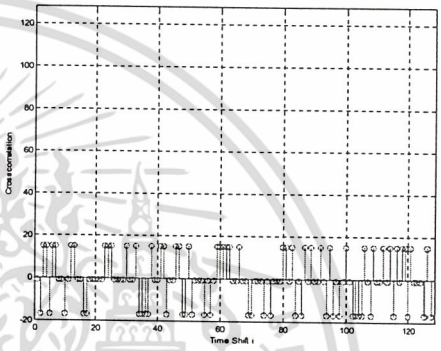
รูปที่ 4.162 แสดงค่าครอสคอรีเลชั่น G1,G95



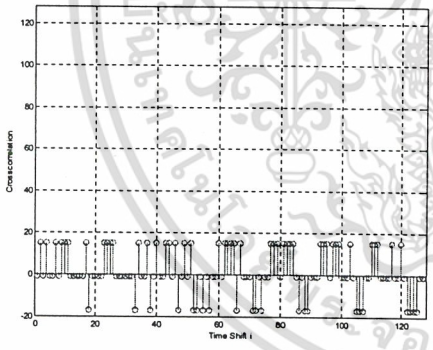
รูปที่ 4.163แสดงค่าครอสคอรีเลชั่น G1,G96



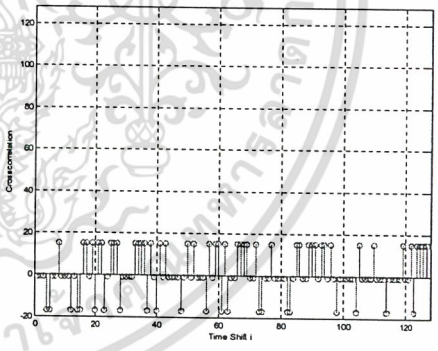
รูปที่ 4.164 แสดงค่าครอสคอรีเลชั่น G1,G97



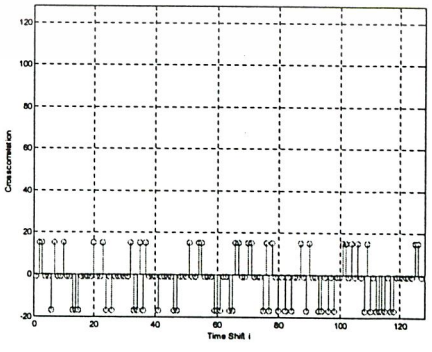
รูปที่ 4.165 แสดงค่าค่าครอสคอรีเลชั่น G1,G98



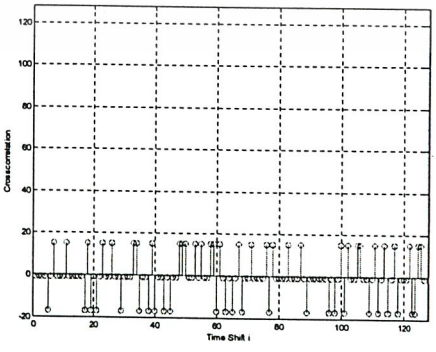
รูปที่ 4.166 แสดงค่าครอสคอรีเลชั่น G1,G99



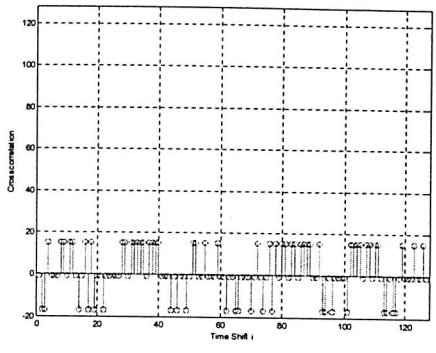
รูปที่ 4.167 แสดงค่าครอสคอรีเลชั่น G1,G100



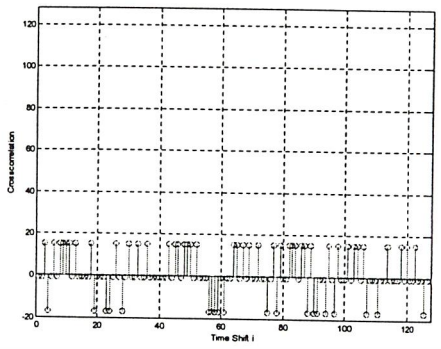
รูปที่ 4.168 แสดงค่าครอสคอรีเลชั่น G1,G101



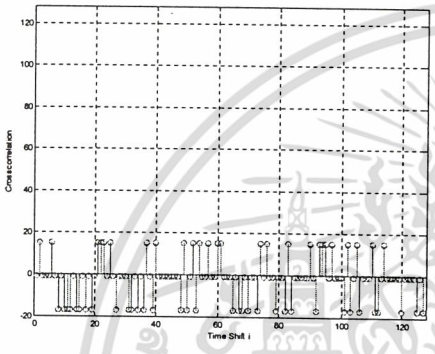
รูปที่ 4.169 แสดงค่าครอสคอรีเลชั่น G1,G102



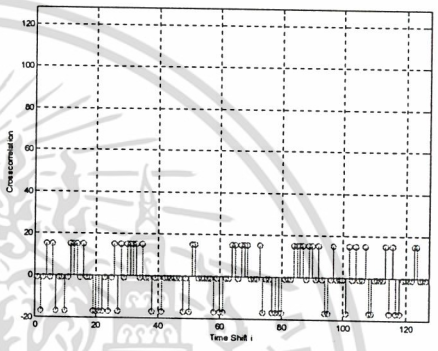
รูปที่ 4.170 แสดงค่าครอสคอรีเลชั่น G1,G103



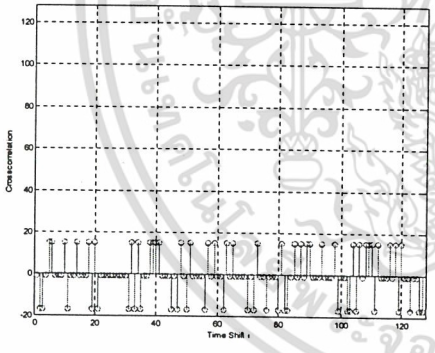
รูปที่ 4.171 แสดงค่าครอสคอรีเลชั่น G1,G104



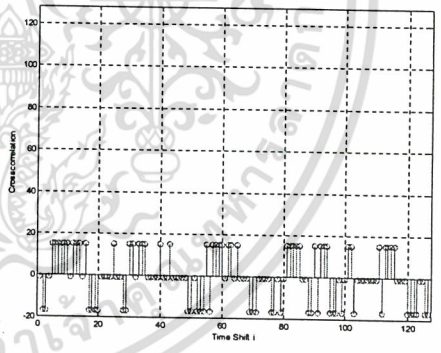
รูปที่ 4.172 แสดงค่าครอสคอรีเลชั่น G1,G105



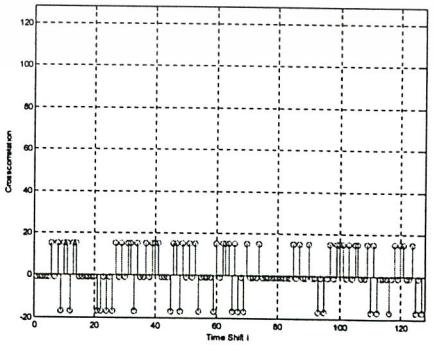
รูปที่ 4.173 แสดงค่าค่าครอสคอรีเลชั่น G1,G106



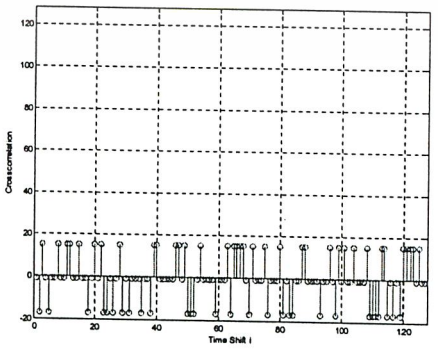
รูปที่ 4.174 แสดงค่าครอสคอรีเลชั่น G1,G107



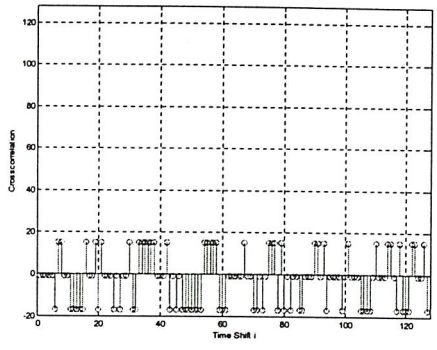
รูปที่ 4.175 แสดงค่าครอสคอรีเลชั่น G1,G108



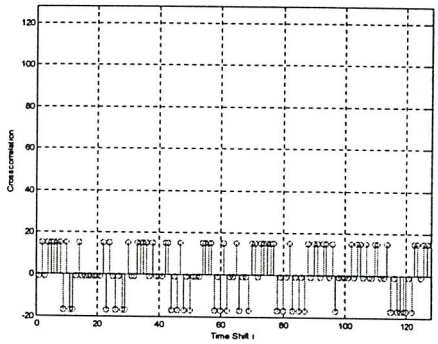
รูปที่ 4.176 แสดงค่าครอสคอรีเลชั่น G1,G109



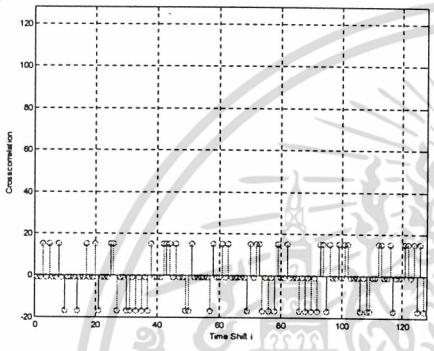
รูปที่ 4.177 แสดงค่าครอสคอรีเลชั่น G1,G110



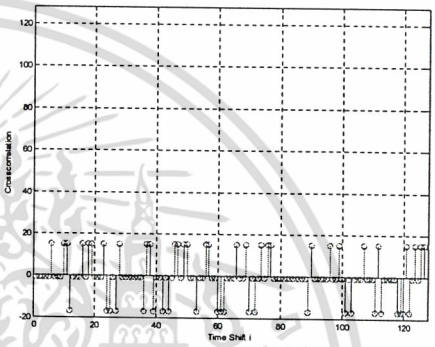
รูปที่ 4.178 แสดงค่าครอสคอรีเลชั่น G1,G111



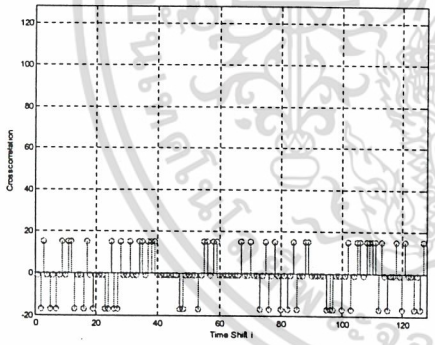
รูปที่ 4.179 แสดงค่าครอสคอรีเลชั่น G1,G112



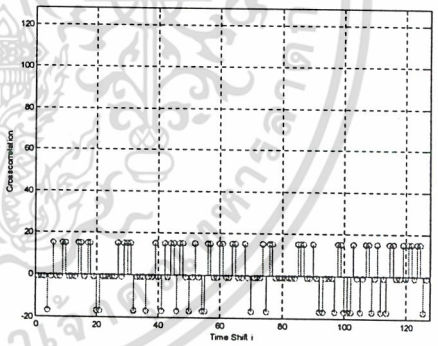
รูปที่ 4.180 แสดงค่าครอสคอรีเลชั่น G1,G113



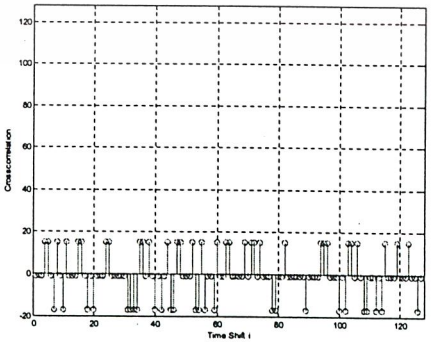
รูปที่ 4.181 แสดงค่าค่าครอสคอรีเลชั่น G1,G114



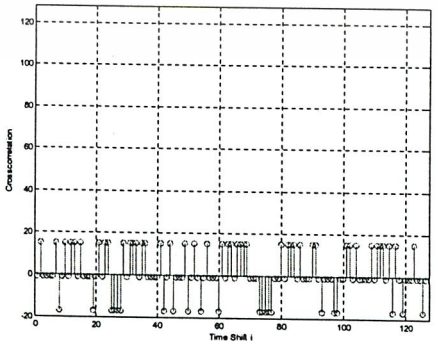
รูปที่ 4.182 แสดงค่าครอสคอรีเลชั่น G1,G115



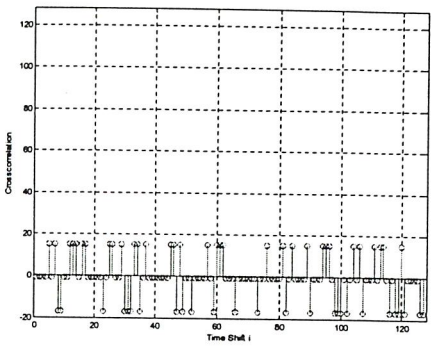
รูปที่ 4.183 แสดงค่าครอสคอรีเลชั่น G1,G116



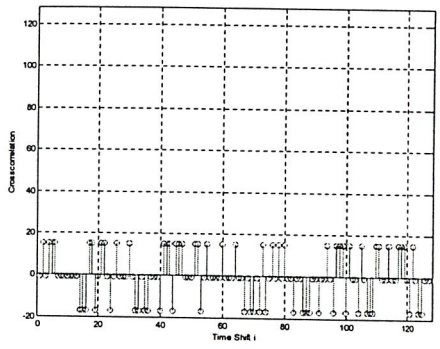
รูปที่ 4.184 แสดงค่าครอสคอรีเลชั่น G1,G117



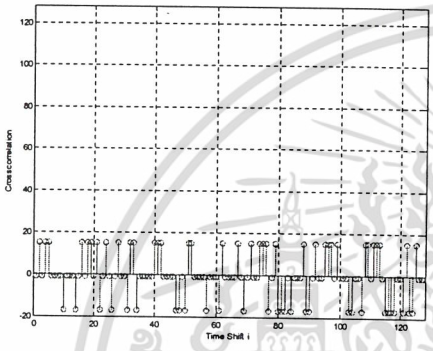
รูปที่ 4.185 แสดงค่าครอสคอรีเลชั่น G1,G118



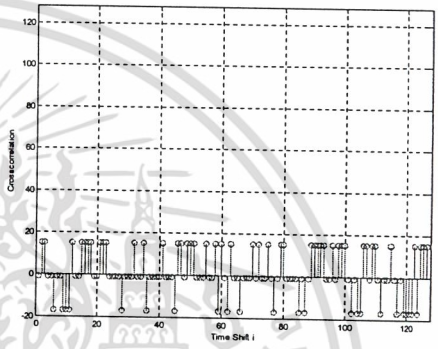
รูปที่ 4.186 แสดงค่าครอสคอรีเลชั่น G1,G119



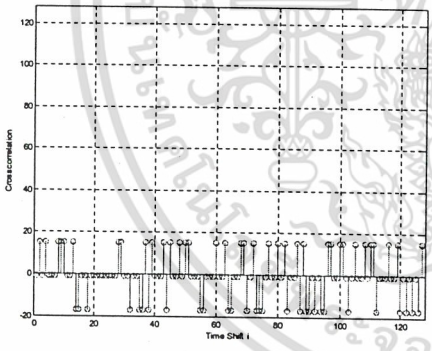
รูปที่ 4.187 แสดงค่าครอสคอรีเลชั่น G1,G120



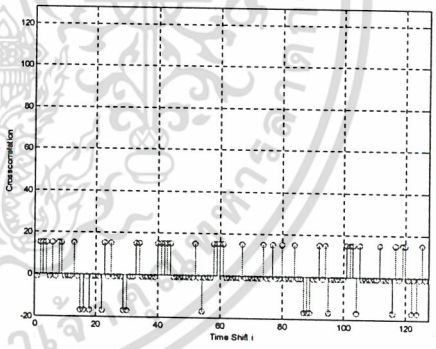
รูปที่ 4.188 แสดงค่าครอสคอรีเลชั่น G1,G121



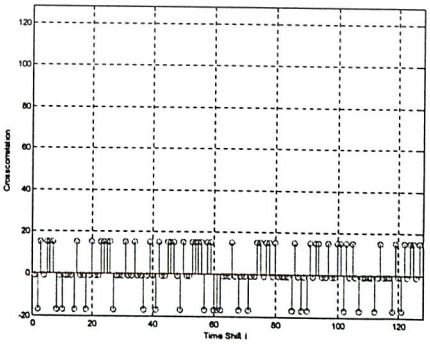
รูปที่ 4.189 แสดงค่าค่าครอสคอรีเลชั่น G1,G122



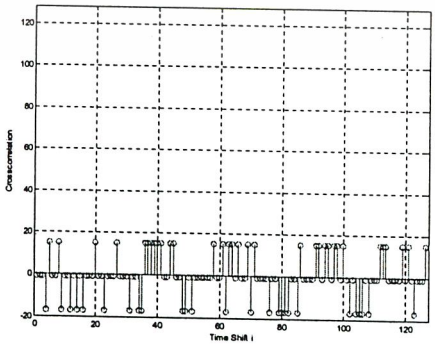
รูปที่ 4.190 แสดงค่าครอสคอรีเลชั่น G1,G123



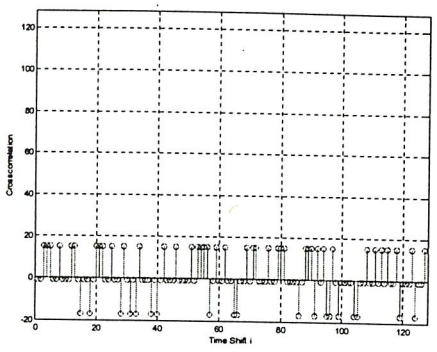
รูปที่ 4.191 แสดงค่าครอสคอรีเลชั่น G1,G124



รูปที่ 4.192 แสดงค่าครอสคอรีเลชั่น G1,G125

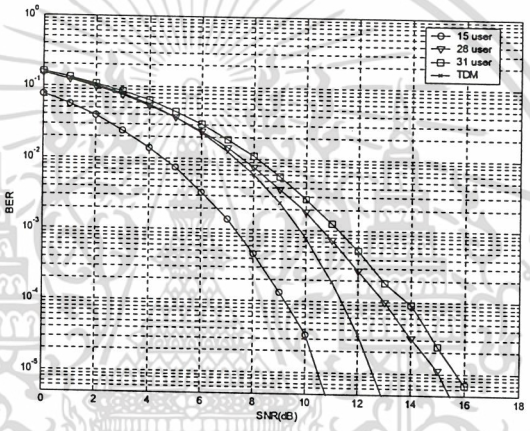


รูปที่ 4.193 แสดงค่าครอสคอรีเลชั่น G1,G126



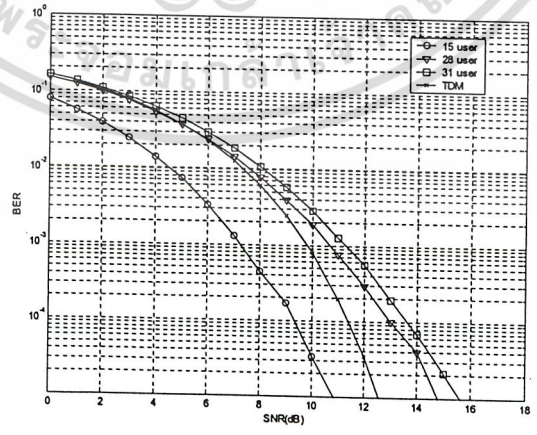
รูปที่ 4.194 แสดงค่าครอสคอรีเลชั่น G1,G127

4.1.11 ผลการพล็อต BER/SNR ของ PN code (m-sequence) 31 modulo



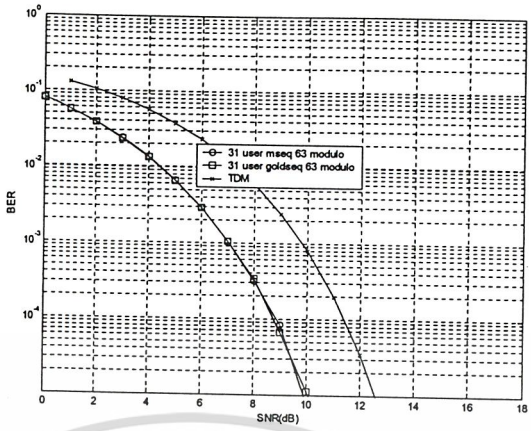
รูปที่ 4.195 BER/SNR ของ mseq เทียบกับ ระบบเดิม

4.1.12 ผลการพล็อต BER/SNR ของ PN code (gold-sequence) 31 modulo



รูปที่ 4.196 BER/SNR ของ gold เทียบกับ ระบบเดิม

4.1.13 ผลการพล็อต BER/SNR ของ PN code (gold-sequence) 63 modulo



รูปที่ 4.197 BER/SNR ของ m, gold เทียบกับ ระบบเดิม

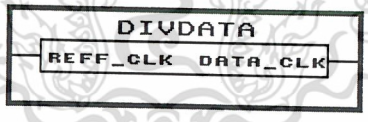
4.2 ผลการทดลองที่ได้จากโปรแกรม MAX+plus II

4.2.1 ผลการทดลองที่ได้จากโปรแกรม MAX+plus II ด้านส่ง

4.2.1.1 วงจรกำเนิดสัญญาณนาฬิกาของข้อมูล (Data)

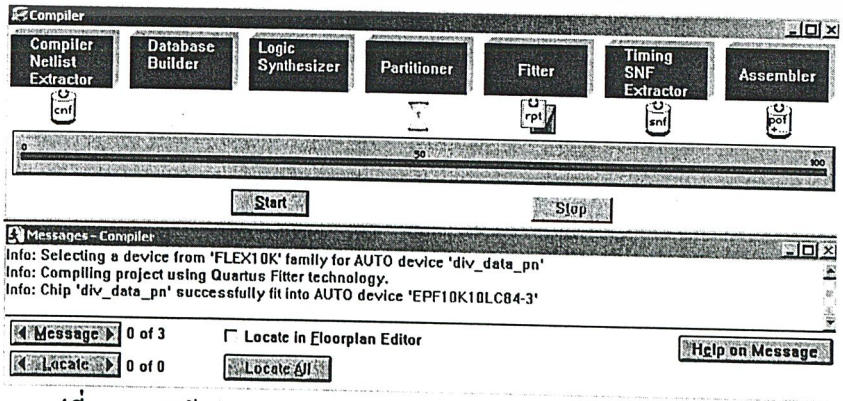
เป็นวงจรที่ใช้ในการสร้างสัญญาณนาฬิกาของ ข้อมูล โดยการนำสัญญาณนาฬิกาของวงจรหลักเข้าผ่านวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล และวงจรนี้จะทำการหารความถี่ของสัญญาณนาฬิกาหลักด้วย 62 ทำให้เกิดสัญญาณนาฬิกาของ ข้อมูล ขึ้น โดยมีขั้นตอนการทดลองดังนี้

1. เปิดไฟล์ชื่อ diviข้อมูล.gdf ที่ได้จากการสร้างขึ้นมาด้วยภาษา VHDL ที่ทำการสังเคราะห์ด้วยโปรแกรม max+plus-2 เป็นบล็อกไดอะแกรมดังรูป



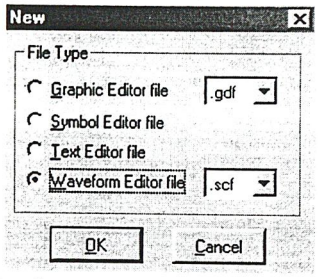
รูปที่ 4.198 บล็อกไดอะแกรมของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

2. ทำการ set project to current file โดยการกด Ctrl+Shift+J จากนั้นทำการ compileโดยการกด Ctrl+L จะได้ command window ดังรูป จะพบว่าไม่มีข้อผิดพลาดเกิดขึ้น



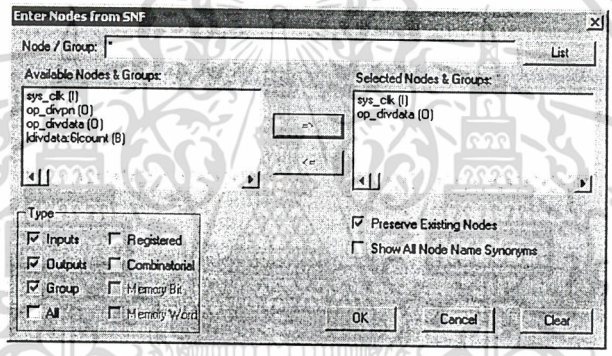
รูปที่ 4.199 หน้าต่างcompiler ของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

3. ทำการสร้าง *.scf file โดยการกด File > New จะได้ดังรูป



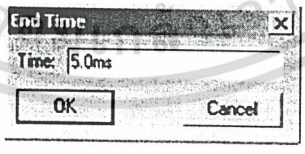
รูปที่ 4.200 หน้าต่างเลือกไฟล์ *.scf file ของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

4. ทำการ add node โดยการเลือกคำสั่ง Node > Enter Nodes fron SNF โดยเลือก add node sys_clock และ op_divข้อมูล จากการกดปุ่ม list ดังรูป



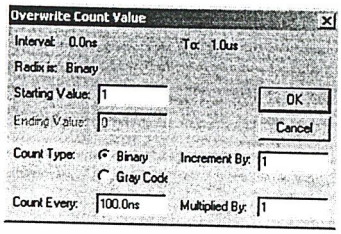
รูปที่ 4.201 หน้าต่างแสดงการ add node ของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

5. ทำการ set ค่า end time โดยการเลือกที่ คำสั่ง File > End time.. ให้เป็น 5.0 ms



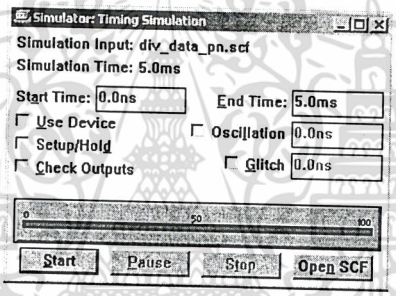
รูปที่ 4.202 หน้าต่างการเลือกค่า end time ของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

6. ทำการกำหนดค่า clk โดยคลิกเมาส์ไปที่ clk จะเกิดแถบดำขึ้นมาจากนั้นก็กำหนดค่า clk โดยกด  ในแถบทางด้านซ้ายมือโดยกำหนดค่า ดังรูป



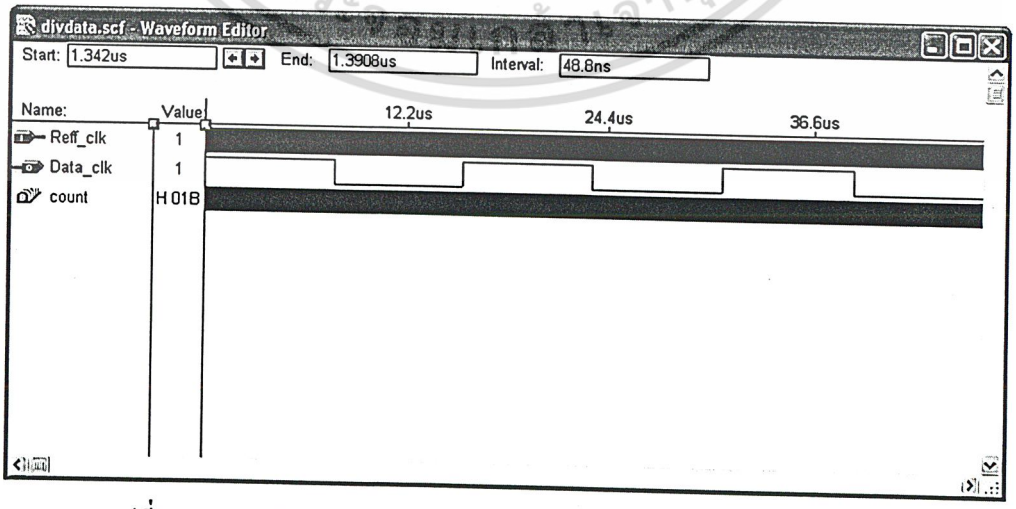
รูปที่ 4.203 หน้าต่างกำหนดค่าสัญญาณอินพุทของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

7. กลับมาที่หน้าต่าง graphic editor ทำการ save and simulate โดยการกด Ctrl+Shift+L โดยถ้าไม่มีข้อผิดพลาดจะได้ command window ดังรูป



รูปที่ 4.204 หน้าต่าง Simulator ของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

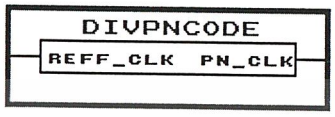
8. ทำการเปิด scf file โดยการกดไปที่ปุ่ม open scf (จากรูปข้อ 7) โดยรูปที่ 4.8 นี้เป็นการวัดเทียบกันระหว่าง sys_clk, op_divข้อมูล และ op_divpn



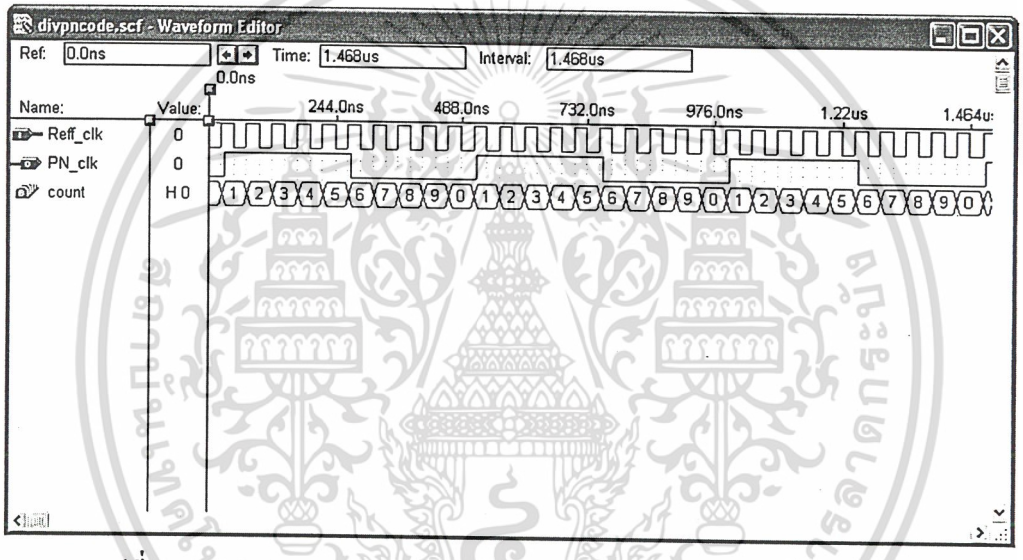
รูปที่ 4.205 การจำลองการทำงานของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล

4.2.1.2 วงจรกำเนิดสัญญาณนาฬิกาของ PN code

เป็นวงจรที่ใช้ในการสร้างสัญญาณนาฬิกาของ PN code โดยการนำสัญญาณนาฬิกาของวงจรหลักเข้าผ่านวงจรกำเนิดสัญญาณนาฬิกาของ PN code และวงจรนี้จะทำการหารความถี่ของสัญญาณนาฬิกาหลักด้วย 62*31 (เพราะว่าใน1บิตของข้อมูลจะมีcode31ตัว) ทำให้เกิดสัญญาณนาฬิกาของ PN code ขึ้น



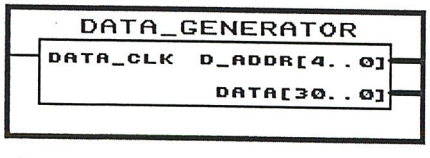
รูปที่ 4.206 บล็อกไดอะแกรมของวงจรวงจรกำเนิดสัญญาณนาฬิกาของ PN code



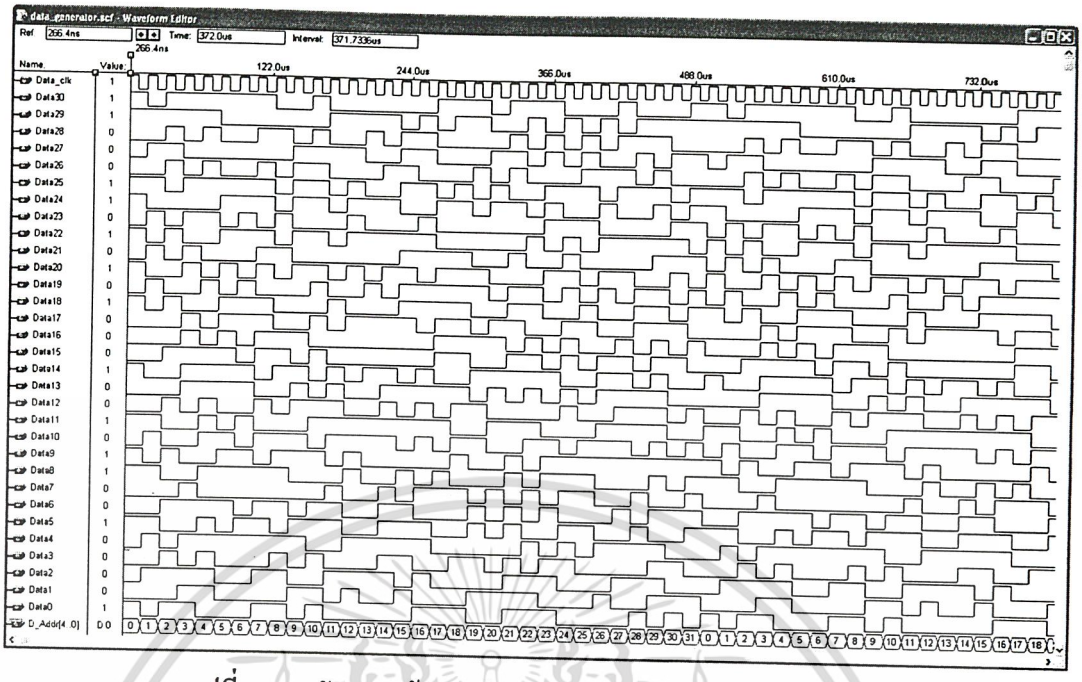
รูปที่ 4.207 การจำลองการทำงานของวงจรกำเนิดสัญญาณนาฬิกาของ PN code

4.2.1.3 วงจรกำเนิดสัญญาณข้อมูล

เป็นวงจรที่ใช้ในการสร้างสัญญาณ ข้อมูล โดยที่ฝั่งอินพุทของวงจรกำเนิดสัญญาณข้อมูล จะเป็นสัญญาณเอาท์พุทของวงจรกำเนิดสัญญาณนาฬิกาของ ข้อมูล ส่วนทางด้านเอาท์พุทของวงจรจะเป็นสัญญาณข้อมูล



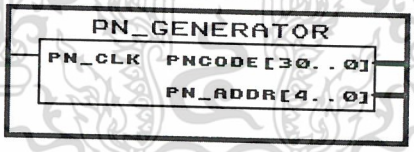
รูปที่ 4.208 บล็อกไดอะแกรมของวงจรวงจรกำเนิดสัญญาณ ข้อมูล



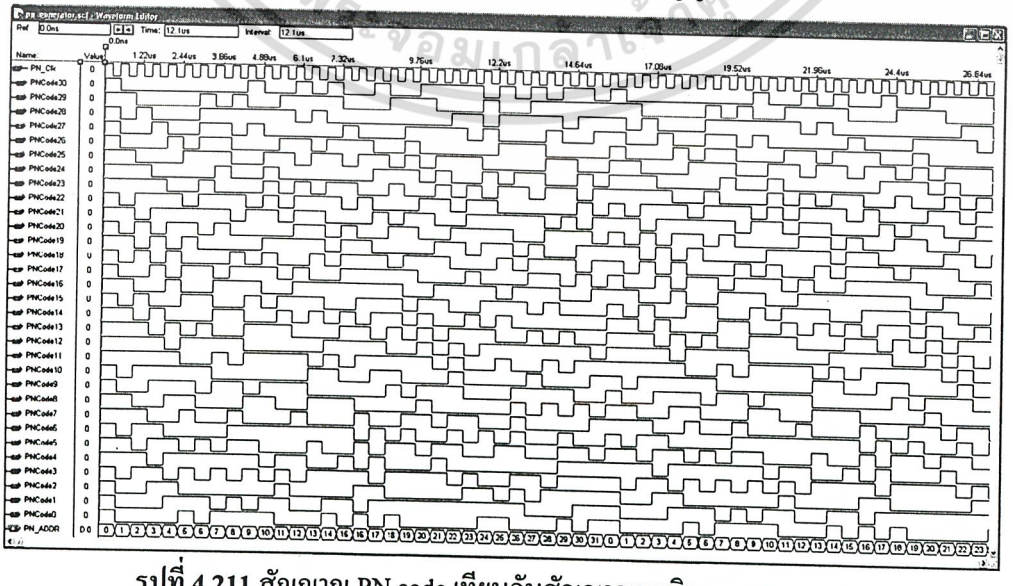
รูปที่ 4.209 สัญญาณข้อมูลเทียบกับสัญญาณนาฬิกาของข้อมูล

4.2.1.4 วงจรกำเนิดสัญญาณ PN code

เป็นวงจรที่ใช้ในการสร้างสัญญาณ PN code โดยที่ ฝั่งอินพุทของวงจรกำเนิดสัญญาณ PN code จะเป็นสัญญาณเอาท์พุทของวงจรกำเนิดสัญญาณนาฬิกาของ PN code ส่วนทางด้านเอาท์พุทของวงจรจะเป็นสัญญาณ PN code



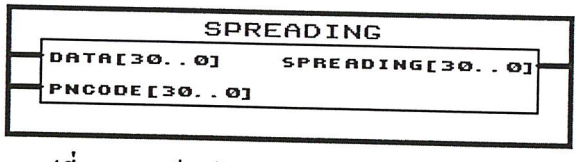
รูปที่ 4.210 บล็อกไดอะแกรมของวงจรกำเนิดสัญญาณ PN code



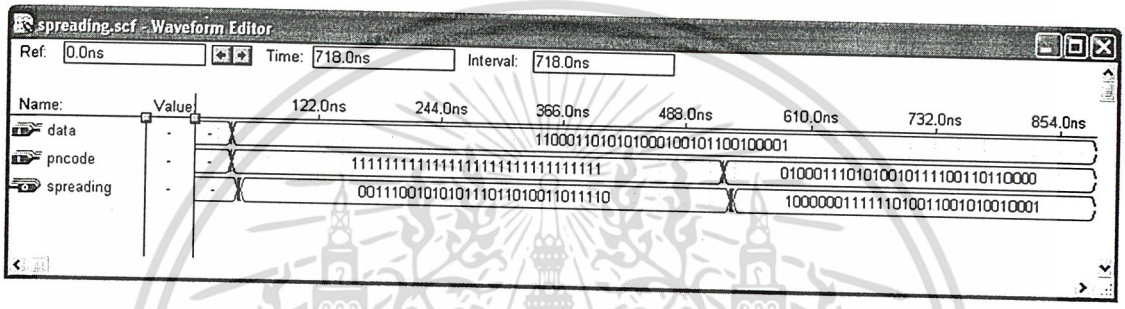
รูปที่ 4.211 สัญญาณ PN code เทียบกับสัญญาณนาฬิกาของ PN code

4.2.1.5 วงจร Spreading

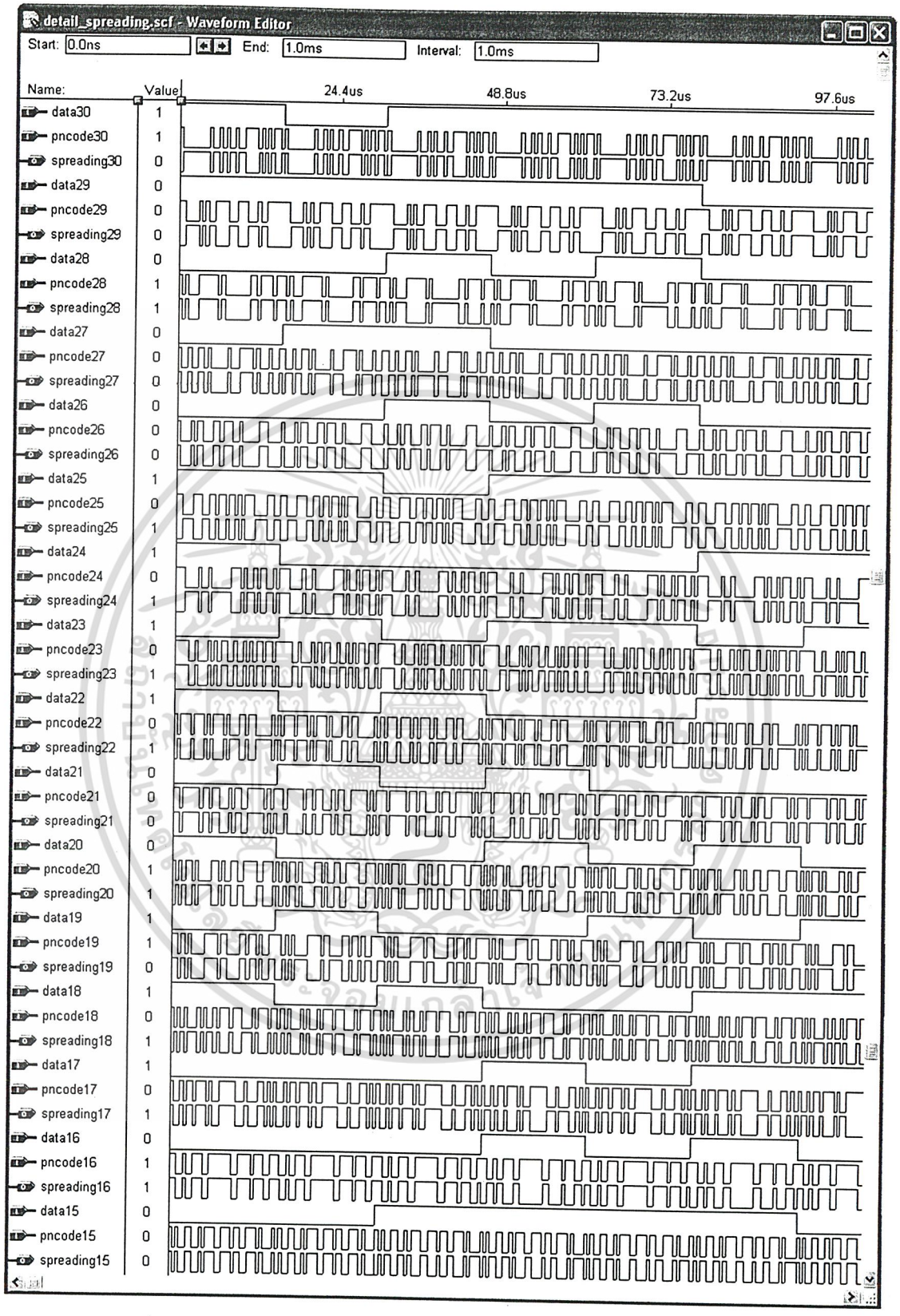
เป็นวงจรที่ทำการนำสัญญาณข้อมูลของ 31 แหล่งข้อมูลมาทำการบวกกันแบบ 2 – modulo กับสัญญาณ PN code 31 ชุดในอัตราส่วน 1/31 จะได้ข้อมูลออกมาเป็นสัญญาณที่ถูกทำการสเพรดออกมา 31 ช่องสัญญาณ



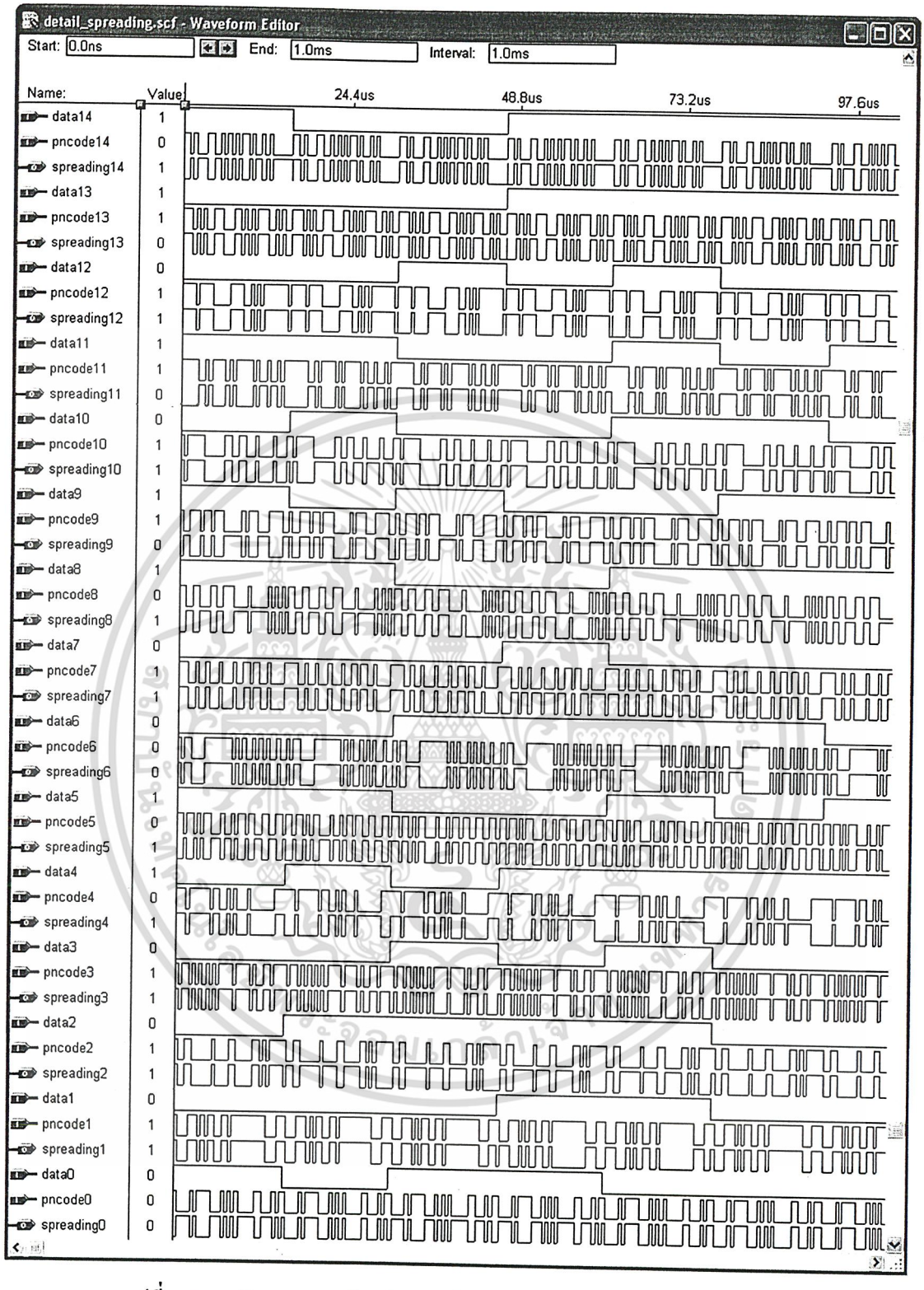
รูปที่ 4.212 บล็อกไดอะแกรมของวงจร Spreading



รูปที่ 4.213a สัญญาณทางด้านอินพุทเทียบกับเอาท์พุทของวงจร Spreading



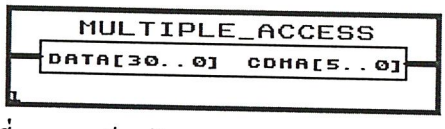
รูปที่ 4.213b สัญญาณทางด้านอินพุทเทียบกับเอาท์พุทของวงจร Spreading



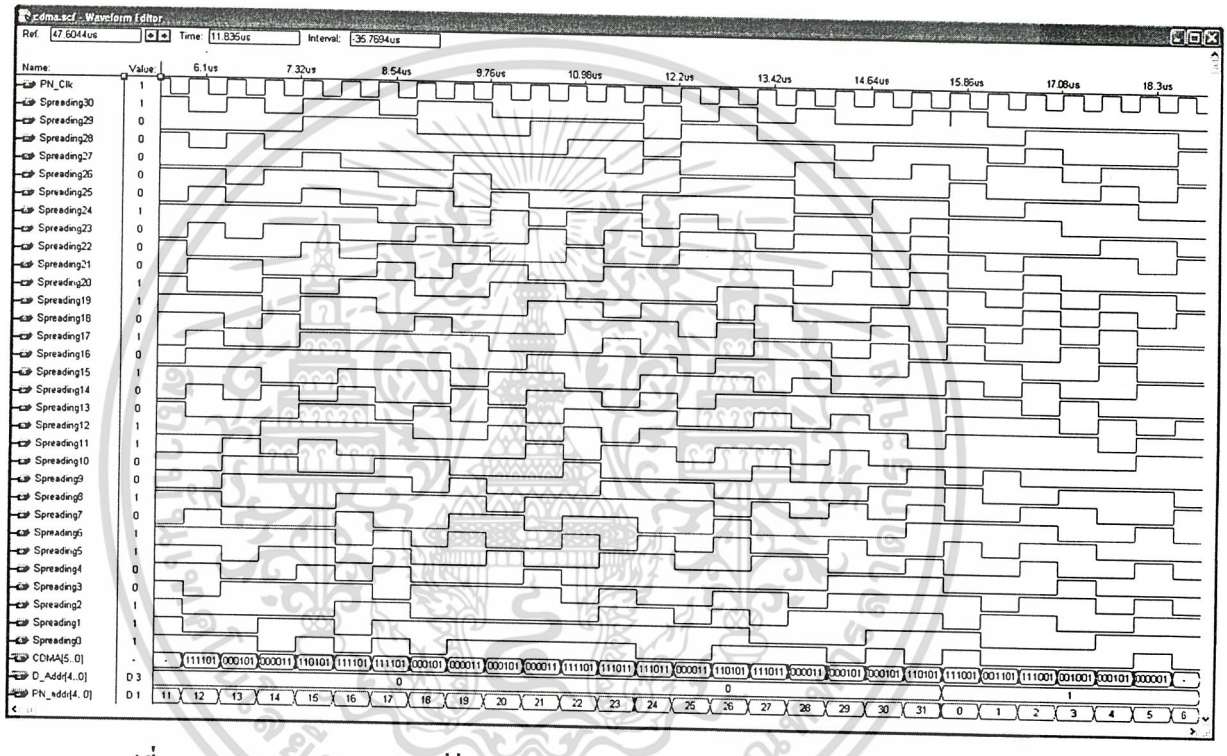
รูปที่ 4.213c สัญญาณทางด้านอินพุทเทียบกับเอาท์พุทของวงจร Spreading

4.2.1.6 วงจร Multiple Access

เป็นวงจรที่นำค่าของสัญญาเอาท์พุทที่ได้จากวงจร Spreading มาทำการเปรียบเทียบค่า โดยที่ให้ค่าโลจิก 0 ให้มีค่าเสมือนเท่ากับ 1 และค่าโลจิก 1 ให้มีค่าเสมือนเป็น 0 แล้วนำสัญญาณ 31 ช่องสัญญาณมาทำการบวกกันแล้วทำการแปลงให้เป็นเลขไบนารีแบบ 2's compliment ให้มีค่าข้อมูล 5 บิต และ บิตเครื่องหมาย 1 บิต



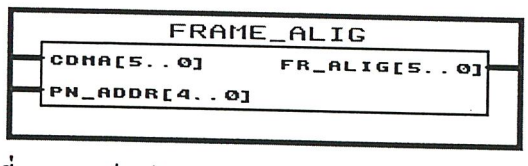
รูปที่ 4.214 บล็อกไดอะแกรมของวงจร Spreading



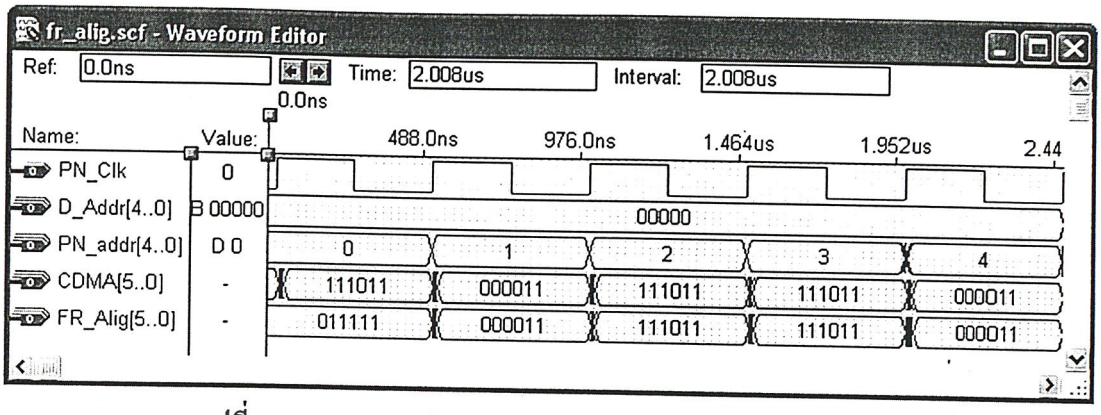
รูปที่ 4.215 สัญญาณSpreading ที่ฝั่งอินพุทเทียบกับสัญญาณ CDMA ที่ฝั่งเอาท์พุท

4.2.1.7 วงจร Frame Alignment

เป็นวงจรที่ทำการใส่ค่า FAW ไปที่หัวเฟรมเพื่อเป็นตัวบอกว่าจุดไหนที่เป็นจุดเริ่มต้นของสัญญาณข้อมูล



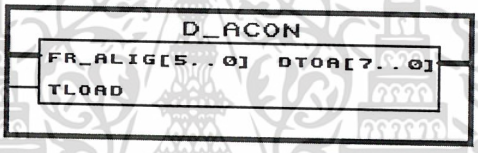
รูปที่ 4.216 บล็อกไดอะแกรมของวงจร Frame Alignment



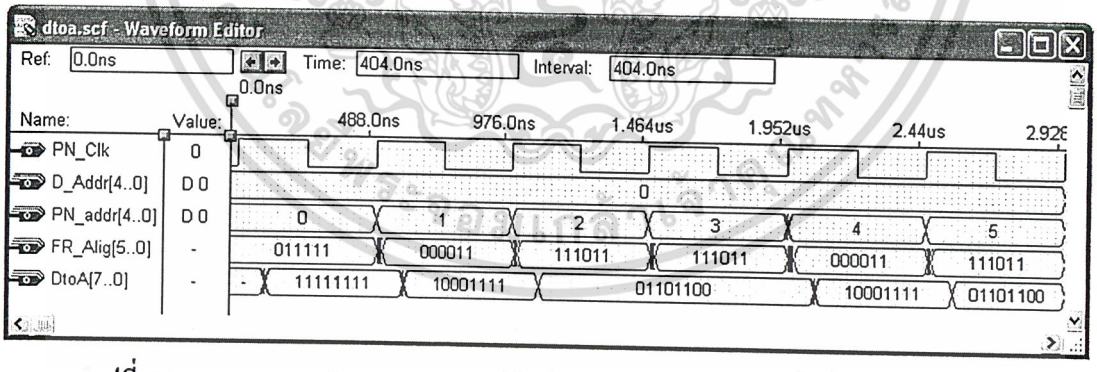
รูปที่ 4.217 แสดงการทำงานของวงจร **Frame Alignment**

4.2.1.8 วงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter

โดยวงจรนี้จะมีหน้าที่ทำการแปลงค่าสัญญาณ digital ที่มีเป็นค่า analog เพื่อเตรียมพร้อมก่อนที่จะไปเข้าวงจร DtoA converter ในส่วนที่เป็นอุปกรณ์จริงอีกที



รูปที่ 4.218 บล็อกไดอะแกรมของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter

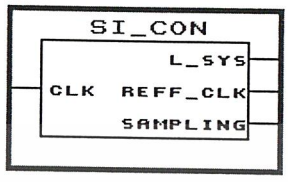


รูปที่ 4.219 แสดงการทำงานของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter

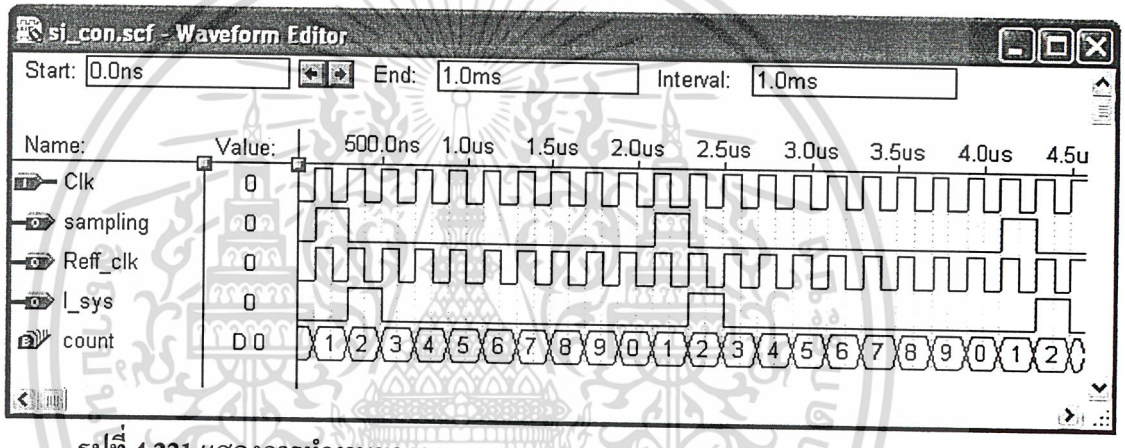
4.2.2 ผลการทดลองที่ได้จากโปรแกรม MAX+plus II ด้านรับ

4.2.2.1 วงจรควบคุมการ sampling และ load_system

วงจรนี้จะมีหน้าที่ในการกำหนดให้เกิดการ sampling ของสัญญาณขึ้นส่วน load_system จะมีหน้าที่ในการloadค่าที่ได้จากการ sampling ไปคำนวณ

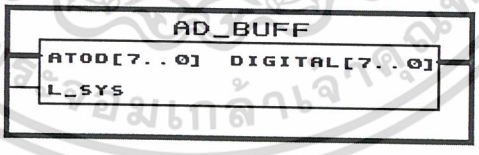


รูปที่ 4.220 บล็อกไดอะแกรมของวงจรควบคุมการ sampling และ load_system

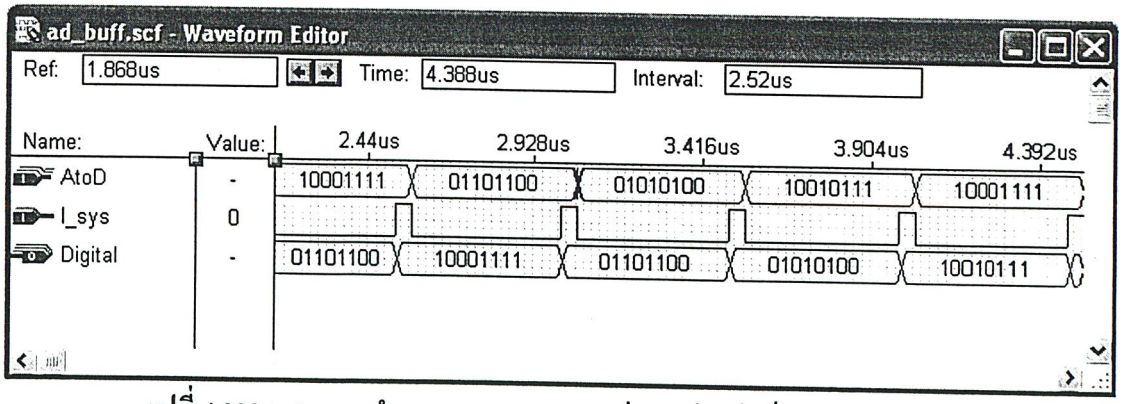


รูปที่ 4.221 แสดงการทำงานของวงจรวงจรควบคุมการ sampling และ load_system

4.2.2.2 วงจร buffer ก่อนที่จะทำการแปลงค่าสัญญาณกลับไปเป็น digital

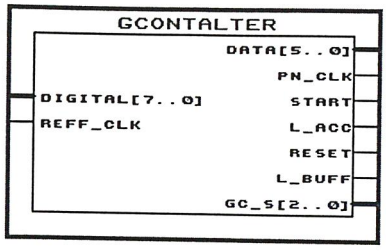


รูปที่ 4.222 บล็อกไดอะแกรมของวงจร buffer ก่อนที่จะทำการแปลงค่าสัญญาณกลับไปเป็น digital

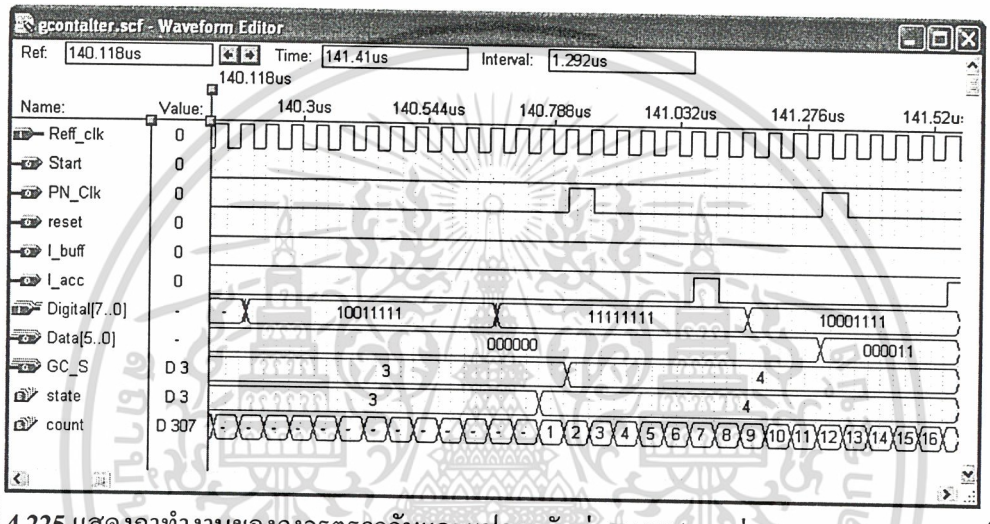


รูปที่ 4.223 แสดงการทำงานของวงจร buffer ก่อนแปลงค่าเป็น digital

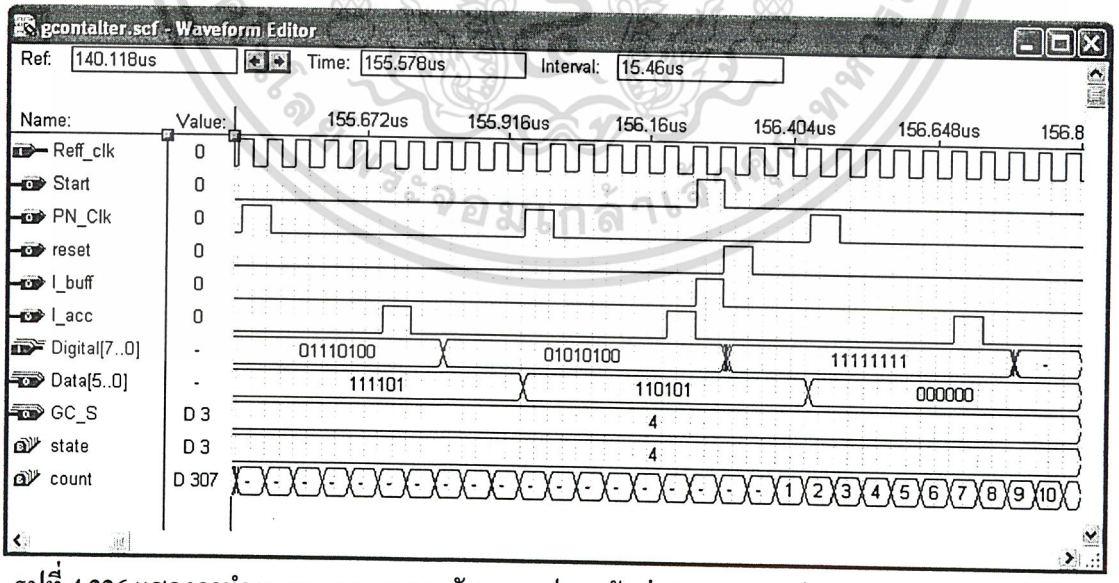
4.2.2.3 วงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)



รูปที่ 4.224 บล็อกไดอะแกรมของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)

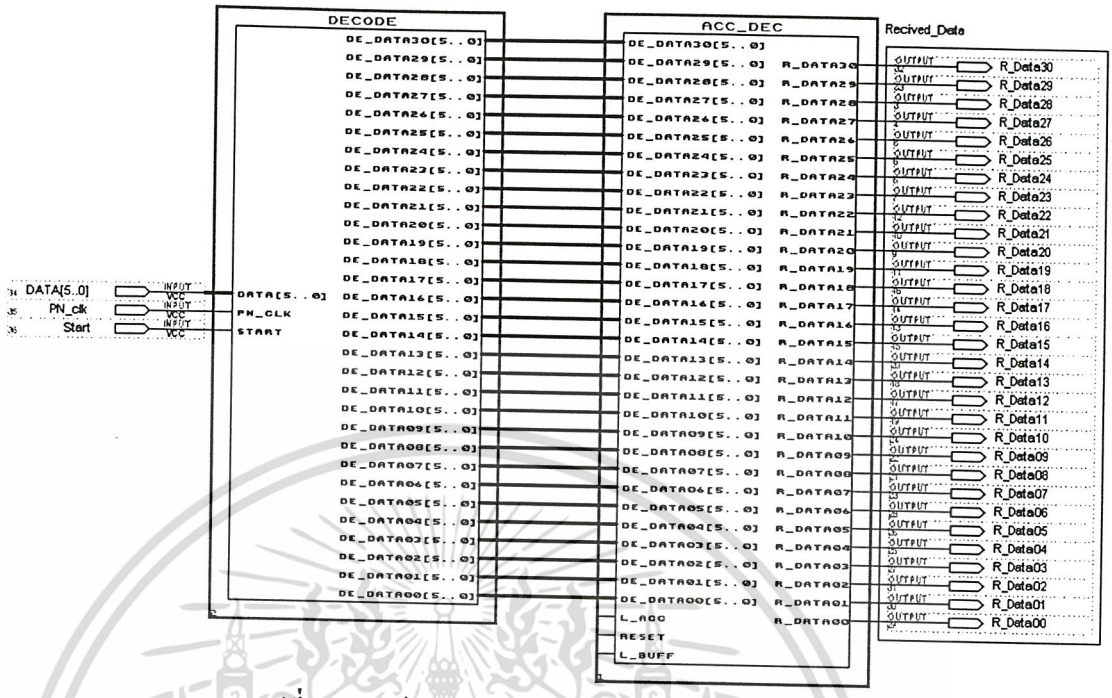


รูปที่ 4.225 แสดงการทำงานของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)



รูปที่ 4.226 แสดงการทำงานของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)

4.2.2.4 วงจร Decode สัญญาณ

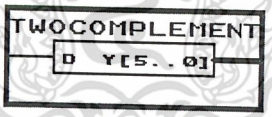


รูปที่ 4.227 บล็อกไดอะแกรมของวงจร Decode

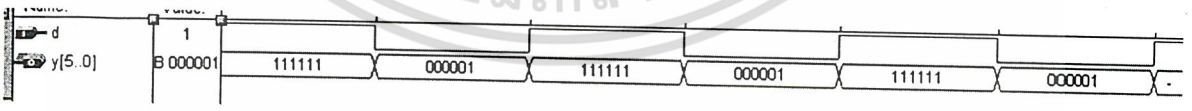
ซึ่งมีส่วนประกอบย่อยภายในวงจร Decode สัญญาณ คือ

4.2.2.4.1 วงจร 2's complement

เป็นวงจรที่ทำหน้าที่เปรียบเทียบค่าจากโลจิก 1 เป็นบิต -1 และปรียบเทียบค่าจากโลจิก 0 เป็นบิต 1 ในเลขฐาน 2



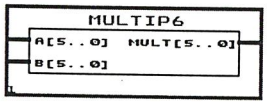
รูปที่ 4.228 บล็อกไดอะแกรมของวงจร 2's complement



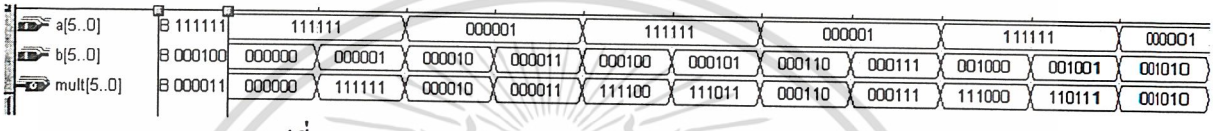
รูปที่ 4.229 สัญญาณก่อนและหลังผ่านวงจร 2's complement

4.2.2.4.2 วงจรคูณ

วงจร multi 6 (วงจรคูณ) เป็นการนำสัญญาณที่ได้จากวงจร 2's complement ช่องสัญญาณละ 6 บิตมาคูณกับสัญญาณส่วนที่เป็น multiple accessที่มีขนาด 6 บิตเช่นกันแล้วให้ผลลัพธ์ออกมามีขนาด 6 บิต



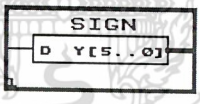
รูปที่ 4.230 บล็อกไดอะแกรมของวงจรคูณ



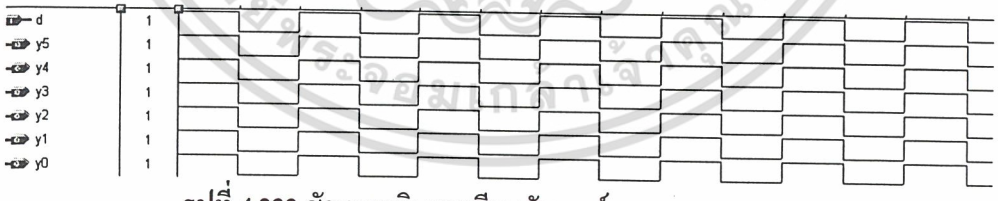
รูปที่ 4.231 สัญญาณอินพุทเทียบกับเอาท์พุทของวงจรคูณ

4.2.2.4.3 วงจร SIGN

เป็นวงจรที่ทำหน้าที่ในการเพิ่มบิตเครื่องหมายขึ้นเพื่อที่จะทำให้ระบบสามารถทำการรองรับค่าสูงสุดและต่ำสุดของการทำ ออโต้คอร์รีเลชัน ได้อย่างถูกต้อง



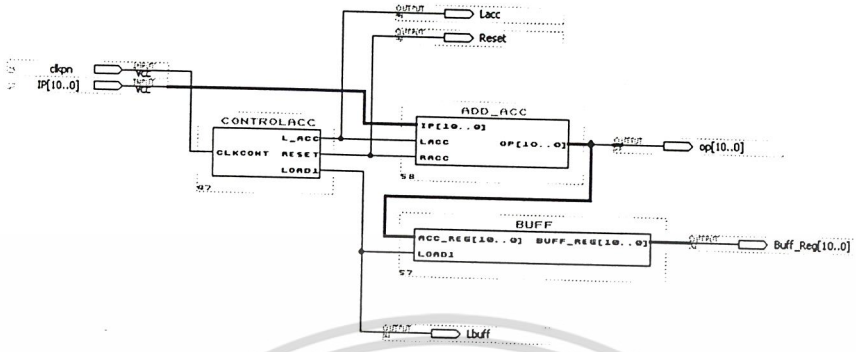
รูปที่ 4.232 บล็อกไดอะแกรมของวงจร SIGN



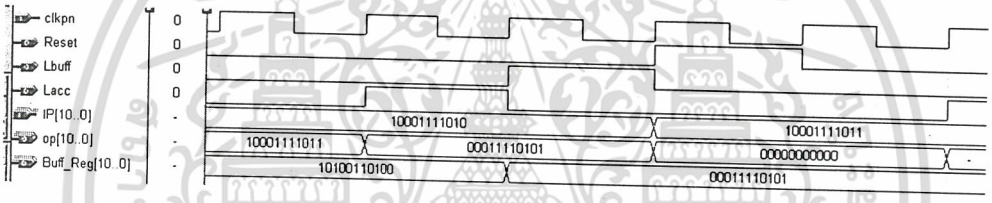
รูปที่ 4.233 สัญญาณอินพุทเทียบกับเอาท์พุทของวงจร SIGN

4.2.2.4.4 วงจร accumulator

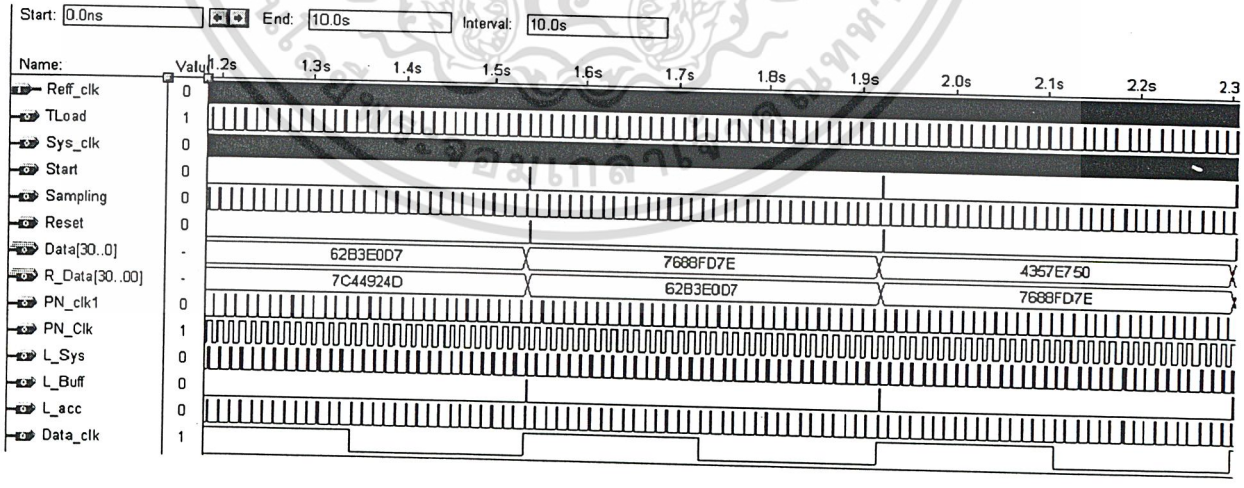
วงจร accumulator จะประกอบไปด้วย 3 ส่วนคือ วงจร control , วงจร buff (บัฟเฟอร์), วงจร add_acc (บวกสะสมค่า) ดังรูป



รูปที่ 4.234 บล็อกไดอะแกรมของวงจร accumulator



รูปที่ 4.235 สัญญาณที่ได้จากส่วนต่างๆของวงจร accumulator



รูปที่ 4.236 สัญญาณที่ได้จากวงจร Decode ทั้งหมด

4.3 ผลการทดลองที่ได้จากเครื่อง Logic Analyzer

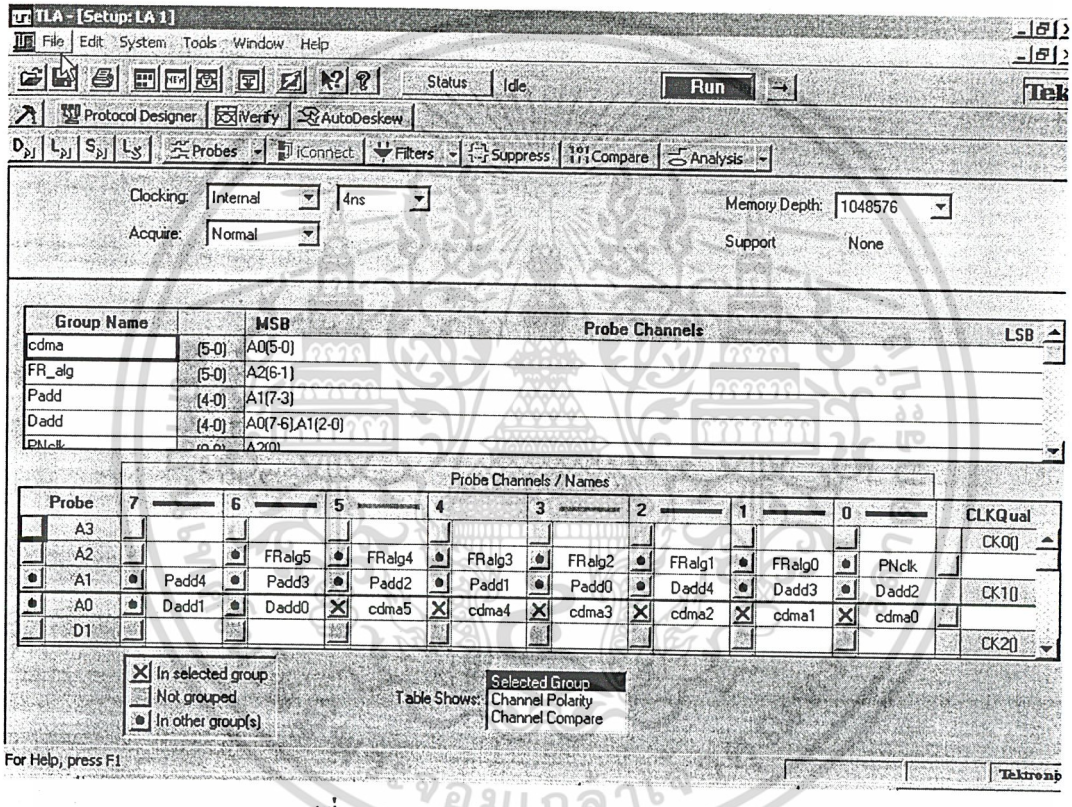
4.3.1 วิธีการใช้เครื่อง Logic Analyzer

1.) หลังจากเปิดเครื่อง Logic Analyzer แล้วทำการเปิดโปรแกรมที่ชื่อ TLA Application ขึ้นมา 

2.) นำสายวัดสัญญาณไปจับที่ขาที่ต้องการทำการวัดสัญญาณที่บอร์ด FPGA

3.) เลือกเมนู setup 

เพื่อทำการจัด group ของสัญญาณต่างๆให้ตรงกับสายสัญญาณที่บอร์ด FPGA



รูปที่ 4.237 แสดงเมนู setup ของ Logic Analyzer

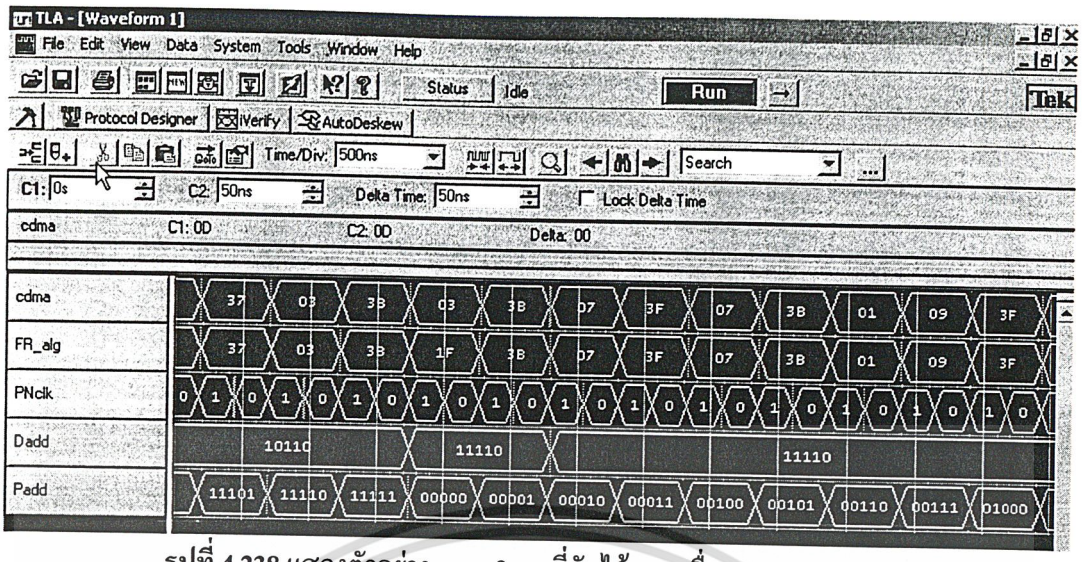
4.) กดเมนู probes 

เพื่อเช็คดูความเรียบร้อยของสายสัญญาณที่ต้องการจะทำการวัด

5.) กดเมนู run 

6.) จากนั้นดูผลที่เครื่อง Logic Analyzer ทำการวัดมาได้โดยกดที่ปุ่ม waveform 

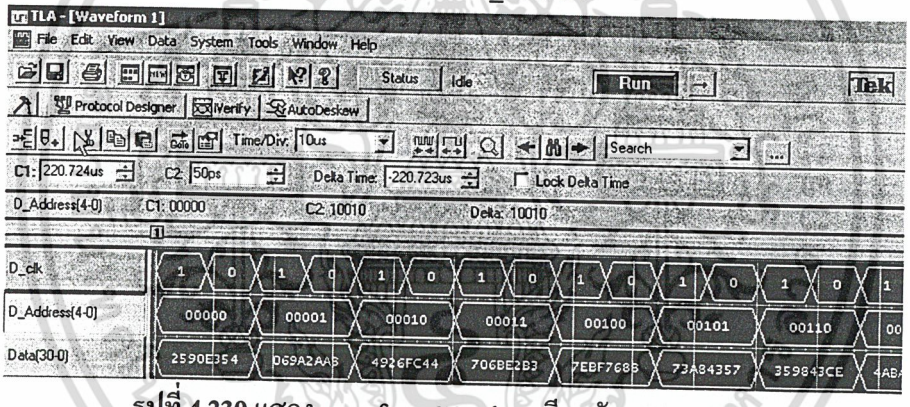
7.) ซึ่งผล waveform ที่ได้จะเป็นในลักษณะดังนี้



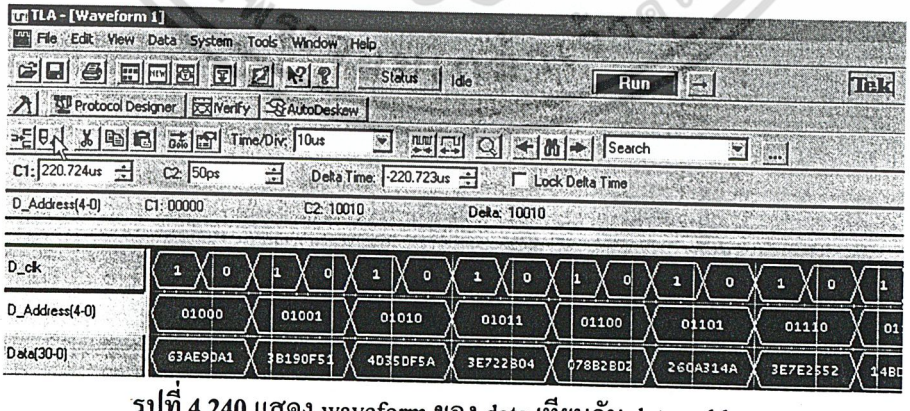
รูปที่ 4.238 แสดงตัวอย่าง waveform ที่วัดได้จากเครื่อง Logic Analyzer

4.3.2 ผลการทดลองจากทางด้านฝั่งส่ง

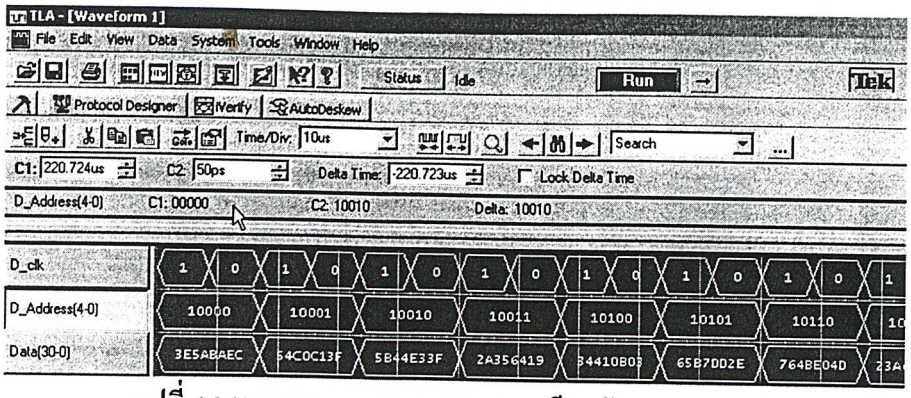
4.3.2.1 ผล waveform ของ data เทียบกับ data_address



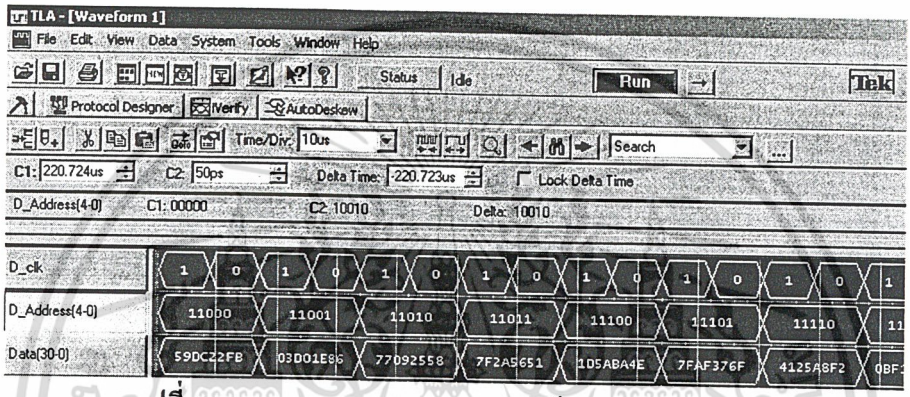
รูปที่ 4.239 แสดง waveform ของ data เทียบกับ data_address



รูปที่ 4.240 แสดง waveform ของ data เทียบกับ data_address

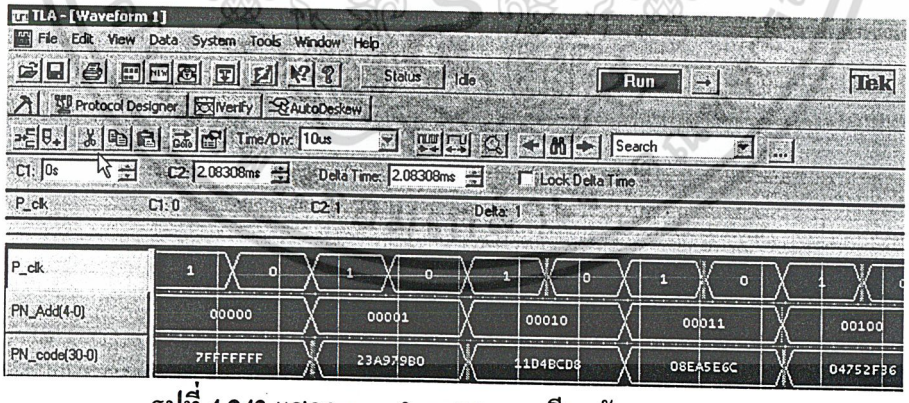


รูปที่ 4.241 แสดง waveform ของ data เทียบกับ data_address

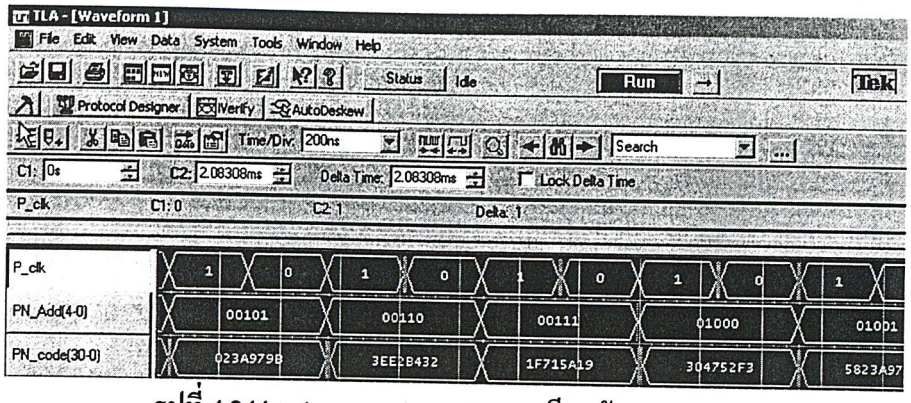


รูปที่ 4.242 แสดง waveform ของ data เทียบกับ data_address

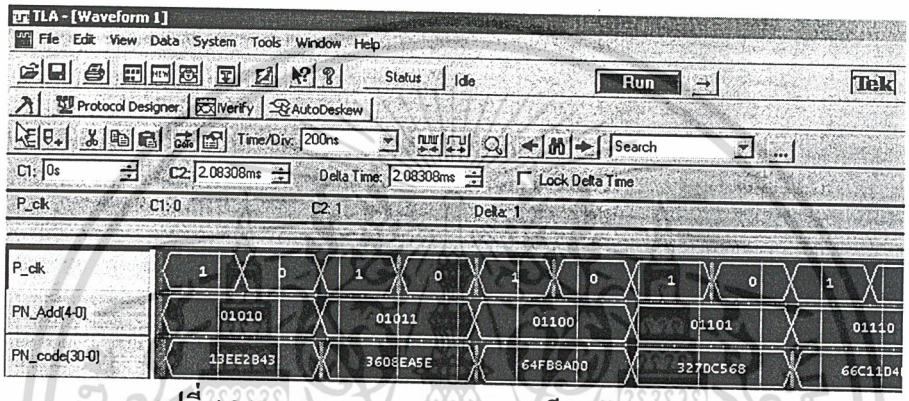
4.3.2.2 ผล waveform ของ pn เทียบกับ pn_address



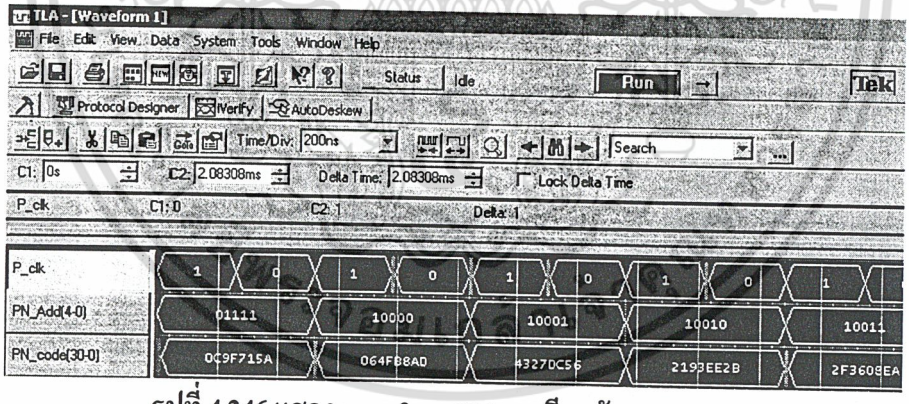
รูปที่ 4.243 แสดง waveform ของ pn เทียบกับ pn_address



รูปที่ 4.244 แสดง waveform ของ pn เทียบกับ pn_address

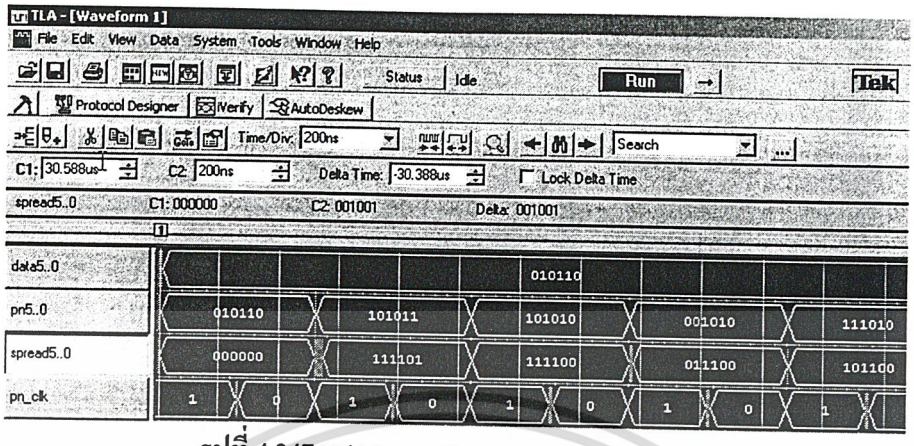


รูปที่ 4.245 แสดง waveform ของ pn เทียบกับ pn_address

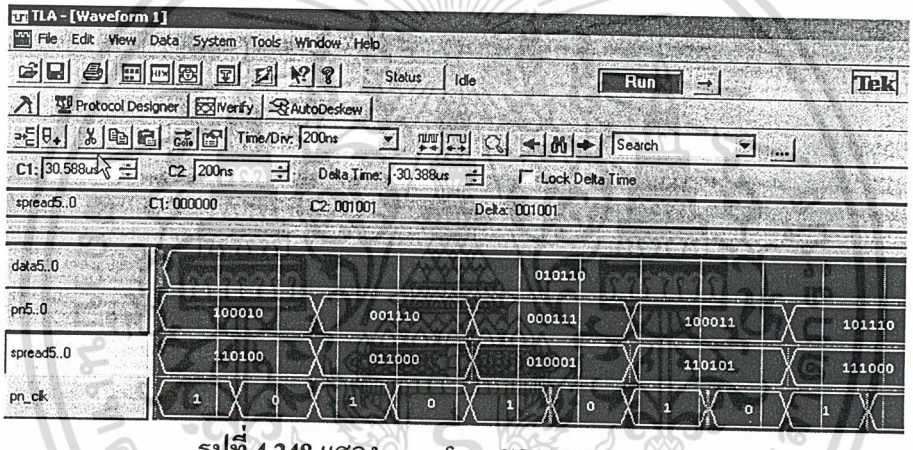


รูปที่ 4.246 แสดง waveform ของ pn เทียบกับ pn_address

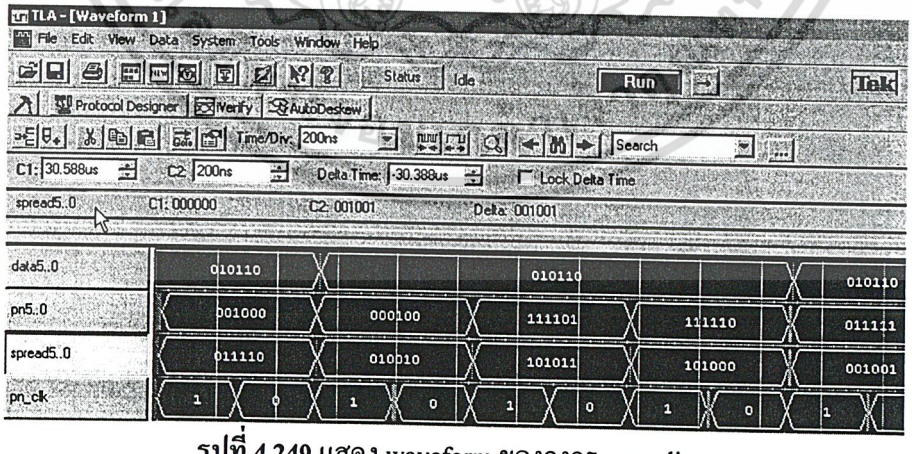
4.3.2.3 ผล waveform ของ วงจร spreading



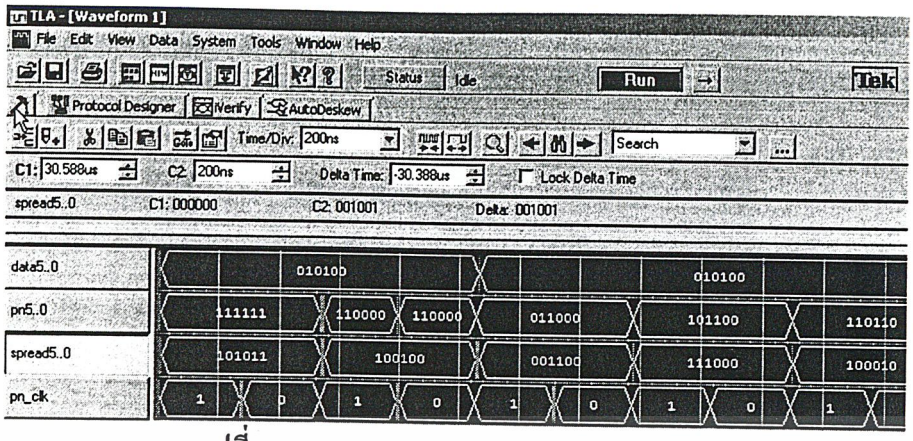
รูปที่ 4.247 แสดง waveform ของวงจร spreading



รูปที่ 4.248 แสดง waveform ของวงจร spreading

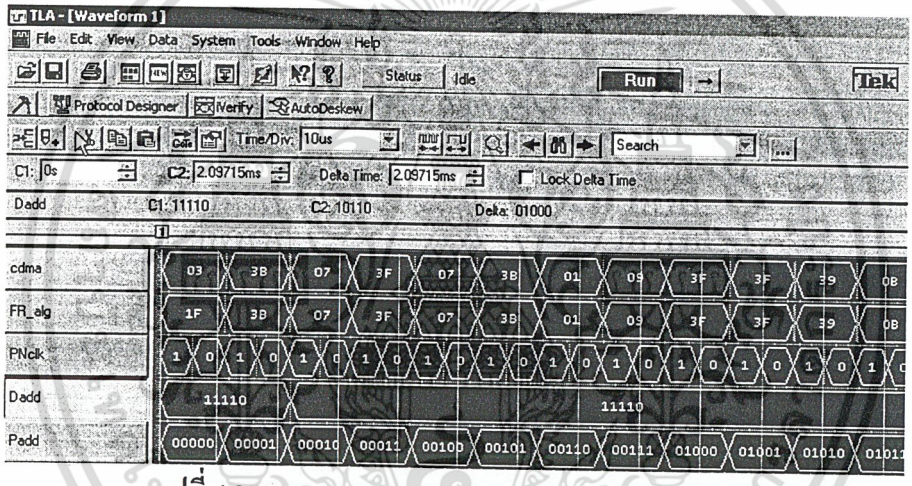


รูปที่ 4.249 แสดง waveform ของวงจร spreading

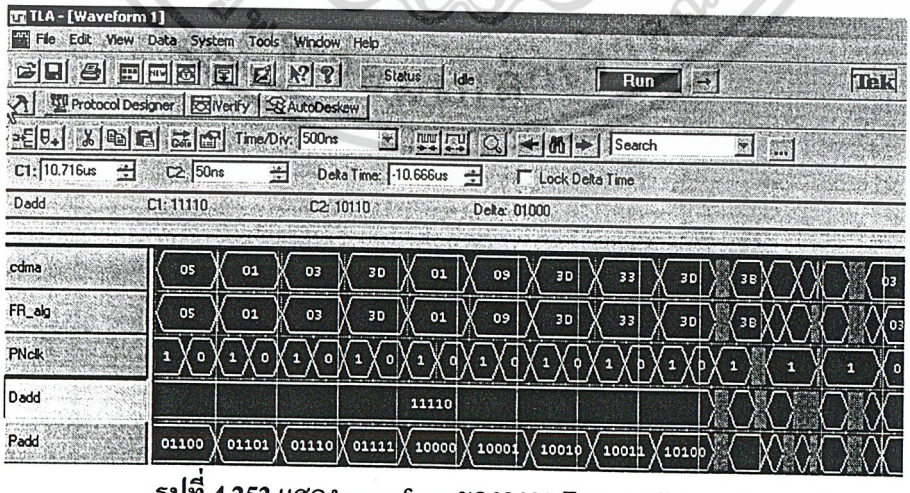


รูปที่ 4.250 แสดง waveform ของวงจร spreading

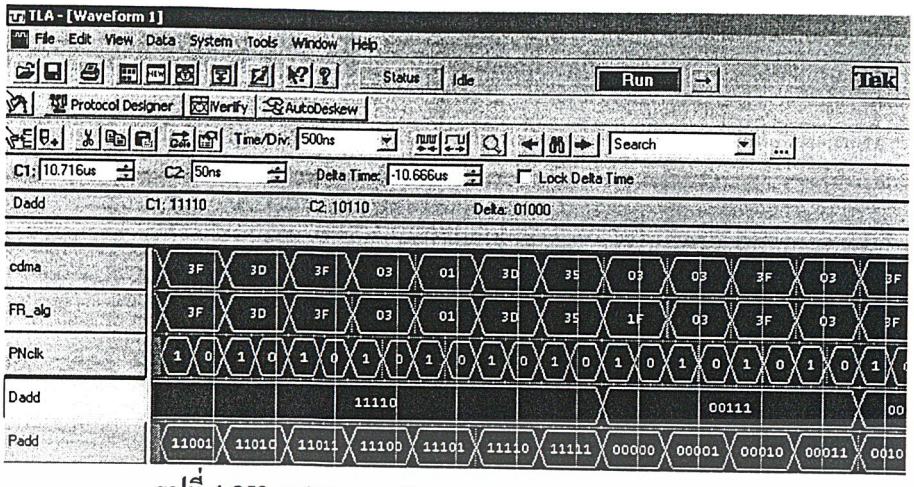
4.3.2.4 ผล waveform ของการสอดแทรกค่า FAW ที่ตำแหน่ง pn_address มีค่าเป็น 0



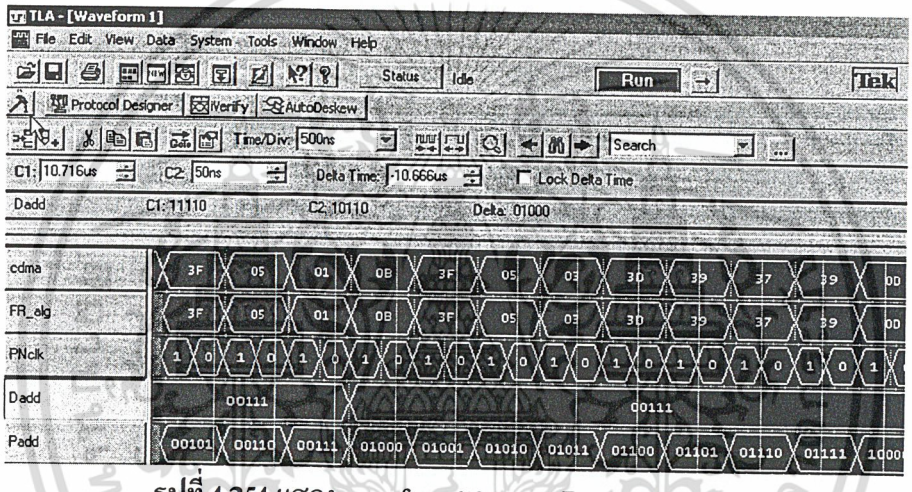
รูปที่ 4.251 แสดง waveform ของวงจร Frame Alignment



รูปที่ 4.252 แสดง waveform ของวงจร Frame Alignment

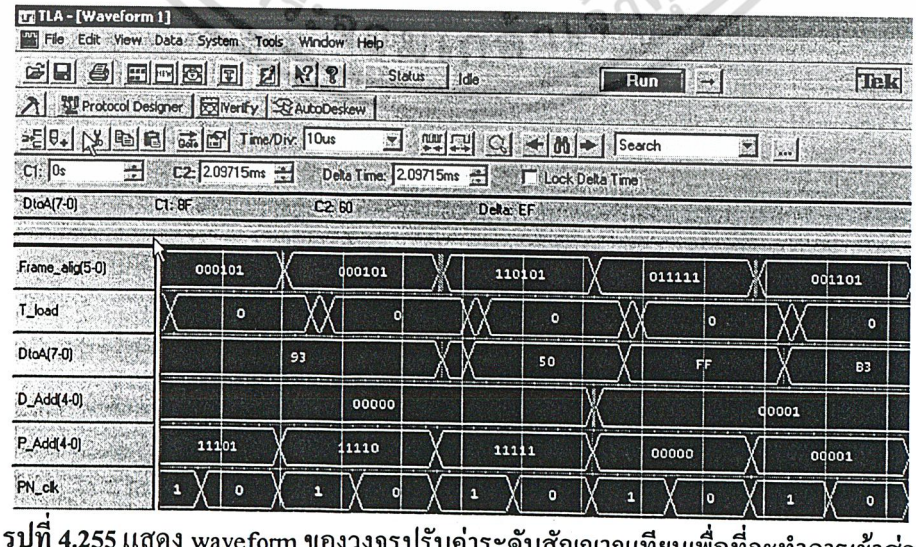


รูปที่ 4.253 แสดง waveform ของวงจร Frame Alignment

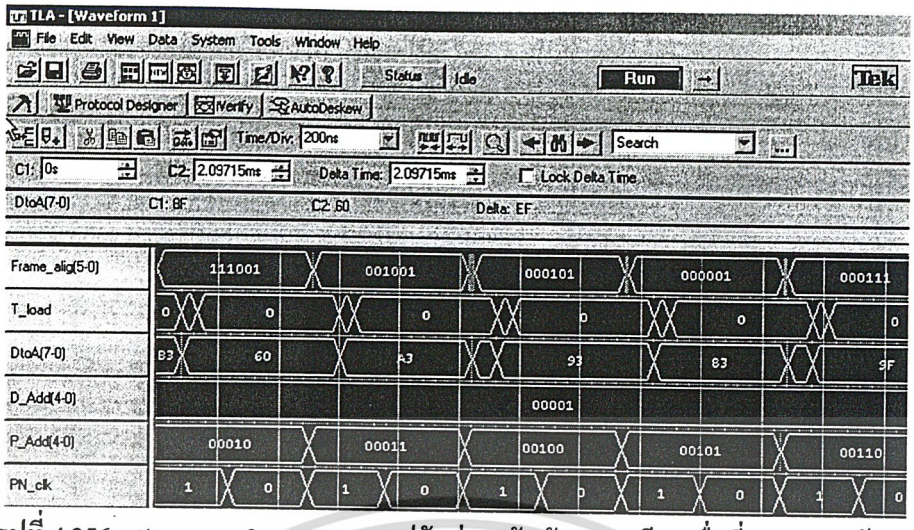


รูปที่ 4.254 แสดง waveform ของวงจร Frame Alignment

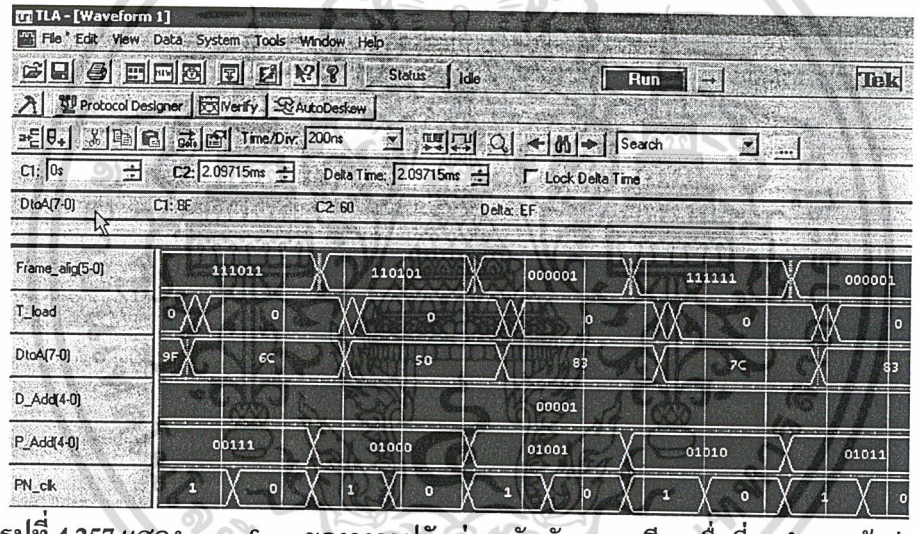
4.3.2.5 ผล waveform ของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter



รูปที่ 4.255 แสดง waveform ของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter



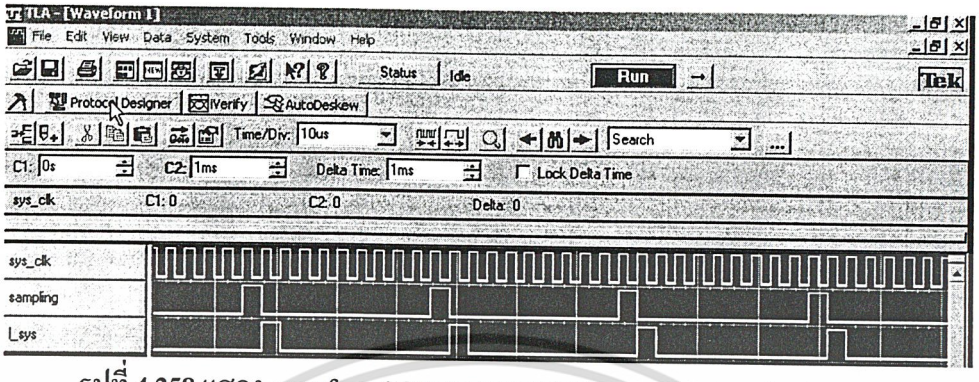
รูปที่ 4.256 แสดง waveform ของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter



รูปที่ 4.257 แสดง waveform ของวงจรปรับค่าระดับสัญญาณเทียมเพื่อที่จะทำการเข้าสู่อุปกรณ์ Digital to Analog Converter

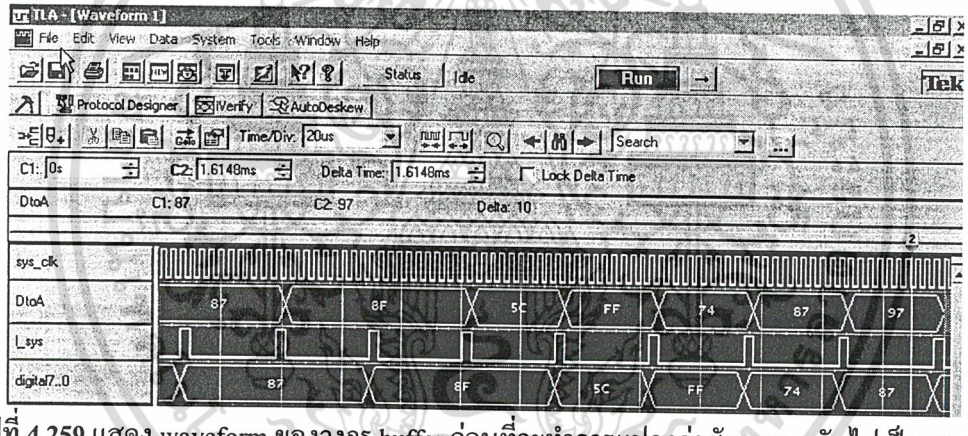
4.3.3 ผลการทดลองจากทางด้านฝั่งรับ

4.3.3.1 ผลwaveformของวงจรควบคุมการ sampling และ load_system



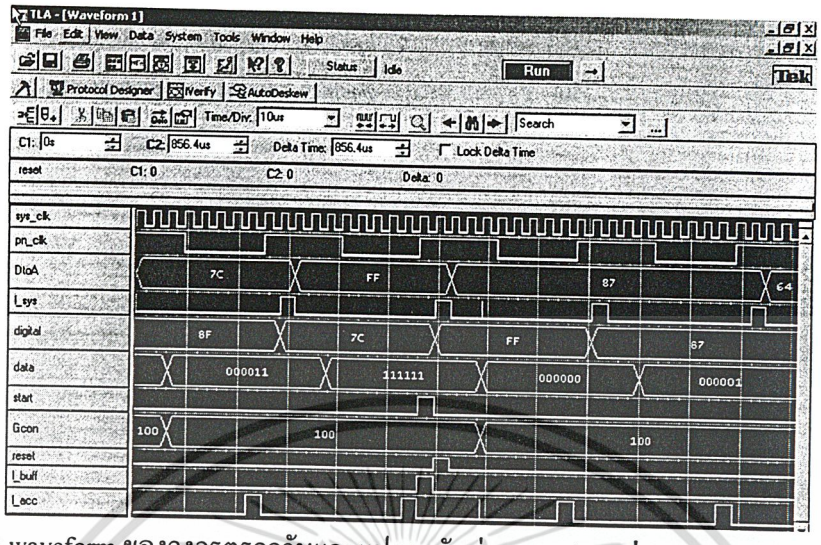
รูปที่ 4.258 แสดง waveform ของวงจรควบคุมการ sampling และ load_system

4.3.3.2 ผลwaveformของวงจร buffer ก่อนที่จะทำการแปลงค่าสัญญาณกลับไปเป็น digital

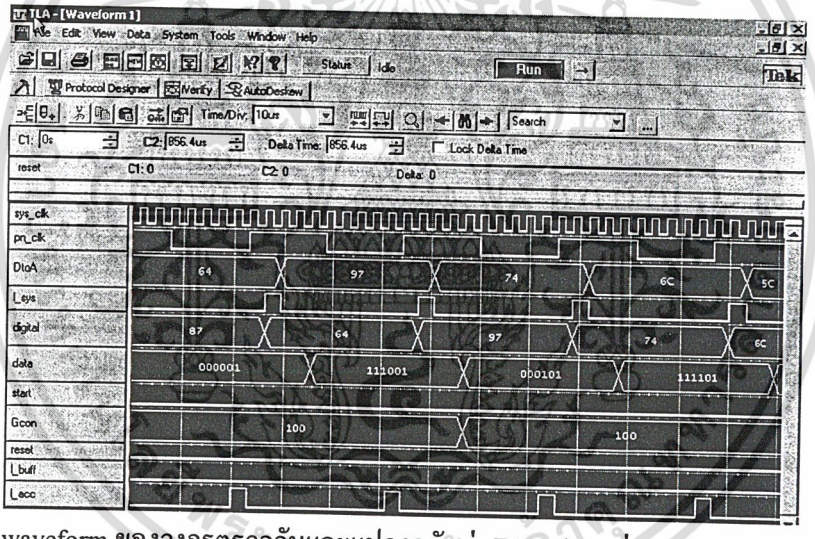


รูปที่ 4.259 แสดง waveform ของวงจร buffer ก่อนที่จะทำการแปลงค่าสัญญาณกลับไปเป็น digital

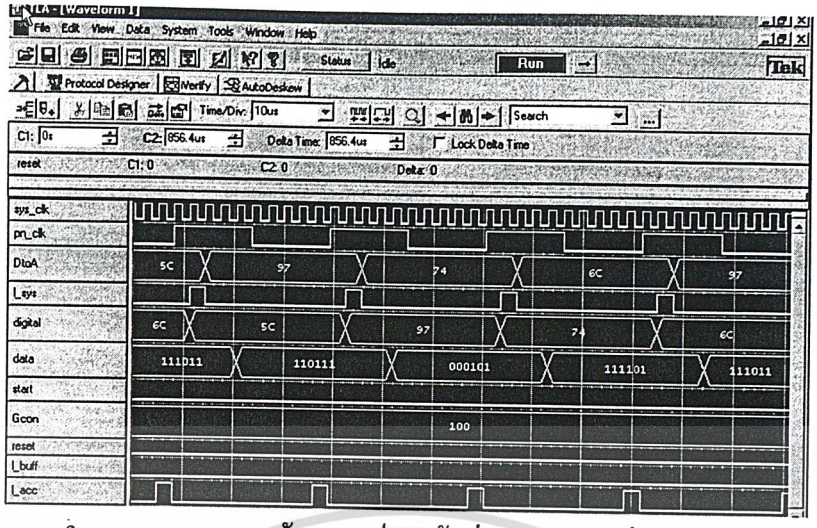
4.3.3.3 ผล waveform ของวงจร ตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)



รูปที่ 4.260 แสดง waveform ของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)

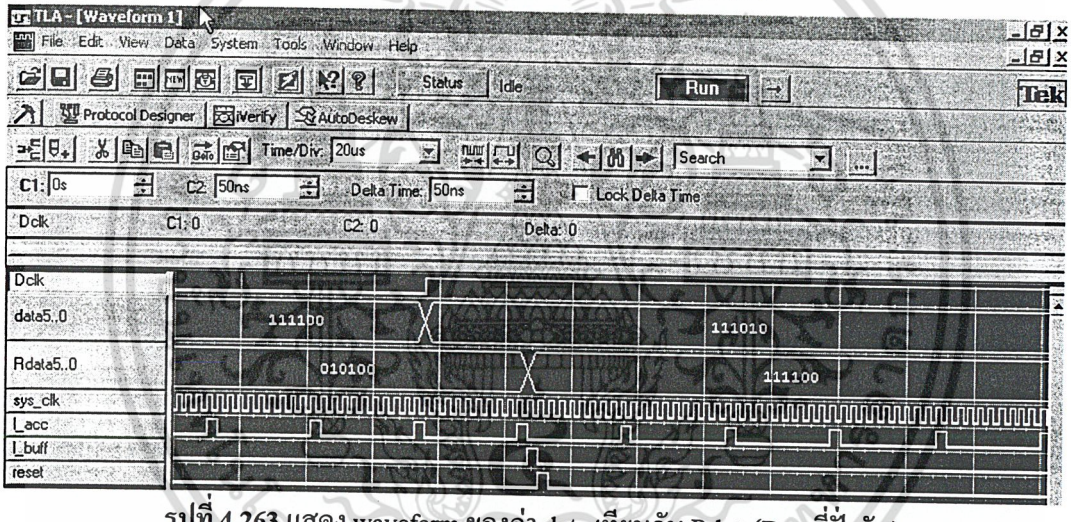


รูปที่ 4.261 แสดง waveform ของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)

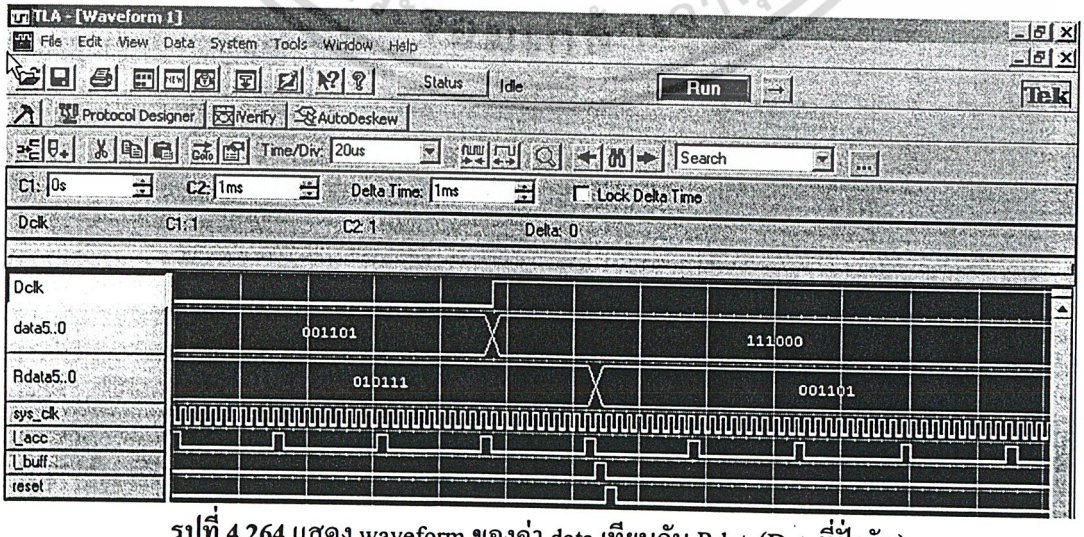


รูปที่ 4.262 แสดง waveform ของวงจรตรวจจับและแปลงกลับค่า FAW (รวมส่วนวงจรควบคุมระบบด้วย)

4.3.3.4 ผล waveform ของ Data ที่ฝั่งส่งเทียบกับค่า Data ที่ฝั่งรับ



รูปที่ 4.263 แสดง waveform ของค่า data เทียบกับ Rdata(Dataที่ฝั่งรับ)



รูปที่ 4.264 แสดง waveform ของค่า data เทียบกับ Rdata(Dataที่ฝั่งรับ)

บทที่ 5
บทวิจารณ์และสรุป

การทำโครงงานเรื่องการออกแบบและจำลองการสร้าง Trunk โดยวิธี CDMA นั้นได้แบ่งการทดลองออกเป็น 2 ส่วน คือ ที่ทำการออกแบบโดยใช้โปรแกรม MATLAB เพื่อทำการวัด Bit Error Rate เปรียบเทียบกับ SNR โดยการเปลี่ยนตัวแปรสองตัวที่มีผลกับระบบคือ ชนิดของโค้ดกับความยาว และจำนวนผู้ใช้ต่อช่องสัญญาณ และ ในส่วนทำเป็นส่วนจำลองการสร้างนั้นได้ทำการทดลองแยกเป็นส่วนๆ โดยสังเคราะห์อุปกรณ์ต่างๆ ขึ้นมาด้วยภาษา VHDL และทำการทดลองในอุปกรณ์ FPGA ซึ่งหลักการบางอย่างที่ใช้ในการออกแบบนี้อาจจะไม่เหมาะสมบางประการเนื่องจาก มีความรู้บางส่วนยังไม่เพียงพอ โดยจะสรุปข้อเสนอแนะเพื่อให้เกิดประโยชน์แก่ผู้อ่านดังนี้

5.1 ส่วนการคำนวณโดยโปรแกรม MATLAB

ในส่วนของการออกแบบครั้งนี้ได้ทำการทดลองเปรียบเทียบประสิทธิภาพของระบบ Trunk ใหม่ที่ได้ทำขึ้นโดยใช้เปรียบเทียบกับระบบ Trunk แบบเก่าที่มีอยู่แล้วโดยเริ่มทำการทดลองตั้งแต่ออกแบบชุดรหัสสัญญาณที่จะนำมาเข้าโค้ดให้มีความเร็วเท่ากับระบบที่มีอยู่ (ใช้รหัส PN code 31-modulo) แล้วทดสอบประสิทธิภาพปรากฏว่าหากมีจำนวนผู้ใช้เต็มที่เท่ากับระบบ TDM แบบเดิมปรากฏว่ามีประสิทธิภาพแย่ลงกว่าระบบดิมที่มีอยู่ จากนั้นลองทำการลดผู้ใช้ภายในระบบลง ซึ่งหมายความว่าส่งด้วยความเร็วเท่าเดิมจะพบว่าประสิทธิภาพเพิ่มขึ้นเป็นลำดับ จากนั้นทำการเพิ่มชุดโค้ดให้ยาวขึ้นปรากฏว่าประสิทธิภาพสูงกว่าระบบเก่า

5.2 ส่วนการออกแบบโดยโปรแกรม MAX+plus II

ส่วนการทดลองในครั้งนี้ทำโดยการใช้ภาษา VHDL ในการบรรยายการทำงานในส่วนต่างๆซึ่งได้แก่การหารสัญญาณนาฬิกา วงจรกำเนิดสัญญาณแบบสุ่ม การทำการแปลงค่าแบบ 2's complement การทำการบวก วงจรเข้าสัญญาณแบบสเพรดสเปกตรัม วงจรรวมค่า วงจรบวกแบบสะสมค่า วงจรคูณ โดยการเก็บค่าผลการทดลองจากอุปกรณ์ FPGA นั้น scope ที่มีใช้งานอยู่ไม่สามารถทำการวัดค่าได้ โดยทั้งนี้ในการเก็บผลการทดลองจึงใช้เครื่อง Logic Analyzer เก็บผลการทดลองซึ่งวิธีการทดสอบอุปกรณ์ FPGA โดย Logic Analyzer นั้นได้มีการบรรยายไว้ในส่วนของผลการทดลองด้วย

5.3 สิ่งที่ควรปรับปรุง

ควรทำการศึกษาเรื่องผลประโยชน์ที่ได้หากทำการนำเอาระบบนี้ไปทำการใช้กับระบบอื่นๆ ว่าสามารถเพิ่มประสิทธิภาพในด้านอื่นๆ เช่น การรองรับการใช้งานกับระบบเดิมที่มีอยู่แล้ว

หนังสืออ้างอิง

[1] ASTRON LOGIC RESEARCH and DEVELOPMENT, **"เปิดโลก FPGA กับ WIZARD PLD-AD1"**, บริษัท Altera Corporation, กรุงเทพฯ 2546

[2] สมศักดิ์ มิตะถา, **"การออกแบบวงจรดิจิตอลและวงจรตรรกะ Digital and Logic Design"**, ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2543

[3] สุวิพล สิทธิชีวภาค, **"เทคโนโลยีการสื่อสารระบบดิจิตอล"**, คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2539

[4] ชาติชาย ดิษฐกุล, นอ. รน., **"หนังสือสอนวิธีการใช้ภาษา VHDL"**

[5] วิศัลย์ พัวรุ่งโรจน์, วิศิษฏ์ พัวรุ่งโรจน์, **"399 ฟังก์ชัน Excel"**, บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), กรุงเทพฯ, 2546

[6] S. C. Yang, **"CDMA RF System Engineering"**

[7] J. Castelo, **"Implementing a Pseudo – Random Number generator In a FPGA "**, report, IFIC Unkversity of Valencia. Spain

[8] J. Stremler, **"Introduction to Communication System"**, Addision, 2533

[9] Robert C. Dixon, **"Spread Spectrum System with Commercial Applicatios"**

[10] H. Harada, R. Prasad, **"Simulation and Software Radio for Mobile Communication"**

[11] J. G. Proakis, **"Digital Communications"**

[12] ดำรง จันทร์เรือง, **"การประยุกต์โครงข่ายภายในการรับส่ง PCM-TDM FRAME"**, วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

[13] มนัส สังวรศิลป์, วรรัตน์ ภัทรอมรกุล, **"คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์"**

[14] John G. Proakis, M. Salehi, **"Communication Systems Engineering Second Edition"**

[15] P. Supnithi, Dr., **"Simulation of a Gaussian channel" Lab sheet, King Mong Kut's Institute of Technology Ladkrabrang, Faculty of Engineering, Telecommunication Dept.**

[16] ลัญฉกร วุฒิสิทธิกุลกิจ, **"MATLAB การประยุกต์ใช้งานทางวิศวกรรมไฟฟ้า"**

ภาคผนวก ก.

ภาคผนวก ก.

โค้ด MATLab

3.1.1 โปรแกรมเลื่อนค่าของข้อมูล [10]

```
% shift.m
%
% Shift the contents of the register.
%

function [outregi] = shift(inregi, shiftr, shiftu)

% ************************************************************

% inregi        : Vector or matrix
% shiftr        : The amount of shift to the right.
% shiftu        : The amount of shift to the top.
% outregi       : Register output

% ************************************************************

      [h, v]   = size(inregi);
      outregi = inregi;

      shiftr = rem(shiftr, v);
      shiftu = rem(shiftu, h);

      if shiftr > 0
            outregi(:,1 :shiftr) = inregi(:,v-shiftr+1:v );
            outregi(:,1+shiftr:v ) = inregi(:,1 :v-shiftr);
      elseif shiftr < 0
            outregi(:,1 :v+shiftr) = inregi(:,1-shiftr:v );
            outregi(:,v+shiftr+1:v ) = inregi(:,1 :-shiftr);
      end
            inregi = outregi;

      if shiftu > 0
            outregi(1 :h-shiftu,:) = inregi(1+shiftu:h, :);
            outregi(h-shiftu+1:h, :) = inregi(1 :shiftu,:);
      elseif shiftu < 0
            outregi(1 :-shiftu,:) = inregi(h+shiftu+1:h, :);
            outregi(1-shiftu:h, :) = inregi(1    :shiftu,:);
      end

% ********************* end of file ***********************
```

3.1.2 โปรแกรมกำเนิดชุดรหัส m-sequence [10]

```
% mseq.m
%
% The generation function of M-sequence
% An example
%   stg     = 3
%   taps    = [1, 3]
%   inidata = [1, 1, 1]
%   n       = 2
%

function [mout] = mseq(stg, taps, inidata, n)

% ************************************************************
% stg       : Number of stages
% taps      : Position of register feedback
% inidata   : Initial sequence
% n         : Number of output sequence(It can be omitted)
% mout      : Output M sequence

% ************************************************************
if nargin < 4
    n = 1;
end

mout = zeros(n,2^stg-1);
fpos = zeros(stg, 1);
fpos(taps) = 1;
for ii=1:2^stg-1
mout(1,ii) = inidata(stg);          % stroge of the
                                    % Output data
num = mod(inidata*fpos,2);          % calculation of the
                                    % feedback data
inidata(2:stg) = inidata(1:stg-1);   % one shifts
                                     % the register
inidata(1)     = num;                % return
                                     % feedback data

end

% ************************************************************
if n > 1
    for ii=2:n
        mout(ii,:) = shift(mout(ii-1,:), 1, 0);
    end
end

% ********************* end of file ***********************
```

3.1.3 โปรแกรมกำเนิดชุด m-sequence [10]

```
% goldseq.m
%
% The generation function of Gold sequence
%

function [gout] = goldseq(m1, m2, n)

% ************************************************************

% m1     : M-sequence 1
% m2     : M-sequence 2
% n      : Number of output sequence(It can be omitted)
% gout   : output Gold sequence

% ************************************************************

if nargin < 3
      n = 1;
end

gout = zeros(n,length(m1));

for ii=1:n
      gout(ii,:)   = xor(m1,m2);
      m2           = shift(m2,1,0);
end

% ********************* end of file ************************
```

3.1.4 โปรแกรมทดสอบค่าออโต้คอร์ลีเลชั่น

```
% autocorr.m
%
% Autocorrelation function of a sequence
%

function [out] = autocorr(indata, tn)

% ************************************************************

% indata : input sequence
% tn     : number of period
% out    : autocorrelation data

% ************************************************************

if nargin < 2
      tn = 1;
end

ln  = length(indata) ;
out = zeros(1,ln*tn) ;

for ii=0:ln*tn-1
    out(ii+1) = sum(indata.*shift(indata,ii,0));
end

% ********************* end of file ************************
```

3.1.5 โปรแกรมทดสอบค่าครอสคอร์ลีเลชั่น

```
% crosscorr.m
%
% Crosscorrelation function of a sequence
%

function [out] = crosscorr(indata1, indata2, tn)

% ********************************************************

% indata1    : input sequence1
% indata2    : input sequence2
% tn         : number of period
% out        : crosscorrelation data

% ********************************************************

if nargin < 3
    tn = 1;
end

ln  = length(indata1);
out = zeros(1,ln*tn);

for ii=0:ln*tn-1
    out(ii+1) = sum(indata1.*shift(indata2,ii,0));
end

% ******************** end of file ***********************
```

3.1.6 โปรแกรมรวมเพื่อทำการหาค่า BER/SNR

```
% clearing all parameter
    clear all                       % clear all variable and data
    clc
    close all
% Constant parameter
    NofData=5000;                   % Set number of data
    NofPNcd=31;                     % Set PN-modulo number
    NofUser=31;                     % number of total user
    Nofcode=NofData*NofPNcd;        % Number of all data transmitting
    NofbP=6;
    NofTbit=Nofcode*NofbP;
    NofTL=NofData*NofUser;

% Block length
    L=2;                            % Number of level

% PN code 31-modulo
    PNcode =  load('Msequence.txt');

%Binary signalling
    Quanti  =   [1 -1];             % !!!! Alphabet !!!!!!
    mPNcode =   PNcode+1.;
    PNcode  =   Quanti(mPNcode);

for  SNR = 17:20                    % Change here for SNR
    NofBlock = 200;
    noe=0;
    nod=0;
    Tnoe=0;
    Tnod=0;
    for loop_num = 1:NofBlock


% ********************************************************** %
%                       Tramsmitter
% ********************************************************** %

  % Generate the input
      gData   =   ceil(L*rand(NofUser,NofData));
  % Randoming data
      Data    =   Quanti(gData);

   % ************************************
   %             Old TDM Trunk
   % ************************************

        % Time Division Multiplexing
            TDM(1,NofUser)=0;
            k=1;
            for i=1:NofUser
                for j=1:NofData
                    TDM(1,k) = Data(i,j);
                    k = k+1;
                end
             end
```

```
    % ************************************
    %              CDMA Trunk
    % ************************************

          % Spreading Data
              for i=1:NofUser
              k=1;
                  for j=1:NofData        % number of data
                      for n=1:NofPNcd    % number of PN modulo order
                          DSSS(i,k) = Data(i,j)*PNcode(i,n);
                                         % spreading data
                          k=k+1;
                      end
                  end
              end                        % End of spreading data

          % Multiple Access
                    CDMA = sum(DSSS);
                    vaCDMA = CDMA;

% ************************************************************** %
%                          Receiver
% ************************************************************** %

  rTDM=zeros(1,Nofcode);

% Received signal CDMA & TDM

          % nosie power calculation
            sp_cdma=sum(vaCDMA.^2)./length(vaCDMA); % normalize=1
            sig1     =  sqrt(sp_cdma/(10.^(SNR/10)));
            Noise    =  sig1*randn(1,length(vaCDMA));
            rvaCDMA  =  vaCDMA + Noise;

            sp_tdm   = sum(TDM.^2)./length(TDM); % normalize=1
            sig2     = sqrt(sp_tdm/(10.^(SNR/10)));
            TNoise   = sig2*randn(1,NofTL);
            rTDM     = TDM + TNoise;

    % ************************************
    %              Old TDM Trunk decision
    % ************************************

           TDM_hat=sign(rTDM); % Quantize 2 level

    % ************************************
    %              CDMA Trunk
    % ************************************

    % Spreading PNcode
          for i=1:NofPNcd              % number of PN modulo order
          k=1;
              for j=1:NofData          % number of data
                  for n=1:NofPNcd      % number of PN modulo order
                      dePNcode(i,k) = PNcode(i,n);
                                       % building PN code matrix
                      k=k+1;
                  end
              end
          end                  % Despreading data in to n-user array
```

```
            CDMA_hat=rvaCDMA;
            for i=1:NofUser
                deDSSS(i,:) = CDMA_hat.*dePNcode(i,:);
            end

        % Autocorrelation
            DataOut1=zeros(NofUser,NofData);

            for i=1:NofUser
              m=1;
              for k=1:NofData
                  for j=1:NofPNcd
                     DataOut1(i,k) = DataOut1(i,k)+deDSSS(i,m);
                     m=m+1;
                  end
              end
            end

          % Bit Dicision
            DataOut=sign(DataOut1);

    % ***************************************
    %              Old TDM Trunk
    % ***************************************

            Tnoe1=sum(TDM_hat(1,:)~=TDM(1,:));
            Tnod1=length(TDM_hat);
            Tnoe=Tnoe+Tnoe1;
            Tnod=Tnod+Tnod1;

    % ***************************************
    %              CDMA Trunk
    % ***************************************

            noe1=sum(Data(1,:)~=DataOut(1,:));
            nod1=length(Data);
            noe=noe+noe1;
            nod=nod+nod1;
          end                        %End of all blocks

    % ***************************************
    %              BER of CDMA Trunk
    % ***************************************
        BER=noe/nod;
        fid=fopen('BERmseqU31.txt','a');
        fprintf(fid,'%d\t%d\t\n',SNR,BER);

    % ***************************************
    %              BER of TDM Trunk
    % ***************************************
        TBER=Tnoe/Tnod;
        Tfid=fopen('BERTDMR.txt','a');
        fprintf(Tfid,'%d\t%d\t\n',SNR,TBER);

end %End of SNR loop
fclose(fid);
fclose(Tfid)
```

ภาคผนวก ข.

ภาคผนวก ข.

โค้ด VHDL จาก MAX+plus II

1. วงจรหารความถี่ 64 เพื่อวงจรกำเนิด ข้อมูล

```
library ieee;
use ieee.std_logic_1164.all;

entity DivData is
port(sys_clk : in std_logic;
     data_clk : out std_logic);
end;

architecture clkosc of DivData is
signal count:integer range 0 to 61;
begin
process(sys_clk)
      begin
      if sys_clk'event and sys_clk = '1' then
            if   count /= 61 then
            count <= count+1;
      else count <=0;
      end if;
            if count < 31 then
            data_clk <= '1';
      else data_clk <= '0';
      end if;
      end if;
end process;
end clkosc;
```

2. วงจรหารความถี่ 2 เพื่อวงจรกำเนิด ข้อมูล

```
library ieee;
use ieee.std_logic_1164.all;

entity divpn is
port(sys_clk : in std_logic;
      Pn_clk : out std_logic);
end;

architecture clkosc of divpn is
signal count:integer range 0 to 1;
begin
process(sys_clk)
begin
if sys_clk'event and sys_clk = '1' then
      if   count /= 1 then
           count <= count+1;
      else count <=0;
      end if;
      if count < 1 then
           Pn_clk <= '1';
      else Pn_clk <= '0';
      end if;
end if;
end process;
end clkosc;
```

3. วงจรกำเนิดสัญญาณเลือกตำแหน่งชุดโค้ดของ PN code

```
library ieee;
use ieee.std_logic_1164.all;

entity PNcodeSelector is
port(sys_clk : in  std_logic;
       addr : out std_logic_vector(4 downto 0));
end;

architecture clkosc of PNcodeSelector is
signal count:integer range 0 to 30;
begin
process(sys_clk)
begin
if sys_clk'event and sys_clk = '1' then
     if   count /= 30 then
          count <= count+1;
     else count <=0;
     end if;

     case count is
          when 0 =>
             addr(4 downto 0) <= "00000";
          when 1 =>
             addr(4 downto 0) <= "00001";
          when 2 =>
             addr(4 downto 0) <= "00010";
          when 3 =>
             addr(4 downto 0) <= "00011";
          when 4 =>
             addr(4 downto 0) <= "00100";
          when 5 =>
             addr(4 downto 0) <= "00101";
          when 6 =>
             addr(4 downto 0) <= "00110";
          when 7 =>
             addr(4 downto 0) <= "00111";
          when 8 =>
             addr(4 downto 0) <= "01000";
          when 9 =>
             addr(4 downto 0) <= "01001";
          when 10 =>
             addr(4 downto 0) <= "01010";
          when 11 =>
             addr(4 downto 0) <= "01011";
          when 12 =>
             addr(4 downto 0) <= "01100";
          when 13 =>
             addr(4 downto 0) <= "01101";
          when 14 =>
             addr(4 downto 0) <= "01110";
          when 15 =>
             addr(4 downto 0) <= "01111";
          when 16 =>
```

```
                addr(4 downto 0) <= "10000";
            when 17 =>
                addr(4 downto 0) <= "10001";
            when 18 =>
                addr(4 downto 0) <= "10010";
            when 19 =>
                addr(4 downto 0) <= "10011";
            when 20 =>
                addr(4 downto 0) <= "10100";
            when 21 =>
                addr(4 downto 0) <= "10101";
            when 22 =>
                addr(4 downto 0) <= "10110";
            when 23 =>
                addr(4 downto 0) <= "10111";
            when 24 =>
                addr(4 downto 0) <= "11000";
            when 25 =>
                addr(4 downto 0) <= "11001";
            when 26 =>
                addr(4 downto 0) <= "11010";
            when 27 =>
                addr(4 downto 0) <= "11011";
            when 28 =>
                addr(4 downto 0) <= "11100";
            when 29 =>
                addr(4 downto 0) <= "11101";
            when 30 =>
                addr(4 downto 0) <= "11110";
        end case;
        end if;
end process;
end clkosc;
```

4. ตัวค่า address ตำแหน่งของชุด PN code *.mif ไฟล์

```
DEPTH = 31; % Memory depth and width are required     %
WIDTH = 31; % Enter a decimal number      %

ADDRESS_RADIX = BIN;     % Address and value radixes are optional%
DATA_RADIX    = BIN;     % Enter BIN, DEC, HEX, or OCT; unless %
                         % otherwise specified, radixes = HEX %

-- Specify values for addresses, which can be single address or range

CONTENT
BEGIN
      00000 :     0100011101010010111100110110000;
      00001 :     0010001110101001011110011011000;
      00010 :     0001000111010100101111001101100;
      00011 :     0000100011101010010111100110110;
      00101 :     0000010001110101001011110011011;
      00110 :     0111110111000101011010000110010;
      00111 :     0011111011100010101101000011001;
      01000 :     0110000010001110101001011110011;
      01001 :     1011000001000111010100101111001;
      01010 :     0010011111011100010101101000011;
      01011 :     0110110000010001110101001011110;
      01100 :     1100100111111011100010101101000;
      01101 :     0110010011111101110001010110100 0;
      01110 :     1100110110000010001110101001011;
      01111 :     0001100100111111011100010101101 0;
      10000 :     0000110010011111101110001010110 1;
      10001 :     1000011001001111101110001010110;
      10010 :     0100001100100111111011100010101 1;
      10011 :     0101111001101100000100011101010;
      10100 :     1101000011001001111101110001010;
      10101 :     1001011110011011000001000111010;
      10110 :     1011010000110010011111011100010;
      10111 :     1010010111110011011000001000111 0;
      11000 :     0101001011110011011000001000111;
      11001 :     1010100101111001101100000100011;
      11010 :     0010101101000011001001111101110;
      11011 :     1110101001011110011011000001000;
      11100 :     0111010100101111001101100000100;
      11101 :     1100010101101000011001001111101;
      11110 :     1110001010110100001100100111110;
      11111 :     0111000101011010000110010011111;
END ;
```

5. วงจรกำเนิดสัญญาณเลือกตำแหน่งชุดโค้ดของ PN code

```
library ieee;
use ieee.std_logic_1164.all;

entity DataSelector is
port(sys_clk : in  std_logic;
     addr : out std_logic_vector(4 downto 0));
end;

architecture clkosc of DataSelector is
signal count:integer range 0 to 30;
begin
process(sys_clk)
begin
if sys_clk'event and sys_clk = '1' then
      if   count /= 30 then
           count <= count+1;
      else count <=0;
      end if;
      case count is
            when 0 =>
               addr(4 downto 0) <= "00000";
            when 1 =>
               addr(4 downto 0) <= "00001";
            when 2 =>
               addr(4 downto 0) <= "00010";
            when 3 =>
               addr(4 downto 0) <= "00011";
            when 4 =>
               addr(4 downto 0) <= "00100";
            when 5 =>
               addr(4 downto 0) <= "00101";
            when 6 =>
               addr(4 downto 0) <= "00110";
            when 7 =>
               addr(4 downto 0) <= "00111";
            when 8 =>
               addr(4 downto 0) <= "01000";
            when 9 =>
               addr(4 downto 0) <= "01001";
            when 10 =>
               addr(4 downto 0) <= "01010";
            when 11 =>
               addr(4 downto 0) <= "01011";
            when 12 =>
               addr(4 downto 0) <= "01100";
            when 13 =>
               addr(4 downto 0) <= "01101";
            when 14 =>
               addr(4 downto 0) <= "01110";
            when 15 =>
               addr(4 downto 0) <= "01111";
            when 16 =>
               addr(4 downto 0) <= "10000";
            when 17 =>
```

```
                addr(4 downto 0) <= "10001";
            when 18 =>
                addr(4 downto 0) <= "10010";
            when 19 =>
                addr(4 downto 0) <= "10011";
            when 20 =>
                addr(4 downto 0) <= "10100";
            when 21 =>
                addr(4 downto 0) <= "10101";
            when 22 =>
                addr(4 downto 0) <= "10110";
            when 23 =>
                addr(4 downto 0) <= "10111";
            when 24 =>
                addr(4 downto 0) <= "11000";
            when 25 =>
                addr(4 downto 0) <= "11001";
            when 26 =>
                addr(4 downto 0) <= "11010";
            when 27 =>
                addr(4 downto 0) <= "11011";
            when 28 =>
                addr(4 downto 0) <= "11100";
            when 29 =>
                addr(4 downto 0) <= "11101";
            when 30 =>
                addr(4 downto 0) <= "11110";
        end case;
        end if;
end process;
end clkosc;
```

6. ตัวค่า address ตำแหน่งของชุด Data *.mif ไฟล์

```
      DEPTH = 31; % Memory depth and width are required     %
      WIDTH = 31; % Enter a decimal number      %

      ADDRESS_RADIX = BIN; % Address and value radixes are optional%
      DATA_RADIX    = BIN; % Enter BIN, DEC, HEX, or OCT; unless  %
                           % otherwise specified, radixes = HEX   %

-- Specify values for addresses, which can be single address or range

      CONTENT
      BEGIN
            00000 :     0100011111010010111100110110000;
            00001 :     0010001110101001011110011011000;
            00010 :     0010010101010100101111001101100;
            00011 :     0000100111101111001111111110110;
            00101 :     0000010001110101001011110011011;
            00110 :     0111110111000111111111100110010;
            00111 :     0011111011110100011001111111001;
            01000 :     0110000111110111011000001110011;
            01001 :     1011000001001111010111101111001;
            01010 :     0010011111011100010101101000011;
            01011 :     0110110111110101110101001011110;
            01100 :     1100100111110111000101011010000;
            01101 :     0110010011101101011110101101000;
            01110 :     1100110110001111011110101001011;
            01111 :     0001100100111011111000101011010;
            10000 :     0000110010011111011100010101101;
            10001 :     1000011000111100001100001010110;
            10010 :     0100111100100111111111000101011;
            10011 :     0101111111001011111100111101010;
            10100 :     1101000011001001111101110001010;
            10101 :     1001011110011011001001000111010;
            10110 :     1011010100110010001010010001110;
            11000 :     0101011011110011011001001000111;
            11001 :     1010100101111001100100100100011;
            11010 :     0010101101000011001101111101110;
            11011 :     1110101001011110011011000001000;
            11100 :     0111010100101111110001101000100;
            11101 :     1100010101101001110011101111101;
            11110 :     1110001011111110001100100111110;
            11111 :     0111000101011010000110010011111;
END ;
```

7. วงจรทำการสเพรดสัญญาณ

```
library ieee;
use   ieee.std_logic_1164.all;

entity spreading is
port ( data      : in  std_logic_vector (30 downto 0);
       pncode    : in  std_logic_vector (30 downto 0);
       spreading : out std_logic_vector(30 downto 0));
end;

ARCHITECTURE spread OF spreading IS
BEGIN
      spreading(30) <= data(30) xor pncode(30);
      spreading(29) <= data(29) xor pncode(29);
      spreading(28) <= data(28) xor pncode(28);
      spreading(27) <= data(27) xor pncode(27);
      spreading(26) <= data(26) xor pncode(26);
      spreading(25) <= data(25) xor pncode(25);
      spreading(24) <= data(24) xor pncode(24);
      spreading(23) <= data(23) xor pncode(23);
      spreading(22) <= data(22) xor pncode(22);
      spreading(21) <= data(21) xor pncode(21);
      spreading(20) <= data(20) xor pncode(20);
      spreading(19) <= data(19) xor pncode(19);
      spreading(18) <= data(18) xor pncode(18);
      spreading(17) <= data(17) xor pncode(17);
      spreading(16) <= data(16) xor pncode(16);
      spreading(15) <= data(15) xor pncode(15);
      spreading(14) <= data(14) xor pncode(14);
      spreading(13) <= data(13) xor pncode(13);
      spreading(12) <= data(12) xor pncode(12);
      spreading(11) <= data(11) xor pncode(11);
      spreading(10) <= data(10) xor pncode(10);
      spreading(9)  <= data(9)  xor pncode(9);
      spreading(8)  <= data(8)  xor pncode(8);
      spreading(7)  <= data(7)  xor pncode(7);
      spreading(6)  <= data(6)  xor pncode(6);
      spreading(5)  <= data(5)  xor pncode(5);
      spreading(4)  <= data(4)  xor pncode(4);
      spreading(3)  <= data(3)  xor pncode(3);
      spreading(2)  <= data(2)  xor pncode(2);
      spreading(1)  <= data(1)  xor pncode(1);
      spreading(0)  <= data(0)  xor pncode(0);
END spread;
```

8. วงจรทำการรวมสัญญาณ CDMA

```
library     ieee;
use   ieee.std_logic_1164.all;

entity multiple_access is
port( data : in  std_logic_vector(30 downto 0);
      CDMA : out std_logic_vector(5 downto 0));
end;

architecture rtl of multiple_access is
component Twocomplement
port( d : in  std_logic;
      y : out std_logic_vector(5 downto 0));
end component;

component Six_adder is
port( a,b : in  std_logic_vector(5 downto 0);
      sum : out std_logic_vector(5 downto 0));
end component;

      signal s1,s2,s3,s4,s5      : std_logic_vector(5 downto 0);
      signal s6,s7,s8,s9,s10     : std_logic_vector(5 downto 0);
      signal s11,s12,s13,s14,s15 : std_logic_vector(5 downto 0);
      signal s16,s17,s18,s19,s20 : std_logic_vector(5 downto 0);
      signal s21,s22,s23,s24,s25 : std_logic_vector(5 downto 0);
      signal s26,s27,s28,s29,s30 : std_logic_vector(5 downto 0);
      signal s31                 : std_logic_vector(5 downto 0);
      signal k1,k2,k3,k4,k5      : std_logic_vector(5 downto 0);
      signal k6,k7,k8,k9,k10     : std_logic_vector(5 downto 0);
      signal k11,k12,k13,k14,k15 : std_logic_vector(5 downto 0);
      signal k16                 : std_logic_vector(5 downto 0);
      signal l1,l2,l3,l4,l5      : std_logic_vector(5 downto 0);
      signal l6,l7,l8            : std_logic_vector(5 downto 0);
      signal m1,m2,m3,m4         : std_logic_vector(5 downto 0);
      signal n1,n2               : std_logic_vector(5 downto 0);

begin
      a1    : Six_adder      port map(n1,n2,CDMA);
      b1    : Six_adder      port map(m1,m2,n1);
      b2    : Six_adder      port map(m3,m4,n2);
      c1    : Six_adder      port map(l1,l2,m1);
      c2    : Six_adder      port map(l3,l4,m2);
      c3    : Six_adder      port map(l5,l6,m3);
      c4    : Six_adder      port map(l7,l8,m4);
      d1    : Six_adder      port map(k1,k2,l1);
      d2    : Six_adder      port map(k3,k4,l2);
      d3    : Six_adder      port map(k5,k6,l3);
      d4    : Six_adder      port map(k7,k8,l4);
      d5    : Six_adder      port map(k9,k10,l5);
      d6    : Six_adder      port map(k11,k12,l6);
      d7    : Six_adder      port map(k13,k14,l7);
      d8    : Six_adder      port map(k15,s31,l8);
      e1    : Six_adder      port map(s1,s2,k1);
      e2    : Six_adder      port map(s3,s4,k2);
      e3    : Six_adder      port map(s5,s6,k3);
```

```
e4    : Six_adder        port map(s7,s8,k4);
e5    : Six_adder        port map(s9,s10,k5);
e6    : Six_adder        port map(s11,s12,k6);
e7    : Six_adder        port map(s13,s14,k7);
e8    : Six_adder        port map(s15,s16,k8);
e9    : Six_adder        port map(s17,s18,k9);
e10   : Six_adder        port map(s19,s20,k10);
e11   : Six_adder        port map(s21,s22,k11);
e12   : Six_adder        port map(s23,s24,k12);
e13   : Six_adder        port map(s25,s26,k13);
e14   : Six_adder        port map(s27,s28,k14);
e15   : Six_adder        port map(s29,s30,k15);
```

---------- ทำการทำ 2'compliments ----------

```
f1    : Twocomplement    port map(data(0),s1);
f2    : Twocomplement    port map(data(1),s2);
f3    : Twocomplement    port map(data(2),s3);
f4    : Twocomplement    port map(data(3),s4);
f5    : Twocomplement    port map(data(4),s5);
f6    : Twocomplement    port map(data(5),s6);
f7    : Twocomplement    port map(data(6),s7);
f8    : Twocomplement    port map(data(7),s8);
f9    : Twocomplement    port map(data(8),s9);
f10   : Twocomplement    port map(data(9),s10);
f11   : Twocomplement    port map(data(10),s11);
f12   : Twocomplement    port map(data(11),s12);
f13   : Twocomplement    port map(data(12),s13);
f14   : Twocomplement    port map(data(13),s14);
f15   : Twocomplement    port map(data(14),s15);
f16   : Twocomplement    port map(data(15),s16);
f17   : Twocomplement    port map(data(16),s17);
f18   : Twocomplement    port map(data(17),s18);
f19   : Twocomplement    port map(data(18),s19);
f20   : Twocomplement    port map(data(19),s20);
f21   : Twocomplement    port map(data(20),s21);
f22   : Twocomplement    port map(data(21),s22);
f23   : Twocomplement    port map(data(22),s23);
f24   : Twocomplement    port map(data(23),s24);
f25   : Twocomplement    port map(data(24),s25);
f26   : Twocomplement    port map(data(25),s26);
f27   : Twocomplement    port map(data(26),s27);
f28   : Twocomplement    port map(data(27),s28);
f29   : Twocomplement    port map(data(28),s29);
f30   : Twocomplement    port map(data(29),s30);
f31   : Twocomplement    port map(data(30),s31);
end;
```

8.1. วงจรย่อย(ทำ 2's compliment)

```
LIBRARY     ieee;
USE ieee.std_logic_1164.ALL;

entity Twocomplement is
port(
      d   : in    std_logic;
      y     : out std_logic_vector(5 downto 0));
end Twocomplement;

architecture Mapper of Twocomplement is
begin
      y <=  "000001" when (d='0') else
            "111111" when (d='1') else
            "000000";
end Mapper;
```

8.2. วงจรย่อย(ทำการบวกข้อมูล 6 บิตแบบไม่มีตัวทด)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity multip6 is
port(a,b  : in  std_logic_vector(5 downto 0);
     mult : out std_logic_vector(5 downto 0));
end;

ARCHITECTURE structa OF multip6 IS
SIGNAL s01 : std_logic_vector(11 downto 0);
begin
      s01   <= a*b;
mult(5 downto 0) <= s01(5 downto 0);
end;
```

9. วงจรรวมทำการกู้สัญญาณกลับโดยการคูณ

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity PNtcom1 is
port(PN                          : in  std_logic_vector(30 downto 0);
     CDMA                        : in  std_logic_vector(5  downto 0);
     Ur00                        : out std_logic_vector(10 downto 0));
     Ur01,Ur02,Ur03,Ur04,Ur05    : out std_logic_vector(10 downto 0);
     Ur06,Ur07,Ur08,Ur09,Ur10    : out std_logic_vector(10 downto 0);
     Ur11,Ur12,Ur13,Ur14,Ur15    : out std_logic_vector(10 downto 0);
     Ur16,Ur17,Ur18,Ur19,Ur20    : out std_logic_vector(10 downto 0);
     Ur21,Ur22                   : out std_logic_vector(10 downto 0));
end;

ARCHITECTURE structa OF PNtcom1 IS
      SIGNAL p01,p02,p03,p04,p05,CD  : std_logic_vector(5 downto 0);
      SIGNAL p06,p07,p08,p09,p10     : std_logic_vector(5 downto 0);
      SIGNAL p11,p12,p13,p14,p15     : std_logic_vector(5 downto 0);
      SIGNAL p16,p17,p18,p19,p20     : std_logic_vector(5 downto 0);
      SIGNAL p21,p22,p23,p24,p25     : std_logic_vector(5 downto 0);
      SIGNAL p26,p27,p28,p29,p30,p00 : std_logic_vector(5 downto 0);
      SIGNAL U00                     : std_logic_vector(5 downto 0);
      SIGNAL U01,U02,U03,U04,U05     : std_logic_vector(5 downto 0);
      SIGNAL U06,U07,U08,U09,U10     : std_logic_vector(5 downto 0);
      SIGNAL U11,U12,U13,U14,U15     : std_logic_vector(5 downto 0);
      SIGNAL U16,U17,U18,U19,U20     : std_logic_vector(5 downto 0);
      SIGNAL U21,U22,U23,U24,U25     : std_logic_vector(5 downto 0);
      SIGNAL U26,U27,U28,U29,U30     : std_logic_vector(5 downto 0);


      component Twocomplement
            port( d     : in  std_logic;
                  y     : out std_logic_vector(5 downto 0));
      end component;

      component multip6
            port( a,b   : in  std_logic_vector(5 downto 0);
                  mult  : out std_logic_vector(5 downto 0));
      end component;

      component Sign1
            port( d     : in  std_logic;
                  y     : out std_logic_vector(5 downto 0));
      end component;
```

```
begin

---------------------------- Two Complement --------------------------

            CD <= CDMA;

            TPN30  : Twocomplement  port map(PN(30),p30);
            TPN29  : Twocomplement  port map(PN(29),p29);
            TPN28  : Twocomplement  port map(PN(28),p28);
            TPN27  : Twocomplement  port map(PN(27),p27);
            TPN26  : Twocomplement  port map(PN(26),p26);
            TPN25  : Twocomplement  port map(PN(25),p25);
            TPN24  : Twocomplement  port map(PN(24),p24);
            TPN23  : Twocomplement  port map(PN(23),p23);
            TPN22  : Twocomplement  port map(PN(22),p22);
            TPN21  : Twocomplement  port map(PN(21),p21);
            TPN20  : Twocomplement  port map(PN(20),p20);
            TPN19  : Twocomplement  port map(PN(19),p19);
            TPN18  : Twocomplement  port map(PN(18),p18);
            TPN17  : Twocomplement  port map(PN(17),p17);
            TPN16  : Twocomplement  port map(PN(16),p16);
            TPN15  : Twocomplement  port map(PN(15),p15);
            TPN14  : Twocomplement  port map(PN(14),p14);
            TPN13  : Twocomplement  port map(PN(13),p13);
            TPN12  : Twocomplement  port map(PN(12),p12);
            TPN11  : Twocomplement  port map(PN(11),p11);
            TPN10  : Twocomplement  port map(PN(10),p10);
            TPN09  : Twocomplement  port map(PN(09),p09);
            TPN08  : Twocomplement  port map(PN(08),p08);
            TPN07  : Twocomplement  port map(PN(07),p07);
            TPN06  : Twocomplement  port map(PN(06),p06);
            TPN05  : Twocomplement  port map(PN(05),p05);
            TPN04  : Twocomplement  port map(PN(04),p04);
            TPN03  : Twocomplement  port map(PN(03),p03);
            TPN02  : Twocomplement  port map(PN(02),p02);
            TPN01  : Twocomplement  port map(PN(01),p01);
            TPN00  : Twocomplement  port map(PN(00),p00);

----------------------------- Multiply -------------------------------

            DeU30         : multip6  port map(p30,CD,U30);
            DeU29         : multip6  port map(p29,CD,U29);
            DeU28         : multip6  port map(p28,CD,U28);
            DeU27         : multip6  port map(p27,CD,U27);
            DeU26         : multip6  port map(p26,CD,U26);
            DeU25         : multip6  port map(p25,CD,U25);
            DeU24         : multip6  port map(p24,CD,U24);
            DeU23         : multip6  port map(p23,CD,U23);
            DeU22         : multip6  port map(p22,CD,U22);
            DeU21         : multip6  port map(p21,CD,U21);
            DeU20         : multip6  port map(p20,CD,U20);
            DeU19         : multip6  port map(p19,CD,U19);
            DeU18         : multip6  port map(p18,CD,U18);
            DeU17         : multip6  port map(p17,CD,U17);
            DeU16         : multip6  port map(p16,CD,U16);
            DeU15         : multip6  port map(p15,CD,U15);
            DeU14         : multip6  port map(p14,CD,U14);
```

```
DeU13          : multip6  port map(p13,CD,U13);
DeU12          : multip6  port map(p12,CD,U12);
DeU11          : multip6  port map(p11,CD,U11);
DeU10          : multip6  port map(p10,CD,U10);
DeU09          : multip6  port map(p09,CD,U09);
DeU08          : multip6  port map(p08,CD,U08);
DeU07          : multip6  port map(p07,CD,U07);
DeU06          : multip6  port map(p06,CD,U06);
DeU05          : multip6  port map(p05,CD,U05);
DeU04          : multip6  port map(p04,CD,U04);
DeU03          : multip6  port map(p03,CD,U03);
DeU02          : multip6  port map(p02,CD,U02);
DeU01          : multip6  port map(p01,CD,U01);
DeU00          : multip6  port map(p00,CD,U00);

------------------------------ Sign ------------------------------

De30    : Sign1    port map(U30(5),Ur30(10 downto 5));
De29    : Sign1    port map(U29(5),Ur29(10 downto 5));
De28    : Sign1    port map(U28(5),Ur28(10 downto 5));
De27    : Sign1    port map(U27(5),Ur27(10 downto 5));
De26    : Sign1    port map(U26(5),Ur26(10 downto 5));
De25    : Sign1    port map(U25(5),Ur25(10 downto 5));
De24    : Sign1    port map(U24(5),Ur24(10 downto 5));
De23    : Sign1    port map(U23(5),Ur23(10 downto 5));
De22    : Sign1    port map(U22(5),Ur22(10 downto 5));
De21    : Sign1    port map(U21(5),Ur21(10 downto 5));
De20    : Sign1    port map(U20(5),Ur20(10 downto 5));
De19    : Sign1    port map(U19(5),Ur19(10 downto 5));
De18    : Sign1    port map(U18(5),Ur18(10 downto 5));
De17    : Sign1    port map(U17(5),Ur17(10 downto 5));
De16    : Sign1    port map(U16(5),Ur16(10 downto 5));
De15    : Sign1    port map(U15(5),Ur15(10 downto 5));
De14    : Sign1    port map(U14(5),Ur14(10 downto 5));
De13    : Sign1    port map(U13(5),Ur13(10 downto 5));
De12    : Sign1    port map(U12(5),Ur12(10 downto 5));
De11    : Sign1    port map(U11(5),Ur11(10 downto 5));
De10    : Sign1    port map(U10(5),Ur10(10 downto 5));
De09    : Sign1    port map(U09(5),Ur09(10 downto 5));
De08    : Sign1    port map(U08(5),Ur08(10 downto 5));
De07    : Sign1    port map(U07(5),Ur07(10 downto 5));
De06    : Sign1    port map(U06(5),Ur06(10 downto 5));
De05    : Sign1    port map(U05(5),Ur05(10 downto 5));
De04    : Sign1    port map(U04(5),Ur04(10 downto 5));
De03    : Sign1    port map(U03(5),Ur03(10 downto 5));
De02    : Sign1    port map(U02(5),Ur02(10 downto 5));
De01    : Sign1    port map(U01(5),Ur01(10 downto 5));
De00    : Sign1    port map(U00(5),Ur00(10 downto 5));
```

```
---------------------------- Map Port ---------------------------------

      Ur30(4 downto 0) <= U30(4 downto 0);
      Ur29(4 downto 0) <= U29(4 downto 0);
      Ur28(4 downto 0) <= U28(4 downto 0);
      Ur27(4 downto 0) <= U27(4 downto 0);
      Ur26(4 downto 0) <= U26(4 downto 0);
      Ur25(4 downto 0) <= U25(4 downto 0);
      Ur24(4 downto 0) <= U24(4 downto 0);
      Ur23(4 downto 0) <= U23(4 downto 0);
      Ur22(4 downto 0) <= U22(4 downto 0);
      Ur21(4 downto 0) <= U21(4 downto 0);
      Ur20(4 downto 0) <= U20(4 downto 0);
      Ur19(4 downto 0) <= U19(4 downto 0);
      Ur18(4 downto 0) <= U18(4 downto 0);
      Ur17(4 downto 0) <= U17(4 downto 0);
      Ur16(4 downto 0) <= U16(4 downto 0);
      Ur15(4 downto 0) <= U15(4 downto 0);
      Ur14(4 downto 0) <= U14(4 downto 0);
      Ur13(4 downto 0) <= U13(4 downto 0);
      Ur12(4 downto 0) <= U12(4 downto 0);
      Ur11(4 downto 0) <= U11(4 downto 0);
      Ur10(4 downto 0) <= U10(4 downto 0);
      Ur09(4 downto 0) <= U09(4 downto 0);
      Ur08(4 downto 0) <= U08(4 downto 0);
      Ur07(4 downto 0) <= U07(4 downto 0);
      Ur06(4 downto 0) <= U06(4 downto 0);
      Ur05(4 downto 0) <= U05(4 downto 0);
      Ur04(4 downto 0) <= U04(4 downto 0);
      Ur03(4 downto 0) <= U03(4 downto 0);
      Ur02(4 downto 0) <= U02(4 downto 0);
      Ur01(4 downto 0) <= U01(4 downto 0);
      Ur00(4 downto 0) <= U00(4 downto 0);

end;
```

9. 1 วงจรย่อย(ทำการคูณแบบ 6 บิต)

```
      library ieee;
      use ieee.std_logic_1164.all;
      use ieee.std_logic_unsigned.all;
      entity multip6 is
            port(a,b    : in  std_logic_vector(5 downto 0);
                 mult   : out std_logic_vector(5 downto 0));
            end;
      ARCHITECTURE structa OF multip6 IS
      SIGNAL       s01  : std_logic_vector(11 downto 0);
      begin
            s01   <= a*b;
            mult(5 downto 0) <= s01(5 downto 0);
end;
```

10. วงจรทำการหารความถี่เพื่อตัวควบคุมทางด้านรับ

```
library ieee;
use ieee.std_logic_1164.all;

entity divclkcount is
port(sys_clk : in std_logic;
      Pn_clk : out std_logic);
end;

architecture clkosc of divclkcount is
signal count:integer range 0 to 1;
begin
process(sys_clk)
begin
if sys_clk'event and sys_clk = '1' then
      if   count /= 1 then
           count <= count+1;
      else count <=0;
      end if;
      if count < 1 then
            Pn_clk <= '1';
      else Pn_clk <= '0';
      end if;
end if;
end process;
end clkosc;
```

11. วงจรควบคุมการทำการบวกค่าสะสมแบบเป็นคาบ

```
library     ieee;
use         ieee.std_logic_1164.all;
use         ieee.std_logic_unsigned.all;

entity      Controlacc is
port(clkCont                : in std_logic;
     l_acc,reset,load1  : out std_logic);
end Controlacc;

ARCHITECTURE con OF Controlacc IS
      signal start:integer    range 0 to 1;
      signal count:integer    range 0 to 123;
      signal coun1:integer    range 0 to 123;
      signal countclk:integer range 0 to 3;

BEGIN

-- generating reset, load1 signal --

generating: Process(clkCont)
Begin
if clkCont'event and clkCont='1' then
      if   count /= 123 then
           count <= count+1;
      else count <=0;
      end if;

-- load data to buffer --

      if    count=122 then
            load1<='1';
      else
            load1<='0';
      end if;

-- reset the accumulator --
      if  count=123 then
            reset<='1';
      else
            reset<='0';
      end if;
```

```
                    -- generator --

        if (count=1)   or (count=5)   or (count=9)   or
           (count=13)  or (count=17)  or (count=21)  or
           (count=25)  or (count=29)  or (count=33)  or
           (count=37)  or (count=41)  or (count=45)  or
           (count=49)  or (count=53)  or (count=57)  or
           (count=61)  or (count=65)  or (count=69)  or
           (count=73)  or (count=77)  or (count=81)  or
           (count=85)  or (count=89)  or (count=93)  or
           (count=97)  or (count=101) or (count=105) or
           (count=109) or (count=113) or (count=117) or
           (count=121) then
                    l_acc<='1';
        else
                    l_acc<='0';
        end if;
        end if;
end process generating;
END con;
```

12. วงจรตัวพักค่า

```
library ieee;
use   ieee.std_logic_1164.all;

entity      buff is
port (acc_reg  : in std_logic_vector(10 downto 0);
      load1    : in std_logic;
      buff_reg : buffer std_logic_vector(10 downto 0));
end;

architecture rtl of buff is
begin
process(load1)
begin
        if load1'event and load1='1'  then
              buff_reg<=acc_reg;
        end if;
end process;
end;
```

13. วงจรตัวบวกสะสมค่า

```
library     ieee;
use   ieee.std_logic_1164.all;

entity      add_acc     is
      port(ip : in  std_logic_vector(10 downto 0);
             op : out std_logic_vector(10 downto 0);
             lacc, racc      : in std_logic);
end;

architecture rtl of add_acc is

      component add_sub
         port(   a : in  std_logic_vector(10 downto 0);
                 b : in  std_logic_vector(10 downto 0);
                  sum : out std_logic_vector(10 downto 0));
      end component;

      component   acc
                  port (sum : in std_logic_vector (10 downto 0);
           load,reset : in std_logic;
                 acc_reg : out std_logic_vector(10 downto 0));
      end component;

      signal        add_out:std_logic_vector(10 downto 0);
      signal        add_in :std_logic_vector(10 downto 0);
      signal        acc_out:std_logic_vector(10 downto 0);
      signal   acc_in :std_logic_vector(10 downto 0);

      begin

         add_in(10 downto 0)<=acc_out(10 downto 0);

-------------------------------------------------------------------

         acc_in(10 downto 0)<=add_out(10 downto 0);

-------------------------------------------------------------------

      summing     : add_sub port map(ip,add_in,add_out);
      accumulator : acc     port map(acc_in,lacc,racc,acc_out);

       op(10 downto 0)<=acc_out(10 downto 0);

-------------------------------------------------------------------
end;
```

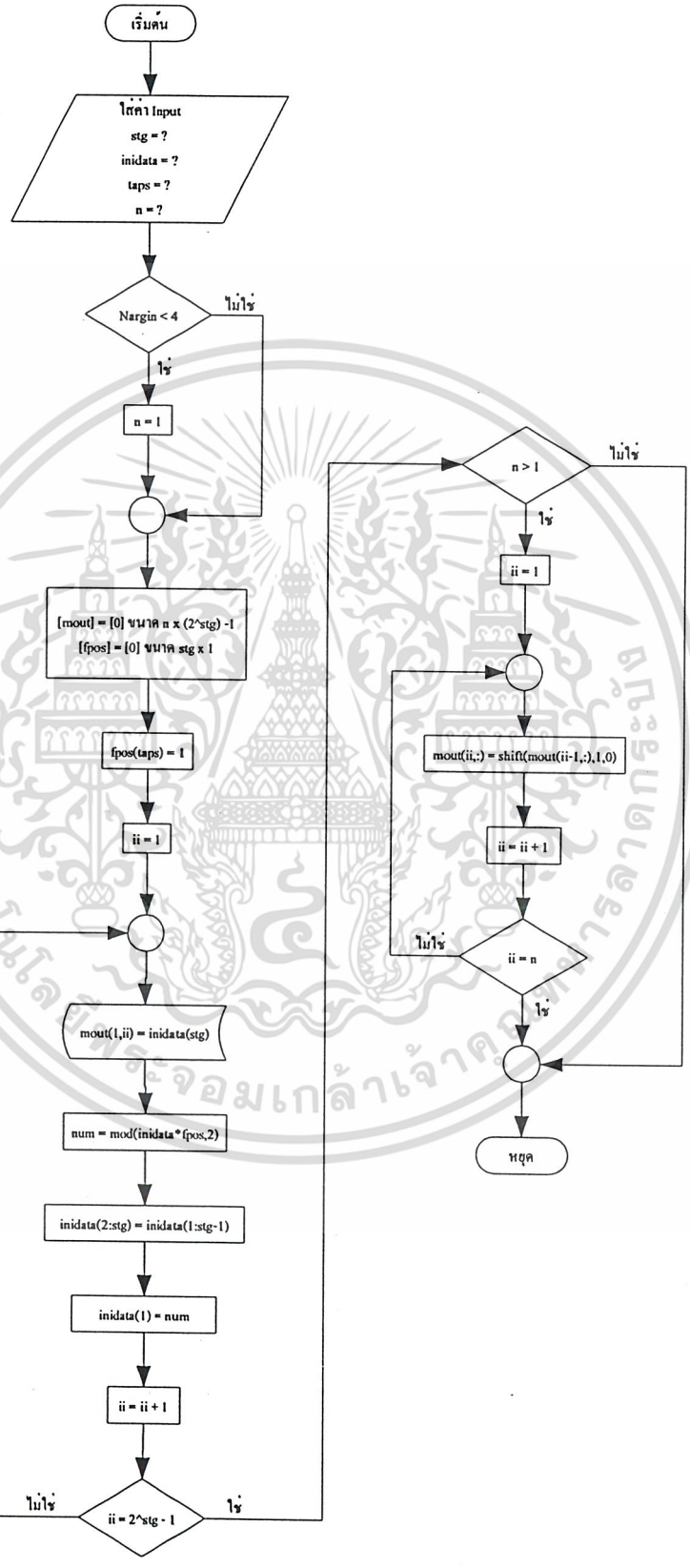
13. 1 วงจรย่อย (บวกสะสมค่าทำหน้าที่ในการบวก)

```
library     ieee;
use   ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity      add_sub is
port(   a : in std_logic_vector(10 downto 0);
      b : in std_logic_vector(10 downto 0);
        sum : out std_logic_vector(10 downto 0));
end;

architecture rtl of add_sub is
      signal temp : std_logic_vector(10 downto 0);
begin
      temp<=a;
      process(a,b)
      begin
            sum<=b+temp;
      end process;
end rtl;
```

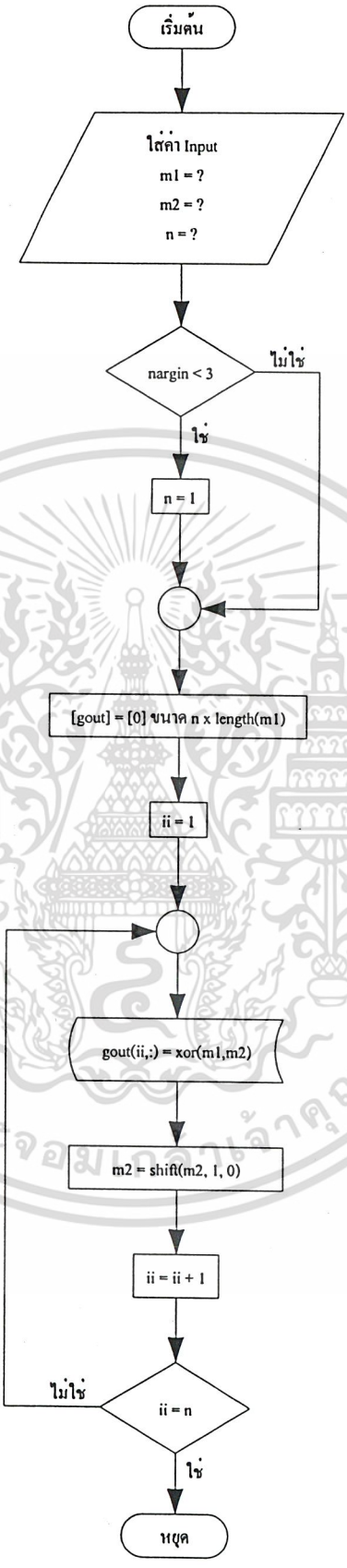
13.2 วงจรย่อย (ทำหน้าที่ในการป้อนกลับ);

```
use   ieee.std_logic_1164.all;

entity      acc is
port ( sum : in std_logic_vector (10 downto 0);
       load,reset : in std_logic;
       acc_reg : out std_logic_vector(10 downto 0));
end;

architecture rtl of acc is
begin
process(load,reset)
begin
        if reset = '1' then
              acc_reg <= (others=>'0');
        elsif load'event and load='1' then
              acc_reg<=sum;
        end if;
end process;
end;
```

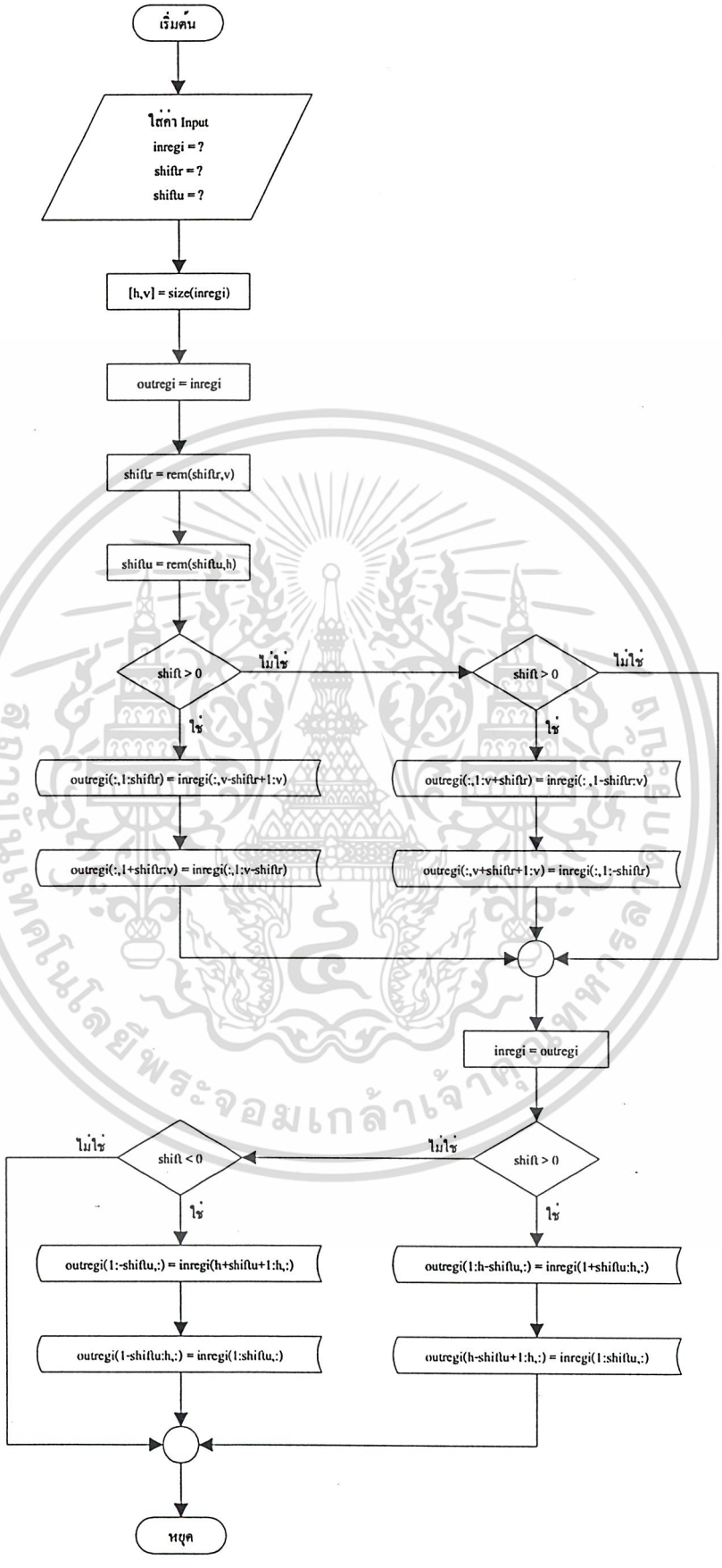
ภาคผนวก ค.



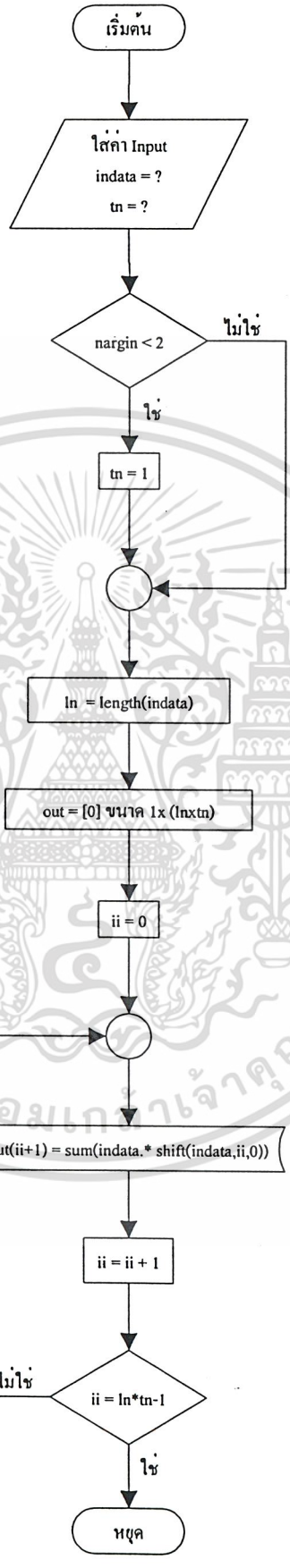
แผนภาพที่ 1ค แผนภาพการทำงานของฟังก์ชัน mseq



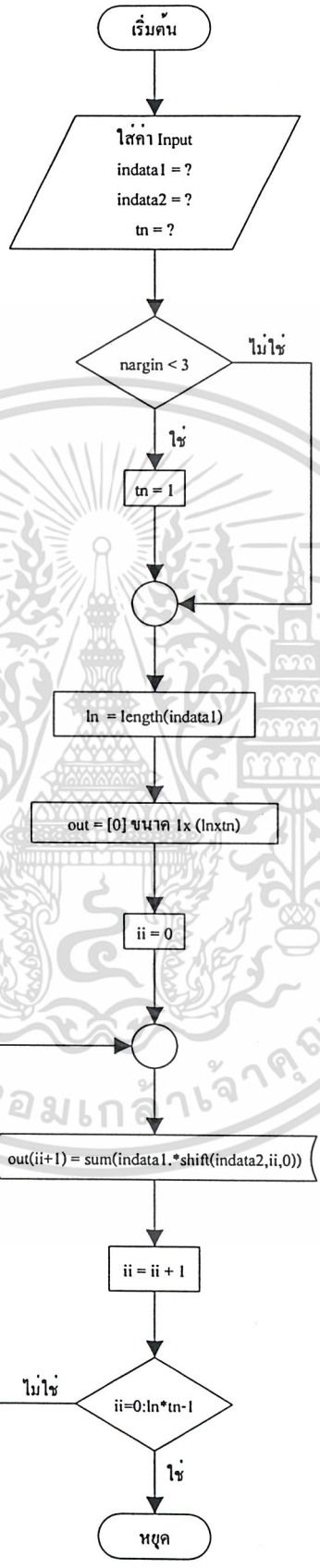
แผนภาพที่ 2ค แผนภาพการทำงานของฟังก์ชัน goldseq



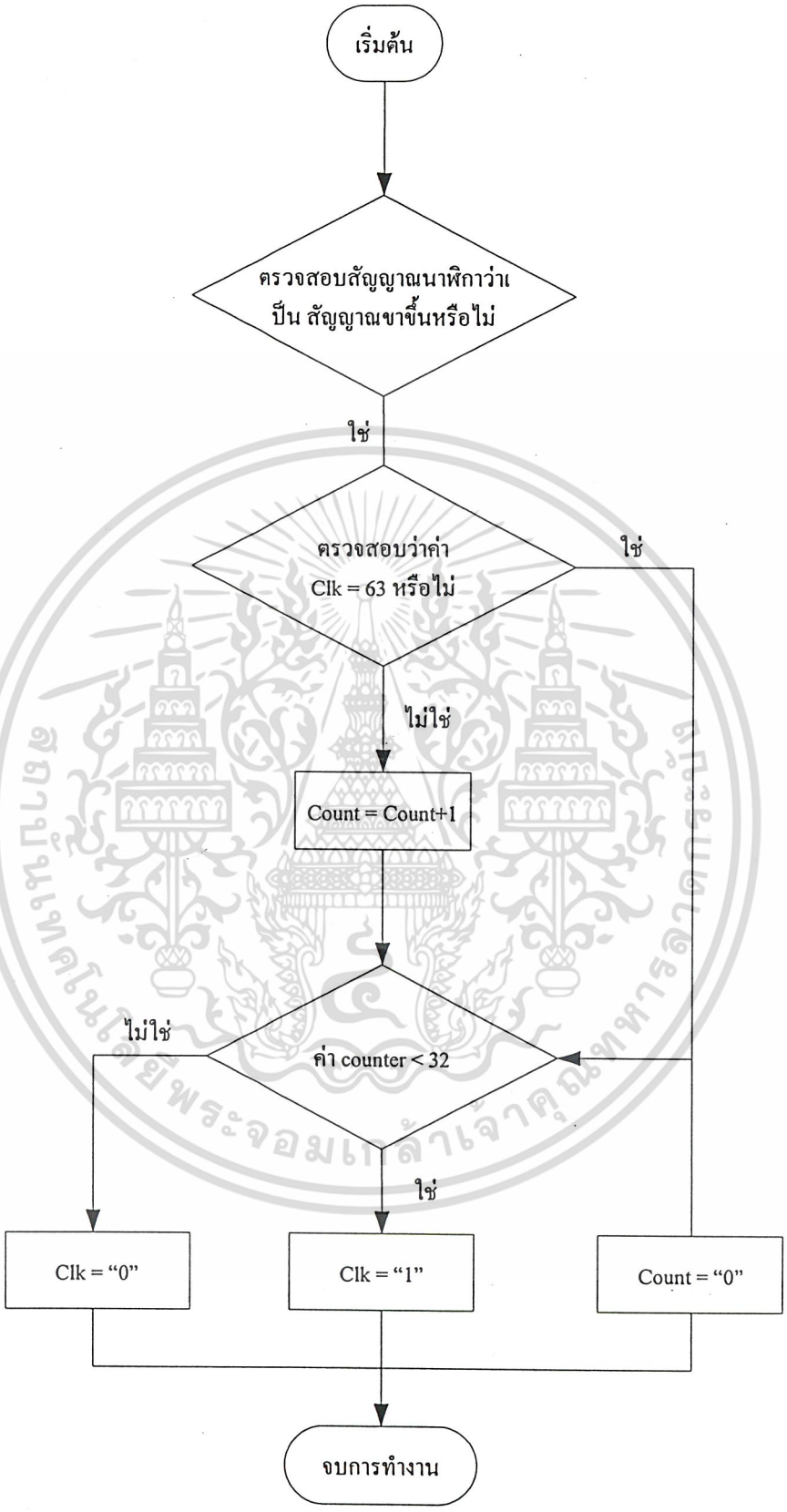
แผนภาพที่ 3ค แผนภาพการทำงานของฟังก์ชัน shift



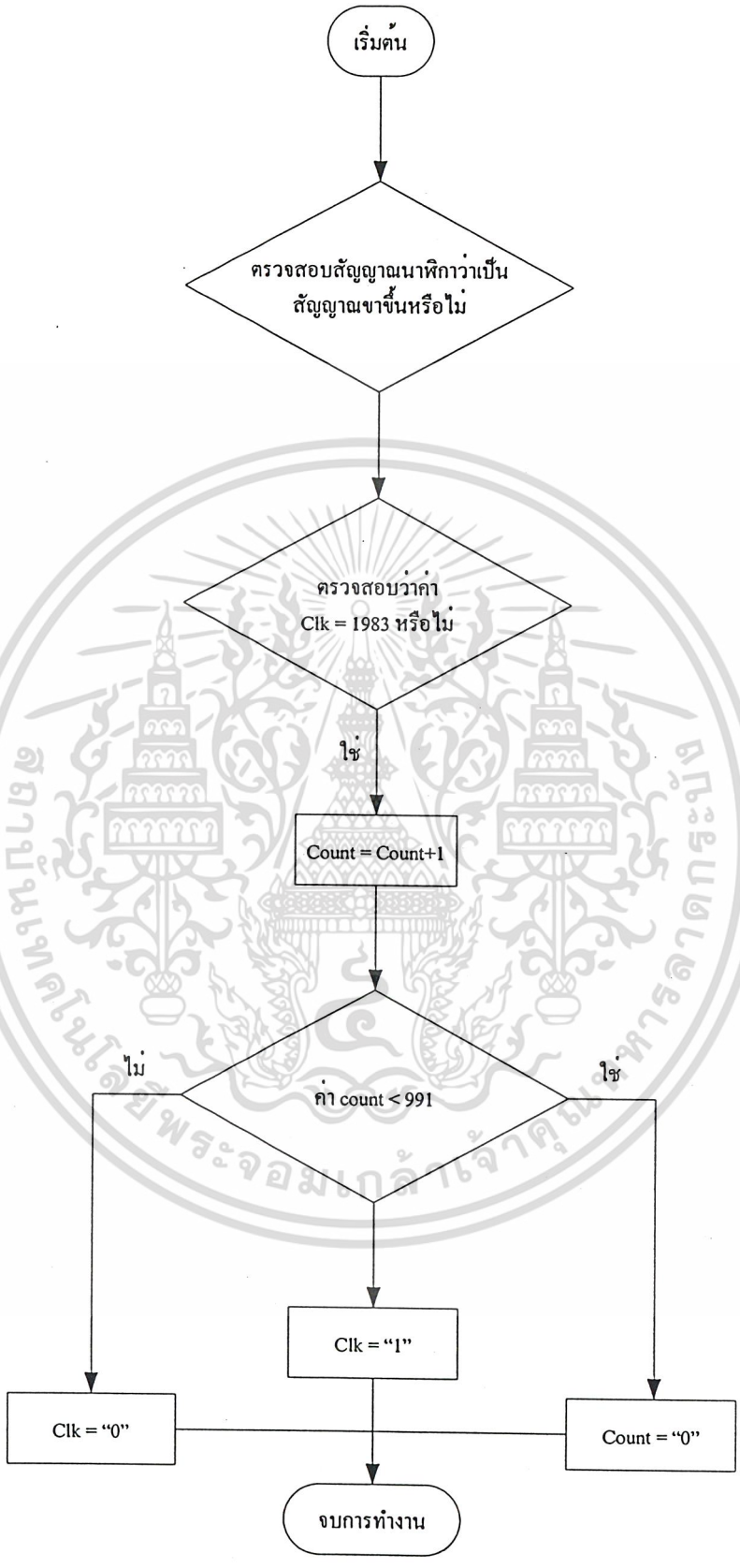
แผนภาพที่ 4ค แผนภาพการทำงานของฟังก์ชัน autocorr



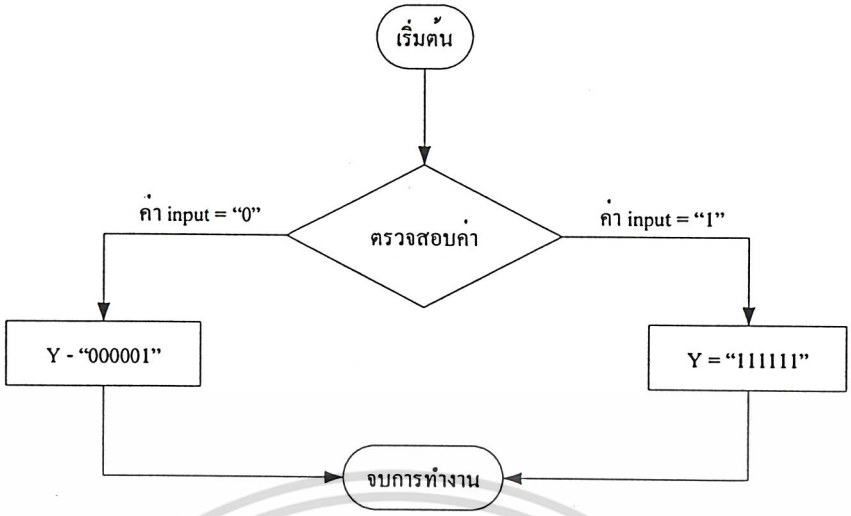
แผนภาพที่ 5ค แผนภาพการทำงานของฟังก์ชัน crosscorr



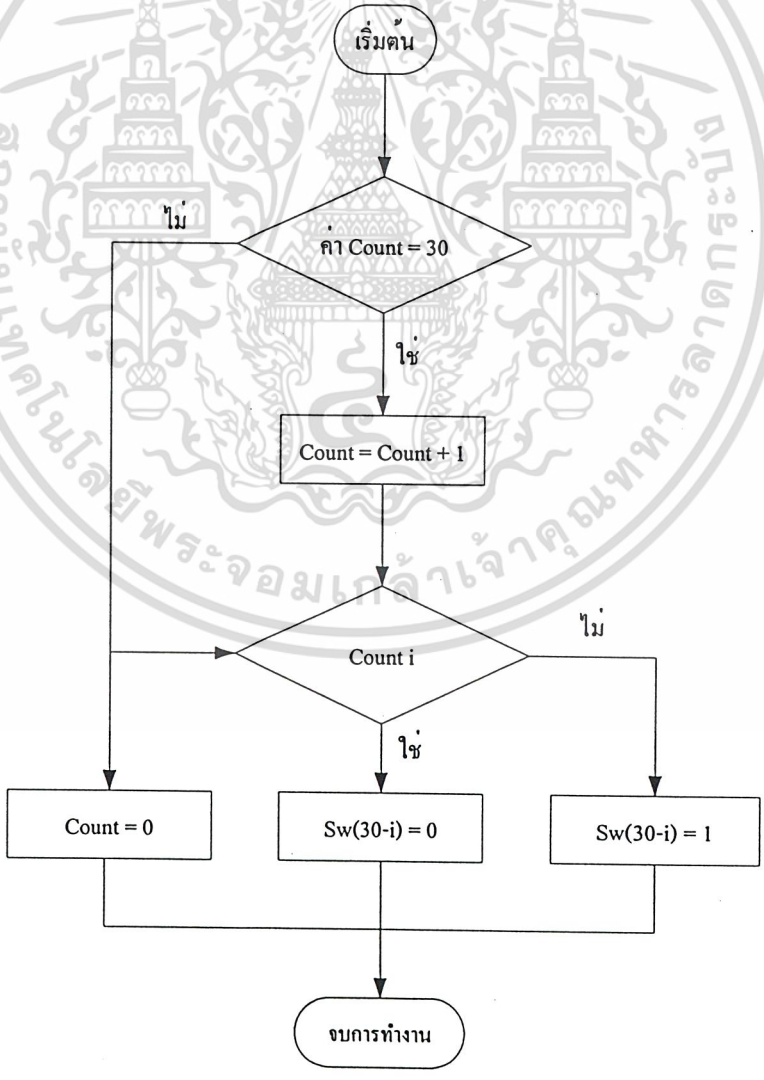
แผนภาพที่ 6ค แผนภาพการทำงานวงจรหาร 64



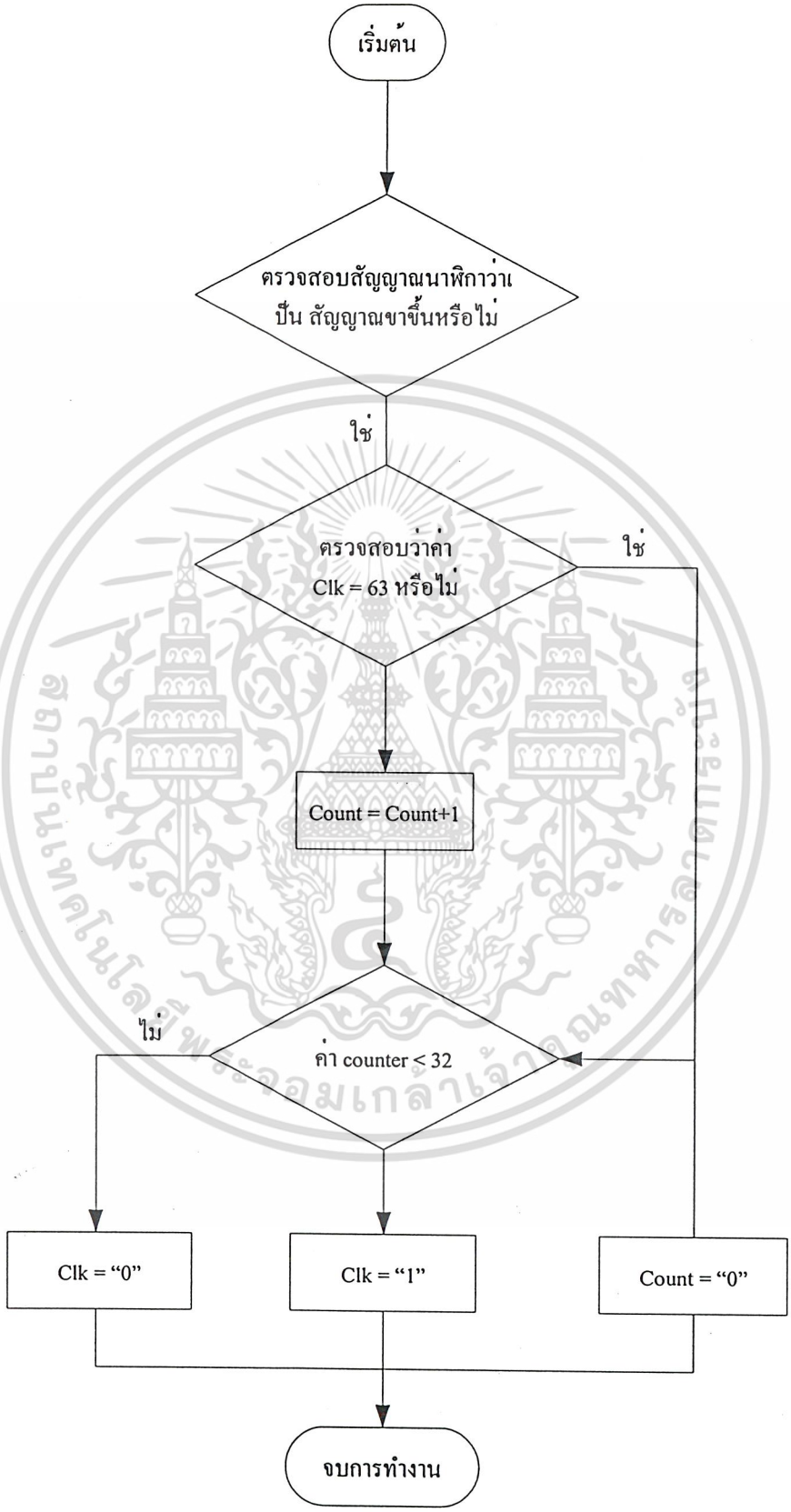
แผนภาพที่ 7ค แผนภาพการทำงานวงจรหาร 2



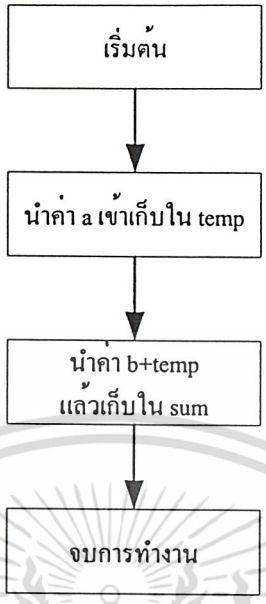
แผนภาพที่ 8ค แผนภาพการทำงานวงจร 2's compliment



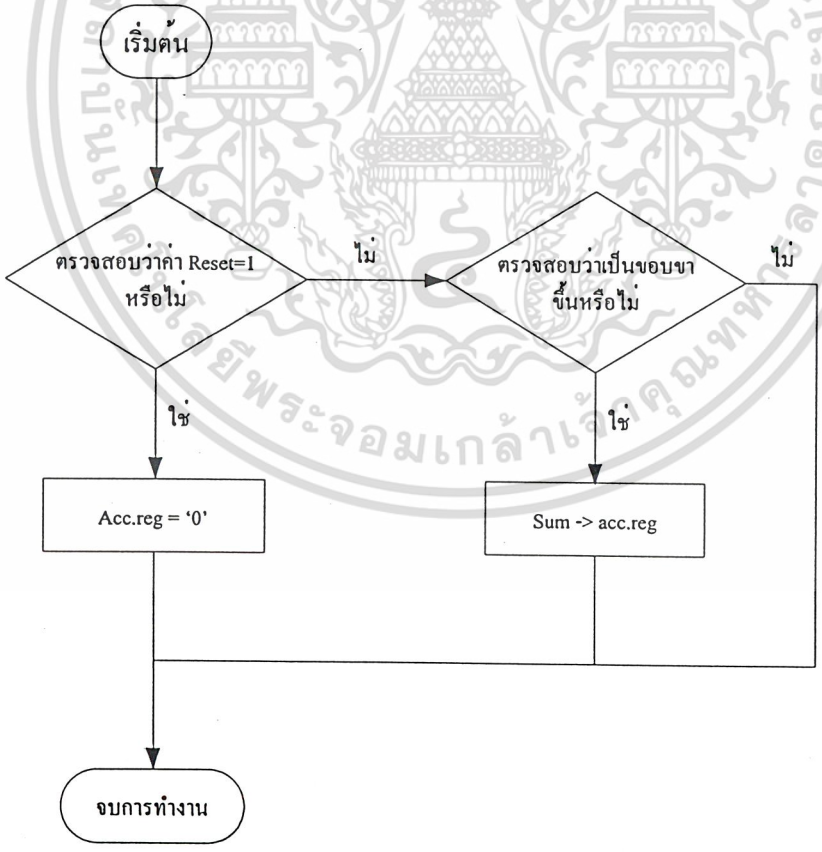
แผนภาพที่ 9ค แผนภาพการทำงาน Counter 30



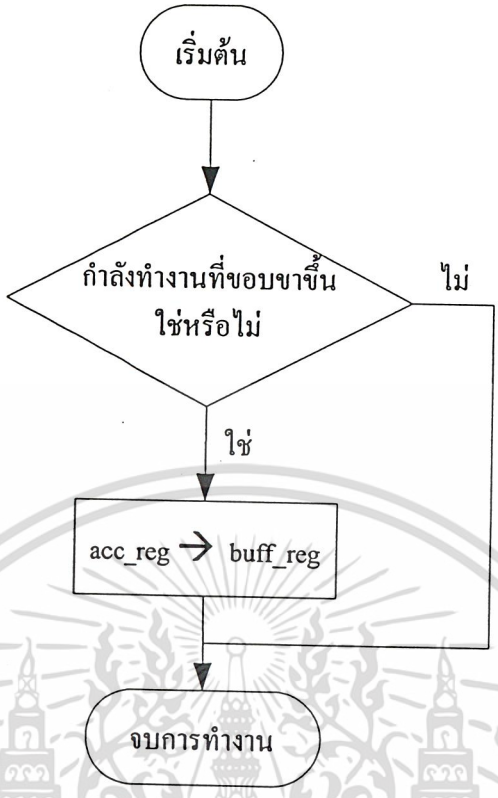
แผนภาพที่ 10ค แผนภาพการทำงานวงจรหาร 64 ด้านรับ



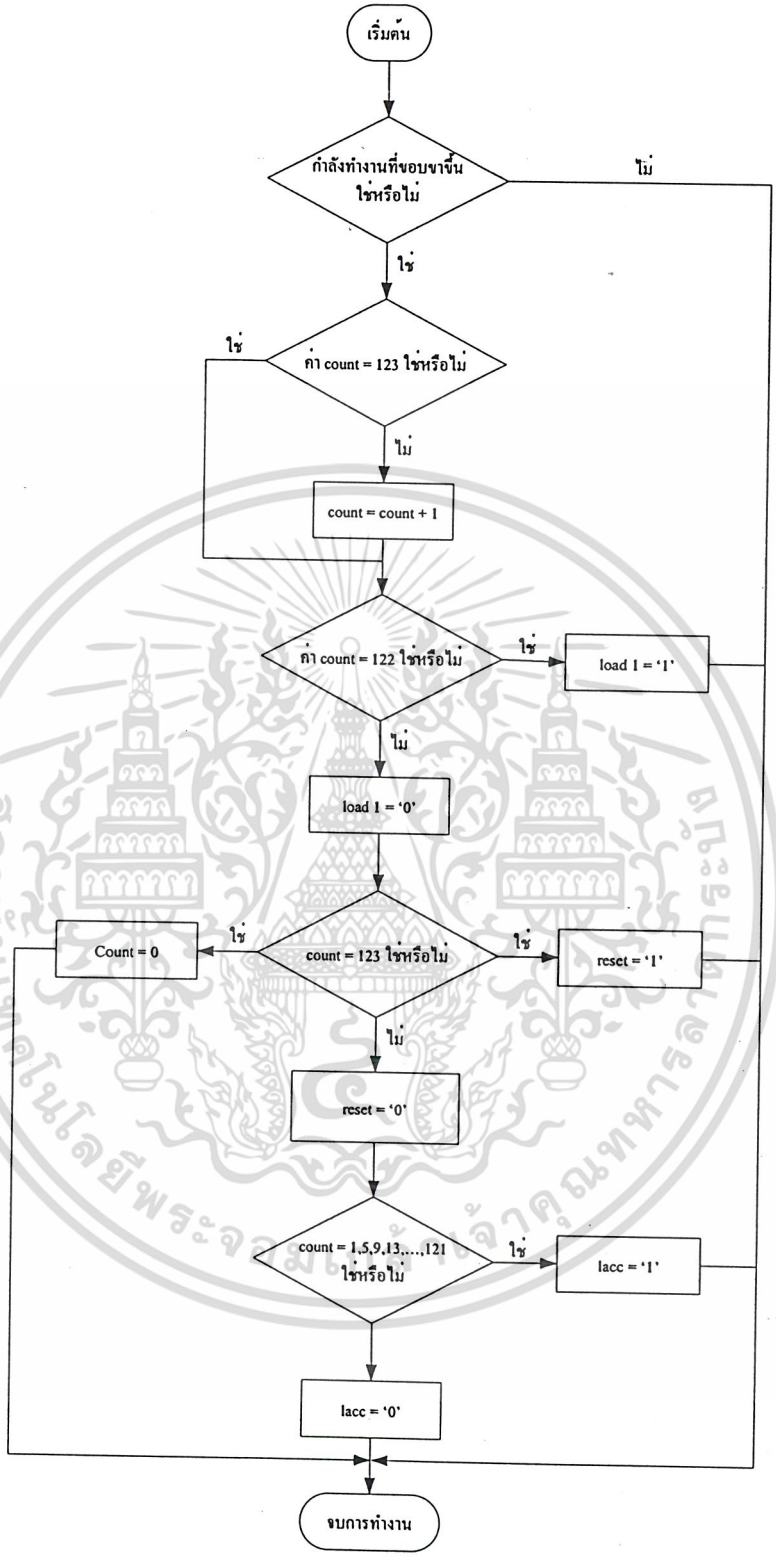
แผนภาพที่ 11ค แผนภาพการทำงานของวงจรป้อนกลับ



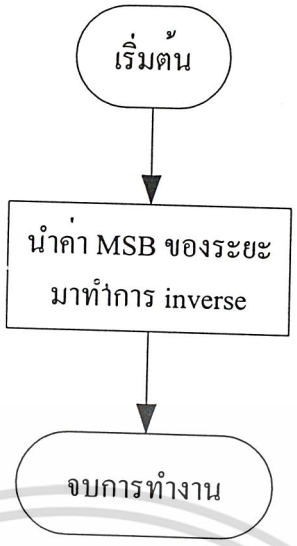
แผนภาพที่ 12ค แผนภาพการทำงานของวงจรบวกป้อนกลับ



แผนภาพที่ 13ค แผนภาพการทำงานของวงจร ป้อนค่าสู่ Buffer



แผนภาพที่ 14ค แผนภาพการทำงานของวงจรคอนโทรลยูนิต



แผนภาพที่ 15ค แผนภาพการทำงานบิต ดิซิชั่น