

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบสร้างชุดถอดรหัส Viterbi ขนาดเล็ก
สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3

A SMALL ARCHITECTURE OF VITERBI DECODER
FOR 3GPP MOBILE SYSTEM



ชัยวัฒน์ แก้วสาย
CHAIWAT KEAWSAI

เลขหมู่.....
เลขทะเบียน 49587
วัน, เดือน, ปี 24 ก.พ. 2547

.b.....
.i.....

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2546

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ISBN 974-324-898-6

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

**A SMALL ARCHITECTURE OF VITERBI DECODER
FOR 3GPP MOBILE SYSTEM**



**A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN INFORMATION ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG**

2003

ISBN 974-324-898-6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



COPYRIGHT 2003

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์

การออกแบบสร้างชุดถอดรหัส Viterbi ขนาดเล็กสำหรับ
ระบบโทรศัพท์เคลื่อนที่ยุคที่ 3

นักศึกษา

นายชัยวัฒน์ แก้วสาย

รหัสนักศึกษา

44061826

ปริญญา

วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา

วิศวกรรมสารสนเทศ

พ.ศ.

2546

อาจารย์ผู้ควบคุมวิทยานิพนธ์

ผศ.ดร.อรรถสิทธิ์ หล้าสกุล

อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม

ดร.เกียรติศักดิ์ ศรีพิมานวัฒน์

บทคัดย่อ

การเข้ารหัสคอนโวลูชันและการถอดรหัส Viterbi เป็นเทคนิคหนึ่งที่ใช้แก้ไขความผิดพลาดจากการรับส่งข้อมูลในระบบสื่อสารเชิงดิจิทัล ซึ่งสำหรับการเพิ่มประสิทธิภาพของการสื่อสารดังกล่าวในช่องสัญญาณเสียง (Voice Services) ของระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นั้น ได้นำการเข้ารหัสคอนโวลูชันและการถอดรหัส Viterbi นี้มาใช้งานอย่างเป็นทางการตามมาตรฐาน TS 25.212 ของ 3GPP สำหรับการสร้างชุดถอดรหัส Viterbi ตามมาตรฐานดังกล่าวจะอยู่ในรูปของฮาร์ดแวร์วงจรรวมที่มีพื้นฐานในข้อจำกัดด้านความเร็ว การใช้พื้นที่ (Silicon Area) และการใช้พลังงาน (Power Consumption) อันเป็นสิ่งที่สำคัญที่แสดงถึงประสิทธิภาพของอุปกรณ์ที่ได้ออกแบบโดยทั่วไปวงจรรวมของชุดถอดรหัส Viterbi นี้มีโครงสร้างสองรูปแบบคือ โครงสร้างแบบขนานและอนุกรม ซึ่งโครงสร้างแบบขนานนั้นให้ความเร็วในการถอดรหัสสูงแต่ใช้พื้นที่และพลังงานมากกว่าแบบอนุกรม จากข้อจำกัดโดยทั่วไปดังกล่าววิทยานิพนธ์นี้จึงได้กำหนดขอบเขตที่มุ่งเน้นถึงโครงสร้างแบบใหม่ที่ทำงานได้รวดเร็วแต่ใช้ขนาดพื้นที่สร้างน้อย และนำเสนอการสร้างชุดถอดรหัส Viterbi โครงสร้างอนุกรมแบบใหม่สำหรับถอดรหัสคอนโวลูชันที่มี Constraint Length (K) เท่ากับ 9 (256 States) อัตราการเข้ารหัส $1/2$ และ $1/3$ สำหรับ Generator Polynomial (561, 753)₈ และ (557, 663, 711)₈ ตามลำดับ มีอัตราเร็วในการถอดรหัสที่ค่า 2 Mbps ตามมาตรฐาน TS 25.212 ของ 3GPP การออกแบบชุดถอดรหัสนี้ใช้โครงสร้างอนุกรมแบบ 4 Add-Compare-Select (ACS) และมีการปรับปรุงส่วน Survivor Memory Unit แบบ Register Exchange ที่ให้ความเร็วในการทำงานที่สูงแต่มีโครงสร้างขนาดใหญ่ เพื่อให้มีขนาดเล็กลงและเหมาะสมสำหรับโครงสร้างแบบ 4 ACS ซึ่งการปรับปรุงดังกล่าวได้ส่งผลให้การใช้อุปกรณ์ในเอฟพีจีเอ Xilinx Vertex II มีจำนวนน้อยและให้อัตราเร็วในการถอดรหัสสูงถึง 3.46 Mbps ซึ่งสามารถใช้งานในระบบ 3GPP ได้ และสามารถปรับปรุงให้ใช้กับระบบสื่อสารอื่นๆ ที่เกี่ยวข้องได้ต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Thesis Title	A Small Architecture of Viterbi Decoder for 3GPP Mobile System
Student	Mr. Chaiwat Keawsai
Student ID.	44061826
Degree	Master of Engineering
Programme	Information Engineering
Year	2003
Thesis Advisor	Asst.Prof.Dr.Attasit Lasakul
Thesis Co-Advisor	Dr.Keattisak Sripimanwat

ABSTRACT

This thesis deals with the implementation of error control coding on voice service channel for 3GPP mobile system by using Convolution encoding and Viterbi decoding. This Viterbi decoding scheme can be categorized into two cases of parallel and serial structure. Basically, its parallel structure operates with a higher speed than that of the serial style but it, of course, obtains a larger implemented area. Due to the limitation of integrated circuits on their speed, silicon area, and power consumption, this work proposes an implementation of a small Viterbi decoder following TS 25.212 standard of 3GPP. Its specifications are coding rates of 1/2 and 1/3, with generator polynomial $(561, 753)_8$ and $(557, 663, 711)_8$ respectively. Both cases are designed with the constraint length of 9 (256 states). In this design, the serial Viterbi decoder architecture with four add-compare-select (ACS) combined with the register exchange method is proposed and it is modified in order to connect with the serial architecture working style. This work is done by using Verilog HDL and implemented on a platform for Xilinx Vertex II FPGA. As a result, it obtains small number of slices and block RAM, achieves up to 3.46 Mbps of data transfer which suitable for 3G W-CDMA mobile system, and could be modified for other related communication systems accordingly.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงได้อย่างดี ด้วยการช่วยเหลือและสนับสนุนจากหลายๆ ฝ่ายด้วยกัน ซึ่งผู้เขียนขอขอบพระคุณดังต่อไปนี้

ขอขอบคุณ ผศ.ดร.อรรถสิทธิ์ หล้าสกุล อาจารย์ผู้ควบคุมวิทยานิพนธ์ และดร.เกียรติศักดิ์ ศรีพิมานวัฒน์ อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม ในการให้ความรู้ คำแนะนำและคำปรึกษาที่เป็นประโยชน์อย่างสูงในการทำวิทยานิพนธ์นี้ ผู้เขียนรู้สึกซาบซึ้งในความอนุเคราะห์ที่ดีเยี่ยมและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณศูนย์พัฒนาธุรกิจออกแบบวงจรรวม(TIDI) ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ที่ให้ความอนุเคราะห์ซอฟต์แวร์ในการทำวิจัย

ขอขอบคุณโครงการวิจัยชุดต้นแบบหน่วยปรับปรุงประสิทธิภาพการสื่อสารสัญญาณสำหรับโครงการ 3G และการประยุกต์สำหรับอุตสาหกรรม (RDT2) ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ที่ให้ที่ให้ความร่วมมือและสนับสนุนอุปกรณ์ในการทำวิจัย

ขอขอบคุณภาควิชาวิศวกรรมสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ประสิทธิประสาทวิชาความรู้ และให้ใช้สถานที่รวมทั้งเครื่องมือในการทำวิทยานิพนธ์

ขอขอบคุณเพื่อนสมาชิกห้องวิจัย Digital System Laboratory ที่ให้ความร่วมมือและความช่วยเหลือต่างๆ ในการทำวิทยานิพนธ์เป็นอย่างดี

สุดท้ายนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ผู้เป็นผู้ให้ทุกสิ่งทุกอย่างเสมอมา

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ทางผู้จัดทำขอบอบแด่ผู้มีพระคุณทุกท่าน ไว้ ณ โอกาสนี้

ชัยวัฒน์ แก้วสาย

สารบัญ

หน้า

บทคัดย่อภาษาไทย.....	I
บทคัดย่อภาษาอังกฤษ.....	II
กิตติกรรมประกาศ.....	III
สารบัญ.....	IV
สารบัญตาราง.....	VIII
สารบัญรูป.....	IX
รายการคำย่อ.....	XI
รายการสัญลักษณ์.....	XIII
บทที่ 1 บทนำ.....	1
1.1 ความเป็นมาและความสำคัญของปัญหา.....	1
1.2 ความมุ่งหมายและวัตถุประสงค์ของการวิจัย.....	2
1.3 ขอบเขตการวิจัย.....	3
1.4 ขั้นตอนของการวิจัย.....	3
1.5 รายละเอียดวิทยานิพนธ์.....	4
บทที่ 2 หลักการและการใช้งานรหัสแก้ไขความผิดพลาดล่วงหน้าและรหัสคอนวอลูชัน.....	5
2.1 พื้นฐานรหัสแก้ไขความผิดพลาด.....	5
2.1.1 รูปแบบระบบสื่อสารดิจิทัล.....	5
2.1.2 วิธีการแก้ไขความผิดพลาดของข้อมูลในระบบสื่อสารดิจิทัล.....	6
2.1.3 ชนิดของการเกิดความผิดพลาดในระบบสื่อสาร.....	7
2.1.4 รีตันแดนซีและอัตราการใช้รหัส.....	7
2.1.5 อัตราขยายการใช้รหัส (Coding Gain).....	8
2.2 รหัสแบบบล็อก (Block Code).....	8
2.2.1 การเข้ารหัสแบบลิเนียร์บล็อก (Linear Block Code).....	8
2.2.2 การถอดรหัสแบบลิเนียร์บล็อก.....	9
2.2.3 ความสามารถในการตรวจจับและแก้ไขความผิดพลาด.....	9
2.3 รหัสคอนวอลูชัน (Convolution Code).....	10
2.3.1 การเข้ารหัสคอนวอลูชัน.....	10

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

	หน้า
2.3.2 การวิเคราะห์การทำงานของการทำงานการเข้ารหัสคอนโวลูชัน.....	11
2.3.3 แผนภาพแสดงการทำงานของการทำงานการเข้ารหัสคอนโวลูชัน.....	12
2.3.3.1 แผนภาพสเตต (State Diagram).....	12
2.3.3.2 แผนภาพทรี (Tree Diagram).....	13
2.3.3.3 แผนภาพ Trellis (Trellis Diagram).....	13
2.3.4 การถอดรหัสคอนโวลูชัน.....	16
2.4 การถอดรหัสด้วยวิธี Viterbi.....	16
2.4.1 รูปแบบในการถอดรหัสแบบ Viterbi.....	16
2.4.2 การถอดรหัสด้วย Viterbi Algorithm.....	17
2.4.3 การถอดรหัสคอนโวลูชันที่ R เท่ากับ $1/2$ และ K เท่ากับ 3 ด้วยวิธี Viterbi	18
2.5 ระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 และการใช้งานรหัสแก้ไขความผิดพลาด.....	21
บทที่ 3 เอฟพีจีเอพื้นฐานและการออกแบบ.....	23
3.1 ความรู้พื้นฐานวงจรรวมแบบดิจิทัล.....	23
3.1.1 วงจรรวมแบบลอจิกมาตรฐาน (Standard Logic).....	23
3.1.2 วงจรรวมแบบพีแอลดี (Programmable Logic Device: PLD).....	23
3.1.2.1 อุปกรณ์เอสพีแอลดี (SPLD).....	24
3.1.2.2 อุปกรณ์ซีพีแอลดี (CPLD).....	28
3.1.2.3 อุปกรณ์เอฟพีจีเอ (FPGA).....	28
3.1.2.4 เทคโนโลยีการโปรแกรมพีแอลดี.....	29
3.1.3 วงจรรวมแบบเอสิก(Application Specific Integrated Circuit: ASIC)...30	
3.1.3.1 วงจรรวมแบบเกตอาร์เรย์ (Gate Array).....	30
3.1.3.2 วงจรรวมแบบเซลล์มาตรฐาน (Standard Cell).....	31
3.1.3.3 วงจรรวมแบบฟูลคัสตัม (Full Custom).....	31
3.2 โครงสร้างของเอฟพีจีเอ Xilinx Vertex-II.....	31
3.2.1 โครงสร้างของเอฟพีจีเอ Xilinx Vertex-II.....	31

สารบัญ (ต่อ)

หน้า

3.2.2	รายละเอียดของอุปกรณ์ภายในเอฟพีจีเอ Xilinx Vertex-II.....	32
3.2.2.1	บล็อกอินพุตเอาต์พุต (Input/Output Block).....	32
3.2.2.2	บล็อก Configurable Logic Block (CLB).....	32
3.2.2.3	บล็อกหน่วยความจำ 18 กิโลบิต	34
3.2.2.4	บล็อกวงจรคูณขนาด 18 x18 บิต.....	34
3.2.2.5	บล็อก Digital Clock Manager (DCM).....	35
3.2.2.6	บล็อกเชื่อมต่อและหาเส้นทาง (Routing Resources).....	36
3.3	การออกแบบและติดตั้งวงจรดิจิทัลบนเอฟพีจีเอ.....	37
3.4	ภาษาเวริล็อก (Verilog HDL).....	39
3.4.1	ประวัติความเป็นมา.....	39
3.4.2	โครงสร้างของภาษาเวริล็อก.....	40
บทที่ 4	โครงสร้างของชุดถอดรหัสและวิธีการออกแบบ.....	41
4.1	โครงสร้างและการทำงานของชุดถอดรหัส Viterbi.....	41
4.1.1	โครงสร้างของภาค Branch Metric Unit (BMU).....	41
4.1.2	โครงสร้างของภาค Add-Compare-Select Unit (ACSU).....	44
4.1.3	โครงสร้างของภาค Survivor Memory Unit (SMU).....	46
4.1.3.1	วิธีการทำงานแบบ Register Exchange.....	46
4.1.3.2	วิธีการทำงานแบบ Trace Back.....	46
4.1.4	การเปรียบเทียบโครงสร้างของชุดถอดรหัส Viterbi แบบต่างๆ.....	48
4.2	วิธีการออกแบบ.....	49
4.2.1	แนวความคิดและที่มาในการออกแบบ.....	49
4.2.2	วิธีการปรับปรุงภาค SMU แบบ Register Exchange.....	51
4.2.3	ภาค BMU แบบ 3 บิต Soft Decision	51
4.2.4	ภาค ACSU โครงสร้างอนุกรมแบบ 4 ACS	53
4.2.5	ภาค SMU ที่มีการปรับปรุง Register Exchange	54

สารบัญ (ต่อ)

หน้า

บทที่ 5 ผลการออกแบบและการทดสอบการทำงาน.....	56
5.1 ผลการออกแบบและจำลองการทำงาน.....	56
5.1.1 มอดูลชุดเข้ารหัสคอนวอลูชันที่ R เท่ากับ $1/2$ และ $1/3$	56
5.1.2 มอดูล Branch Metric Unit ที่ R เท่ากับ $1/2$ และ $1/3$	59
5.1.3 มอดูล Add-Compare-Select Unit.....	62
5.1.4 มอดูล Survivor Memory Unit	66
5.1.5 มอดูล Controller.....	67
5.1.6 มอดูลชุดถอดรหัส Viterbi	70
5.2 การทดสอบประสิทธิภาพของชุดเข้ารหัสในช่องสัญญาณ AWGN	75
5.3 การวิเคราะห์ผลการออกแบบและทดสอบ.....	78
5.3.1 การวิเคราะห์ผลทดสอบประสิทธิภาพในช่องสัญญาณ AWGN.....	78
5.3.2 การวิเคราะห์ผลการใช้อุปกรณ์ในเอฟพีจีเอ.....	78
5.3.3 การวิเคราะห์ประสิทธิภาพอัตราเร็วในการถอดรหัส.....	80
บทที่ 6 บทสรุปและข้อเสนอแนะ.....	82
6.1 สรุปผลการวิจัย.....	82
6.2 ปัญหาและอุปสรรคที่พบในงานวิจัย.....	83
6.3 ข้อเสนอแนะในการพัฒนา.....	83
เอกสารอ้างอิง.....	85
ภาคผนวก ก. โปรแกรมชุดถอดรหัส.....	89
ภาคผนวก ข. โปรแกรมทดสอบ.....	121
ภาคผนวก ค. สัญญาณ สัญญาณรบกวนและความสัมพันธ์ในการจำลองด้วย โปรแกรมคอมพิวเตอร์.....	133
ภาคผนวก ง. ตารางข้อมูลอุปกรณ์ชุดรหัสคอนวอลูชันและชุดถอดรหัส Viterbi จากผู้ผลิตรายอื่นๆ.....	135
ผลงานวิจัยที่ได้รับการตีพิมพ์.....	136
ประวัติผู้เขียน.....	142

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้拿去ใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

ตารางที่	หน้า
3.1 ขนาดหน่วยความจำที่สามารถเลือกใช้ได้กับบล็อกหน่วยความจำ	34
4.1 ผลเปรียบเทียบข้อดีข้อเสียของโครงสร้างของชุดถอดรหัส Viterbi แบบต่างๆ.....	48
4.2 รายละเอียดของชุดถอดรหัส Viterbi ในวิทยานิพนธ์นี้	49
4.3 การแทนค่าข้อมูลอินพุตแบบ Soft Decision 3 บิต.....	53
5.1 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูลชุดเข้ารหัส.....	57
5.2 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูลชุดเข้ารหัส.....	57
5.3 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล BMU	60
5.4 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล BMU.....	60
5.5 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล ACSU.....	64
5.6 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล ACSU.....	64
5.7 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล SMU.....	67
5.8 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล SMU.....	67
5.9 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล Controller.....	69
5.10 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล Controller.....	69
5.11 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูลชุดถอดรหัส.....	72
5.12 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูลชุดถอดรหัส.....	73
5.13 การเปรียบเทียบกับชุดถอดรหัสฮาร์ดแวร์ของบริษัท Xilinx ใน FPGA xc2v1000-6.....	80
5.14 การเปรียบเทียบกับชุดถอดรหัส Viterbi ที่มีค่า K เท่ากับ 9 ของงานวิจัยอื่นๆ.....	81

สารบัญรูป

รูปที่	หน้า
2.1 บล็อกไดอะแกรมองค์ประกอบพื้นฐานของระบบสื่อสารดิจิทัล.....	6
2.2 วงจรเข้ารหัสคอนโวลูชันที่ค่า R เท่ากับ $1/2$ และค่า K เท่ากับ 3.....	11
2.3 แผนภาพสเตทที่ R เท่ากับ $1/2$ และ K เท่ากับ 3.....	13
2.4 แผนภาพทรีที่ R เท่ากับ $1/2$ และ K เท่ากับ 3.....	14
2.5 แผนภาพ Trellis ที่ R เท่ากับ $1/2$ และ K เท่ากับ 3.....	14
2.6 ตัวอย่างการใช้งานแผนภาพ Trellis.....	15
2.7 ค่า Branch Metric ในแผนภาพ Trellis	19
2.8 ค่า Path Metric ในเวลา t_1 ถึง t_3	19
2.9 เส้นทางของข้อมูลที่ได้จากการถอดรหัสด้วยวิธี Viterbi	20
2.10 วงจรคอนโวลูชันที่ใช้ในมาตรฐาน TS25.212.....	22
3.1 ไดอะแกรมเทคโนโลยีการออกแบบวงจรรวมแบบดิจิทัล.....	24
3.2 โครงสร้างของพรม.....	25
3.3 โครงสร้างของพีแอลเอ.....	26
3.4 โครงสร้างของพีเอแอล.....	27
3.5 โครงสร้างของซีพีแอลดี.....	27
3.6 โครงสร้างทั่วไปของเอฟพีจีเอ.....	28
3.7 โครงสร้างของเอฟพีจีเอตระกูล Vertex-II.....	31
3.8 โครงสร้างของบล็อกอินพุตเอาต์พุต	33
3.9 โครงสร้างของ Slice	33
3.10 บล็อกหน่วยความจำ 18 กิโลบิตและบล็อกวงจรขนาด 18×18 บิต.....	35
3.11 โครงสร้างของบล็อกเชื่อมต่อและหาเส้นทาง.....	36
3.12 ขั้นตอนการออกแบบและติดตั้งวงจรดิจิทัลบนเอฟพีจีเอ.....	37
4.1 บล็อกไดอะแกรมของชุดถอดรหัส Viterbi.....	42
4.2 ไดอะแกรมภาค BMU	43
4.3 ไดอะแกรม Add-Compare-Select (ACS).....	44
4.4 โครงสร้าง ACS แบบ Butterfly.....	45
4.5 วงจร SMU แบบ Register Exchange ที่ K เท่ากับ 3.....	47
4.6 ไดอะแกรม SMU แบบวิธี Trace Back ที่ค่า K เท่ากับ 3.....	48

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป (ต่อ)

รูปที่	หน้า
4.7 ภาค SMU แบบ Register Exchange ที่มีการปรับปรุง.....	50
4.8 ชุดถอดรหัส Viterbi แบบ 4 ACS ที่มีการปรับปรุง Register Exchange	52
4.9 การทำงานของวงจร BMU แบบ 3 บิต Soft Decision.....	52
4.10 บล็อกไดอะแกรมการทำงานของภาค ACSU ที่นำเสนอ.....	54
4.11 ไดอะแกรม ACS ที่ใช้เทคนิค Modulo Normalization.....	54
4.12 บล็อกไดอะแกรม SMU ที่มีการปรับปรุง Register Exchange	55
5.1 ผังวงจรระดับ RTL ของมอดูลชุดเข้ารหัสที่ได้จากการสังเคราะห์.....	58
5.2 ผลการจำลองการทำงานของมอดูลชุดเข้ารหัสที่ R เท่ากับ $1/2$ และ $1/3$	59
5.3 ผังวงจรระดับ RTL ของมอดูล BMU ที่ได้จากการสังเคราะห์.....	61
5.4 ผลการจำลองการทำงานของมอดูล BMU จากอินพุต “11111111” ที่ R เท่ากับ $1/2$ และ $1/3$	63
5.5 ผังวงจรระดับ RTL ของมอดูล ACSU ที่ได้จากการสังเคราะห์.....	65
5.6 ผลการจำลองการทำงานของมอดูล ACS.....	66
5.7 ผังวงจรระดับ RTL ของมอดูล SMU ที่ได้จากการสังเคราะห์.....	68
5.8 ผลการจำลองการทำงานของมอดูล SMU.....	69
5.9 ผลการจำลองการทำงานของมอดูล Controller.....	70
5.10 ชื่อสัญญาณของมอดูลชุดถอดรหัส.....	71
5.11 ผังวงจรของมอดูลชุดถอดรหัสได้จากการสังเคราะห์.....	73
5.12 ผังการเชื่อมต่ออุปกรณ์ภายในเอฟพีจีเอ Xilinx Vertex II xc2v1000-6 ที่ถูกใช้.....	74
5.13 ผลการจำลองการทำงานระดับฐานเวลาของมอดูลชุดถอดรหัสที่ R เท่ากับ $1/3$	74
5.14 บล็อกไดอะแกรมการทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณ AWGN.....	75
5.15 ผลของอัตราความผิดพลาดบิต (BER) กับค่ากำลังของสัญญาณต่อสัญญาณรบกวน (E_b/N_0) ที่ R เท่ากับ $1/2$ และ $1/3$ ในช่องสัญญาณ AWGN	76
5.16 ตัวอย่างผลการจำลองการทำงานชุดถอดรหัสในช่องสัญญาณ AWGN ที่ R เท่ากับ $1/2$	77
5.17 ตัวอย่างผลการจำลองการทำงานชุดถอดรหัสในช่องสัญญาณ AWGN ที่ R เท่ากับ $1/3$	77
5.18 ประสิทธิภาพของชุดถอดรหัสที่นำเสนอกับงานวิจัย [29] ที่ R เท่ากับ $1/2$	79
5.19 ประสิทธิภาพของชุดถอดรหัสที่นำเสนอกับงานวิจัย [2] ที่ R เท่ากับ $1/3$	79

รายการคำย่อ

3 rd Generation Partnership Project	3GPP
Application Specific Integrated Circuit	ASIC
Add Compare Select	ACS
Add Compare Select Unit	ACSU
Additive White Gaussian Noise	AWGN
Binary Phase Shift Keying	BPSK
Bit Error Rate	BER
Branch Metric Unit	BMU
Code Division Multiple Access	CDMA
Complex Programmable Logic Device	CPLD
Configurable Logic Block	CLB
Cyclic Redundancy Check	CRC
Digital Signal Processing	DSP
Forward Error Correction	FEC
Frequency Maximum	Fmax
Field Programmable Gate Array	FPGA
Hardware Description Language	HDL
Intellectual Property	IP
Look Up Table	LUT
Mega bit per second	Mbps
Motion Picture Experts Group	MPEG
Non-Recurring Engineering	NRE
Open Verilog International	OVI
Programmable Array Logic	PAL
Programmable Logic Array	PLA
Programmable Logic Device	PLD
Path Metric	PM
Path Metric Memory	PMM

รายการคำย่อ (ต่อ)

Programmable Read Only Memory	PROM
Erasable Programmable Read Only Memory	EPROM
Random Access Memory	RAM
Register Exchange	RE
Read Only Memory	ROM
Register Transfer Level	RTL
Serial to Parallel	S/P
Survivor Memory Unit	SMU
Signal-to-Noise Ration	SNR
Static Random Access Memory	SRAM
Third Generation Mobile System	3G
Trace Back	TB
Very High Speed Integrated Circuit	VHSIC
Very Large Scale Integration Circuit	VLSI
VHISC Hardware Description Language	VHDL
Wideband CDMA	W-CDMA
Wireless Local Access Network	WLAN

รายการสัญลักษณ์

Branch Metric	BM
Coding Rate	R
Constraint Length	K
Decision Bit	DB
Generator Polynomial	G และ g
Minimum Hamming Distance	d_{min}
Path Metric	PM
Redundant	r
State Metric	SM
Syndrome	S



บทที่ 1

บทนำ

ในระบบสื่อสารต่างๆ นั้น การรบกวนจากสัญญาณรบกวนเป็นปัญหาสำคัญปัญหาหนึ่งที่ทำให้การรับส่งข้อมูลเกิดความผิดพลาด ซึ่งระบบสื่อสารเหล่านี้จำเป็นต้องได้รับการป้องกันและแก้ไขปัญหาดังกล่าว หนึ่งในวิธีการป้องกันและลดความผิดพลาดที่เกิดขึ้นนี้คือ การเข้ารหัสแบบคอนโวลูชัน (Convolution) และการถอดรหัสแบบ Viterbi ซึ่งเป็นวิธีการป้องกันความผิดพลาดล่วงหน้า (Forward Error Correction) หรือเรียกว่า “FEC” ซึ่งการเข้ารหัสคอนโวลูชันและการถอดรหัส Viterbi นี้ได้ถูกนำมาใช้งานในระบบสื่อสารดิจิทัลในปัจจุบันอย่างแพร่หลาย อาทิ ระบบสื่อสารดาวเทียม ระบบโทรศัพท์เคลื่อนที่ และระบบ Wireless LAN เป็นต้น โดยเฉพาะในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 (Third Generation Mobile System) ได้้นำการเข้ารหัสคอนโวลูชันและการถอดรหัส Viterbi มาใช้งานสำหรับการเพิ่มประสิทธิภาพการรับส่งข้อมูลในการรับส่งข้อมูลภาพและเสียง (Voice Services) ซึ่งในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ (3rd Generation Partnership Project) ได้กำหนดให้ใช้รหัสเพื่อป้องกันความผิดพลาดล่วงหน้าไว้สองแบบ คือ รหัสคอนโวลูชันในการบริการสัญญาณภาพและเสียง (Voice Services) และรหัสเทอร์โบ (Turbo Code) ในการบริการสัญญาณข้อมูล (Data Services) [1][2]

1.1 ความเป็นมาและความสำคัญของปัญหา

การติดต่อสื่อสารนั้นเป็นส่วนหนึ่งในชีวิตประจำวัน โดยเฉพาะการติดต่อสื่อสารโดยใช้โทรศัพท์เคลื่อนที่นั้นมีการใช้งานเป็นจำนวนมากและมีแนวโน้มที่เพิ่มขึ้นในอนาคต การพัฒนาระบบโทรศัพท์เคลื่อนที่ในปัจจุบันได้พัฒนามาถึงยุคที่ 3 หรือที่เรียกว่า “3G” ซึ่งเป็นระบบโทรศัพท์ที่สามารถสื่อสารข้อมูลด้วยความเร็วสูง สามารถรองรับการรับส่งข้อมูลที่มีปริมาณข้อมูลข่าวสารจำนวนมาก เช่น ข้อมูลแบบมัลติมีเดีย เป็นต้น ซึ่งในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP ได้กำหนดอัตราเร็วการรับส่งข้อมูลที่ 2 Mbps และกำหนดให้ใช้อุปกรณ์ป้องกันความผิดพลาดล่วงหน้าเพื่อเพิ่มประสิทธิภาพการรับส่งข้อมูลซึ่งการสร้างชิ้นส่วนอุปกรณ์นี้สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นั้น นอกจากการทำงานที่ถูกต้องตามมาตรฐานแล้ว ความเร็วในการทำงาน พื้นที่ในการสร้าง (Silicon Area) และพลังงาน (Power Consumption) เป็นสิ่งสำคัญที่แสดงถึงประสิทธิภาพของอุปกรณ์ชนิดนี้

ในการสร้างชุดถอดรหัส Viterbi ซึ่งเป็นอุปกรณ์ป้องกันความผิดพลาดล่วงหน้า โดยทั่วไป โครงสร้างทาง VLSI (Very Large Scale Integration Circuit) ของชุดถอดรหัส Viterbi แบ่งได้เป็น สองประเภทคือ โครงสร้างแบบขนาน (Parallel Architecture) และโครงสร้างแบบอนุกรม (Serial Architecture) โครงสร้างแบบขนานนั้นจะให้อัตราเร็วในการถอดรหัสสูง แต่พื้นที่ในการสร้างนั้นมีขนาดที่ใหญ่กว่าแบบอนุกรม [3] โดยเฉพาะชุดถอดรหัส Viterbi ที่ใช้ถอดรหัสคอนโวลูชันที่มีจำนวนสเตต (State) ของการทำงานที่สูง เช่น 256 สเตต ในมาตรฐานโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP การสร้างในแบบขนานต้องใช้อุปกรณ์ในการสร้างจำนวนมากและมีการเชื่อมต่อภายใน (Interconnect) จำนวนมากเช่นกัน ซึ่งการสร้างต้องใช้พื้นที่มากและทำให้มีต้นทุนที่สูง จากปัญหาดังกล่าวในวิทยานิพนธ์นี้ได้นำเสนอถึงวิธีการสร้างชุดถอดรหัส Viterbi สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP ที่มีโครงสร้างขนาดเล็กเพื่อแก้ปัญหาดังกล่าว โดยจะใช้อุปกรณ์ในการสร้างจำนวนน้อย และสามารถทำให้ต้นทุนในการสร้างอุปกรณ์นี้ลดลง

1.2 ความมุ่งหมายและวัตถุประสงค์ของการวิจัย

วิทยานิพนธ์นี้นำเสนอการออกแบบและวิจัยพัฒนาสร้างชุดถอดรหัส Viterbi ซึ่งเป็นชิ้นส่วนอุปกรณ์ที่สำคัญส่วนหนึ่งของระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 การออกแบบพัฒนาสร้างต้นแบบของชุดถอดรหัสนี้ใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language: HDL) ในการบรรยายพฤติกรรมการทำงานของชุดถอดรหัสเพื่อนำไปติดตั้ง (Implementation) ในอุปกรณ์เอฟพีจีเอ (Field Programmable Gate Array) ซึ่งวัตถุประสงค์ของการวิจัยมีดังนี้

- เพื่อศึกษาการทำงานของอุปกรณ์ชุดเข้ารหัสคอนโวลูชันและชุดถอดรหัส Viterbi และสามารถออกแบบสร้างอุปกรณ์ชุดเข้ารหัสและถอดรหัสนี้ให้ใช้งานกับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP ได้
- เพื่อพัฒนาสร้างอุปกรณ์ชุดถอดรหัส Viterbi ให้มีโครงสร้างขนาดเล็ก ซึ่งจะช่วยให้ใช้พื้นที่ในการสร้างน้อยและทำให้ราคาของอุปกรณ์ถูกลง
- เพื่อพัฒนาสร้างอุปกรณ์ชุดเข้ารหัสคอนโวลูชันและชุดถอดรหัส Viterbi นี้ให้เป็นต้นแบบเชิงอุตสาหกรรม
- สามารถนำต้นแบบอุปกรณ์ชุดเข้ารหัสคอนโวลูชันและชุดถอดรหัส Viterbi นี้ไปประยุกต์ใช้งานในระบบสื่อสารดิจิทัลอื่นๆ ได้

1.3 ขอบเขตการวิจัย

วิทยานิพนธ์นี้นำเสนอการออกแบบและพัฒนาสร้างต้นแบบชุดถอดรหัส Viterbi ที่มีโครงสร้างขนาดเล็กสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ซึ่งชุดถอดรหัส Viterbi สามารถถอดรหัสคอนโวลูชันที่มีค่า Constraint Length (K) เท่ากับ 9 (256 States) อัตราการเข้ารหัส $1/2$ และ $1/3$ และ Generator Polynomial $(561, 753)_8$ และ $(557, 663, 711)_8$ ตามลำดับ ซึ่งกำหนดตามมาตรฐาน TS 25.212 ของ 3GPP การออกแบบชุดเข้ารหัสและถอดรหัสนี้ได้ใช้ภาษาเวริล็อก (Verilog HDL) ในการบรรยายพฤติกรรมการทำงาน และใช้โปรแกรม Xilinx ISE และ ModelSim ในการจำลองการทำงาน การตรวจสอบและแก้ไข การสังเคราะห์วงจร การวางอุปกรณ์และเชื่อมต่อสัญญาณ (Place & Route) ในอุปกรณ์เอฟพีจีเอ ในการตรวจสอบการทำงานและทดสอบชุดเข้ารหัสคอนโวลูชันและถอดรหัส Viterbi นี้ใช้การจำลองการทำงานระดับฐานเวลา (Timing Simulation) ตรวจสอบความถูกต้องเป็นขั้นตอนสุดท้าย โดยความเร็วการทำงานต้องมากกว่า 2 Mbps ซึ่งเป็นความเร็วในการรับส่งข้อมูลสูงสุดในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP ซึ่งผลการออกแบบที่ได้นี้อยู่ในรูปของซอฟต์แวร์ (Intellectual Property Core: IP Core) ของชุดเข้ารหัสคอนโวลูชันและถอดรหัส Viterbi ที่สามารถนำไปโปรแกรม (Configuration) ลงเอฟพีจีเอเพื่อใช้งานในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 (3GPP) หรือปรับปรุงให้ใช้กับระบบสื่อสารอื่นๆ ที่เกี่ยวข้องได้

1.4 ขั้นตอนของการวิจัย

การวิจัยพัฒนาชุดถอดรหัส Viterbi สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นี้ กำหนดขั้นตอนต่างๆ ของการวิจัยไว้ตามลำดับดังนี้

- ศึกษาการทำงานและการสำรวจข้อมูลอุปกรณ์ชุดรหัสคอนโวลูชัน และชุดถอดรหัส Viterbi จากผู้ผลิตรายอื่นๆ ทั้งในด้านรายละเอียดและลักษณะการใช้งานของอุปกรณ์
- ออกแบบพัฒนาให้อุปกรณ์ชุดเข้ารหัสและถอดรหัสมีโครงสร้างขนาดเล็ก ทำให้ใช้พื้นที่ (Silicon Area) ในการสร้างอุปกรณ์น้อย
- นำผลที่ได้จากการพัฒนามาออกแบบด้วยภาษาฮาร์ดแวร์ แล้วตรวจสอบความถูกต้องด้วยการจำลองการทำงาน (Simulation)
- สังเคราะห์ (Synthesis) เป็นวงจรระดับเกต และตรวจสอบความถูกต้องด้วยการจำลองการทำงานในระดับฟังก์ชัน (Functional Simulation)
- วางอุปกรณ์และการเชื่อมต่อสัญญาณ (Place & Route) และตรวจสอบความถูกต้องด้วยการจำลองการทำงานระดับฐานเวลา (Timing Simulation) ซึ่งจะได้ซอฟต์แวร์ (IP Core) ที่สามารถโปรแกรมลงอุปกรณ์เอฟพีจีเอเพื่อใช้งานจริงต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.5 รายละเอียดวิทยานิพนธ์

วิทยานิพนธ์นี้ได้แบ่งเนื้อหาออกเป็นทั้งหมด 6 บท โดยบทที่ 1 กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์การทำวิจัย ขอบเขตการวิจัย และขั้นตอนของการวิจัย ส่วนบทอื่นๆ นั้นมีเนื้อหาดังนี้

บทที่ 2 กล่าวถึงหลักการพื้นฐานต่างๆ ของรหัสแก้ไขความผิดพลาด รหัสแบบบล็อก (Block Code) รหัสคอนโวลูชัน (Convolution Code) การถอดรหัสด้วยวิธี Viterbi และการใช้งานรหัสแก้ไขความผิดพลาดในระบบโทรศัพท์เคลื่อนที่ในยุคที่ 3

บทที่ 3 กล่าวถึงพื้นฐานเทคโนโลยีเชิงจรรวมแบบดิจิทัล โครงสร้างพื้นฐานของอุปกรณ์เอพฟิจีเอ Xilinx Vertex-II ตลอดจนเทคนิควิธีการการออกแบบและติดตั้งวงจรดิจิทัลบนเอพฟิจีเอ ประวัติความเป็นมาและโครงสร้างของภาษาเวริลล็อก

บทที่ 4 กล่าวถึงโครงสร้างโดยทั่วไปและหลักการการทำงานของชุดถอดรหัส Viterbi แบบต่างๆ และการออกแบบของชุดถอดรหัส Viterbi แบบอนุกรม 4 ACS ที่มีการปรับปรุง Survivor Memory Unit (SMU) แบบ Register Exchange สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3

บทที่ 5 กล่าวถึงผลการออกแบบและทดสอบ การใช้อุปกรณ์ภายในเอพฟิจีเอ ความถี่สูงสุดที่สามารถทำงานได้ และผลการจำลองการทำงาน ผลการทดสอบประสิทธิภาพของในช่องสัญญาณแบบ AWGN (Additive White Gaussian Noise) และการวิเคราะห์ประสิทธิภาพและลักษณะเด่นของชุดถอดรหัสเมื่อเปรียบเทียบกับชุดถอดรหัสของบริษัท Xilinx และชุดถอดรหัสในงานวิจัยอื่นๆ

ส่วนบทสุดท้ายเป็นการสรุปผลการวิจัย ปัญหาและอุปสรรคในการวิจัย และข้อเสนอแนะในการพัฒนา

บทที่ 2

หลักการและการใช้งานรหัสแก้ไขความผิดพลาด และรหัสคอนวอลูชัน

รหัสแก้ไขความผิดพลาด (Error Control Coding) เป็นเทคนิคที่ใช้ในการแก้ไขความผิดพลาดจากการรับส่งข้อมูลในระบบสื่อสารดิจิทัล ซึ่งความผิดพลาดที่เกิดขึ้นนั้นเกิดจากสาเหตุหลายประการ และสาเหตุเหล่านี้ไม่สามารถคาดคะเนเวลาการเกิดได้ โดยเฉพาะสาเหตุที่สำคัญคือการรบกวนจากสัญญาณรบกวน (Noise) ดังนั้นจึงจำเป็นต้องใช้รหัสดังกล่าวเพื่อป้องกันหรือแก้ไขความผิดพลาดจากสาเหตุเหล่านี้ โดยในบทนี้จะกล่าวถึงพื้นฐานของรหัสแก้ไขความผิดพลาด รหัสแบบบล็อก (Block Code) รหัสคอนวอลูชัน (Convolution Code) การถอดรหัสคอนวอลูชันด้วยวิธี Viterbi และระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 และการใช้งานรหัสแก้ไขความผิดพลาด

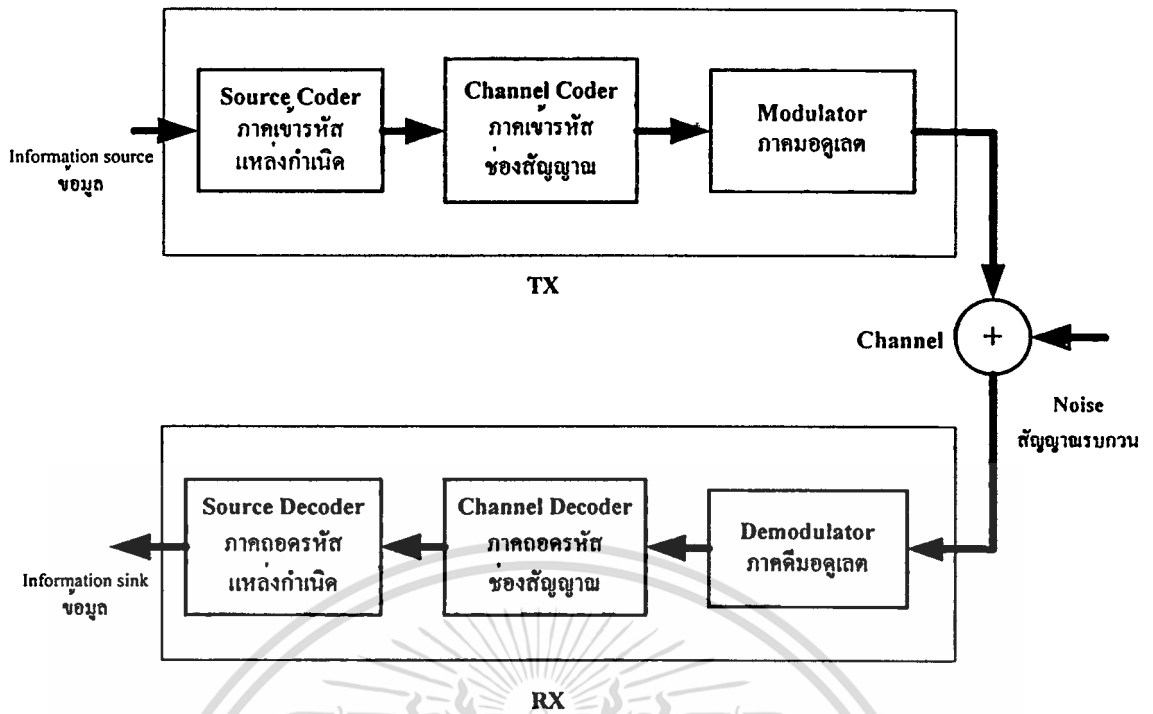
2.1 พื้นฐานรหัสแก้ไขความผิดพลาด

รหัสแก้ไขความผิดพลาดเป็นส่วนที่สำคัญในระบบสื่อสารดิจิทัล ซึ่งการใช้งานรหัสแก้ไขความผิดพลาดทำให้ค่าอัตราความผิดพลาด (Error Rate) ในการส่งข้อมูลลดลงและทำให้ประสิทธิภาพของระบบสื่อสารเพิ่มขึ้น ซึ่งส่วนนี้จะกล่าวถึงหลักการพื้นฐานต่างๆ ของรหัสแก้ไขความผิดพลาดนี้

2.1.1 รูปแบบระบบสื่อสารดิจิทัล

พื้นฐานระบบสื่อสารดิจิทัลทั่วไปนั้นประกอบด้วยภาคส่ง (Transmitter: Tx) ภาครับ (Receiver: Rx) และช่องสัญญาณการสื่อสาร (Channel) ในการทำงานของภาคส่งนั้น ข้อมูลข่าวสาร (Information Source) ต้องถูกเข้ารหัสข้อมูลโดยภาคเข้ารหัสแหล่งกำเนิด (Source Coder) เพื่อทำให้ขนาดของข้อมูลข่าวสารที่ถูกส่งลดลง ตัวอย่างการเข้ารหัสนี้ ได้แก่ การบีบอัดข้อมูลภาพแบบ JPEG หรือ MPEG เป็นต้น จากนั้นข้อมูลนี้จะถูกเข้ารหัสเพื่อป้องกันความผิดพลาดอีกครั้งที่ภาคเข้ารหัสช่องสัญญาณ (Channel Coder) เพื่อป้องกันหรือลดความผิดพลาดต่างๆ ที่เกิดสัญญาณรบกวนหรืออื่นๆ ในการส่งข้อมูลผ่านช่องสัญญาณการสื่อสาร ซึ่งข้อมูลที่ถูกเข้ารหัสแก้ไขความผิดพลาดนี้ประกอบด้วย ข้อมูลข่าวสารและส่วนของข้อมูลที่เพิ่มขึ้นมา ซึ่งเรียกว่า รีดันแดนซี (Redundancy) สุดท้ายข้อมูลนี้จะถูกมอดูเลต (Modulate) เช่น การมอดูเลตเชิงเฟส (Phase Shift Keying: PSK) หรือการมอดูเลตเชิงความถี่ (Frequency Shift Keying: FSK) ที่ภาคมอดูเลต (Modulator) เพื่อให้เหมาะสมกับช่องสัญญาณต่อไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.1 บล็อกไดอะแกรมองค์ประกอบพื้นฐานของระบบสื่อสารดิจิทัล

ส่วนในภาครับนั้น การทำงานของส่วนต่างๆ จะสอดคล้องกับภาคส่ง สัญญาณข้อมูลที่ได้รับได้ถูกดีมอดูเลตกลับคืนมาที่ภาคดีมอดูเลต (Demodulator) ซึ่งข้อมูลที่ได้รับได้นี้อาจมีความผิดพลาดรวมเข้ามาด้วย ข้อมูลนี้จะถูกถอดรหัสและแก้ไขความผิดพลาดที่เกิดขึ้นให้ถูกต้องที่ภาคถอดรหัสช่องสัญญาณ (Channel Decoder) จากนั้นข้อมูลที่ได้จากภาคถอดรหัสช่องสัญญาณจะถูกแปลงให้เป็นข้อมูลข่าวสารเดิม โดยภาคถอดรหัสแหล่งกำเนิด (Source Decoder) โดยบล็อกไดอะแกรมองค์ประกอบพื้นฐานของระบบสื่อสารดิจิทัลทั่วไปนั้นแสดงดังรูปที่ 2.1

2.1.2 วิธีการแก้ไขความผิดพลาดของข้อมูลในระบบสื่อสารดิจิทัล

วิธีที่ใช้ในการป้องกันหรือแก้ไขความผิดพลาดของข้อมูลในระบบสื่อสารดิจิทัลนั้น แบ่งได้เป็นสองลักษณะใหญ่ๆ ด้วยกันคือ วิธีการส่งซ้ำอัตโนมัติ (Automatic Retransmission Query) หรือเรียกว่า “ARQ” และวิธีการแก้ไขความผิดพลาดล่วงหน้า (Forward Error Correction) หรือเรียกว่า “FEC” [4] วิธีการทำงานของ ARQ นั้น เมื่อภาครับตรวจสอบได้ว่าข้อมูลที่ได้รับได้เกิดความผิดพลาดก็ทำการร้องขอให้มีการส่งข้อมูลใหม่อีกครั้ง การตรวจสอบความผิดพลาดที่เกิดขึ้นนั้นใช้ส่วนของรีดันแดนซีในการทำงาน วิธี ARQ นั้นใช้ในการสื่อสารแบบสองทาง (Duplex) เท่านั้น ส่วนวิธีการทำงานของ FEC นั้น เมื่อภาครับตรวจสอบได้ว่าข้อมูลที่ได้รับได้เกิดความผิดพลาดภาครับก็สามารถทำการแก้ไขข้อมูลที่ผิดพลาดให้ถูกต้องได้ โดยใช้ส่วนของข้อมูลข่าวสารและ

รีคันแดนซีในแก้ไขข้อมูลที่ผิดพลาดดังกล่าว วิธี FEC สามารถใช้ในการสื่อสารแบบทางเดียว (Simplex) หรือสองทางก็ได้

การทำงานของวิธี ARQ นั้นง่ายและใช้รีคันแดนซีน้อยกว่า FEC แต่มีความเร็วที่ช้ารับส่งข้อมูลนั้นไม่แน่นอน เนื่องจากต้องมีการส่งข้อมูลใหม่อีกครั้งเมื่อตรวจสอบได้ว่ามีความผิดพลาดเกิดขึ้น ส่วนวิธี FEC นั้นมีความซับซ้อนกว่า ARQ แต่วิธี FEC นี้สามารถรับส่งข้อมูลได้อย่างรวดเร็ว สำหรับระบบสื่อสารแบบดิจิทัลที่ต้องการประสิทธิภาพสูงนั้นการป้องกันการผิดพลาดจะใช้สองวิธีมารวมกันซึ่งเรียกวธีนี้ว่า การแก้ไขความผิดพลาดแบบไฮบริด (Hybrid Error Control) วิธีการป้องกันและแก้ไขใช้วิธี FEC ในขั้นตอนแรกและส่วนเกินความสามารถของวิธี FEC ก็ใช้วิธี ARQ [5]

2.1.3 ชนิดของการเกิดความผิดพลาดในระบบสื่อสาร

การเกิดความผิดพลาดกับการรับส่งข้อมูลในระบบสื่อสารแบบดิจิทัลนั้นสามารถแบ่งได้เป็นสองประเภทคือ การเกิดความผิดพลาดแบบสุ่ม (Random) และการเกิดความผิดพลาดแบบ Burst [6] การเกิดความผิดพลาดแบบสุ่มนั้นเป็นการเกิดความผิดพลาดที่สามารถเกิดขึ้นได้อย่างอิสระกับข้อมูลที่รับส่งในแต่ละบิต ตัวอย่างการเกิดความผิดพลาดเช่น ความผิดพลาดที่เกิดจากการสอดแทรก (Interference) จากสัญญาณอื่นๆ เป็นต้น ส่วนการเกิดความผิดพลาดแบบ Burst นั้นเป็นการเกิดความผิดพลาดที่เกิดกับข้อมูลที่รับส่งหลายๆ บิตติดกัน โดยจำนวนบิตที่ผิดพลาดจากบิตแรกจนถึงบิตสุดท้ายเรียกว่า Burst Length การเกิดความผิดพลาดแบบสุ่มสามารถใช้รหัสแก้ไขความผิดพลาดในแก้ไขได้ ส่วนความผิดพลาดแบบ Burst นั้นการแก้ไขยากกว่าแบบสุ่ม การแก้ไขต้องใช้รหัสแก้ไขความผิดพลาดร่วมกับส่วน Interleaver

2.1.4 รีคันแดนซีและอัตราการเข้ารหัส

วิธีการเข้ารหัสข้อมูลเพื่อแก้ไขความผิดพลาดนั้น เป็นการนำเอาข้อมูลข่าวสารมาแบ่งออกเป็นส่วนๆ ซึ่งขนาดเท่ากับ k บิต โดยข้อมูลที่ได้จากการเข้ารหัสแล้วจำนวน n บิต เรียกว่า คำรหัส หรือ Codeword ซึ่งประกอบด้วยข้อมูลข่าวสารเดิมและส่วนของข้อมูลที่เพิ่มขึ้นซึ่งเรียกว่า รีคันแดนซี (Redundancy) หรือ r ซึ่งรีคันแดนซีนีใช้ในการตรวจสอบหรือแก้ไขข้อมูลที่ผิดพลาด [4] สามารถคำนวณได้จาก

$$r = n - k \quad (2.1)$$

ส่วนจำนวนข้อมูลข่าวสารและจำนวนข้อมูลที่ได้จากการเข้ารหัสนั้นเรียกว่า อัตราการเข้ารหัส หรือ Coding Rate (R) ซึ่งนิยามเขียนเป็น (n, k) หรือ k/n

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.5 อัตราขยายการเข้ารหัส (Coding Gain)

การเข้ารหัสข้อมูลเพื่อแก้ไขความผิดพลาดนั้น ทำให้ค่าอัตราการเกิดความผิดพลาดในการรับส่งข้อมูลลดลง ซึ่งมีค่าอัตราขยายการเข้ารหัส (Coding Gain) เป็นค่าที่แสดงถึงความสามารถในการป้องกันความผิดพลาดของข้อมูล โดยค่าอัตราขยายการเข้ารหัสเป็นค่าความแตกต่างระหว่างกำลังงานที่ใช้ในการส่งข้อมูลต่อสัญญาณรบกวนหรือ E_b/N_0 ซึ่งมีหน่วยเป็น dB ในกรณีที่มีการเข้ารหัสข้อมูลกับกรณีที่ไม่มีการเข้ารหัสข้อมูล (Uncoded) ที่ค่าของอัตราการเกิดความผิดพลาด (Error Rate) เท่ากัน [4] ซึ่งถ้าค่าอัตราขยายการเข้ารหัสของการเข้ารหัสใดมีค่าสูงก็แสดงว่ามีการผิดพลาดในการส่งข้อมูลน้อยเมื่อเทียบกับกรณีที่ไม่มีการเข้ารหัสข้อมูล

2.2 รหัสแบบบล็อก (Block Code)

รหัสแบบบล็อก (Block Code) เป็นหนึ่งในรหัสที่ใช้ป้องกันความผิดพลาดล่วงหน้า (FEC) ซึ่งในการเข้ารหัสแบบบล็อกนั้น ข้อมูลข่าวสารจะถูกแบ่งออกเป็นชุดหรือบล็อกย่อยๆ ก่อนการเข้ารหัส ซึ่งรหัสแบบบล็อกสามารถตรวจสอบและบอกตำแหน่งข้อมูลที่ผิดพลาดได้ ซึ่งขึ้นอยู่กับเงื่อนไขของการเข้ารหัส รหัสแบบบล็อกนี้มีหลายรูปแบบ เช่น รหัสแบบลิเนียร์บล็อก (Linear Block Code) รหัสแบบแฮมมิง รหัสแบบ Cyclic และรหัสแบบ BCH เป็นต้น

2.2.1 การเข้ารหัสแบบลิเนียร์บล็อก (Linear Block Code)

การเข้ารหัสแบบลิเนียร์บล็อกนั้นเป็นพื้นฐานที่สำคัญของรหัสแบบบล็อก รหัสแบบลิเนียร์บล็อกเป็นการตรวจเช็คพริตติชนิดหนึ่ง โดยการเข้ารหัสแบบลิเนียร์บล็อกนั้นเป็นการนำข้อมูลข่าวสารที่ต้องการเข้ารหัสมาแบ่งเป็นชุดๆ หรือบล็อกย่อยๆ ซึ่งแต่ละชุดนั้นมีขนาดเท่ากับ k บิต แล้วนำมาหาค่าการรหัสหรือ Codeword ขนาด n บิต ซึ่งโดยทั่วไปแทนด้วยสัญลักษณ์ (n, k) การเข้ารหัสแบบลิเนียร์บล็อกนั้นการคำนวณหาค่าการรหัสเป็นแบบเชิงเส้น (Linear) ซึ่งการหาค่าการรหัสจะใช้ Generator Matrix ในการคำนวณ ดังนี้

$$U = mG \quad (2.2)$$

โดย U คือ ข้อมูลที่ได้หลังจากการเข้ารหัส m คือ ชุดของข้อมูลจำนวน k บิต ซึ่งนำมาเข้ารหัส และ G คือ Generator Matrix ของการเข้ารหัส

2.2.2 การถอดรหัสแบบลิเนียร์บล็อก

การตรวจจับความผิดพลาดของข้อมูลที่ถูกส่งไปของรหัสแบบบล็อกนั้น วิธีการคือการนำข้อมูลที่รับได้มาทำการคูณกับเมตริกที่ใช้สำหรับตรวจจับความผิดพลาดของข้อมูล (H) ซึ่งผลลัพธ์ที่ได้จากการคูณนั้นเรียกว่าค่า Syndrome (S) ซึ่งเป็นค่าแสดงถึงลักษณะความผิดพลาดของข้อมูลที่เกิดขึ้นกับข้อมูลชุดนั้นๆ [4] ค่า Syndrome ทำให้สามารถนำไปใช้แก้ไขความผิดพลาดของข้อมูลให้ถูกต้องได้ โดยลักษณะการถอดรหัสแบบลิเนียร์บล็อกมีการทำงานดังสมการ

$$S = (U + e)H^T \quad (2.3)$$

โดย e คือค่าความผิดพลาดที่เกิดขึ้น และ H จะมีความสัมพันธ์กับ G ซึ่งทำให้ UH^T มีค่าเป็นศูนย์ ดังนั้นค่า Syndrome ที่ได้นั้นมีค่าเท่ากับ

$$S = eH^T \quad (2.4)$$

ค่า Syndrome ที่ได้จากการคำนวณนั้นใช้ในการตรวจสอบว่ามีการผิดพลาดเกิดขึ้นที่บิตใด ทำให้สามารถทำการแก้ไขข้อมูลให้เกิดความผิดพลาดให้ถูกต้องได้ แต่รหัสแบบลิเนียร์บล็อกนั้นมีข้อจำกัดในการตรวจสอบหาตำแหน่งผิดพลาดถ้าความผิดพลาดเกิดขึ้นเกินขอบเขตของรหัส

2.2.3 ความสามารถในการตรวจจับและแก้ไขความผิดพลาด

ความสามารถในการตรวจจับและแก้ไขความผิดพลาด เป็นตัวบ่งชี้ประสิทธิภาพของรหัส การวิเคราะห์คุณสมบัติต่างๆ ของการเข้ารหัสต้องมีการพิจารณาถึงองค์ประกอบหลายส่วน ได้แก่ น้ำหนักแฮมมิง (Hamming Weight) และระยะห่างแฮมมิง (Hamming Distance) สำหรับค่าน้ำหนักแฮมมิงหรือ $w(U)$ นั้นหมายถึงจำนวนบิตของข้อมูลที่ไม่เท่ากับ “0” ซึ่งในกรณีของข้อมูลไบนารี หมายถึงจำนวนข้อมูลที่เป็น “1” สำหรับค่าระยะห่างแฮมมิงระหว่างข้อมูล U และ V หรือ $d(U, V)$ นั้นหมายถึงจำนวนบิตของข้อมูลทั้งสองที่มีค่าแตกต่างกัน ซึ่งทั้งสองค่านี้มีความสัมพันธ์กันดังสมการ [4]

$$d(U, V) = w(U + V) \quad (2.5)$$

ข้อมูลที่ผ่านมาการเข้ารหัสนั้นค่าระยะห่างแฮมมิงที่อาจเท่ากันหรือแตกต่างกันก็ได้ แต่ในการวิเคราะห์การทำงานจะพิจารณาเพียงข้อมูลที่มีค่าระยะห่างแฮมมิงที่มีค่าน้อยที่สุด ซึ่งเรียกว่า ค่าระยะห่างแฮมมิงต่ำสุด (Minimum Hamming Distance) หรือ $d_{\min}$ โดย $d_{\min}$ จะถูกนำไปใช้ในการ
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คำนวณความสามารถในการแก้ไขความผิดพลาดของข้อมูล สำหรับการเข้ารหัสแบบบล็อกนั้น ความสัมพันธ์ระหว่างค่า $d_{\min}$ และจำนวนบิตที่มากที่สุดของข้อมูลที่เกิดความผิดพลาดที่ยังสามารถแก้ไขให้ถูกต้องได้ (t) แสดงดังสมการ

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \quad (2.6)$$

ส่วนค่าของจำนวนบิตที่มากที่สุดของข้อมูลที่ยังสามารถตรวจจับได้ว่าเกิดความผิดพลาดขึ้น (Error Detecting Capability) นั้นแสดงดังสมการ

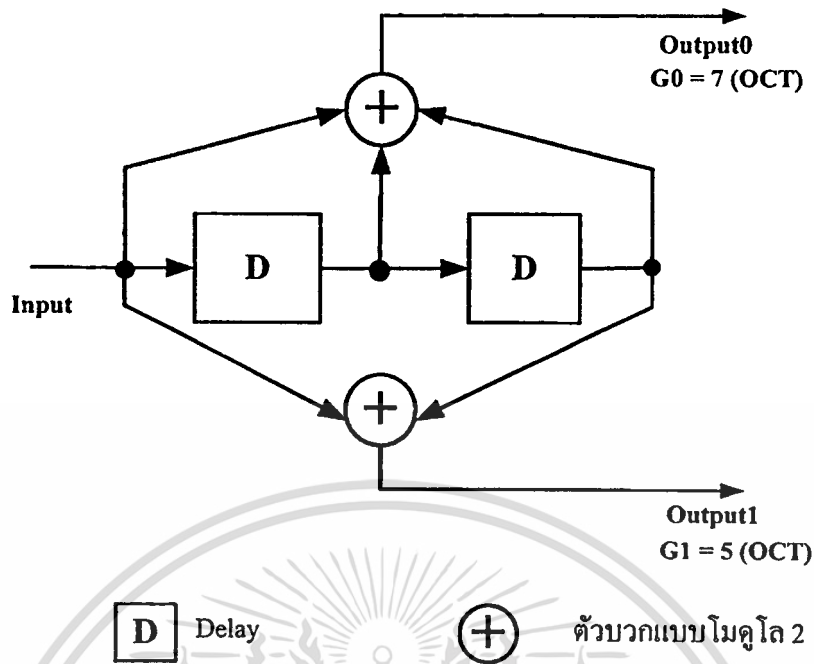
$$e = d_{\min} - 1 \quad (2.7)$$

2.3 รหัสคอนโวลูชัน (Convolution Code)

รหัสคอนโวลูชัน (Convolution Code) ถูกนำมาเผยแพร่ในปี 1955 โดย Elias การเข้ารหัสคอนโวลูชันเป็นการหาความสัมพันธ์ของกลุ่มข้อมูลอินพุตที่เรียงลำดับอย่างต่อเนื่อง [7] โดยวงจรเข้ารหัสคอนโวลูชันประกอบด้วยตัวคูณ (D) หรือชิฟเรจิสเตอร์ (Shift Register) และตัวบวกแบบมอดูโล 2 (Modulo-2 Adder) หรือ Exclusive OR Gate ซึ่งแสดงดังรูปที่ 2.2

2.3.1 การเข้ารหัสคอนโวลูชัน

การทำงานของวงจรเข้ารหัสคอนโวลูชันนั้น ข้อมูลอินพุต (k) จะถูกส่งเข้าอินพุตของวงจรเข้ารหัสคอนโวลูชันแบบต่อเนื่อง โดยข้อมูลเฮดพุตของการเข้ารหัส (n) เกิดจากการนำข้อมูลอินพุต (k) และข้อมูลในชิฟเรจิสเตอร์มาบวกแบบมอดูโล 2 ซึ่งขึ้นอยู่กับค่า Constraint Length (K) ซึ่งเป็นค่าที่แสดงจำนวนของข้อมูลอินพุต (k) และข้อมูลสูงสุดในชิฟเรจิสเตอร์ที่นำมาบวกแบบมอดูโล 2 ในการหาค่าเฮดพุต และค่า Generator Polynomial (g) ซึ่งเป็นค่าที่ใช้แสดงตำแหน่งข้อมูลอินพุตและข้อมูลในชิฟเรจิสเตอร์ที่นำมาบวกแบบมอดูโล 2 ค่า Generator Polynomial นี้ นิยมเขียนในรูปแบบเลขฐานแปด เช่น ตำแหน่งข้อมูลที่นำมาบวกแบบมอดูโล 2 เป็น “111” นิยมเขียนเป็น 7₈ ดังตัวอย่างวงจรเข้ารหัสคอนโวลูชันแสดงในรูปที่ 2.2



รูปที่ 2.2 วงจรเข้ารหัสคอนโวลูชันที่ค่า R เท่ากับ $1/2$ และค่า K เท่ากับ 3

ส่วนค่าอัตราเข้ารหัส (R) เป็นอัตราส่วนของจำนวนข้อมูลอินพุต (k) ต่อจำนวนเอาต์พุต (n) หรือ k/n โดยจำนวนเอาต์พุตของการเข้ารหัสคอนโวลูชันมีจำนวนมากกว่าอินพุต จากวงจรตัวอย่างการเข้ารหัสคอนโวลูชันในรูปที่ 2.2 มีค่าอัตราส่วนของเข้ารหัสเท่ากับ $1/2$ ค่า Constraint Length เท่ากับ 3 และมีค่า Generator Polynomials G_0 เท่ากับ 7_8 และ G_1 เท่ากับ 5_8

2.3.2 การวิเคราะห์การทำงานของวงจรเข้ารหัสคอนโวลูชัน

การวิเคราะห์การทำงานของวงจรเข้ารหัสคอนโวลูชันนั้นใช้ค่า Generator Polynomial ในการคำนวณหาผลลัพธ์ของการเข้ารหัสของเอาต์พุตแต่ละค่า กรณีของวงจรเข้ารหัสในรูปที่ 2.2 นั้นค่า Generator Polynomial ในรูปแบบเลขฐานสองแสดงคือ

$$g_0 = [1 \ 0 \ 0]$$

$$g_1 = [1 \ 1 \ 1]$$

การคำนวณหาค่าเอาต์พุตที่ได้จากการเข้ารหัสนั้นนำค่า Generator Polynomial มาคอนโวลูชัน (Convolution) กับข้อมูลอินพุตดังสมการ

$$\text{Output}_i = [\text{Input}] * g_i \quad (2.8)$$

ตัวอย่างในกรณีวงจรเข้ารหัสในรูปที่ 2.2 ข้อมูลที่เข้ารหัสมีค่าเท่ากับ “1 0 0 1” การคำนวณหาค่าแฮดพุตที่ได้จากการเข้ารหัสสามารถคำนวณได้ดังนี้

$$\text{Output}_0 = [1\ 0\ 0\ 1] * [1\ 0\ 0] = 1\ 0\ 0\ 1\ 0\ 0$$

$$\text{Output}_1 = [1\ 0\ 0\ 1] * [1\ 1\ 1] = 1\ 1\ 1\ 1\ 1\ 1$$

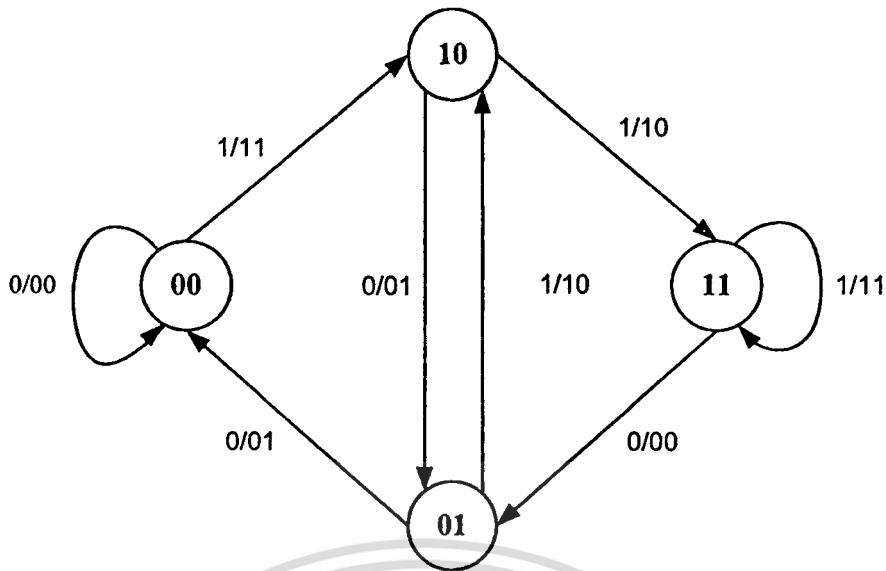
ค่าแฮดพุตที่ได้จากการเข้ารหัสข้อมูลนั้นเกิดจากข้อมูลที่ได้จาก O/P_0 และ O/P_1 ซึ่งเมื่อนำมารวมกันมีค่าเท่ากับ “11 01 01 11 01 01”

2.3.3 แผนภาพแสดงการทำงานของวงจรเข้ารหัสคอนโวลูชัน

การวิเคราะห์การทำงานของวงจรเข้ารหัสคอนโวลูชัน นอกจากการใช้สมการ Generator Polynomial ในการวิเคราะห์การทำงานแล้วยังสามารถใช้แผนภาพ (Diagram) วิเคราะห์การทำงานได้ โดยแผนภาพแสดงความสัมพันธ์ระหว่างข้อมูลที่ถูกเก็บไว้ในวงจรเข้ารหัส ข้อมูลอินพุตที่ถูกป้อนเข้ามา ณ เวลานั้น และค่าของแฮดพุตที่ถูกส่งออกไป ซึ่งรูปแบบของแผนภาพที่ใช้แสดงการทำงานของวงจรเข้ารหัสนั้นมีสามรูปแบบ [4] คือ แผนภาพสเตต (State Diagram) แผนภาพทรี (Tree Diagram) และแผนภาพ Trellis (Trellis Diagram) ซึ่งมีรายละเอียดดังต่อไปนี้

2.3.3.1 แผนภาพสเตต (State Diagram)

แผนภาพสเตต เป็นแผนภาพแสดงค่าของข้อมูลในชิฟเรจิสเตอร์และแฮดพุตของการเข้ารหัสคอนโวลูชันซึ่งเปลี่ยนตามข้อมูลอินพุตที่ป้อนเข้ามา ตัวอย่างของแผนภาพสเตตของวงจรเข้ารหัสในรูปที่ 2.2 แสดงดังรูปที่ 2.3 โดยตัวเลขที่อยู่ในวงกลมแต่ละวงนั้นหมายถึงสเตต (State) หรือสถานะต่างๆ ของข้อมูลที่ถูกเก็บไว้ในชิฟเรจิสเตอร์ โดยค่าสเตตเท่ากับ 2^{K-1} สเตต ซึ่งวงจรเข้ารหัสตัวอย่างในรูปที่ 2.2 มีจำนวนสเตตทั้งหมด 4 สเตต สำหรับลูกศรที่ถูกแสดงไว้ในรูปนั้นแสดงถึงลักษณะการเปลี่ยนแปลงการทำงานจากสถานะหนึ่งไปเป็นอีกสถานะหนึ่ง ซึ่งขึ้นอยู่กับข้อมูลที่ป้อนเข้ามา ณ เวลานั้นๆ เช่น กรณีที่ข้อมูลที่เก็บไว้มีค่าเป็น “00” เมื่อมีข้อมูล “1” ป้อนเข้ามาจะมีผลทำให้ข้อมูลที่เก็บไว้ถูกเปลี่ยนไปเป็น “10” และ “11” คือ แฮดพุตของการเข้ารหัสซึ่งแทนด้วยสัญลักษณ์ “1/11” แต่ถ้าหากว่ามีการป้อนข้อมูล “0” เข้ามา วงจรยังคงมีสถานะเป็น “00” เหมือนเดิม และ “00” คือ แฮดพุตของการเข้ารหัสซึ่งแทนด้วยสัญลักษณ์ “0/00” ซึ่งลักษณะแผนภาพสเตตแสดงดังรูปที่ 2.3



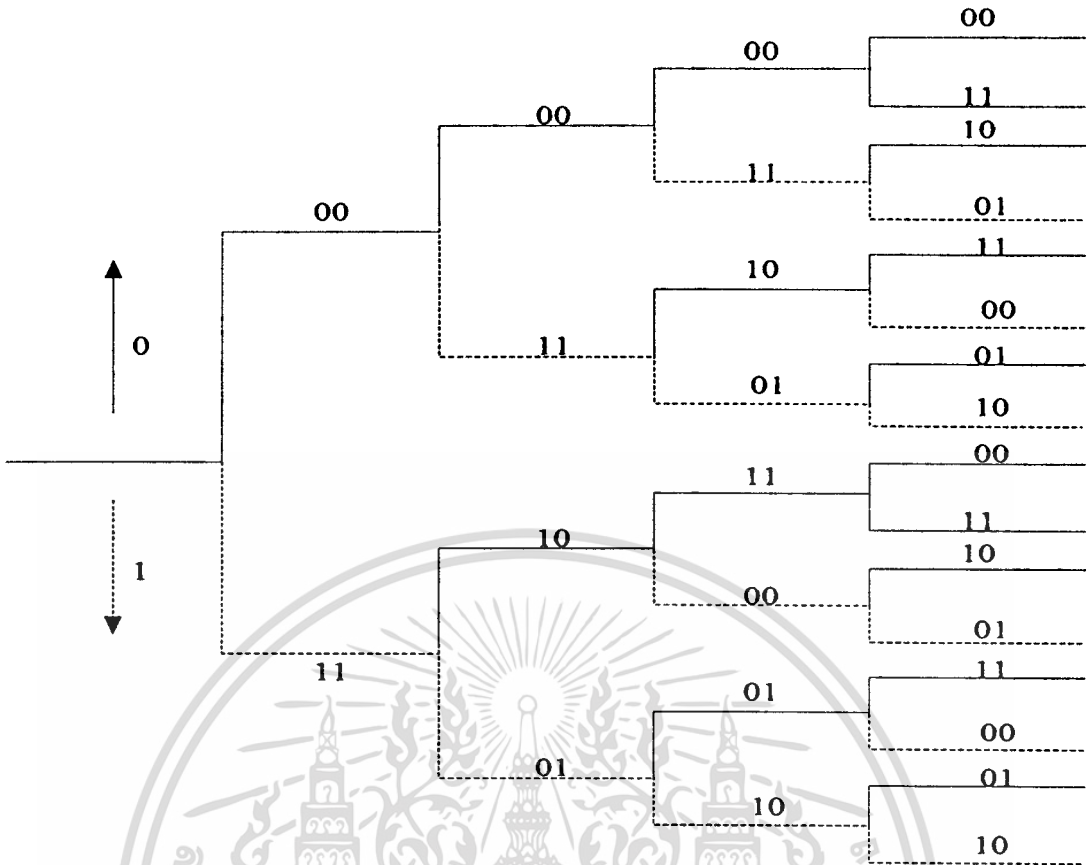
รูปที่ 2.3 แผนภาพสเตทที่ R เท่ากับ $1/2$ และ K เท่ากับ 3

2.3.3.2 แผนภาพทรี (Tree Diagram)

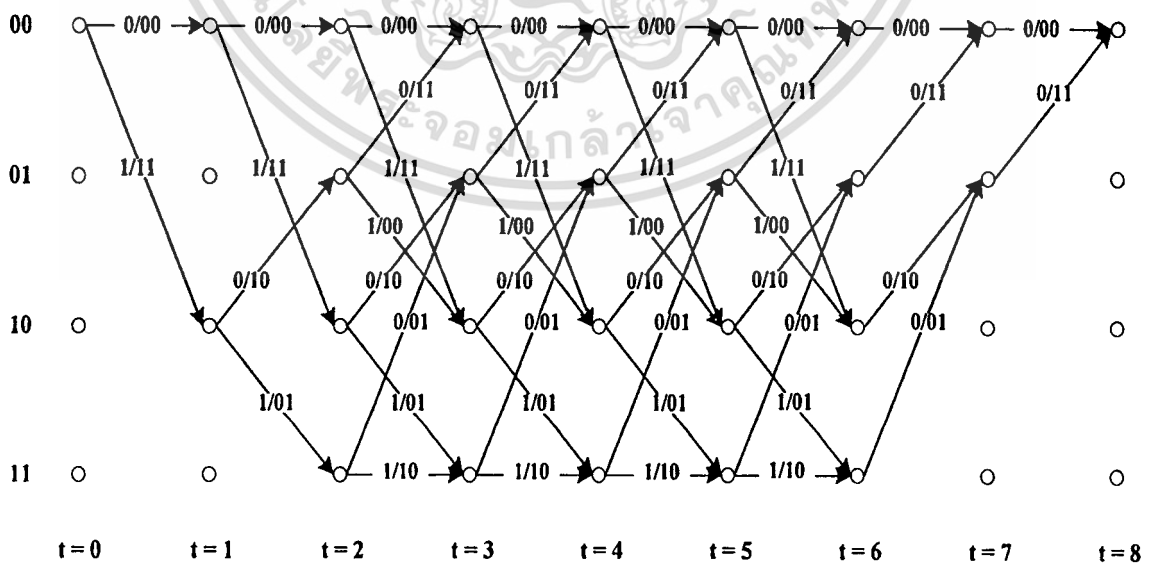
แผนภาพทรี (Tree Diagram) เป็นแผนภาพที่แสดงค่าเอาต์พุตของการเข้ารหัสที่ได้จากการป้อนข้อมูลอินพุต โดยแผนภาพทรีสามารถแสดงการเปลี่ยนแปลงตามเวลาได้ [4] การพิจารณาการทำงานนั้นเริ่มจากตำแหน่งราก (Root) ของแผนภาพทรีซึ่งอยู่ในตำแหน่งซ้ายมือสุด โดยข้อมูลที่ป้อนเข้าวงจรเข้ารหัสเป็นตัวกำหนดทิศทางของข้อมูลในแผนภาพทรี โดยให้เลื่อนตำแหน่งไปข้างบนเมื่อมีการรับข้อมูล “0” และเลื่อนตำแหน่งลงล่างเมื่อมีการรับข้อมูล “1” ซึ่งหลังจากที่เลื่อนตำแหน่งแล้ว แผนภาพจะแสดงข้อมูลเอาต์พุตของข้อมูลที่ถูกเข้ารหัส ณ เวลานั้นๆ โดยข้อมูลนี้อยู่เหนือเส้นทาง ณ ตำแหน่งนั้นๆ ตัวอย่างการทำงานของแผนภาพทรีวงจรเข้ารหัสคอนไวลูชันในรูปที่ 2.2 แสดงในรูปที่ 2.4

2.3.3.3 แผนภาพ Trellis (Trellis Diagram)

เมื่อความยาวของข้อมูลอินพุตมีความยาวมากๆ ทำให้เกิดปัญหาค้นหาขึ้นกับแผนภาพทรี [4] จากรูปที่ 2.4 ที่เวลา t_4 แผนภาพทรีจะซ้ำแบบเดิม ซึ่งครึ่งบนและครึ่งล่างของแผนภาพทรีนั้นมีค่าเหมือนกัน ดังนั้นสามารถนำแผนภาพทรีนี้มาเขียนแผนภาพใหม่ซึ่งเรียกว่า แผนภาพ Trellis (Trellis Diagram) เพื่อใช้ในการวิเคราะห์การเข้าและถอดรหัสคอนไวลูชัน โดยแผนภาพ Trellis สามารถแสดงสถานะของข้อมูลหรือสเตท (State) ของการทำงานมาแสดง ณ เวลาต่างๆ ได้ ซึ่งตัวอย่างแผนภาพ Trellis ของวงจรเข้ารหัสในรูปที่ 2.2 แสดงดังรูปที่ 2.5 โดยแนวตั้งเป็นค่าสเตท ส่วนทางแนวนอนเป็นค่าเวลา

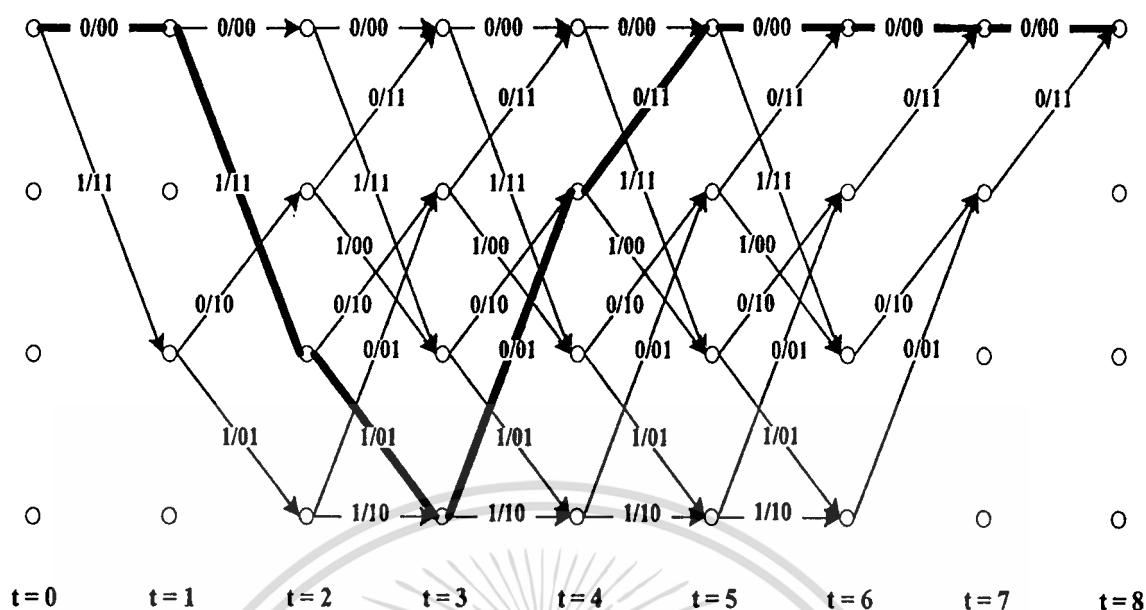


รูปที่ 2.4 แผนภาพทรีที่ R เท่ากับ $1/2$ และ K เท่ากับ 3 [4]



รูปที่ 2.5 แผนภาพ Trellis ที่ R เท่ากับ $1/2$ และ K เท่ากับ 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.6 ตัวอย่างการใช้งานแผนภาพ Trellis

ส่วนเส้นทางต่างๆ ที่อยู่ในแผนภาพ Trellis นั้นแสดงถึงลักษณะการเปลี่ยนแปลงสถานะของวงจร โดยตัวเลขที่อยู่เหนือทางเดินในแต่ละเส้นทางนั้นแสดงถึงข้อมูลที่ป้อนเข้ามา k และข้อมูลที่ถูส่งออกไป n ณ เวลานั้นๆ โดยค่าสเตตของแผนภาพนั้นเท่ากับ 2^{K-1} และค่าสาขา (Branch) ที่เข้ามาที่จุดเชื่อมต่อ (Node) เท่ากับ 2^k ในแต่ละเวลาลักษณะของแผนภาพ Trellis จะมีรูปแบบที่คล้ายกัน แต่แตกต่างกันเฉพาะส่วนหัวและท้ายเท่านั้น ซึ่งเป็นผลมาจากข้อมูลที่เก็บอยู่ในวงจรนั้นมีค่าเริ่มต้นจากสถานะที่มีข้อมูล “0” และจบลงที่สถานะข้อมูล “0” เช่นกัน ซึ่งขนาดของความยาวในแผนภาพ Trellis ขึ้นอยู่กับข้อมูลที่ถูนำมาเข้ารหัส ตัวอย่างเช่นกรณีที่มีการป้อนข้อมูลอินพุต “0 1 1 0 0 0 0 0” เข้ามาภายในวงจรเข้ารหัส เมื่อใช้แผนภาพ Trellis ข้อมูลเอด์พุตมีค่าเท่ากับ “00 11 10 00 01 00 00 00” ซึ่งแสดงได้ดังรูปที่ 2.6 โดยข้อมูลที่ได้หลังจากการเข้ารหัสแสดงในเส้นทึบ

2.3.4 การถอดรหัสคอนโวลูชัน

การถอดรหัสคอนโวลูชันนั้นมีขั้นตอนที่ซับซ้อนกว่าวิธีการเข้ารหัสแบบอื่น ซึ่งวิธีที่ใช้ในการถอดรหัสคอนโวลูชันนั้นมี 3 วิธี คือ Sequential Decoding, Threshold Decoding และ Viterbi Decoding ซึ่งมีรายละเอียดดังนี้

- วิธี Sequential Decoding เป็นวิธีการถอดรหัสคอนโวลูชันแบบแรกที่น่าเสนอโดย Wozencraft ในปี 1961 [4] วิธีการถอดรหัสใช้แผนภาพทรีมาใช้ในการงานในการค้นหาเส้นทางของข้อมูล วิธีนี้มีข้อเสียคือใช้เวลาในการถอดรหัสนานและมีการทำงานที่ซับซ้อน
- วิธี Threshold Decoding เป็นวิธีที่น่าเสนอโดย Massey ในปี 1963 [5] การถอดรหัสด้วยวิธีนี้ใช้หน่วยตัดสินใจที่เรียกว่า Threshold Gate ในการพิจารณาตัดสินใจแก้ไขความผิดพลาด ซึ่งวิธีนี้มีความซับซ้อนน้อยแต่มีประสิทธิภาพการถอดรหัสที่ต่ำ
- วิธี Viterbi Decoding เป็นวิธีที่น่าเสนอโดย A. J. Viterbi ในปี 1967 [7] การถอดรหัสคอนโวลูชันด้วย Viterbi Decoding นิยมใช้กันมากเนื่องจากให้ประสิทธิภาพสูง แต่มีวิธีนี้มีความซับซ้อนในการทำงานสูงและใช้หน่วยความจำขนาดใหญ่ในการถอดรหัส วิธีนี้จะใช้อัลกอริทึมที่เรียกว่า Maximum Likelihood Decoding และแผนภาพ Trellis ในการถอดรหัส

2.4 การถอดรหัสด้วยวิธี Viterbi

การถอดรหัสคอนโวลูชันด้วยวิธี Viterbi เป็นวิธีการถอดรหัสที่ให้ประสิทธิภาพสูง ซึ่งลักษณะการทำงานของวิธีการถอดรหัสด้วยวิธี Viterbi เป็นแบบ Maximum Likelihood Decoding โดยผลลัพธ์ที่ได้จากการถอดรหัสจะเป็นเส้นทางเพียงเส้นทางเดียวที่มีความน่าจะเป็นสูงสุดจากเส้นทางทั้งหมดในแผนภาพ Trellis ซึ่งมีลักษณะคล้ายกับข้อมูลที่ถูกส่งมากที่สุด โดยสำหรับกระบวนการที่จะใช้ในการค้นหาเส้นทางนั้นเรียกว่า “Viterbi Algorithm Decoding” [7]

2.4.1 รูปแบบการทำงานของวิธีการถอดรหัสด้วยวิธี Viterbi

การรูปแบบการทำงานของวิธีการถอดรหัสด้วยวิธี Viterbi มีสองแบบ ได้แก่ Hard Decision และ Soft Decision ซึ่งอธิบายได้ดังนี้คือ การทำงานของวงจรถอดรหัสแบบ Hard Decision นั้น ข้อมูลอินพุตที่รับจากส่วนของมอดูเลชันที่ถูกจัดระดับเป็นสองระดับคือ “0” และ “1” ต่ออินพุตหนึ่งบิต ส่วนการทำงานของวงจรถอดรหัสแบบ Soft Decision นั้น ข้อมูลอินพุตที่รับจากส่วนของมอดูเลชันที่ถูกจัดระดับมากกว่าสองระดับต่ออินพุตหนึ่งบิต เช่น 4 ระดับ (2 บิต) หรือ 8 ระดับ (3 บิต) เป็นต้น การทำงานแบบ Soft Decision นั้นมีการทำงานที่ซับซ้อนกว่าแบบ Hard Decision

แต่ให้ค่า Coding Gain ที่มากกว่า โดยการทำงานแบบ Hard Decision นั้นใช้สำหรับช่องสัญญาณ Binary Symmetric หรือ Discrete Memoryless Channel ส่วนการทำงานแบบ Soft Decision นั้นเหมาะสมกับช่องสัญญาณแบบ AWGN (Additive White Gaussian Noise) การทำงานแบบ Soft Decision นั้นให้ค่า Coding Gain ที่มากกว่าแบบ Hard Decision ประมาณ 2 dB ที่ Soft Decision แบบ 8 ระดับ [4]

2.4.2 การถอดรหัสด้วย Viterbi Algorithm

การทำงานของถอดรหัสด้วย Viterbi Algorithm เป็นการหาเส้นทางในแผนภาพ Trellis ซึ่งมีลักษณะที่ใกล้เคียงกับข้อมูลที่ถูกส่งมามากที่สุด โดย Viterbi Algorithm นั้น มีขั้นตอนในการทำงานแบ่งเป็น 3 ขั้นตอน [8] ดังต่อไปนี้

ขั้นตอนที่ 1 Branch Metric Generation

ขั้นตอนนี้คำนวณหาค่า Branch Metric จากข้อมูลอินพุตที่รับเข้ามา r กับค่าเอาต์พุตของการเข้ารหัส C การคำนวณหาค่า Branch Metric ต้องคำนวณทุกๆ สาขาหรือ Branch ตามสเตทการทำงานโดย Branch เท่ากับ 2^K การคำนวณหาค่า Branch Metric แสดงดังสมการ [8]

$$BM_{i,j,n} = (r_n - C_{i,j})^2 \quad (2.9)$$

โดย ค่า BM แทนค่า Branch Metric ระหว่างสเตท i ไปยังสเตท j ณ เวลา n

ค่า r แทนค่า ข้อมูลอินพุตที่รับเข้ามา ณ เวลา n

ค่า C แทนค่าเอาต์พุตของการเข้ารหัสระหว่างสเตท i ไปยังสเตท j ณ เวลา n

ขั้นตอนที่ 2 Survivor Path and Path Metric Update

ขั้นตอนนี้คำนวณหาค่า Survivor Path และ Path Metric จากจำนวนสเตทการทำงานทั้งหมดค่า Path Metric ที่เลือกไว้เพื่อใช้ในการหาค่า Path Metric ครั้งต่อไป (Update) ส่วนค่า Survivor Path เป็นค่าที่ใช้ในการตัดสินใจหาค่าเอาต์พุต โดยการคำนวณหาค่า Survivor Path และ Path Metric นั้นค่าของ Branch Metric และ Path Metric จะถูกเข้าด้วยกัน ซึ่งผลการบวกนั้นมีสองค่าที่เข้ามาในแต่ละจุดเชื่อมต่อ (Trellis Node) ของแผนภาพ Trellis โดยค่า Path Metric เป็นค่าที่เลือกจากค่าผลบวกที่น้อยกว่า ส่วนค่า Survivor Path เป็นสเตทการทำงานที่น้อยกว่าจากการเลือก Path Metric ซึ่งแสดงดังสมการ

$$PM_{j,n} = \min(PM_{i,n-1} + BM_{i,j,n}, PM_{i+1,n-1} + BM_{i+1,j,n}) \quad (2.10)$$

โดย ค่า PM แทน Path metric ระหว่างสเตต i ไปยังสเตต j ณ เวลา n

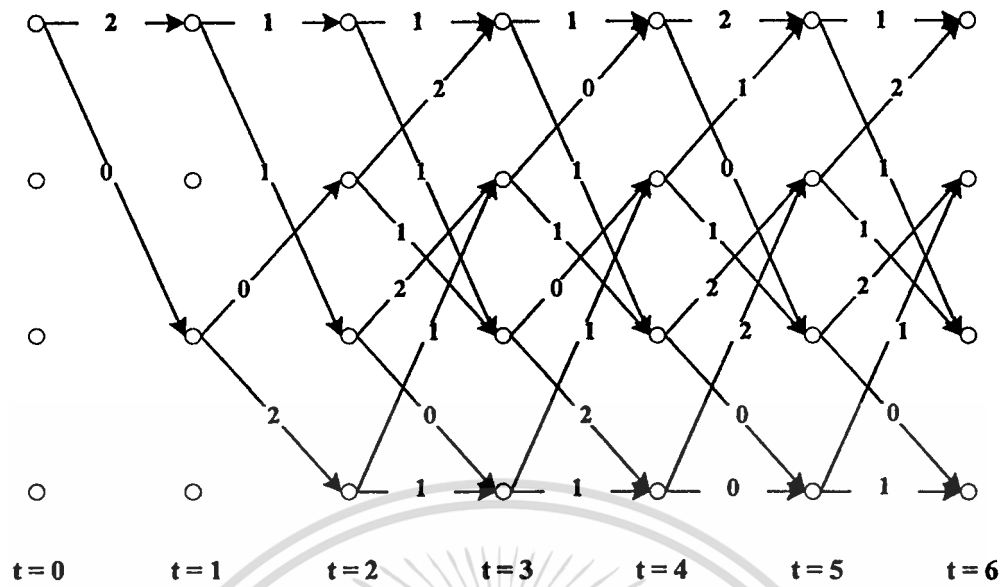
ค่า BM แทนค่า Branch metric ระหว่างสเตต i ไปยังสเตต j ณ เวลา n

ขั้นตอนที่ 3 Optimum Paths Trace Back

ขั้นตอนนี้เป็นขั้นตอนการตัดสินใจหาค่าเอาต์พุต โดยใช้ค่า Survivor Path ในแต่ละสเตตที่บันทึกไว้มาตัดสินใจเลือกเส้นทางของข้อมูล โดยการตัดสินใจหาเส้นทางของข้อมูลจะเริ่มจาก Survivor Path ในอดีต (Trace Back) โดยในการเริ่มต้นจากเวลาผ่านไป L (Latency) โดยในทางปฏิบัติการ L ต้องมีค่ามากกว่าห้าเท่าของค่า K (Constrain Length) จึงทำให้ข้อมูลที่ถูกลอครหัสมีความถูกต้องสูง

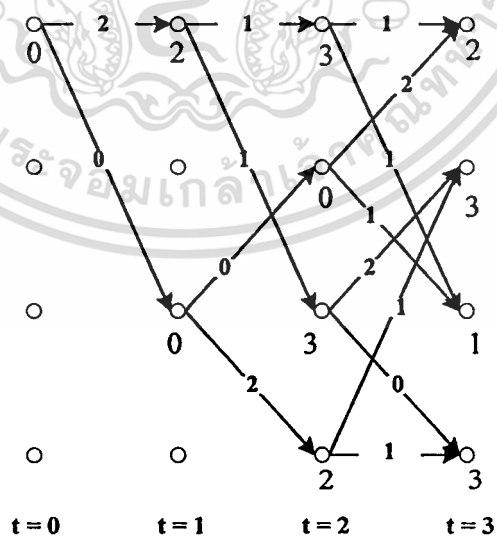
2.4.3 การลอครหัสคอนโวลูชันที่ R เท่ากับ $1/2$ และ K เท่ากับ 3 ด้วยวิธี Viterbi

กรณีของการลอครหัสข้อมูลที่ถูกเข้ารหัสคอนโวลูชันด้วยวงจรในรูปที่ 2.2 เมื่อป้อนข้อมูลข่าวสารอินพุตคือ “1 0 1 0 1 1” ข้อมูลเอาต์พุตคือ “1 1 0 1 1 0 0 1 1 0 1 0 0 0 0 1” สมมุติให้ข้อมูลที่รับได้นั้นเกิดความผิดพลาดขึ้นทำให้ข้อมูลที่รับได้เท่ากับ “1 1 0 1 1 0 0 1 1 1 1 0 0 0 0 1” ในวิธีการลอครหัสโดยใช้วิธี Viterbi นั้นการทำงานเริ่มต้นจากการนำข้อมูลที่ได้นั้นมาแบ่งออกเป็นชุดๆ ที่มีขนาดเท่ากับจำนวนค่ารหัสซึ่งในวงจรนี้มีค่าเท่ากับสองบิต ดังนั้นข้อมูลนี้จะถูกแบ่งออกเป็น “11 01 10 01 11 10 00 01” โดยข้อมูลนี้จะถูกนำมาเปรียบเทียบกับข้อมูลที่อยู่ในเส้นทางในแผนภาพ Trellis เพื่อหาค่า Branch Metric ระหว่างข้อมูลที่ทั้งสอง ในเวลา t_1 เป็นการนำข้อมูล “11” ที่รับได้มาเปรียบเทียบกับเส้นทางที่ลากจากสเตตเริ่มต้นที่มีข้อมูลเป็น “0” มายังสเตต “00” ในเวลาถัดไป ซึ่งมีข้อมูลในเส้นทางเป็น “0/00” ซึ่งค่า Branch Metric เท่ากับสอง และเมื่อเปรียบเทียบกับเส้นทางที่ไปสเตต “10” ในเวลาถัดไป ค่า Branch Metric เท่ากับศูนย์ ในเวลา t_2 นั้นข้อมูล “01” ที่รับได้จะถูกนำมาเปรียบเทียบกับเส้นทางและมีการคำนวณเช่นเดียวกันนี้ไปเรื่อยๆ จนครบทุกเส้นทาง จากนั้นในขั้นตอนต่อไปเป็นการคำนวณหาค่า Path Metric โดยในรูปที่ 2.7 แสดงการคำนวณหาค่า Branch Metric ในแผนภาพ Trellis ของวงจรเข้ารหัสในรูปที่ 2.2



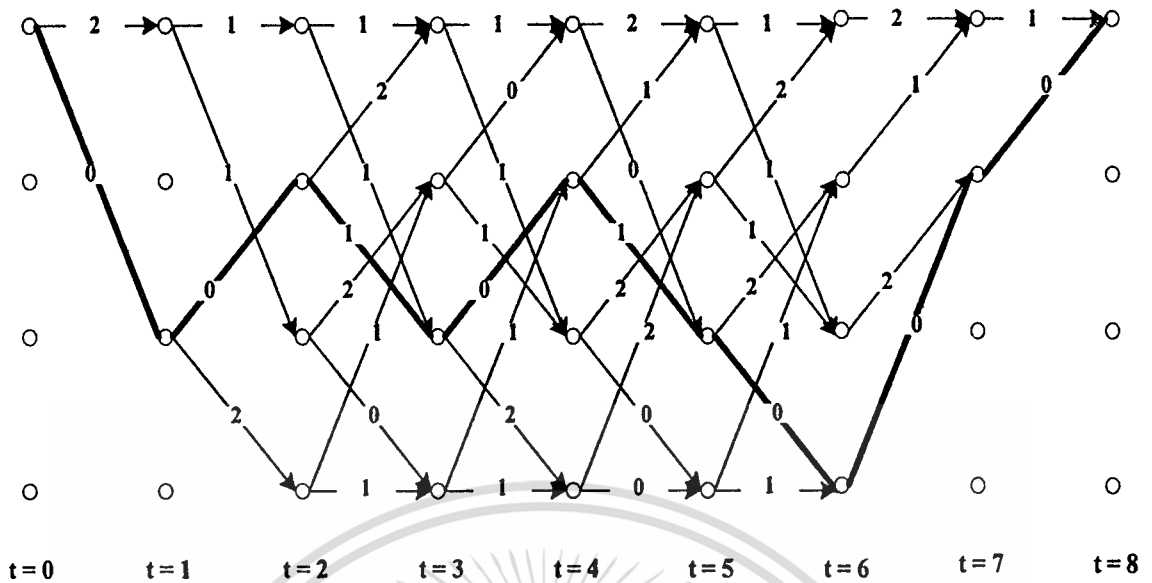
ข้อมูลอินพุต	1	0	1	0	1	1
ข้อมูลเอาต์พุต	11	01	10	01	10	10
ข้อมูลที่ได้รับได้	11	01	10	01	11	10

รูปที่ 2.7 ค่า Branch Metric ในแผนภาพ Trellis



รูปที่ 2.8 ค่า Path Metric ในเวลา t_1 ถึง t_3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



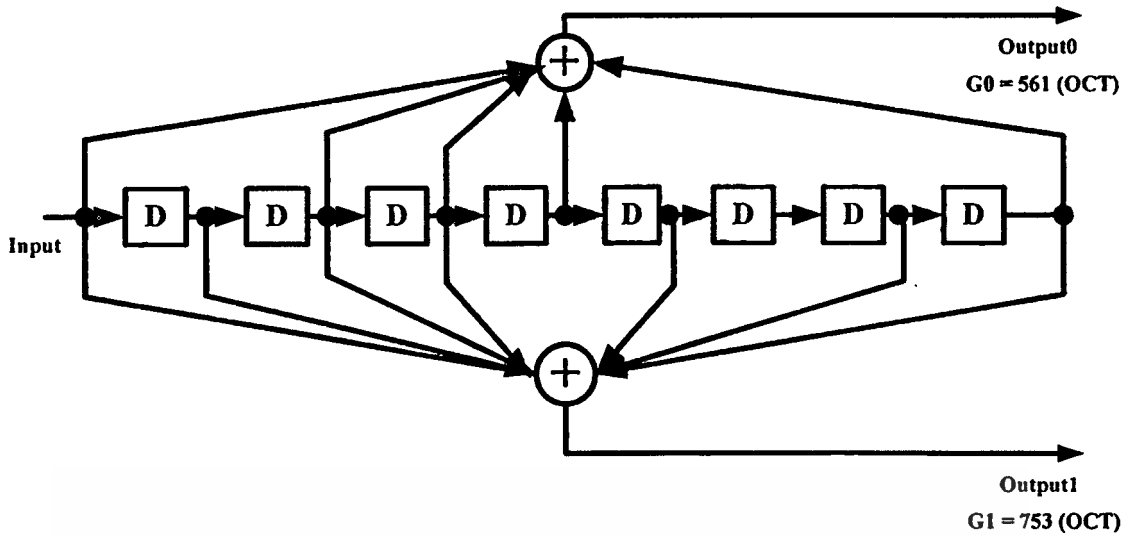
รูปที่ 2.9 เส้นทางของข้อมูลที่ได้รับการถอดรหัสด้วยวิธี Viterbi

การคำนวณค่า Path Metric เริ่มต้นจากสเตต “00” ที่อยู่ซ้ายบนสุดของรูปที่ 2.8 ซึ่งกำหนดให้มีค่า Path Metric เป็นศูนย์ กับสเตตที่เชื่อมต่อกับสเตตเริ่มต้นนี้ซึ่งมีสองสเตตคือ “00” และ “10” ในเวลาถัดไป การคำนวณค่า Path Metric จะนำ Branch Metric ที่อยู่ในเส้นทางนั้นมาบวกกับ Path Metric สะสมที่ถูกเก็บไว้ ผลลัพธ์ที่ได้จะถูกเก็บไว้ในสเตตที่อยู่ปลายทางของเส้นทางนั้นๆ เช่น ในการคำนวณค่า Path Metric สะสมที่สเตต “00” ในเวลา $t=3$ นั้น เป็นการตัดสินใจเลือกเส้นทางระหว่างเส้นทางที่มาจากสเตต “00” และสเตต “01” ของเวลา $t=2$ ซึ่งจะเลือกเส้นทางที่มีค่า Path metric สะสมที่น้อยที่สุด โดยสำหรับเส้นทางที่มาจากสเตต “00” มีค่า Path Metric สะสมเท่ากับ 4 ($3+1$) และเส้นทางที่มาจากสเตต “01” มีค่าเท่ากับ 2 ($0+2$) ดังนั้นจึงเลือกเส้นทางที่มาจากสเตต “01” ซึ่งได้ค่า Path metric เท่ากับ 2 การทำงานลักษณะนี้จะทำซ้ำจนครบทุกช่วงเวลา โดยในเวลา $t=1$ ถึง $t=3$ นั้นค่า Path Metric แสดงดังรูปที่ 2.8 เมื่อเวลาผ่านไปมากกว่าห้าเท่าของค่า Constrain Length จึงเริ่มการตัดสินใจเพื่อเลือกเส้นทางของข้อมูลที่ถูกถอดรหัส โดยคำนวณจากเส้นทางที่มีค่า Path Metric สะสมน้อยที่สุดเท่านั้นโดยการเริ่มต้นที่สเตต “00” ที่อยู่ขวาสุดของรูป จากนั้นจึงย้อนกลับไปในสเตตที่อยู่ซ้ายมือที่มีเส้นทางต่อกับสเตต “00” โดยเลือกเส้นทางที่ต่อกับสเตตที่มีค่า Path Metric สะสมที่มีค่าน้อยที่สุดไปยังสเตตที่อยู่ทางซ้ายมือจนมีการค้นหาเส้นทางครบทุกช่วงเวลา ผลลัพธ์ที่ได้จากการทำงานได้ข้อมูลที่ถอดรหัสเท่ากับ “1 0 1 0 1 1” ซึ่งแสดงดังรูปที่ 2.9

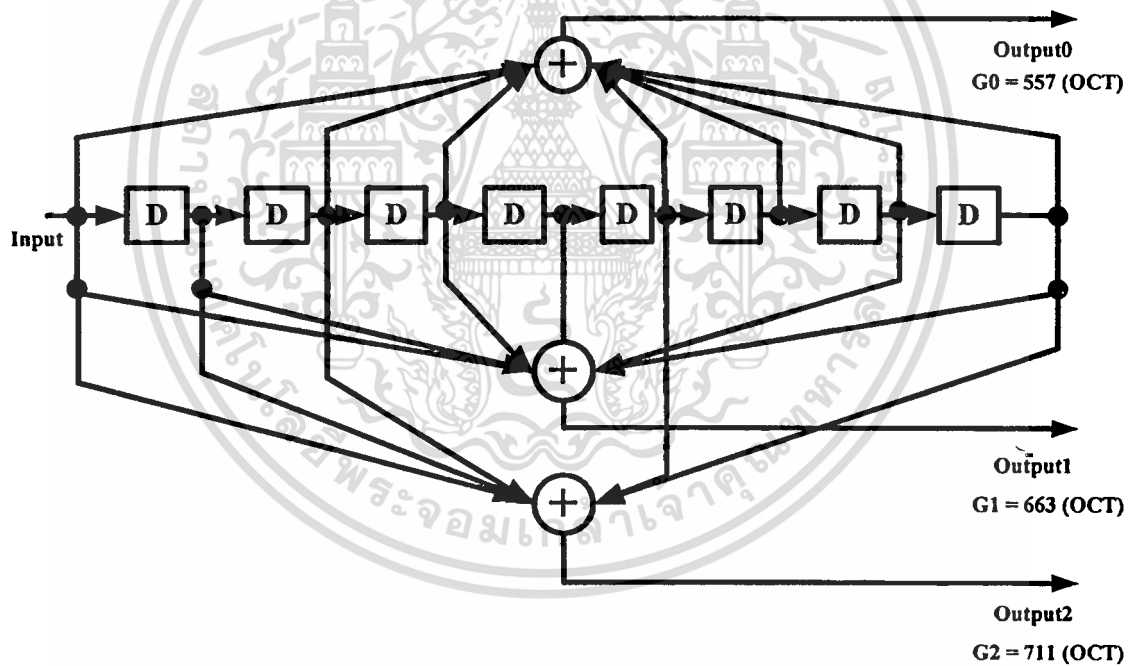
2.5 ระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 และการใช้งานรหัสแก้ไขความผิดพลาด

ระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 หรือ 3G เป็นระบบโทรศัพท์เคลื่อนที่ที่พัฒนาจากระบบโทรศัพท์เคลื่อนที่ยุคที่ 2 ซึ่งเป็นระบบโทรศัพท์แบบดิจิทัลที่ใช้สนทนาหรือสื่อสารข้อมูลด้วยความเร็วต่ำซึ่งมีอัตราเร็วในการรับส่งข้อมูล 9,600 bps [9] ในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นั้นถูกพัฒนาให้สามารถสื่อสารข้อมูลด้วยความเร็วสูง ทำให้สามารถรองรับการรับส่งข้อมูลที่มีปริมาณข้อมูลข่าวสารจำนวนมาก เช่น ข้อมูลแบบมัลติมีเดีย โดยมีอัตราการรับส่งข้อมูลไว้สูงถึง 2 Mbps เป็นต้น โดยระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นั้นมีสองมาตรฐานที่สำคัญ คือ CDMA2000 (IS-2000) ซึ่งเป็นระบบที่ถูกพัฒนาจากระบบโทรศัพท์เคลื่อนที่มาตรฐาน IS-95 โดย Telecommunication Industry Association (TIA) และ W-CDMA ซึ่งถูกพัฒนาโดย International Telecommunication Union (ITU) ซึ่งในปัจจุบันหน่วยงาน 3GPP (3rd Generation Partnership Project) เป็นผู้พัฒนามาตรฐานต่างๆ ของระบบนี้

สำหรับการป้องกันการผิดพลาดของข้อมูลในระบบของ 3GPP นั้นอยู่ในส่วนของภาคเข้ารหัสช่องสัญญาณ (Channel Coding) [1] ซึ่งใน TS25.212 ซึ่งเป็นรายละเอียดทางเทคนิค (Technical Specification) การมัลติเพล็กซ์และการเข้ารหัสช่องสัญญาณ (Multiplexing and Channel Coding) ของมาตรฐาน 3GPP ได้กำหนดให้ใช้เทคนิคการตรวจสอบความผิดพลาดของข้อมูลแบบ CRC (Cyclic Redundancy Check) ร่วมกับการเข้ารหัสป้องกันการผิดพลาดล่วงหน้า (FEC) ซึ่งได้กำหนดรหัสนี้ไว้สองแบบ คือ รหัสแบบคอนวูลูชันและรหัสแบบเทอร์โบ (Turbo Code) สำหรับรหัสแบบคอนวูลูชันนั้นใน TS25.212 ได้กำหนดให้มีค่า Constraint Length (K) เท่ากับ 9 อัตราการเข้ารหัส $1/2$ และ $1/3$ และ Generator Polynomial (561, 753)₈ และ (557, 663, 711)₈ ตามลำดับ ซึ่งรายละเอียดของวงจรคอนวูลูชันใน TS25.212 นั้นแสดงดังรูปที่ 2.10



(ก) อัตราการเข้ารหัสเท่ากับ 1/2 ค่า Constraint Length เท่ากับ 9



(ข) อัตราการเข้ารหัสเท่ากับ 1/3 ค่า Constraint Length เท่ากับ 9

รูปที่ 2.10 วงจรคอนโวลูชันที่ใช้ในมาตรฐาน TS25.212 [1]

(ก) อัตราการเข้ารหัสเท่ากับ 1/2 ค่า Constraint Length เท่ากับ 9

(ข) อัตราการเข้ารหัสเท่ากับ 1/3 ค่า Constraint Length เท่ากับ 9

บทที่ 3

เอฟพีจีเอพื้นฐานและการออกแบบ

ปัจจุบันเทคโนโลยีวงจรรวม (Integration Circuit) แบบดิจิทัลได้พัฒนาอย่างรวดเร็ว การออกแบบวงจรรวมนั้น สามารถรวมอุปกรณ์ดิจิทัลที่มีจำนวนมากให้อยู่ในรูปของวงจรรวมตัวเดียว ซึ่งสามารถลดค่าใช้จ่ายและการใช้พลังงานลง ในบทนี้กล่าวถึงความรู้พื้นฐานวงจรรวมแบบดิจิทัล โครงสร้างพื้นฐานของอุปกรณ์เอฟพีจีเอ Xilinx Vertex-II ตลอดจนเทคนิควิธีการการออกแบบและติดตั้งวงจรดิจิทัลบนอุปกรณ์เอฟพีจีเอ ประวัติความเป็นมาและโครงสร้างของภาษาเวริล็อก (Verilog HDL)

3.1 ความรู้พื้นฐานวงจรรวมแบบดิจิทัล

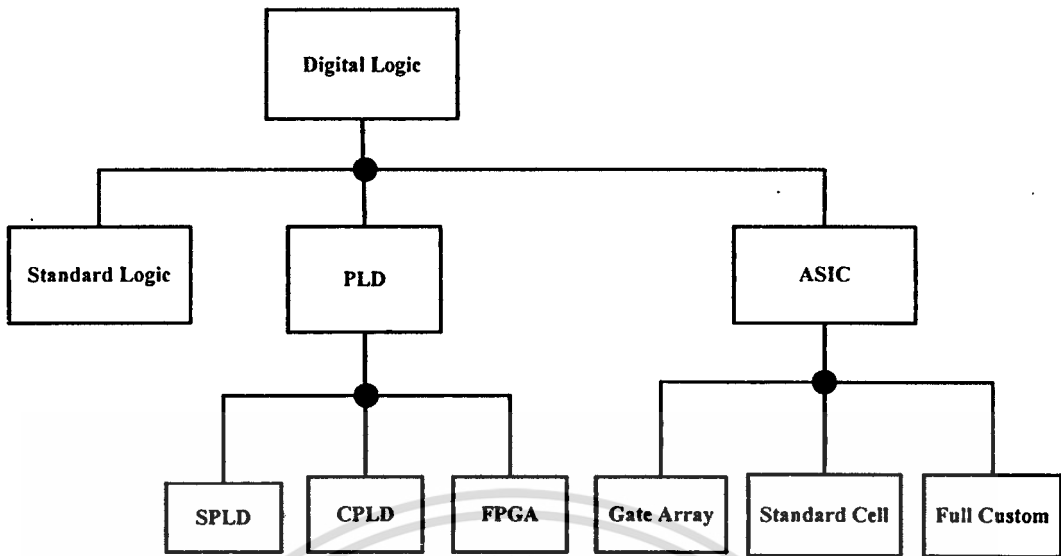
วงจรรวมแบบดิจิทัลสามารถแบ่งออกได้เป็น 3 ประเภทใหญ่ๆ คือ ลอจิกมาตรฐาน (Standard Logic) พีแอลดี (Programmable Logic Device: PLD) และเอสิก (Application Specific Integrated Circuit: ASIC) [10] ซึ่งแสดงในรูปที่ 3.1

3.1.1 วงจรรวมแบบลอจิกมาตรฐาน (Standard Logic)

วงจรรวมประเภทลอจิกมาตรฐานเป็นการออกแบบวงจรดิจิทัลที่ใช้วงจรรวมที่ออกแบบเป็นมาตรฐานไว้แล้ว เช่น วงจรรวมแบบทีทีแอล (TTL) หรือแบบซีเอ็มอส (CMOS) ชุดควบคุม เช่น ไมโครคอนโทรลเลอร์ หน่วยความจำต่างๆ เป็นต้น วงจรรวมประเภทนี้มีขนาดใหญ่และมีการใช้พลังงานมาก ซึ่งปัจจุบันการออกแบบวงจรดิจิทัลนิยมใช้อุปกรณ์พีแอลดีและเอสิกมากกว่า เนื่องจากมีขนาดเล็กและประสิทธิภาพสูง

3.1.2 วงจรรวมแบบพีแอลดี (Programmable Logic Device: PLD)

วงจรรวมดิจิทัลประเภทพีแอลดีเป็นอุปกรณ์ที่สามารถโปรแกรมสร้างวงจรดิจิทัลภายในได้ โดยพีแอลดีประกอบด้วยกลุ่มของลอจิกพื้นฐานทั้งวงจรคอมบิเนชัน (Combination) วงจรซีควENTIAL (Sequential) [11] การออกแบบและโปรแกรมพีแอลดีสามารถทำได้เองโดยใช้คอมพิวเตอร์ส่วนบุคคล ซึ่งการออกแบบวงจรดิจิทัลด้วยอุปกรณ์พีแอลดีนั้นใช้เวลาน้อยกว่าการออกแบบด้วยวงจรรวมประเภทเอสิก และมีต้นทุนที่ถูกกว่าเอสิกในกรณีที่ต้องการผลิตอุปกรณ์จำนวนน้อย เพราะในการออกแบบนั้นไม่ต้องเสียค่า NRE (Non-Recurring Engineering cost)

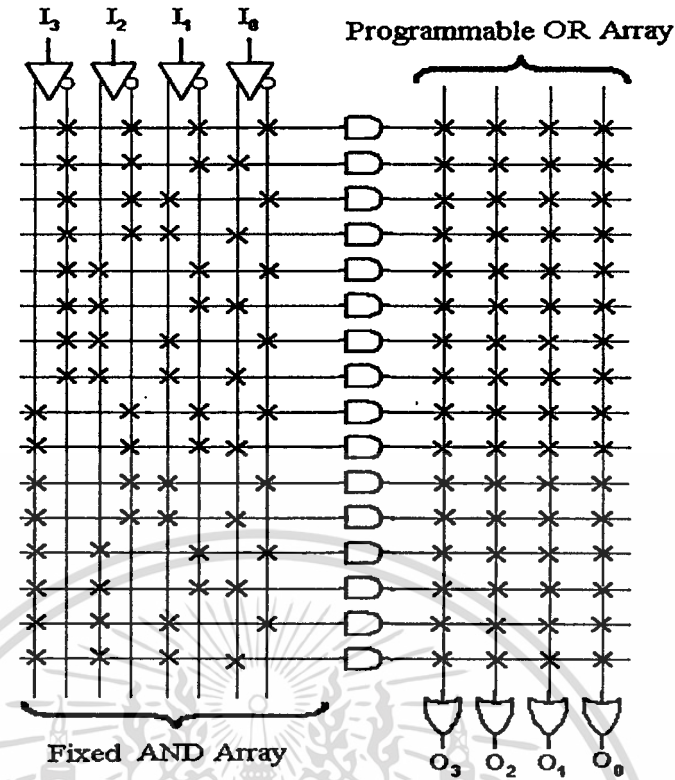


รูปที่ 3.1 โค้ดแแกรมเทคโนโลยีวงจรรวมแบบดิจิทัล [10]

ซึ่งเป็นค่าติดตั้งเครื่องจักร (Set-up Machine) ในการผลิตวงจรรวมซึ่งมีราคาแพง และข้อดีของอุปกรณ์พีแอลดีคือ สามารถโปรแกรมเพื่อปรับเปลี่ยนวงจรในพีแอลดีได้หลายครั้ง ทำให้ผู้ใช้สามารถปรับปรุงความสามารถของอุปกรณ์ให้ดีขึ้นโดยไม่ต้องเปลี่ยนอุปกรณ์ใหม่ ซึ่งพีแอลดีนี้แบ่งได้เป็น 3 ประเภท คือ เอสพีแอลดี (Simple Programmable Logic Device: SPLD) ซีพีแอลดี (Complex Programmable Logic Device: CPLD) และเอฟพีจีเอ (Field Programmable Gate Array: FPGA) ซึ่งอธิบายรายละเอียดดังต่อไปนี้

3.1.2.1 อุปกรณ์เอสพีแอลดี (Simple Programmable Logic Device: SPLD)

เอสพีแอลดีเป็นอุปกรณ์พีแอลดีที่สร้างขึ้นชนิดแรก เพื่อใช้ในการออกแบบวงจรรวมที่มีวงจรลอจิกเกตหลายๆ วงจรเข้าด้วยกัน ซึ่งทำให้ประหยัดเนื้อที่ของแผ่นวงจร (Printed Circuit Board) [10] ซึ่งโครงสร้างเอสพีแอลดีมีความซับซ้อนน้อยกว่าอุปกรณ์พีแอลดีในปัจจุบัน โดยโครงสร้างของอุปกรณ์เอสพีแอลดีนี้ประกอบไปด้วยแอนด์เกตและออร์เกต ซึ่งเรียงในรูปแบบของแถวลำดับ (Array) ซึ่งผู้ใช้สามารถโปรแกรมให้เกตเหล่านี้เชื่อมต่อเข้าด้วยกันให้เป็นวงจรตามความต้องการ โดยการตัดหรือต่อฟิวส์ในตัวอุปกรณ์ ซึ่งเอสพีแอลดีบางชนิดใช้มอสทรานซิสเตอร์แทนฟิวส์ทำให้สามารถลบและโปรแกรมได้หลายครั้ง อุปกรณ์เอสพีแอลดีมีหลายชนิด อาทิเช่น พรอม (Programmable Read Only Memory: PROM) พีแอลเอ (Programmable Logic Array: PLA) พีเอแอล (Programmable Array Logic: PAL) ซึ่งอธิบายดังต่อไปนี้



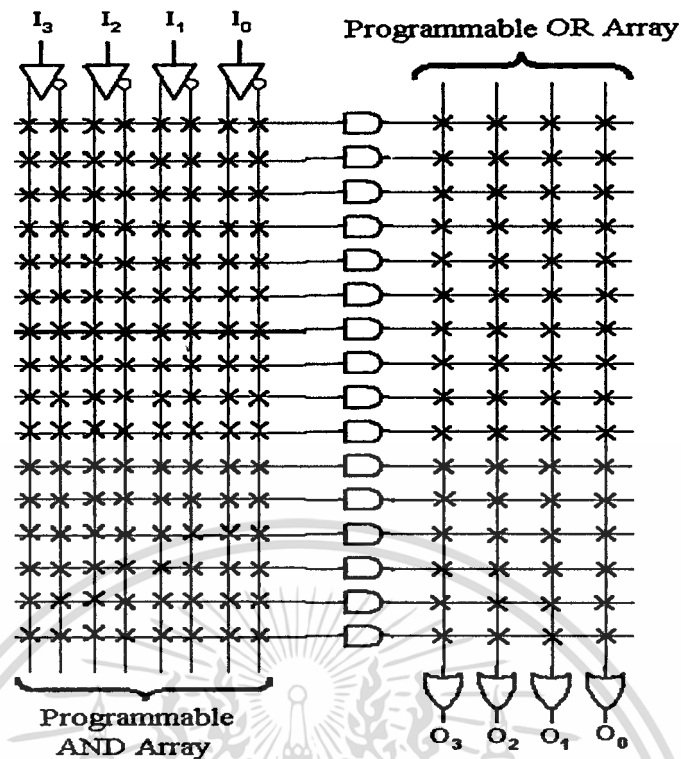
รูปที่ 3.2 โครงสร้างของพรม [11]

- **พรม (Programmable Read-Only Memory: PROM)**

พรมเป็นอุปกรณ์ที่ใช้ทำงานเป็นหน่วยความจำ ซึ่งโครงสร้างประกอบด้วยแถวลำดับของ แอนด์เกตและออร์เกต โดยเอาต์พุตของพรมแสดงเป็นสมการฟังก์ชันของผลคูณร่วมบวก (Sum of Product) ของอินพุต [11] แถวลำดับของแอนด์เกตและออร์เกตถูกเชื่อมต่อในลักษณะของวงจร ทรานซิสเตอร์ แถวลำดับของแอนด์เกตต่อกับอินพุตและแถวลำดับของออร์เกตต่อกับเอาต์พุตโดยแถว ลำดับของออร์เกตสามารถโปรแกรมเชื่อมต่อได้ โครงสร้างของพรมแสดงดังรูปที่ 3.2

- **พีแอลเอ (Programmable Logic Array: PLA)**

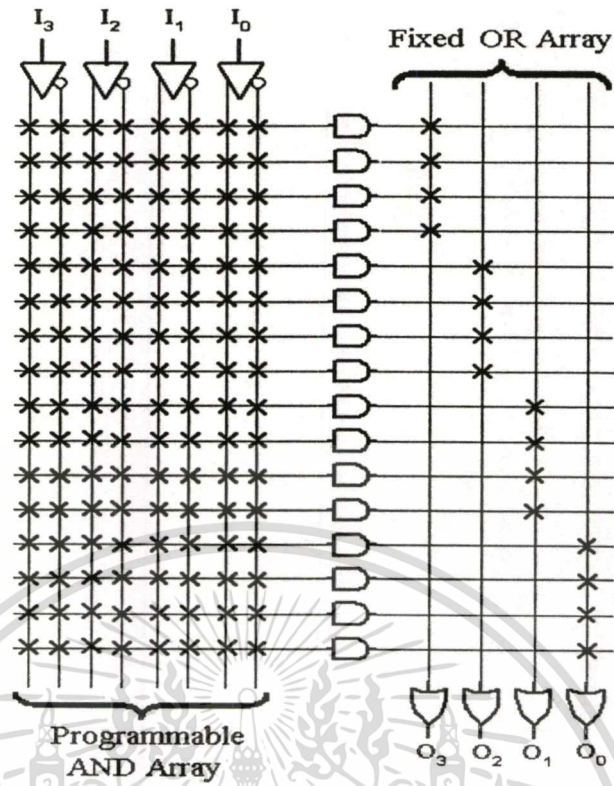
อุปกรณ์พีแอลเอถูกคิดค้นโดยบริษัท Philips ประมาณปี 1970 [12] โดยโครงสร้างของ พีแอลเอประกอบด้วยแถวลำดับของแอนด์เกตและออร์เกต แถวลำดับของแอนด์เกตต่อกับอินพุต และแถวลำดับของออร์เกตต่อกับเอาต์พุต อุปกรณ์พีแอลเอนี้ผู้ใช้สามารถกำหนดการเชื่อมต่อได้ทั้ง แถวลำดับของแอนด์เกตและออร์เกต โครงสร้างของพีแอลเอแสดงดังรูปที่ 3.3



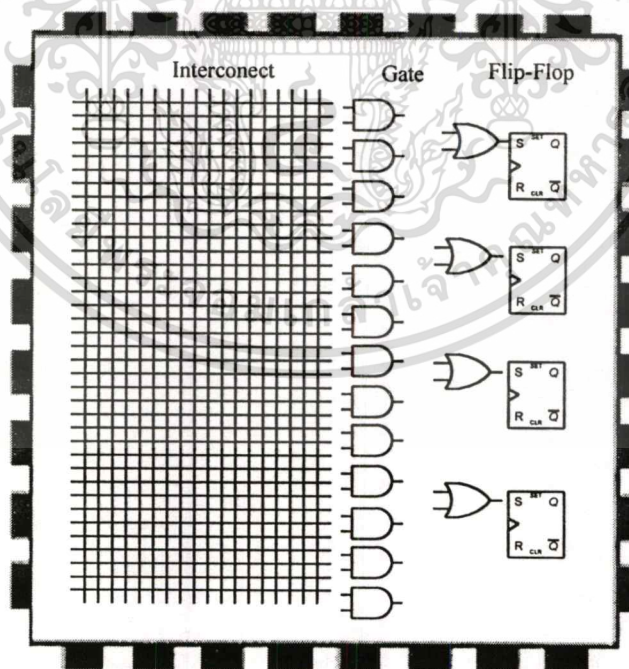
รูปที่ 3.3 โครงสร้างของพีแอลเอ [11]

- **พีแอลเอ (Programmable Array Logic: PAL)**

อุปกรณ์พีแอลเอถูกพัฒนาขึ้นเพื่อลดข้อจำกัดของพีแอลเอจากการหน่วงของเวลาที่เกิดจากการเชื่อมต่อฟิวส์ในแถวลำดับ (Propagation Delay Time) [10] ซึ่งโครงสร้างของพีแอลเอประกอบด้วยแถวลำดับของแอนด์เกตที่สามารถโปรแกรมการเชื่อมต่อได้ และแถวลำดับออร์เกตที่กำหนดการเชื่อมต่อไว้แล้ว โดยแถวลำดับของแอนด์เกตต่อกับอินพุตและแถวลำดับของออร์เกตต่อกับเอาต์พุต และที่เอาต์พุตของออร์เกตอาจมีฟลิปฟล็อปต่อไว้เพื่อทำให้พีแอลเอสามารถโปรแกรมวงจรดิจิทัลแบบซีควนเชียล (Sequential Circuit) ได้ ซึ่งโครงสร้างของพีแอลเอแสดงดังรูปที่ 3.4

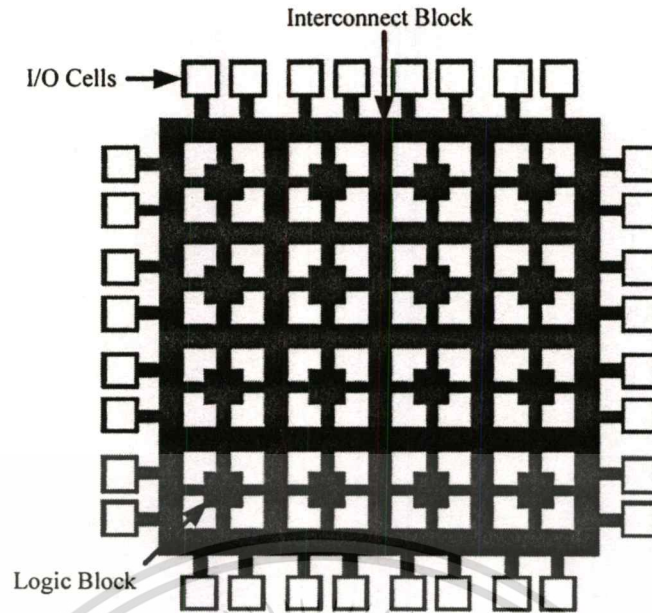


รูปที่ 3.4 โครงสร้างของพีเอแอล [12]



รูปที่ 3.5 โครงสร้างของซีพีแอลดี [12]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.6 โครงสร้างทั่วไปของเอฟพีจีเอ [10]

3.1.2.2 อุปกรณ์ซีพีแอลดี (Complex Programmable Logic Device: CPLD)

ซีพีแอลดีเป็นอุปกรณ์ที่รวมเอชพีแอลดี (S-PLD) หลายตัวหรือเรียกว่า ไมโครเซลล์ (Macro Cells) เข้าเป็นอุปกรณ์เพียงตัวเดียว [12] ซึ่งทำให้ซีพีแอลดีมีความจุของจำนวนเกตที่มากขึ้นกว่าเอชพีแอลดี โดยโครงสร้างของซีพีแอลดีประกอบด้วย เกตต่างๆ ฟลิปฟล็อป และส่วนที่ใช้เชื่อมต่อภายใน (Interconnect) ซึ่งโครงสร้างของซีพีแอลดีแสดงดังรูปที่ 3.5

3.1.2.3 อุปกรณ์เอฟพีจีเอ (Field Programmable Gate Array: FPGA)

เอฟพีจีเอหรือ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์อิเล็กทรอนิกส์ที่สามารถโปรแกรมได้เพื่อให้มีฟังก์ชันการทำงานตามที่ต้องการ ซึ่งถูกพัฒนาขึ้นโดยบริษัท Xilinx ในปี 1985 [12] เอฟพีจีเอนั้นมีปริมาณความหนาแน่นของเกตสูงกว่าอุปกรณ์ซีพีแอลดีมาก การออกแบบวงจรดิจิทัลด้วยเอฟพีจีเอนั้นเป็นที่นิยมมากในปัจจุบัน เพราะการออกแบบด้วยเอฟพีจีเอสะดวกรวดเร็วเนื่องจากการออกแบบและการโปรแกรมเอฟพีจีเอสามารถทำได้โดยใช้คอมพิวเตอร์ส่วนบุคคล การเปลี่ยนแปลงแก้ไขทำได้ง่ายและการออกแบบด้วยเอฟพีจีเอนั้นไม่ต้องเสียค่า NRE ซึ่งมีราคาแพงทำให้ประหยัดค่าใช้จ่ายและบริษัทผู้ผลิตเอฟพีจีเอ เช่น Xilinx หรือ Altera นั้นพยายามนำอุปกรณ์นี้แข่งขันด้านการตลาดกับอุปกรณ์เอสิก (ASIC) จึงส่งผลให้ราคาของอุปกรณ์เอฟพีจีเอมีแนวโน้มที่ถูกลง

โครงสร้างของเอฟพีจีเอนั้นมีโครงสร้างที่ซับซ้อนมาก โดยโครงสร้างภายในของเอฟพีจีเอแบ่งได้เป็น 3 ส่วนคือ บล็อกลอจิก (Logic Block) บล็อกเชื่อมต่อภายใน (Interconnect Block) และเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไอโอเซลล์ (I/O Cell) ซึ่งแสดงดังรูปที่ 3.6 โดยลอจิกบล็อกซึ่งถูกจัดในอยู่ในรูปเมตริก ภายในประกอบไปด้วย วงจรเกตต่างๆ เรจิสเตอร์ หน่วยความจำ เพื่อใช้สร้างฟังก์ชันในการทำงาน โดยมีบล็อกเชื่อมต่อภายในทำหน้าที่เชื่อมต่อระหว่างลอจิกบล็อกและเชื่อมต่อกับไอโอเซลล์ (I/O Cell) ซึ่งทำหน้าที่เป็นอินพุตและเอาต์พุตเพื่อติดต่อกับวงจรภายนอก

3.1.2.4 เทคโนโลยีการโปรแกรมฟิแอลดี

การโปรแกรมอุปกรณ์ฟิแอลดีนั้น ใช้คุณสมบัติสวิตช์ที่สามารถโปรแกรมได้โดยไฟฟ้า (Electrical Programmable Switch) [10] โดยเทคโนโลยีการ โปรแกรมอุปกรณ์ฟิแอลดีมีดังนี้

- เทคโนโลยี Anti-Fuse

การโปรแกรมอุปกรณ์ฟิแอลดีนี้ จะใช้แรงดันไฟฟ้าทำให้เกิดการเปลี่ยนแปลงค่าของความต้านทานจากค่าสูงเป็นค่าต่ำทำให้เกิดการเชื่อมต่อขึ้น ข้อดีของเทคโนโลยีการ โปรแกรมแบบนี้คือ ให้ขนาดวงจรเล็ก แต่ข้อเสียคือโปรแกรมได้เพียงครั้งเดียว โดยอุปกรณ์ที่ใช้เทคโนโลยีนี้ในการโปรแกรมคือ อุปกรณ์ฟรอม ฟิแอล เป็นต้น

- เทคโนโลยี EPROM/ EEPROM

การโปรแกรมอุปกรณ์ฟิแอลดีรูปแบบนี้จะสร้าง EPROM หรือ EEPROM ไว้ในตัวฟิแอลดี ควบคุมผ่าน Pass Transistor หรือ Multiplexer ข้อดีของเทคโนโลยีการ โปรแกรมแบบนี้คือ สามารถโปรแกรมซ้ำได้เหมือนเทคโนโลยี Static RAM ไม่ต้องใช้หน่วยความจำภายนอกในการเก็บข้อมูลในการโปรแกรม และโปรแกรมยังคงอยู่เมื่อไม่มีการจ่ายพลังงานให้อุปกรณ์ แต่มีข้อเสียคือ การโปรแกรมไม่สามารถทำได้รวดเร็ว และขนาดวงจรใหญ่จึงไม่สามารถสร้างอุปกรณ์ชนิดนี้ให้มีความจุเกตสูงได้ โดยอุปกรณ์ที่ใช้เทคโนโลยีนี้ในการ โปรแกรมคือ อุปกรณ์ซีฟิแอลดี เป็นต้น

- เทคโนโลยี Static RAM

การโปรแกรมอุปกรณ์ฟิแอลดีนี้ ใช้ Static RAM ในการกำหนดการเชื่อมต่อของวงจร โดยการกำหนดจะควบคุมผ่าน Pass Transistor หรือ Multiplexer โดย Pass Transistors ทำให้เกิดการเชื่อมต่อระหว่างเทอร์มินอลทั้งสองข้าง และ Multiplexer จะควบคุมการเชื่อมต่อกันระหว่างเอาต์พุตและอินพุตตัวใดตัวหนึ่ง ข้อดีของเทคโนโลยีการ โปรแกรมแบบนี้คือ สามารถโปรแกรมได้อย่างรวดเร็วและโปรแกรมการเชื่อมต่อได้ไม่จำกัดจำนวนครั้ง และให้ขนาดวงจรเล็กจึงสามารถสร้างอุปกรณ์ฟิแอลดีให้มีความจุเกตสูงได้ แต่มีข้อเสียคือ ต้องใช้หน่วยความจำภายนอกในการเก็บข้อมูลที่จะโปรแกรม และต้องโปรแกรมใหม่ทุกครั้งเมื่อเริ่มจ่ายพลังงานให้กับอุปกรณ์ โดยอุปกรณ์ที่ใช้เทคโนโลยีนี้ในการ โปรแกรมคือ อุปกรณ์เอฟฟิแอล เป็นต้น

3.1.3 วงจรรวมแบบเอสิก (Application Specific Integrated Circuit: ASIC)

วงจรรวมประเภทเอสิกหรือ ASIC (Application Specific Integrated Circuit) นั้นนิยมใช้ในเชิงพาณิชย์ เนื่องจากมีต้นทุนที่ถูกกว่าแบบอื่นๆ เมื่อมีการผลิตจำนวนมาก [11] การออกแบบวงจรประเภทเอสิคนั้นใช้โปรแกรมจากบริษัทผู้ผลิต เมื่อทำการจำลองการทำงานเสร็จแล้วก็สามารถส่งไฟล์ที่ได้จากการออกแบบนี้ไปให้บริษัทผู้ผลิตทำการเจียสสารเพื่อสร้างวงจรรวมต่อไป ระยะเวลาตั้งแต่การออกแบบจนได้วงจรรวมนั้นใช้เวลาจนถึงหลายเดือน และต้องเสียค่าใช้จ่ายในการเจียสสารสูง ซึ่งวงจรรวมประเภทเอสิกแบ่งได้เป็น 3 ประเภท ดังนี้

3.1.3.1 วงจรรวมแบบเกตอาร์เรย์ (Gate Array)

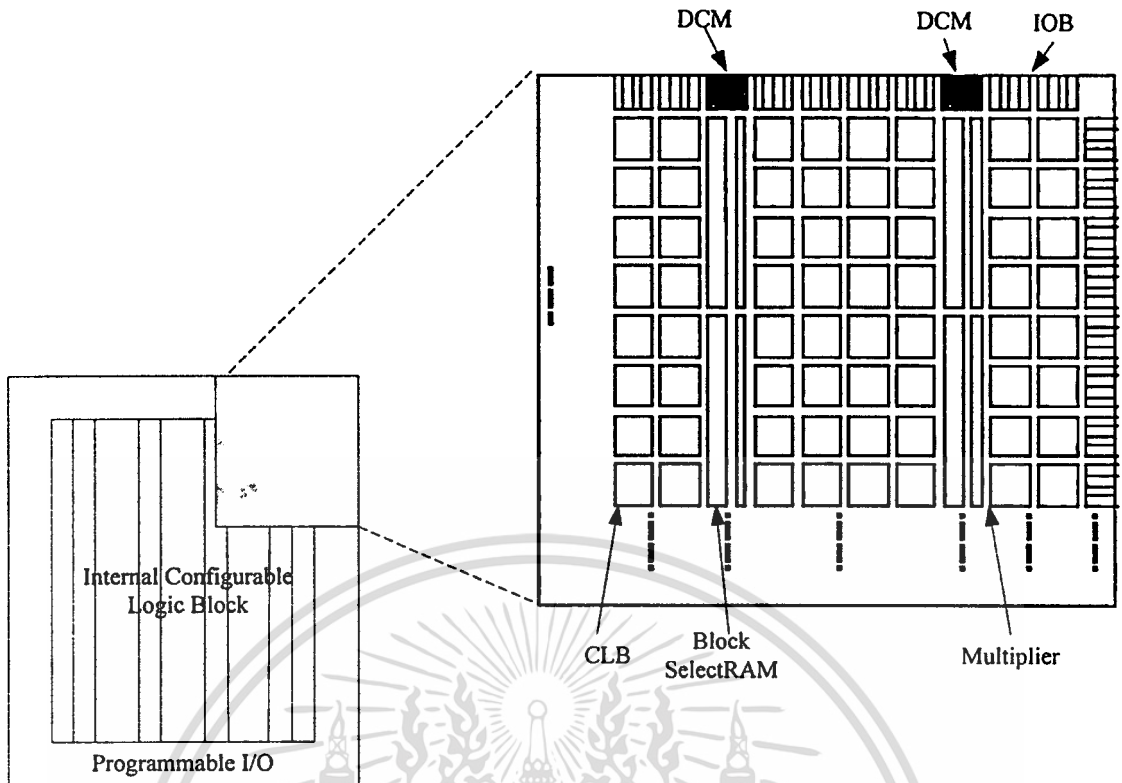
วงจรรวมประเภทเกตอาร์เรย์ (Gate Array) นี้ ประกอบไปด้วยแถวลำดับของวงเกตกับขั้วต่อสายไฟ (Pad) สำหรับเชื่อมต่อกับวงจรภายนอก ซึ่งผู้ใช้ออกแบบการเชื่อมต่อระหว่างเกตและขั้วต่อสายไฟให้ได้วงจรตามความต้องการ แล้วนำไฟล์ที่ได้จากการออกแบบไปทำการเจียสสารที่บริษัทผู้ผลิต โดยทั่วไปแล้ววงจรถูกที่ใช้สร้างวงจรมีพื้นที่ประมาณ 25-30 % ของพื้นที่ซิลิกอน [11] ส่วนพื้นที่ที่เหลือนั้นใช้สำหรับเชื่อมต่อวงจรเกตเข้าด้วยกันกับขั้วต่อสายไฟ

3.1.3.2 วงจรรวมแบบเซลล์มาตรฐาน (Standard Cell)

ปัญหาการใช้พื้นที่ซิลิกอนจากการเชื่อมโยงมากของวงจรรวมแบบเกตอาร์เรย์ ทำให้เกิดข้อจำกัดสำหรับการใช้งานกับวงจรที่มีความซับซ้อน และความต้องการของผู้ใช้จำนวนมากคือ การรวบรวมวงจรรวมมาตรฐาน เช่น วงจรถะกูล 7400 วงจรถะกูล 4000 จำนวนหลายวงจรเข้าด้วยกัน ด้วยเหตุนี้ทำให้เกิดวงจรรวมเอสิกประเภทเซลล์มาตรฐาน (Standard Cell) ขึ้นมา ซึ่งวงจรรวมประเภทนี้ผู้ใช้จะเป็นผู้เลือกกลุ่มวงจรที่ทำหน้าที่ต่างๆ เช่น เกต ฟลิปฟลอป ตัวนับ หน่วยความจำ หรือไมโครโปรเซสเซอร์ จากแฟ้มข้อมูล (Library) ของบริษัทผู้ผลิตมาออกแบบวงจรรวม การผลิตวงจรรวมเซลล์มาตรฐานนั้นมีต้นทุนสูงกว่าวงจรรวมแบบเกตอาร์เรย์ และใช้ระยะเวลาการเจียสสารนานกว่าสองถึงสามเท่า วงจรรวมประเภทนี้จึงเหมาะกับการใช้งานเชิงพาณิชย์ที่มีการผลิตสูงนับหมื่นตัวขึ้นไป

3.1.3.3 วงจรรวมแบบฟูลคัสตัม (Full Custom)

วงจรรวมประเภทฟูลคัสตัม (Full Custom) นี้ผู้ใช้เป็นผู้ออกแบบทั้งหมด แล้วจึงส่งไฟล์ที่ได้จากการออกแบบไปให้บริษัทผู้ผลิตเจียสสารในรูปของไฟล์ข้อมูลมาตรฐาน เช่น GDSII, CIF [11] การเจียสสารเป็นลักษณะที่รวบรวมการออกแบบหลายวงจรรวมลงบนแผ่นซิลิกอนแผ่นเดียวกัน เพื่อประหยัดค่าใช้จ่ายที่ใช้เจียสสาร



รูปที่ 3.7 โครงสร้างของเอฟพีจีเอตระกูล Vertex-II [13]

3.2 โครงสร้างของเอฟพีจีเอ Xilinx Vertex-II

วิทยานิพนธ์นี้ได้ใช้เอฟพีจีเอตระกูล Vertex-II รุ่น xc2v1000-6 ของบริษัท Xilinx เป็นต้นแบบสำหรับติดตั้งชุดถอดรหัส Viterbi ซึ่งเอฟพีจีเอตระกูลนี้ถูกออกแบบให้มีประสิทธิภาพและความจุของเกตสูง โดยมีความจุตั้งแต่ 40K เกตจนถึง 8M เกต ในกระบวนการผลิตเอฟพีจีเอตระกูล Vertex-II นี้ใช้เทคโนโลยีแบบ CMOS 0.15 μm / 0.12 μm 8-Layer Metal ทำให้เอฟพีจีเอตระกูลนี้มีความเร็วในการทำงานสูงและมีการใช้พลังงานน้อย [13] รายละเอียดต่างๆ ของเอฟพีจีเอตระกูล Vertex-II มีดังนี้

3.2.1 โครงสร้างภายในเอฟพีจีเอ Xilinx Vertex-II

โครงสร้างของเอฟพีจีเอตระกูล Vertex-II ประกอบด้วยสองส่วนใหญ่ๆ คือ บล็อกอินพุต-เอาต์พุต (Programmable Input/Output: I/OB) และ Internal Configurable Logic Block (ICLB) ซึ่งแสดงดังรูปที่ 3.7 [13] โดยบล็อกอินพุตเอาต์พุตสามารถโปรแกรมให้เป็นอินพุตหรือเอาต์พุตเพื่อติดต่อกับอุปกรณ์ภายนอกได้ ส่วน ICLB นั้นประกอบด้วย 4 อุปกรณ์หลัก คือ

- บล็อก Configurable Logic Block (CLB) ประกอบด้วยวงจรลอจิกแบบคอมบินเนชัน (Combinatorial) และแบบซิงโครนัส (Synchronous) เรจิสเตอร์และบัฟเฟอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- บล็อก SelectRAM เป็นหน่วยความจำแบบสองพอร์ต (Dual-port RAM) ที่สามารถเลือกขนาดได้ โดยมีขนาดสูงสุด 18 กิโลบิต
- บล็อก Multiplier เป็นบล็อกวงจรคูณขนาด 18 x 18 บิต
- บล็อก DCM (Digital Clock Manager) เป็นบล็อกที่จัดการสัญญาณนาฬิกา ซึ่งบล็อกนี้สามารถคูณหรือหารความถี่สัญญาณนาฬิกา และนอกจากนั้นยังสามารถเลื่อนเฟส (Phase Shift) ของสัญญาณนาฬิกาได้

อุปกรณ์ใน ICLB นี้อยู่ในรูปแบบของแถวลำดับ (Array) อย่างสม่ำเสมอและอุปกรณ์เหล่านี้สามารถโปรแกรมการเชื่อมต่อกันอย่างรวดเร็วโดยส่วนของบล็อกเชื่อมต่อ (Interconnects) ซึ่งในเอฟพีจีเอตระกูล Vertex-II เรียกเทคโนโลยีการเชื่อมต่ออย่างรวดเร็วนี้ว่า “Active Interconnect Technology”

3.2.2 รายละเอียดของอุปกรณ์ภายในเอฟพีจีเอ Xilinx Vertex-II

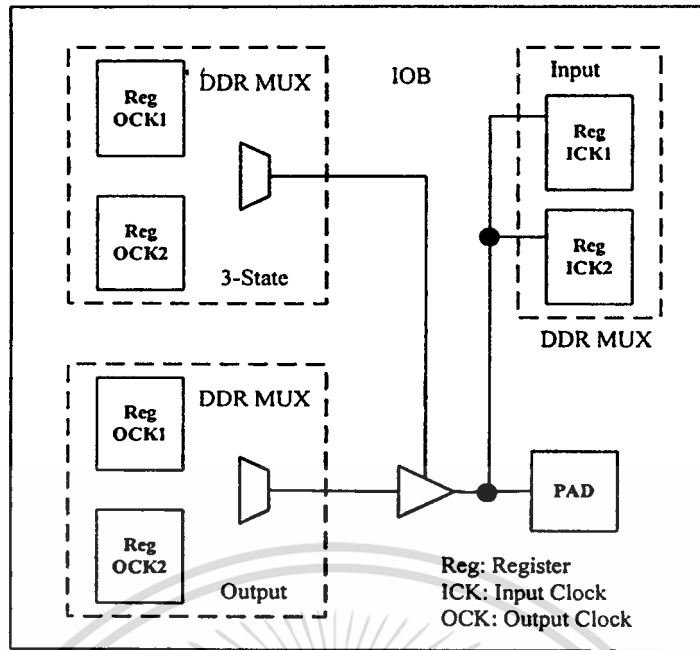
ในส่วนนี้กล่าวถึงรายละเอียดของอุปกรณ์ภายในของเอฟพีจีเอตระกูล Xilinx Vertex-II ซึ่งอุปกรณ์ภายในเอฟพีจีเอที่สำคัญที่ใช้ในการออกแบบในงานวิจัยมีดังนี้

3.2.2.1 บล็อกอินพุตเอาต์พุต (Input/Output Block: IOB)

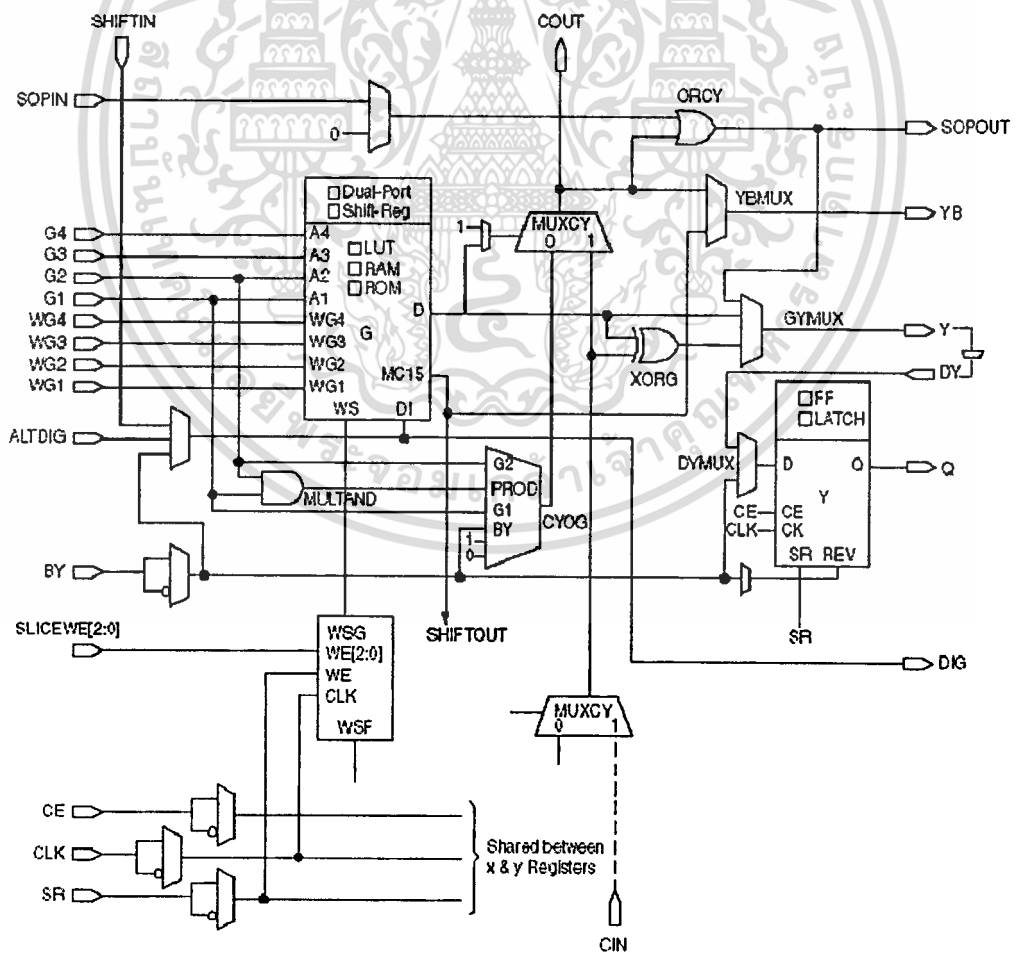
บล็อกอินพุตเอาต์พุต (IOB) เป็นส่วนที่ใช้ติดต่อกับวงจรภายนอก ซึ่งบล็อกอินพุตและเอาต์พุตสามารถโปรแกรมให้เป็นอินพุตหรือเอาต์พุตแบบ Single Data Rate หรือแบบ Double Data Rate (DDR) ได้ ซึ่งบล็อกอินพุตเอาต์พุตนี้ประกอบด้วย 6 เรจิสเตอร์ (Register) และ 3 DDR Multiplexers ซึ่งแสดงดังรูปที่ 3.9 โดยเรจิสเตอร์นี้สามารถโปรแกรมเป็นดีฟลิปฟล็อป (D Flip-Flop) หรือแลตช์ (Latch) ได้ บล็อกอินพุตเอาต์พุตนี้ต่ออยู่กับส่วน Switch Matrix เพื่อเชื่อมต่อกับอุปกรณ์ภายในอื่นๆ ส่วน DDR Multiplexer ใช้ในการส่งข้อมูลแบบ DDR

3.2.2.2 บล็อก Configurable Logic Block (CLB)

บล็อก Configurable Logic Block (CLB) ประกอบด้วย วงจรลอจิกแบบคอมไบเนชัน และแบบซิงโครนัส ซึ่ง CLB ต่ออยู่กับส่วน Switch Matrix เพื่อเชื่อมต่อกับอุปกรณ์ภายในอื่นๆ โดยในแต่ละ CLB นั้นประกอบด้วย Slices จำนวน 4 Slices ซึ่งแต่ละ Slices มีลักษณะคล้ายกัน ซึ่ง Slice ประกอบด้วยอุปกรณ์ดังนี้ คือ ตัวสร้างฟังก์ชันสี่อินพุต (4-input Function Generators) จำนวน 2 ตัว วงจรลอจิกตัวทด (Carry Logic) วงจรบวก (Arithmetic Logic Gates) วงจรมัลติเพล็กซ์ขนาดใหญ่ (Wide Function Multiplexers) และหน่วยความจำ (Storage Elements) จำนวน 2 ตัว



รูปที่ 3.8 โครงสร้างของบล็อกอินพุตเอาต์พุต [13]



รูปที่ 3.9 โครงสร้างของ Slice [13]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.1 ขนาดของหน่วยความจำที่สามารถโปรแกรมใช้งานได้ของบล็อกหน่วยความจำ [13]

16K x 1 บิต	2K x 9 บิต
8K x 2 บิต	1K x 18 บิต
4K x 4 บิต	512 x 36 บิต

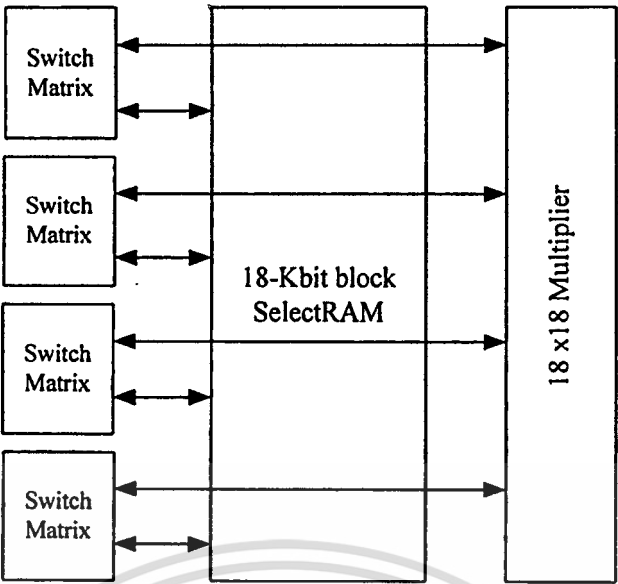
โดยตัวสร้างฟังก์ชันสี่อินพุตสามารถโปรแกรมให้เป็นตัวเปิดตารางสี่อินพุต (4-input Look-up Table) หรือโปรแกรมให้เป็นหน่วยความจำขนาด 16 บิต (16-bits of Distributed SelectRAM Memory) หรือโปรแกรมให้ชิพเรจิสเตอร์ 16 บิตที่ปรับเปลี่ยนได้ (16-bit Variable Tap Shift Register) โดยเอาต์พุตของตัวสร้างฟังก์ชันนี้ต่อกับหน่วยความจำ (Storage Elements) ซึ่งเชื่อมต่อกับอุปกรณ์อื่นๆ โดยหน่วยความจำนี้สามารถโปรแกรมเป็นดีฟลิปฟล็อป (D Flip-Flop) หรือแลตช์ (Latch) ก็ได้ ซึ่งรูปที่ 3.9 นั้นแสดงโครงสร้างของ Slice

3.2.2.3 บล็อกหน่วยความจำ 18 กิโลบิต (18 Kbit Block SelectRAM Memory)

บล็อกหน่วยความจำ (18 Kbit Block SelectRAM Memory) เป็นหน่วยความจำที่สามารถโปรแกรมให้เป็นหน่วยความจำแบบพอร์ตเดียว (Single-port RAM) หรือหน่วยความจำแบบสองพอร์ต (Dual-port RAM) ได้ สามารถเลือกขนาดจาก 16 K x 1 บิตจนถึง 512 x 32 บิต และสามารถเลือกให้ทำงานเป็นแบบซิงโครนัสหรืออะซิงโครนัสได้ ซึ่งตารางที่ 3.1 แสดงขนาดหน่วยความจำที่สามารถเลือกโปรแกรมให้ใช้งานได้ของบล็อกหน่วยความจำนี้

3.2.2.4 บล็อกวงจรคูณขนาด 18 x 18 บิต (18-Bit x 18-Bit Multiplier)

บล็อกวงจรคูณขนาด 18 x 18 บิตนี้เป็นวงจรคูณที่สามารถปรับเปลี่ยนขนาดได้ โดยมีขนาดสูงสุดคือ 18 x 18 บิต และสามารถทำการคูณแบบคิเคเรื่องหมาย (2's Complement Signed Multiplier) ได้ บล็อกวงจรคูณขนาด 18 x 18 บิตนี้สามารถทำงานอย่างอิสระหรือทำงานร่วมกับบล็อกหน่วยความจำ 18 กิโลบิตในการทำงานที่เรียกว่า "Multiply-Accumulator (MAC)" ซึ่งเป็นฟังก์ชันการทำงานของการประมวลผลสัญญาณดิจิทัล (Digital Signal Processing) โดยรูปที่ 3.10 แสดงการเชื่อมต่อของบล็อกหน่วยความจำ 18 กิโลบิตและบล็อกวงจรคูณขนาด 18 x 18 บิต ซึ่งบล็อกทั้งสองจะถูกเชื่อมต่อกับ Switch Matrix



รูปที่ 3.10 การเชื่อมต่อบล็อกหน่วยความจำ 18 กิโลบิตและบล็อกวงจรคูณขนาด 18 x 18 บิต [13]

3.2.2.5 บล็อก Digital Clock Manager (DCM)

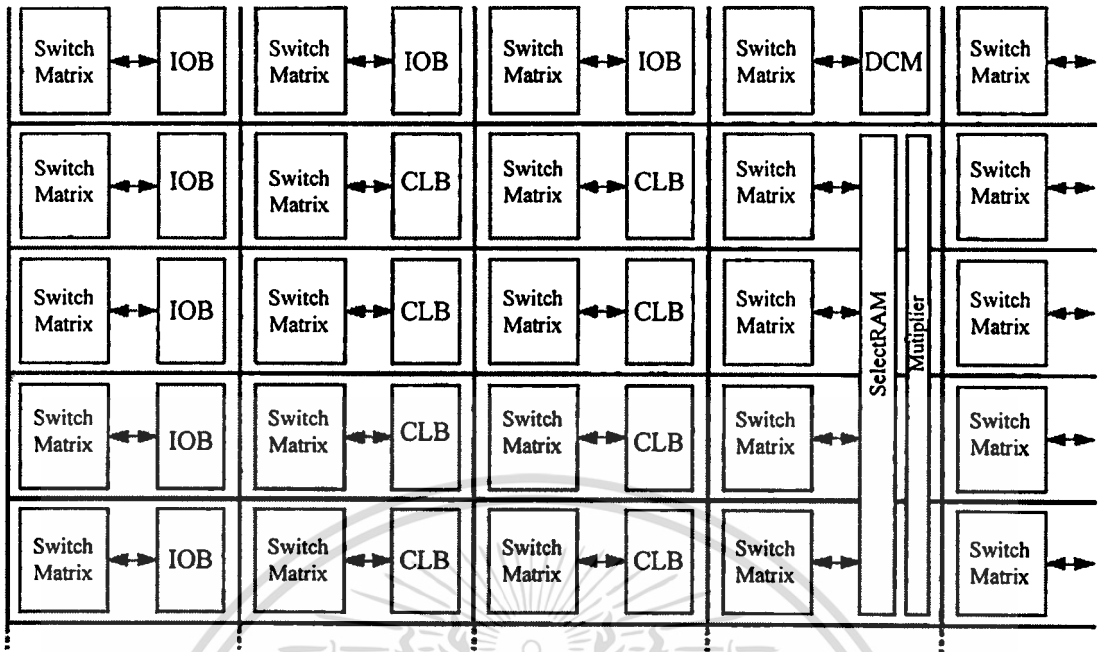
บล็อก DCM (Digital Clock Manager) เป็นบล็อกที่จัดการสัญญาณนาฬิกาของเอฟพีจีเอ โดยการทำงานใช้หลักการ Digital Delay Line และการป้อนกลับ ซึ่งมีลักษณะการทำงาน 3 แบบ ดังนี้

- Clock De-skew

การทำงานนี้ DCM จะกำเนิดสัญญาณนาฬิกาของระบบ ซึ่งมาจากภายในและภายนอก เอฟพีจีเอใหม่ ซึ่งมีเฟสตรงกับอินพุตของสัญญาณนาฬิกา ดังนั้นจึงสามารถกำจัดปัญหาการหน่วงเวลาของสัญญาณนาฬิกา สัญญาณนาฬิกาที่มาถึงขาของเอฟพีจีเอ (Pad) สามารถจับอุปกรณ์ภายในได้โดยทันที

- Frequency Synthesis

การทำงานนี้ DCM สามารถกำเนิดความถี่สัญญาณนาฬิกาใหม่ โดยใช้การคูณหรือการหารสัญญาณนาฬิกาอินพุต ในการหารความถี่นั้นสำหรับเอฟพีจีเอตระกูลนี้มีตัวหารเท่ากับ 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13, 14, 15 และ 16 ส่วนการคูณนั้นมีตัวคูณคือ M/D โดย M และ N เป็นจำนวนเต็ม แต่ค่าความถี่ที่ได้จากการคูณต้องไม่เกิน DCM Timing Parameter ของเอฟพีจีเอแต่ละรุ่น (สูงสุดไม่เกิน 350 MHz ในเอฟพีจีเอตระกูล Vertex II) [13]



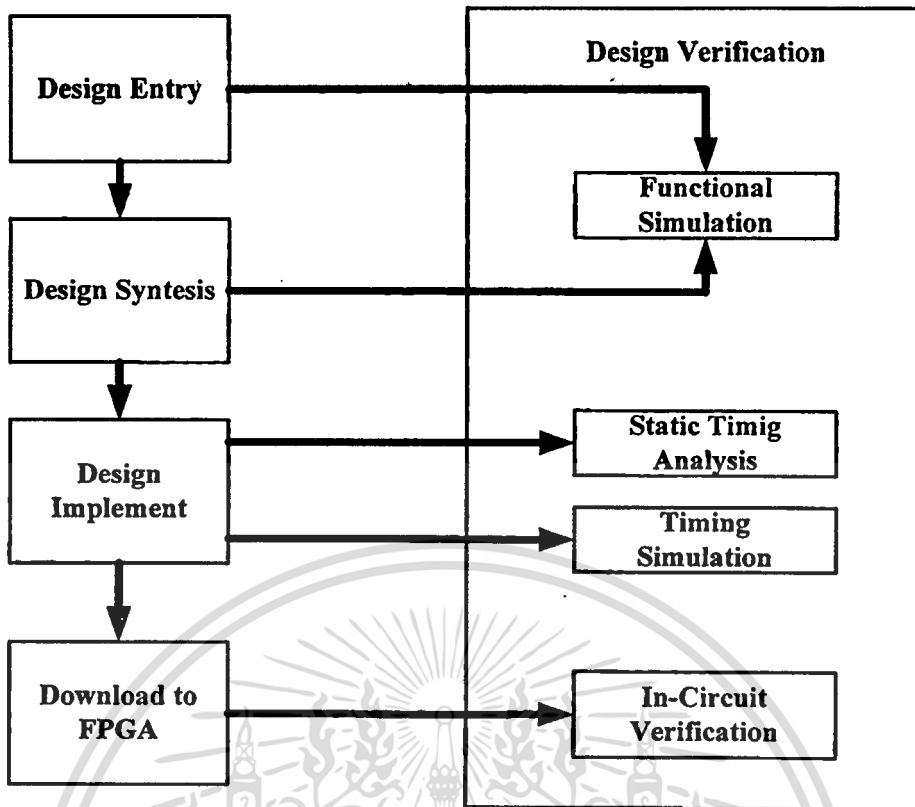
รูปที่ 3.11 โครงสร้างของบล็อกเชื่อมต่อและหาเส้นทาง [13]

- **Phase shifting**

DCM สามารถเลื่อนเฟสของสัญญาณนาฬิกาได้ ซึ่งการเลื่อนเฟสของสัญญาณนาฬิกาสามารถทำได้ 90, 180 และ 270 องศา

3.2.2.6 บล็อกเชื่อมต่อและหาเส้นทาง (Routing Resources)

การเชื่อมต่อและหาเส้นทางของอุปกรณ์ภายในเอฟพีจีเอตระกูลนี้ใช้อุปกรณ์ Switch Matrix ในการเชื่อมต่ออุปกรณ์ภายใน เช่น CLB, IOB, Block RAM, Multiplier และ DCM เข้าด้วยกัน ซึ่งการเชื่อมต่อและหาเส้นทางใช้เทคโนโลยีที่เรียกว่า “Active Interconnect Technology” ซึ่งทำให้การเข้าถึงอุปกรณ์ต่างๆ ทำได้อย่างรวดเร็ว ซึ่งรูปที่ 3.11 แสดงโครงสร้างการเชื่อมต่อและหาเส้นทางของเอฟพีจีเอตระกูล Vertex-II



รูปที่ 3.12 ขั้นตอนการออกแบบและติดตั้งวงจรดิจิทัลบน FPGA [15]

3.3 การออกแบบและติดตั้งวงจรดิจิทัลบนเอฟพีจีเอ

การออกแบบวงจรดิจิทัลเพื่อติดตั้งกับเอฟพีจีเอนั้นใช้ภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language: HDL) เช่น ภาษาวีเอชดีแอล (VHDL) หรือภาษาเวริล็อก (Verilog HDL) ในการออกแบบวงจร จากนั้นนำโค้ดที่ได้ไปสังเคราะห์ (Synthesis) ให้เป็นวงจร ผลการสังเคราะห์ที่ได้คือ เน็ตลิสต์ในระดับเกต (Gate-level Netlist) ซึ่งมีรูปแบบมาตรฐาน คือ EDIF (Electronic Data Interchange Format) หรือ XNF (Xilinx Netlist Format) ซึ่งใช้สำหรับเอฟพีจีเอของบริษัท Xilinx [14] ขั้นตอนการออกแบบและติดตั้งวงจรดิจิทัลบนเอฟพีจีเอ [15] แสดงดังรูปที่ 3.12 ส่วนรายละเอียดขั้นตอนการออกแบบมีดังนี้

- การออกแบบ (Design Entry)

การออกแบบวงจรเริ่มต้นจาก การเขียนภาษาอธิบายฮาร์ดแวร์หรือเอชดีแอล (HDL) ในการอธิบายพฤติกรรม (Behavioral) การทำงานของวงจรหลังจากเข้าใจอัลกอริทึมการทำงานหรือมีการออกแบบวงจรไว้แล้ว ในการเขียนภาษาเอชดีแอลสามารถใช้โปรแกรม Text Editor หรือใช้ซอฟต์แวร์ที่สนับสนุนการเขียนภาษาเอชดีแอล ซึ่งทำให้ง่ายในการเขียนและสามารถตรวจสอบความถูกต้องตามหลักไวยากรณ์ได้ เช่น ModelSim ของบริษัท Model Technology เป็นต้น เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากการออกแบบวงจรด้วยเขียนภาษาเอชดีแอล ขั้นตอนที่สำคัญคือ การตรวจสอบว่าวงจรที่ออกแบบภาษาเอชดีแอลนั้นสามารถทำงานได้ตามที่ออกแบบ ซึ่งสามารถตรวจสอบได้โดยการจำลองการทำงานระดับพฤติกรรม (Behavioral Simulation) ของวงจร โดยการเขียนภาษาเอชดีแอลเพื่อทดสอบวงจร ที่เรียกว่า “Testbench” ซึ่งเป็นการกำหนดรูปแบบของสัญญาณอินพุต และเมื่อจำลองการทำงานด้วยโปรแกรมซิมูเลเตอร์ (Simulator) เช่น ModelSim ก็จะได้สัญญาณเอาต์พุตทำให้สามารถตรวจสอบว่าการทำงานของวงจรที่ถูกตรวจสอบ (Design Under Test: DUT) ทำงานถูกต้องหรือไม่ ซึ่งถ้าไม่ถูกต้องก็สามารถกลับไปแก้ไขใหม่ได้

- การสังเคราะห์ (Design Synthesis)

เมื่อวงจรที่ออกแบบถูกตรวจสอบแล้วว่าการทำงานถูกต้อง จากนั้นโค้ดภาษาเอชดีแอล ก็จะถูกทำการสังเคราะห์เป็นวงจรเพื่อให้ได้เนตลิสต์ในระดับเกตออกมา แล้วตรวจสอบความถูกต้องอีกครั้ง โดยการจำลองการทำงานระดับฟังก์ชัน (Functional Simulation) ด้วยโปรแกรมซิมูเลเตอร์ โดยใช้ Testbench เดิม ซึ่งถ้าการทำงานไม่ถูกต้องก็สามารถกลับไปแก้ไขโค้ดภาษาเอชดีแอลใหม่

- การติดตั้ง (Design Implement)

ในขั้นตอนการติดตั้งนี้ ผลที่ได้จากการสังเคราะห์ที่เป็นเนตลิสต์ในระดับเกตนั้นจะถูกแปลงให้เป็นกลุ่มของลอจิก จากนั้นจะทำการวางและเชื่อมเส้นทาง (Place & Route) ในเอฟพีจีเอ โดยซอฟต์แวร์ของผู้ผลิตเอฟพีจีเอ เช่น Xilinx ISE เป็นต้น ซึ่งโปรแกรมจะรายงานเวลาการทำงานของวงจร (Static Timing) และสามารถตรวจสอบความถูกต้องโดยทำการจำลองการทำงานระดับฐานเวลา (Timing Simulation) ด้วยโปรแกรมซิมูเลเตอร์โดยใช้ Testbench เดิม การจำลองการทำงานระดับฐานเวลาถือว่าการจำลองการทำงานขั้นตอนสุดท้ายซึ่งต้องทำหลังจากการวางและเชื่อมต่อเส้นทาง การจำลองการทำงานจะให้รายละเอียดที่แน่นอนเกี่ยวกับเรื่องของเวลา เช่น ความล่าช้าของสัญญาณไฟฟ้าตามเส้นทางต่างๆ ภายในวงจร ระยะเวลาของสัญญาณนาฬิกา (Clock Period) น้อยที่สุดที่สามารถใช้กับวงจรได้ เป็นต้น ซึ่งถ้าการทำงานไม่ถูกต้องก็สามารถกลับไปแก้ไขโค้ดภาษาเอชดีแอลใหม่

- การโปรแกรมเอฟพีจีเอ (Download to FPGA)

เมื่อตรวจสอบด้วยการจำลองการทำงานระดับฐานเวลาแล้ว สามารถแปลงเนตลิสต์ในระดับเกตเป็นข้อมูลวงจร (Configuration Data) ซึ่งอยู่ในรูปของบิตสตรีม (Bit-stream) เพื่อนำบิตสตรีมนี้นำไปโปรแกรมลงเอฟพีจีเอต่อไป ขั้นตอนนี้ใช้ซอฟต์แวร์ของผู้ผลิตเอฟพีจีเอ ซึ่งการตรวจสอบความถูกต้องในการทำงานลำดับสุดท้ายคือ การตรวจสอบในวงจรใช้งานจริง (In-Circuit Verification)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4 ภาษาเวริล็อก (Verilog HDL)

ภาษาเวริล็อก (Verilog HDL) เป็นภาษาอธิบายฮาร์ดแวร์ (Hardware Description Language: HDL) ที่ใช้ในการออกแบบวงจรระบบดิจิทัล ภาษาเวริล็อกนั้นเป็นหนึ่งในสองภาษาอธิบายฮาร์ดแวร์ที่นิยมใช้กันมากในการออกแบบวงจรระบบดิจิทัลในปัจจุบัน ซึ่งได้แก่ ภาษาวีเอชดีแอล (Very high speed integrated circuit Hardware Description Language: VHDL) และ ภาษาเวริล็อก เนื่องจากภาษาเวริล็อกมีลักษณะคล้ายภาษาซี จึงทำให้วิศวกรผู้ออกแบบซึ่งส่วนใหญ่ นั้นมีความรู้ด้านภาษาซี สามารถเรียนรู้และเข้าใจได้ง่าย [16] และโครงสร้างของภาษาเวริล็อกยังมีความซับซ้อนน้อยกว่าภาษาวีเอชดีแอล

3.4.1 ประวัติความเป็นมา

ภาษาเวริล็อกถูกคิดค้นโดย บริษัท Gateway Design Automation ในปี 1981 ซึ่งเหตุการณ์ที่สำคัญและความเป็นมาหลายเรื่องตามลำดับได้ดังนี้ [17]

ปี 1983 บริษัท Gateway Design Automation ได้เปิดตัวภาษาอธิบายฮาร์ดแวร์ ซึ่งเรียกว่า “Verilog HDL” หรือเรียกง่ายๆ ว่า “Verilog” พร้อมทั้งส่วนการจำลองการทำงานของภาษาเวริล็อก (Verilog Simulator)

ปี 1985 ภาษาเวริล็อกได้มีการปรับปรุงเพิ่มเติมและได้ออกเวอร์ชันใหม่ของการจำลองการทำงาน ซึ่งเรียกว่า “Verilog XL”

ปี 1987 ภาษาเวริล็อกเป็นที่นิยมใช้ ขนาดของวงจรที่ออกแบบนั้นเริ่มเกินความสามารถของชุดจำลองการทำงานของ Gateway Design System และบริษัทอื่นๆ

ปี 1989 บริษัท Cadence ซึ่งเป็นบริษัทผู้ผลิตซอฟต์แวร์ออกแบบวงจรรวมชั้นนำในปัจจุบัน ได้พัฒนาภาษาเวริล็อกต่อจากบริษัท Gateway Design System

ปี 1990 บริษัท Cadence ได้แยกภาษาเวริล็อกและส่วนการจำลองการทำงานออกจากกัน โดยได้นำภาษาเวริล็อกเปิดเผยต่อสาธารณะ และในปีนี้ผู้ใช้ภาษาเวริล็อกและบริษัทต่างๆ ได้จัดตั้งสมาคม Open Verilog International (OVI) เป็นผู้กำหนดควบคุมรายละเอียด (Specification) ของภาษาเวริล็อก และได้จัดทำคู่มืออ้างอิงสำหรับภาษาเวริล็อกเวอร์ชัน 1.0 (Verilog 1.0 Reference Manual) ขึ้นมา

ปี 1993 สมาคม OVI ได้ปรับปรุงเพิ่มเติมความสามารถของภาษาเวริล็อก เช่น สามารถของอาร์เรย์ และได้จัดทำคู่มืออ้างอิงสำหรับภาษาเวริล็อกใหม่ เป็นเวอร์ชัน 2.0 (Verilog 2.0 Reference Manual)

ปี 1995 สมาคม OVI ได้ส่งภาษาเวริล็อกเวอร์ชัน 2.0 ให้สมาคม IEEE ซึ่งสมาคม IEEE ได้ปรับปรุงและได้กำหนดเป็นมาตรฐานใหม่ที่เรียกว่า มาตรฐาน IEEE 1394-1995

ปี 2001 มีปรับปรุงให้เพิ่มความสามารถของภาษาเวรลิคมาตรฐาน IEEE 1394 ทั้งด้านการสังเคราะห์วงจรและการจำลองการทำงาน อีกทั้งยังปรับปรุงให้ง่ายต่อการใช้งาน ซึ่งสมาคม IEEE กำหนดเป็นมาตรฐานใหม่ที่เรียกว่า มาตรฐาน IEEE 1394-2001

3.4.2 โครงสร้างของภาษาเวรลิค

ภาษาเวรลิคนั้นสามารถอธิบายพฤติกรรมของวงจรดิจิทัลได้หลายระดับ [16] ได้แก่ การอธิบายวงจรทรานซิสเตอร์และการเชื่อมต่อ (Wire) ซึ่งเรียกว่า ระดับการสวิตช์ (Switch Level) การอธิบายการเชื่อมต่อของลอจิกเกตและฟลิปฟล็อปในวงจร ซึ่งเรียกว่า ระดับเกต (Gate Level) และการอธิบายการโอนถ่ายสัญญาณระหว่างเรจิสเตอร์ เรียกว่า ระดับ RTL (Register Transfer Level) ซึ่งการออกแบบวงจรดิจิทัลด้วยภาษาเวรลิคส่วนใหญ่จะใช้อธิบายพฤติกรรมของวงจรดิจิทัลใน ระดับ RTL นี้

ภาษาเวรลิคนั้นมีไวยากรณ์คล้ายภาษาซี เช่น จบประโยคด้วย ; หรือใช้ // เพื่อไม่ให้นำไปใช้ในการคอมไพล์ เป็นต้น โครงสร้างของภาษาเวรลิคนั้นประกอบด้วยมอดูล (Module) ซึ่งแต่ละมอดูลนั้นสามารถรวมเข้าด้วยกันเป็นมอดูลขนาดใหญ่หรือเรียกว่า ท็อปมอดูล (Top Module) โดยแต่ละมอดูลย่อยนั้นสามารถทำงานพร้อมกัน (Concurrent) ซึ่งโครงสร้างของภาษาเวรลิคประกอบด้วยส่วนต่างๆ ดังแสดงในตัวอย่างการอธิบายพฤติกรรมของวงจรดีฟลิปฟล็อป (D Flip-Flop) ได้ดังนี้

```

module dff(clk,rst,din,dout);    // <ชื่อมอดูล> (<ชื่อพอร์ต>)
input din, clk, rst;           //<ชื่อสัญญาณอินพุต>
output dout;                   //<ชื่อสัญญาณเอาต์พุต>
reg dout;                      //<ชื่อสัญญาณ>
always @(posedge clk or posedge rst) //<รายละเอียดโปรแกรม>
    if (rst)
        dout <= 1'b0;
    else
        dout <= din;
endmodule                      //<จบมอดูล>

```

โครงสร้างของชุดถอดรหัสและวิธีการออกแบบ

บทนี้กล่าวถึงหลักการทำงานและการออกแบบชุดถอดรหัส Viterbi ซึ่งเป็นชุดถอดรหัสที่ให้ประสิทธิภาพสูงในการถอดรหัสคอนวูลูชัน ในส่วนแรกของบทนี้กล่าวถึงโครงสร้างโดยทั่วไปและหลักการทำงานของชุดถอดรหัส Viterbi แบบต่างๆ และส่วนหลังนั้นกล่าวถึงการออกแบบชุดถอดรหัส Viterbi ที่นำเสนอสำหรับระบบ 3GPP ที่มีโครงสร้างขนาดเล็กที่ใช้โครงสร้างอนุกรมแบบ 4 Add-Compare-Select (ACS) ซึ่งเป็นจำนวนของ ACS ที่ใช้อุปกรณ์ในการสร้างจำนวนน้อยและให้ความเร็วในการถอดรหัสสูง [18] นอกจากนั้นส่วน Survivor Memory Unit (SMU) ได้ปรับปรุงวิธีการทำงานแบบ Register Exchange ที่ให้ความเร็วในการทำงานสูงแต่มีโครงสร้างขนาดใหญ่ [19] ให้ขนาดเล็กลงและเหมาะสมกับโครงสร้างแบบ 4 ACS

4.1 โครงสร้างและการทำงานของชุดถอดรหัส Viterbi

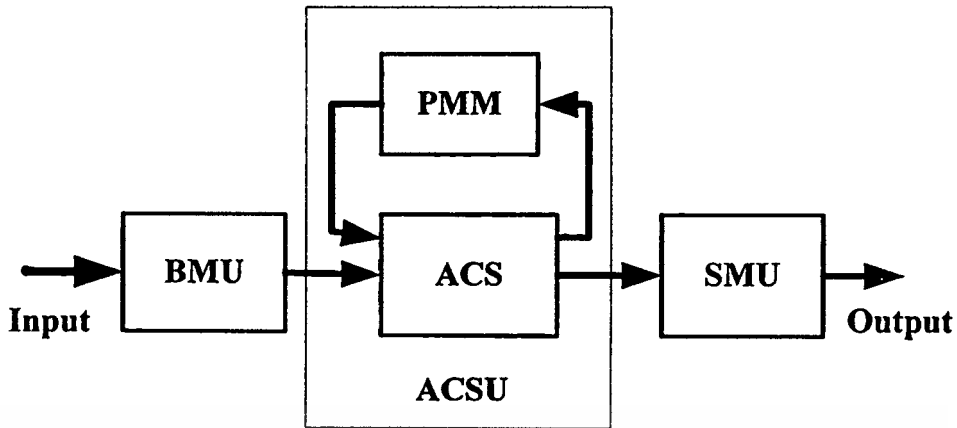
โครงสร้างโดยทั่วไปของชุดถอดรหัส Viterbi นั้น สามารถแบ่งได้เป็น 3 ส่วน ตามลักษณะการทำงานในการถอดรหัสของ Viterbi Algorithm ได้แก่ ภาค Branch Metric Unit (BMU) ภาค Add-Compare-Select Unit (ACSU) และภาค Survivor Memory Unit (SMU) ซึ่งบล็อกไดอะแกรมแสดงการทำงานของชุดถอดรหัส Viterbi แสดงดังในรูปที่ 4.1 โครงสร้างและการทำงานในส่วนต่างๆ ของชุดถอดรหัส Viterbi นี้ สามารถอธิบายได้ดังต่อไปนี้

4.1.1 โครงสร้างภาค Branch Metric Unit (BMU)

ภาค BMU เป็นส่วนที่ทำหน้าที่คำนวณหาค่า Branch Metric (BM) จากข้อมูลอินพุตที่รับเข้ามาซึ่งเป็นลำดับแรกของการถอดรหัสด้วยวิธีของ Viterbi Algorithm โดยข้อมูลอินพุตนี้จะถูกจัดระดับสัญญาณ (Quantizer) และทำให้สัมพันธ์ (Synchronizer) กับการทำงานของชุดถอดรหัสก่อนการคำนวณค่า Branch Metric เป็นการคำนวณค่าความแตกต่างของข้อมูลอินพุตกับค่าเอาต์พุตของการเข้ารหัสที่กำหนดไว้ตามแผนภาพ Trellis (Branch Word) แสดงดังสมการ [8]

$$BM_{i,j,n} = (r_n - C_{i,j})^2 \quad (4.1)$$

โดย ค่า BM แทนค่า Branch Metric r แทนค่าข้อมูลอินพุตและ C แทนค่าเอาต์พุตของการเข้ารหัสระหว่างสเตต i ไปยังสเตต j ณ เวลา n



BMU: Branch Metric Unit

ACS: Add-Compare-Select

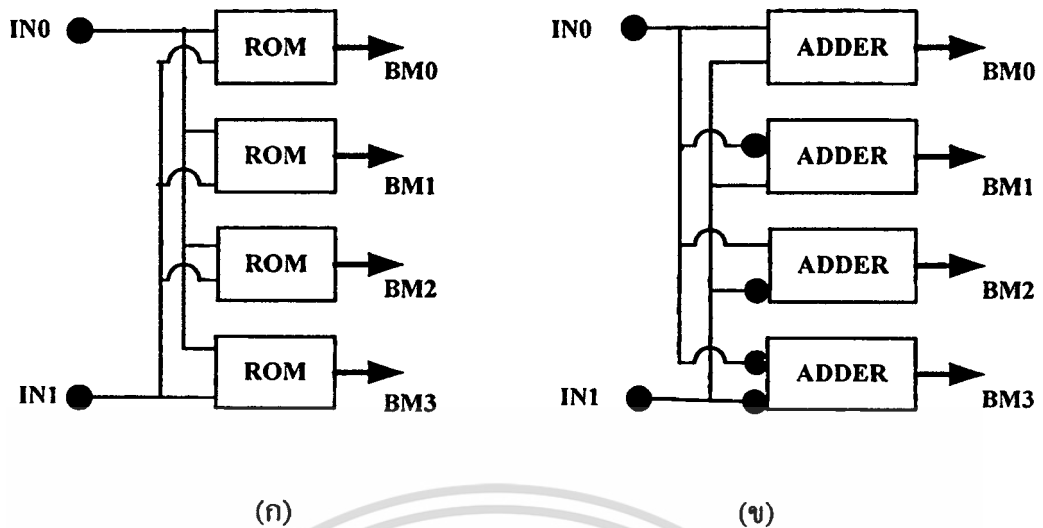
PMM: Path Metric Memory

ACSU: Add-Compare-Select Unit

SMU: Survivor Memory Unit

รูปที่ 4.1 บล็อกไดอะแกรมของชุดถอดรหัส Viterbi

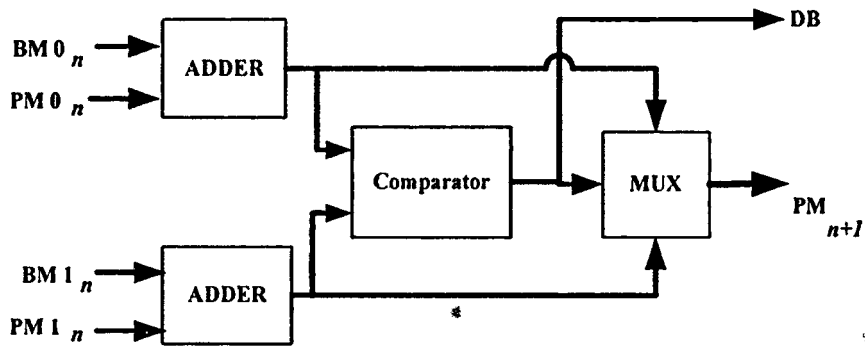
ซึ่งค่า Branch metric เกิดจากค่า Constraint Length ค่า Generator Polynomial และค่าอัตราการเข้ารหัส (Coding Rate) การคำนวณหาค่า Branch Metric นี้ต้องคำนวณทุกๆ สาขา (Branch) ตามแผนภาพ Trellis การทำงานของภาค BMU นั้นแบ่งตามรูปแบบในการถอดรหัสแบบ Viterbi ซึ่งมีอยู่ 2 รูปแบบ คือ Hard Decision และ Soft Decision ในการทำงานหาค่า Branch Metric แบบ Hard Decision ใช้วิธี “Hamming Distance” ซึ่งเป็นการเปรียบเทียบข้อมูลที่รับเข้ามากับค่าเอาต์พุตของการเข้ารหัสที่กำหนดไว้ในแผนภาพ Trellis (Branch Word) โดยค่า Branch Metric ซึ่งเป็นความแตกต่างข้อมูลอินพุตกับค่า Branch Word เช่น กรณีอัตราการเข้ารหัสเป็น 1/2 ข้อมูลอินพุตเป็น “00” ค่า Branch Word “11” จะได้ค่า Branch metric เท่ากับ 2 เป็นต้น ส่วน Soft Decision ใช้วิธี “Euclidean Distance” ซึ่งเป็นการเปรียบเทียบข้อมูลที่รับเข้ามาที่ถูกจัดระดับมากกว่าสองระดับกับค่า Branch Word เช่น กรณี Soft Decision แบบ 3 บิตและอัตราการเข้ารหัส 1/2 ข้อมูลอินพุตที่รับเข้ามาเป็น “000000” ค่า Branch Word “11” การหาค่า Branch Metric นี้จะนำค่า “000” มาเปรียบเทียบกับค่า Branch Word ทีละบิต ซึ่งกำหนดให้เท่ากับจำนวนของ Soft Decision ในที่นี้คือ “111” ซึ่งความแตกต่างสูงสุดคือ 7 ดังนั้นค่า Branch Metric ของกรณีนี้เท่ากับ 14 [20] โดยทั่วไปแล้วเทคนิคการออกแบบภาค BMU มีอยู่ 2 วิธี [20] คือ



รูปที่ 4.2 โค้ดแอมแกรมภาค BMU (ก) วิธีการเปิดตาราง (ข) วิธีการบวก ที่ R เท่ากับ 1/2 [20]

- วิธีการเปิดตาราง (ROM Look-up Table) เป็นวิธีการหาค่า Branch Metric โดยใช้ค่าอินพุตเลือกค่า Branch Metric ที่มีการกำหนดก่อนแล้วใน ROM
- วิธีการบวก (Haft Adder) เป็นวิธีการหาค่า Branch Metric โดยการบวกค่าอินพุตที่รับเข้ามาเข้าด้วยกัน ซึ่งวงจรการบวกจะกำหนดตามแผนภาพ Trellis ซึ่งวิธีการเปิดตารางและการบวกของอัตราการเข้ารหัสเท่ากับ 1/2 แสดงดังรูปที่ 4.2

โครงสร้างวิธีการเปิดตารางนั้นให้ง่ายที่ขนาดเล็กกว่าวิธีการบวก ซึ่งวิธีการนี้เหมาะสมกับการทำงานแบบ Hard Decision เนื่องจากการใช้งานวิธีการเปิดตารางกับแบบ Soft Decision นั้นมีความซับซ้อนจึงนิยมใช้วิธีการบวกมากกว่า วิธีการเปิดตารางนั้นมีโครงสร้างที่เหมาะสมสำหรับตัวถอดรหัสแบบขนาน เนื่องจากค่า Branch Metric ที่ส่งให้ภาค ACSU นั้นจะคงที่ไม่มีการเปลี่ยนแปลง ส่วนแบบวิธีการบวกนั้นเหมาะสมสำหรับตัวถอดรหัสแบบอนุกรมเนื่องจากค่า Branch Metric ใน Add-Compare-Select (ACS) จะเปลี่ยนแปลงตามค่าสเตทของการทำงานของชุดถอดรหัส



รูปที่ 4.3 โค้ดแแกรม Add-Compare-Select (ACS)

4.1.2 โครงสร้างภาค Add-Compare-Select Unit (ACSU)

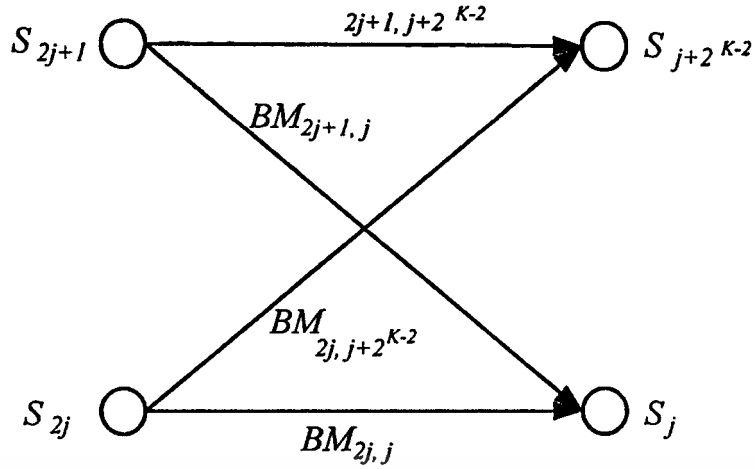
ภาค ACSU เป็นภาคที่ทำงานต่อจากภาค BMU ซึ่งภาค ACSU นี้ทำหน้าที่คำนวณค่า Path Metric (PM) หรือเรียกอีกอย่างหนึ่งว่าค่า State Metric (SM) และ Decision Bit (DB) ซึ่งต้องคำนวณทุกๆ สเตตตามแผนภาพ Trellis

วิธีการหาค่า Path Metric และ Decision Bit แต่ละสเตตนั้นจะทำโดย Add-Compare-Select (ACS) ซึ่งนำเอาค่า Branch Metric จาก BMU และค่า Path Metric หรือค่า State Metric ที่ถูกเก็บไว้ใน Path Metric Memory (PMM) มาบวกเข้าด้วยกัน ซึ่งการบวกนั้นจะทำทั้งสองค่าตามเส้นทางของแผนภาพ Trellis โดยค่า Path Metric ใหม่ (Update Path Metric) และ Decision Bit นั้นเลือกจากผลบวกของเส้นทางที่มีค่าน้อยกว่า ซึ่งการหา Path Metric ใน ACSU แสดงดังสมการ

$$PM_{n+1}^0 = \min(PM_n^0 + BM_n^0, PM_n^1 + BM_n^1) \quad (4.2)$$

โดยที่ $PM_{n+1}^i (i = 0, 1)$ และ $BM_{n+1}^i (i = 0, 1)$ คือค่า Path Metric และ Branch Metric ที่เวลา n ตามลำดับ [21] หลังจากนั้น Decision Bit ถูกส่งไปยัง SMU เพื่อใช้ในการถอดรหัส ส่วนค่า Path Metric ที่ไว้เลือกนั้นจะเก็บไว้ใน PMM สำหรับใช้คำนวณครั้งต่อไป การทำงานของวงจรดังแสดงในรูปที่ 4.3

โครงสร้างการทำงานของ ACSU ในคำนวณหาค่า Path Metric แบ่งตามโครงสร้างของชุดถอดรหัส Viterbi ซึ่งมีสองประเภท [3] คือ โครงสร้างแบบอนุกรม (Serial Architecture) และโครงสร้างแบบขนาน (Parallel Architecture)



รูปที่ 4.4 โครงสร้าง ACS แบบ Butterfly [22]

- โครงสร้างแบบอนุกรม (Serial Architecture)

โครงสร้างการทำงานของ ACSU นี้ประกอบไปด้วย ACS ตัวเดียวหรือเท่ากับ 2^n โดย n เท่ากับ 0, 1, 2, โดยการเพิ่มจำนวนของ ACS จะทำให้อัตราเร็วในการถอดรหัสของชุดถอดรหัสแบบอนุกรมเพิ่มขึ้น แต่การเพิ่มจำนวนของ ACS จะต้องไม่เท่ากับจำนวนเท่าที่แสดงของการทำงาน โดยการทำงานของ ACSU แบบนี้ใช้ Random Access Memory (RAM) ในการเก็บบันทึกข้อมูลต่างๆ ไว้โดยมีภาค Controller เป็นตัวควบคุมการทำงาน โครงสร้างแบบนี้จะทำให้อัตราเร็วในการถอดรหัสต่ำ แต่ข้อดีของตัวถอดรหัสจะมีขนาดเล็ก และโครงสร้างนี้เหมาะสำหรับการสร้างตัวถอดรหัส Viterbi ที่ที่ค่า Constraint Length สูง

- โครงสร้างแบบขนาน (Parallel Architecture)

โครงสร้างแบบนี้ประกอบไปด้วยตัว ACS เท่ากับจำนวนเท่าที่แสดงของการทำงาน เช่น ถ้าค่า Constraint Length เท่ากับ 9 จะใช้ 256 ACS และนิยมใช้โครงสร้างแบบ Butterfly [22] ซึ่งจะนำ 2 ACS มาต่อร่วมกันดังแสดงดังรูปที่ 4.4 โดยค่า Branch Metric ในโครงสร้างแบบ Butterfly จะเท่ากันซึ่งแสดงดัง

$$BM(j, 2j) = BM(j + 2^{K-2}, 2j+1) \quad (4.3)$$

$$BM(j, 2j+1) = BM(j + 2^{K-2}, 2j) = -BM(j, 2j) \quad (4.4)$$

โดย i และ j เป็นลำดับของสเตทการทำงานหรือลำดับของ ACS ที่นำมาต่อร่วมกัน ส่วน BM เป็นค่า Branch Metric และ K เป็นค่า Constraint Length โครงสร้างนี้จะทำให้ลดการเชื่อมต่อลงจึงทำให้

พื้นที่ในการสร้างชุดถอดรหัสลดลง และทำให้การใช้พลังงานลดลงด้วย ข้อดีของแบบขนานคือให้อัตราเร็วของข้อมูลที่สูง แต่ขนาดของวงจรมีขนาดใหญ่ ซึ่งโครงสร้างแบบขนานเหมาะสำหรับการสร้างตัวถอดรหัส Viterbi ที่มีค่า Constraint Length ต่ำเท่านั้น

4.1.3 โครงสร้างภาค Survivor Memory Unit (SMU)

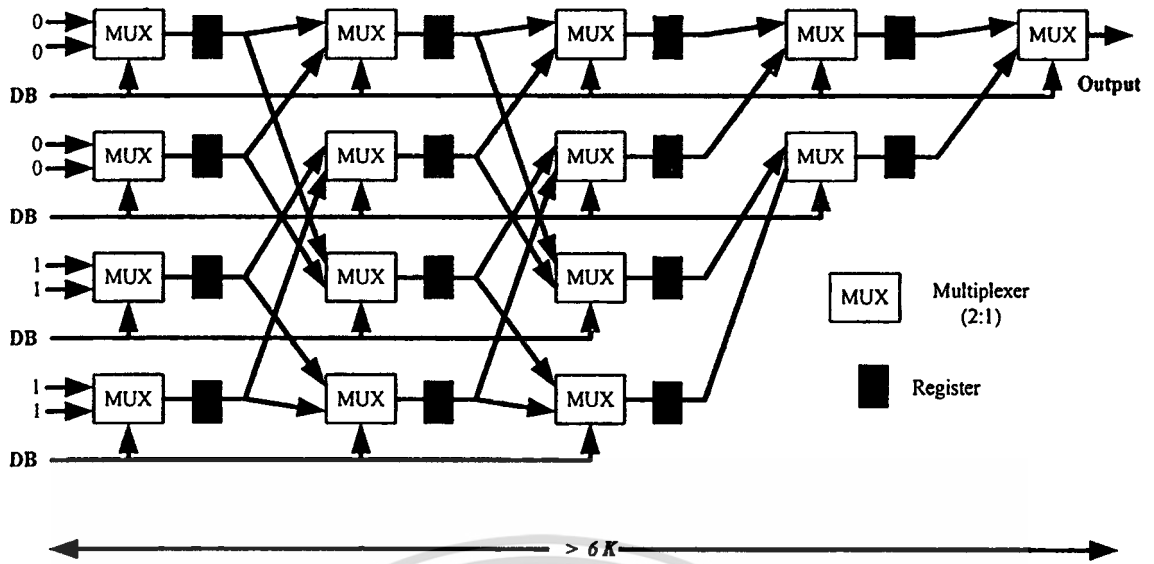
ภาค SMU เป็นภาคสุดท้ายในการทำงานของ Viterbi Algorithm โดยภาค SMU ทำหน้าที่ถอดรหัสข้อมูลจากข้อมูลอินพุตให้เป็นข้อมูลที่ถูกต้อง โดยการหาเส้นทางที่ของข้อมูลในแผนภาพ Trellisในการทำงานถอดรหัสจะใช้ Decision Bit จาก ACSU เป็นตัวตัดสินใจ โดยทั่วไปแล้วการทำงานของภาค SMU มี 2 วิธี [19] คือ วิธี Register Exchange (RE) และ วิธี Trace Back (TB) ซึ่งรายละเอียดมีดังนี้

4.1.3.1 วิธีทำงานแบบ Register Exchange

การทำงานของภาค SMU วิธี Register Exchange เป็นแบบ Forward ดังนั้นจึงสามารถทำงานถอดรหัสข้อมูลได้อย่างต่อเนื่องและรวดเร็ว [23] โดยโครงสร้างประกอบด้วยมัลติเพล็กซ์ (Multiplex) แบบ 2:1 และเรจิสเตอร์ (Register) โดยมัลติเพล็กซ์และเรจิสเตอร์จะถูกเชื่อมต่อกันเหมือนกับแผนภาพ Trellis โดยมัลติเพล็กซ์จะถูกควบคุมการทำงานโดย Decision Bit จากภาค ACSU ซึ่งลักษณะวงจรแสดงดังรูปที่ 4.5 โดยจำนวนของ Register ต้องมากกว่า 6 เท่าของค่า Constraint Length จึงจะทำให้ข้อมูลที่ถูกลดรหัสมีความถูกต้อง ซึ่งกรณีนี้ค่า Constraint Length เท่ากับ 9 จำนวนของ Register ต้องมากกว่า 54 จากรูปจะเห็นว่าวิธี Register Exchange มีโครงสร้างที่ไม่ซับซ้อน แต่มีโครงสร้างที่ใหญ่สำหรับชุดถอดรหัสที่มีค่า Constraint Length สูง และมีการใช้พลังงานมากเนื่องจากในการทำงานนั้นต้องมีการส่งข้อมูลให้กันตลอดเวลา [24]

4.1.3.2 วิธีทำงานแบบ Trace Back

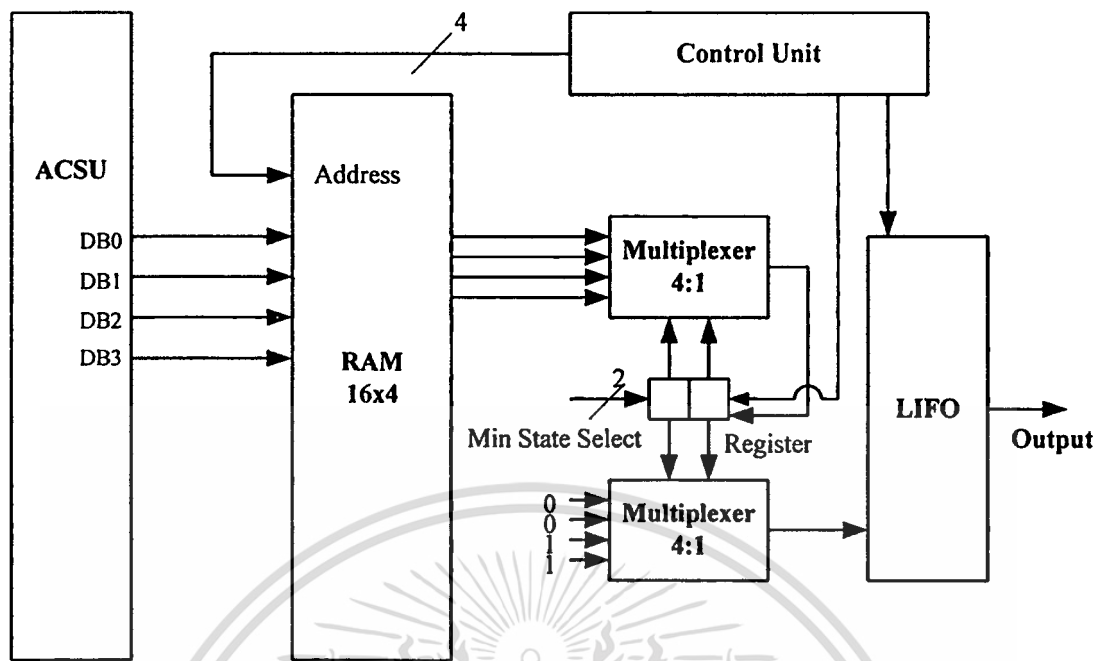
การทำงานของภาค SMU วิธี Trace Back จะเป็นการทำงานแบบ Backward ซึ่งการถอดรหัสเริ่มต้นจากจุดที่เวลามากกว่าหกเท่าของค่า Constraint Length เหมือนกับการทำงานของวิธี Register Exchange ซึ่งเรียกค่านี้ว่า “Trace Back Depth” และการเริ่มต้นจะเริ่มจากค่าสแตกการทำงานที่มีค่า Path Metric น้อยที่สุด แล้วทำการย้อนกลับหาค่าเส้นทางของข้อมูลตามแผนภาพ Trellisโดยใช้ Decision Bit จาก ACSU ตัดสินใจในการหาเส้นทาง [20] โดยตัวอย่างวงจรการทำงานแสดงดังรูปที่ 4.6 ซึ่งเป็นวงจร Trace Back ที่ค่า Constraint Length เท่ากับ 3 โดยวงจรประกอบด้วยหน่วยความจำขนาด 16×4 ซึ่งทำหน้าที่บันทึกค่า Decision Bit เมื่อหน่วยความจำเก็บ Decision Bit



รูปที่ 4.5 วงจร SMU แบบ Register Exchange ที่ค่า K เท่ากับ 3

ครบ 16 ค่า ซึ่งมากกว่าหกเท่าของค่า Constraint Length ก็จะเริ่มทำการถอดรหัสหรือ Trace Back โดยเริ่มจากการหาค่า Path Metric ที่น้อยที่สุดหรือเรียกว่า “Best State” มาเป็นจุดเริ่มต้น โดยภาค Min Path Metric Selection ส่วนตัดสินใจเลือกเส้นทางจะใช้มัลติเพล็กซ์แบบ 4:1 โดยเลือกจาก Decision Bit ที่ส่งจากหน่วยความจำ โดยสัญญาณการควบคุมมาจากข้อมูลที่ถูกเลือกโดยมัลติเพล็กซ์ในเวลาก่อนนั้นซึ่งอยู่ในชิพเรจิสเตอร์ ข้อมูลที่ถูกเลือกนี้จะเป็นข้อมูลที่ถูกถอดรหัสแต่ยังเป็นข้อมูลแบบย้อนกลับจึงจำเป็นต้องใช้ LIFO (Last In First Out RAM) กลับข้อมูลให้ถูกต้อง

วิธี Trace back นั้นเหมาะสมสำหรับชุดถอดรหัส Viterbi ที่มีค่า Constraint Length สูง เนื่องจากใช้อุปกรณ์ในการสร้างน้อย ส่วนความเร็วในการถอดรหัสจะช้ากว่า Register Exchange เนื่องจากต้องมีการหาค่า Path Metric น้อยที่สุด และต้องใช้หน่วยความจำในการทำงาน ส่วนความซับซ้อนของวงจรมากกว่าเนื่องจากต้องมีส่วนของภาค Controller ควบคุมการทำงาน ส่วนความเร็วในการถอดรหัสนั้นไม่สามารถทำได้ต่อเนื่องได้ เนื่องจากต้องรอ Decision Bit ที่เก็บไว้ในหน่วยความจำก่อนจึงสามารถถอดรหัสได้ โดยในการหา Path Metric ที่น้อยที่สุดนั้นเป็นวิธีการที่เหมาะสมสำหรับ ที่ชุดถอดรหัสที่มีค่า Constraint Length ขนาดเล็ก เพราะว่าการสร้าง ส่วน Min Path Metric Selection ที่มีการเปรียบเทียบจำนวนมากทำได้ยากและเกิดการซ้ำของข้อมูลมากสำหรับถอดรหัสแบบต่อเนื่องนั้นวงจรจะมีขนาดใหญ่และต้องใช้หน่วยความจำเพิ่มขึ้นเป็นสองเท่าเมื่อเทียบกับแบบไม่ต่อเนื่อง



รูปที่ 4.6 โค้ดแกรม SMU แบบ Trace Back ที่ค่า K เท่ากับ 3 [20]

4.1.4 การเปรียบเทียบโครงสร้างของชุดถอดรหัส Viterbi แบบต่างๆ

การสร้างชุดถอดรหัส Viterbi จากลักษณะ โครงสร้างและการทำงานของชุดถอดรหัส Viterbi ที่กล่าวมา การใช้พื้นที่ (Silicon Area) การใช้พลังงาน (Power Consumption) และความเร็วในการทำงานสามารถเปรียบเทียบกัน ได้ดังแสดงในตารางที่ 4.1

ตารางที่ 4.1 ผลเปรียบเทียบข้อดีข้อเสียของโครงสร้างของชุดถอดรหัส Viterbi แบบต่างๆ

	BMU		ACSU		SMU	
	เปิดตาราง	การบวก	อนุกรม	ขนาน	Register Exchange	Trace Back
การใช้พื้นที่	น้อย	มาก	น้อย	มาก	มาก	น้อย
การใช้พลังงาน	ต่ำ	สูง	ต่ำ	สูง	สูง	ต่ำ
ความเร็ว	สูง	ต่ำ	ต่ำ	สูง	สูง	ต่ำ

ตารางที่ 4.2 รายละเอียดของชุดถอดรหัส Viterbi ในวิทยานิพนธ์นี้

Coding Rate	1/2 และ 1/3
Constraint Length	9
Generator Polynomials	561 ₈ , 753 ₈ ที่ R เท่ากับ 1/2 557 ₈ , 663 ₈ , 711 ₈ ที่ R เท่ากับ 1/3
Survivor Path Length	64
Decision Level	Soft Decision แบบ 8 ระดับ (3 บิต) การแทนค่าแบบ Offset Binary Representation
Path Metric	8 บิต Modulo Normalization
ACS	4
Data Transmission	แบบต่อเนื่อง (Continuous)

4.2 วิธีการออกแบบ

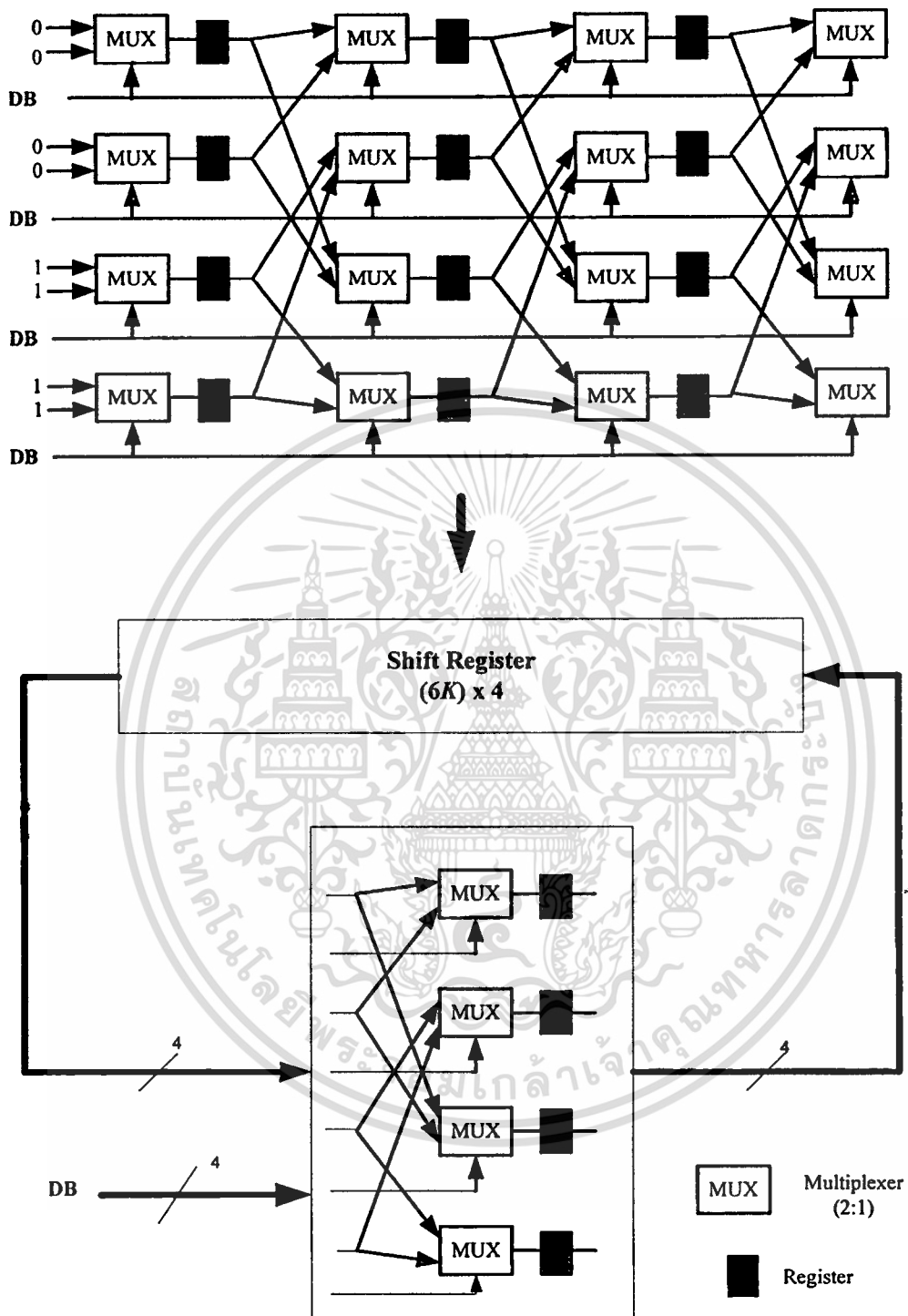
ชุดถอดรหัสแบบ Viterbi สำหรับงานวิทยานิพนธ์นี้ได้ออกแบบตามมาตรฐานของ 3GPP โดยรายละเอียดการออกแบบระบุไว้ดังตารางที่ 4.2

4.2.1 แนวความคิดและที่มาในการออกแบบสร้าง

แนวความคิดและที่มาในการออกแบบชุดถอดรหัสในวิทยานิพนธ์นี้ได้เริ่มต้นพัฒนาจาก

- การสร้างชุดถอดรหัส Viterbi ในงานวิจัย [25] การสร้างใช้โครงสร้างแบบขนานจึงใช้พื้นที่ในการสร้าง (Silicon Area) มากและทำให้ต้นทุนในการสร้างสูง วิทยานิพนธ์นี้จึงนำได้เสนอชุดถอดรหัส Viterbi แบบอนุกรมมาใช้งานแทน ทำให้พื้นที่ในการสร้างลดลงและราคาถูกลงกว่างานวิจัย [25]
- การสร้างชุดถอดรหัสแบบ Viterbi ในงานวิจัย [3] การสร้างใช้โครงสร้างแบบอนุกรมที่ใช้ ACS ตัวเดียว การทำงานใช้เวลาเท่ากับจำนวนสเตท ในกรณีค่า Constraint Length เท่ากับ 9 (256 สเตท) ใช้เวลาจำนวน 256 สัญญาณนาฬิกาต่อการถอดรหัสข้อมูลหนึ่งบิต การเพิ่มโครงสร้างเป็น 4 ACS จะช่วยลดเวลาในการทำงานลงเหลือ 64 สัญญาณนาฬิกา และวิธีการปรับปรุงภาค SMU แบบ Register Exchange นี้สามารถออกแบบให้ทั้งสองทำงานพร้อมกัน ทำให้การทำงานในการถอดรหัสเร็วขึ้นกว่างานวิจัย [3]
- การสร้างชุดถอดรหัสแบบ Viterbi ในงานวิจัย [3][26] การสร้างจะใช้ภาค SMU แบบ Trace Back มีการทำงานที่ช้า วิทยานิพนธ์นี้จึงได้เสนอวิธีการปรับปรุงภาค SMU แบบ Register Exchange ซึ่งจะช่วยลดขนาดของวงจรลง และทำงานเร็วกว่าการสร้างในงานวิจัย [3][26]

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.7 ภาค SMU แบบ Register Exchange ที่มีการปรับปรุง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

4.2.2 วิธีการปรับปรุงภาค SMU แบบ Register Exchange

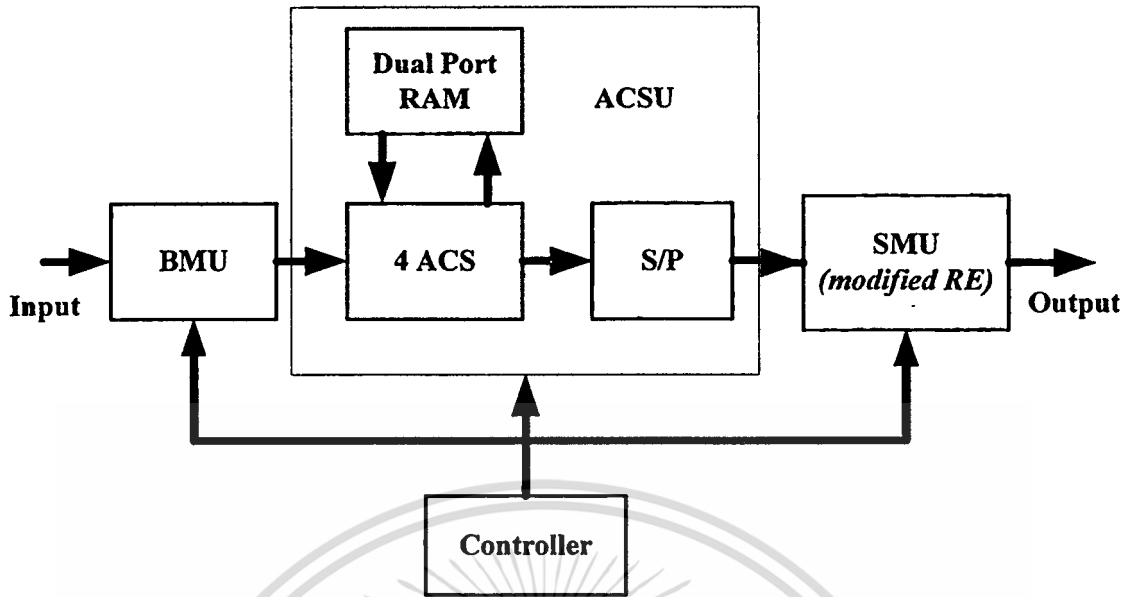
การปรับปรุงภาค SMU แบบ Register Exchange นี้จะทำให้ขนาดวงจรลดลงและเหมาะสมกับโครงสร้างอนุกรมแบบ 4 ACS ซึ่งวิธีการปรับปรุง Register Exchange นี้มีหลักการคือ จะใช้การทำงานแบบวนรอบ (Circular) โดยวงจรประกอบด้วยชิพเรจิสเตอร์และมัลติเพล็กซ์ ซึ่งการทำงานจะใช้มัลติเพล็กซ์เพียงชุดเดียวในการแลกเปลี่ยนข้อมูล (Decision Bit) ซึ่งข้อมูลจะเก็บไว้ที่ชิพเรจิสเตอร์ ซึ่งตัวอย่างการทำงานของภาค SMU แบบ Register Exchange ที่มีการปรับปรุงแสดงดังรูปที่ 4.7 โดยการปรับปรุงนี้ทำให้ใช้มัลติเพล็กซ์เพียงชุดเดียวจากเดิมที่ต้องใช้เท่ากับจำนวนเท่ากับค่า Survivor Path Length ซึ่งเป็นความยาวของ Register Exchange ที่ทำให้ได้ข้อมูลที่ถูกถอดรหัสแล้ว ความถูกต้อง (ต้องมีค่ามากกว่า 6 เท่าของ Constraint Length) จึงทำให้ขนาดวงจรลดลง การปรับปรุง Register Exchange นี้ทำให้เวลาการทำงานเพิ่มขึ้นเป็น 64 สัญญาณนาฬิกา (เท่ากับค่า Survivor Path Length ในการออกแบบใช้ 64 ซึ่งเหมาะสมกับโครงสร้างอนุกรม 4 ACS ในการออกแบบตัวถอดรหัส Viterbi แบบอนุกรมที่มีการปรับปรุง Register Exchange มีไดอะแกรมของชุดถอดรหัสที่ออกแบบใหม่แสดงดังรูปที่ 4.8 โดยส่วนที่สำคัญคือจะใช้ Dual-port RAM สำหรับบันทึก Path Metric และใช้วงจร Serial to Parallel (S/P) ในการเปลี่ยนข้อมูลของอนุกรมให้เป็นขนาน ซึ่งรายละเอียดการออกแบบอธิบายได้ตามการทำงานของภาคต่างๆ ได้ดังนี้

4.2.3 ภาค BMU แบบ 3 บิต Soft Decision

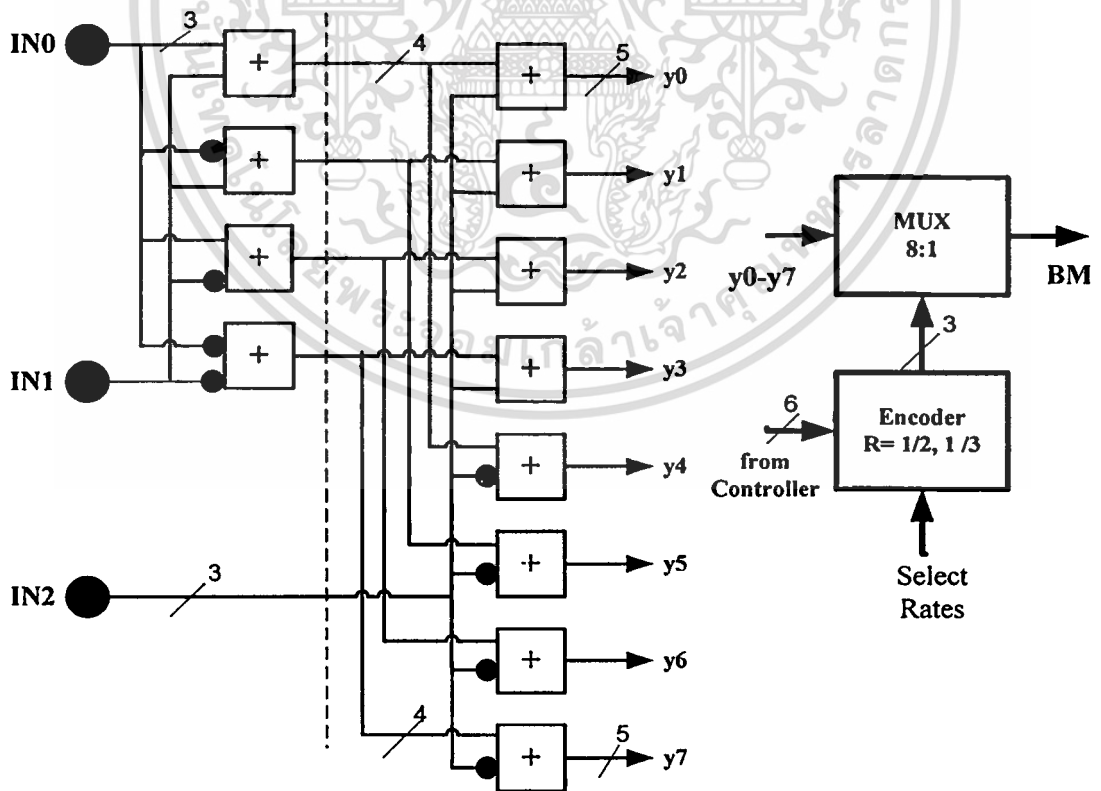
การออกแบบให้ชุดถอดรหัสสามารถการเลือกอัตราการถอดรหัสได้สองอัตรา คือ 1/2 และ 1/3 จะอยู่ในส่วนของภาค BMU นี้ ซึ่งภาค BMU ถูกออกแบบให้ทำงานแบบ Soft Decision 8 ระดับ (3 บิต) เนื่องจากการทำงานแบบ Soft Decision 8 ระดับนี้ให้ค่า Coding Gain มากกว่าแบบ Hard Decision ประมาณ 2 dB ในช่องสัญญาณแบบ AWGN [4] การแทนค่าใน BMU นี้มีสองแบบคือแบบ Signed Magnitude และ Offset Binary [27] ซึ่งการแทนค่าแสดงดังตารางที่ 4.3 การแทนค่าแบบ Offset Binary จะใช้หน่วยในการสร้างน้อยกว่า [18] จึงถูกเลือกใช้สำหรับวิทยานิพนธ์นี้ โดยวงจรของภาค BMU นี้ประกอบด้วย 12 วงจรบวกขนาด 8 บิต (8 bit Adders) 8 มัลติเพล็กซ์ (4:1) (Multiplexes 8:1) และ 8 วงจรเข้ารหัสคอนโวลูชัน (Convolution Encoders) ดังแสดงในรูปที่ 4.9 การทำงานของภาค BMU อธิบายได้คือ วงจรบวกทั้งหมดจะทำหน้าที่ในการหาค่า Branch Metric (Euclidean Distance) ซึ่งมี 8 ค่าคือ Y0 ถึง Y7 ซึ่งการคำนวณค่าแสดงดัง

$$BM(b_0, b_1, b_2) = -IN_0s(b_0) - IN_1s(b_1) - IN_2s(b_2) \quad (4.4)$$

โดย ค่า b_0, b_1, b_2 แทนตำแหน่ง Y0 ถึง Y7 ส่วน S แทนค่าเครื่องหมายจะตรงกันข้ามกับ b_0, b_1, b_2 เช่น b_0, b_1, b_2 เท่ากับ (0, 1, 0) ค่าเครื่องหมาย S เท่ากับ (-, +, -) ส่วนในการเลือกอัตราการถอดรหัส เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.8 ชุดถอดรหัส Viterbi แบบ 4 ACS ที่มีการปรับปรุง Register Exchange



รูปที่ 4.9 การทำงานของวงจร BMU แบบ 3 บิต Soft Decision

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 4.3 การแทนข้อมูลอินพุต แบบ Soft Decision 3 บิต [27]

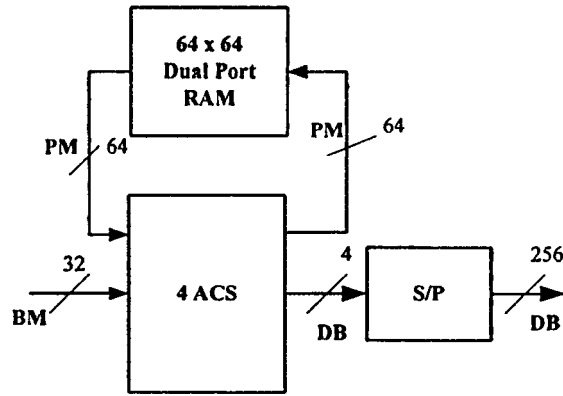
	Signed Magnitude	Offset Binary
เท่ากับ “1” มากที่สุด (Strongest “1”)	111	111
	110	110
	101	101
เท่ากับ “1” น้อยที่สุด (Weakest “1”)	100	100
เท่ากับ “0” น้อยที่สุด (Weakest “0”)	000	011
	001	010
	010	001
เท่ากับ “0” มากที่สุด (Strongest “0”)	011	000

ที่สามารถกำหนดให้เป็น 1/2 หรือ 1/3 นั้นจะกำหนดที่วงจรเข้ารหัสคอนโวลูชัน โดยการกำหนดค่า อัตราการเข้ารหัส ซึ่งเอาต์พุตวงจรเข้ารหัสจะทำหน้าที่เลือกค่า Branch Metric โดยเลือกค่า Branch Metric Y0 ถึง Y7 โดยอัตราการถอดรหัส 1/2 นั้นมีสองเอาต์พุตทำให้ใช้ค่า Y0 ถึง Y4 ส่วนอัตรา การถอดรหัส 1/3 นั้นมีสามเอาต์พุตทำให้ใช้ครบทั้งหมด (Y0 ถึง Y7) ซึ่งการทำงาน BMU นี้จะถูก ควบคุมโดย Controller เพื่อให้สัมพันธ์กับการทำงานของ ACSU ต่อไป ชุดถอดรหัสนี้ต้องทำงานที่ ความถี่สูง จึงได้แบ่งการทำงานของ BMU เป็น 2 Pipelines ซึ่งทำให้ BMU สามารถทำงานได้เร็วขึ้น ซึ่งวงจรการทำงานแสดงดังรูปที่ 4.9 ชุดถอดรหัส Viterbi นี้ใช้โครงสร้างอนุกรมแบบ 4 ACS มี ค่าสเตทเท่ากับ 256 สเตท การทำงานของภาค BMU จะใช้เวลาในการทำงาน 64 สัญญาณนาฬิกา

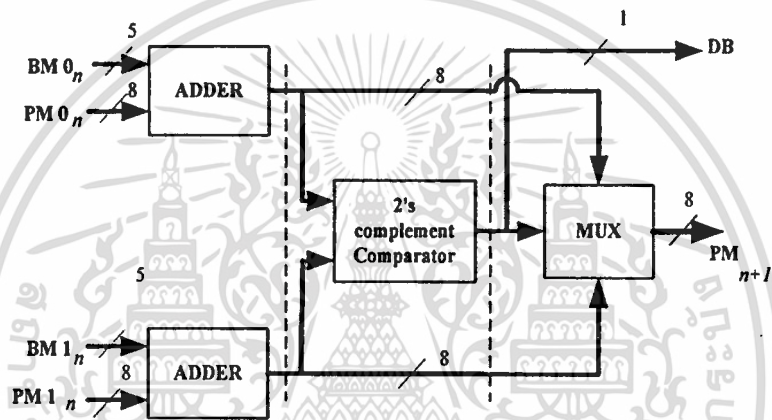
4.2.2 ภาค ACSU แบบอนุกรม 4 ACS

การออกแบบภาค ACSU จะออกแบบเป็นโครงสร้างอนุกรมแบบ 4 ACS ซึ่งชุดถอดรหัส Viterbi มีค่าสเตทเท่ากับ 256 สเตท ดังนั้นการทำงานของภาค ACSU ใช้เวลาในการทำงาน 64 สัญญาณนาฬิกา โดยภาค ACSU นี้ประกอบด้วย 4 ACS Dual-port RAM ขนาด 64 x 64 และ Serial to Parallel (S/P) ดังแสดงในรูปที่ 4.10 วงจร ACS ทำหน้าที่คำนวณค่า Path Metric และ Decision Bit โดยค่า Path Metric ที่ได้จะถูกเก็บไว้ที่ Dual-port RAM ซึ่งทำหน้าที่เป็น Path Metric Memory (PMM) เมื่อครบรอบการทำงานทั้ง 256 สเตท (64 สัญญาณนาฬิกา) Decision Bit จะถูก เปลี่ยนเป็นข้อมูลแบบขนาน 256 บิตโดย Serial to Parallel แล้วส่งให้ภาค SMU ต่อไปโดยสัญญาณ การควบคุมการทำงานของ ACSU นั้นมาจากภาค Controller

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.10 บล็อกไดอะแกรมการทำงานของภาค ACSU ที่นำเสนอ

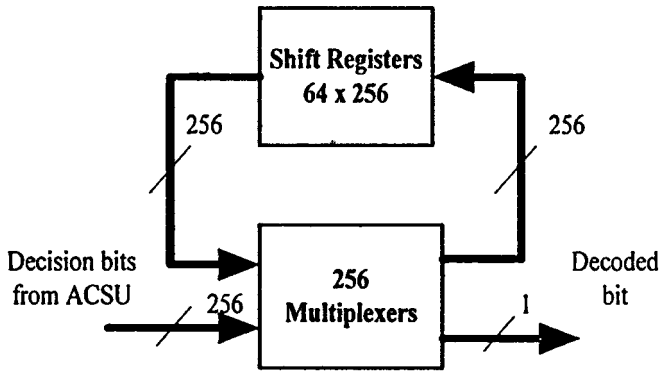


รูปที่ 4.11 ไดอะแกรม ACS ที่ใช้เทคนิค Modulo Normalization [28]

เนื่องจากการบวกสะสมของตัว ACS จะเกิด Overflow ทำให้การทำงานไม่ถูกต้อง การแก้ปัญหาการเกิด Overflow ของ ACS ในวิทยานิพนธ์นี้ได้ใช้เทคนิค Modulo Normalization [28] โดย ACS ประกอบด้วยวงจรบวกจะมีขนาด 8 บิต วงจรตัวเปรียบเทียบแบบ 2's Complement และมัลติเพล็กซ์ ซึ่งวงจรการทำงานของ ACS นี้แสดงดังรูปที่ 4.11 และชุดลอจิกเหล่านี้ต้องทำงานที่ความถี่สูง จึงได้แบ่งการทำงานของ ACS เป็น 3 Pipelines ดังรูปที่ 4.11 ซึ่งทำให้ ACS สามารถทำงานได้เร็วขึ้น

4.2.3 ภาค SMU ที่มีการปรับปรุง Register Exchange

การออกแบบภาค SMU ที่มีการปรับปรุง Register Exchange นี้ ซึ่งโครงสร้างแบบนี้จะลดขนาดของวงจรลง โดยในภาค SMU นี้ประกอบด้วยชิพเรจิสเตอร์ขนาด 64 x 256 (64 x 256 Shift Register) และ 256 มัลติเพล็กซ์ 2:1 (Multiplexers 2:1) การทำงานเริ่มต้นที่ Decision Bit จำนวน 256 บิตจาก ACSU ถูกบันทึกที่ชิพเรจิสเตอร์ จากนั้นข้อมูลในชิพเรจิสเตอร์นี้ถูกส่งเข้าไปยัง



รูปที่ 4.12 บล็อกไดอะแกรม SMU ที่มีการปรับปรุง Register Exchange

256 มัลติเพล็กซ์ตามจังหวะสัญญาณนาฬิกา ซึ่งข้อมูลจากมัลติเพล็กซ์นี้จะถูกเขียนกลับชิฟเรจิสเตอร์ซึ่งการทำงานของภาค SMU นี้แสดงดังรูปที่ 4.12 โดยการทำงานนี้จะเหมือนกับการทำงานในรูปที่ 4.5 แต่ใช้เวลามากกว่า 64 เท่า ซึ่งการทำงานนี้เข้ากันพอดีกับภาค ACSU ที่ออกแบบไว้ ดังนั้นชุดถอดรหัส Viterbi นี้ใช้เวลา 64 สัญญาณนาฬิกาในการถอดรหัสข้อมูลแต่ละบิต โดยสัญญาณการอ่านและเขียนชิฟเรจิสเตอร์และการควบคุมการถอดรหัสทั้งหมดมาจากภาค Controller ซึ่งภาค Controller ประกอบด้วย ตัวนับขนาด 6 บิต (6-bit Counter) และวงจรสร้างสัญญาณเมื่อเริ่มต้นการทำงานที่ตัวนับเท่ากับ 0 และจบการทำงานที่ตัวนับเท่ากับ 63 บล็อกไดอะแกรมภาค SMU ที่มีการปรับปรุง Register Exchange นี้แสดงดังรูปที่ 4.12

บทที่ 5

ผลการออกแบบและทดสอบการทำงาน

การออกแบบชุดถอดรหัส Viterbi ขนาดเล็กสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 สำหรับ วิทยานิพนธ์นี้ใช้ภาษาเวริล็อก (Verilog HDL) ในการบรรยายพฤติกรรมการทำงานของวงจรชุดเข้ารหัสและถอดรหัส สำหรับติดตั้งในเอฟพีจีเอ Xilinx Vertex II รุ่น xc2v1000-6 โดยใช้โปรแกรม Xilinx ISE ในการออกแบบ การสังเคราะห์วงจร (Synthesis) และการติดตั้ง และใช้โปรแกรม ModelSim ในการจำลองการทำงานและการตรวจสอบ ในบทนี้แสดงถึงผลการออกแบบและทดสอบการทำงาน โดยส่วนแรกแสดงผลการออกแบบ การใช้อุปกรณ์ภายในเอฟพีจีเอ ความถี่สูงสุดที่วงจรสามารถทำงานได้ และผลการจำลองการทำงานของชุดถอดรหัสโดยแบ่งตามภาคต่างๆ ในส่วนที่สองแสดงผลการทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณ AWGN (Additive White Gaussian Noise) และในส่วนสุดท้ายกล่าวถึงการวิเคราะห์ประสิทธิภาพและลักษณะเด่นของชุดถอดรหัสเมื่อเปรียบเทียบกับชุดถอดรหัสของบริษัท Xilinx และชุดถอดรหัสในงานวิจัยอื่นๆ

5.1 ผลการออกแบบและจำลองการทำงาน

การออกแบบชุดเข้ารหัสและถอดรหัสนี้ได้ใช้ภาษาเวริล็อกในการออกแบบ ผลที่ได้จากการออกแบบเมื่อนำไปสังเคราะห์วงจรโดยโปรแกรม Xilinx ISE จะได้เป็นกลุ่มของเน็ตลิสต์ (Netlist) ในระดับเกต ซึ่งสามารถตรวจสอบความถูกต้องโดยการจำลองการทำงาน ผลการออกแบบในส่วนนี้แสดงถึงผังวงจรในระดับ RTL จากโปรแกรม Xilinx ISE ผลการใช้อุปกรณ์ภายในเอฟพีจีเอ Xilinx Vertex II รุ่น xc2v1000-6 และความถี่สูงสุดที่วงจรสามารถทำงานได้ ซึ่งชุดถอดรหัส Viterbi นี้แบ่งเป็น 4 ส่วน ประกอบด้วยภาค Branch Metric Unit (BMU) ภาค Add-Compare-Select Unit (ACSU) ภาค Survivor Memory Unit (SMU) และภาค Controller ในแต่ละส่วนนั้นที่ใช้ภาษาเวริล็อกในการออกแบบ เรียกว่า “มอดูล” ซึ่งเมื่อทดสอบโดยการจำลองการทำงานทุกมอดูลแล้วก็จึงนำมอดูลย่อยต่างๆ มาประกอบกันเป็นมอดูลหลัก (Top Module) ซึ่งผลการออกแบบและการจำลองการทำงานชุดเข้ารหัสคอนโวลูชันและชุดถอดรหัส Viterbi แบ่งตามภาคต่างๆ มีดังนี้

5.1.1 มอดูลชุดเข้ารหัสคอนโวลูชันที่ R เท่ากับ $1/2$ และ $1/3$

มอดูลชุดเข้ารหัสคอนโวลูชันเป็นส่วนหนึ่งของมอดูลทดสอบชุดถอดรหัส Viterbi นี้ ซึ่งประกอบด้วยชิพรีจิสเตอร์ และ XOR Gate โดยมอดูลชุดเข้ารหัสคอนโวลูชันนี้แบ่งเป็นสองส่วนคือ อัตราการเข้ารหัสเท่ากับ $1/2$ และ $1/3$ ซึ่งผลการออกแบบและการจำลองการทำงานของมอดูลชุดเข้ารหัสมีดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.1 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูลชุดเข้ารหัส

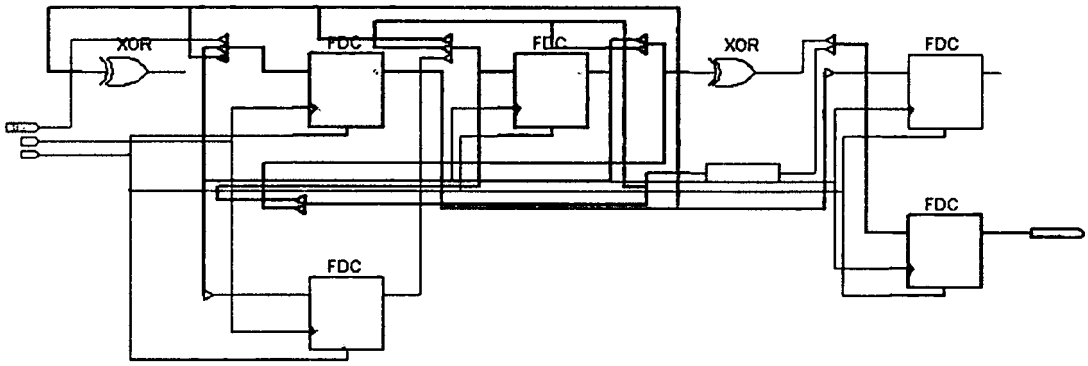
อุปกรณ์	จำนวน
R= 1/2	
Registers	11
1-bit register	11
XORs	3
1-bit xor2	1
1-bit xor4	2
R= 1/3	
Registers	12
1-bit register	12
XORs	7
1-bit xor2	5
1-bit xor3	1
1-bit xor4	1

ตารางที่ 5.2 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูลชุดเข้ารหัส

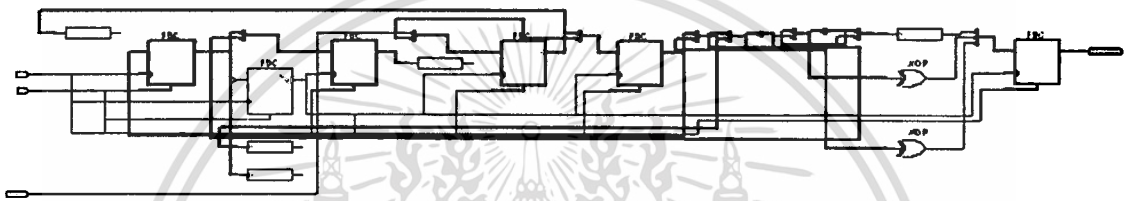
อุปกรณ์	Slices	Slice Flip Flops	4 input LUTs	Block RAMs
จำนวน (R=1/2)	7	11	4	0
จำนวน (R=1/3)	8	12	6	0
Rate (R)			1/2	1/3
Maximum Frequency			437 MHz	424 MHz
Minimum input arrival time before clock			1.962 ns	2.000 ns
Maximum output required time after clock			5.880 ns	5.880 ns

ผลการสังเคราะห์มอดูลชุดเข้ารหัส

ผลการสังเคราะห์มอดูลชุดเข้ารหัสด้วยโปรแกรม Xilinx ISE จะได้กลุ่มของเน็ตลิสต์ในระดับเกต ซึ่งประกอบไปด้วยอุปกรณ์ดิจิทัลประเภทต่างๆ ซึ่งแสดงดังในตารางที่ 5.1 และเมื่อนำอุปกรณ์เหล่านี้มาติดตั้งในเอฟพีจีเอ รุ่น xc2v1000 อุปกรณ์เหล่านี้จะถูกติดตั้งที่ในเอฟพีจีเอ ซึ่ง



(ก) อัตราการเข้ารหัสเท่ากับ 1/2

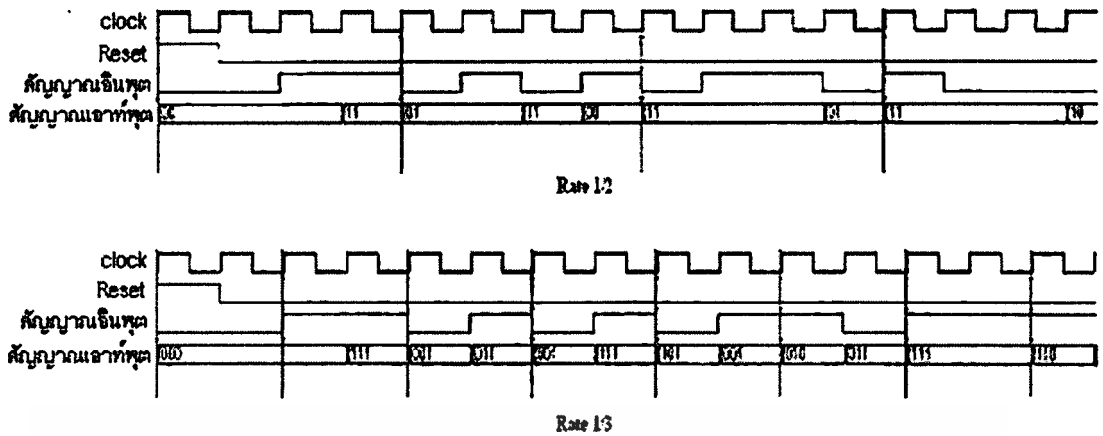


(ข) อัตราการเข้ารหัสเท่ากับ 1/3

รูปที่ 5.1 ผังวงจรระดับ RTL ของมอดูลชุดเข้ารหัสที่ได้จากการสังเคราะห์

(ก) อัตราการเข้ารหัสเท่ากับ 1/2 (ข) อัตราการเข้ารหัสเท่ากับ 1/3

อุปกรณ์ในเอฟพีจีเอประกอบด้วยตัว Slices ฟลิปฟลอป (Slice Flip-Flop) ตัวเปิดตารางสี่อินพุต (4-input LUT) และบล็อกหน่วยความจำ (Block RAM) ตารางที่ 5.2 แสดงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v-1000 และความเร็วการทำงานของมอดูลชุดเข้ารหัส โดยตารางประกอบด้วยจำนวน Slices จำนวนฟลิปฟลอป (Slice Flip Flop) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) ซึ่งหมายถึง การป้อนสัญญาณอินพุตต้องป้อนเข้าก่อนเวลานี้นับจากเวลาเกิดการเปลี่ยนแปลงของสัญญาณนาฬิกา และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตจะเกิดหลังจากเวลานี้นับจากเวลาเกิดการเปลี่ยนแปลงสัญญาณนาฬิกา ส่วนรูปที่ 5.1 แสดงผังวงจรระดับ RTL ของมอดูลชุดเข้ารหัสที่ได้จากการสังเคราะห์ซึ่งประกอบด้วยดีฟลิปฟลอปและ XOR Gate



รูปที่ 5.2 ผลการจำลองการทำงานของมอดูลชุดเข้ารหัสที่ R เท่ากับ $1/2$ และ $1/3$

ผลการจำลองการทำงานของมอดูลชุดเข้ารหัส

การจำลองการทำงานของมอดูลชุดเข้ารหัสนี้เป็นการป้อนสัญญาณอินพุต ซึ่งสัญญาณอินพุตทดสอบสามารถป้อนโดยใช้โปรแกรม Xilinx ISE หรือเขียนโปรแกรมทดสอบ (Testbench) โดยภาษาเวริลล็อก การจำลองการทำงานจะได้สัญญาณเอาต์พุตซึ่งสามารถตรวจสอบความถูกต้องของการทำงานของวงจรได้ ซึ่งสัญญาณทดสอบของมอดูลชุดเข้ารหัสนี้ประกอบด้วยสัญญาณนาฬิกา (clock) สัญญาณรีเซ็ต (reset) สัญญาณนาฬิกาของข้อมูลอินพุต ในส่วนแรกทดสอบที่อัตราเข้าถอดรหัสเท่ากับ $1/2$ และส่วนหลังทดสอบที่อัตราการเข้ารหัส $1/3$ ซึ่งผลการจำลองการทำงานของมอดูลชุดเข้ารหัสนี้แสดงดังรูปที่ 5.2 ซึ่งผลการจำลองการทำงานที่ได้นี้สามารถตรวจสอบได้ว่าการทำงานของวงจรถูกต้องจากสมการของวงจรรหัสเข้ารหัส หรือแผนภาพชนิดต่างๆ ดังกล่าวไว้ในบทที่ 2

5.1.2 มอดูล Branch Metric Unit ที่ R เท่ากับ $1/2$ และ $1/3$

มอดูล BMU (Branch Metric Unit) เป็นส่วนแรกของชุดถอดรหัสและเป็นส่วนที่ใช้เลือกอัตราการถอดรหัสว่าให้เป็น $1/2$ หรือ $1/3$ ซึ่งมอดูล BMU ประกอบด้วย วงจรบวกขนาด 8 บิต จำนวน 12 วงจร วงจรมัลติเพล็กซ์ (8:1) จำนวน 8 วงจร และวงจรเข้ารหัสคอนโวลูชันจำนวน 8 วงจร

ตารางที่ 5.3 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล BMU

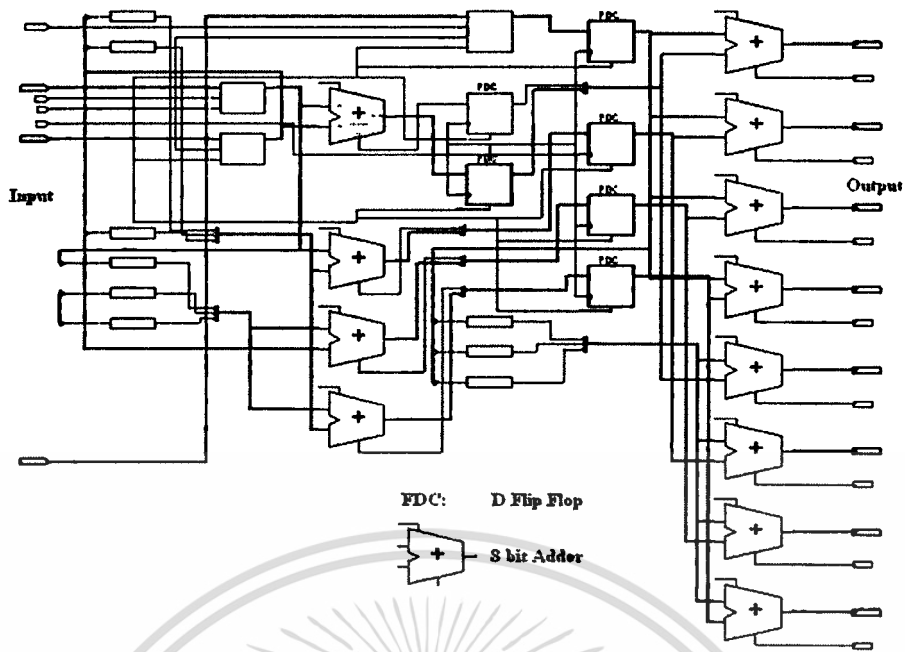
อุปกรณ์	จำนวน
Registers	48
3-bit register	4
5-bit register	16
4-bit register	4
1-bit register	24
Multiplexers	24
2-to-1 multiplexer	16
5-bit 8-to-1 multiplexer	8
Adders	12
4-bit adder carry out	8
3-bit adder carry out	4
XORs	82
1-bit xor2	40
1-bit xor3	40

ตารางที่ 5.4 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล BMU

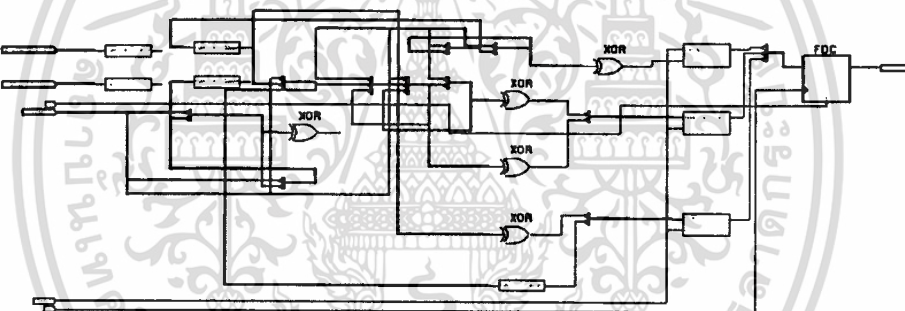
อุปกรณ์	Slices	Slice Flip Flops	4 input LUTs	Block RAMs
จำนวน	139	122	219	0
Maximum Frequency			295 MHz	
Minimum input arrival time before clock			3.208 ns	
Maximum output required time after clock			5.880 ns	

ผลการสังเคราะห์มอดูล Branch Metric Unit (BMU)

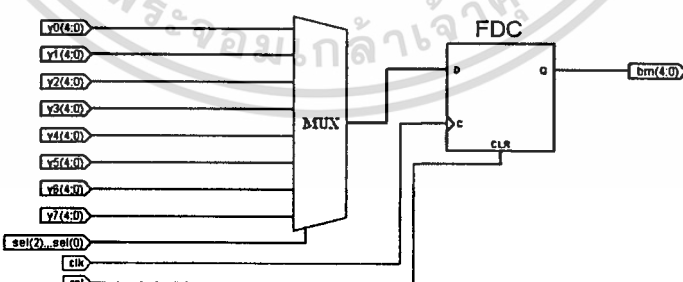
ผลการสังเคราะห์วงจรภาค BMU โดยโปรแกรม Xilinx ISE จะได้กลุ่มของเน็ตลิสต์ซึ่งประกอบไปด้วยอุปกรณ์ดิจิทัลประเภทต่างๆ แสดงดังในตารางที่ 5.3 และในตารางที่ 5.4 แสดงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล BMU โดยตารางประกอบด้วย จำนวน Slices จำนวนฟลิปฟล็อป (Slice Flip Flop) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนบล็อกหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum



(ก) วงจรบวกขนาด 8 บิตจำนวน12 วงจร



(ข) วงจรเข้ารหัสคอนไวลูชัน



(ค) วงจรมัลติเพล็กซ์ (8:1)

รูปที่ 5.1 ผังวงจรระดับ RTL ของมอดูล BMUที่ได้จากการสังเคราะห์

- (ก) วงจรบวกขนาด 8 บิตจำนวน12 วงจร
- (ข) วงจรเข้ารหัสคอนไวลูชัน
- (ค) วงจรมัลติเพล็กซ์ (8:1)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

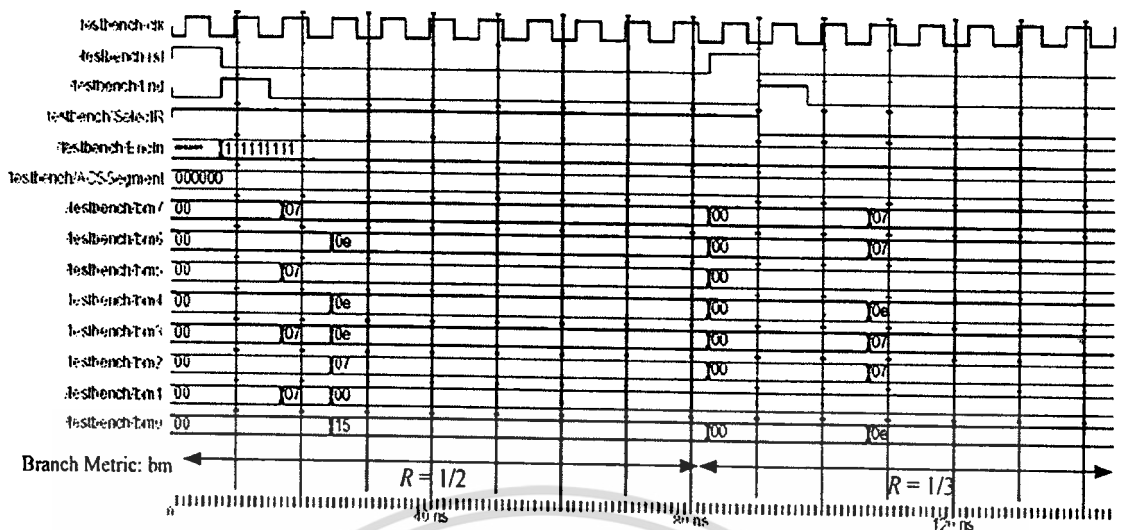
Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตเกิดหลังสัญญาณนาฬิกา (Maximum output required time after clock) โดยค่าทั้งสองนี้เป็นสิ่งที่สำคัญในการป้อนสัญญาณอินพุตในการทดสอบจำลองการทำงาน การเชื่อมต่อมอดูลต่างๆ เข้าด้วยกัน และมีผลต่อความถี่สูงสุดของที่ป้อนมอดูล โดยความถี่สูงสุดที่ที่ป้อนมอดูลจะไม่เกินความถี่สูงสุดของมอดูลนี้ ส่วนผังวงจรระดับ RTL ของมอดูล BMU ที่ได้จากการสังเคราะห์แสดงในรูปที่ 5.3

ผลการจำลองการทำงานมอดูล Branch Metric Unit

การจำลองการทำงานของมอดูล BMU นี้เป็นการป้อนสัญญาณอินพุต ซึ่งสัญญาณอินพุตทดสอบป้อนโดยใช้โปรแกรม Xilinx ISE ซึ่งสัญญาณทดสอบของมอดูล BMU นี้ประกอบด้วย สัญญาณนาฬิกาซึ่งกำหนดเป็น clk สัญญาณรีเซ็ตซึ่งกำหนดเป็น rst สัญญาณนาฬิกาของข้อมูลอินพุตซึ่งกำหนดเป็น Ld สัญญาณเลือกอัตราการถอดรหัสซึ่งกำหนดเป็น SerelectR สัญญาณอินพุตซึ่งกำหนดเป็น EncIn และสัญญาณควบคุมให้สัมพันธ์กับการทำงานของภาค ACSU ซึ่งมาจากภาค Controller ซึ่งกำหนดเป็น ACSSegment การทดสอบการทำงานจะทำการทดสอบทั้งสองอัตราการถอดรหัส ซึ่งในส่วนแรกทดสอบที่อัตราการถอดรหัสเท่ากับ 1/2 ส่วนหลังทดสอบที่อัตราการถอดรหัส 1/3 โดยการทดสอบต้องกำหนดการเริ่มต้นการทำงานโดยกำหนดให้ rst เป็น “1” โดยมีสัญญาณนาฬิกาข้อมูล (Ld) เป็นสัญญาณบอกการเข้ามาของสัญญาณข้อมูลกำหนดให้เป็น “1” ซึ่งผลการจำลองการทำงานของมอดูล BMU ในการหาค่า Branch metric จากอินพุต “11111111” เอาต์พุตของมอดูลคือค่า Branch Metric (bm0 ถึง bm7) ซึ่งแสดงเป็นเลขฐานสิบหกนั้นแสดงดังรูปที่ 5.4 ซึ่งผลการจำลองการทำงานที่ได้นี้สามารถตรวจสอบได้ว่าการทำงานของวงจรถูกต้อง

5.1.3 มอดูล Add-Compare-Select Unit

มอดูล ACSU เป็นส่วนในการทำงานที่สองต่อจากมอดูล BMU มอดูลนี้ทำหน้าที่หาค่า Path Metric และ Decision Bit ซึ่งมอดูลนี้ประกอบด้วย มอดูล ACS มอดูล Dual-port RAM มอดูล Serial to Parallel (S/P) ซึ่งผลการออกแบบและการจำลองการทำงานมีดังนี้



รูปที่ 5.4 ผลการจำลองการทำงานของมอดูล BMU จากอินพุต “11111111” ที่ R เท่ากับ $1/2$ และ $1/3$

ผลการสังเคราะห์มอดูล Add-Compare-Select Unit

ผลการสังเคราะห์วงจรภาค ACSU โดยโปรแกรม Xilinx ISE จะได้กลุ่มของเน็ตลิสต์ในระดับเกต ซึ่งประกอบไปด้วยอุปกรณ์ดิจิทัลประเภทต่างๆ ซึ่งแสดงดังในตารางที่ 5.5 และเมื่อนำอุปกรณ์เหล่านี้มาติดตั้งในเอฟพีจีเอ รุ่น xc2v1000 ตารางที่ 5.6 แสดงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ และความเร็วการทำงานของมอดูล ACSU โดยตารางประกอบไปด้วย การใช้จำนวน Slices จำนวนฟลิปฟล็อป Slices (Slice Flip Flop) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตเกิดหลังสัญญาณนาฬิกา (Maximum output required time after clock) ส่วนรูปที่ 5.5 แสดงผังวงจรระดับ RTL ของมอดูล ACSU ที่ได้จากการสังเคราะห์ ซึ่งประกอบด้วยผังวงจร ACS และผังวงจร ACSU ทั้งหมด ซึ่งแสดงรายละเอียดการเชื่อมต่อของ วงจร ACS Dual-port RAM Serial to Parallel (S/P) และสัญญาณควบคุมการทำงาน

ผลการจำลองการทำงานของมอดูล Add-Compare-Select

การจำลองการทำงานของมอดูล ACS ด้วยโปรแกรม ModelSim เป็นการป้อนสัญญาณอินพุตทดสอบประกอบด้วยสัญญาณนาฬิกาซึ่งกำหนดเป็น clk สัญญาณรีเซ็ตกำหนดเป็น rst สัญญาณอินพุตค่า Branch Metric กำหนดเป็น bm โดยเอาต์พุตคือ สัญญาณ Update Path Metric ซึ่งกำหนดเป็น Metric และสัญญาณ Decision Bit (Db) ซึ่งผลการทดสอบการทำงานของ ACS

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

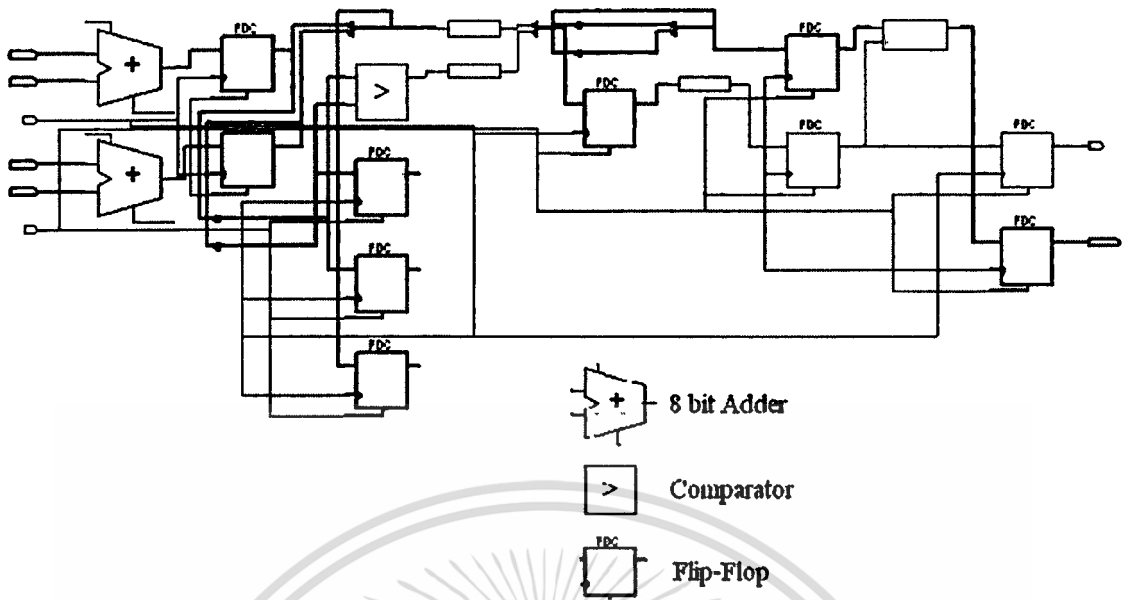
ตารางที่ 5.5 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล ACSU

อุปกรณ์	จำนวน
RAMs	1
64x64-bit dual-port block RAM	1
Registers	48
8-bit register	28
1-bit register	16
32-bit register	1
256-bit register	1
128-bit register	2
Multiplexers	4
2-to-1 multiplexer	4
Adders	8
8-bit adder	8
Comparators	4
7-bit comparator greater	4
XORs	8
1-bit xor2	8

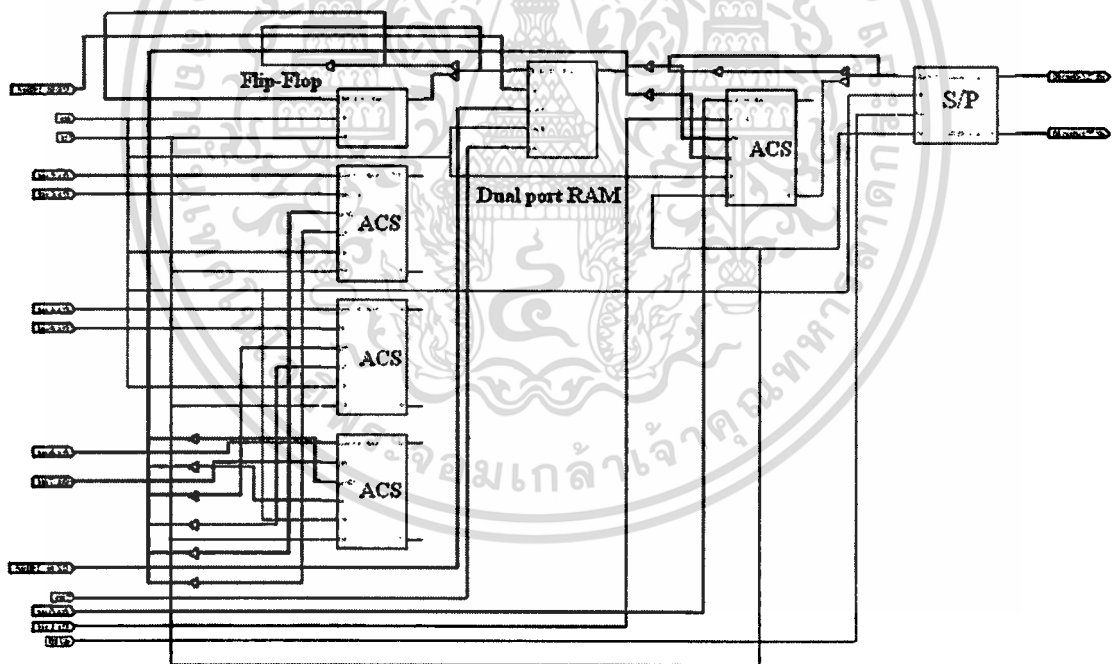
ตารางที่ 5.6 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล ACSU

อุปกรณ์	Slices	Slice Flip Flops	4-input LUTs	Block RAMs
จำนวน	462	784	128	2
Maximum frequency			222 MHz	
Minimum input arrival time before clock			4.828 ns	
Maximum output required time after clock			5.880 ns	

จะทำงานตามสมการการทำงานของ ACS (สมการที่ 4.2) ผลการจำลองการทำงานของมอดูล ACS แสดงดังในรูปที่ 5.6 เนื่องจาก ACS เป็นแบบ 3 Pipelines สัญญาณเอาต์พุตคือ Decision Bit (Db) และ Update Path Metric จึงถูกหน่วงไป 3 สัญญาณนาฬิกา



(ก) วงจร ACS

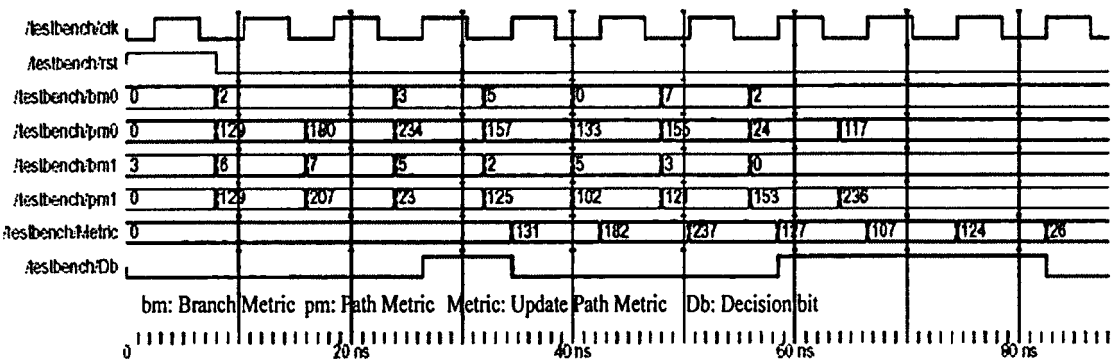


(ข) วงจร ACSU ทั้งหมด

รูปที่ 5.5 ผังวงจรระดับ RTL ของมอดูล ACSU ที่ได้จากการสังเคราะห์

(ก) วงจร ACS (ข) วงจร ACSU ทั้งหมด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.6 ผลการจำลองการทำงานของมอดูล ACS

5.1.4 มอดูล Survivor Memory Unit

มอดูล SMU เป็นส่วนในการทำงานลำดับสุดท้าย ซึ่งมอดูลนี้ทำหน้าที่ถอดรหัสข้อมูลโดยประกอบด้วย มอดูลชิพเรจิสเตอร์ขนาด 64 x 256 และมอดูล 256 มัลติเพล็กซ์ (2:1) ซึ่งผลการออกแบบและการจำลองการทำงานมีดังนี้

ผลการสังเคราะห์มอดูล Survivor Memory Unit

ผลการสังเคราะห์วงจรภาค SMU จะได้อุปกรณ์ดิจิทัลประเภทต่างๆ ซึ่งแสดงดังในตารางที่ 5.7 ในตารางที่ 5.8 แสดงถึงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ และความเร็วการทำงานของมอดูล SMU โดยตารางประกอบไปด้วย การใช้จำนวน Slices จำนวนฟลิปฟล็อป Slices (Slice Flip Flops) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตเกิดหลังสัญญาณนาฬิกา (Maximum output required time after clock) ส่วนรูปที่ 5.7 แสดงผังวงจรระดับ RTL ของมอดูล SMU ที่ได้จากการสังเคราะห์ ซึ่งประกอบด้วย มอดูลชิพเรจิสเตอร์ขนาด 64 x 256 และมอดูล 256 มัลติเพล็กซ์ 2:1 (RE) และสัญญาณควบคุมการทำงาน โดยมอดูลชิพเรจิสเตอร์ขนาด 64 x 256 และมอดูล RE แบ่งออกเป็นเป็นสองมอดูลย่อยที่คล้ายกันเพื่อสะดวกในการออกแบบและจำลองการทำงานเนื่องจากทั้งสองมอดูลที่จำนวนสัญญาณเข้าออกมาก

ผลการจำลองการทำงานของมอดูล Survivor Memory Unit

การจำลองการทำงานของมอดูล SMU เป็นการป้อนสัญญาณอินพุตทดสอบประกอบด้วย สัญญาณนาฬิกา สัญญาณรีเซ็ต สัญญาณอินพุต 256 Decision Bit จาก ACSU ซึ่งกำหนดเป็น Db โดยมีสัญญาณ Init จาก Controller เป็นสัญญาณส่งค่า 256 Decision Bit ให้ SMU ส่วนเอาต์พุตของ SMU เป็นค่าข้อมูลที่ได้จากการถอดรหัสโดยมีสัญญาณ rdy เป็นสัญญาณส่งข้อมูลออกทางเอาต์พุต

ตารางที่ 5.7 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล SMU

อุปกรณ์	จำนวน
Registers	257
1-bit register	257
Shift Registers	256
64-bit shift register	256

ตารางที่ 5.8 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล SMU

อุปกรณ์	Slices	Slice Flip Flops	4-input LUTs	Block RAMs
จำนวน	789	513	1280	0
Maximum frequency			319 MHz	
Minimum input arrival time before clock			3.096 ns	
Maximum output required time after clock			5.880 ns	

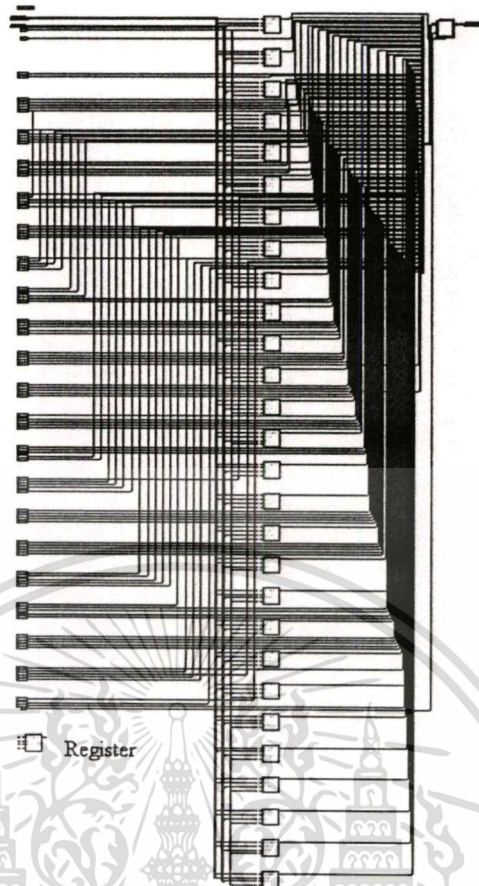
ของ SMU ซึ่งการทำงานทั้งหมดนั้นใช้เวลา 64 สัญญาณนาฬิกา และข้อมูลในการถอดจะถูกต้องก็ต่อเมื่อเวลาผ่านไปช่วงหนึ่ง (Latency) ซึ่งขึ้นอยู่กับค่า Survivor Path Length ของชุดถอดรหัสในที่นี้คือ 64 ผลการจำลองการทำงานของมอดูล SMU แสดงดังในรูปที่ 5.8 ซึ่งเป็นตัวอย่างบางส่วนของผลการจำลองการทำงานซึ่งทดสอบโดยการป้อนค่าให้ Decision Bit

5.1.5 มอดูล Controller

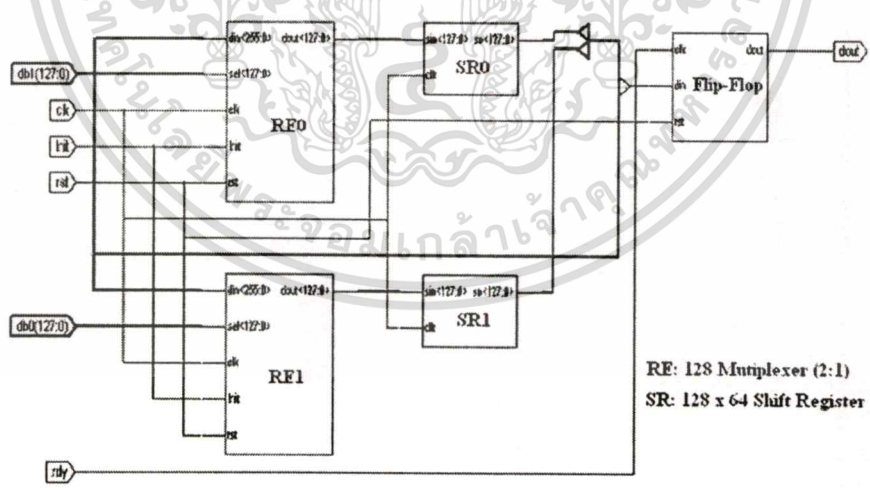
มอดูล Controller ทำหน้าที่ควบคุมการทำงานของมอดูลอื่นๆ ทั้งหมดให้ทำงานสัมพันธ์กันโดยมอดูล Controller ประกอบด้วย 5 มอดูลวงจรนับและเรจิสเตอร์ ซึ่งทำหน้าที่สร้างจังหวะของสัญญาณควบคุมต่างๆ ซึ่งถูกควบคุมให้สัมพันธ์กับสัญญาณอินพุตภายนอกได้แก่ สัญญาณนาฬิกา สัญญาณรีเซ็ต สัญญาณนาฬิกาของข้อมูลอินพุต ผลการออกแบบและการจำลองการทำงานมีดังนี้

ผลการสังเคราะห์มอดูล Controller

ผลการสังเคราะห์วงจรภาค Controller จะได้อุปกรณ์ดิจิทัลประเภทต่างๆ ซึ่งแสดงดังในตารางที่ 5.9 ในตารางที่ 5.10 แสดงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ และความเร็วการทำงานของมอดูล Controller โดยตารางประกอบไปด้วย การใช้จำนวน Slices จำนวนฟลิปฟล็อป Slices



(ก) วงจรมอดูล 256 มัลติเพล็กซ์ 2:1 (RE)



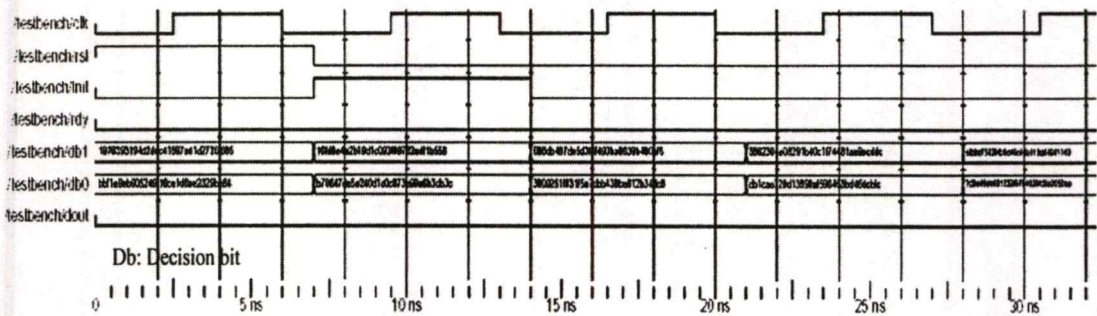
RE: 128 Multiplexer (2:1)
SR: 128 x 64 Shift Register

(ข) วงจร SMU ทั้งหมด

รูปที่ 5.5 ผังวงจรระดับ RTL ของมอดูล SMU ที่ได้จากการสังเคราะห์

(ก) วงจรมอดูล 256 มัลติเพล็กซ์ 2:1 (RE) (ข) วงจร SMU

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.8 ผลการจำลองการทำงานของมอดูล SMU โดยการป้อนค่า Decision Bit (db)

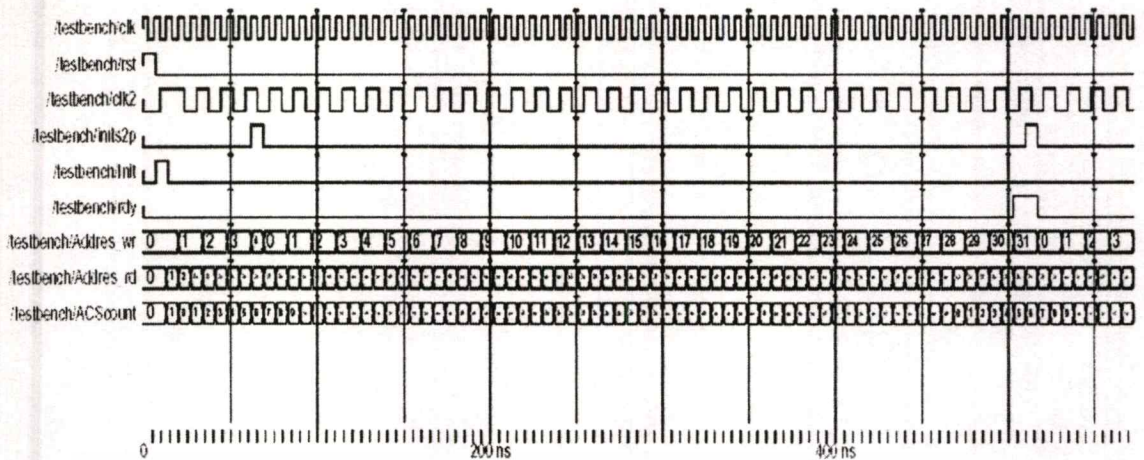
Slices (Slice Flip Flops) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตเกิดหลังสัญญาณนาฬิกา (Maximum output required time after clock)

ตารางที่ 5.9 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล Controller

อุปกรณ์	จำนวน
Registers	7
1-bit register	7
Counters	4
5-bit up counter	1
6-bit up counter	3

ตารางที่ 5.10 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของมอดูล Controller

อุปกรณ์	Slices	Slice Flip Flops	4 input LUTs	Block RAMs
จำนวน	20	30	35	0
Maximum frequency			233 MHz	
Minimum input arrival time before clock			2.588 ns	
Maximum output required time after clock			6.954 ns	



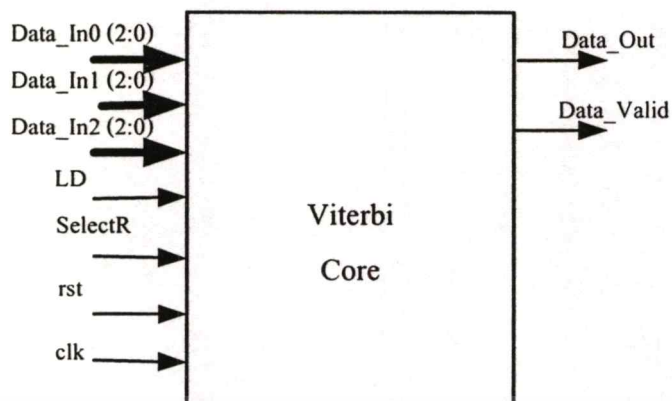
รูปที่ 5.9 ผลการจำลองการทำงานของมอดูล Controller

ผลการจำลองการทำงานของมอดูล Controller

การจำลองการทำงานของมอดูล Controller ด้วยโปรแกรม ModelSim เป็นการป้อนสัญญาณอินพุตทดสอบประกอบด้วยสัญญาณนาฬิกา (clk) สัญญาณรีเซ็ต (rst) สัญญาณนาฬิกาอินพุต (Int) ส่วนสัญญาณเอาต์พุตของ Controller ได้แก่ สัญญาณควบคุม BMU คือ ACSSegment สัญญาณควบคุม Dual Port RAM ใน ACSU คือ Address_wr Address_rd และ clk2 สัญญาณควบคุม S/P และ SMU ได้แก่ Inits2p และ rdy การทำงานทั้งหมดนั้นใช้เวลา 64 สัญญาณนาฬิกา ผลการจำลองการทำงานของมอดูล Controller แสดงดังในรูปที่ 5.9 ซึ่งเป็นผลการจำลองการทำงานของมอดูล Controller ในหนึ่งสัญญาณนาฬิกาของข้อมูลอินพุต

5.1.6 มอดูลชุดถอดรหัส Viterbi

มอดูลที่ออกแบบไว้ทั้งหมดถูกนำมารวมเป็นท้อปมอดูลชุดถอดรหัส โดยประกอบด้วยส่วนสำคัญทั้ง 4 ส่วน คือ มอดูล Branch Metric Unit (BMU) มอดูล Add-Compare-Select Unit (ACSU) มอดูล Survivor Memory Unit (SMU) และมอดูล Controller ส่วนที่เพิ่มเติมประกอบด้วยมอดูล Syn_clk และดีฟลิปฟล็อป มอดูล Syn_clk ทำหน้าที่กำเนิดสัญญาณเริ่มต้น (Init) ในการทำงานของชุดเข้ารหัสในการทำงานในการถอดรหัสในแต่ละบิต โดยสัญญาณเริ่มต้นจะสัมพันธ์กับสัญญาณนาฬิกาของข้อมูล (LD) และสัญญาณนาฬิกา (clk) ตัวดีฟลิปฟล็อปทำหน้าที่ทำให้สัญญาณข้อมูลเอาต์พุตสัมพันธ์กับสัญญาณนาฬิกาของข้อมูล ซึ่งชื่อของสัญญาณของมอดูลชุดถอดรหัสประกอบด้วย สัญญาณอินพุต (Data_in) 9 บิต (3 บิต Soft Decision R เท่ากับ $1/2$ และ $1/3$) สัญญาณนาฬิกาของระบบ (clk) สัญญาณรีเซ็ต (rst) สัญญาณนาฬิกาของข้อมูลอินพุต (LD) สัญญาณเลือกอัตราการถอดรหัส (SelectR) และสัญญาณเอาต์พุต (Data_out) ซึ่งแสดงดังรูปที่ 5.10



รูปที่ 5.10 ชื่อสัญญาณของมอดูลชุดถอดรหัส

ผลการสังเคราะห์มอดูลชุดถอดรหัส

ผลการสังเคราะห์วงจรที่ออกแบบมอดูลโดยโปรแกรม Xilinx ISE จะได้กลุ่มของเน็ตลิสต์ในระดับเกต ซึ่งประกอบไปด้วยอุปกรณ์ดิจิทัลประเภทต่างๆ ซึ่งแสดงดังในตารางที่ 5.11 และเมื่อนำอุปกรณ์เหล่านี้มาติดตั้งในเอฟพีจีเอ รุ่น xc2v1000 ซึ่งตารางที่ 5.12 แสดงจำนวนอุปกรณ์ที่ใช้ในเอฟพีจีเอ และความเร็วการทำงาน โดยตารางประกอบไปด้วย จำนวน Slices จำนวนฟลิปฟล็อป Slices (Slice Flip Flop) จำนวนตัวเปิดตารางสี่อินพุต (4-input LUT) และจำนวนหน่วยความจำ (Block RAM) ค่าความถี่สูงสุด (Maximum Frequency) ค่าเวลาน้อยที่สุดที่สัญญาณอินพุตต้องมาก่อนสัญญาณนาฬิกา (Minimum input arrival time before clock) และค่าเวลามากที่สุดที่สัญญาณเอาต์พุตเกิดหลังสัญญาณนาฬิกา (Maximum output required time after clock) ส่วนรูปที่ 5.11 แสดงถึงผังวงจรระดับ RTL ของที่ออกแบบมอดูลชุดถอดรหัสที่ได้จากการสังเคราะห์ ซึ่งประกอบด้วยมอดูล Branch Metric Unit (BMU) มอดูล Add-Compare-Select Unit (ACSU) มอดูล Survivor Memory Unit (SMU) มอดูล Controller มอดูล Syn_clk และฟลิปฟล็อป ส่วนผังการเชื่อมต่ออุปกรณ์ภายในเอฟพีจีเอ Xilinx Vertex II xc2v1000-6 ที่ถูกใช้แสดงดังรูปที่ 5.12

ผลการจำลองระดับฐานเวลาการทำงานมอดูลชุดถอดรหัส

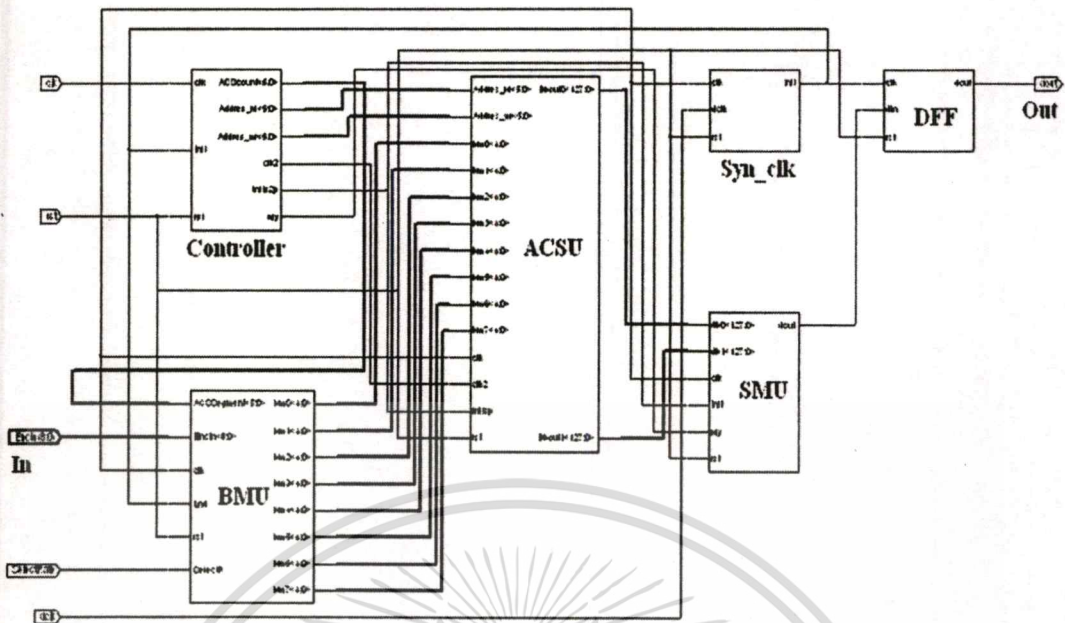
การจำลองการทำงานของมอดูลชุดถอดรหัส เป็นการป้อนสัญญาณข้อมูลอินพุตแบบสุ่ม (Random) เข้าวงจรเข้ารหัสแบบคอนโวลูชันจากนั้นได้นำเอาต์พุตของวงจรนี้ป้อนให้กับชุดถอดรหัส ข้อมูลในการถอดจะถูกต้องก็ต่อเมื่อเวลาผ่านไปช่วงหนึ่ง (Latency) ซึ่งขึ้นอยู่กับค่า Survivor Path Length ของชุดถอดรหัสในที่นี้เท่ากับ 64 โดยเอาต์พุตของชุดถอดรหัสจะถูกเปรียบเทียบกับสัญญาณข้อมูลอินพุตที่ถูกหน่วงเวลาไว้ ซึ่งโปรแกรมทดสอบชุดถอดรหัสทั้ง R เท่ากับ 1/2 และ 1/3 ซึ่งเขียนโดยภาษาเวริล็อกแสดงไว้ในภาคผนวก ข.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.11 จำนวนอุปกรณ์ที่ได้จากการสังเคราะห์ของมอดูล Controller

อุปกรณ์	จำนวน
RAMs	1
64x64-bit dual-port block RAM	1
Registers	363
1-bit register	307
3-bit register	4
5-bit register	16
4-bit register	4
8-bit register	28
32-bit register	1
256-bit register	1
128-bit register	2
Counters	4
5-bit up counter	1
6-bit up counter	3
Shift Registers	256
64-bit shift register	256
Multiplexers	28
2-to-1 multiplexer	20
5-bit 8-to-1 multiplexer	8
Adders	20
4-bit adder carry out	8
3-bit adder carry out	4
8-bit adder	8
Comparators	4
7-bit comparator greater	4
XORs	88
1-bit xor2	48
1-bit xor3	40

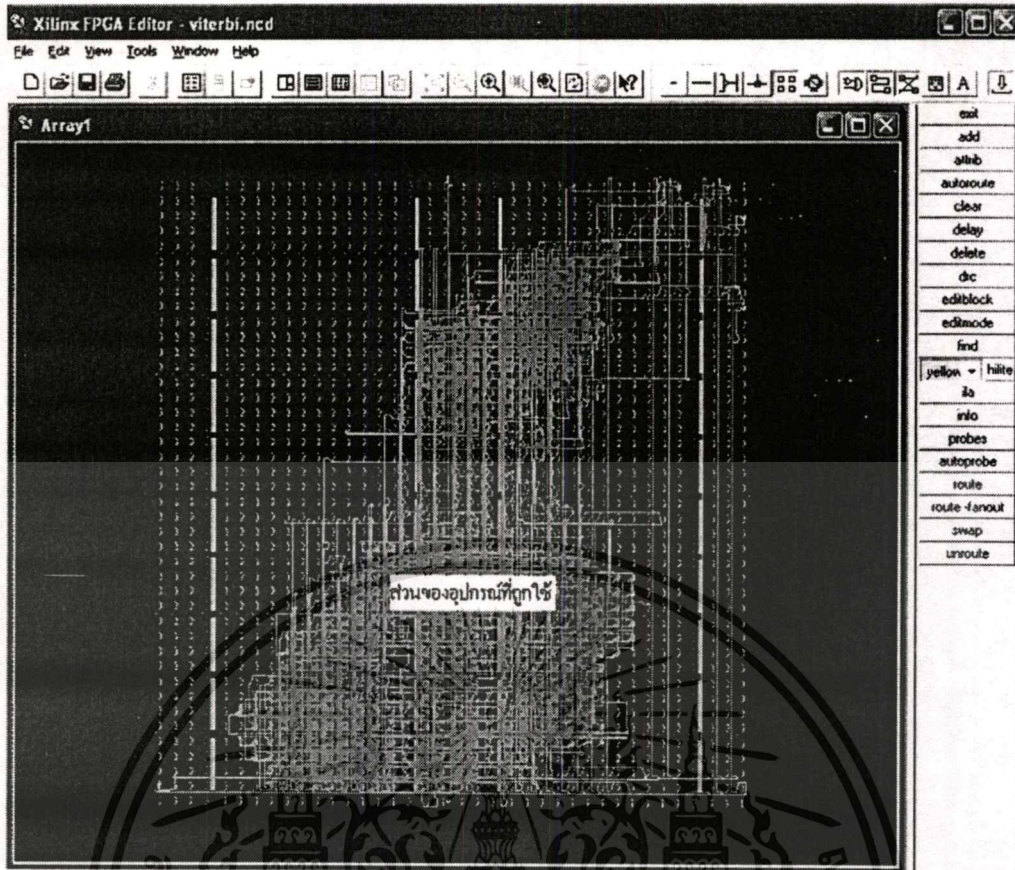


รูปที่ 5.11 พังวงจรของมอดูลชุดถอดรหัสได้จากการสังเคราะห์

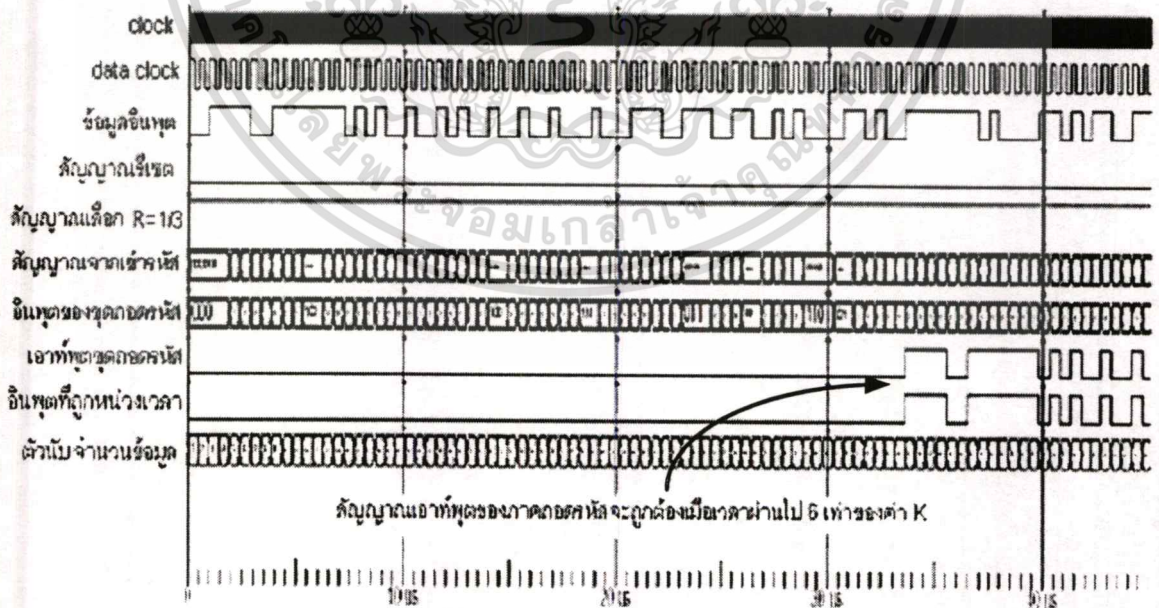
ตารางที่ 5.12 อุปกรณ์ที่ใช้ในเอฟพีจีเอ รุ่น xc2v1000 และความเร็วการทำงานของทีอปมอดูล

อุปกรณ์	Slices	Slice Flip Flops	4-input LUTs	Block RAMs
จำนวน	1416	1460	1663	2
Maximum frequency			222 MHz	
Minimum input arrival time before clock			4.828 ns	
Maximum output required time after clock			5.880 ns	

ซึ่งรูปที่ 5.13 แสดงผลการจำลองการทำงานของมอดูลชุดถอดรหัส Viterbi ด้วยโปรแกรม ModelSim ในระดับฐานเวลา (Timing Simulation)

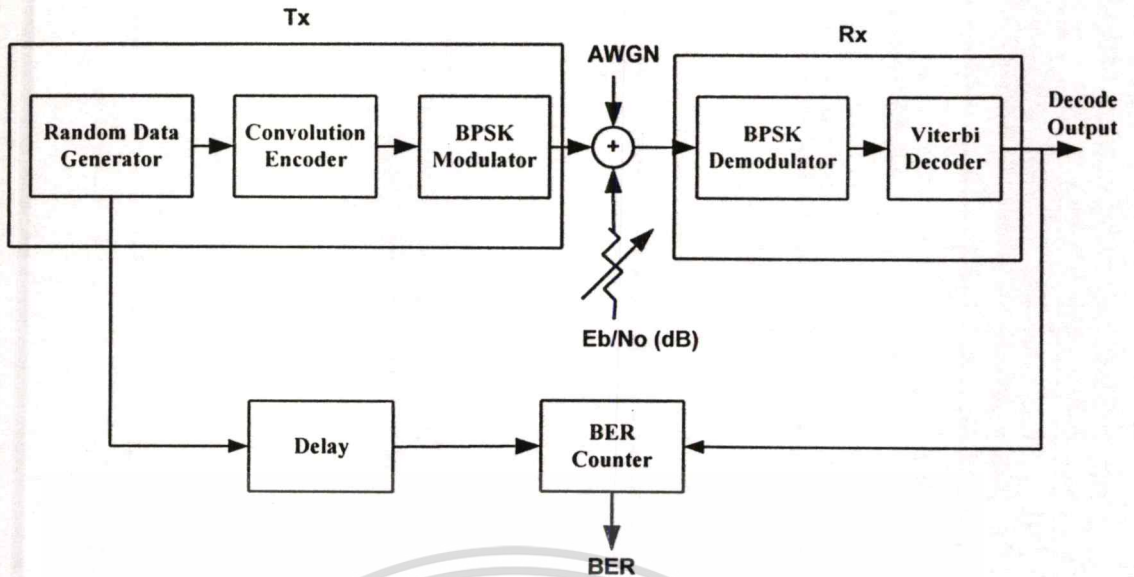


รูปที่ 5.12 ผังการเชื่อมต่ออุปกรณ์ภายในเอฟพีอีเอ Xilinx Vertex II xc2v1000-6 ที่ถูกใช้



รูปที่ 5.13 ผลการจำลองการทำงานระดับฐานเวลาของมอดูลชุดถอดรหัสที่ R เท่ากับ $1/3$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5.14 บล็อกไดอะแกรมการทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณ AWGN

5.2 การทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณแบบ AWGN

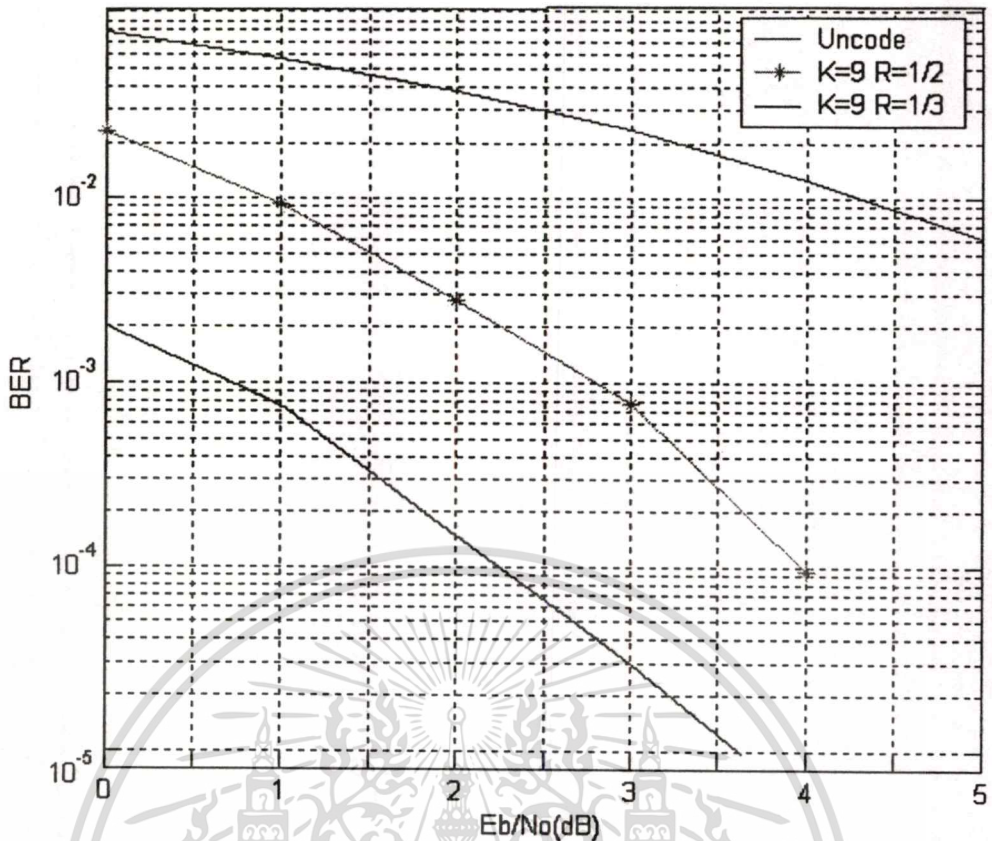
การทดสอบประสิทธิภาพชุดถอดรหัส Viterbi เป็นการทดสอบหาอัตราความผิดพลาดบิต (Bit Error Rate: BER) จากผลกระทบของสัญญาณรบกวนในช่องสัญญาณการสื่อสารแบบ AWGN (Additive White Gaussian Noise) ซึ่งเป็นที่นิยมใช้ในการทดสอบฮาร์ดแวร์ของชุดเข้าและถอดรหัส ป้องกันความผิดพลาดโดยทั่วไป ในการทดสอบนั้นจะเปลี่ยนแปลงค่าของกำลังของสัญญาณต่อสัญญาณรบกวนเป็นค่าต่างๆ เพื่อบันทึกอัตราความผิดพลาดบิตในกรณีที่เข้ารหัสและไม่เข้ารหัส (Uncode) โดยโปรแกรมจำลองการทำงานภาษาเวรีล็อก ขั้นตอนการทดสอบแสดงดังบล็อกไดอะแกรมในรูปที่ 5.14 ซึ่งอธิบายได้ดังนี้

- **Random Data Generator** การสร้างสัญญาณข้อมูลแบบสุ่ม (Random) แบบไบนารี
- **Convolution Encoder** การเข้ารหัสข้อมูลแบบคอนโวลูชัน
- **BPSK Modulator** การมอดูเลตแบบ BPSK (Binary Phase Shift Keying) ซึ่งจะเปลี่ยนบิต “0” ให้มีค่าเป็น 1 และบิต “1” ให้มีค่าเป็น -1 ซึ่งแสดงดังสมการ

$$y = 1 - 2x \quad (5.1)$$

โดย y เป็นสัญญาณที่ได้จากการมอดูเลต

x เป็นสัญญาณที่เข้ามามอดูเลต



รูปที่ 5.15 ผลของอัตราความผิดพลาดบิต (BER) กับค่ากำลังของสัญญาณต่อสัญญาณรบกวน (E_b/N_0) ที่ R เท่ากับ $1/2$ และ $1/3$ เมื่อเทียบกับกรณีที่ไม่มีการเข้ารหัส ในช่องสัญญาณ AWGN

- **AWGN Channel** การสร้างสัญญาณรบกวน Gaussian ซึ่งแสดงดังสมการ [32]

$$n = \sqrt{2\sigma^2 \ln(1/(1-u))} = \sigma \sqrt{2 \ln(1/(1-u))} \quad (5.2)$$

โดย n เป็นค่าสัญญาณรบกวนแบบเกาส์ (Gaussian) ที่ถูกเพิ่มเข้าไปในช่องสัญญาณ (Additive) σ เป็นค่าความแปรปรวน (Variant) ส่วน u เป็นค่าของตัวแปรสุ่มแบบยูนิฟอร์ม (Uniform Random Variable) โปรแกรมทดสอบประสิทธิภาพชุดถอดรหัสในช่องสัญญาณ AWGN โดยภาษาเวรลี้อยู่ในภาคผนวก ข. และรายละเอียดของการจำลองการทำงานในช่องสัญญาณ AWGN อยู่ในภาคผนวก ค.

- **BPSK Demodulator** การจัดระดับสัญญาณที่รับได้ (Quantize) หรือการคิมอดูเลตแบบ BPSK กรณีนี้ชุดถอดรหัสเป็น Soft Decision แบบ 8 ระดับ
- **Viterbi Decoder** การถอดรหัสแบบ Viterbi

5.3 การวิเคราะห์ผลการออกแบบและทดสอบ

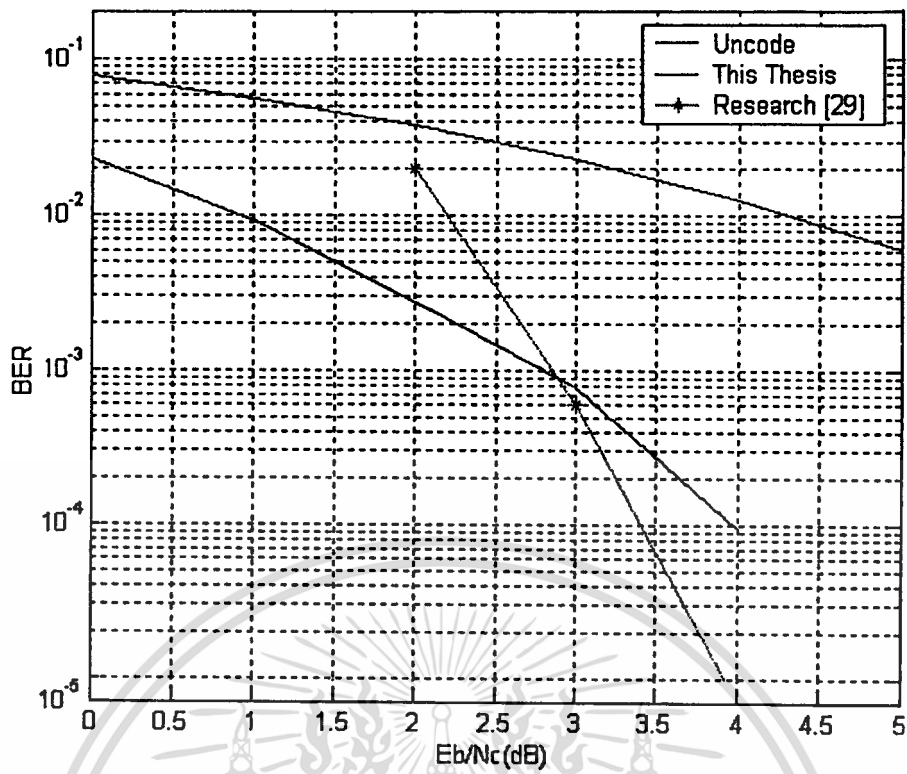
การวิเคราะห์ผลการออกแบบและทดสอบชุดถอดรหัส Viterbi แบบอนุกรมสำหรับระบบ โทรศัพท์เคลื่อนที่ยุคที่ 3 นี้ แบ่งเป็น 3 ส่วนคือ การวิเคราะห์ผลทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณแบบ AWGN การวิเคราะห์ผลการใช้อุปกรณ์ในเอฟพีจีเอ และการวิเคราะห์อัตราเร็วในการถอดรหัส

5.3.1 การวิเคราะห์ผลทดสอบประสิทธิภาพในช่องสัญญาณแบบ AWGN

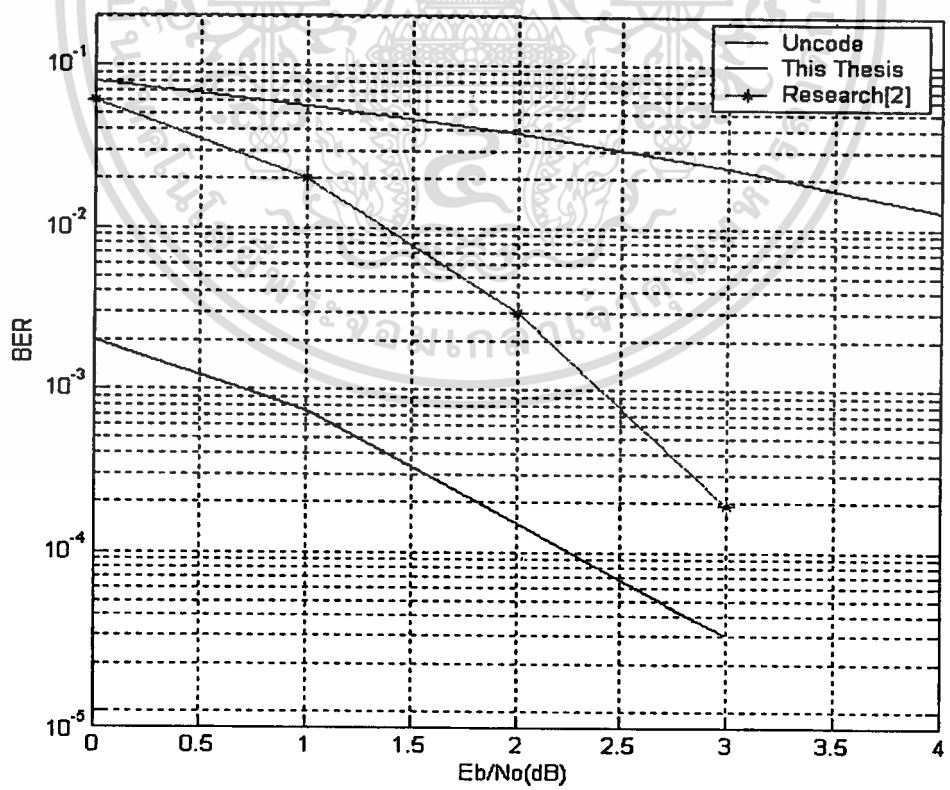
การทดสอบประสิทธิภาพของชุดถอดรหัสในช่องสัญญาณแบบ AWGN แบ่งเป็นอัตราการเข้ารหัสเท่ากับ $1/2$ และ $1/3$ ซึ่งแสดงในรูปที่ 5.15 เมื่อเปรียบเทียบประสิทธิภาพของชุดถอดรหัสกับชุดถอดรหัส Viterbi ในงานวิจัยเรื่อง “Low-Power Viterbi Decoder for CDMA Mobile Terminals” [29] ซึ่งเป็นชุดถอดรหัสที่ใช้โครงสร้างแบบอนุกรม 4 ACS ค่า Constraint Length เท่ากับ 9 อัตราการเข้ารหัสเท่ากับ $1/2$ Generator Polynomials เท่ากับ 561_8 และ 753_8 และ Survivor Path Length เท่ากับ 64 ซึ่งผลของอัตราความผิดพลาดบิต (BER) กับค่ากำลังของสัญญาณต่อสัญญาณรบกวน (E_b/N_0) ของชุดถอดรหัสที่นำเสนอนี้กับงานวิจัย [29] ชุดถอดรหัสที่นำเสนอนี้มีผลของอัตราความผิดพลาดบิตดีกว่าที่ E_b/N_0 น้อยกว่า 3 dB ซึ่งรูปที่ 5.18 แสดงผลของอัตราความผิดพลาดบิตกับค่ากำลังของสัญญาณต่อสัญญาณรบกวนของชุดถอดรหัสกับงานวิจัย [29] และเมื่อเปรียบเทียบประสิทธิภาพของชุดถอดรหัสกับชุดถอดรหัส Viterbi ในงานวิจัยเรื่อง “A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18-/spl mu/m CMOS.” [2] ซึ่งเป็นชุดถอดรหัส Viterbi ที่รวมอยู่กับชุดถอดรหัสแบบเทอร์โบ โดยมีค่า Constraint Length เท่ากับ 9 อัตราการเข้ารหัสเท่ากับ $1/3$ Generator Polynomials เท่ากับ 557_8 , 663_8 , และ 711_8 ซึ่งผลของอัตราความผิดพลาดบิต (BER) กับค่ากำลังของสัญญาณต่อสัญญาณรบกวน (E_b/N_0) ของชุดถอดรหัสที่นำเสนอนี้มีประสิทธิภาพที่ดีกว่างานวิจัย [2] ซึ่งรูปที่ 5.19 แสดงผลของอัตราความผิดพลาดบิตกับค่ากำลังของสัญญาณต่อสัญญาณรบกวนของชุดถอดรหัสกับงานวิจัย [2]

5.3.2 การวิเคราะห์ผลการใช้อุปกรณ์ในเอฟพีจีเอ

ชุดถอดรหัส Viterbi นี้ใช้โครงสร้างแบบอนุกรมซึ่งจะให้โครงสร้างขนาดเล็ก โดยการใช้ อุปกรณ์ในเอฟพีจีเอของชุดถอดรหัสแสดงในตารางที่ 5.12 ซึ่งจำนวนการใช้อุปกรณ์ที่สำคัญ คือ Slices จำนวน 1416 slices และหน่วยความจำ 2 Block RAM และความถี่สูงสุดที่สามารถทำงานได้ (Maximum Frequency) เท่ากับ 222 MHz ตารางที่ 5.13 แสดงผลการเปรียบเทียบคุณสมบัติในการทำงานชุดถอดรหัสกับชุดถอดรหัส Viterbi แบบอนุกรมของบริษัท Xilinx ข้อดีชุดถอดรหัส Viterbi นี้คือ ใช้พื้นที่ในเอฟพีจีเอคือ Area (Slices) และ Block RAM น้อยกว่า และสามารถ



รูปที่ 5.18 ประสิทธิภาพของชุดถอดรหัสที่นำเสนอกับงานวิจัย [29] ที่ R เท่ากับ 1/2



รูปที่ 5.19 ประสิทธิภาพของชุดถอดรหัสที่นำเสนอกับงานวิจัย [2] ที่ R เท่ากับ 1/3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 5.13 การเปรียบเทียบกับชุดถอดรหัสอนุกรมของบริษัท Xilinx ใน FPGA xc2v1000-6 [27]

คุณสมบัติของชุดถอดรหัส	วิทยานิพนธ์นี้	Xilinx Logic Core
Constraint Length	9	9
Coding Rate	1/2, 1/3	1/2
Survivor Path Length	64	64
Soft Decision (Bit)	3	3
Area (Slices)	1416	1775
Block RAM	2	4
Max. Clock Frequency (MHz)	222	172
Max. Data Rate (Mbps)	3.46	15

ถอดรหัสได้ทั้งสองอัตราการเข้ารหัสคือ 1/2 และ 1/3 ถึงแม้ว่าชุดถอดรหัสนี้มีอัตราเร็วในการถอดรหัส (Data Rate) น้อยกว่า แต่ก็เหมาะสมและการทำงานสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ซึ่งมีอัตราเร็วในการรับส่งข้อมูลสูงสุดที่ 2 Mbps ได้

5.3.3 การวิเคราะห์ประสิทธิภาพอัตราเร็วในการถอดรหัส

ชุดถอดรหัส Viterbi นี้สามารถทำงานที่ความถี่สูงสุดเท่ากับ 222 MHz ซึ่งอัตราเร็วในการถอดรหัสสูงสุดเท่ากับ 3.84 Mbps การสร้างชุดถอดรหัสแบบ Viterbi แบบ 4 ACS ทำให้อัตราเร็วในการถอดรหัสสูงกว่างานวิจัย [3] ซึ่งใช้โครงสร้างอนุกรมที่ใช้ ACS ตัวเดียว และวิธีการปรับปรุงภาค SMU แบบวิธี Register Exchange ซึ่งมีการทำงานที่รวดเร็วกว่าวิธี Trace Back ทำให้การทำงานในการถอดรหัสเร็วขึ้นกว่างานวิจัย [26] ซึ่งตารางที่ 5.14 แสดงการเปรียบเทียบกับชุดถอดรหัสแบบอนุกรมของงานวิจัยอื่นๆ ที่มีค่า K เท่ากับ 9

ตารางที่ 5.14 การเปรียบเทียบกับชุดถอดรหัส Viterbi แบบอนุกรมที่มีค่า K เท่ากับ 9 ของงานวิจัยอื่นๆ

คุณสมบัติ	วิทยานิพนธ์นี้	งานวิจัย [3]	งานวิจัย [26]	Xilinx Logic Core [27]	งานวิจัย [29]	งานวิจัย [30]
เทคโนโลยี (Technology)	FPGA Xilinx Virtex II	FPGA Xilinx xc4000	0.8 um CMOS	FPGA Xilinx Virtex II	0.8 um CMOS	FPGA Altera FLEX81500
อัตราการเข้ารหัส (Coding Rate)	1/2, 1/3	1/2	1/3	1/2	1/2	1/2
Decision	3 บิต Soft Decision	3 บิต Soft Decision	4 บิต Soft Decision	3 บิต Soft Decision	3 บิต Soft Decision	4 บิต Soft Decision
ACS	4	2	4	N/A	4	1
SMU	Register Exchange	Trace Back	Trace Back	Trace Back	Trace Back	Trace Back
พื้นที่ (Area)	1416 Slices, 2 Block RAM	N/A	50000 Gate	1775 slices, 4 Block RAM	1914.2 x 3448.6 um	12000 Gate
Maximum Data Rate	3.46 Mbps	19.2 Kbps	9.6 Kbps	15 Mbps	14.4 Kbps	32 Kbps
การใช้งาน (Applications)	3GPP หรืออื่นๆ	N/A	CDMA (IS-95)	3G หรืออื่นๆ	CDMA (IS-95 และ ANSI J-STD-008)	CDMA

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6

สรุปผลและข้อเสนอแนะ

การเข้ารหัสแบบคอนโวลูชันและการถอดรหัสแบบ Viterbi เป็นเทคนิคหนึ่งที่ใช้แก้ไขความผิดพลาดในการรับส่งข้อมูลในระบบสื่อสารดิจิทัล ซึ่งเรียกว่าการป้องกันการผิดพลาดล่วงหน้า (FEC) ซึ่งการเข้ารหัสแบบคอนโวลูชันและการถอดรหัสแบบ Viterbi นี้ได้ถูกนำมาใช้งานในระบบสื่อสารดิจิทัลในปัจจุบันอย่างแพร่หลาย โดยเฉพาะในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ซึ่งเป็นระบบโทรศัพท์เคลื่อนที่ที่สามารถสื่อสารข้อมูลด้วยความเร็วสูง ในมาตรฐานระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ของ 3GPP ได้กำหนดให้ใช้การเข้ารหัสแบบคอนโวลูชันและการถอดรหัสแบบ Viterbi นี้ในช่องสัญญาณเสียง โดยกำหนดอัตราการรับส่งข้อมูลไว้สูงสุดที่ 2 Mbps

6.1 สรุปผลการวิจัย

การสร้างชุดถอดรหัส Viterbi ซึ่งเป็นอุปกรณ์ที่ใช้ถอดรหัสคอนโวลูชันในช่องสัญญาณเสียงของระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 นั้น ในโครงสร้างทาง VLSI โครงสร้างแบบขนานจะใช้พื้นที่ในการสร้างขนาดใหญ่กว่าแบบอนุกรม โดยเฉพาะชุดถอดรหัสที่ใช้ถอดรหัสคอนโวลูชันที่มีจำนวนสเตต (State) ของการทำงานที่สูง เช่น 256 สเตต ในมาตรฐานของ 3GPP การสร้างในแบบขนานต้องใช้อุปกรณ์จำนวนมาก จากปัญหาดังกล่าวในวิทยานิพนธ์นี้ได้นำเสนอถึงวิธีการในการออกแบบสร้างชุดถอดรหัส Viterbi สำหรับถอดรหัสคอนโวลูชันที่กำหนดในมาตรฐาน TS 25.212 ของ 3GPP ที่มีโครงสร้างขนาดเล็กใช้อุปกรณ์ในการสร้างจำนวนน้อย โดยในการออกแบบชุดถอดรหัสนี้ใช้โครงสร้างอนุกรมแบบ 4 Add-Compare-Select (ACS) ซึ่งเป็นจำนวนของ ACS ที่ใช้อุปกรณ์ในการสร้างจำนวนน้อยแต่ให้ความเร็วในการถอดรหัสสูง และใน Survivor Memory Unit (SMU) ได้ปรับปรุงวิธีการทำงานแบบ Register Exchange ที่มีความเร็วในการทำงานสูงแต่มีโครงสร้างขนาดใหญ่ให้มีขนาดเล็กลงและให้เหมาะสมกับโครงสร้างแบบ 4 ACS ซึ่งชุดถอดรหัสนี้มีโครงสร้างที่ไม่ซับซ้อนและใช้อุปกรณ์ในการสร้างจำนวนน้อย ดังนั้นการสร้างสามารถทำได้ง่ายและมีความเร็วในการทำงานสามารถใช้งานกับระบบ 3GPP ได้ ซึ่งข้อดีของชุดถอดรหัส Viterbi แบบอนุกรมที่ใช้ 4 ACS ที่มีการปรับปรุง Register Exchange คือ โครงสร้างทั้งสองภาคนั้นเหมาะสมกันเนื่องจากใช้เวลาในการทำงานเท่ากัน ทำให้สามารถออกแบบให้ทั้งสองภาคทำงานพร้อมกัน ทำให้การทำงานของชุดถอดรหัสมีความรวดเร็ว และการปรับปรุง Register Exchange ทำให้ขนาดของวงจรลดลงกว่าชุดถอดรหัสแบบอนุกรม ที่ใช้ภาค Survivor Memory Unit แบบ Trace Back เนื่องจากใช้หน่วยความจำน้อยกว่าและใช้หน่วยความจำ LIFO ในการจัดเรียงข้อมูลเอาต์พุตให้ถูกต้อง

ต้นแบบชุดถอดรหัส Viterbi นี้ถูกออกแบบโดยใช้ภาษาเวริล็อก (Verilog HDL) สำหรับติดตั้งในเอฟพีจีเอ Xilinx Vertex II ผลการออกแบบและทดสอบชุดถอดรหัสนี้ใช้อุปกรณ์ในเอฟพีจีเอคือ Slices จำนวน 1,416 Slices และหน่วยความจำ (Block RAM) จำนวน 2 Block RAM ความถี่สูงสุดที่สามารถทำงานได้ (Maximum Frequency) เท่ากับ 222 MHz ซึ่งจะให้อัตราเร็วในการถอดรหัสนั้นสูงถึง 3.46 Mbps ซึ่งเมื่อเปรียบเทียบกับชุดถอดรหัส Viterbi แบบอนุกรมของบริษัท Xilinx มีข้อดีคือ ใช้พื้นที่ในเอฟพีจีเอ คือใช้จำนวน Slices และ Block RAM น้อยกว่า และสามารถถอดรหัสได้ทั้งสองอัตราการเข้ารหัสคือ 1/2 และ 1/3 ถึงแม้ว่าชุดถอดรหัสนี้มีอัตราเร็วในการถอดรหัสน้อยกว่า แต่ก็เหมาะสมและสามารถใช้งานในระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ซึ่งกำหนดอัตราการรับส่งข้อมูลไว้สูงสุดที่ 2 Mbps ได้ ซึ่งผลการวิจัยที่ได้นี้ได้สำเร็จตามวัตถุประสงค์และขอบเขตของงานวิจัยที่ตั้งไว้

6.2 ปัญหาและอุปสรรคที่พบในงานวิจัย

- การทดสอบประสิทธิภาพชุดเข้ารหัสแบบคอนโวลูชันและการถอดรหัส Viterbi ในระบบสื่อสารในช่องสัญญาณแบบ AWGN ซึ่งเป็นการทดสอบถึงผลกระทบของสัญญาณรบกวนต่ออัตราการเกิดความผิดพลาดบิต (BER) ในการรับส่งข้อมูล ซึ่งชุดเข้ารหัสแบบคอนโวลูชันและการถอดรหัสแบบ Viterbi ที่มี Constraint length เท่ากับ 9 (256 สเตท) มีประสิทธิภาพสูง การทดสอบหาค่าอัตราการเกิดความผิดพลาดจึงต้องใช้เวลาานาน

- โปรแกรมทดสอบประสิทธิภาพของชุดเข้ารหัสคอนโวลูชันและการถอดรหัส Viterbi ในช่องสัญญาณแบบ AWGN นั้นเขียนด้วยภาษาเวริล็อก แต่ในโครงสร้างภาษาเวริล็อกนั้นไม่มีฟังก์ชันตรีโกณมิติและลอการิทึมทำให้ต้องใช้ฟังก์ชันนี้จากงานวิจัย [30] ซึ่งการทดสอบประสิทธิภาพต้องใช้เวลาในการทำงานเพิ่มขึ้น และเกิดความคลาดเคลื่อนเล็กน้อย

6.3 ข้อเสนอแนะในการพัฒนา

- ในระบบ 3GPP มีการส่งรับข้อมูลหลายอัตรา ซึ่งชุดถอดรหัส Viterbi นี้สามารถเปลี่ยนอัตราเร็วในการส่งรับข้อมูล (Adaptive Data Rate) ได้โดยการเปลี่ยนความถี่สัญญาณพาหะของชุดถอดรหัส การพัฒนาให้ชุดถอดรหัสสามารถเปลี่ยนอัตราเร็วในการส่งข้อมูลโดยการเปลี่ยนเพียงความถี่สัญญาณพาหะของสัญญาณข้อมูล โดยความถี่สัญญาณพาหะของชุดถอดรหัสคงที่ จะทำให้อุปกรณ์ชุดถอดรหัสสามารถใช้งานกับอุปกรณ์อื่นๆ ของระบบได้สะดวกขึ้น

- ในส่วน Path Metric Memory (PMM) ของภาค ACSU ในงานวิจัยนี้ใช้หน่วยความจำแบบ Dual-port RAM ในการทำงาน การนำชิพเรจิสเตอร์มาออกแบบใช้งานแทน Dual-port RAM จะทำให้ความเร็วในการทำงานของชุดถอดรหัสเพิ่มขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- อุปกรณ์ป้องกันความผิดพลาดล่วงหน้า (FEC) สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ประกอบด้วยรหัสคอนวอลูชันและรหัสเทอร์โบ ดังนั้นสามารถนำงานวิจัยนี้ไปรวมกับอุปกรณ์ชุดรหัสเทอร์โบ ทำให้เป็นอุปกรณ์ป้องกันความผิดพลาดล่วงหน้าสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ที่สมบูรณ์ และอาจมีบางมอดูลสามารถทำงานร่วมกันได้ ซึ่งจะส่งผลให้ใช้อุปกรณ์ในการสร้างลดลง

- การนำเอามอดูลชุดเข้าและถอดรหัสนี้ไปรวมกับงานทาง Baseband หรือวงจรอื่นๆ ของการวิจัยระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 หรืออื่นๆ ที่เกี่ยวข้อง จะทำให้การพัฒนาระบบนั้นเป็นไปอย่างรวดเร็วและมีประสิทธิภาพมากยิ่งขึ้น



เอกสารอ้างอิง

- [1] 3rd Generation Partnership Project (3GPP). **Technical Specification Group (TSG) Multiplexing and channel coding (FDD)**. TS 25.212 v2.3.0, 1999.
- [2] Bickerstaff M.A., Garrett D., Prokop T., Thomas C., Widdup B., Gongyu Zhou, Davis L.M., Woodward G., Nicol C., Ran-Hong Yan. "A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18-/spl mu/m CMOS." *IEEE Journal of Solid-State Circuits*, Vol. 37, 2002. pp 1555-1564.
- [3] Bupesh Pandita, Subir K Roy. "Design and implementation of a Viterbi decoder using FPGAs." *IEEE VLSI Design Twelfth International Conference*, 1999. pp. 611-614.
- [4] Bernard Sklar. **Digital Communications Fundamentals and Applications**. New Jersey: Prentice Hall, Inc. 1988.
- [5] เกียรติศักดิ์ ศรีพิมานวัฒน์. "การแก้ไขความผิดพลาดของการสื่อสารร่วมสัญญาณเสียงและข้อมูล." วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2536.
- [6] Hideki Imai. **Essentials of Error-Control Coding Techniques**. Academic Press, Inc. 1990.
- [7] Kanu Chadha. "A Reconfigurable Decoding Architecture for Wireless LAN and Cellular Systems." Sc.M. Thesis of RICE University. 2001
- [8] HL Lou. "Implementing the Viterbi Algorithm." *IEEE Signal Processing Magazine*, Vol. 15, 1995. pp 42-52.
- [9] ไพโรจน์ ไววานิชกิจ. **เปิดโลกมือถือ GSM สู่ UMTS**. กรุงเทพมหานคร: บริษัทซีเอ็ดดูเคชั่น, 2545.
- [10] วรณารถ แสงฉาย. "การกำเนิดสัญญาณ PWM โดยใช้ FPGA." วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2544.
- [11] วัชรกร หนูทอง. "การออกแบบไมโครคอนโทรลเลอร์โดยใช้เอฟพีจีเอ." วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2546.
- [12] Xilinx Inc. "Programmable Logic Design Quick Start Hand Book." [Online]. Available: <http://www.xilinx.com> 2002.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- [13] Xilinx Inc. "Virtex™-II Platform FPGAs: Detailed Description." [Online]. Available: <http://www.xilinx.com> 2002.
- [14] วิศวกรรม หนุทอง. **FPGA Design Workshop**. ศูนย์พัฒนาธุรกิจออกแบบวงจรรวม (TIDI) ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC). 2545
- [15] Xilinx Inc. "Xilinx ISE5.1 Software Manuals." [Online]. Available: <http://www.xilinx.com> 2002.
- [16] Daniel C. Hyde. **Handbook on Verilog HDL**. Computer Science Department Bucknell University. 1997.
- [17] Douglas J Smith. **HDL Chip Design**. Madison, AL, USA: Doone Publication. 1999.
- [18] Sayed Aon Mujtaba. "An area-efficient VLSI architecture of the Viterbi decoder for reverse link IS-95 (CDMA) air interface." *IEEE Signal Processing Proceedings*. ICSP '98, 1998. pp. 525-528.
- [19] Gennady Feygin, P.G. Gulak. "Architectural Tradeoff for Survivor Sequence Memory in Viterbi Decoder." *IEEE Transactions on Communications*, Vol. 41, pp. 425-429, 1993.
- [20] Syed Shahzad Shah, Saqib Yaqub, Eaisal Suleman. "Self-correcting codes conquer noise part one: Viterbi codecs." [Online]. Available: <http://www.ednmag.com> 2001.
- [21] Hiroshi Suzuki, Yun-Nan Chang, Keshab K. Parhi. "Low-power Bit-Serial Viterbi decoder for 3rd Generation W-CDMA Systems." *IEEE 1999 Custom Integrated Circuits Conference*, 1999. pp 589-592.
- [22] Chadha K., Cavallaro J.R. "A reconfigurable viterbi decoder architecture." *IEEE Systems and Computers Conference Record of the Thirty-Fifth Asilomar Conference*, 2001. pp 66-71.
- [23] S.B. Wicker. **Error Control System for Digital Communication and Storage**. New Jersey: Prentice Hall, 1995.
- [24] El-Dib, D.A.F., Elmasry, M.I. "Low-power register-exchange Viterbi decoder for high-speed wireless communications." *Circuits and Systems, IEEE International Symposium*, 2002.
- [25] Jung Chul Suh, Byonghyo ShimGlobal. "Pipelined VLSI architecture of the Viterbi decoder for IMT-2000" *IEEE Telecommunications Conference, GLOBECOM '99*, 1999.
- [26] Jun Jin Kong, Yong Woo Park, Min-Goo Kim, Youngky Kim., Yeun-Cheul Jeung. "Fast serial Viterbi decoder ASIC for CDMA cellular base station" *TENCON '96. Proceedings., 1996 IEEE TENCON, Digital Signal Processing Applications*, 1996

- [27] Xilinx Inc. "LogiCore Viterbi decoder V 2.0." [Online]. Available: <http://www.xilinx.com> 2002.
- [28] Shung C.B., Siegel P.H., Ungerboeck G., Thapar H.K.. "VLSI architectures for metric normalization in the Viterbi algorithm." *IEEE SUPERCOMM/ICC '90 Conference*, 1990. pp. 1723-1728.
- [29] Inyup Kang, Alan N. Willson Jr. "Low-Power Viterbi Decoder for CDMA Mobile Terminals" *IEEE Journal of Solid-State Circuits*, Vol. 33, MARCH 1998. pp. 473-482
- [30] Jang-Hyun Park, Yea-Chul Rho. "Performance test of Viterbi decoder for wideband CDMA system" *Design Automation Conference 1997. Proceedings of the ASP-DAC '97*, 1997. pp. 19 -23.
- [31] M. G. Arnold, C. Walter, F. Engineer, "Verilog Transcendental Functions for Numerical Testbenches" *Proceedings of the Tenth International HDL conference*, Santa Clara, California, March 1, 2001.
- [32] Chip Fleming. "A Tutorial on Convolutional Coding with Viterbi Decoding" *Spectrum Applications* [Online]. Available: <http://pw1.netcom.com/~chip.f/viterbi/tutorial.html> 2003.



ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก.

โปรแกรมชุดถอดรหัส

//โปรแกรมชุดเข้ารหัสที่ $R=1/2$

```
module EncoderR12(clk,rst,data_in,enc_out);
```

```
    input clk;
```

```
    input rst;
```

```
    input data_in;
```

```
    output[1:0] enc_out;
```

```
    reg[1:0] enc_out;
```

```
    reg[8:0] D;
```

```
    always @(posedge clk or posedge rst)
```

```
    begin
```

```
        if (rst)
```

```
        begin
```

```
            D <= 0;
```

```
            enc_out <= 0;
```

```
        end
```

```
    else
```

```
        begin
```

```
            D <= {data_in,D[8:1]};
```

```
            enc_out[0] <= D[8] ^ D[6] ^ D[5] ^ D[4] ^ D[0]; //561
```

```
            enc_out[1] <= D[8] ^ D[7] ^ D[6] ^ D[5] ^ D[3] ^ D[1] ^ D[0]; //753
```

```
        end
```

```
    end
```

```
endmodule
```

//โปรแกรมชุดเข้ารหัสที่ $R=1/3$

```
module EncoderR13(clk,rst,data_in,enc_out);
```

```
    input clk;
```

```
    input rst;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

input data_in;
output[2:0] enc_out;
reg[2:0] enc_out;
reg[8:0] D;
always @(posedge clk or posedge rst)
begin
    if (rst)
        begin
            D <=0;
            enc_out <=0;
        end
    else
        begin
            D<= {data_in,D[8:1]};
            enc_out[0] <= D[8] ^ D[6] ^ D[5] ^ D[3] ^ D[2] ^ D[1] ^ D[0]; //557
            enc_out[1] <= D[8] ^ D[7] ^ D[5] ^ D[4] ^ D[1] ^ D[0];
            //663
            enc_out[2] <= D[8] ^ D[7] ^ D[6] ^ D[3] ^ D[0]; //711
        end
    end
end
endmodule

```

//โปรแกรมชุดถอดรหัส

//ท้อปโมดูลชุดถอดรหัส

```
module viterbi(clk,dclk,rst,SelectRate,EncIn,dout);
```

```
    input clk,dclk,rst,SelectRate;
```

```
    input[8:0] EncIn;
```

```
    output dout;
```

```
    wire Init,initsp,clk2,rdout,rdy;
```

```
    wire[4:0] bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0;
```

```
    wire [5:0] ACSSegment,Address_rd,Address_wr;
```

```
    wire[127:0] Dbout1,Dbout0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

syndclk syndclk(clk,rst,dclk,Init);

bmg BMU(clk,rst,Init,SelectRate,EncIn,ACSSegment,bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0);

acsu ACSU(clk,clk2,rst,initsp,bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0,Address_wr,Address_rd,
Dbout1,Dbout0);

controller CONTROLLER(clk,Init,rst,ACSSegment,Address_rd,Address_wr,initsp,rdy,clk2);

smu smu(clk,rst,initsp,rdy,Dbout1,Dbout0,rdout);

dff Lacthdata(Init,rst,rdout,dout);

endmodule

```

//โมดูล Synchronizer Clock

```

module syndclk(clk,rst,dclk,Init);
    input clk,rst;
    input dclk;
    output Init;
    reg R,Init;
always @(posedge clk or posedge rst)
begin
    if (rst)
    begin
        R <= 0;
        Init <= 0;
    end
    else
        begin
            if (dclk)
            begin
                R <= dclk;
                Init <= ~R;
            end
        end
    end
end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        R <= 0;

        Init <= 0;

        end

    end

end

endmodule

```

//ท็อปปโมดูล BMU

```

module bmg(clk,rst,Lnd,SelectR,EncIn,ACSSegment,bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0);

    input clk,rst,Lnd,SelectR;

    input[5:0] ACSSegment;

    input[8:0] EncIn;

    output[4:0] bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0;

    wire[5:0] ACSSegment;

    wire[8:0] BranchID7,BranchID6,BranchID5,BranchID4,BranchID3,BranchID2,BranchID1,
        BranchID0;

    wire[4:0] y7,y6,y5,y4,y3,y2,y1,y0;

    wire[2:0] EncOut7,EncOut6,EncOut5,EncOut4,EncOut3,EncOut2,EncOut1,EncOut0;

    assign BranchID0 = {ACSSegment,3'b000};
    assign BranchID1 = {ACSSegment,3'b001};
    assign BranchID2 = {ACSSegment,3'b010};
    assign BranchID3 = {ACSSegment,3'b011};
    assign BranchID4 = {ACSSegment,3'b100};
    assign BranchID5 = {ACSSegment,3'b101};
    assign BranchID6 = {ACSSegment,3'b110};
    assign BranchID7 = {ACSSegment,3'b111};

    Adders Adders(clk,rst,Lnd,SelectR,EncIn,y7,y6,y5,y4,y3,y2,y1,y0);

    encode ENC0(clk,rst,SelectR,BranchID0, EncOut0);
    encode ENC1(clk,rst,SelectR,BranchID1, EncOut1);
    encode ENC2(clk,rst,SelectR,BranchID2, EncOut2);
    encode ENC3(clk,rst,SelectR,BranchID3, EncOut3);
    encode ENC4(clk,rst,SelectR,BranchID4, EncOut4);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

encode ENC5(clk,rst,SelectR,BranchID5, EncOut5);
encode ENC6(clk,rst,SelectR,BranchID6, EncOut6);
encode ENC7(clk,rst,SelectR,BranchID7, EncOut7);

mux8_1 mux0(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut0,bm0);
mux8_1 mux1(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut1,bm1);
mux8_1 mux2(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut2,bm2);
mux8_1 mux3(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut3,bm3);
mux8_1 mux4(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut4,bm4);
mux8_1 mux5(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut5,bm5);
mux8_1 mux6(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut6,bm6);
mux8_1 mux7(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,EncOut7,bm7);

endmodule

```

//โมดูล Adders

```

module Adders(clk,rst,Lnd,SelectR,EncIn,y7,y6,y5,y4,y3,y2,y1,y0);
    input clk,rst,Lnd,SelectR;
    input[8:0] EncIn;
    output[4:0] y7,y6,y5,y4,y3,y2,y1,y0;
    wire[2:0] in2,in1,in0,nin1,nin0,nrin2;
    reg[4:0] y7,y6,y5,y4,y3,y2,y1,y0;
    reg[3:0] ry3,ry2,ry1,ry0;
    reg[2:0] rin2;

    dff3 latch0(Lnd,rst,EncIn[2:0],in0);
    dff3 latch1(Lnd,rst,EncIn[5:3],in1);
    dffe3 latch2(Lnd,rst,SelectR,EncIn[8:6],in2);

    assign nin0 = ~ in0;
    assign nin1 = ~ in1;
    assign nrin2 = ~rin2;

    always @(posedge clk or posedge rst)
    begin
        if (rst)
            begin
                y0 <= 0;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

y1 <= 0;
y2 <= 0;
y3 <= 0;
y4 <= 0;
y5 <= 0;
y6 <= 0;
y7 <= 0;
ry0 <= 0;
ry1 <= 0;
ry2 <= 0;
ry3 <= 0;
rin2 <= 0;
end
else
begin
rin2 <= in2;
ry0 <= in1 + in0;
ry1 <= in1 + nin0;
ry2 <= nin1 + in0;
ry3 <= nin1 + nin0;
y0 <= rin2 + ry0;
y1 <= rin2 + ry1;
y2 <= rin2 + ry2;
y3 <= rin2 + ry3;
y4 <= nrin2 + ry0;
y5 <= nrin2 + ry1;
y6 <= nrin2 + ry2;
y7 <= nrin2 + ry3;
end
end
endmodule

```

endmodule

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

//โมดูล 1 bit D Flip Flop

```
module dff(clk,rst,din,dout);
    input  clk,rst;
    input  din;
    output dout;
    reg    dout;

    always @(posedge clk or posedge rst )
    begin
        if (rst)
            dout <= 0;
        else
            dout <= din;
    end
endmodule
```

//โมดูล 3 bit D Flip Flop

```
module dff3(clk,rst,din,dout);
    input  clk,rst;
    input[2:0] din;
    output[2:0] dout;
    reg[2:0] dout;

    always @(posedge clk or posedge rst )
    begin
        if (rst)
            dout <= 0;
        else
            dout <= din;
    end
endmodule
```

//โมดูล 3 bit D Flip Flop with enable

```
module dffe3(clk,rst,ce,din,dout);
    input  clk,rst,ce;
    input[2:0] din;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

output[2:0] dout;

reg[2:0] dout;

always @(posedge clk or posedge rst )
begin
    if (rst)

        dout <= 0;

    else

        if (ce)

            dout <= din;

end

endmodule

```

//โมดูล Encoder ใน BMU

```

module encode(clk,rst,SelectR,BranchID,EncOut);

```

```

    input clk,rst,SelectR;

```

```

    input[8:0] BranchID;

```

```

    output[2:0] EncOut;

```

```

    reg[2:0] EncOut;

```

```

    always @(posedge clk or posedge rst)

```

```

begin

```

```

    if (rst)

```

```

        begin

```

```

            EncOut[0] <= 0;

```

```

            EncOut[1] <= 0;

```

```

            EncOut[2] <= 0;

```

```

        end

```

```

    else

```

```

        if (SelectR)

```

```

            begin //R=1/3

```

```

            EncOut[0] <= BranchID[8] ^ BranchID[6] ^ BranchID[5] ^ BranchID[3] ^ BranchID[2] ^

```

```

                BranchID[1] ^ BranchID[0];    //557

```

```

            EncOut[1] <= BranchID[8] ^ BranchID[7] ^ BranchID[5] ^ BranchID[4] ^ BranchID[1] ^

```

```

                BranchID[0]; //663

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

EncOut[2] <= BranchID[8] ^ BranchID[7] ^ BranchID[6] ^ BranchID[3] ^ BranchID[0];

//711

end

else // R= 1/2

begin

EncOut[0] <= BranchID[8] ^ BranchID[6] ^ BranchID[5] ^ BranchID[4] ^ BranchID[0]; //561

EncOut[1] <= BranchID[8] ^ BranchID[7] ^ BranchID[6] ^ BranchID[5] ^ BranchID[3] ^

BranchID[1] ^ BranchID[0];//753

EncOut[2] <= 0;

end

end

endmodule

//โมดูล Multiplexer (8:1)

module mux8_1(clk,rst,y7,y6,y5,y4,y3,y2,y1,y0,sel,bm);

input clk,rst;

input[4:0] y7,y6,y5,y4,y3,y2,y1,y0;

input[2:0] sel;

output[4:0] bm;

reg[4:0] bm;

always @(posedge clk or posedge rst )

begin

if (rst)

bm <= 0;

else

begin

case (sel)

3'b000 : bm <= y0;

3'b001 : bm <= y1;

3'b010 : bm <= y2;

3'b011 : bm <= y3;

3'b100 : bm <= y4;

3'b101 : bm <= y5;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

3'b110 : bm <= y6;
3'b111 : bm <= y7;
default : bm <= 5'b0;
endcase
end
end
endmodule

```

//ท็อปปโมดูล ACSU

```

module acsu(clk,clk2,rst,initsp,bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0,Address_wr,
            Address_rd,Dbout1,Dbout0);
    input  clk,clk2,rst,initsp;
    input[4:0]  bm7,bm6,bm5,bm4,bm3,bm2,bm1,bm0;
    input[5:0]  Address_wr,Address_rd;
    output[127:0] Dbout1,Dbout0;
    wire[7:0]    Metric3,Metric2,Metric1,Metric0;
    wire[7:0]    PM7,PM6,PM5,PM4,PM3,PM2,PM1,PM0;
    wire[31:0]    SMetric,Metric_out;
    wire[63:0]    PM,Metric;
    wire[3:0]     Db;

    assign SMetric={Metric3,Metric2,Metric1,Metric0};
    assign Metric = {SMetric,Metric_out};
    assign {PM7,PM6,PM5,PM4,PM3,PM2,PM1,PM0} = PM;
    acs ACS0(clk,rst,bm1,bm0,PM1,PM0,Db[0],Metric0);
    acs ACS1(clk,rst,bm3,bm2,PM3,PM2,Db[1],Metric1);
    acs ACS2(clk,rst,bm5,bm4,PM5,PM4,Db[2],Metric2);
    acs ACS3(clk,rst,bm7,bm6,PM7,PM6,Db[3],Metric3);
    dff32 Lacth(clk,rst,SMetric,Metric_out);
    dpam64x64 DualPortRAM(clk2,clk,Address_wr,Address_rd,Metric,PM);
    S2P S2P(clk,rst,initsp,Db,Dbout1,Dbout0);
Endmodule

```

//โมดูล ACS

```

module acs(clk,rst,bm1,bm0,pm1,pm0,Db,Metric);

    input clk,rst;

    input[4:0] bm1,bm0;

    input[7:0] pm1,pm0;

    output Db;

    output[7:0] Metric;

    reg [7:0] sum1,sum0,Rsum3,Rsum2,Rsum1,Rsum0,Metric;

    reg RDb,Db,ucomp,RMetric;

    always @(posedge clk or posedge rst)
    begin
    if (rst)
        begin
            sum0 <= 0;
            sum1 <= 0;
            Rsum0 <= 0;
            Rsum1 <= 0;
            Rsum2 <= 0;
            Rsum3 <= 0;
            RDb <= 0;
            Db <= 0;
            ucomp <= 0;
            RMetric <= 0;
            Metric <= 0;
        end
    else
        begin
            sum0 <= bm0 + pm0;
            sum1 <= bm1 + pm1;
            RMetric <= sum1[7] ^ sum0[7];
            ucomp <= (sum1[6:0] > sum0[6:0])? 1'b0 : 1'b1;
            Rsum0 <= sum0;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    Rsum1 <= sum1;

    RDb <= RMetric ^ ucomp;

    Rsum2 <= Rsum0;

    Rsum3 <= Rsum1;

    Metric <= RDb ? Rsum3: Rsum2;

    Db <= RDb;

end

end

endmodule

//โมดูล Dual-port RAM 64 x 64
module dpram64x64(clkwr,clkrd,wa,ra,di,dpo);
    input clkrd,clkwr;

    input [5:0] wa;
    input [5:0] ra;
    input [63:0] di;
    output [63:0] dpo;
    reg [63:0] ram [63:0];
    reg [63:0] dpo;
    integer i;
    initial
    begin
        for (i = 0; i < 65; i = i+1)
            ram[i] <= 0;
        end

    always @(posedge clkwr)
    begin
        ram[wa] <= di;
    end

    always @(posedge clkrd)
    begin
        dpo <= ram [ra] ;
    end
end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
endmodule
```

```
//โมดูล Serial to Parallel
```

```
module S2P(clk,rst,Init,Dbin,Dbout1,Dbout0);
```

```
    input clk,rst,Init;
```

```
    input[3:0]    Dbin;
```

```
    output[127:0] Dbout1,Dbout0;
```

```
    reg[127:0]          Dbout1,Dbout0;
```

```
    reg[255:0]    RDb;
```

```
    always @(posedge clk or posedge rst)
```

```
begin
```

```
    if (rst)
```

```
        RDb <= 0;
```

```
    else
```

```
        RDb <= {Dbin, RDb[255:4]};
```

```
end
```

```
always @(posedge Init or posedge rst )
```

```
begin
```

```
    if (rst)
```

```
    begin
```

```
        Dbout0 <= 0;
```

```
        Dbout1 <= 0;
```

```
    end
```

```
    else
```

```
    begin
```

```
        Dbout0 <= RDb[127:0];
```

```
        Dbout1 <= RDb[255:128];
```

```
    end
```

```
end
```

```
endmodule
```

```
//โมดูล D Flip-Flop 32 bit
```

```
module dff32(clk,rst,din,dout);
```

```
    input clk, rst;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        input[31:0] din;
        output[31:0] dout;
        reg[31:0] dout;

always @(posedge clk or posedge rst )
begin
if (rst)
    dout <=0 ; //asynchronous RESET active High
else
    dout <=din ;
end
endmodule

```

//ท็อปปโมดูล SMU

```

module smu(clk,rst,Init,rdy,db1,db0,dout);
    input clk,rst,Init,rdy;
    input[127:0] db1,db0;
    output dout;
    wire rdy;
    wire[127:0] sout1,sout0,reout1,reout0;
    wire[255:0] rein;
    assign rein = {sout1,sout0};
    sr64x128 sr0(clk,reout0,sout0);
    sr64x128 sr1(clk,reout1,sout1);
    re0 re0(clk,rst,Init,rein,db0,reout0);
    re1 re1(clk,rst,Init,rein,db1,reout1);
    dff dff(rdy,rst,rein[0],dout);
endmodule

```

endmodule

//โมดูล Shift Register 64x128

```

module sr64x128(clk,sin,so);
    input      clk;
    input[127:0] sin;
    output[127:0] so;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

sr8 sr0(clk,sin[0],so[0]);
sr8 sr1(clk,sin[1],so[1]);
sr8 sr2(clk,sin[2],so[2]);
sr8 sr3(clk,sin[3],so[3]);
sr8 sr4(clk,sin[4],so[4]);
sr8 sr5(clk,sin[5],so[5]);
sr8 sr6(clk,sin[6],so[6]);
sr8 sr7(clk,sin[7],so[7]);
sr8 sr8(clk,sin[8],so[8]);
sr8 sr9(clk,sin[9],so[9]);
sr8 sr10(clk,sin[10],so[10]);
sr8 sr11(clk,sin[11],so[11]);
sr8 sr12(clk,sin[12],so[12]);
sr8 sr13(clk,sin[13],so[13]);
sr8 sr14(clk,sin[14],so[14]);
sr8 sr15(clk,sin[15],so[15]);
sr8 sr16(clk,sin[16],so[16]);
sr8 sr17(clk,sin[17],so[17]);
sr8 sr18(clk,sin[18],so[18]);
sr8 sr19(clk,sin[19],so[19]);
sr8 sr20(clk,sin[20],so[20]);
sr8 sr21(clk,sin[21],so[21]);
sr8 sr22(clk,sin[22],so[22]);
sr8 sr23(clk,sin[23],so[23]);
sr8 sr24(clk,sin[24],so[24]);
sr8 sr25(clk,sin[25],so[25]);
sr8 sr26(clk,sin[26],so[26]);
sr8 sr27(clk,sin[27],so[27]);
sr8 sr28(clk,sin[28],so[28]);
sr8 sr29(clk,sin[29],so[29]);
sr8 sr30(clk,sin[30],so[30]);
sr8 sr31(clk,sin[31],so[31]);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

sr8 sr32(clk,sin[32],so[32]);
sr8 sr33(clk,sin[33],so[33]);
sr8 sr34(clk,sin[34],so[34]);
sr8 sr35(clk,sin[35],so[35]);
sr8 sr36(clk,sin[36],so[36]);
sr8 sr37(clk,sin[37],so[37]);
sr8 sr38(clk,sin[38],so[38]);
sr8 sr39(clk,sin[39],so[39]);
sr8 sr40(clk,sin[40],so[40]);
sr8 sr41(clk,sin[41],so[41]);
sr8 sr42(clk,sin[42],so[42]);
sr8 sr43(clk,sin[43],so[43]);
sr8 sr44(clk,sin[44],so[44]);
sr8 sr45(clk,sin[45],so[45]);
sr8 sr46(clk,sin[46],so[46]);
sr8 sr47(clk,sin[47],so[47]);
sr8 sr48(clk,sin[48],so[48]);
sr8 sr49(clk,sin[49],so[49]);
sr8 sr50(clk,sin[50],so[50]);
sr8 sr51(clk,sin[51],so[51]);
sr8 sr52(clk,sin[52],so[52]);
sr8 sr53(clk,sin[53],so[53]);
sr8 sr54(clk,sin[54],so[54]);
sr8 sr55(clk,sin[55],so[55]);
sr8 sr56(clk,sin[56],so[56]);
sr8 sr57(clk,sin[57],so[57]);
sr8 sr58(clk,sin[58],so[58]);
sr8 sr59(clk,sin[59],so[59]);
sr8 sr60(clk,sin[60],so[60]);
sr8 sr61(clk,sin[61],so[61]);
sr8 sr62(clk,sin[62],so[62]);
sr8 sr63(clk,sin[63],so[63]);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

sr8 sr64(clk,sin[64],so[64]);
sr8 sr65(clk,sin[65],so[65]);
sr8 sr66(clk,sin[66],so[66]);
sr8 sr67(clk,sin[67],so[67]);
sr8 sr68(clk,sin[68],so[68]);
sr8 sr69(clk,sin[69],so[69]);
sr8 sr70(clk,sin[70],so[70]);
sr8 sr71(clk,sin[71],so[71]);
sr8 sr72(clk,sin[72],so[72]);
sr8 sr73(clk,sin[73],so[73]);
sr8 sr74(clk,sin[74],so[74]);
sr8 sr75(clk,sin[75],so[75]);
sr8 sr76(clk,sin[76],so[76]);
sr8 sr77(clk,sin[77],so[77]);
sr8 sr78(clk,sin[78],so[78]);
sr8 sr79(clk,sin[79],so[79]);
sr8 sr80(clk,sin[80],so[80]);
sr8 sr81(clk,sin[81],so[81]);
sr8 sr82(clk,sin[82],so[82]);
sr8 sr83(clk,sin[83],so[83]);
sr8 sr84(clk,sin[84],so[84]);
sr8 sr85(clk,sin[85],so[85]);
sr8 sr86(clk,sin[86],so[86]);
sr8 sr87(clk,sin[87],so[87]);
sr8 sr88(clk,sin[88],so[88]);
sr8 sr89(clk,sin[89],so[89]);
sr8 sr90(clk,sin[90],so[90]);
sr8 sr91(clk,sin[91],so[91]);
sr8 sr92(clk,sin[92],so[92]);
sr8 sr93(clk,sin[93],so[93]);
sr8 sr94(clk,sin[94],so[94]);
sr8 sr95(clk,sin[95],so[95]);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

sr8 sr96(clk,sin[96],so[96]);
sr8 sr97(clk,sin[97],so[97]);
sr8 sr98(clk,sin[98],so[98]);
sr8 sr99(clk,sin[99],so[99]);
sr8 sr100(clk,sin[100],so[100]);
sr8 sr101(clk,sin[101],so[101]);
sr8 sr102(clk,sin[102],so[102]);
sr8 sr103(clk,sin[103],so[103]);
sr8 sr104(clk,sin[104],so[104]);
sr8 sr105(clk,sin[105],so[105]);
sr8 sr106(clk,sin[106],so[106]);
sr8 sr107(clk,sin[107],so[107]);
sr8 sr108(clk,sin[108],so[108]);
sr8 sr109(clk,sin[109],so[109]);
sr8 sr110(clk,sin[110],so[110]);
sr8 sr111(clk,sin[111],so[111]);
sr8 sr112(clk,sin[112],so[112]);
sr8 sr113(clk,sin[113],so[113]);
sr8 sr114(clk,sin[114],so[114]);
sr8 sr115(clk,sin[115],so[115]);
sr8 sr116(clk,sin[116],so[116]);
sr8 sr117(clk,sin[117],so[117]);
sr8 sr118(clk,sin[118],so[118]);
sr8 sr119(clk,sin[119],so[119]);
sr8 sr120(clk,sin[120],so[120]);
sr8 sr121(clk,sin[121],so[121]);
sr8 sr122(clk,sin[122],so[122]);
sr8 sr123(clk,sin[123],so[123]);
sr8 sr124(clk,sin[124],so[124]);
sr8 sr125(clk,sin[125],so[125]);
sr8 sr126(clk,sin[126],so[126]);
sr8 sr127(clk,sin[127],so[127]);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
endmodule
```

```
//โมดูล Shift Register 8 x 1
```

```
module sr8(clk,sin,so);
```

```
    input  clk;
```

```
    input  sin;
```

```
    output so;
```

```
    reg[63:0] tmp;
```

```
    assign  so = tmp[0];
```

```
    always @(posedge clk)
```

```
    begin
```

```
        tmp <= {sin,tmp[63:1]};
```

```
    end
```

```
endmodule
```

```
//โมดูล RE0
```

```
module re0(clk,rst,Init,din,sel,dout);
```

```
    input clk,rst,Init;
```

```
    input[255:0]  din;
```

```
    input[127:0]  sel;
```

```
    output[127:0] dout;
```

```
    reg[127:0]    dout;
```

```
    always @(posedge clk or posedge rst)
```

```
    begin
```

```
        if (rst)
```

```
            dout <= 128'h0;
```

```
        else
```

```
            begin
```

```
                if (Init)
```

```
                    dout <= 128'h0;
```

```
            else
```

```
                begin
```

```
                    dout[0] <= sel[0] ? din[1] : din[0];
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[1] <= sel[1] ? din[3] : din[2];
dout[2] <= sel[2] ? din[5] : din[4];
dout[3] <= sel[3] ? din[7] : din[6];
dout[4] <= sel[4] ? din[9] : din[8];
dout[5] <= sel[5] ? din[11] : din[10];
dout[6] <= sel[6] ? din[13] : din[12];
dout[7] <= sel[7] ? din[15] : din[14];
dout[8] <= sel[8] ? din[17] : din[16];
dout[9] <= sel[9] ? din[19] : din[18];
dout[10] <= sel[10] ? din[21] : din[20];
dout[11] <= sel[11] ? din[23] : din[22];
dout[12] <= sel[12] ? din[25] : din[24];
dout[13] <= sel[13] ? din[27] : din[26];
dout[14] <= sel[14] ? din[29] : din[28];
dout[15] <= sel[15] ? din[31] : din[30];
dout[16] <= sel[16] ? din[33] : din[32];
dout[17] <= sel[17] ? din[35] : din[34];
dout[18] <= sel[18] ? din[37] : din[36];
dout[19] <= sel[19] ? din[39] : din[38];
dout[20] <= sel[20] ? din[41] : din[40];
dout[21] <= sel[21] ? din[43] : din[42];
dout[22] <= sel[22] ? din[45] : din[44];
dout[23] <= sel[23] ? din[47] : din[46];
dout[24] <= sel[24] ? din[49] : din[48];
dout[25] <= sel[25] ? din[51] : din[50];
dout[26] <= sel[26] ? din[53] : din[52];
dout[27] <= sel[27] ? din[55] : din[54];
dout[28] <= sel[28] ? din[57] : din[56];
dout[29] <= sel[29] ? din[59] : din[58];
dout[30] <= sel[30] ? din[61] : din[60];
dout[31] <= sel[31] ? din[63] : din[62];
dout[32] <= sel[32] ? din[65] : din[64];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[33] <= sel[33] ? din[67] : din[66];
dout[34] <= sel[34] ? din[69] : din[68];
dout[35] <= sel[35] ? din[71] : din[70];
dout[36] <= sel[36] ? din[73] : din[72];
dout[37] <= sel[37] ? din[75] : din[74];
dout[38] <= sel[38] ? din[77] : din[76];
dout[39] <= sel[39] ? din[79] : din[78];
dout[40] <= sel[40] ? din[81] : din[80];
dout[41] <= sel[41] ? din[83] : din[82];
dout[42] <= sel[42] ? din[85] : din[84];
dout[43] <= sel[43] ? din[87] : din[86];
dout[44] <= sel[44] ? din[89] : din[88];
dout[45] <= sel[45] ? din[91] : din[90];
dout[46] <= sel[46] ? din[93] : din[92];
dout[47] <= sel[47] ? din[95] : din[94];
dout[48] <= sel[48] ? din[97] : din[96];
dout[49] <= sel[49] ? din[99] : din[98];
dout[50] <= sel[50] ? din[101] : din[100];
dout[51] <= sel[51] ? din[103] : din[102];
dout[52] <= sel[52] ? din[105] : din[104];
dout[53] <= sel[53] ? din[107] : din[106];
dout[54] <= sel[54] ? din[109] : din[108];
dout[55] <= sel[55] ? din[111] : din[110];
dout[56] <= sel[56] ? din[113] : din[112];
dout[57] <= sel[57] ? din[115] : din[114];
dout[58] <= sel[58] ? din[117] : din[116];
dout[59] <= sel[59] ? din[119] : din[118];
dout[60] <= sel[60] ? din[121] : din[120];
dout[61] <= sel[61] ? din[123] : din[122];
dout[62] <= sel[62] ? din[125] : din[124];
dout[63] <= sel[63] ? din[127] : din[126];
dout[64] <= sel[64] ? din[129] : din[128];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[65] <= sel[65] ? din[131] : din[130];
dout[66] <= sel[66] ? din[133] : din[132];
dout[67] <= sel[67] ? din[135] : din[134];
dout[68] <= sel[68] ? din[137] : din[136];
dout[69] <= sel[69] ? din[139] : din[138];
dout[70] <= sel[70] ? din[141] : din[140];
dout[71] <= sel[71] ? din[143] : din[142];
dout[72] <= sel[72] ? din[145] : din[144];
dout[73] <= sel[73] ? din[147] : din[146];
dout[74] <= sel[74] ? din[149] : din[148];
dout[75] <= sel[75] ? din[151] : din[150];
dout[76] <= sel[76] ? din[153] : din[152];
dout[77] <= sel[77] ? din[155] : din[154];
dout[78] <= sel[78] ? din[157] : din[156];
dout[79] <= sel[79] ? din[159] : din[158];
dout[80] <= sel[80] ? din[161] : din[160];
dout[81] <= sel[81] ? din[163] : din[162];
dout[82] <= sel[82] ? din[165] : din[164];
dout[83] <= sel[83] ? din[167] : din[166];
dout[84] <= sel[84] ? din[169] : din[168];
dout[85] <= sel[85] ? din[171] : din[170];
dout[86] <= sel[86] ? din[173] : din[172];
dout[87] <= sel[87] ? din[175] : din[174];
dout[88] <= sel[88] ? din[177] : din[176];
dout[89] <= sel[89] ? din[179] : din[178];
dout[90] <= sel[90] ? din[181] : din[180];
dout[91] <= sel[91] ? din[183] : din[182];
dout[92] <= sel[92] ? din[185] : din[184];
dout[93] <= sel[93] ? din[187] : din[186];
dout[94] <= sel[94] ? din[189] : din[188];
dout[95] <= sel[95] ? din[191] : din[190];
dout[96] <= sel[96] ? din[193] : din[192];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[97] <= sel[97] ? din[195] : din[194];
dout[98] <= sel[98] ? din[197] : din[196];
dout[99] <= sel[99] ? din[199] : din[198];
dout[100] <= sel[100] ? din[201] : din[200];
dout[101] <= sel[101] ? din[203] : din[202];
dout[102] <= sel[102] ? din[205] : din[204];
dout[103] <= sel[103] ? din[207] : din[206];
dout[104] <= sel[104] ? din[209] : din[208];
dout[105] <= sel[105] ? din[211] : din[210];
dout[106] <= sel[106] ? din[213] : din[212];
dout[107] <= sel[107] ? din[215] : din[214];
dout[108] <= sel[108] ? din[217] : din[216];
dout[109] <= sel[109] ? din[219] : din[218];
dout[110] <= sel[110] ? din[221] : din[220];
dout[111] <= sel[111] ? din[223] : din[222];
dout[112] <= sel[112] ? din[225] : din[224];
dout[113] <= sel[113] ? din[227] : din[226];
dout[114] <= sel[114] ? din[229] : din[228];
dout[115] <= sel[115] ? din[231] : din[230];
dout[116] <= sel[116] ? din[233] : din[232];
dout[117] <= sel[117] ? din[235] : din[234];
dout[118] <= sel[118] ? din[237] : din[236];
dout[119] <= sel[119] ? din[239] : din[238];
dout[120] <= sel[120] ? din[241] : din[240];
dout[121] <= sel[121] ? din[243] : din[242];
dout[122] <= sel[122] ? din[245] : din[244];
dout[123] <= sel[123] ? din[247] : din[246];
dout[124] <= sel[124] ? din[249] : din[248];
dout[125] <= sel[125] ? din[251] : din[250];
dout[126] <= sel[126] ? din[253] : din[252];
dout[127] <= sel[127] ? din[255] : din[254];
end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

end

end

endmodule

//โมดูล RE1

module re1(clk,rst,Init,din,sel,dout);

    input clk,rst,Init;

    input[255:0]  din;

    input[127:0]  sel;

    output[127:0] dout;

    reg[127:0] dout;

    always @(posedge clk or posedge rst)
    begin
        if (rst)
            dout <= 128'h0;
        else
            begin
                if (Init)
                    dout <= 128'hffffffffffffffff;
                else
                    begin
                        dout[0] <= sel[0] ? din[1] : din[0];
                        dout[1] <= sel[1] ? din[3] : din[2];
                        dout[2] <= sel[2] ? din[5] : din[4];
                        dout[3] <= sel[3] ? din[7] : din[6];
                        dout[4] <= sel[4] ? din[9] : din[8];
                        dout[5] <= sel[5] ? din[11] : din[10];
                        dout[6] <= sel[6] ? din[13] : din[12];
                        dout[7] <= sel[7] ? din[15] : din[14];
                        dout[8] <= sel[8] ? din[17] : din[16];
                        dout[9] <= sel[9] ? din[19] : din[18];
                        dout[10] <= sel[10] ? din[21] : din[20];
                        dout[11] <= sel[11] ? din[23] : din[22];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[12] <= sel[12] ? din[25] : din[24];
dout[13] <= sel[13] ? din[27] : din[26];
dout[14] <= sel[14] ? din[29] : din[28];
dout[15] <= sel[15] ? din[31] : din[30];
dout[16] <= sel[16] ? din[33] : din[32];
dout[17] <= sel[17] ? din[35] : din[34];
dout[18] <= sel[18] ? din[37] : din[36];
dout[19] <= sel[19] ? din[39] : din[38];
dout[20] <= sel[20] ? din[41] : din[40];
dout[21] <= sel[21] ? din[43] : din[42];
dout[22] <= sel[22] ? din[45] : din[44];
dout[23] <= sel[23] ? din[47] : din[46];
dout[24] <= sel[24] ? din[49] : din[48];
dout[25] <= sel[25] ? din[51] : din[50];
dout[26] <= sel[26] ? din[53] : din[52];
dout[27] <= sel[27] ? din[55] : din[54];
dout[28] <= sel[28] ? din[57] : din[56];
dout[29] <= sel[29] ? din[59] : din[58];
dout[30] <= sel[30] ? din[61] : din[60];
dout[31] <= sel[31] ? din[63] : din[62];
dout[32] <= sel[32] ? din[65] : din[64];
dout[33] <= sel[33] ? din[67] : din[66];
dout[34] <= sel[34] ? din[69] : din[68];
dout[35] <= sel[35] ? din[71] : din[70];
dout[36] <= sel[36] ? din[73] : din[72];
dout[37] <= sel[37] ? din[75] : din[74];
dout[38] <= sel[38] ? din[77] : din[76];
dout[39] <= sel[39] ? din[79] : din[78];
dout[40] <= sel[40] ? din[81] : din[80];
dout[41] <= sel[41] ? din[83] : din[82];
dout[42] <= sel[42] ? din[85] : din[84];
dout[43] <= sel[43] ? din[87] : din[86];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[44] <= sel[44] ? din[89] : din[88];
dout[45] <= sel[45] ? din[91] : din[90];
dout[46] <= sel[46] ? din[93] : din[92];
dout[47] <= sel[47] ? din[95] : din[94];
dout[48] <= sel[48] ? din[97] : din[96];
dout[49] <= sel[49] ? din[99] : din[98];
dout[50] <= sel[50] ? din[101] : din[100];
dout[51] <= sel[51] ? din[103] : din[102];
dout[52] <= sel[52] ? din[105] : din[104];
dout[53] <= sel[53] ? din[107] : din[106];
dout[54] <= sel[54] ? din[109] : din[108];
dout[55] <= sel[55] ? din[111] : din[110];
dout[56] <= sel[56] ? din[113] : din[112];
dout[57] <= sel[57] ? din[115] : din[114];
dout[58] <= sel[58] ? din[117] : din[116];
dout[59] <= sel[59] ? din[119] : din[118];
dout[60] <= sel[60] ? din[121] : din[120];
dout[61] <= sel[61] ? din[123] : din[122];
dout[62] <= sel[62] ? din[125] : din[124];
dout[63] <= sel[63] ? din[127] : din[126];
dout[64] <= sel[64] ? din[129] : din[128];
dout[65] <= sel[65] ? din[131] : din[130];
dout[66] <= sel[66] ? din[133] : din[132];
dout[67] <= sel[67] ? din[135] : din[134];
dout[68] <= sel[68] ? din[137] : din[136];
dout[69] <= sel[69] ? din[139] : din[138];
dout[70] <= sel[70] ? din[141] : din[140];
dout[71] <= sel[71] ? din[143] : din[142];
dout[72] <= sel[72] ? din[145] : din[144];
dout[73] <= sel[73] ? din[147] : din[146];
dout[74] <= sel[74] ? din[149] : din[148];
dout[75] <= sel[75] ? din[151] : din[150];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[76] <= sel[76] ? din[153] : din[152];
dout[77] <= sel[77] ? din[155] : din[154];
dout[78] <= sel[78] ? din[157] : din[156];
dout[79] <= sel[79] ? din[159] : din[158];
dout[80] <= sel[80] ? din[161] : din[160];
dout[81] <= sel[81] ? din[163] : din[162];
dout[82] <= sel[82] ? din[165] : din[164];
dout[83] <= sel[83] ? din[167] : din[166];
dout[84] <= sel[84] ? din[169] : din[168];
dout[85] <= sel[85] ? din[171] : din[170];
dout[86] <= sel[86] ? din[173] : din[172];
dout[87] <= sel[87] ? din[175] : din[174];
dout[88] <= sel[88] ? din[177] : din[176];
dout[89] <= sel[89] ? din[179] : din[178];
dout[90] <= sel[90] ? din[181] : din[180];
dout[91] <= sel[91] ? din[183] : din[182];
dout[92] <= sel[92] ? din[185] : din[184];
dout[93] <= sel[93] ? din[187] : din[186];
dout[94] <= sel[94] ? din[189] : din[188];
dout[95] <= sel[95] ? din[191] : din[190];
dout[96] <= sel[96] ? din[193] : din[192];
dout[97] <= sel[97] ? din[195] : din[194];
dout[98] <= sel[98] ? din[197] : din[196];
dout[99] <= sel[99] ? din[199] : din[198];
dout[100] <= sel[100] ? din[201] : din[200];
dout[101] <= sel[101] ? din[203] : din[202];
dout[102] <= sel[102] ? din[205] : din[204];
dout[103] <= sel[103] ? din[207] : din[206];
dout[104] <= sel[104] ? din[209] : din[208];
dout[105] <= sel[105] ? din[211] : din[210];
dout[106] <= sel[106] ? din[213] : din[212];
dout[107] <= sel[107] ? din[215] : din[214];

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

dout[108] <= sel[108] ? din[217] : din[216];
dout[109] <= sel[109] ? din[219] : din[218];
dout[110] <= sel[110] ? din[221] : din[220];
dout[111] <= sel[111] ? din[223] : din[222];
dout[112] <= sel[112] ? din[225] : din[224];
dout[113] <= sel[113] ? din[227] : din[226];
dout[114] <= sel[114] ? din[229] : din[228];
dout[115] <= sel[115] ? din[231] : din[230];
dout[116] <= sel[116] ? din[233] : din[232];
dout[117] <= sel[117] ? din[235] : din[234];
dout[118] <= sel[118] ? din[237] : din[236];
dout[119] <= sel[119] ? din[239] : din[238];
dout[120] <= sel[120] ? din[241] : din[240];
dout[121] <= sel[121] ? din[243] : din[242];
dout[122] <= sel[122] ? din[245] : din[244];
dout[123] <= sel[123] ? din[247] : din[246];
dout[124] <= sel[124] ? din[249] : din[248];
dout[125] <= sel[125] ? din[251] : din[250];
dout[126] <= sel[126] ? din[253] : din[252];
dout[127] <= sel[127] ? din[255] : din[254];
end
end
end
endmodule

```

//ที่อปโมดูล Controller

```

module controller(clk,Init,rst,ACScount,Address_rd,Address_wr,init2p,rdy,clk2);
    input clk,Init,rst;
    output init2p,rdy,clk2;
    output[5:0] ACScount,Address_rd,Address_wr;
    wire InitRD;
    wire[5:0] Address_rd;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

reg inits2p,clk2;

reg [5:0] ACScount;

AddrcounterRd counterRd(clk,rst,InitRD,Address_rd);
AddrcounterWR counterWr(clk,rst,inits2p,Address_rd[5],Address_wr);

control_smu controlSMU(clk,rst,inits2p,rdy);

dff dff0(clk,rst,Init,InitControl);
dff dff1(clk,rst,InitControl,InitRD);

always @(posedge clk or posedge rst )
begin
if (rst)
begin
ACScount <= 0;
inits2p <= 0;
end
else
begin
if (InitControl)
ACScount <= 0;
else
begin
ACScount <= ACScount + 1;
if (ACScount== 7'b0000101)
inits2p <= 1;
else
inits2p <= 0;
end
end
end
always @(negedge clk or posedge rst )
begin
if (rst)
clk2 <= 0;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

else
begin
    if (InitControl)
        clk2 <= 1;
    else
        clk2 <= ~clk2;
    end
end
endmodule

```

//โมดูล Address Read

```

module AddrcounterRd(clk,rst,Init,Address_rd);

```

```

    input  clk,Init,rst;

```

```

    output[5:0] Address_rd;

```

```

    reg PR;

```

```

    reg[4:0] Addr;

```

```

    assign Address_rd = {PR,Addr};

```

```

    always @(posedge clk or posedge rst )

```

```

begin

```

```

    if (rst)

```

```

    begin

```

```

        PR <= 0;

```

```

        Addr <= 5'b0;

```

```

    end

```

```

else

```

```

    begin

```

```

        if (Init)

```

```

        begin

```

```

            PR <= ~PR;

```

```

            Addr <= 5'b0;

```

```

        end

```

```

    else

```

```

        Addr <= Addr + 1;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

end

end

endmodule

//โมดูล Address Write

module AddrcounterWR(clk,rst,Init,PR,Address_wr);

    input clk,Init,rst,PR;
    output[5:0] Address_wr ;

    reg PWR;
    reg[5:0] Addr;

    assign Address_wr = {PWR,Addr[5:1]};
    always @(posedge clk or posedge rst )
    begin
        if (rst)
            begin
                Addr <= 6'b0;
                PWR <= 0;
            end
        else
            begin
                if (Init)
                    begin
                        Addr <= 6'b0;
                        PWR <= ~PR;
                    end
                else
                    Addr <= Addr + 1;
                end
            end
        end
    end

endmodule

//โมดูล Cuntrol SMU

module control_smu(clk,rst,Init,rdy);

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

output rdy;

reg rdy;

reg [5:0] count;

always @(posedge clk or posedge rst )
begin
    if (rst)
        begin
            count <= 0;
            rdy <=0 ;
        end
    else
        if (Init)
            begin
                count <= 0;
                rdy <=0 ;
            end
        else
            begin
                count <= count + 1;
                if (count== 6'b111101)
                    rdy <= 1'b1;
            end
        end
    end
endmodule

```

ภาคผนวก ข.

โปรแกรมทดสอบ

//โปรแกรมทดสอบประสิทธิภาพชุดถอดรหัสในช่องสัญญาณ AWGN ที่ $R = 1/2$

//และที่ $R = 1/3$ โดยแก้ไขตามหมายเหตุ (//)

module testbench();

reg clk;

reg dclk;

reg data;

reg rst;

reg SelectR;

real data_counter;

real error_counter;

real display_counter;

real ber;

real d;

wire dout;

wire [8:0] vdec_in;

wire data_delay;

wire [1:0] enc_out ; //R = 1/2 or

//wire [2:0] enc_out ; //R = 1/3

initial

begin

clk = 0;

data_counter = 0;

data = 0;

error_counter = 0;

ber = 0;

d=0;

SelectR = 0; //R = 1/2 or

//SelectR = 1; //R = 1/3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

        display_counter = 1;

        rst = 1;

        #12500

        rst =0;

    end

    //clock process

    always

    begin

        clk = 1'b0;

        #3750

        clk = 1'b1;

        #3750

        clk = 1'b0;

    end

    // random data process

    always

    begin

        dclk = 1'b0;

        #15000

        dclk = 1'b1;

        data = {$random} % 2; // random data

        data_counter = data_counter + 1; // Counter data bit

        display_counter = display_counter +1;

        #240000

        dclk= 1'b0;

        #225000

        dclk= 1'b0;

    end

    // Convolution Encoder

    encoderk9 ENCK9(dclk,rst,data,enc_out);

    //Channel

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

assign vdec_in[2:0] = channel(enc_out[0]);
assign vdec_in[5:3] = channel(enc_out[1]);
assign vdec_in[8:6] = 0; // R=1/2 or
//assign vdec_in[8:6] = channel(enc_out[3]); // R=1/3
//Viterbidecoder
viterbi uut(clk,dclk,rst,SelectR,vdec_in,dout);
// data delay
data_delay Delay(dclk,rst,data,data_delay);
//Compare data (bit error)
always @(negedge dclk)
begin
    if (data_delay != dout)
        error_counter = error_counter + 1;
    if (display_counter >= 500)
    begin
        display_counter = 0;
        ber = error_counter / data_counter;
        $display("Nubber of data bit:",data_counter);
        $display("Nubber of error bit:",error_counter);
        $display("BER:",ber);
    end
end

end

function [2:0] channel;
    input enc_in;
    real chanel_symbol,recived_signal,wgn;
    real ebno;
    real snr,x1,x2,ran1,ran2,fx,gx;
begin
    ebno = 4; // Set EB/No

```

```

//BPSK modulator

if (enc_in)
    chanel_symbol = -1;
else
    chanel_symbol = 1;

//wgn

snr = 1/sqrt(pow(10,(ebno/10)));
ran1 = {$random}%10000;
x1 = ran1/10000;
ran2 = {$random}%10000;
x2 = ran2/10000;
fx = sqrt(-log(x1));
gx = cos(6.28319 * x2);
wgn= (fx * gx) * snr;
//AWGN
recived_signal = 32 * (chanel_symbol + wgn);
//Quantizer
//BPSK demodulator
//Soft decition
d = (16 * 0.7071) * snr;
if (recived_signal >= (3 * d))
    channel = 3'b000;
if (recived_signal < (3 * d))
    channel = 3'b001;
if (recived_signal < (2 * d))
    channel = 3'b010;
if (recived_signal < d)
    channel = 3'b011;
if (recived_signal < 0)
    channel = 3'b100;
if (recived_signal < (-1 * d))
    channel = 3'b101;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

if (recived_signal < (-2 * d))
channel = 3'b110;
if (recived_signal < (-3 * d))
channel = 3'b111;

```

```

// hard decition
// if (recived_signal >= 0)
// channel = 3'b000;
// else
// channel = 3'b111;
end
endfunction

```

```

function real sin;
input x;
real x;
real x1,x2,x4,x6,x8,x10,sum;
real sign;
begin
sign = 1.0;
x1 = x;
if (x1<0)
begin
x1 = -x1;
sign = -1.0;
end
while (x1 > 3.14159265/2.0)
begin
x1 = x1 - 3.14159265;
sign = -1.0*sign;
end
x2 = x1*x1;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

x4 = x2*x2;
x6 = x4*x2;
x8 = x6*x2;
x10 = x8*x2;
sum = 1.0
      -0.1666666664*x2
      +0.0083333315*x4
      -0.0001984090*x6
      +0.0000027526*x8
      -0.0000000239*x10;
sin = sign*x1*sum;
end
endfunction

function real rootof2;
input n;
integer n;
real power;
integer i;
begin
power = 0.82629586;
power = power / 10000000.0;
power = power + 1.0;
i = -23;
if (n >= 1)
begin
power = 2.0;
i = 0;
end
for (i=i; i< n; i=i+1)
begin
power = power * power;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    end
    rootof2 = power;
end
endfunction

```

```

function real log;
input x;
real x;
real re;
real log2;
integer i;
begin
if (x <= 0.0)
begin
//$display("log illegal argument:",x);
// $stop;
log = 0;
end
else
begin
if (x<1.0)
re = 1.0/x;
else
re = x;
log2 = 0.0;
for (i=7; i>=-23; i=i-1)
begin
if (re > rootof2(i))
begin
re = re/rootof2(i);
log2 = 2.0*log2 + 1.0;
end
end
end
end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

else
    log2 = log2*2;
end
if (x < 1.0)
    log = -log2/12102203.16;
else
    log = log2/12102203.16;
end
end
endfunction

```

```

function real pow;
input x,y;
real x,y;
begin
if (x<0.0)
begin
$display("pow illegal argument:",x);
//$stop;
end
pow = exp(y*log(x));
end
endfunction

```

```

function real sqrt;
input x;
real x;
begin
if (x<0.0)
begin
$display("sqrt illegal argument:",x);
// $stop;

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    end

    sqrt = exp(0.5*log(x));

end

endfunction

function real cos;

    input x;

    real x;

begin

    cos = sin(x + 3.14159265/2.0);

end

endfunction

```

```

function real exp;

    input x;

    real x;

    real x1,power,prod;

    integer i;

begin

    if (x < 0)

        x1 = -x*1.44269504;

    else

        x1 = x*1.44269504;

    if (x1 > 255.0)

        begin

            exp = 0.0;

            if (x>0.0)

                begin

                    $display("exp illegal argument:",x);

                    //$stop;

                end

            end

        end

    end

end

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

else
begin
    prod = 1.0;
    power = 128.0;
    for (i=7; i>=-23; i=i-1)
        begin
            if (x1 > power)
                begin
                    prod = prod * rootof2(i);
                    x1 = x1 - power;
                end
            power = power / 2.0;
        end
    if (x < 0)
        exp = 1.0/prod;
    else
        exp = prod;
    end
end
endfunction
endmodule

//Convolution encoder  $K = 9$   $R = 1/2$ 
module encoderk9(dclk,rst,din,enc_out);
    input dclk;
    input rst;
    input din;
    output [1:0] enc_out ;
    reg [1:0] enc_out;
    reg [8:0] m;
    always @(posedge dclk or posedge rst)
begin

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

if (rst)

    begin

        m <=0;

        enc_out <=0;

    end

else

    begin

        m <= {din,m[8:1]};

        enc_out[0] <= m[8] ^ m[6] ^ m[5] ^ m[4] ^ m[0]; //561
        enc_out[1] <= m[8] ^ m[7] ^ m[6] ^ m[5] ^ m[3] ^ m[1] ^ m[0]; //753

    end

end

endmodule // or
//Convolution encoder  $K = 9$   $R = 1/3$ 
/*module encoderk9(clk,rst,din,enc_out);
    input clk;
    input rst;
    input din;
    output [2:0] enc_out;
    reg [2:0] enc_out;
    reg [8:0] m;
    always @(posedge clk or posedge rst)
    begin
        if (rst)

            begin

                m <=0;

                enc_out <=0;

            end

        else

            begin

                m <= {din,m[8:1]};

                enc_out[0] <= m[8] ^ m[6] ^ m[5] ^ m[3] ^ m[2] ^ m[1] ^ m[0]; //557

```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

    enc_out[1] <= m[8] ^ m[7] ^ m[5] ^ m[4] ^ m[1] ^ m[0]; //663
    enc_out[2] <= m[8] ^ m[7] ^ m[6] ^ m[3] ^ m[0]; //711
end

end

endmodule */

//module data delay
module data_delay(dclk,rst,din,data_delay);
    input dclk;
    input rst;
    input din;
    output data_delay;
    reg [66:0] SR;
    assign data_delay =SR[0];
    always @(posedge dclk or posedge rst)
    begin
    if (rst)
        SR <=0;
    else
        SR <= {din,SR[66:1]};
    end
endmodule

```

สัญญา สัญญาณรบกวนและความสัมพันธ์ในการจำลองด้วย โปรแกรมคอมพิวเตอร์

การจำลองการทำงานในช่องสัญญาณ AWGN (Additive White Gaussian Noise) ดังรูปที่ 5.14 ในบทที่ 5 เป็นการทดสอบประสิทธิภาพของรหัสแก้ไขความผิดพลาดในระบบสื่อสารชนิดหนึ่ง ซึ่งเป็นทดสอบถึงผลกระทบของสัญญาณรบกวนต่ออัตราความผิดพลาดบิต (Bit Error Rate: BER) ซึ่งการทดสอบในช่องสัญญาณแบบ AWGN นี้สัญญาณข้อมูลที่ผ่านการมอดูเลตจะถูกรวมกับสัญญาณรบกวนที่สร้างขึ้นจากกระบวนการสุ่มแบบเกาส์ (Gaussian Random Process) ที่มีค่าเฉลี่ยเป็นศูนย์ (Zero-Mean) [4] ซึ่งขึ้นอยู่กับฟังก์ชันค่าความหนาแน่นของความน่าจะเป็น (Gaussian Probability Density Function: $p(n)$) ซึ่งแสดงดังสมการ

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2\right] \quad (1)$$

โดย σ^2 เป็นความแปรปรวน (Variant) ของเวลา n ซึ่งค่า $p(n)$ ของกระบวนการสุ่มแบบเกาส์ที่มีค่าเฉลี่ยเป็นศูนย์จะสมมติให้ σ เท่ากับหนึ่ง

การจำลองการทำงานด้วยซอฟต์แวร์นั้น สัญญาณรบกวนที่สร้างขึ้นเกิดจากตัวแปรสุ่มแบบเกาส์ซึ่งมีความสัมพันธ์กับตัวแปรสุ่มแบบยูนิฟอร์ม (Uniform Random Variable; u) และตัวแปรสุ่มแบบ Rayleigh (Rayleigh Random Variable; R) [32] ซึ่งตัวแปรสุ่มแบบ Rayleigh แสดงดังสมการ

$$R = \sqrt{2\sigma^2 \ln(1/(1-u))} = \sigma\sqrt{2\ln(1/(1-u))} \quad (2)$$

$$\text{หรือ } R = \sigma\sqrt{2\sqrt{-\ln(u)}} \quad (3)$$

โดย σ เป็นค่าความแปรปรวน (Variant) ของตัวแปรสุ่มแบบ Rayleigh ซึ่งถ้าให้ V เป็นตัวแปรสุ่มแบบยูนิฟอร์ม ตัวแปรสุ่มแบบเกาส์ (G) แสดงดังสมการ

$$G = R \cos(2\pi V) \quad (4)$$

ในการทดสอบประสิทธิภาพของการเข้ารหัสในช่องสัญญาณที่มีการมอดูเลตแบบ BPSK นั้น เป็นการทดสอบหาอัตราความผิดพลาดบิต (BER) ในกรณีที่เข้ารหัสและไม่เข้ารหัส (Uncode) เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ซึ่งจะเปลี่ยนแปลงตามค่าพลังงานที่ใช้ส่งข้อมูลบิตต่อค่าความเข้มของสัญญาณรบกวน (E_b/N_0) ผลการทดสอบดังกล่าวแสดงดังรูปที่ 5.15 ซึ่งเป็นกราฟแสดงผลความสัมพันธ์ระหว่างอัตราความผิดพลาดบิตกับค่า E_b/N_0 โดยอัตราความผิดพลาดบิตได้จากจำนวนบิตที่ผิดพลาดหารด้วยจำนวนข้อมูลอินพุตที่ใช้ในการทดสอบ การทดสอบหาค่าอัตราความผิดพลาดบิตจะทำการเปลี่ยนค่า E_b/N_0 ในหน่วย dB เป็นค่าต่างๆ ซึ่งการเปลี่ยนแปลงดังกล่าวทำได้โดยการเปลี่ยนค่าความแปรปรวน (σ) โดยในช่องสัญญาณ AWGN นั้น สัญญาณรบกวนจะมีค่าสเปกตรัมพลังงานเท่ากับ $N_0/2$ watts/Hz ดังนั้นค่าความแปรปรวน σ^2 จึงเท่ากับ $N_0/2$ ถ้าให้ค่าพลังงานที่ใช้ส่งข้อมูลสัญญาณ (E_s/N_0) เท่ากับหนึ่ง ดังนั้นค่าความแปรปรวนแสดงดังสมการ

$$\frac{E_s}{N_0} = \frac{1}{2\sigma^2} \quad (5)$$

$$\text{หรือ } \sigma = \sqrt{\frac{1}{2(E_s/N_0)}} \quad (6)$$

โดยค่าพลังงานที่ใช้ส่งข้อมูลสัญญาณต่อความเข้มของสัญญาณรบกวน (E_s/N_0) มีความสัมพันธ์กับค่าพลังงานที่ใช้ส่งข้อมูลบิตต่อความเข้มของสัญญาณรบกวน (E_b/N_0) ดังสมการ

$$\frac{E_s}{N_0} = \frac{nE_b}{N_0} \quad (7)$$

ในการจำลองการทำงานนี้ การมอดูเลตเป็นแบบ BPSK ค่า n เท่ากับหนึ่ง ค่าของพลังงานต่อสัญญาณรบกวนจึงเท่ากัน ในการจำลองการทำงานค่าดังกล่าวมีหน่วยเป็น dB การหาค่า E_b/N_0 เพื่อคำนวณหาค่าความแปรปรวนแสดงดังสมการ

$$\frac{E_b}{N_0} = 10^{\frac{E_b/N_0(\text{dB})}{10}} \quad (8)$$

ดังนั้นโปรแกรมการจำลองการทำงานที่ใช้ทดสอบประสิทธิภาพชุดถอดรหัส Viterbi ในช่องสัญญาณ AWGN นี้ จึงได้กำหนดค่า E_b/N_0 ในหน่วย dB เป็นค่าต่างๆ เพื่อคำนวณหาค่าอัตราความผิดพลาดบิตโดยนับจากจำนวนบิตที่ผิดพลาดและจำนวนข้อมูลที่ทดสอบ โดยรายละเอียดของโปรแกรมทดสอบการทำงานนี้อยู่ในภาคผนวก ข. ซึ่งเป็นโปรแกรมภาษาเวริล็อก (Verilog HDL)

ภาคผนวก ง. ข้อมูลอุปกรณ์ชุดรหัสคอนไวท์ชันและชุดถอดรหัส Viterbi จากผู้ผลิตรายอื่นๆ

ประเภท	รุ่น	ผู้ผลิต	ลักษณะ	คุณสมบัติความสามารถ	ราคา	แหล่งข้อมูล
Chip	STEL-2030C	Intel	Encoder/Decoder	17 Mbps, $K=7$, $R=1/2$, $2/3$, $3/4$, BER monitor	-	http://developer.intel.com/design/digital
Chip	STEL-5269	Intel	Encoder/Decoder	152 Kbps, $K=7$, $R=1/2$, $2/3$, BER monitor	-	http://developer.intel.com/design/digital
Chip	STEL-2040C	Intel	Encoder/Decoder	256 Kbps, $K=7$, $R=1/2$, $2/3$, $3/4$, BER monitor	-	http://developer.intel.com/design/digital
Chip	STEL-2060C/	Intel	Decoder	45 Mbps, $K=7$, $R=1/2$, $2/3$, $7/8$, BER monitor	-	http://developer.intel.com/design/digital
Chip	AHA4210 RSVP	AHA	Decoder	62Mbps, $K=7$, $R=1/2$, $2/3$, $3/4$, $5/6$, $7/8$, BER monitor	-	http://www.aha.com
Board	COM-1010	Com Block	Encoder	40 Mbps, select $K=5$, $R=1/7$ $K=7$, $R=1/2$, $2/3$, $3/4$ $5/6$, $7/8$ $K=9$, $R=1/3$, $1/2$, $2/3$	\$295	http://www.comblock.com
Board	COM-1015	Com Block	Decoder	8 Mbps, select $K=5$, $R=1/7$ $K=7$, $R=1/2$, $3/4$, $7/8$, 4 bit soft decision	\$295	http://www.comblock.com
IP Core	V1.0	Xilinx	Encoder	High speed, select $K=3-9$, $R=2/3-12/13$	-	http://www.xilinx.com/ipcenter/ipevaluation
IP Core	V1.0	Xilinx	Decoder	High speed, select $K=3-9$, $R=2/3-12/13$, BER monitor	-	http://www.xilinx.com/ipcenter/ipevaluation
IP Core	V1.0	Xilinx	Encoder	High speed, select $K=3-9$, $R=2/3-12/13$	-	http://www.xilinx.com/ipcenter/ipevaluation
IP Core	VA08	Small world	Encoder	3 Mbps, $K=9$, $R=2/3$, $1/3$, $1/4$, 4 bit decision	-	http://www.sworld.com.au
IP Core	4i2i	4i2i	Encoder/Decoder	$K=7$, $R=1/2$, $1/3$, $2/3$, $1/4$, $5/6$, 1 or n bit decision	-	http://www.sworld.com.au

ผลงานวิจัยที่ได้รับการตีพิมพ์

ชัยวัฒน์ แก้วสาข, เกียรติศักดิ์ ศรีพิมานวัฒน์, อรรถสิทธิ์ หล้าสกุล “การออกแบบสร้างชุด
ถอดรหัส Viterbi ขนาดเล็กสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3” การประชุมวิชาการทาง
วิศวกรรมไฟฟ้า ครั้งที่ 26 (EECON-26) ปี 2546.



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 26

26th Electrical Engineering Conference



Seagate
We turn on ideas



ABB



Volume III

พลังงานและความ

อิเล็กทรอนิกส์ (EL)

สื่อสาร (CM)



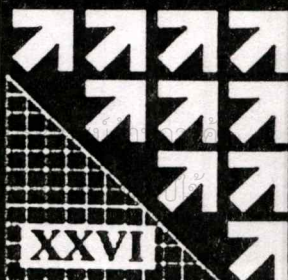
6-7 พฤศจิกายน 2546

โรงแรมโกลเด้นแลนด์ ชะอำ จ.เพชรบุรี

ดำเนินการจัดประชุมโดย

ภาควิชาวิศวกรรมไฟฟ้า และภาควิชาเทคโนโลยีไฟฟ้าอุตสาหกรรม

E
E
C
O
N



XXVI

การออกแบบสร้างชุดถอดรหัส Viterbi ขนาดเล็กสำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3

A Small Architecture of Viterbi Decoder for 3GPP Mobile System

ชัยวัฒน์ แก้วสาย* เกียรติศักดิ์ ศรีพิมานวัฒน์** อรรถสิทธิ์ หล้าสกุล*

*ภาควิชาวิศวกรรมสารสนเทศ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ถ. จลลกรุง เขตลาดกระบัง กรุงเทพฯ 10520 Email: s4061826@kmitl.ac.th klattasi@kmitl.ac.th

**ห้องปฏิบัติการโทรคมนาคมและเครือข่ายคอมพิวเตอร์ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

112 อุทยานวิทยาศาสตร์ประเทศไทย ถ. พหลโยธิน ต. คลองหนึ่ง อ. คลองหลวง ปทุมธานี 12120 Email: keattisak.sripimanwat@nectec.or.th

บทคัดย่อ

บทความนี้นำเสนอวิธีการใหม่ในการออกแบบสร้างชุดถอดรหัสด้วยวิธี Viterbi ที่มีโครงสร้างขนาดเล็กสำหรับถอดรหัสคอนโวลูชันที่มี Constraint length (K) เท่ากับ 9 (256 states) อัตราการเข้ารหัส 1/2 และ 1/3 และ Generator polynomial (561, 753), และ (557, 663, 711), ตามลำดับ ซึ่งกำหนดตามมาตรฐาน TS 25.212 ของ 3GPP ในการออกแบบชุดถอดรหัสนี้ใช้โครงสร้างอนุกรมแบบ 4 Add Compare Select (ACS) และได้ปรับปรุงส่วน Survivor Memory Unit (SMU) แบบ Register Exchange (RE) ให้เหมาะสมกับการทำงานของโครงสร้างแบบ 4 ACS ซึ่งส่งผลให้จำนวนโครงสร้างอุปกรณ์ที่ซิลิคอน ดันแบบชุดถอดรหัส Viterbi นี้ออกแบบโดยใช้ภาษา Verilog HDL สำหรับติดตั้งใน Xilinx Vertex II FPGA ผลการออกแบบและทดสอบชุดถอดรหัสใช้อุปกรณ์ใน FPGA (slices และ block RAM) จำนวนน้อย และให้อัตราเร็วในการถอดรหัสสูงถึง 3.46 Mbps ซึ่งสามารถใช้งานในระบบ 3GPP ได้

คำสำคัญ: รหัสคอนโวลูชัน, ชุดถอดรหัส Viterbi, ระบบโทรศัพท์เคลื่อนที่ยุคที่ 3, FPGA

ABSTRACT

This paper presents a new design and implementation of the small architecture Viterbi decoder. Its specifications are coding rates 1/2 and 1/3, with generator polynomial (561, 753), and (557, 663, 711), respectively. Both cases are designed with the constraint length of 9 (256 states). In addition, they comply with 3GPP (3rd Generation Partnership Project) standard. In this design, the serial Viterbi decoder architecture with four add-compare-select (ACS) combined with the register exchange method is proposed and it is modified to connect with the serial architecture style. This work is designed by using Verilog HDL and implemented on Xilinx Vertex II FPGA. As a result, it obtains small number of slices and block RAM. Furthermore, it achieves up to 3.46 Mbps of data transfer which suitable for 3G W-CDMA mobile system and could be modified for other related communication systems.

Keywords: Convolution Code, Viterbi Decoder, 3GPP, FPGA

1. บทนำ

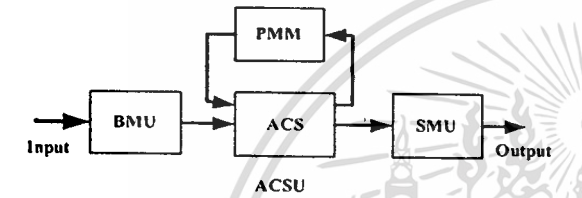
ในระบบสื่อสารนั้น สัญญาณรบกวนเป็นปัญหาสำคัญปัญหาหนึ่งที่ทำให้การรับส่งข้อมูลเกิดความผิดพลาด ซึ่งระบบสื่อสารเหล่านี้จำเป็นต้องได้รับการป้องกันและแก้ไขปัญหาดังกล่าว หนึ่งในวิธีการป้องกันและลดความผิดพลาดเหล่านี้คือ การเข้ารหัสคอนโวลูชัน (Convolution code) และการถอดรหัสแบบ Viterbi ซึ่งเป็นวิธีการป้องกันความผิดพลาดล่วงหน้า (Forward Error Correction: FEC) การเข้ารหัสคอนโวลูชันและการถอดรหัสแบบ Viterbi นี้ได้ถูกนำมาใช้งานในระบบสื่อสารแบบดิจิทัลในปัจจุบันอย่างแพร่หลาย อาทิ ระบบสื่อสารดาวเทียม ระบบโทรศัพท์เคลื่อนที่ และระบบ Wireless LAN เป็นต้น โดยเฉพาะในโทรศัพท์เคลื่อนที่ยุคที่ 3 (3G) การป้องกันความผิดพลาดดังกล่าวเป็นส่วนประกอบหนึ่งที่สำคัญของระบบ ซึ่งในระบบ 3GPP (3rd Generation Partnership Project) กำหนดรหัสเพื่อป้องกันการผิดพลาดนี้ไว้สองแบบ คือ รหัสแบบคอนโวลูชันในช่องสัญญาณเสียง (voice services) และรหัสแบบ Turbo code ในช่องสัญญาณข้อมูล (data service) และกำหนดอัตราการรับส่งข้อมูลที่ 2 Mbps [1-2]

การสร้างชุดถอดรหัส Viterbi สำหรับโครงสร้างทาง VLSI (Very Large Scale Integration Circuit) แบ่งได้เป็นสองประเภท [3] คือ โครงสร้างแบบขนาน (parallel architecture) และแบบอนุกรม (serial architecture) โดยแบบขนานนั้นให้อัตราเร็วในการถอดรหัสสูงแต่พื้นที่ในการสร้าง (silicon area) จะมีขนาดที่ใหญ่กว่าแบบอนุกรม โดยเฉพาะชุดถอดรหัสที่มีจำนวน state สูงๆ เช่น 256 states ในระบบ 3GPP แบบขนานจะใช้อุปกรณ์ในการสร้างจำนวนมาก และมีการเชื่อมต่อภายใน (interconnect) จำนวนมากจึงยากแก่การสร้าง และมีต้นทุนในการสร้างที่สูง จากปัญหาดังกล่าวในบทความนี้จึงได้นำเสนอการออกแบบชุดถอดรหัส Viterbi สำหรับระบบ 3GPP ที่มีโครงสร้างขนาดเล็ก โดยในการออกแบบชุดถอดรหัสนี้ใช้โครงสร้างอนุกรมแบบ 4 Add Compare Select (ACS) ซึ่งเป็นจำนวนของ ACS ที่ใช้อุปกรณ์ในการสร้างจำนวนน้อยและให้ความเร็วในการถอดรหัสสูง [4] นอกจากนั้นส่วน Survivor Memory Unit (SMU) ได้ปรับปรุงวิธีแบบ Register Exchange (RE) ที่ให้ความเร็วในการทำงานสูงแต่มีโครงสร้างขนาดใหญ่ [5] เพื่อให้มีขนาดเล็กและเหมาะสมกับโครงสร้างแบบ 4 ACS ซึ่งชุดถอดรหัส Viterbi นี้มี

โครงสร้างที่ไม่ซับซ้อน การสร้าง (implementation) สามารถทำได้ง่าย และมีความเร็วที่สามารถใช้งานในระบบ 3GPP ได้

2. โครงสร้างของชุดถอดรหัส Viterbi

วิธีถอดรหัสแบบ Viterbi เป็นวิธีที่ให้ประสิทธิภาพสูงในการถอดรหัสคอนวูลูชัน โดยลักษณะการทำงานเป็นแบบ Maximum likelihood decoding ซึ่งผลลัพธ์จากการถอดรหัสจะได้เส้นทางเดียวที่มีความน่าจะเป็นสูงสุดจากเส้นทางทั้งหมดใน Trellis diagram สำหรับโครงสร้างโดยทั่วไปของชุดถอดรหัส Viterbi นั้น แบ่งได้ 3 ส่วน คือ Branch Metric Unit (BMU) Add Compare and Select Unit (ACSU) และ Survivor Memory Unit (SMU) ซึ่งแสดงดังไดอะแกรมในรูปที่ 1



รูปที่ 1 ไดอะแกรมของชุดถอดรหัส Viterbi

การทำงานของชุดถอดรหัสอธิบายตามส่วนต่างๆ ได้ดังต่อไปนี้

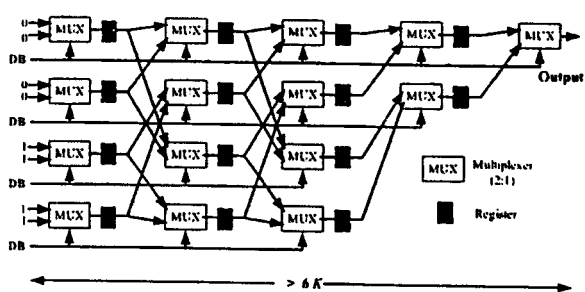
BMU ทำหน้าที่คำนวณหาค่า Branch Metric (BM) จากอินพุตที่เข้ามา ซึ่งการทำงานของ BMU มี 2 แบบ คือ แบบ Hard-decision และ Soft-decision ในการหาค่า BM แบบ Hard-decision ใช้วิธีหาค่า Hamming distance ส่วน Soft-decision ใช้วิธีหาค่า Euclidean distance [6]

ACSU ทำหน้าที่คำนวณค่า Path Metric (PM) และ Decision bit (DB) ในการหาค่า PM และ DB แต่ละ state นั้น ค่า BM จาก BMU และ PM จาก Path Metric Memory (PMM) จะถูกบวกเข้าด้วยกันโดย ACS ซึ่งการบวกนั้นจะได้ค่า PM สองค่าตามเส้นทาง Trellis diagram โดยค่า PM ใหม่ (Update Path Metric) และ DB นั้นจะเลือกจากเส้นทางที่มีค่าน้อยกว่า ซึ่งการหา PM ใน ACSU แสดงดังสมการที่ 1

$$PM_{n+1}^0 = \min(PM_n^0 + BM_n^0, PM_n^1 + BM_n^1) \tag{1}$$

โดยที่ $PM_{n+1}^i (i = 0,1)$ และ $BM_{n+1}^i (i = 0,1)$ คือค่า PM และ BM ที่เวลา n ตามลำดับ [7] หลังจากนั้น DB ถูกส่งไปยัง SMU เพื่อใช้ในการถอดรหัส ส่วนค่า PM ที่ไว้เลือกนั้นจะเก็บไว้ใน PMM สำหรับใช้คำนวณครั้งต่อไป

SMU ทำหน้าที่ถอดรหัสข้อมูลออกมาโดยใช้ DB สำหรับตัดสินใจหาเส้นทางของเกาท์พุทข้อมูล ซึ่งวิธีการของ SMU มี 2 วิธี คือ Register Exchange (RE) และ Trace Back (TB) การทำงานของ RE จะทำงานแบบ forward ซึ่งสามารถถอดรหัสข้อมูลได้แบบต่อเนื่องและมี



รูปที่ 2 วงจร Register Exchange ที่ K เท่ากับ 3

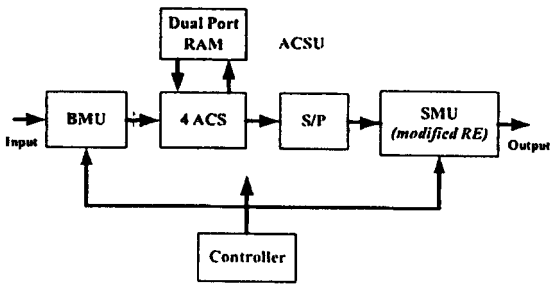
ความเร็วสูง โดยโครงสร้างประกอบด้วย multiplexer (2:1) และ register ซึ่งแสดงในรูปที่ 2 โดยความยาวของ RE ต้องมากกว่า 6 เท่าของค่า K การถอดรหัสจึงจะมีความถูกต้อง [8] วิธีแบบ RE นั้นเหมาะสมสำหรับชุดถอดรหัส Viterbi ที่มีค่า K ต่ำ เนื่องจากค่า K ที่สูงต้องใช้อุปกรณ์การสร้างจำนวนมาก ส่วนวิธี TB การทำงานเป็นแบบ backward ซึ่งจะเริ่มต้นย้อนกลับหาค่าเส้นทางของข้อมูลได้เมื่อเวลามากกว่า 6 เท่าของค่า K (Trace Back depth) และเริ่มต้นจาก state ที่มีค่า PM น้อยที่สุด วิธี TB นั้นเหมาะสมสำหรับชุดถอดรหัส Viterbi ที่มีค่า K สูง เนื่องจากใช้อุปกรณ์ในการสร้างน้อยกว่าแต่ความเร็วในการถอดรหัสช้ากว่า RE เนื่องจากรู้จักการหาค่า PM น้อยที่สุด และต้องใช้หน่วยความจำในการทำงาน วิธีแบบ TB นั้นจะมีสองรูปแบบ คือแบบต่อเนื่องและไม่ต่อเนื่อง สำหรับถอดรหัสแบบต่อเนื่องนั้นวงจรจะมีขนาดใหญ่ และต้องใช้หน่วยความจำเพิ่มขึ้นเป็นสองเท่าของแบบไม่ต่อเนื่อง [5]

3. การออกแบบ

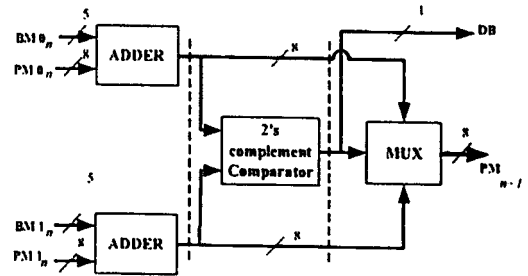
ชุดถอดรหัสแบบ Viterbi สำหรับงานวิจัยนี้ได้ออกแบบตามมาตรฐานของ 3GPP โดยรายละเอียดการออกแบบระบุไว้ดังตารางที่ 1 ส่วนไดอะแกรมของชุดถอดรหัสที่ออกแบบใหม่แสดงดังรูปที่ 3 ซึ่งรายละเอียดการออกแบบอธิบายได้ดังต่อไปนี้

ตารางที่ 1 รายละเอียดของชุดถอดรหัส Viterbi

Coding rate	1/2 และ 1/3
Constraint length	9
Generator polynomials	561 _x , 753 _x ที่ R เท่ากับ 1/2 557 _x , 663 _x , 711 _x ที่ R เท่ากับ 1/3
Survivor path length	64
Decision level	Soft decision แบบ 8 ระดับ (3 บิต) Offset binary representation
Path Metric	8 บิต Modulo Normalization
ACS unit	4
Data transmission	แบบต่อเนื่อง (Continuous)



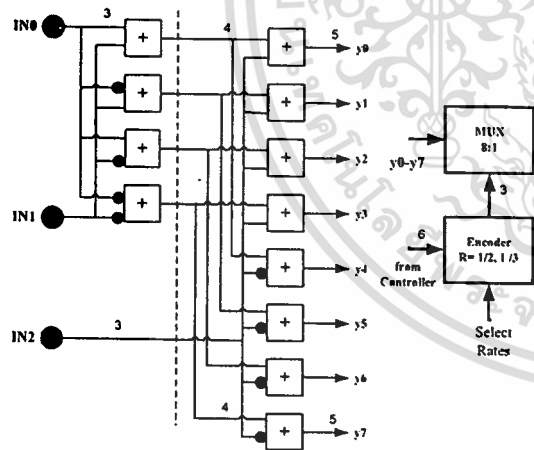
รูปที่ 3 ชุดถอดรหัส Viterbi แบบ 4 ACS ที่มีการปรับปรุงส่วน RE



รูปที่ 5 วงจร ACS ที่ใช้เทคนิค Modulo Normalization

3.1 ภาค Branch Metric Unit: BMU

ภาค BMU ถูกออกแบบให้ทำงานแบบ *Soft decision* 8 ระดับ (3 บิต) โดยการแทนค่าใน BMU นี้มีสองแบบคือ *Signed Magnitude* และ *Offset Binary* [4][9] ซึ่ง *Offset Binary* จะใช้หน่วยในการสร้างน้อยกว่า จึงถูกเลือกใช้สำหรับงานวิจัยนี้ ภาค BMU นี้ประกอบด้วย 12 Adders 8 multiplexers (8:1) และ 8 encoders โดย encoder จะทำหน้าที่เลือกค่า BM จาก Adder และเลือกอัตราการถอดรหัสให้เป็น 1/2 หรือ 1/3 ซึ่งการทำงานถูกควบคุมโดย Controller และเพื่อให้ BMU มีอัตราเร็วในการทำงานที่สูงขึ้น จึงได้แบ่งการทำงานของ BMU เป็น 2 pipelines ซึ่งวงจรการทำงานแสดงดังรูปที่ 4

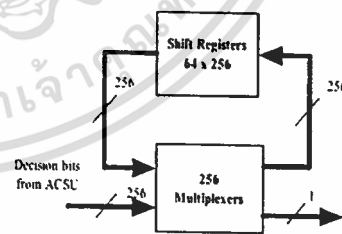


รูปที่ 4 วงจร BMU แบบ 3 bit Soft decision

เพื่อทำให้ความเร็วในการทำงานของ ACSU เพิ่มขึ้น งานวิจัยนี้ได้แบ่งการทำงานของ ACSU เป็น 3 pipelines และการแก้ปัญหาการเกิด *Overflow* ของ ACS ในงานวิจัยนี้ได้ใช้เทคนิค *Modulo Normalization* [10] ซึ่งวงจรการทำงานของ ACS นี้แสดงดังรูปที่ 5

3.3 การปรับปรุงภาค Survivor Memory Unit: SMU

ในภาค SMU นี้วิธี RE ได้ถูกปรับปรุงให้ขนาดของวงจรลดลง โดย SMU นี้ประกอบด้วย 64 x 256 shift register และ 256 multiplexers (2:1) การทำงานเริ่มต้นจาก DB 256 บิตจาก ACSU จะถูกบันทึกที่ shift register จากนั้นข้อมูลใน shift register จะถูกส่งเข้า 256 multiplexers ตามสัญญาณนาฬิกา ซึ่งข้อมูลจาก multiplexers จะถูกเขียนกลับ shift register ซึ่งรูปที่ 6 แสดงการทำงานของภาค SMU นี้ โดยการทำงานนี้จะเหมือนกับการทำงานในรูปที่ 2 แต่ใช้เวลานานกว่า 64 เท่า ซึ่งการทำงานนี้เข้ากันพอดีกับภาค ACSU ที่ออกแบบไว้ ดังนั้นชุดถอดรหัส Viterbi นี้ใช้เวลา 64 clock ในการถอดรหัสข้อมูลแต่ละบิต



รูปที่ 6 วงจร SMU ที่มีการปรับปรุง RE

3.2 ภาค Add Compare and Select Unit: ACSU

ในภาค ACSU นี้ประกอบด้วย 4 ACS dual port RAM ขนาด 64 x 64 และ Serial to Parallel (S/P) ดังแสดงในรูปที่ 3 โดย ACS คำนวณค่า PM และ DB โดยค่า PM จะถูกเก็บไว้ที่ dual port RAM ซึ่งทำหน้าที่เป็น PMM เมื่อครบรอบการทำงาน (64 clock cycle) DB จะถูกเปลี่ยนเป็นข้อมูลแบบขนาน 256 บิตโดย S/P แล้วส่งให้ภาค SMU ต่อไป

4. ผลการออกแบบและทดสอบ

ชุดถอดรหัส Viterbi นี้ถูกออกแบบโดยใช้ภาษา Verilog HDL และติดตั้งใน Xilinx Vertex II FPGA xc2v1000-6 โดยใช้โปรแกรม Xilinx ISE และ ModelSim ในการออกแบบ ตรวจสอบ และจำลองการทำงาน ซึ่งผลการใช้โครงสร้างและอุปกรณ์ใน FPGA ของชุดถอดรหัสนี้สรุปได้ดังในตารางที่ 2

ตารางที่ 2 จำนวนการใช้อุปกรณ์ใน FPGA (xc2v1000-6)

ภาค	Area (slices)	Block RAM
BMU	139	-
ACSU	468	2
SMU	795	-
Controller	14	-
รวม	1416	2

และในส่วนของการจำลองการทำงาน (Timing simulation) ความถี่สูงสุดที่สามารถทำงานได้ (Maximum Frequency) คือ 222 MHz ซึ่งให้ความเร็วในการถอดรหัสสูงถึง 3.46 Mbps เมื่อเปรียบเทียบกับ IP core ของ Xilinx [9] ชุดถอดรหัสนี้ใช้อุปกรณ์และพื้นที่ใน FPGA (slices และ block RAM) น้อยกว่า และยังสามารถถอดรหัสได้ทั้งอัตรา 1/2 และ 1/3 แต่มีความเร็วในการถอดรหัสที่ต่ำกว่า ผลการเปรียบเทียบคุณสมบัติของชุดถอดรหัสนี้กับ [9] ที่ติดตั้งใน FPGA รุ่น xc2v1000-6 แสดงในตารางที่ 3

ตารางที่ 3 การเปรียบเทียบคุณสมบัติของชุดถอดรหัส

คุณสมบัติของชุดถอดรหัส	งานวิจัยนี้	Xilinx [9]
Constraint length	9	9
Coding rate	1/2, 1/3	1/2
Survivor path length	64	64
Soft decision (bit)	3	3
Area (slices)	1416	1775
Block RAM	2	4
Max. clock frequency (MHz)	222	172
Max. data rate (Mbps)	3.46	15

5. สรุป

บทความนี้ได้นำเสนอชุดถอดรหัส Viterbi ที่มีโครงสร้างขนาดเล็ก โดยชุดถอดรหัสใช้โครงสร้างแบบอนุกรม 4 ACS และได้ปรับปรุง SMU แบบ RE ให้เหมาะสมกับ ข้อดีของชุดถอดรหัสนี้คือ สามารถถอดรหัสได้ทั้งอัตราการเข้ารหัส 1/2 และ 1/3 และใช้อุปกรณ์ใน Xilinx Vertex II FPGA (slices และ block RAM) น้อย ส่วนอัตราเร็วในการถอดรหัสนั้นสูงถึง 3.46Mbps ซึ่งสามารถใช้สำหรับระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ได้

6. กิตติกรรมประกาศ

งานวิจัยนี้ได้รับความอนุเคราะห์ซอฟต์แวร์จากศูนย์พัฒนารัฐกิจออกแบบวงจรรวม (TIDA) และ Xilinx Vertex II FPGA Board จากโครงการวิจัยชุดค้นแบบหน่วยปรับปรุงประสิทธิภาพการสื่อสารวิทยุ-สำหรับโครงการ 3G และการประยุกต์สำหรับอุตสาหกรรม (RDT2) ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

เอกสารอ้างอิง

- [1] 3rd Generation Partnership Project (3GPP), *Technical Specification Group (TSG) Multiplexing and channel coding (FDD)*, TS 25.212 v2.3.0, 1999.
- [2] Bickerstaff M.A., Garrett D., Prokop T., Thomas C., Widdup B., Gongyu Zhou, Davis L.M., Woodward G., Nicol C., Ran-Hong Yan, "A unified turbo/viterbi channel decoder for 3GPP mobile wireless in 0.18-/spl mu/m CMOS", *IEEE Journal of Solid-State Circuits*, Vol. 37, pp 1555-1564, Nov 2002.
- [3] Bupesh Pandita, Subir K Roy., "Design and implementation of a Viterbi decoder using FPGAs", *IEEE VLSI Design Twelfth International Conference*, pp. 611-614, 1999.
- [4] Sayed Aon Mujtaba, "An area-efficient VLSI architecture of the Viterbi decoder for reverse link IS-95 (CDMA) air interface", *IEEE Signal Processing Proceedings, ICSP '98*, pp. 525-528, 1998.
- [5] Gennady Feygin, P.G. Gulak, "Architectural Tradeoff for Survivor Sequence Memory in Viterbi Decoder", *IEEE Transactions on Communications*, Vol. 41, pp. 425-429, 1998.
- [6] HL Lou, "Implementing the Viterbi Algorithm", *IEEE Signal Processing Magazine*, Vol. 15, pp 42-52, 1995.
- [7] Hiroshi Suzuki, Yun-Nan Chang, Keshab K. Parhi, "Low-power Bit-Serial Viterbi decoder for 3rd Generation W-CDMA Systems", *IEEE 1999 Custom Integrated Circuits Conference*, pp 589-592, 1999.
- [8] S.B. Wicker, *Error Control System for Digital Communication and Storage*, Prentice Hall, 1995.
- [9] Xilinx Inc. *LogiCore Viterbi decoder V 2.0*, 2002. [Online] <http://www.xilinx.com>
- [10] Shung C.B., Siegel P.H., Ungerboeck G., Thapar H.K., "VLSI architectures for metric normalization in the Viterbi algorithm", *IEEE SUPERCOMM/ICC '90 Conference*, pp. 1723-1728, 1990.

ประวัติผู้เขียน

นายชัยวัฒน์ แก้วสาย เกิดเมื่อวันที่ 28 กันยายน 2519 ที่จังหวัดลำปาง สำเร็จการศึกษา
อุตสาหกรรมศาสตรบัณฑิต (อิเล็กทรอนิกส์) เกียรตินิยมอันดับ 2 จากสถาบันเทคโนโลยีพระจอม
เกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2543 และประกาศนียบัตรวิชาชีพชั้นสูง
(อิเล็กทรอนิกส์สื่อสาร) จากวิทยาลัยเทคนิคลำปาง ในปีการศึกษา 2540

ในระหว่างปี 2545-2546 เคยทำงานในตำแหน่งผู้ช่วยวิจัยในโครงการวิจัยชุดต้นแบบหน่วย
ปรับปรุงประสิทธิภาพการสื่อสารสัญญาณสำหรับโครงการ 3G และการประยุกต์สำหรับอุตสาหกรรม
(RDT2) ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

