

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การออกแบบวงจรรวมเฉพาะกิจเพื่อป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM
AN ASIC DESIGN FOR EEPROM ACCESS SECURITY



วิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2538

ISBN 974-621-301-6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษานาน ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ลิขสิทธิ์ของบัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

AN ASIC DESIGN FOR EEPROM ACCESS SECURITY



A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
GRADUATE SCHOOL
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1995

ISBN 974-621-301-6

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อวิทยานิพนธ์	การออกแบบวงจรรวมเฉพาะกิจเพื่อป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM
นักศึกษา	นายธีรศักดิ์ชัย อังสกุล
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ. ดร. บุญวัฒน์ อัดชู
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2538

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบวงจรรวมเฉพาะกิจเพื่อช่วยป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM ทำให้การติดต่อใดๆกับหน่วยความจำชนิด EEPROM นี้กระทำได้โดยการส่งผ่านรหัสเข้าไปก่อน ซึ่งเมื่ออินเทอร์เฟสเข้ากับ EEPROM แล้วสามารถนำไปประยุกต์ใช้งานได้หลายลักษณะเช่น วงจรกำหนดรหัสประจำตัว (Identification Code) และวงจรฮาร์ดลอค หรือวงจรที่ช่วยป้องกันการคัดลอกซอฟต์แวร์ (Hardlock) สำหรับวิทยานิพนธ์ฉบับนี้ได้นำไปใช้เป็นอุปกรณ์ป้องกันการคัดลอกซอฟต์แวร์ ในการออกแบบวงจรได้ออกแบบเป็น 2 ส่วนคือ ส่วนที่เป็นวงจรรวมเพื่อช่วยป้องกันข้อมูลในหน่วยความจำออกเป็นผังวงจรไฟฟ้า (Schematic) และส่วนของหน่วยความจำออกแบบโดยใช้ภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) เขียนบรรยายพฤติกรรมของหน่วยความจำที่ต้องการป้องกันการเข้าถึงข้อมูลภายใน หลังจากนั้นนำทั้งสองส่วนมารวมกัน แล้วทำการจำลองการทำงาน (Simulation) โดยขั้นตอนทั้งหมดนี้ใช้ซอฟต์แวร์ของ Mentor Graphics บนเครื่อง Engineering Workstation ผลการทดสอบการทำงานพบว่าวงจรที่ออกแบบนี้สามารถทำงานได้อย่างถูกต้อง และเพื่อเป็นการยืนยันความถูกต้องของวงจรที่ออกแบบได้ ได้ทำการพัฒนาต่อเป็นตัววงจรรวมเฉพาะกิจจริงขึ้นมาโดยใช้เทคโนโลยีของ Field Programmable Logic Array (FPGA) ตัวอุปกรณ์ป้องกันการคัดลอกที่พัฒนานี้ภายในประกอบด้วย FPGA เบอร์ XC3042A , Serial PROM XC1736D ที่ใช้เก็บโปรแกรมคอนฟิกของ FPGA และ Serial EEPROM เบอร์ HY93C46 วงจรที่ออกแบบได้มีจำนวนเกตทั้งหมด 1,335 เกต การทดลองใช้งานโดยต่อกับพอร์ตขนาน ไม่ต้องมีแหล่งจ่ายไฟจากภายนอก ในการติดต่อกับ EEPROM จะต้องส่งรหัสลับบนอนุกรมจำนวน 72 บิต ซึ่งประกอบด้วยคำสั่ง 24 บิต รหัสผ่าน 32 บิต และข้อมูล 16 บิต หลังจากนั้นกรณีที่ต้องการอ่านข้อมูลจาก EEPROM จะได้เอาพุตเป็นข้อมูลที่ผ่านการถอดรหัสออกมา และกรณีอื่นๆที่มีการผ่านรหัสผิดพลาดจะได้เอาพุตเป็นข้อมูลสุ่มออกมา

Thesis Title	An ASIC Design for EEPROM Access Security
Student	Mr. Teerasakchai Aungsakul
Thesis Advisor	Assist. Prof. Dr. Boonwat Attachoo
Level of Student	Master of Engineering in Electrical Engineering
Department	Computer Engineering King Mongkut's Institute of Technology Ladkrabang
Year	1995

ABSTRACT

This thesis presents the ASIC design for security of accessible EEPROM. Such a circuit can have wide range of applications where password or secret code is required to guarantee the access right. Hardlock and identification code circuit are good examples for this application. The design of the hardlock to support the circuit is highlight. Design and simulation are completed by using Mentor Graphics Software. Schematic capture is used in designing of the ASIC part while VHDL is used in designing of the EEPROM. The designed ASIC and the EEPROM are combined and the simulation is carried out. The obtained results have been demonstrated with the satisfied output. To verify the design idea and to prove the design methodology as well as the simulated results, the prototype hardlock supported circuit was implemented using FPGA technology. The whole hardware consists of three parts: a FPGA support circuit, a serial PROM which is used to load configuration program (Bitstream) on LCA structure of the FPGA, and a serial EEPROM. There are 1,335 gates involved in the FPGA circuit. The prototype hardware is connected to the parallel port without additional power supply. It requires absolutely correct 72-bits code in series (24-bits for command, 32-bits for password, 16-bits for data) to access the EEPROM, and if the 72-bits code is not correct, the output data are random.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีเพราะได้รับการสนับสนุนในทุกๆ ด้านจาก ผู้ช่วยศาสตราจารย์ ดร.บุญวัฒน์ อัฐุ ผู้วิจัยรู้สึกซาบซึ้ง และขอกราบขอบพระคุณเป็นอย่างสูง

สุดท้ายนี้ผู้วิจัยขอขอบพระคุณศูนย์เทคโนโลยีอิเล็กทรอนิกส์ และคอมพิวเตอร์แห่งชาติ มูลนิธิคอมพิวเตอร์และการสื่อสาร ที่ได้มอบทุนสนับสนุนงานวิจัยจนทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงด้วยดี

นายธีรศักดิ์ชัย อังสกุล



สารบัญ

	หน้า
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VIII
สารบัญภาพ	IX
บทที่	
1. บทนำ	1
1.1 ความเป็นมาและความสำคัญของปัญหา	1
1.2 วัตถุประสงค์และแนวความคิดในการวิจัย	1
1.3 วิธีดำเนินการวิจัย	2
1.4 รวบรวมเฉพาะกิจชนิดต่างๆ	2
1.4.1 Field Programmable	3
1.4.2 Mask Programmable	3
1.5 เทคโนโลยีของต้นแบบฮาร์ดลอค	4
1.6 ซอฟต์แวร์ที่ใช้ออกแบบฮาร์ดลอค	4
1.7 เปรียบเทียบกับฮาร์ดลอคแบบเดิม	5
1.8 ภาษา VHDL	7
1.9 การเรียบเรียงเนื้อหาภายในวิทยานิพนธ์	7
2. การออกแบบวงจรฮาร์ดลอคเป็นผังวงจรไฟฟ้า และ VHDL	9
2.1 วงจรช่วยเสริมการทำงานให้กับฮาร์ดลอค และทฤษฎีของ DES	9
2.1.1 วงจรเข้ารหัส และถอดรหัส	10
2.1.2 ทฤษฎีของ DES	12
2.1.2.1 ขั้นตอนการเข้ารหัสและถอดรหัส	12
2.1.2.2 การสับเปลี่ยนตำแหน่ง	13
2.1.2.3 Function_F และ S_box	17
2.1.2.4 การคำนวณหมายเลขกุญแจ (Subkey)	20
2.1.2.5 ตัวอย่างการเข้ารหัส และถอดรหัสโดยทำ Function_F 2 ครั้ง	24
2.1.3 ซิฟทีร์จิสเตอร์	27
2.1.4 วงจรสุ่มข้อมูลแบบเทียมขนาด 16 บิต	34

2.2	การออกแบบวงจรฮาร์ดล็คเป็นผังวงจรไฟฟ้า	36
2.2.1	รายละเอียดของวงจรฮาร์ดล็ค	36
2.2.2	ลักษณะการทำงานของฮาร์ดล็ค	39
2.2.3	การจัดเก็บข้อมูลของฮาร์ดล็ค	43
2.3	การออกแบบหน่วยความจำชนิด EEPROM โดยใช้ VHDL	43
2.3.1	องค์ประกอบพื้นฐานใน VHDL	44
2.3.1.1	การกำหนดการเชื่อมต่อ	44
2.3.1.2	การกำหนดรูปแบบการบรรยาย	45
2.3.1.3	โปรแกรมย่อย	45
2.3.1.4	โอเปอเรเตอร์	47
2.3.1.5	เวลาและความพร้อมเพรียง	47
2.3.1.6	สัญญาณและตัวแปร	47
2.3.2	การบรรยายเชิงพฤติกรรม	47
2.3.2.1	โพรเซส	48
2.3.2.2	การกำหนดตัวกระทำภายในโพรเซส	48
2.3.2.3	การกำหนดการกระทำภายในโพรเซส	49
2.3.2.4	การกระตุ้นและยับยั้งการกระทำของโพรเซส	50
2.3.2.5	การหยุดรอ	51
2.3.2.6	การสร้างองค์ประกอบย่อย	51
2.3.2.7	แพคเกจ	53
2.3.3	การบรรยายการทำงานระดับพฤติกรรมของ EEPROM	53
2.3.3.1	กำหนดไลบรารี	53
2.3.3.2	Basic_Uilities Package	54
2.3.3.3	การบรรยายพฤติกรรมหน่วยความจำภายในของ EEPROM	55
2.3.3.4	องค์ประกอบหลักของ EEPROM	55
2.4	การสร้างสัญญาณทดสอบการทำงานด้วยภาษา MISL	58
2.5	ผลการทดสอบ การจำลองการทำงาน	60
3.	การพัฒนาอุปกรณ์ฮาร์ดล็คโดยใช้เทคโนโลยีของ FPGA	63
3.1	องค์ประกอบของต้นแบบฮาร์ดล็ค	63
3.1.1	64*16 Bit Serial Electrically Erasable PROM	64
3.1.1.1	คุณลักษณะเฉพาะ	64
3.1.1.2	รายละเอียดขา	64
3.1.1.3	รายละเอียดการใช้งาน	65

3.1.2	Serial Configuration PROMs (XC17xxx Family)	69
3.1.2.1	คุณลักษณะเฉพาะ	69
3.1.2.2	รายละเอียดขา	69
3.1.2.3	รายละเอียดการใช้งาน	71
3.1.3	FPGA (XC3042)	71
3.1.3.1	คุณลักษณะเฉพาะ	72
3.1.3.2	โครงสร้างภายใน	72
3.1.3.3	รายละเอียดขา	75
3.1.3.4	รายละเอียดการใช้งาน	78
3.1.3.4.1	การใช้งานในโหมด Slave Serial	80
3.1.3.4.2	การใช้งานในโหมด Master Serial	82
3.1.3.4.3	การใช้งานในโหมด Master Parallel	82
3.1.3.5	ข้อควรระวังในการใช้ FPGA	83
3.2	การใช้ซอฟต์แวร์ ViewLogic และ Xilinx	84
3.2.1	ขั้นตอนในการออกแบบ และจำลองการทำงานโดยใช้ซอฟต์แวร์ ViewLogic	84
3.2.2	ขั้นตอนในการทดสอบวงจรโดยใช้ Xilinx	85
3.3	ฮาร์ดแวร์ระบบ	86
3.4	ซอฟต์แวร์ระบบ	87
3.4.1	การสร้างสัญญาณผ่านพอร์ตขนาน	87
3.4.2	โปรแกรมสนับสนุนฮาร์ดลอค	88
3.4.2.1	โปรแกรม ID_GEN	89
3.4.2.2	โปรแกรม CODE_GEN	90
3.4.2.3	โปรแกรม HL_SHELL	90
4.	การสร้าง และการทดสอบ	93
4.1	ต้นแบบฮาร์ดลอค	93
4.2	วิธีเชื่อมต่อต้นแบบฮาร์ดลอค	94
4.3	การทดสอบต้นแบบ	96
4.3.1	การทดสอบด้วยโปรแกรมเขียนรหัส CODE_GEN	96
4.3.2	การทดสอบด้วยโปรแกรมอ่านรหัส CODE_RD	97
4.3.3	การทดสอบด้วยโปรแกรมอ่านรหัสสุ่ม RAND_RD	97
4.3.4	การทดสอบด้วยโปรแกรมป้องกันการคัดลอก HL_SHELL	98
5.	สรุปผลวิจัยและข้อเสนอแนะ	103

เอกสารอ้างอิง

104

ภาคผนวก ก. วงจรต้นแบบฮาร์ดล็ค

ภาคผนวก ข. โปรแกรมสนับสนุนฮาร์ดล็ค

ภาคผนวก ค. ข้อมูลไอซี EEPROM HY93C46, PROM XC17xxx และ FPGA XC3xxx



สารบัญตาราง

ตารางที่	หน้า
1. ตารางสับเปลี่ยนตำแหน่ง IP	15
2. ตารางสับเปลี่ยนตำแหน่ง IP ⁻¹	15
3. ตัวอย่าง IP ใหม่	15
4. ตัวอย่าง IP ⁻¹ ใหม่	15
5. ตาราง E	18
6. ตาราง P	18
7. S_box	19
8. ตาราง PC_1	21
9. ตาราง PC_2	21
10. ตารางแสดงจำนวนครั้งในการหมุนซ้าย (LS)	21
11. แสดงค่าหมายเลขกุญแจที่คำนวณได้ตามขั้นตอนการ DES	22
12. ค่าของโพลีโนเมียลและคาบ	32
13. จำนวนข้อมูลที่ได้จากวงจรสุ่มที่มีภาคต่างกัน	36
14. อธิบายการทำงานของชาสัญญาณต่างๆของวงจรทุกส่วน	38
15. แสดงฟังก์ชันการทำงานของแต่ละคำสั่ง	41
16. แอดเดรสที่ใช้เป็นรหัสผ่าน	41
17. แสดงชาสัญญาณของ EEPROM	65
18. ชุดคำสั่งของ EEPROM (HY93C46)	67
19. แสดงหน้าที่ชาสัญญาณต่างๆของ EEPROM	70
20. แสดงขนาดที่เหมาะสมของ SCPs กับ FPGA เบอร์ต่างๆ	70
21. แสดงคุณสมบัติของ FPGA ตระกูลต่างๆ	72
22. ตารางประมาณการนับเกทของเกทพื้นฐาน	73
23. โหมดต่างๆของการคอนฟิก (Configuration)	79

สารบัญภาพ

ภาพที่	หน้า
1. ผังแสดงประเภทของวงจรรวมเฉพาะกิจ	3
2. โพล์ชาร์ตการทำงานของฮาร์ดลอค	5
3. โพล์ชาร์ตการทำงานบนเครื่องคอมพิวเตอร์	6
4. โพล์ชาร์ตการทำงานบนฮาร์ดลอค	6
5. บล็อกแสดงขั้นตอนการจำลองการทำงาน	9
6. ระบบการสร้างรหัสลับ	10
7. ระบบการสร้างรหัสลับด้วยกุญแจรหัสตัวเดียว	11
8. ระบบการสร้างรหัสลับด้วยกุญแจการเข้ารหัส และถอดรหัสแยกออกจากกัน	11
9. สร้างวงจรเข้ารหัสด้วย Serial-in Parallel-out Shift Register	11
10. สร้างวงจรถอดรหัสด้วย Serial-in Parallel-out Shift Register	12
11. บล็อกแสดงขบวนการเข้าและถอดรหัส	13
12. บล็อกแสดงขบวนการ IP และ IP^{-1}	14
13. รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP	14
14. รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP^{-1}	14
15. บล็อกแสดงขั้นตอนการเข้ารหัส	16
16. บล็อกไดอะแกรมแสดงขั้นตอนการสร้าง Function $F(R, K)$	18
17. บล็อกแสดงการคำนวณหมายเลขกุญแจ 16 ชุด	20
18. บล็อกแสดงขั้นตอนการถอดรหัส	23
19. บล็อกแสดงขั้นตอนการเข้ารหัสโดยทำ Function F 2 ครั้ง	24
20. บล็อกแสดงขั้นตอนการถอดรหัสโดยทำ Function F 2 ครั้ง	26
21. บล็อกไดอะแกรมของชิพรีจิสเตอร์ที่มีสัญญาณย้อนกลับ	27
22. แสดงชิพรีจิสเตอร์แบบสามภาค การเปลี่ยนสถานะ และไดอะแกรมของสถานะ	28
23. แสดงชิพรีจิสเตอร์แบบสามภาคที่สร้างสัญญาณย้อนกลับจากภาค 1 และภาค 3	29
24. แสดงชิพรีจิสเตอร์แบบสี่ภาคที่สร้างสัญญาณย้อนกลับจากภาค 1, 2, 3, 4	31
25. วงจรกำหนดตัวเลขสุ่มแบบเทียมขนาด 6 ภาค	34
26. การต่อวงจรสำหรับ 6 และ 16 ภาค	35
27. วงจรที่ป้องกันกรณีเป็น "0" ทุกรีจิสเตอร์	35
28. แสดงขาสัญญาณของ MODULE1-MODULE7	37
29. องค์ประกอบหลักของฮาร์ดลอค	39
30. ลำดับการรับข้อมูลคำสั่งของฮาร์ดลอค	39
31. โพล์ชาร์ตแสดงการทำงานของฮาร์ดลอค	40

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

32. การจัดแบ่งข้อมูลที่อยู่ภายในฮาร์ดล็ค	43
33. บล็อกแสดงการเชื่อมต่อ Symbol ของฮาร์ดล็คและหน่วยความจำ	43
34. แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม	44
35. แสดงบล็อกไดอะแกรมและการเชื่อมต่อของ clock_component	44
36. แสดงการบรรยายเชิงพฤติกรรมของ clock_component	45
37. แสดงตัวอย่างฟังก์ชัน	46
38. แสดงตัวอย่างโปรซีเจอร์	46
39. แสดงโอเพอร์เรเตอร์ใน VHDL	47
40. แสดงรูปแบบและตัวอย่างของการบรรยายแบบโปรเซส	48
41. แสดงตัวอย่างการประกาศตัวกระทำภายในโปรเซส	49
42. แสดงเงื่อนไขการกระทำในโปรเซส	49
43. (a) โมเดล D-FlipFlop (b) การบรรยายการเชื่อมต่อของ D-FlipFlop (c) การบรรยายเชิงพฤติกรรมของ D-FlipFlop	50 50
44. แสดงการบรรยายเชิงโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1	52
45. แสดงโครงสร้างของแพ็คเกจ	53
46. กำหนดไลบรารี	54
47. แสดงแพ็คเกจ basic_utilities	54
48. แสดงการบรรยายพฤติกรรมของหน่วยความจำภายใน EEPROM	55
49. แสดงการบรรยายพฤติกรรมองค์ประกอบหลักของ EEPROM	56
50. บล็อกแสดงการจำลองการทำงานด้วย QuickSim II	58
51. ตัวอย่างโปรแกรมสร้างสัญญาณ และตารางแสดงลักษณะของสัญญาณที่ได้	59
52. โปรแกรมสร้างสัญญาณทดสอบการอ่านรหัสลับที่แอดเดรส 37 D	60
53. Waveform ของการอ่านรหัสลับที่แอดเดรส 37 D	62
54. บล็อกแสดงองค์ประกอบต้นแบบฮาร์ดล็ค และการเชื่อมต่อขาสัญญาณ	63
55. โครงสร้างภายนอกและการจัดขาของ HY93C46	64
56. โครงสร้างภายในของ EEPROM HY93C46	65
57. แผนภูมิเวลาค่าสั่งของการอ่านข้อมูล	67
58. แผนภูมิเวลาค่าสั่งของการเขียนข้อมูล	68
59. แผนภูมิเวลาค่าสั่งของการลบข้อมูล	68
60. แผนภูมิเวลาค่าสั่งของการลบข้อมูลทั้งหมด	68
61. แผนภูมิเวลาค่าสั่งของการเขียนข้อมูลทั้งหมด	68
62. แผนภูมิเวลาค่าสั่ง อีเนเบิล/ดิสเอเบิล การเขียนและลบข้อมูล	68
63. โครงสร้างภายนอกและการจัดขาของ PROM XC17xxx	69

64. โครงสร้างภายในอย่างง่ายของ PROM XC17xxx	70
65. แสดงการจัดเรียงเซลล์เป็นเมทริกซ์ของ FPGA ตระกูล XC3000	74
66. บล็อกแสดงวงจรภายในของ CLB	75
67. ลำดับขั้นตอนการคอนฟิกเมื่อเริ่มป้อนแหล่งไฟเข้าไอซี และการโปรแกรมใหม่	79
68. แสดงการต่อใช้งานในโหมด Slave Serial	80
69. รูปแบบข้อมูลโปรแกรมคอนฟิก (Configuration Bitstream)	81
70. แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในโหมด Slave Serial	81
71. ตัวอย่างบางส่วนของข้อมูลที่ใช้โปรแกรมคอนฟิก	81
72. แสดงการต่อใช้งานในโหมด Master Serial	82
73. แสดงการต่อใช้งานในโหมด Master Parallel	83
74. โฟล์วชาร์ตการออกแบบวงจรโดยใช้ ViewLogic และ Xilinx	84
75. ตัวอย่างสัญญาณที่ป้อนให้ฮาร์ดลอค	87
76. รูปแบบคำสั่งที่ป้อนตามสัญญาณคล็อก	87
77. โปรแกรมสร้างสัญญาณผ่านพอร์ตขนาน	88
78. โฟล์วชาร์ตการเขียนรหัสเฉพาะ	89
79. โฟล์วชาร์ตการเขียนรหัสครั้งแรก	90
80. การทำงานของโปรแกรม HL_SHELL	91
81. การทำงานของโปรแกรม DECODE	92
82. วงจรต้นแบบฮาร์ดลอค	93
83. ชื่อขาสัญญาณพอร์ตขนาน	94
84. การเชื่อมต่อต้นแบบฮาร์ดลอคกับพอร์ตขนาน	95
85. ต้นแบบฮาร์ดลอคพร้อมคอนเน็คเตอร์	95
86. ลายพิมพ์ต้นแบบฮาร์ดลอค	96
87. ตัวอย่างรูปแบบข้อมูลที่ใช้เขียนลงฮาร์ดลอค	97
88. ข้อมูลส่วนบางส่วนที่สร้าง	98
89. ข้อมูลแบบ Hex บางส่วนของโปรแกรม CW.EXE ก่อนลอค	99
90. ข้อมูลแบบ Hex บางส่วนของโปรแกรม CW.EXE หลังลอค	101

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ข้อมูลทางธุรกิจเป็นสิ่งสำคัญและมีคุณค่า โดยเฉพาะในยุคของการแข่งขัน สิ่งที่สำคัญคือการป้องกันข้อมูลไม่ให้ตกไปอยู่ในมือของคู่แข่งทางการค้า การป้องกันข้อมูลในระดับนี้เป็นระดับที่หลายธุรกิจให้ความสำคัญ การป้องกันนั้นนอกจากจะป้องกันในระดับคู่แข่งทางการค้าแล้ว ในบางครั้งฝ่ายบริหารยังต้องป้องกันความลับของข้อมูลไม่ให้รั่วไหลลงไปในระดับปฏิบัติงานอีกด้วย

หากข้อมูลที่ถูกเก็บไว้มีคุณค่าและมีความสำคัญ การโจรกรรมหรือการแอบค้นข้อมูลก็จะมีมากขึ้น และที่สำคัญคือบางแห่งอาจจะมีการเปลี่ยนแปลงแก้ไขข้อมูลโดยไม่ได้รับอนุญาต การกระทำในลักษณะนี้สร้างความเสียหายให้กับเจ้าของข้อมูลได้ วิธีการป้องกันบุคคลที่ไม่มีสิทธิในการเข้าถึงข้อมูลได้หลายวิธี มีการนำมาประยุกต์ใช้ในระดับต่างๆ กันอยู่มาก เช่น ใช้โปรแกรมควบคุมอยู่ในโปรแกรมประยุกต์ ใช้รหัสผ่าน หรือใช้วิธีการเข้ารหัสลับ ซึ่งทั้งสามวิธีอาจกล่าวได้ว่าเป็นการป้องกันโดยใช้ซอฟต์แวร์ (Software-Lock) ที่สามารถป้องกันข้อมูลได้เพียงระดับหนึ่งเท่านั้น เพื่อให้ความสามารถในการป้องกันข้อมูลสูงขึ้น จึงได้ออกแบบอุปกรณ์เพิ่มขึ้นมาป้องกันข้อมูลร่วมกับวิธีการป้องกันที่กล่าวมาข้างต้น (Hardware-Lock)

1.2 วัตถุประสงค์และแนวความคิดในการวิจัย

จากการที่อุตสาหกรรมซอฟต์แวร์ในประเทศไทยกำลังมีการพัฒนามากขึ้น ซึ่งหลายๆ หน่วยงาน หรือบริษัทต่างแข่งขันกันผลิตซอฟต์แวร์ออกมาสู่ผู้บริโภค แต่ปัญหาสำคัญที่ทำให้ซอฟต์แวร์ที่ผลิตออกมาสู่ผู้บริโภคมีความเจริญเติบโตไม่มากนัก ทั้งนี้เนื่องมาจากมีการคัดลอกและเผยแพร่ซอฟต์แวร์กันอย่างแพร่หลายในประเทศไทย ดังนั้นผู้ผลิตซอฟต์แวร์จึงต้องหาวิธีต่างๆ ที่จะเพิ่มความปลอดภัยให้กับซอฟต์แวร์ที่ได้ผลิตขึ้นเพื่อไม่ให้ถูกคัดลอก และ เผยแพร่ออกไปได้โดยไม่ได้รับอนุญาต วิธีหนึ่งที่ผู้ผลิตนำมาใช้ก็คือ การใช้ฮาร์ดแวร์ล็อก (Hardware-Lock) ซึ่งเป็นวิธีการป้องกันการคัดลอกซอฟต์แวร์ โดยต้องให้อุปกรณ์ที่มักจะเรียกกันว่าฮาร์ดล็อก (Hardlock) อันหนึ่งไปพร้อมกับซอฟต์แวร์ที่จำหน่ายให้กับผู้ใช้ เพื่อให้ผู้ใช้ต่ออุปกรณ์ตัวนี้เข้ากับพอร์ตแบบขนาน (Parallel Port) ในขณะที่ใช้ซอฟต์แวร์

หลักการทำงานของฮาร์ดล็อกก็คือ ทุกครั้งที่ผู้ใช้ (User) เริ่มเรียกใช้โปรแกรม (Load) หรือระหว่างการรันโปรแกรมนั้นอยู่ จะมีโปรแกรมส่วนหนึ่งที่แฝงตัวอยู่ในโปรแกรมนั้นทำการติดต่อกับฮาร์ดล็อกทางพอร์ตขนานโดยการตรวจสอบข้อมูลบางอย่างที่อยู่ในฮาร์ดล็อก ถ้าตัวฮาร์ดล็อกตอบสนองข้อมูลได้อย่างถูกต้องการใช้โปรแกรมจึงจะดำเนินต่อไป มิฉะนั้นโปรแกรมจะหยุดการทำงานลงทันที

ในปัจจุบันอุปกรณ์ฮาร์ดล็อกได้ผลิตออกมาใช้กันอย่างแพร่หลาย ความสามารถในการป้องกันการ

คัดลอกมีต่างกันหลายระดับ โดยถ้าเป็นในระดับต่ำที่ง่ายต่อการแอบดูข้อมูลที่ติดต่อกันระหว่างคอมพิวเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

และฮาร์ดลอค โดยนำเครื่องมือบันทึกสัญญาณเช่น ลอจิกอานาไลเซอร์ (Logic Analyzer) หรือสโตเรสโคป (Storage Scope) เพื่อจำลองสัญญาณตอบรับเสมือนว่าเป็นสัญญาณตอบรับจากตัวฮาร์ดลอคให้กับคอมพิวเตอร์ ดังนั้นจึงมีความจำเป็นอย่างยิ่งที่จะต้องออกแบบสร้างฮาร์ดลอคที่มีความสามารถป้องกันการคัดลอกได้ดีและยากต่อการแอบดูข้อมูลที่ติดต่อกันระหว่างคอมพิวเตอร์และฮาร์ดลอค ซึ่งวิทยานิพนธ์นี้นำเสนอการออกแบบวงจรเป็นวงจรรวมเฉพาะกิจ และมีฟังก์ชันการทำงานที่มีการส่งสัญญาณได้อย่างไม่ซ้ำแบบเพื่อให้ยากต่อการสังเกตรูปแบบของสัญญาณ การมีอุปกรณ์ที่สามารถป้องกันการเผยแพร่โปรแกรมได้ดีจะส่งผลให้อุตสาหกรรมซอฟต์แวร์ในประเทศมีการแข่งขันและพัฒนาสูงขึ้น ส่งผลให้ผู้ผลิตซอฟต์แวร์ในประเทศมีมากขึ้นตาม นอกจากนั้นยังทำให้เกิดการพัฒนาเทคโนโลยีวิทยาการทางรหัสลับได้ต่อไป

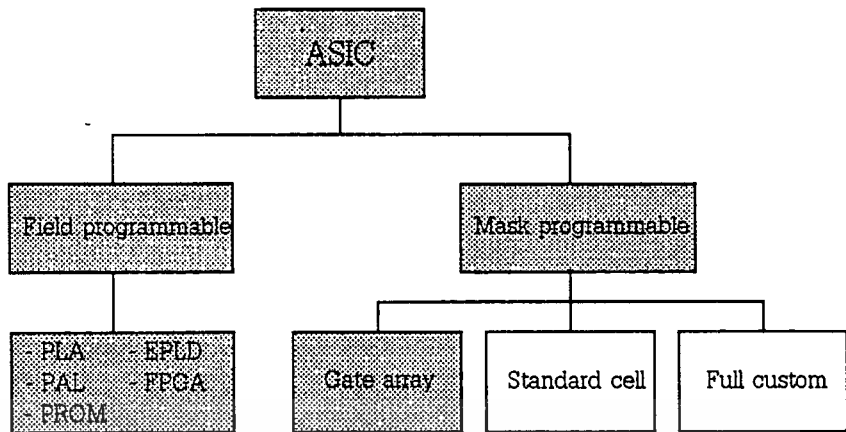
1.3 วิธีที่ใช้ในการดำเนินการวิจัย

ในวิทยานิพนธ์ฉบับนี้แบ่งการวิจัยออกเป็น 4 ขั้นตอนคือ ขั้นตอนแรกทำการออกแบบวงจรป้องกันการเข้าถึงข้อมูลของหน่วยความจำเป็นผังวงจรไฟฟ้า และเขียนบรรยายลักษณะพฤติกรรมของหน่วยความจำชนิด EEPROM ที่ใช้เก็บรหัสลับด้วยภาษา VHDL ซึ่งไม่มีอยู่ในไลบรารี นำทั้งสองส่วนมาจำลองการทำงานร่วมกันเพื่อให้ได้ผังวงจรไฟฟ้าที่ทำงานได้ตามความต้องการ ตลอดขั้นตอนนี้ใช้ซอฟต์แวร์ของ Mentor Graphics บนเครื่อง Engineering Workstation ขั้นตอนที่สองนำผังวงจรไฟฟ้าที่ได้จากขั้นตอนแรกมาเขียนเป็นผังวงจรไฟฟ้า และจำลองการทำงานวงจรอีกครั้งด้วยซอฟต์แวร์ของ ViewLogic ขั้นตอนที่สามใช้ซอฟต์แวร์ของ Xilinx โปรแกรมการทำงานของวงจรป้องกันการเข้าถึงข้อมูลที่ทำงานได้ตามความต้องการลงบน Serial PROM เพื่อทำงานร่วมกับ FPGA (Field Programmable Gate Array) ในขั้นตอนสุดท้ายทดสอบวงจรป้องกันการเข้าถึงข้อมูล โดยใช้โปรแกรมเขียนรหัสเฉพาะ (ID_GEN) โปรแกรมเขียนรหัสลับ (CODE_GEN) โปรแกรมอ่านรหัสลับ (CODE_RD) โปรแกรมอ่านรหัสสุ่ม (RAND_RD) และโปรแกรมประยุกต์ที่ใช้ป้องกันการคัดลอกโปรแกรม .EXE (HL_SHELL) ที่เขียนด้วยซอฟต์แวร์ของ Borland C ตั้งแต่ขั้นตอนที่สองเป็นต้นมาพัฒนาบนเครื่องไมโครคอมพิวเตอร์

1.4 วงจรรวมเฉพาะกิจชนิดต่างๆ

วงจรรวมเฉพาะกิจ (ASIC : Application Specific Integrated Circuit) เกิดจากการที่ผู้ออกแบบต้องการวงจรที่มีลักษณะแตกต่างไปจากที่มีอยู่ในท้องตลาด การออกแบบสร้างอาจใช้วิธีการนำวงจรรวมพื้นฐานตั้งแต่ขนาดเล็กถึงขนาดใหญ่เข้ามาบรรจุในวงจรรวมตัวเดียวกัน หรือออกแบบตั้งแต่ระดับทรานซิสเตอร์ประกอบกันเป็นวงจรรวมก็ได้

วงจรรวมเฉพาะกิจสามารถแบ่งตามวิธีการสร้างได้เป็น 2 กลุ่มคือ Field Programmable และ Mask Programmable ดังแสดงในรูปที่ 1.1



รูปที่ 1.1 ผังแสดงประเภทของวงจรรวมเฉพาะกิจ

1.4.1 Field Programmable

ผู้ใช้งานสามารถออกแบบ และสร้างวงจรที่ต้องการโปรแกรมลงบนตัวอุปกรณ์ได้เองโดยไม่ต้องพึ่งโรงงานเพื่อผลิต โดยเฉพาะอย่างยิ่งในปัจจุบันนี้มีเครื่องมือที่ช่วยในการออกแบบ และสร้างวงจรที่ทำงานร่วมกับไมโครคอมพิวเตอร์ที่มีความสามารถสูงในการพัฒนาตั้งแต่ขั้นการออกแบบ การจำลองการทำงาน โปรแกรมวงจรรลงบนอุปกรณ์ จนถึงขั้นแปลงวงจรเป็นหน้ากากของวงจรรวม (Mask Layout) อุปกรณ์ประเภทนี้มีหลายประเภท ที่มีความสามารถวงจรได้สูงมี Field Programmable Gate Array (FPGA) , Erase Programmable Logic Device (EPLD)

1.4.2 Mask Programmable

การใช้วงจรรวม ASIC ในเชิงพาณิชย์ จำเป็นต้องใช้วงจรรวม ASIC แบบ Mask Programmable เนื่องจากต้นทุนต่อหนึ่งตัวต่ำกว่าวงจรรวมแบบ Field Programmable ASIC ในกรณีที่ปริมาณการผลิตสูงนับพันนับหมื่นตัวขึ้นไป ยกตัวอย่างเช่น วงจรรวม FPGA XC3042A ตัวหนึ่งมีราคาในประเทศไทยประมาณ 1,300 บาท ในขณะที่ถ้าผลิตวงจรรวมที่ทำงานได้เหมือนกับวงจรที่โปรแกรมลงบนตัว FPGA โดยใช้วงจรรวมแบบ Mask Programmable แล้ว ราคาจะถูกลง ดังนั้นการใช้วงจรรวมแบบ Mask Programmable จึงมีบทบาทสำคัญในการผลิตสินค้าอิเล็กทรอนิกส์ในเชิงพาณิชย์ในปัจจุบัน

วงจรรวม Mask Programmable ASIC นี้ หลังจากผู้ใช้ออกแบบวงจรและตรวจสอบการทำงานจนเป็นที่พอใจแล้ว ต้องส่งให้ผู้ผลิตทำการเจียรสาร ไม่สามารถโปรแกรมด้วยตัวเองเหมือนวงจรรวมแบบ Field Programmable ช่วงเวลาในการผลิตออกใช้งานจึงใช้เวลานานนับเดือนและค่าใช้จ่ายเบื้องต้นสูง ซึ่งเป็นสิ่งสำคัญที่ทำให้การออกแบบและพัฒนาวงจรให้เป็น Mask Programmable ในประเทศไม่สำเร็จเท่าที่ควร วงจรรวมแบบ Mask Programmable ในปัจจุบันได้แก่ Gate array, Standard cell และ Full custom

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.5 เทคโนโลยีของต้นแบบฮาร์ดลอค

เนื่องจากวงจรแต่ละวงจรมีความเหมาะสมกับเทคโนโลยีที่ต่างกัน ประกอบกับการมีเทคโนโลยีให้เลือกใช้ได้มากมาย ทำให้การออกแบบวงจรรวมฮาร์ดลอคจำเป็นต้องเลือกใช้เทคโนโลยีให้มีความเหมาะสมมากที่สุด โดยพิจารณาจากคุณสมบัติที่สำคัญดังต่อไปนี้

1. ความสะดวกในการใช้งาน

สามารถทำงานได้เมื่อต่อเข้ากับพอร์ตนานโดยไม่ต้องใช้แหล่งจ่ายไฟจากภายนอก ดังนั้นเทคโนโลยีที่ใช้สร้างเป็นต้นแบบฮาร์ดลอคจะต้องใช้กระแสน้อย เพราะหาสัญญาณของพอร์ตนานไม่มีขาไฟ +5 Volts กระแสไฟฟ้าที่นำมาใช้เลี้ยงวงจรตั้งมาจากสัญญาณต่างๆ

2. ความเหมาะสมในการสร้างเป็นวงจรต้นแบบ (Prototype)

- เทคโนโลยีนั้นจะต้องมีความสามารถจำนวนเกตได้สูง และมีแนวโน้มที่จะสูงขึ้นได้อีกในอนาคต
- ราคาไม่สูงมากนัก และมีแนวโน้มที่จะต่ำลงได้อีกในอนาคตเนื่องจากผู้ใช้มีมากขึ้น
- ความเร็วในการทำงานสูง

3. ความเหมาะสมในการพัฒนาผลเป็นเชิงพาณิชย์

สามารถนำวงจรต้นแบบที่ได้ผลผลิตเป็นเชิงพาณิชย์ได้สะดวก และรวดเร็ว

จากคุณสมบัติดังที่กล่าวมาข้างต้น เทคโนโลยีที่ใช้ในการออกแบบสร้างต้นแบบวงจรฮาร์ดลอคจึงเป็น FPGA (Field Programmable Gate Array) ซึ่งมีโครงสร้างภายในเป็น LCA (Logic Cell Array) คล้ายกับวงจรรวมประเภทเกตอะเรย์ (Mask Programmable : Gate Array)

1.6 ซอฟต์แวร์ที่ใช้ออกแบบฮาร์ดลอค

ปัจจุบันซอฟต์แวร์ที่ใช้ออกแบบมีจำนวนมาก และความสามารถในการใช้งานมีมากน้อยต่างกัน ทำให้การเลือกใช้ซอฟต์แวร์ในการออกแบบวงจรจะต้องละเอียดรอบคอบพอสมควร โดยพิจารณาตามคุณสมบัติต่อไปนี้

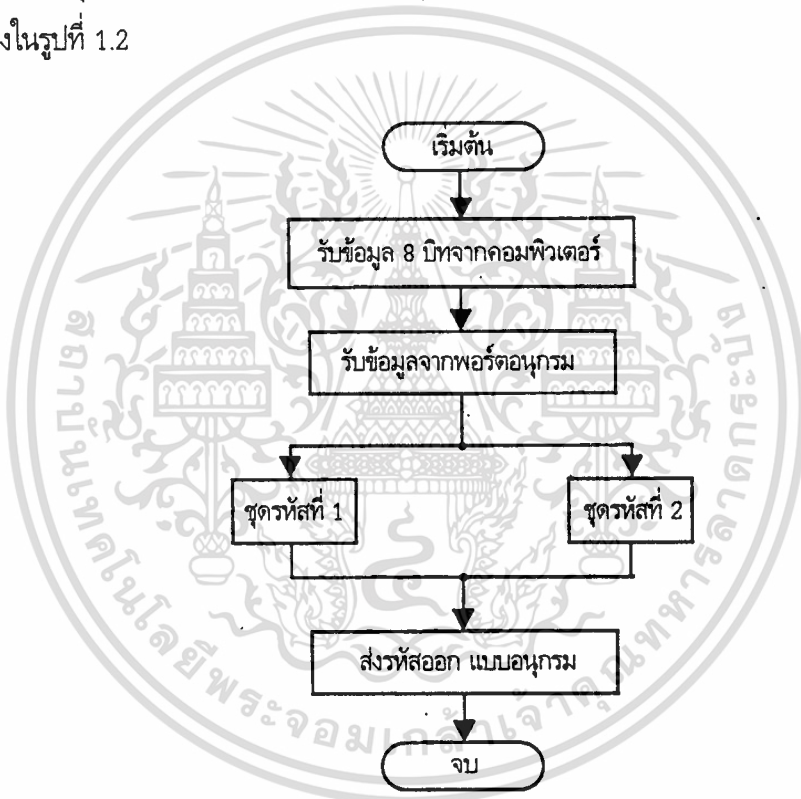
ต้องสามารถออกแบบวงจรด้วยการเขียนเป็นผังวงจรไฟฟ้า (Schematic) และที่สำคัญคือ สามารถเขียนอธิบายการทำงานของวงจรด้วยภาษา VHDL (VHSIC Hardware Description Language) ซึ่งจะนำมาใช้แทนการเขียนเป็นผังวงจรไฟฟ้า เพราะสะดวกและง่ายต่อการเขียน แก้ไข ประสิทธิภาพในการออกแบบวงจรดีกว่าการเขียนเป็นผังวงจรไฟฟ้า จำลองการทำงาน (Simulation) ได้ดี ที่สำคัญที่สุดสามารถทดสอบการทำงานของวงจรได้ด้วยไอซีที่โปรแกรมการทำงานของวงจรมานั้นได้ และสามารถสังเคราะห์ผังวงจรไฟฟ้าจากวงจรที่ออกแบบได้ (ยุบจำนวนเกต)

เนื่องจากในขณะที่ทำการศึกษา ศูนย์เทคโนโลยีอิเล็กทรอนิกส์ และคอมพิวเตอร์แห่งชาติ (NECTEC) ได้สนับสนุน ให้มหาวิทยาลัยของรัฐใช้ซอฟต์แวร์ ViewLogic ซึ่งผู้วิจัยเห็นว่ามีความสอดคล้องกับที่กล่าวมาข้างต้น

1.7 เปรียบเทียบกับฮาร์ดลอคแบบเดิม

ปัจจุบันฮาร์ดลอคที่สร้างในประเทศไทย มีหลายแบบ การเชื่อมต่อกับพอร์ตไมโครคอมพิวเตอร์ต่างกัน และที่สำคัญคือเทคโนโลยีของวงจรรวมที่ใช้ฮาร์ดลอค เช่นสร้างจากวงจรรวมพื้นที่มีขายทั่วไป ใช้ไมโครคอนโทรลเลอร์ทำงานด้วยโปรแกรมที่ฝังอยู่ในตัว [17] หรือจะออกแบบเป็นวงจรรวมเฉพาะกิจอย่างเดียวกับวิทยานิพนธ์ฉบับนี้ [18] ซึ่งในหัวข้อนี้จะยกตัวอย่างและอธิบายการทำงานของสองแบบแรกอย่างคร่าวๆ เพื่อเปรียบเทียบกับแบบที่สาม

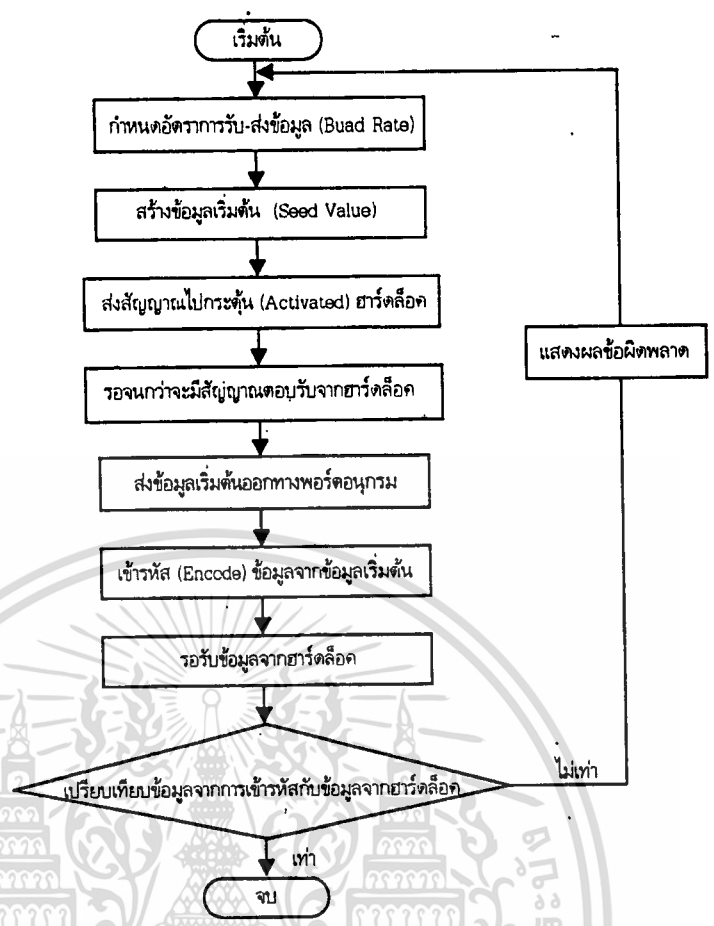
แบบที่หนึ่งออกแบบฮาร์ดลอคให้ใช้งานกับพอร์ตขนาน โดยใช้วงจรรวมพื้นที่มีขายในท้องตลาดทั่วไป รับข้อมูลแบบอนุกรมทางขา D1 หลังจากที่ได้รับข้อมูลแล้วจะเลือกชุดรหัสที่ใช้ดิฟเฟอเรนเชียล 8 บิตสองตัว กำหนดค่ารหัส สุดท้ายส่งรหัสที่อ่านได้เป็นแบบอนุกรมออกที่ขาสัญญาณ -BUSY การทำงานของฮาร์ดลอคแบบนี้แสดงในรูปที่ 1.2



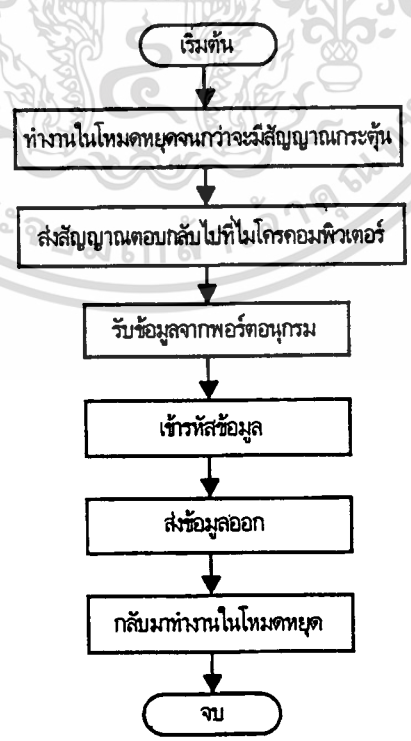
รูปที่ 1.2 โฟลว์ชาร์ตการทำงานของฮาร์ดลอค

แบบที่สองเป็นฮาร์ดลอคที่ต่ออยู่กับพอร์ตอนุกรมเป็นฮาร์ดลอคที่มีการรับส่งข้อมูลกับคอมพิวเตอร์เป็นแบบอนุกรม ออกแบบให้ทำการรับข้อมูลรหัสผ่าน (Password) จากสัญญาณ TX จากนั้นข้อมูลรหัสผ่านที่รับเข้ามาจะถูกนำมาผ่านการคำนวณด้วยฟังก์ชันอันหนึ่งซึ่งได้กำหนดไว้แล้วในโปรแกรมที่ฝังอยู่ในตัวไมโครคอนโทรลเลอร์ของฮาร์ดลอค หลังจากนั้นฮาร์ดลอคจะนำเอาผลลัพธ์ที่ได้จากการคำนวณส่งกลับมาให้คอมพิวเตอร์โดยผ่านทางขาสัญญาณ RX ซึ่งมีการทำงานบนเครื่องไมโครคอมพิวเตอร์เป็นดังรูปที่ 1.3 และมีการทำงานบนฮาร์ดลอค (ไมโครคอนโทรลเลอร์) เป็นดังรูปที่ 1.4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 1.3 โฟลว์ชาร์ตการทำงานบนเครื่องคอมพิวเตอร์



รูปที่ 1.4 โฟลว์ชาร์ตการทำงานบนฮาร์ดลิค

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อเปรียบเทียบกับระหว่างสามแบบจะเห็นว่า แบบที่หนึ่งนั้นง่ายต่อการคัดลอกฮาร์ดแวร์ภายในมากที่สุด และสามารถแอบดูสัญญาณที่ติดต่อกันระหว่างพอร์ตกับฮาร์ดลอคได้ง่ายเนื่องจากมีคำสั่งเพียง 8 บิต อีกทั้งการทำงานของฮาร์ดลอคก็ง่ายมาก ส่วนแบบที่สองนั้นก็สามารคัดลอกฮาร์ดแวร์ภายในได้เช่นเดียวกัน เชื่อมต่อกับพอร์ตอนุกรมซึ่งมีสัญญาณรับส่งข้อมูลน้อยเส้นจึงสามารถรู้ได้ว่าสัญญาณไหนเป็นอินพุต หรือเอาพุต และยังไม่สามารถต่ออุปกรณ์ใดๆต่อจากฮาร์ดลอคได้อีกเมื่อต่อฮาร์ดลอคเข้าไปแล้ว แบบนี้ยอมรับว่าความสลับซับซ้อนในการทำงานมีมากเพราะทำงานด้วยซอฟต์แวร์ภายใน แต่จะเสียเวลาที่ใช้ในการทำงานมาก ส่วนแบบที่สามเมื่อพัฒนาผลิตเป็นเชิงพาณิชย์แล้วจะคัดลอกฮาร์ดแวร์ภายในได้ยาก การใช้งานเชื่อมต่อกับพอร์ตอนุกรมซึ่งออกแบบให้สามารถต่ออุปกรณ์ต่อท้ายฮาร์ดลอคได้ คำสั่งที่ใช้ติดต่อกับฮาร์ดลอคมี 73 บิต การส่งสัญญาณควบคุมออกทางขาสัญญาณ D0-D3 เป็นแบบอนุกรม เวลาในการทำงานสูงเนื่องจากไม่ได้ทำงานด้วยซอฟต์แวร์ภายใน ส่วนวงจรภายในที่ออกแบบมีความซับซ้อนมากกว่าฮาร์ดลอคแบบที่หนึ่งมากเมื่อคิดเป็นจำนวนเกตแล้วมีประมาณ 1,356 เกต

1.8 ภาษา VHDL (VHSIC Hardware Description Language)

VHDL เป็นภาษาที่ใช้บรรยายหรืออธิบายรูปแบบการทำงาน และความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์เป็นโปรแกรม เป็นภาษาที่ถูกนำมาใช้เป็นประโยชน์ในการออกแบบอุปกรณ์ฮาร์ดแวร์ต่างทั้งในระดับเกตจนถึงวงจรที่มีความซับซ้อน ทั้งนี้เนื่องจาก VHDL เป็นภาษาที่มีความสามารถในการเขียนเลียนแบบการทำงานของอุปกรณ์

การออกแบบวงจรในอดีตนั้นออกแบบวงจรส่วนมากจะออกแบบวงจรเป็น ผังวงจรไฟฟ้า (Schematic) การออกแบบในลักษณะนี้เหมาะสำหรับออกแบบวงจรที่มีขนาดเล็ก แต่ในปัจจุบันความต้องการออกแบบวงจรที่มีขนาดใหญ่ และซับซ้อนมีมากขึ้น จึงจำเป็นต้องมีวิธีหนึ่ง หรือหนทางใหม่ๆ เพื่อนำมาใช้ในการจัดการกับการออกแบบวงจรในลักษณะนี้ และ VHDL ก็เป็นวิธีหนึ่งที่นำมาใช้ในการจัดการกับปัญหานี้

สำหรับวงจรฮาร์ดลอคนั้น ได้ออกแบบเป็นผังวงจรไฟฟ้าเนื่องจากในขณะที่ยังออกแบบซอฟต์แวร์ที่สามารถใช้ภาษา VHDL ได้ยังไม่แพร่หลายบนเครื่องไมโครคอมพิวเตอร์

1.9 การเรียบเรียงเนื้อหาภายในวิทยานิพนธ์

เนื้อหาภายในวิทยานิพนธ์ฉบับนี้ได้เรียบเรียงไว้มีทั้งหมด 5 บท บทที่ 1 กล่าวถึงความเป็นมา แนวความคิด การดำเนินการวิจัย และการเลือกใช้เทคโนโลยีพร้อมกับซอฟต์แวร์ที่ใช้ออกแบบฮาร์ดลอค บทที่ 2 กล่าวถึงการออกแบบวงจรฮาร์ดลอคเป็นผังวงจรไฟฟ้า และการออกแบบหน่วยความจำชนิด EEPROM เป็นภาษา VHDL รวมทั้งผลการจำลองการทำงานของระบบที่มีวงจรฮาร์ดลอค และหน่วยความจำ บทที่ 3 เป็นการพัฒนางจรที่ออกแบบได้จากบทที่ 2 เพื่อสร้างเป็นต้นแบบฮาร์ดลอค เนื้อหาภายในประกอบด้วยรายละเอียดของไอซีแต่ละตัวที่ใช้สร้างต้นแบบวงจรฮาร์ดลอค การใช้ซอฟต์แวร์ทดสอบการทำงานของวงจรต้นแบบฮาร์ดแวร์ของต้นแบบฮาร์ดลอค และซอฟต์แวร์ที่ใช้ควบคุมการทำงานของฮาร์ดลอค บทที่ 4 กล่าวถึงวิธีการสร้าง และทดสอบการทำงานจริงของต้นแบบฮาร์ดลอค บทที่ 5 สรุปผลการวิจัย และข้อเสนอแนะ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวกแบ่งออกเป็น 3 บท ภาคผนวก ก. แสดงวงจรของฮาร์ดลิคแต่ละส่วน ภาคผนวก ข. โปรแกรมที่ใช้ทดสอบการทำงานของฮาร์ดลิค ภาคผนวก ค. ข้อมูลของ FPGA XC3xxx PROM XC1736 และ EEPROM HY93C46



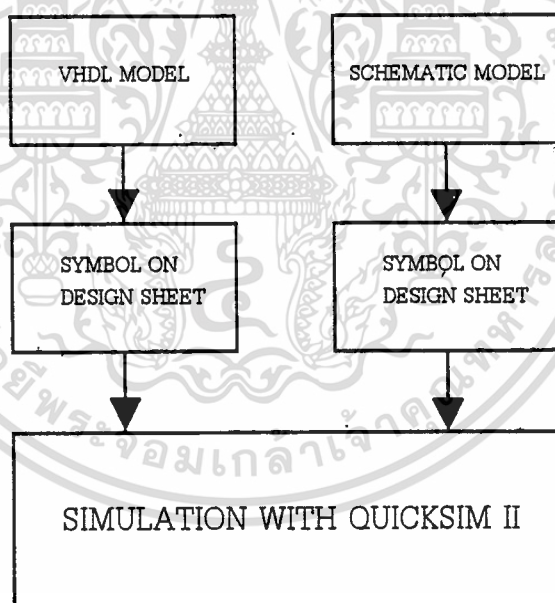
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2

การออกแบบวงจรฮาร์ดลอคเป็น Schematic และ VHDL

วงจรฮาร์ดลอค หมายถึงวงจรรวมที่ออกแบบมาเพื่อใช้ป้องกันการเข้าถึงข้อมูลภายในหน่วยความจำชนิด EEPROM ซึ่งในบทนี้ได้ออกแบบวงจรฮาร์ดลอคเป็นผังวงจรไฟฟ้า (Schematic) ส่วนหน่วยความจำชนิด EEPROM ได้ออกแบบด้วยภาษา VHDL [5,6] เนื่องจากวงจรที่จะนำไปออกแบบสร้างเป็นวงจรรวมเฉพาะกิจคือวงจรฮาร์ดลอคเท่านั้น ส่วนหน่วยความจำชนิด EEPROM เป็นไอซีที่มีขายอยู่ในท้องตลาดอยู่แล้ว แต่ที่ต้องออกแบบส่วนของหน่วยความจำด้วยก็เพราะว่าต้องการจำลองการทำงานร่วมกันระหว่างวงจรฮาร์ดลอค กับหน่วยความจำดังแสดงในรูปที่ 2.1 ทั้งนี้เพื่อให้ได้วงจรฮาร์ดลอคที่ทำงานร่วมกับหน่วยความจำได้อย่างถูกต้องมากที่สุด

วงจรฮาร์ดลอคที่ออกแบบเป็นแผงวงจรไฟฟ้า และหน่วยความจำที่ออกแบบโดยใช้ภาษา VHDL สร้างจากโปรแกรม Design Architecture (DA) [3] ส่วนในการจำลองการทำงานใช้โปรแกรม QuickSim II [4] โดยรับสัญญาณอินพุตจากโปรแกรมที่เขียนด้วยภาษา MISL (Mentor Interactive Stimulus Language) [7]



รูปที่ 2.1 บล็อกแสดงขั้นตอนการจำลองการทำงาน

2.1 วงจรช่วยเสริมการทำงานให้กับฮาร์ดลอค และทฤษฎีของ DES

เพื่อให้วงจรฮาร์ดลอคที่จะออกแบบมีประสิทธิภาพสูง จึงได้นำวงจรประเภทต่างๆ มาเป็นส่วนหนึ่งของวงจรฮาร์ดลอคซึ่งมีวงจรเข้ารหัสและถอดรหัสลับ เพื่อเข้ารหัสและถอดรหัสลับที่อยู่ในหน่วยความจำชนิด EEPROM และวงจรสร้างข้อมูลสุ่มแบบเทียม (Pseudorandom Sequences) ที่มีไว้เพื่อสร้างข้อมูลหลอกผู้แอบดูข้อมูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

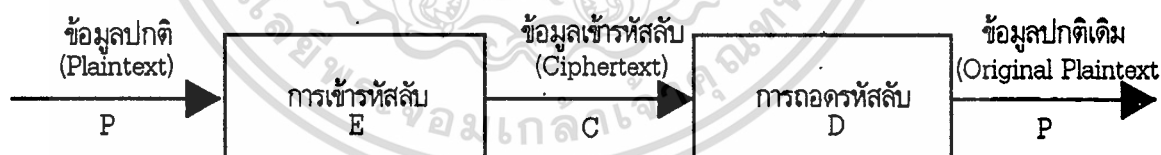
2.1.1 วงจรเข้า และถอดรหัส

ในการส่งข้อมูลข่าวสารจากเครื่องคอมพิวเตอร์ S ออกไปทางสายส่งสัญญาณพอร์ตนาน (Parallel Port) T เพื่อไปยังฮาร์ดดิสก์ R หากมีบุคคลภายนอก O มาทำการขัดขวาง หรือโจรกรรมข้อมูลข่าวสารนั้น เขาสามารถกระทำได้หลายกรณี คือ

- ป้องกันไม่ให้ข้อมูลนั้นไปถึงฮาร์ดดิสก์ R (Interruption)
- ขโมยข้อมูลที่ส่งออกจากพอร์ตนาน T (Interception)
- ยึด และแปรเปลี่ยนข้อมูลข่าวสาร (Modification)
- สอดแทรกข้อมูล เสมือนข้อมูลที่ส่งมาจากคอมพิวเตอร์ S (Fabrication)

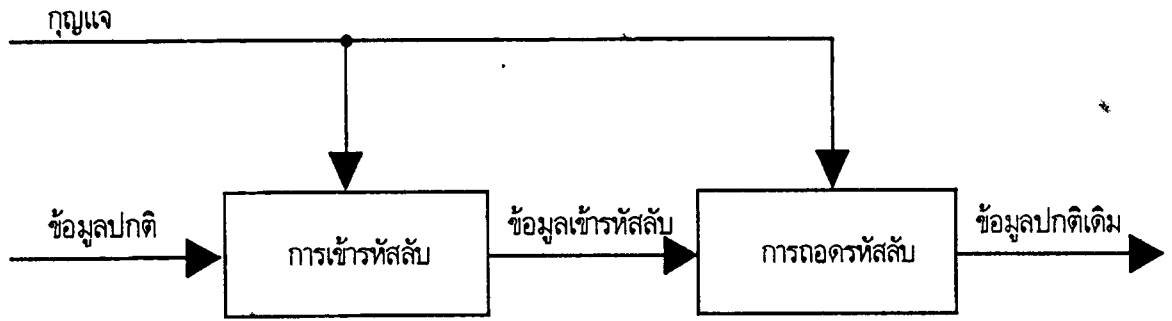
การเข้ารหัสลับ (Encryption) เป็นวิธีป้องกันข้อมูลแบบหนึ่งที่มีการแปลงเปลี่ยนข้อมูลเดิมไปเป็นข้อมูลชุดใหม่ ซึ่งชุดข้อมูลใหม่เป็นข้อมูลที่ใช้ส่งออกจากพอร์ตนานเพื่อป้องกันผู้แอบดูข้อมูล ผู้เข้ารหัสลับจึงจะเป็นผู้ทำการถอดรหัส (Decryption) เปลี่ยนข้อมูลที่เข้ารหัสไปเป็นข้อมูลชุดเดิม ในระบบที่ประกอบด้วยทั้งส่วนการเข้ารหัสลับ และการถอดรหัสลับ ดังรูปที่ 2.2 เรียกว่า ระบบสร้างรหัสลับ (Cryptosystem)

ข้อมูลปกติ (Plaintext) เขียนย่อว่า P จะเป็นการเรียงต่อกันของข้อมูลเป็นบิตๆ $P=[p_1, p_2, p_3, \dots, p_n]$ และข้อมูลที่ผ่านการเข้ารหัสลับ (Ciphertext) เขียนได้เป็น $C=[c_1, c_2, c_3, \dots, c_m]$ ขั้นตอนการเข้ารหัสลับก็คือ การแปลงระหว่างข้อมูลปกติไปเป็นข้อมูลเข้ารหัสลับ จะเขียนแทนด้วยสมการได้เป็น $C=E(P)$ ขณะเดียวกันขั้นตอนการถอดรหัสลับจะเขียนสมการได้เป็น $P=D(C)$ และระบบการสร้างรหัสลับเขียนสมการรวมได้เป็น $P=D(E(P))$

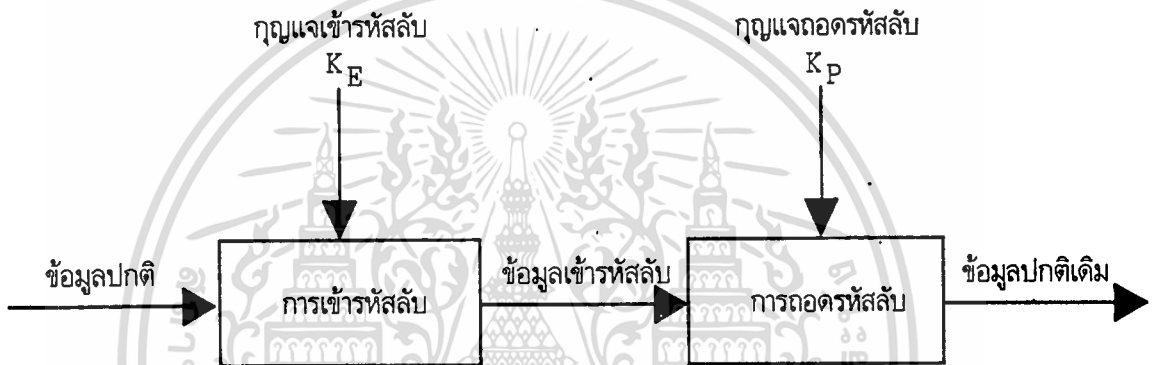


รูปที่ 2.2 ระบบการสร้างรหัสลับ

ขั้นตอนการเข้ารหัสลับได้หลายแบบ บางแบบจะต้องมีคีย์หรือกุญแจ (Key) เข้ามาร่วมกับข้อมูลปกติเพื่อไปเป็นข้อมูลเข้ารหัสลับ กรณีนี้สมการเข้ารหัสลับจะเป็น $C=E(K,P)$ โดยกุญแจ K จะเป็นค่าหรือขั้นตอนเฉพาะหนึ่ง ถ้ากุญแจนี้ใช้ทั้งการเข้ารหัสลับ และถอดรหัสลับดังแสดงในรูปที่ 2.3 สมการสร้างรหัสลับจะเขียนได้เป็น $P=D(K,E(K,P))$ แต่ในบางกรณีกุญแจการเข้ารหัสลับและการถอดรหัสลับเป็นคั่นละค่าหรือขั้นตอนเฉพาะ การเข้ารหัสลับ และการถอดรหัสลับจะเป็นดังรูปที่ 2.4 โดยสมการการสร้างรหัสลับเป็น $P=D(K_D,E(K_E,P))$ และ K_D กับ K_E จะแทนกุญแจการถอดรหัสลับ และการเข้ารหัสลับตามลำดับ

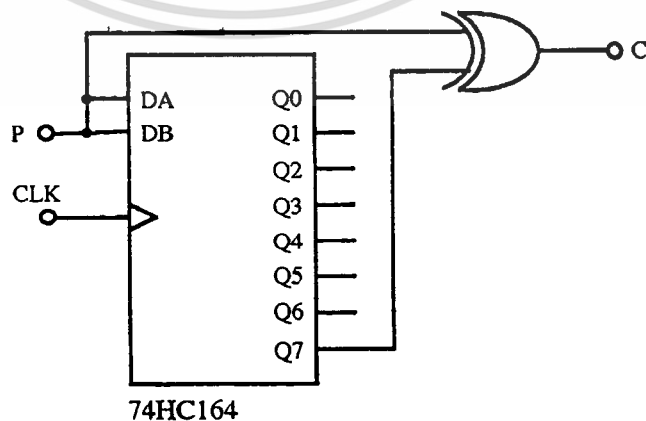


รูปที่ 2.3 ระบบการสร้างรหัสลับด้วยกุญแจรหัสตัวเดียว



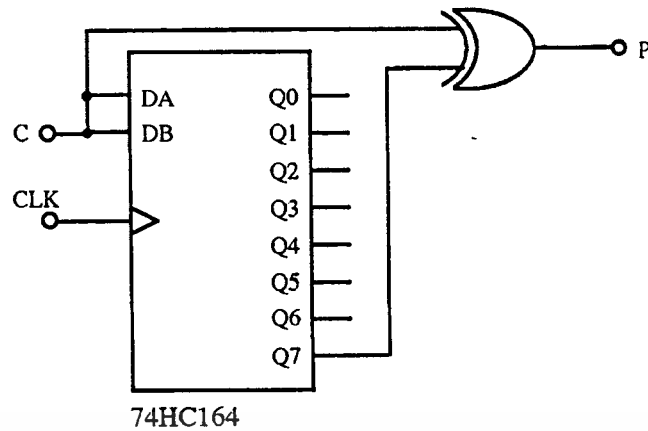
รูปที่ 2.4 ระบบการสร้างรหัสลับด้วยกุญแจการเข้ารหัส และถอดรหัสลับแยกจากกัน

เมื่อนำวิธีการเข้ารหัสและถอดรหัสด้วยกุญแจตัวเดียวกันที่กล่าวมาข้างต้น มาสร้างเป็นวงจรเข้ารหัสและถอดรหัสอย่างง่ายเพื่อใช้ในวงจรฮาร์ดลอค ทำให้ได้วงจรดังรูปที่ 2.5 และ 2.6



รูปที่ 2.5 สร้างวงจรเข้ารหัสด้วย Serial-in Parallel-out Shift Register

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2.6 สร้างวงจรถอดรหัสด้วย Serial-in Parallel-out Shift Register

สมการเข้ารหัส	$= C = E(K,P) = p_i \oplus p_{i+8} \quad ; i = \text{เป็นบิตที่ } 15-0$
สมการถอดรหัส	$= P = D(K,C) = c_i \oplus c_{i+8}$
ตัวอย่าง ข้อมูลปกติ	$P = 1010\ 0001\ 1011\ 0110\ (A1B6)$
เราจะได้ข้อมูลเข้ารหัสกลับเป็น	$E(K,P) = C = 0001\ 0111\ 1011\ 0110\ (17B6)$
หลังจากนั้นถอดรหัสกลับได้เป็น	$D(C) = 1010\ 0001\ 1011\ 0110\ (A1B6)$
จะเห็นว่า	$P = D(K,E(K,P))$

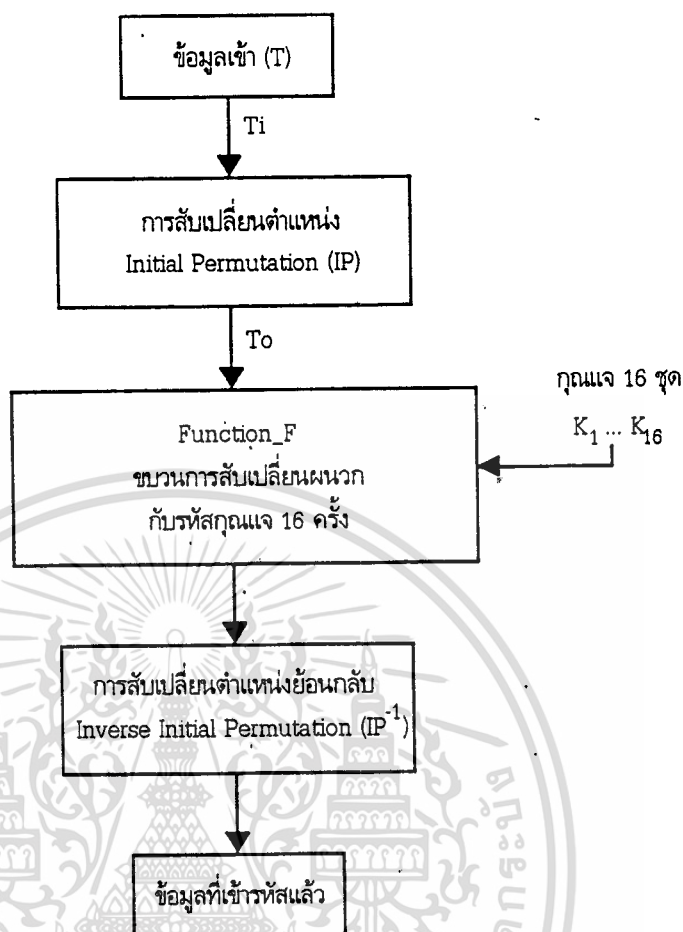
2.1.2 ทฤษฎีของ DES

จากวิธีการเข้ารหัสถอดรหัสที่กล่าวไว้ก่อนหน้านี้เป็นวิธีที่นำไปสร้างวงจรเข้ารหัสและถอดรหัส เฉพาะส่วนที่เป็นข้อมูล (Data) ของรหัสคำสั่งฮาร์ดลอค ดังนั้นวงจรนี้จึงเป็นเพียงวงจรช่วยเสริมการทำงานของฮาร์ดลอคเท่านั้น อย่างไรก็ตามในกรณีที่ต้องการวิธีการเข้ารหัสและถอดรหัสที่ซับซ้อนมากกว่าวิธีที่กล่าวมาสามารถทำได้โดยใช้วิธีของ DES (Data Encryption Standard) [12 ,13 ,14] ซึ่งมีหลักการคือ ใช้วิธีทางคณิตศาสตร์เข้ามาสลับเปลี่ยนข้อมูล โดยข้อมูลที่เข้ามาจะผ่านขั้นตอนการเข้ารหัสจะถูกสลับเปลี่ยนเป็นข้อมูลที่ไม่มีรูปแบบเรียกว่า Cipher การถอดรหัสจะนำข้อมูลที่ไม่มีรูปแบบนี้มาเข้าขั้นตอนการถอดรหัส เปลี่ยนเป็นข้อมูลเดิม วิธีทางคณิตศาสตร์ที่จะกล่าวถึงเป็นการเข้ารหัสและถอดรหัสโดยใช้เลขฐานสอง กุญแจประกอบด้วยเลขฐานสอง 64 บิตซึ่งใช้ในขั้นตอนการเข้ารหัสและถอดรหัส อีก 8 บิตใช้ในการตรวจสอบข้อผิดพลาด (Error Detection) ดังนั้นกุญแจที่ใช้จริงจึงมีขนาดเพียง 56 บิต

2.1.2.1 ขั้นตอนการเข้ารหัสและถอดรหัส

ขั้นตอนการเข้ารหัสและถอดรหัสแสดงในรูปที่ 2.7 ข้อมูลเข้าเป็นบล็อก T_i เข้าสู่ขั้นตอนการสลับเปลี่ยนตำแหน่ง IP และ ข้อมูลออก $To = IP(T)$ ข้อมูล To เข้าสู่ขั้นตอนการสลับเปลี่ยนผนวกกับรหัสกุญแจ 16 ครั้ง ซึ่งเรียกว่า Function_F เมื่อเสร็จขั้นตอนการจะเข้าสู่ขั้นตอนการสลับเปลี่ยนย้อนกลับ IP^{-1} ผลลัพธ์ที่ได้เป็นข้อมูลที่รับ

เอกสารนี้เข้ามาก่อนผ่านขั้นตอนการเข้ารหัสใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



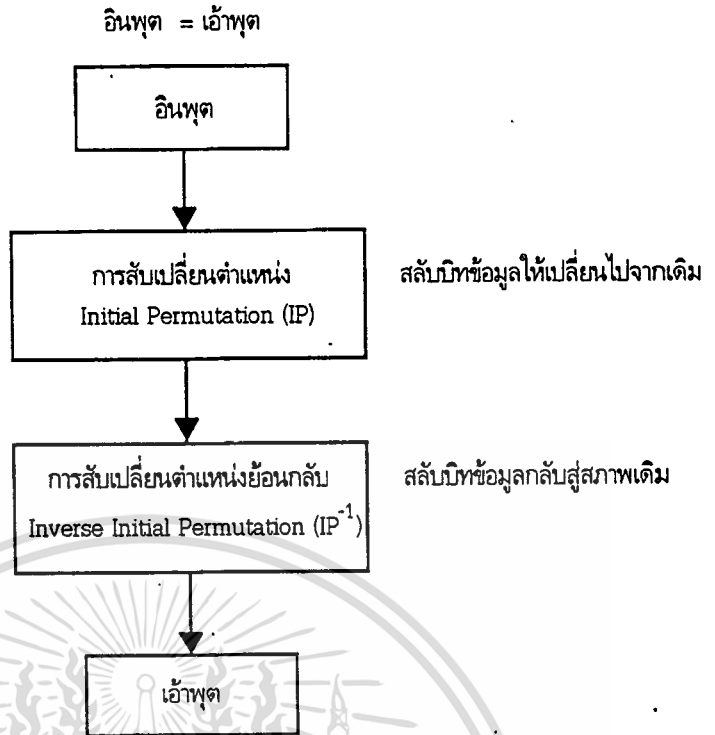
รูปที่ 2.7 บล็อกแสดงขั้นตอนการเข้ารหัสและถอดรหัส

2.1.2.2 การสับเปลี่ยนตำแหน่ง

ขั้นตอนการ DES ใช้หลักการสับเปลี่ยนตำแหน่งหลายขั้นตอนทั้งในด้านการเข้ารหัส ถอดรหัส และคำนวณหมายเลขกุญแจ การสับเปลี่ยนดังกล่าวได้แก่ ตาราง IP, IP^{-1} , E, P, PC_1 และ PC_2 การสับเปลี่ยนตำแหน่ง IP และ IP^{-1}

จุดประสงค์ของการสับเปลี่ยน IP และ IP^{-1} คือการโยกย้ายบิตให้แตกต่างไปจากข้อมูลเดิม และเมื่อทำการสับเปลี่ยนย้อนกลับจะได้ข้อมูลเดิม

จากรูปที่ 2.8 จะทำให้การอธิบายดูง่ายขึ้นถ้าตัดส่วนที่เป็นขั้นตอนการสับเปลี่ยนผวกกับรหัสกุญแจ 16 ชุดออกจะเห็นว่าข้อมูลเข้าและออกจะมีค่าเดียวกัน นั่นคือขั้นตอนการสับเปลี่ยน IP จะรับอินพุตขนาด 64 บิต มาสับเปลี่ยนตำแหน่งตาม Initial Permutation (IP) และส่งเอาพุตให้กับ IP^{-1} ซึ่งเป็นการสับเปลี่ยนตำแหน่งย้อนกลับทำให้ได้ข้อมูลกลับสู่สภาพเดิม



รูปที่ 2.8 บล็อกแสดงกระบวนการ IP และ IP^{-1}

รูปที่ 2.9 รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP

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

รูปที่ 2.10 รายละเอียดตารางสับเปลี่ยนตำแหน่ง IP^{-1}

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

1 ← ตำแหน่งบิตที่ 1 ของตารางนั้นๆ
58 ← ค่าของบิตที่ 58 ของอินพุต

IP ข้อมูลตำแหน่งบิตที่ 1 และ 64 ของรูปที่ 2.9 จะถูกสลับที่ไปอยู่ตำแหน่งบิตที่ 58 และ 7
 IP^{-1} ข้อมูลตำแหน่งบิตที่ 58 และ 7 ของรูปที่ 2.10 จะถูกสลับที่ไปอยู่ตำแหน่งบิตที่ 1 และ 64

ตารางที่ 2.1 และ 2.2 แสดงตารางสลับเปลี่ยนตำแหน่ง IP และ IP^{-1} อย่างกระชับ การใช้ตารางควรวาดจากซ้ายไปขวา บนลงล่างเช่น สับเปลี่ยนจาก $T_i = t_1 t_2 \dots t_{64}$ เป็น $T_o = t_{58} t_{50} \dots t_7$

ตารางที่ 2.1 ตารางสับเปลี่ยนตำแหน่ง IP

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

ตารางที่ 2.2 ตารางสับเปลี่ยนตำแหน่ง IP^{-1}

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

ตัวอย่าง ข้อมูลอินพุต = A2DC 8F5A 6ACE 160C

หลังจากทำ IP = 3A4A E604 2711 BE7D

หลังจากทำ IP^{-1} = A2DC 8F5A 6ACE 160C

จะเห็นว่า อินพุต = เอาพุต (IP^{-1}) จุดประสงค์ของขบวนการสับเปลี่ยน IP และ IP^{-1} คือการโยกย้ายข้อมูลให้แตกต่างไปจากเดิมเพื่อให้สับสนยากต่อการแกะรอย (Trace)

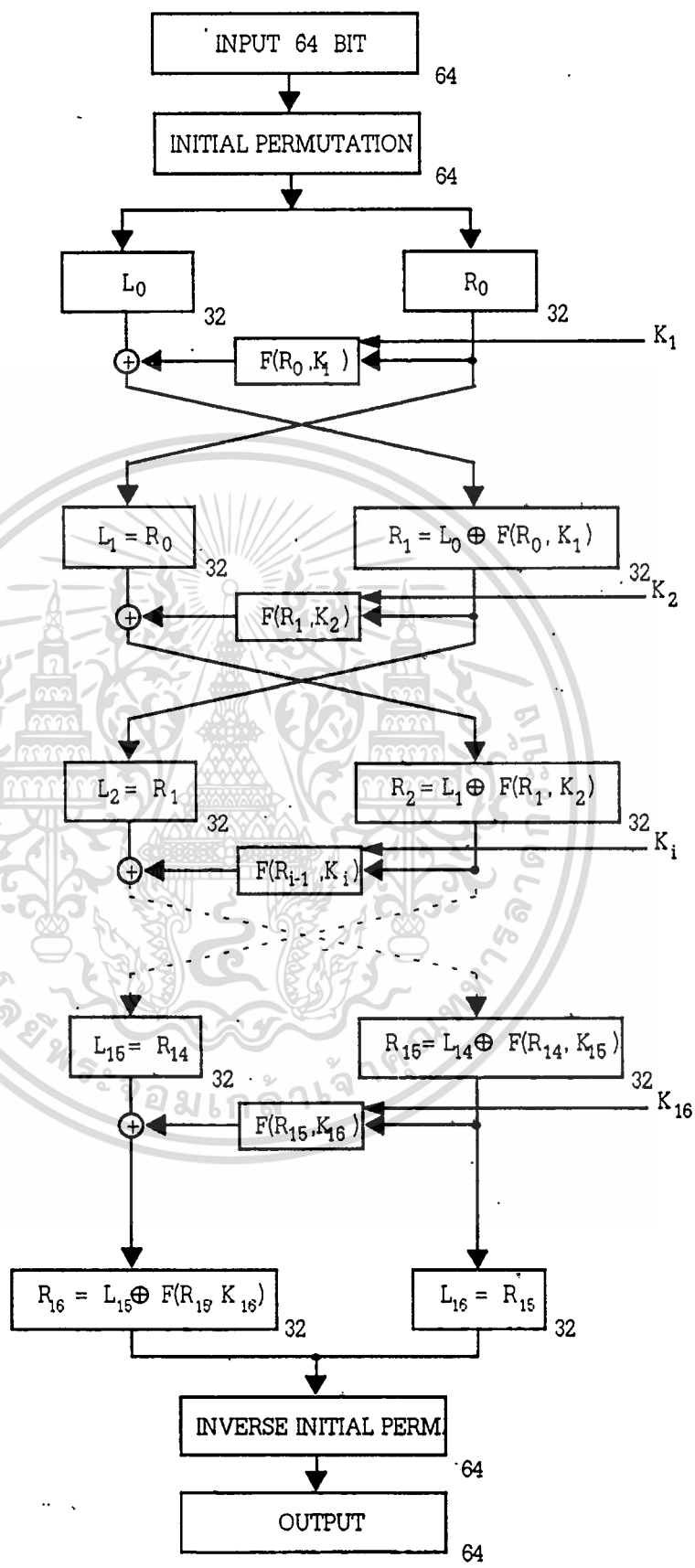
* ตารางการสับเปลี่ยน IP และ IP^{-1} สามารถกำหนดขึ้นเองได้เช่น

ตารางที่ 2.3 ตัวอย่าง IP ใหม่

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

ตารางที่ 2.4 ตัวอย่าง IP^{-1} ใหม่

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



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้ภายในเท่านั้น ไม่ควรเผยแพร่สู่สาธารณะโดยไม่ได้รับอนุญาต
รูปที่ 2.11 บล็อกแสดงขั้นตอนการเข้ารหัส นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 2.11 ขบวนการสลับเปลี่ยนผนวกกับรหัสทุกแฉ 16 ครั้งแต่ละครั้งเรียกว่า Function_F อยู่ระหว่างการสลับเปลี่ยนตำแหน่ง IP กับการสลับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1} ตารางที่ 2.1 และ 2.2 เป็นข้อมูลที่ได้จากทฤษฎีของ DES [12], [13], [14], [15]

ให้ T_i แสดงถึงผลลัพธ์ของการทำครั้งที่ i แบ่ง T_i ออกเป็น 2 ส่วนๆละ 32 บิต ให้ด้านซ้ายเป็น L และด้านขวาเป็น R ฉะนั้น L_i, R_i จะเป็น

$$L_i = t1 \dots t32$$

$$R_i = t33 \dots t64$$

สมการที่ใช้ในขบวนการเข้ารหัสเป็นดังนี้

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \quad (1)$$

เครื่องหมาย $\oplus$ หมายถึงการทำเอ็กครูซีฟออร์ (XOR) และ K_i คือหมายเลขทุกแฉที่มีความยาว 48 บิต ขบวนการสลับเปลี่ยนผนวกกับรหัสทุกแฉทั้งหมด 16 รอบ แสดงในรูปที่ 2 รอบที่ 1 ถึง 15 สลับเปลี่ยนตามสมการ (1) ในรอบที่ 16 ซึ่งเป็นรอบสุดท้ายด้านซ้าย L และด้านขวา R จะไม่สลับที่กัน ข้อมูล L_{16}, R_{16} ถูกใช้เป็นข้อมูลเข้าเพื่อทำการสลับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1}

2.1.2.3 Function_F และ S_box

รูปที่ 2.12 แสดงการทำ Function_F (R_{i-1}, K_i) การทำงานเริ่มจาก R_{i-1} ถูกสลับเปลี่ยนเป็น 48 บิต โดยใช้ $E(R_{i-1})$ ตารางที่ 2.5 แสดงถึงตำแหน่งบิต E ซึ่งเป็นลักษณะเดียวกับที่ใช้การสลับเปลี่ยนตำแหน่ง IP และสลับเปลี่ยนตำแหน่งย้อนกลับ IP^{-1} มีส่วนที่ไม่เหมือนกันคือ R_{i-1} ถูกใช้มากกว่า 1 ครั้งเช่น $R_{i-1} = r_1 r_2 \dots r_{32}$, $E(R_{i-1}) = r_{32} r_1 r_2 \dots r_{32} r_1$

ต่อมา $E(R_{i-1})$ และ K_i มาทำ XOR ผลลัพธ์ที่ได้ถูกแบ่งเป็นข้อมูลขนาด 6 บิต 8 ชุด คือชุดที่ $B_1, \dots, B_8$ เขียนเป็นสมการจะได้

$$E(R_{i-1}) \oplus K_i = B_1 B_2 \dots B_8 \quad (2)$$

ข้อมูลในแต่ละชุด B_j มีขนาด 6 บิต จะใช้เป็นข้อมูลเข้าใน Function (S_box) S_j และข้อมูลออกจะมีขนาด 4 บิต เขียนในรูปของฟังก์ชันได้

$$S_j(B_j) \quad (3)$$

S_box จะมีทั้งหมด 8 ชุด หลังจากผ่านขบวนการ S_box จะได้ผลลัพธ์ขนาด 32 บิต และเข้าขบวนการสลับเปลี่ยน P ซึ่งแสดงในตาราง P ตารางที่ 2.6 เป็นข้อมูลที่ได้จากทฤษฎีของ DES

ผลลัพธ์ของ S_box หลังการเข้าขบวนการสลับเปลี่ยน P จะอยู่ในรูปของ $F(R_{i-1}, K_i)$ เขียนได้เป็น

$$P(S_1(B_1) \dots S_8(B_8)) \quad (4)$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 2.5 ตาราง E

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

ตารางที่ 2.6 ตาราง P

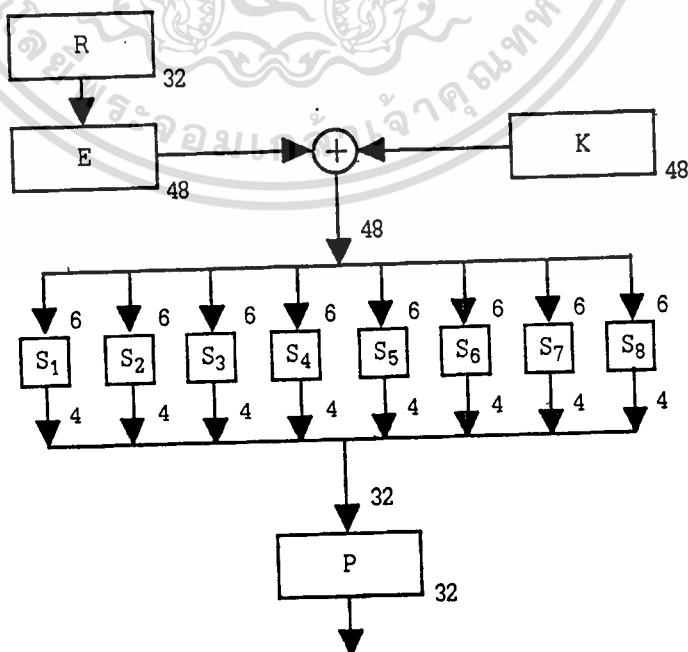
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

วิธีการเข้า S_box อธิบายได้ดังนี้

ให้ B_j เป็นข้อมูลเข้ามีขนาด 6 บิต ($b_1b_2b_3b_4b_5b_6$) ให้ j เป็นเลขจำนวนเต็มมีค่า 1 ถึง 8 แบ่ง B_j ออกเป็น 2 ส่วนเรียกว่า Row และ Column ให้ในส่วนของ Row = b_1b_6 และส่วนของ Column = $b_2b_3b_4b_5$ นำค่า Row และ Column มาเทียบในตาราง S_j จะได้ผลลัพธ์เป็นค่า 4 บิต ดังนั้นเขียนได้เป็น

$S_j(B_j)$ ให้ผลลัพธ์ขนาด 4 บิต

ตัวอย่าง ถ้า $B_1 = 101010$ S_1 จะได้ผลลัพธ์เป็น Row = 10 = 2 Column = 0101 = 5 เทียบในตาราง S_1 จะได้ 6 ซึ่งเท่ากับ 0110
 ถ้า $B_7 = 100101$ S_7 จะได้ผลลัพธ์เป็น Row = 11 = 3 Column = 0010 = 2 เทียบในตาราง S_7 จะได้ 13 ซึ่งเท่ากับ 1101



รูปที่ 2.12 บล็อกไดอะแกรมแสดงขั้นตอนการสร้าง Function_F(R, K)

เอกสารนี้เป็นเอกสารที่สงวนไว้เพื่อใช้ในการวิจัยเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

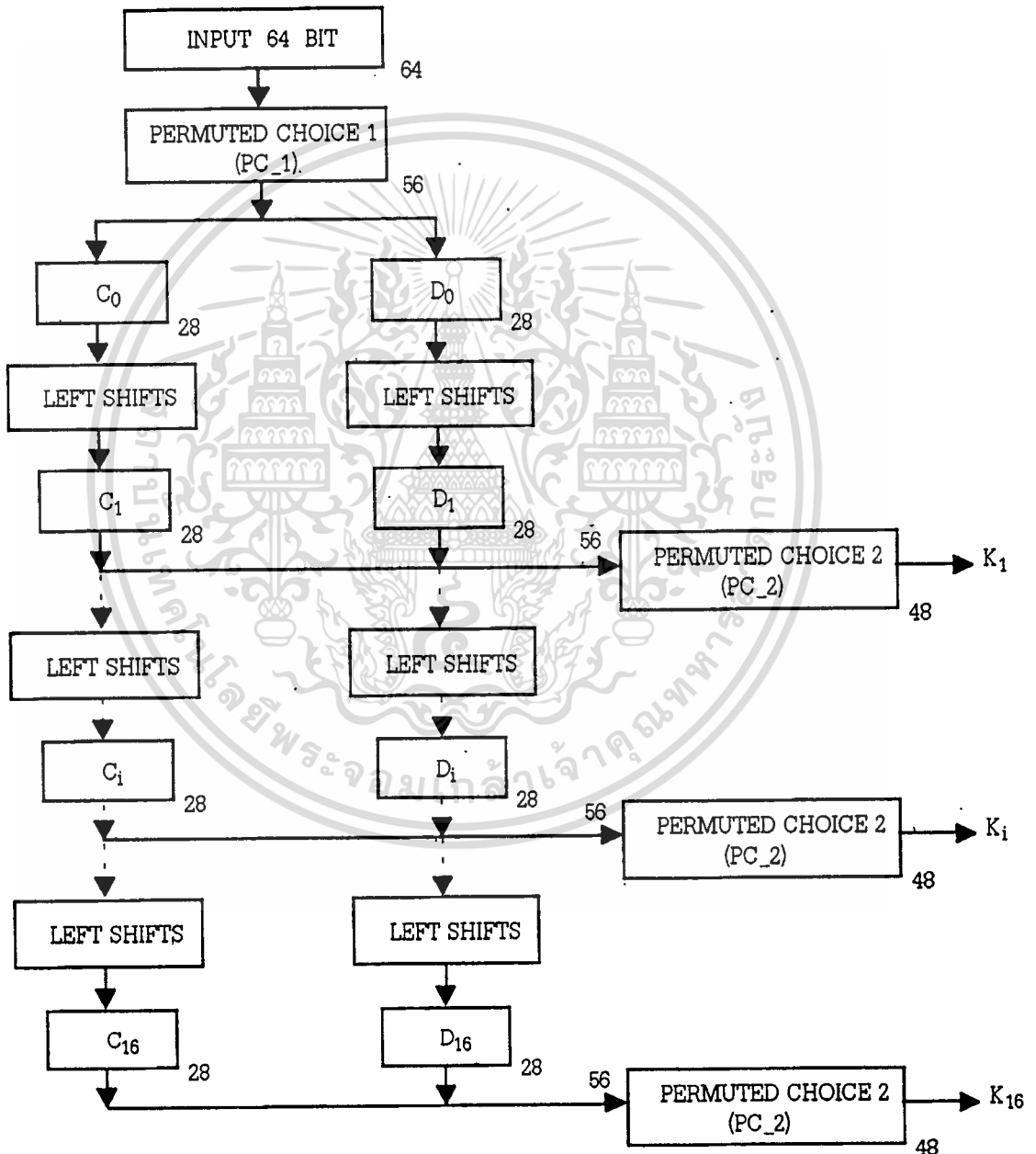
ตารางที่ 2.7 S_box

Column																	
Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Box
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S ₁
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S ₂
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S ₃
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S ₄
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S ₅
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S ₆
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S ₇
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	1	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S ₈
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.1.2.4 การคำนวณหมายเลขกุญแจ (Subkey)

ในแต่ละรอบของการสับเปลี่ยนผนวกกับรหัสกุญแจ 16 ชุด จะใช้หมายเลขกุญแจแต่ละชุดมีขนาด 48 บิต หมายเลขกุญแจ 16 ชุดได้มาจาก Key ดังรูปที่ Key เป็นข้อมูลเข้ามีขนาด 64 บิต 8 บิตในตำแหน่ง บิตที่ 8, 16, ..., 64 ใช้เป็นบิตตรวจสอบข้อผิดพลาด (Parity bit) การทำขบวนการสับเปลี่ยน Permuted PC₁ หรือ Permuted choice 1 จะตัดในส่วนของบิตตรวจสอบข้อผิดพลาดออก ใช้ 56 บิตที่เหลือเท่านั้น ตารางสับเปลี่ยนตำแหน่งของ PC₁ ดูในตารางที่ 2.8



รูปที่ 2.13 บล็อกแสดงการคำนวณหมายเลขกุญแจ 16 ชุด

ผลลัพธ์หลังจากการทำ PC_1 จะถูกแบ่งเป็น 2 ส่วนเท่าๆกัน เรียกว่า C และ D ใช้หาค่าหมายเลข
กุญแจ K_i

กำหนดให้ C_i และ D_i ซึ่งได้มาจาก C, D ใช้หาค่า K_i มีสมการเป็น

$$C_i = LS_i(C_{i-1})$$

$$D_i = LS_i(D_{i-1})$$

LS_i คือ การทำการหมุนทางซ้ายเท่ากับจำนวนครั้งที่กำหนดในตารางที่ C_0 และ D_0 เป็นข้อมูลเริ่มแรก
ของ C และ D ดังนั้นหมายเลขกุญแจ K_i เขียนสมการได้เป็น

$$K_i = PC_2(C_i D_i)$$

ขบวนการสลับเปลี่ยนตำแหน่ง PC_2 อยู่ในตารางที่ 2.9

ตารางที่ 2.8 ตาราง PC_1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

ตารางที่ 2.9 ตาราง PC_2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

ตารางที่ 2.10 ตารางแสดงจำนวนครั้งในการหมุนซ้าย (LS)

ลำดับ	จำนวนครั้ง
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2

ลำดับ	จำนวนครั้ง
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

กำหนดให้ หมายเลขกุญแจเท่ากับ D465 9CAE 367C D9EB นำหมายเลขกุญแจที่ได้ทำการสับเปลี่ยน PC₁ ได้ผลลัพธ์เท่ากับ CDE3 BA70 983F EC50 ข้อมูลนี้แบ่งออกเป็นสองส่วนคือ

$$C_0 = \text{CDE3 BA70}$$

$$D_0 = \text{983F EC50}$$

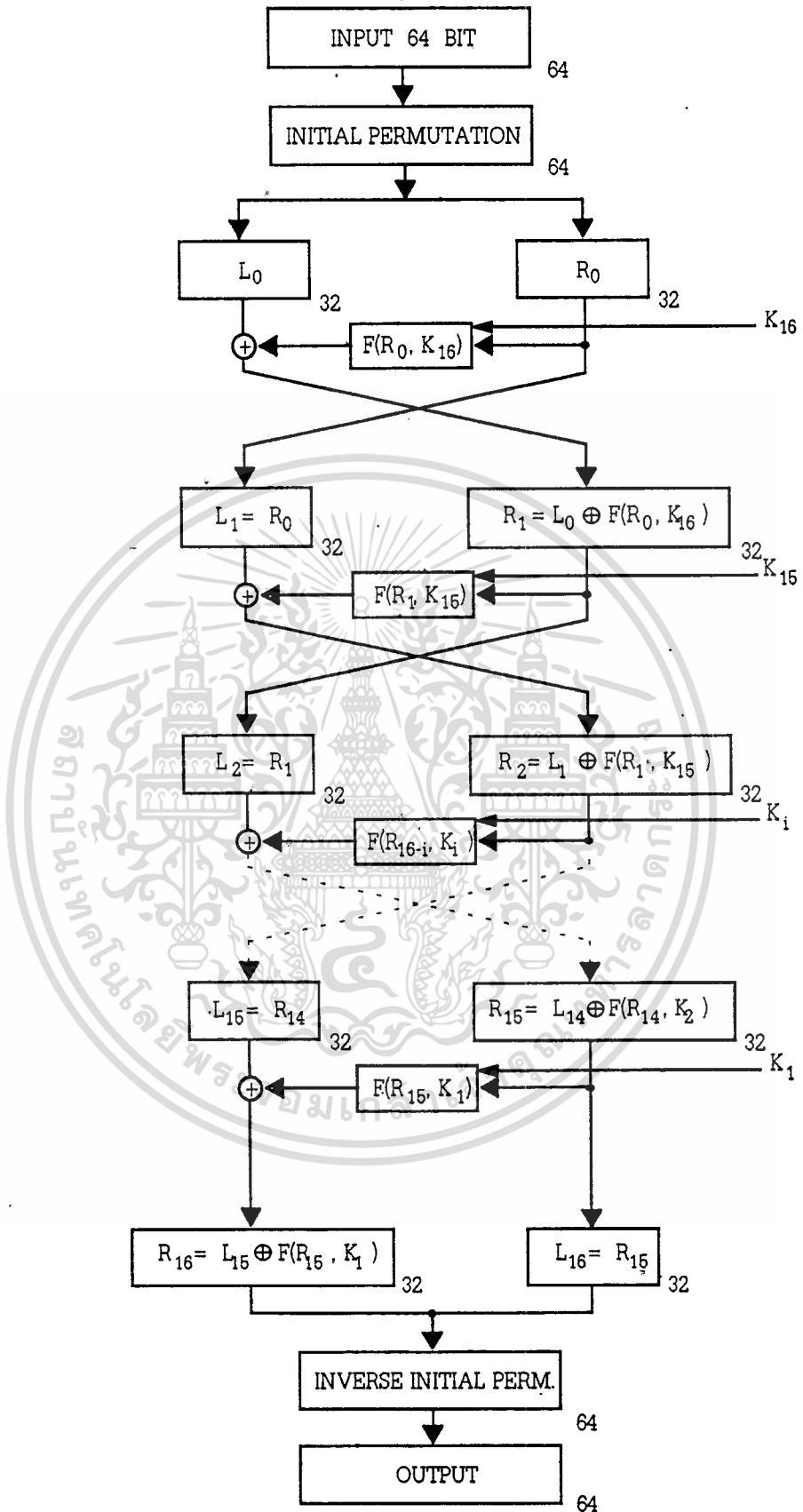
ขั้นตอนต่อไปนำ C_0 และ D_0 มาหมุนตามจำนวนครั้งในตารางที่ 2.10

	$C_0 = \text{CDE3 BA70}$	0	$D_0 = \text{983F EC50}$	0
ลำดับที่ 1 หมุน 1 ครั้ง	$C_1 = \text{9BC7 74F}$	0	$D_1 = \text{983F EC50}$	0
ลำดับที่ 2 หมุน 1 ครั้ง	$C_2 = \text{CDE3 BA70}$	0	$D_2 = \text{983F EC50}$	0
ลำดับที่ 3 หมุน 1 ครั้ง	$C_3 = \text{CDE3 BA70}$	0	$D_3 = \text{983F EC50}$	0
⋮	⋮		⋮	
ลำดับที่ 16 หมุน 1 ครั้ง	$C_{16} = \text{CDE3 BA70}$	0	$D_{16} = \text{983F EC50}$	0

นำข้อมูลที่ได้หลังจากการหมุนในแต่ละลำดับเข้าขั้นตอนการสับเปลี่ยน PC₂ ได้เป็นกุญแจ K_1 ถึง K_{16} ดังตารางที่ 2.11 กุญแจ 16 ชุดนี้จะนำไปใช้ในขั้นตอนการเข้ารหัสและถอดรหัสต่อไป

ตารางที่ 2.11 แสดงค่าหมายเลขกุญแจที่คำนวณได้ตามขั้นตอนการ DES

ลำดับ	C	D	#Key (K)
1	9BC774F0	307FD8B0	2319 173C 2915 3D31
2	378EE9F0	60FFB160	343E 171A 3F35 2810
3	DE3BA7C0	83FEC590	1F0C 3E33 3414 0D3E
4	78EE9F30	FFB16600	3D3B 240F 251B 2A0C
5	E3BA7CD0	3FEC5980	0A3A 1B17 3C03 1B35
6	8EE9F370	FFB16600	1F15 323E 0E3A 2A2F
7	3BA7CDE0	FEC59830	3D2A 0738 0D27 3613
8	EE9F3780	FB1660F0	272C 3837 0B32 0537
9	DD3E6F10	F62CC1F0	2F1E 3D03 3924 0F33
10	74F9BC70	D8B307F0	1837 1E2F 153A 2D0B
11	D3E6F1D0	62CC1FF0	3E19 1715 3D29 1518
12	4F9BC770	8B307FD0	151C 2B3B 121B 1D2E
13	3E6F1DD0	2CC1FF60	2D3F 0636 1F0F 3228
14	F9BC7740	B307FD80	2F20 3F07 1A05 313F
15	E6F1DD30	CC1FF620	1E37 083D 233F 223A
16	CDE3BA70	983FEC50	1B1F 332D 2216 3C3B



รูปที่ 2.14 บล็อกแสดงขั้นตอนการถอดรหัส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทำขบวนการย้อนกลับ หรือการถอดรหัส (Decryphering) ทำโดยใช้วิธีการเดิมแต่หมายเลขกุญแจจะใช้ จาก $K_{16}, K_{15}, \dots, K_1$ เป็นจำนวน 16 ครั้ง ดังแสดงขั้นตอนในรูปที่ 2.14

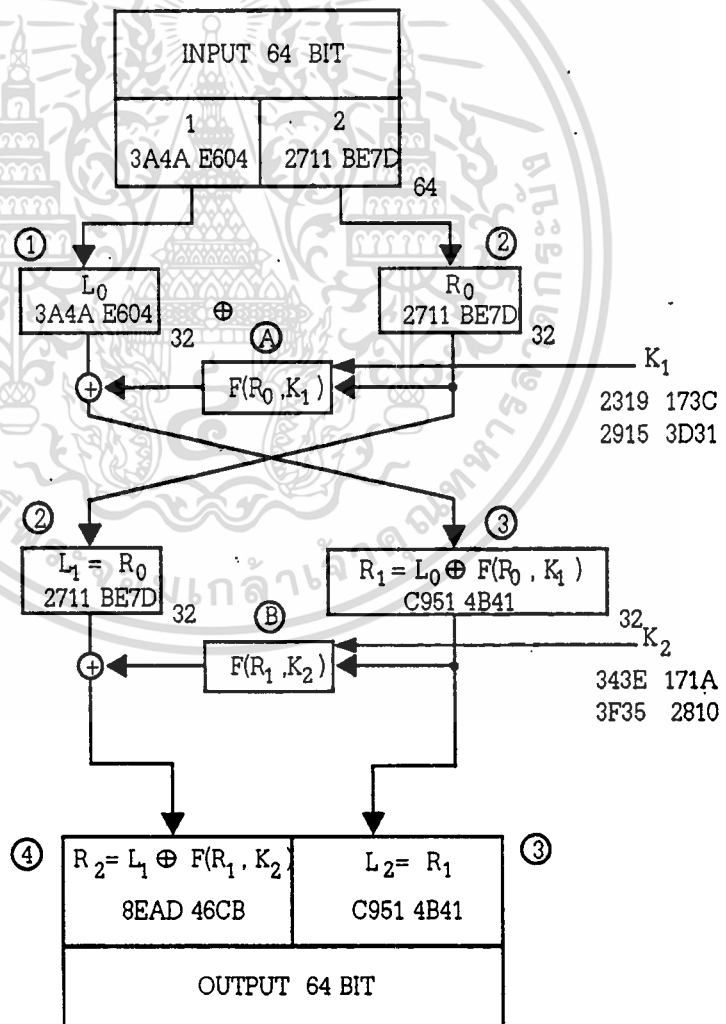
$$R_{i-1} = L_i \quad : i \text{ เริ่มจาก 16 ถึง 1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_{17-i})$$

การทำกระบวนการย้อนกลับจะย้อนกลับในส่วนของกุญแจจาก 16 ถึง 1 ส่วนวิธีจะไม่ย้อนกลับ

2.1.2.5 ตัวอย่างการเข้ารหัสและถอดรหัสโดยทำ Function_F 2 ครั้ง

ข้อมูลเข้าสู่ขบวนการนี้ คือข้อมูล 32 บิตซ้าย และหมายเลขกุญแจ 48 บิต เพื่อให้เข้าใจถึงการ DES ได้ง่ายขึ้น จึงตัดส่วนของขั้นตอนการสลับเปลี่ยน IP และ IP^{-1} ออก และให้ทำ Function_F เพียง 2 ครั้ง เท่านั้น ดังแสดงขั้นตอนการทำงานตามรูปที่ 2.15 ซึ่งตามวิธีการของ DES ต้องทำถึง 16 ครั้ง



รูปที่ 2.15 บล็อกแสดงขั้นตอนการเข้ารหัสโดยทำ Function_F 2 ครั้ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กำหนดให้

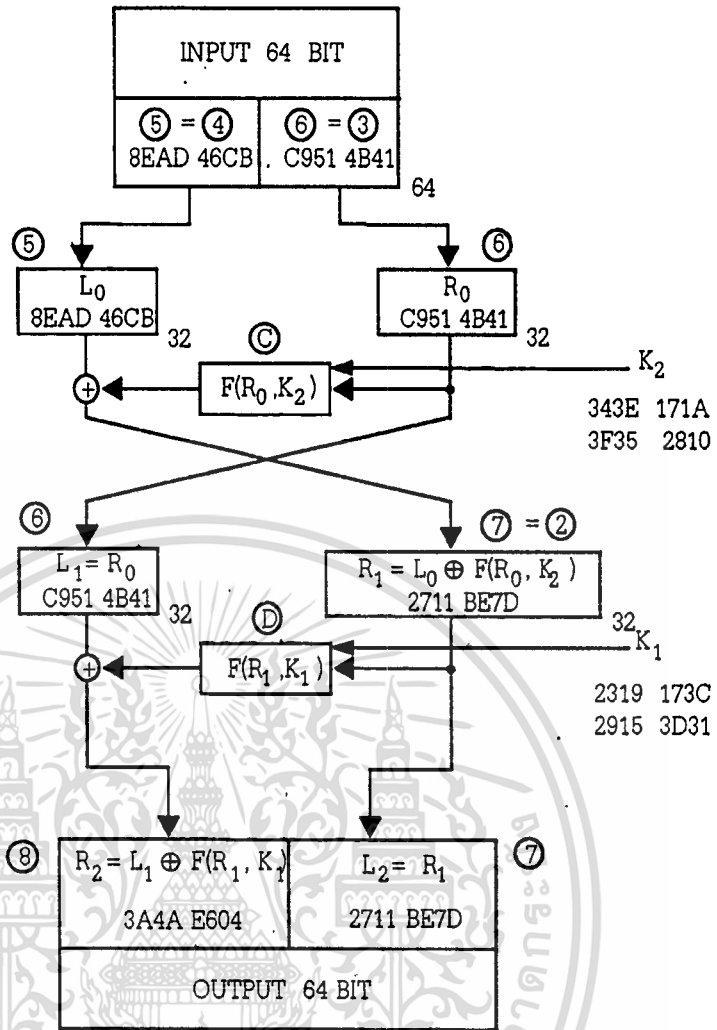
$$\text{อินพุต} = \text{LR} = 3\text{A4A E604 2711 BE7D} \quad (64 \text{ บิต})$$

จากตารางที่ 2.11 เมื่อนำ C_0 และ D_0 มา 56 บิต = 9BC774F307FD8B มาผ่าน PC_2 ในตารางที่ 2.9 จะได้ข้อมูลหลังจากสลับเปลี่ยนเป็นบิตเรียงต่อกัน 48 บิตดังนี้

1 0 0 0 1 1	= 23
0 1 1 0 0 1	= 19
0 1 0 1 1 1	= 17
1 1 1 1 0 0	= 3C
1 0 1 0 0 1	= 29
0 1 0 1 0 1	= 15
1 1 1 1 0 1	= 3D
1 1 0 0 0 1	= 31

จะได้หมายเลขกุญแจ K_1 เป็น 2319 173C 2915 3D31 (48 บิต) ส่วน K_2 ก็ใช้วิธีหาเช่นเดียวกัน
 หมายเลขกุญแจ = K_1 = 2319 173C 2915 3D31 (48 บิต)
 หมายเลขกุญแจ = K_2 = 343E 171A 3F35 2810 (48 บิต)

จะได้เอาพุตหลังจากทำ
 Function_F ครั้งที่ 1 = $R_0 [L_0 \oplus F(R_0, K_1)]$ = 2711 BE7D C951 4B41 (64 บิต)
 Function_F ครั้งที่ 2 = $[L_1 \oplus F(R_1, K_2)] R_1$ = 8EAD 46CB C951 4B41 (64 บิต)



รูปที่ 2.16 บล็อกแสดงขั้นตอนการถอดรหัสโดยทำ Function_F 2 ครั้ง

ส่วนรูปที่ 2.16 แสดงการถอดรหัสโดยนำข้อมูลเข้าพุดจากขบวนการเข้ารหัสมาเป็นข้อมูลอินพุตของขบวนการถอดรหัส

ดังนั้นอินพุต $= [L_1 \oplus F(R_1, K_1)] R_1 = 8EAD\ 46CB\ C951\ 4B41$ (64 บิต)

หมายเลขกุญแจ $= K_1 = 2319\ 173C\ 2915\ 3D31$ (48 บิต)

หมายเลขกุญแจ $= K_2 = 343E\ 171A\ 3F35\ 2810$ (48 บิต)

จะได้เข้าพุดหลังจากทำ

Function_F ครั้งที่ 1 $= R_0 ([L_0 \oplus F(R_0, K_2)] \oplus F(R_0, K_2)) = C951\ 4B41\ 2711\ BE7D$ (64 บิต)

Function_F ครั้งที่ 2 $= ([L_1 \oplus F(R_1, K_1)] \oplus F(R_1, K_1)) R_1 = 3A4A\ E604\ 2711\ BE7D$ (64 บิต) = LR

รูปที่ 2.15 และ 2.16 จะเห็นว่าในรอบแรกอินพุตที่จุด C จะเท่ากับอินพุตที่จุด B คือ 3=6 และ K_2

จะได้ผลลัพธ์ Function_F เท่ากัน เมื่อผลลัพธ์ของ $5 \oplus C$ ได้เหมือนกับจุด 7 และข้อมูลที่จุด 7 จะเท่า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

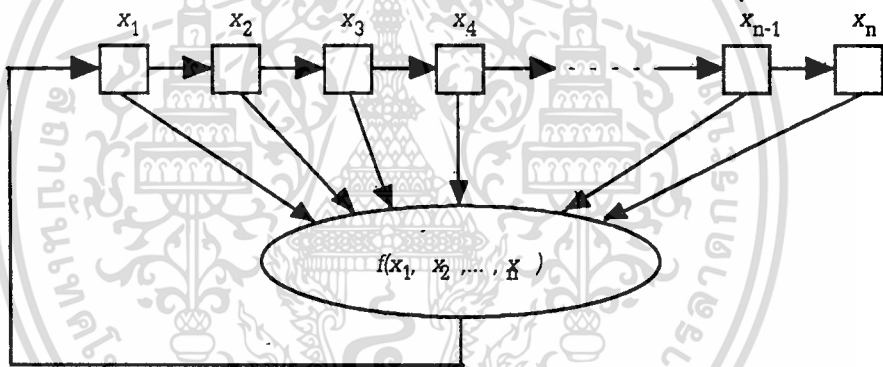
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กับที่จุด 2 ซึ่งเป็นการทำขบวนการย้อนกลับในรอบแรกของการถอดรหัส โดยใช้หลักการของการทำ XOR สองครั้งจะได้ค่าเดิม

ในรอบที่สองอินพุตที่จุด D จะเท่ากับจุด A คือ $2=7$ และ K_1 เนื่องจากจุด $7=2$ และใช้ K_1 ซึ่งจะให้ผลลัพธ์ของ Function_F เท่ากัน เนื่องจาก $7=2$ และใช้ K_1 ชุดเดียวกัน ผลลัพธ์ของ Function_F ที่จุด D $\oplus 6$ ได้ผลลัพธ์เท่ากับที่จุด 8 ซึ่ง $8=1$ โดยใช้หลักการของการทำ XOR กับข้อมูลเดิม 2 ครั้ง

2.1.3 ชิฟทรีจิสเตอร์ (Shift Register)

ในรูปที่ 2.17 บล็อกไดอะแกรมทั่วไปของชิฟทรีจิสเตอร์ที่มีสัญญาณย้อนกลับ (Feedback) สี่เหลี่ยมแต่ละรูปเป็นส่วนที่เก็บข้อมูลเป็นเลขฐานสอง (ฟลิปฟล็อป หรืออุปกรณ์ประเภทหน่วยความจำต่างๆ) ถูกแทนด้วยสัญลักษณ์ $x_1, x_2, \dots, x_n$ ทำงานด้วยสัญญาณนาฬิกาทำให้ข้อมูลที่อยู่ใน x_i ถูกส่งไปเก็บไว้ใน x_{i+1} อย่างไรก็ตามเพื่อให้ได้ข้อมูลใหม่เข้ามาที่ x_1 อยู่เสมอในขณะที่เลื่อนข้อมูล โดยการใช้ฟังก์ชัน $f(x_1, x_2, \dots, x_n)$ คำนวณจากข้อมูลที่อยู่ภายในชิฟทรีจิสเตอร์



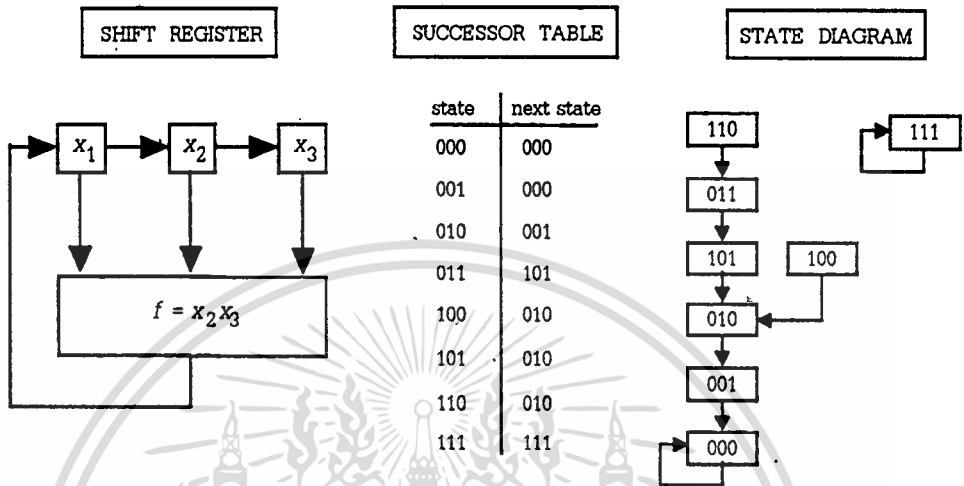
รูปที่ 2.17 บล็อกไดอะแกรมของชิฟทรีจิสเตอร์ที่มีสัญญาณย้อนกลับ

ส่วนที่เก็บข้อมูลฐานสองทั้งหมด n ตัวเรียกว่า ภาค (stage) ของชิฟทรีจิสเตอร์และข้อมูลที่อยู่ภายในภาคเรียกว่า สถานะ (state) ของชิฟทรีจิสเตอร์ ทุกๆสัญญาณนาฬิกาสถานะจะเปลี่ยนจากสถานะหนึ่ง ไปเป็นอีกสถานะหนึ่ง สำหรับชิฟทรีจิสเตอร์ที่มี n ภาค จะมีสถานะทั้งหมด 2^n (เลขฐานสองจาก 00...0 ถึง 11...1) ถ้า ฟังก์ชันของสัญญาณย้อนกลับ แสดงเป็นสมการ

$$f(x_1, x_2, \dots, x_n) = c_1 x_1 \oplus c_2 x_2 \oplus c_3 x_3 \oplus \dots \oplus c_n x_n$$

โดย c_i เป็นค่าคงที่มีค่าเป็น 0 หรือ 1 และ $\oplus$ แทนการ XOR (add mod 2)

ตัวอย่าง ในรูปที่ 2.18 เป็นชิฟท์รีจิสเตอร์ที่มี 3 บิต และมีฟังก์ชันสัญญาณย้อนกลับเป็น $f(x_1, x_2, \dots, x_n) = x_2 x_3$ จากสมการนี้สามารถคำนวณหาคำวนสถานะถัดไปได้ดังแสดงในส่วนของ "successor table" ซึ่งแสดงสถานะเริ่มต้นและสถานะที่เปลี่ยนไปทั้ง 8 สถานะของชิฟท์รีจิสเตอร์ สำหรับตัวอย่างนี้ที่สถานะเริ่มต้นเป็น 4 (เลขฐานสอง 100) ทำให้ได้สถานะเป็นลำดับดังนี้ 4, 2, 1, 0, 0, 0, 0, ...



รูปที่ 2.18 แสดงชิฟท์รีจิสเตอร์แบบสามภาค การเปลี่ยนสถานะ และไดอะแกรมของสถานะ

จากรูปที่ 2.18 จะเห็นว่ามีบางสถานะที่มีสถานะนำหน้า (predecessors) หลายสถานะ เช่นสถานะที่เป็น 2 มีสถานะนำหน้าเป็น 4 และ 5 และสถานะ 0 มีสถานะนำหน้าเป็น 0 และ 1 ในขณะที่สถานะ 4 และ 6 ไม่มีสถานะนำหน้า ชิฟท์รีจิสเตอร์แบบนี้จึงเป็นแบบไม่เชิงเส้น (nonlinear shift register) เพราะว่ามีบางสถานะที่มีสถานะนำหน้า (predecessors) หลายสถานะ ส่วนรูปที่ 2.17 เป็นชิฟท์รีจิสเตอร์แบบเชิงเส้น (linear shift register) เพราะว่าทุกสถานะจะมีสถานะนำหน้า (predecessors) เพียงสถานะเดียว

ทฤษฎีที่ 1 การเปลี่ยนแปลงสถานะในชิฟท์รีจิสเตอร์มีลักษณะเป็นคาบ (periodic) โดยมีคาบ $p < 2^r - 1$ เมื่อ r เป็นจำนวนภาค

พิสูจน์ แต่ละสถานะของชิฟท์รีจิสเตอร์ถูกกำหนดโดยสถานะก่อนหน้านั้น ดังนั้นถ้าสถานะก่อนหน้าและสถานะถัดมานั้นได้เคยเกิดขึ้นแล้วอย่างใด ก็จะเป็นเช่นนี้ตลอดไป ดังนั้นลักษณะสถานะที่เป็นคาบๆจึงเกิดขึ้น ในกรณีที่ชิฟท์รีจิสเตอร์มีสถานะเดิมเป็น 0 ทั้งหมด สถานะต่อมาก็จะเป็น 0 เหมือนเดิมทำให้ $p = 1$

เพื่อความสะดวกในการคาดการณ์สถานะถัดไปของชิฟท์รีจิสเตอร์ จึงสนใจสถานะเฉพาะเพียงหนึ่งภาคของชิฟท์รีจิสเตอร์ กำหนดให้การเปลี่ยนแปลงของสถานะหนึ่งภาคเป็นตัวเลขเรียงต่อกันเป็น $a_0, a_1, a_2, \dots, a_n$

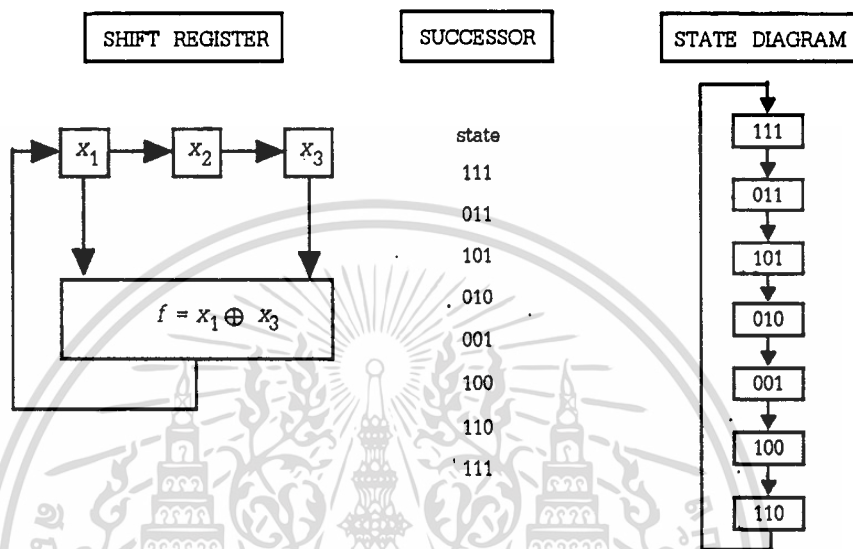
ทฤษฎีที่ 2 สมการของบิต a_n เป็นดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้ภายในของสำนักงานส่งเสริมการค้าในต่างประเทศ ณ นครเซี่ยงไฮ้ ไม่ควรเผยแพร่โดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อสัมประสิทธิ์ $c_1, c_2, \dots, c_r$ มีค่าเป็น 1 หรือ 0 ไม่ขึ้นกับตัวแปร n ลำดับของ $\{a_n\}$ ที่สอดคล้องกับสมการข้างบนเรียกว่า "linear recurring sequence"

ตัวอย่าง พิจารณาซีฟรี่จิสเตอร์ที่มี 3 ภาสดังรูปที่ 2.19 ถ้าสถานะเริ่มต้นเป็น 111 ลำดับของสถานะต่อมาเป็นดัง state ในรูป 2.19



รูปที่ 2.19 แสดงซีฟรี่จิสเตอร์แบบสามภาคที่สร้างสัญญาณย้อนกลับจากภาค 1 กับ 3

สถานะที่ 8 จะกลับมาเหมือนสถานะแรก ดังนั้นจึงมีคาบมากที่สุดเป็น $7 = 2^3 - 1$ เป็นไปตามทฤษฎีที่ 1 ซึ่งสถานะใดๆสามารถยอมให้เกิดได้ยกเว้นสถานะ 000 เพราะจะทำให้สถานะต่อไปเป็น 000 ตลอด ลำดับสถานะของภาคแรกเป็น 1010011 สอดคล้องกับทฤษฎีที่ 2 เป็นสมการดังนี้

$$a_n = a_{n-1} + a_{n-3} \quad : \quad c_2 = 0, c_1 = 1, c_3 = 1$$

จากตัวอย่างจะเห็นว่า การเปลี่ยนแปลงสถานะของภาคที่ 2 เหมือนกับช่วงเวลาของภาคที่ 1 หนึ่งสัญญาณนาฬิกา สถานะที่ภาคอื่นๆก็เช่นเดียวกัน และสถานะที่ภาคต่างๆสามารถคำนวณได้จากสมการ a_n เช่น ลำดับสถานะของภาคแรกเป็น 1010011 ดังนั้น $a_0, a_1, \dots, a_6$ จึงเป็น 1, 0, 1, 0, 0, 1, 1 ตามลำดับ และสอดคล้องกับสมการ $a_6 = a_5 + a_3$: $a_5 = a_4 + a_2$: $a_4 = a_3 + a_1$: $a_3 = a_2 + a_0$ วิธีนี้สามารถนำไปใช้กับซีฟรี่จิสเตอร์ที่มีสัญญาณย้อนกลับ ที่มีจำนวนภาคเท่าใดก็ได้เพื่อหาสถานะต่อไป

วิธี Matrix

สมการชีพจรจิสเตอร์โดยทั่วไป เขียนอยู่ในรูปของ Matrix ได้เป็น

$$M = \begin{bmatrix} c_1 & 1 & 0 & \cdots & 0 \\ c_2 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ c_{r-1} & 0 & 0 & & 1 \\ c_r & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$(a_{n-1}, a_{n-2}, \dots, a_{n-r}) \begin{bmatrix} c_1 & 1 & 0 & \cdots & 0 \\ c_2 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ c_{r-1} & 0 & 0 & & 1 \\ c_r & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$= (c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_r a_{n-r}, a_{n-1}, a_{n-2}, \dots, a_{n-r+1})$$

$$= (a_n, a_{n-1}, a_{n-2}, \dots, a_{n-r+1})$$

ตัวอย่างเช่น ชีพจรจิสเตอร์ขนาด 3 ภาคตามรูปที่ 2.19 เขียนเป็นสมการ matrix ให้อยู่ในรูปสมการทั่วไปได้เป็น

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

และ

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

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะเห็นว่าถ้าสถานะเริ่มต้นเป็น $111 = (1\ 1\ 1)$ จะได้สถานะต่อไปเป็น $011 = (0\ 1\ 1)$ และถ้าสถานะเริ่มต้นเป็น $011 = (0\ 1\ 1)$ จะได้สถานะต่อไปเป็น $101 = (1\ 0\ 1)$

โพลิโนเมียล (Polynomial)

สมการโพลิโนเมียล

$$f(x) = \sum_{i=0}^n c_i x^i$$

ข้อมูลที่เขียนในตารางคือ (สมการโพลิโนเมียลที่ค่าของ $x = 2$)

$$\sum_{i=0}^n c_i 2^i$$

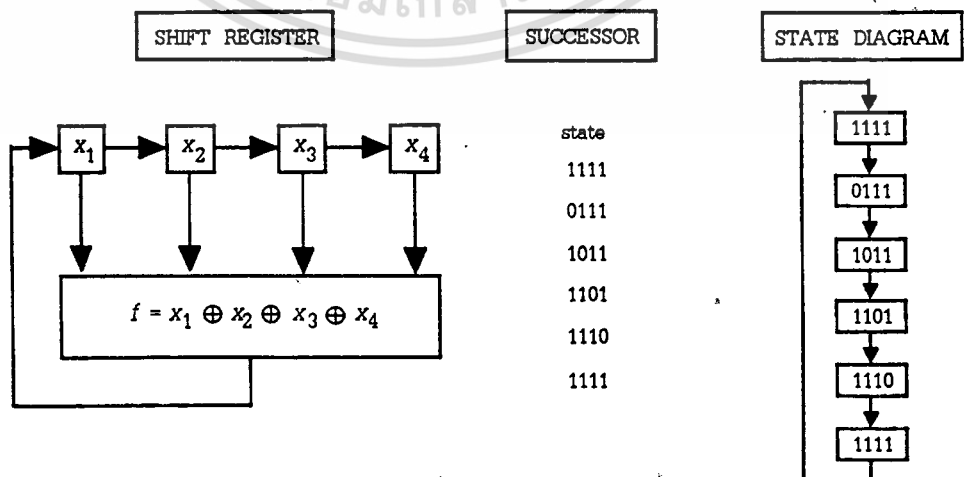
สมการโพลิโนเมียล $x^5 + x^4 + x^3 + x^2 + x + 1$ กรณีที่ $x = 2$ เขียนเป็นเลขฐานสองได้เป็น 101, 111 ซึ่งมีค่า 57 ในเลขฐาน 8

นำความรู้ที่ได้จากทฤษฎีของชิฟร์รีจิสเตอร์ที่กล่าวมาใช้สร้างเป็นตารางแสดงกำลังของโพลิโนเมียล กับ คาบสถานะของชิฟร์รีจิสเตอร์ที่มี 11 ภาควัด ได้ตามตารางที่ 2.12

ความรู้เรื่องชิฟร์รีจิสเตอร์นี้จะได้นำไปใช้สร้างข้อมูลสุมแบบเทียมในหัวข้อต่อไป ซึ่งเป็นที่มาของตารางที่ 2.13 นั้นเอง เช่นจากตารางที่ 2.12 ที่ระดับ (ภาค) มีค่าเท่ากับ 4 ค่าของโพลิโนเมียลที่เป็นไปได้คือ 23, 31, 37 ในฐาน 8 มีค่าในเลขฐานสองเท่ากับ 010 011, 011 001, 011 111 ตามลำดับ ดูที่ค่า 37 จะมีคาบน้อยที่สุดคือ 5 ส่วนที่เหลือมีคาบสูงสุด คือ $2^4 - 1 = 15$ ที่ค่าโพลิโนเมียลเป็น $37 = 011\ 111 = x^4 + x^3 + x^2 + x + 1$ เขียนเป็นชิฟร์รีจิสเตอร์ได้เป็นดังรูปที่ 2.20 แสดงชิฟร์รีจิสเตอร์ขนาด 4 ภาค

ดังนั้นเอาพุตของภาคที่จะนำมา XOR กันเพื่อสร้างสัญญาณย้อนกลับให้มีคาบสูงสุดจึงมีค่าโพลิโนเมียลเป็น $23 = 010\ 011$ หรือ $31 = 011\ 001$

จุดประสงค์ของการหาค่าโพลิโนเมียลกับคาบ ที่ระดับ (degree) ต่างๆ คือต้องการสร้างสัญญาณย้อนกลับเพื่อทำให้ได้ชิฟร์รีจิสเตอร์ที่มีคาบสูงสุด ทำให้ได้ค่าสุมมากขึ้นตาม จากเอาพุตของภาคต่างๆ



รูปที่ 2.20 แสดงชิฟร์รีจิสเตอร์แบบสี่ภาคที่สร้างสัญญาณย้อนกลับจากภาค 1, 2, 3, 4

ตารางที่ 2.12 ค่าของโพลิโนเมียล และคาบ

Degree (bold) & polynomial*	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period
1	—	211	127	613	85
2	1	217	127	615	255
3	3	221	127	617	51
4	7	235	127	643	85
5	15	247	127	651	255
6	31	253	127	661	51
7	7	271	127	675	85
8	15	277	127	703	255
9	31	301	127	717	51
10	7	313	127	727	255
11	15	323	127	735	85
12	31	325	127	747	255
13	7	345	127	763	51
14	15	357	127	765	255
15	31	361	127	771	85
16	7	367	127		
17	15	375	127		
18	31				
19	7				
20	15				
21	31				
22	7				
23	15				
24	31				
25	7				
26	15				
27	31				
28	7				
29	15				
30	31				
31	7				
32	15				
33	31				
34	7				
35	15				
36	31				
37	7				
38	15				
39	31				
40	7				
41	15				
42	31				
43	7				
44	15				
45	31				
46	7				
47	15				
48	31				
49	7				
50	15				
51	31				
52	7				
53	15				
54	31				
55	7				
56	15				
57	31				
58	7				
59	15				
60	31				
61	7				
62	15				
63	31				
64	7				
65	15				
66	31				
67	7				
68	15				
69	31				
70	7				
71	15				
72	31				
73	7				
74	15				
75	31				
76	7				
77	15				
78	31				
79	7				
80	15				
81	31				
82	7				
83	15				
84	31				
85	7				
86	15				
87	31				
88	7				
89	15				
90	31				
91	7				
92	15				
93	31				
94	7				
95	15				
96	31				
97	7				
98	15				
99	31				
100	7				
101	15				
102	31				
103	7				
104	15				
105	31				
106	7				
107	15				
108	31				
109	7				
110	15				
111	31				
112	7				
113	15				
114	31				
115	7				
116	15				
117	31				
118	7				
119	15				
120	31				
121	7				
122	15				
123	31				
124	7				
125	15				
126	31				
127	7				
128	15				
129	31				
130	7				
131	15				
132	31				
133	7				
134	15				
135	31				
136	7				
137	15				
138	31				
139	7				
140	15				
141	31				
142	7				
143	15				
144	31				
145	7				
146	15				
147	31				
148	7				
149	15				
150	31				
151	7				
152	15				
153	31				
154	7				
155	15				
156	31				
157	7				
158	15				
159	31				
160	7				
161	15				
162	31				
163	7				
164	15				
165	31				
166	7				
167	15				
168	31				
169	7				
170	15				
171	31				
172	7				
173	15				
174	31				
175	7				
176	15				
177	31				
178	7				
179	15				
180	31				
181	7				
182	15				
183	31				
184	7				
185	15				
186	31				
187	7				
188	15				
189	31				
190	7				
191	15				
192	31				
193	7				
194	15				
195	31				
196	7				
197	15				
198	31				
199	7				
200	15				
201	31				
202	7				
203	15				
204	31				
205	7				
206	15				
207	31				
208	7				
209	15				
210	31				
211	7				
212	15				
213	31				
214	7				
215	15				
216	31				
217	7				
218	15				
219	31				
220	7				
221	15				
222	31				
223	7				
224	15				
225	31				
226	7				
227	15				
228	31				
229	7				
230	15				
231	31				
232	7				
233	15				
234	31				
235	7				
236	15				
237	31				
238	7				
239	15				
240	31				
241	7				
242	15				
243	31				
244	7				
245	15				
246	31				
247	7				
248	15				
249	31				
250	7				
251	15				
252	31				
253	7				
254	15				
255	31				
256	7				
257	15				
258	31				
259	7				
260	15				
261	31				
262	7				
263	15				
264	31				
265	7				
266	15				
267	31				
268	7				
269	15				
270	31				
271	7				
272	15				
273	31				
274	7				
275	15				
276	31				
277	7				
278	15				
279	31				
280	7				
281	15				
282	31				
283	7				
284	15				
285	31				
286	7				
287	15				
288	31				
289	7				
290	15				
291	31				
292	7				
293	15				
294	31				
295	7				
296	15				
297	31				
298	7				

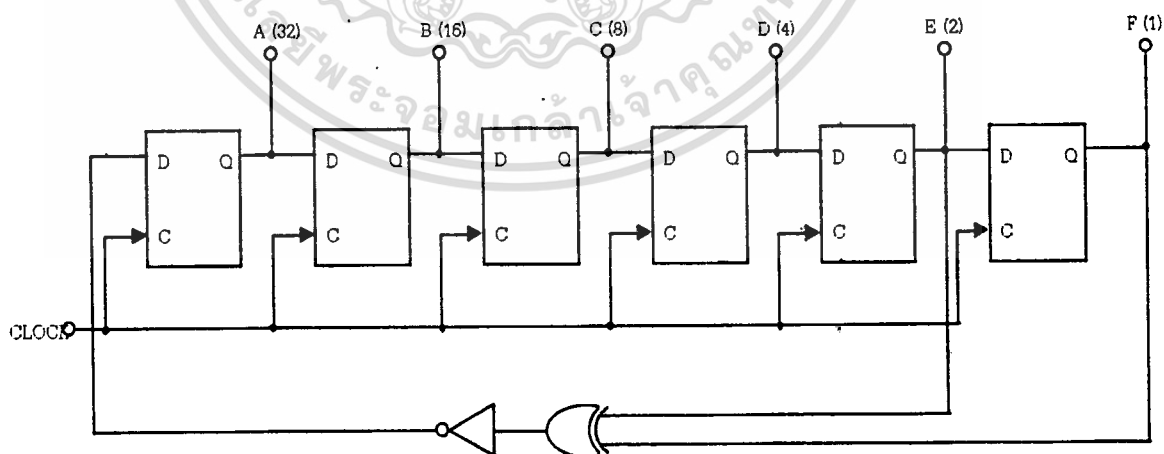
Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period	Degree (bold) & polynomial	Period
1023	3375	4161	2047	4767	2047	5607	2047	6367	2047	7137	2047		
341	3417	4173	2047	5001	2047	5613	2047	6403	2047	7161	2047		
341	3421	4215	2047	5007	2047	5623	2047	6417	2047	7173	2047		
1023	3427	4225	2047	5023	2047	5625	2047	6435	2047	7175	2047		
1023	3435	4237	2047	5025	2047	5657	2047	6447	2047	7201	2047		
1023	3441	4251	2047	5051	2047	5667	2047	6455	2047	7223	2047		
3453	93	4261	2047	5111	2047	5675	2047	6501	2047	7237	2047		
3465	341	4303	89	5111	2047	5711	2047	6507	2047	7243	2047		
3471	1023	4317	2047	5155	2047	5733	2047	6525	2047	7273	2047		
3507	1023	4321	2047	5171	2047	5735	2047	6531	2047	7311	89		
3515	1023	4341	2047	5177	2047	5747	2047	6543	2047	7317	2047		
3525	1023	4347	2047	5205	2047	5755	2047	6557	2047	7335	2047		
3531	1023	4353	2047	5221	2047	6013	2047	6561	2047	7363	2047		
3543	1023	4365	2047	5235	2047	6015	2047	6623	2047	7371	2047		
3573	341	4415	2047	5247	2047	6031	2047	6637	2047	7413	2047		
3575	1023	4423	2047	5253	2047	6037	2047	6651	2047	7431	2047		
3601	341	4445	2047	5263	2047	6061	89	6673	2047	7461	2047		
3607	341	4451	2047	5265	2047	6127	2047	6675	2047	7467	2047		
3615	1023	4467	89	5325	2047	6141	2047	6711	2047	7535	2047		
3623	1023	4473	2047	5337	2047	6153	2047	6727	2047	7553	2047		
3651	341	4475	2047	5343	23	6163	2047	6733	2047	7555	2047		
3661	1023	4505	2047	5351	2047	6165	23	6741	2047	7565	2047		
3705	341	4511	2047	5357	2047	6205	2047	6747	89	7571	2047		
3733	1023	4521	2047	5361	2047	6211	2047	6765	2047	7603	2047		
3763	341	4533	2047	5373	2047	6227	2047	6777	2047	7621	2047		
3763	1023	4563	2047	5403	2047	6233	2047	7005	2047	7627	2047		
3771	1023	4565	2047	5411	2047	6235	2047	7035	2047	7633	2047		
3777	11	4577	2047	5421	2047	6263	2047	7041	2047	7647	2047		
		4603	2047	5463	2047	6277	2047	7047	2047	7655	2047		
		4617	2047	5477	2047	6307	2047	7053	2047	7665	2047		
4005	2047	4653	2047	5501	2047	6315	2047	7063	2047	7715	2047		
4027	2047	4655	2047	5513	2047	6323	2047	7071	2047	7723	2047		
4053	2047	4671	2047	5531	2047	6325	2047	7107	2047	7745	2047		
4055	2047	4707	2047	5537	2047	6343	2047	7113	2047	7751	2017		
4107	2047	4731	2047	5545	2047	6351	2047	7125	2047	7773	2017		
4143	2047	4745	2047	5557	2047								
4145	2047	4757	89	5575	2047								

2.1.4 วงจรสุ่มข้อมูลแบบเทียมขนาด 16 บิต

วงจรสุ่มข้อมูลที่ออกแบบไว้ใช้วิธีการหมุนของข้อมูลในรีจิสเตอร์ โดยข้อมูลที่จะย้อนกลับเข้ามาเกิดจากข้อมูลเดิมในรีจิสเตอร์บางส่วนมาทำการเอ็กซclusive-OR (Exclusive-OR: XOR) กัน สิ่งที่น่าสนใจในการนำสัญญาณย้อนกลับ (Feedback) ของวงจรชิฟท์รีจิสเตอร์ [15, 16] มาใช้คือลำดับข้อมูลสุ่มที่ได้ออกมา ข้อมูลที่ออกมาเหล่านี้มีลักษณะที่ดูอยู่หลายอย่าง ประการแรกมันสามารถมีค่าสูงสุดเท่าไรก็ได้ขึ้นกับจำนวนรีจิสเตอร์ จึงเปรียบได้กับวงจรนับเลขฐานสอง (Binary Counter) นั่นเอง ประการที่สอง ตัวเลขที่ปรากฏเป็นค่าสุ่ม ซึ่งแน่นอนว่ามันจะซ้ำกันทุกๆ 1 ครั้งในจำนวนคล็อก $2^n - 1$ ลูก

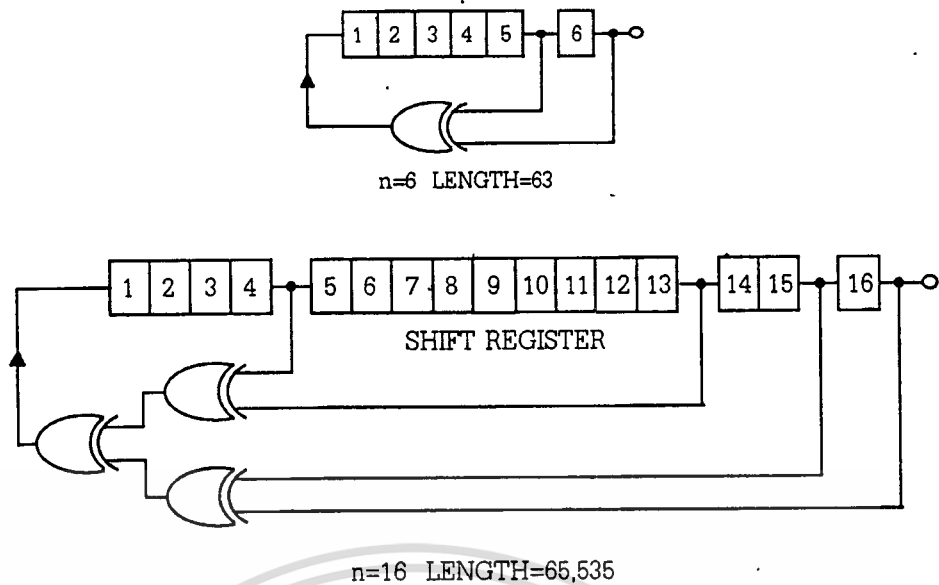
ดังนั้นวงจรเลื่อนข้อมูลในรีจิสเตอร์ที่นำเอาพุตของรีจิสเตอร์มาสร้างเป็นอินพุตใหม่ ตามหลักการที่กล่าวมาเปรียบเสมือนการสร้างค่ารบกวนที่สามารถมีค่าซ้ำได้ (noise repetition) ซึ่งใช้ในการทดสอบระบบวิทยุรักษาความปลอดภัยให้คอมพิวเตอร์ ระบบการเข้ารหัสลับและถอดรหัสลับ และดนตรีอิเล็กทรอนิกส์ ที่จริงการได้สัญญาณรบกวนที่มีค่าซ้ำได้ (Pseudo-noise) มีข้อดีกว่าสัญญาณลบกวนจริงๆ (real noise) ตรงค่าสุ่มแบบเทียมสามารถสร้างได้จากคล็อกหนึ่งลูก ส่วนค่าสุ่มแบบจริงต้องใช้เวลาเฉลี่ยมากกว่า ในการหาค่าสุ่ม

รูปที่ 2.21 แสดงวงจรนับกำหนดข้อมูลสุ่มแบบเทียม 6 ภาค (Stage) ที่ให้ลำดับข้อมูลออกมาได้ $2^6 - 1 = 63$ ค่า นำเอาพุตของฟลิปฟล็อปตัวที่ 5 และ 6 มาเอ็กซclusive-OR กันแล้วใช้เป็นอินพุตใหม่ของฟลิปฟล็อปตัวแรก ข้อมูลขนาดที่ออกมาในขณะที่บิตคล็อกจะซ้ำกันทุกๆ $1/63$ ของจำนวนคล็อกที่บิต ส่วนวงจรในรูปที่ 2.22 แสดงวงจรนับกำหนดข้อมูลสุ่มแบบเทียม 16 ภาค (Stage) ที่ให้ลำดับข้อมูลออกมาได้ $2^{16} - 1 = 65,535$ ค่า นำเอาพุตของฟลิปฟล็อปตัวที่ 4, 13 และ 15, 16 มาเอ็กซclusive-OR กัน ใช้เป็นค่าอินพุตใหม่ของฟลิปฟล็อปตัวแรก ส่วนวิธีการสร้างข้อมูลสุ่มแบบเทียมให้ได้มากขึ้นทำได้โดยการเพิ่มจำนวนรีจิสเตอร์ โดยมีวิธีการเลือกสร้างสัญญาณย้อนกลับต่างกันดังแสดงในตารางที่ 2.13 เพื่อให้ได้ค่าของสถานะมีมากที่สุด



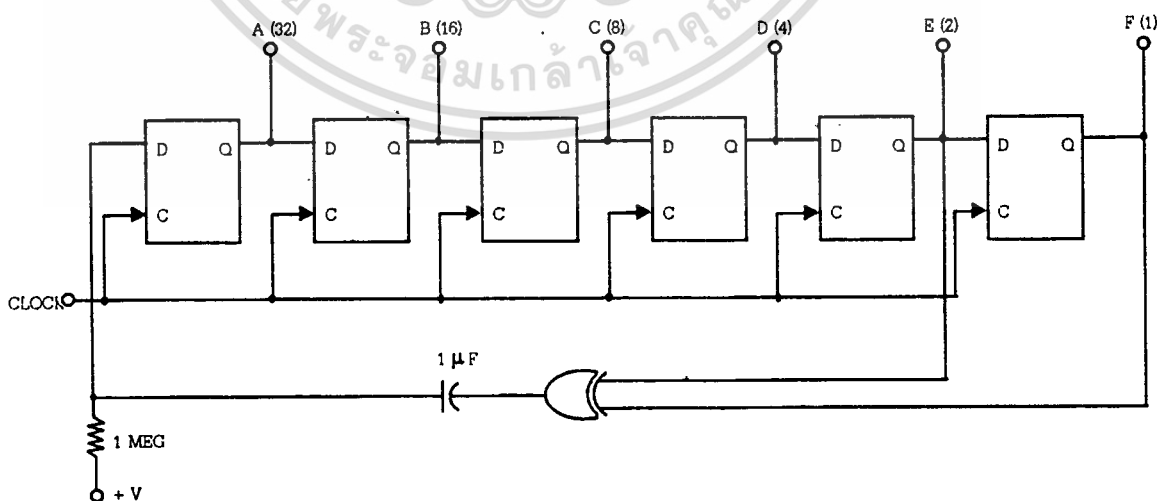
WITH INVERTER ---STATE 111111 IS DISALLOWED; SERIES IS COMPLEMENTARY
WITHOUT INVERTER ---STATE 000000 IS DISALLOWED. SERIES IS NORMAL

รูปที่ 2.21 วงจรกำหนดตัวเลขสุ่มแบบเทียมขนาด 6 ภาค



รูปที่ 2.22 การต่อวงจรสำหรับ 6 และ 16 บิต

สิ่งที่ทำให้การทำงานของวงจรที่ยกตัวอย่างมาทั้งสองทำงานไม่ถูกต้อง ก็คือในกรณีที่ค่าของรีจิสเตอร์ทุกตัวเป็น "0" ในกรณีที่ใช้ XOR และเป็น "1" ในกรณีที่ใช้ XNOR สามารถแก้ไขได้โดยใช้คาร์ปาซิเตอร์ และรีจิสเตอร์ต่อที่เอาพุตของ XOR ดังแสดงในรูปที่ 2.23 ทำให้อินพุตของฟลิปฟล็อปตัวแรกเป็น "1" ในกรณีที่ข้อมูลเอาพุตแบบอนุกรมเป็น "0" ระยะเวลาหนึ่ง เพื่อป้องกันไม่ให้เอาพุตของฟลิปฟล็อปทุกตัวเป็น "0" ตลอด เพราะจะทำให้วงจรไม่สามารถกำหนดข้อมูลสุ่มออกมาได้ต่อไป หรือจะแก้ไขโดยใช้เกทพื้นฐาน นำเอาเอาพุตของรีจิสเตอร์ทั้งหมดมาทำการ NOR กันแล้วป้อนให้อินพุตรีจิสเตอร์ตัวแรก ซึ่งเป็นวิธีการที่นำมาใช้สร้างวงจรสุ่มเพื่อเสริมการทำงานของวงจรฮาร์ดลอค (วงจร MODULE 7 ในภาคผนวก ก. ประกอบ)



รูปที่ 2.23 วงจรที่ป้องกันการเป็น "0" ทุกรีจิสเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 2.13 จำนวนข้อมูลที่ได้จากวงจรสุ่มที่มีภาคต่างกัน

Stages	Sequence Length	Fig.	EXOR Inputs	Stages	Sequence Length	Fig.	EXOR Inputs
2	3	a	1,2	17	131,071	a	14,17
3	7	a	2,3	18	262,143	a	11,18
4	15	a	3,4	19	524,287	b	14,17,18,19
5	31	a	3,5	20	1,048,575	a	17,20
6	63	a	5,6	21	2,097,151	a	19,21
7	127	a	6,7	22	4,194,303	a	21,22
8	255	b	4,5,6,8	23	8,388,607	a	18,23
9	511	a	5,9	24	16,777,215	b	17,22,23,24
10	1023	a	7,10	25	33,554,431	a	22,25
11	2047	a	9,11	26	67,108,863	b	20,24,25,26
12	4095	b	6,8,11,12	27	134,217,727	b	22,25,26,27
13	8191	b	9,11,12,13	28	268,435,455	a	25,28
14	16,383	b	4,8,13,14	29	536,870,911	a	27,29
15	32,767	a	14,15	30	1,073,741,823	b	7,28,29,30
16	65,535	b	4,13,15,16	31	2,147,483,647	a	28,31

2.2 การออกแบบวงจรฮาร์ดลอคเป็นผังวงจรไฟฟ้า

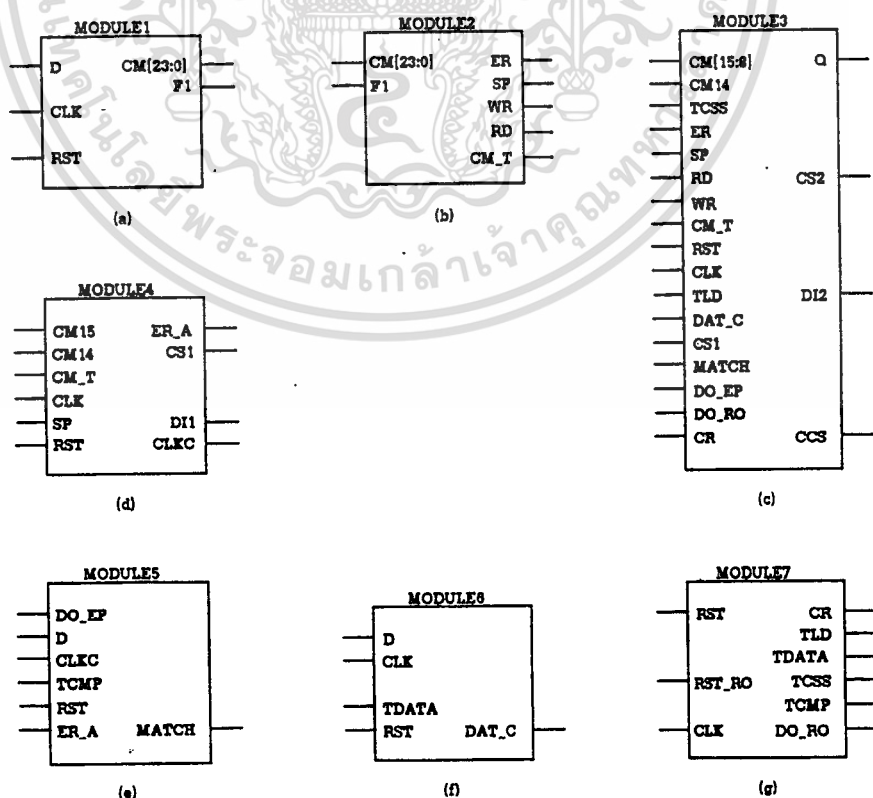
ในหัวข้อนี้จะแบ่งการอธิบายออกเป็นสองส่วน คือ ส่วนที่หนึ่งอธิบายหน้าที่ของวงจรรวมฮาร์ดลอคเป็นส่วนๆอย่างพอสังเขป เพื่อให้เข้าใจได้ง่ายจึงอธิบายลักษณะการทำงานของวงจรเป็นตารางแสดงหน้าที่สำคัญของวงจรทุกส่วนตามตารางที่ 2.14 โดยจะอธิบายเฉพาะขาสัญญานเข้าพุตในกรณีที่ขาสัญญานนั้นเป็นทั้งอินพุตและเข้าพุตของวงจรแต่ละส่วน ส่วนที่สองอธิบายลักษณะการทำงานของฮาร์ดลอค ลักษณะคำสั่งที่ใช้ควบคุมการทำงานของฮาร์ดลอค และวิธีการสร้างชุดคำสั่งอ่านรหัสลับ

2.2.1 รายละเอียดของวงจรฮาร์ดลอค

เพื่อให้สะดวกในการค้นหาจุดบกพร่อง และแก้ไขวงจร วงจรที่ออกแบบจึงได้แยกออกแบบเป็นส่วนๆตามลักษณะการทำงาน มีทั้งหมด 7 ส่วน ดังแสดงในรูปที่ 2.24 (a)-(g) ซึ่งแต่ละส่วนมีหน้าที่ดังนี้
หมายเหตุ ขาสัญญานด้านซ้ายเป็นอินพุต ส่วนขาสัญญานด้านขวาเป็นเข้าพุต

MODULE1
เป็นรีจิสเตอร์ขนาด 1 (Start bit)+24 บิต ใช้สำหรับเก็บรหัสคำสั่ง (Command Word) ระโยชน์ด้านการค้า

MODULE2
ถอดรหัสคำสั่งหลังจากที่รับคำสั่งเข้ามาครบ 24 บิต
MODULE4
สร้างสัญญาณเพื่อทำการอ่านรหัสผ่านจาก EEPROM สองชุด ชุดละ 16 บิต
MODULE5
เปรียบเทียบข้อมูลขนาด 32 บิต ระหว่างรหัสผ่านที่อ่านออกมาจาก EEPROM กับข้อมูลส่วนที่เป็น Password
MODULE3
สร้างสัญญาณป้อนให้กับ EEPROM ตามรหัสคำสั่งที่รับมาตอนแรกหลังจากการตรวจสอบรหัสคำสั่งแล้วถูกต้อง และมีรหัสผ่าน (Password Code) ที่รับเข้ามาตรงกับรหัสชี้เฉพาะ (Identify Code) ที่มีอยู่ในตัวฮาร์ดลอค
MODULE6
เก็บและส่งข้อมูล (Data) ที่ผ่านการเข้ารหัส และถอดรหัส เพื่อเขียนลง EEPROM ซึ่งใช้เป็นรหัสผ่านหรือเป็นรหัสชี้เฉพาะได้ต่อไป
MODULE7
สร้างข้อมูลรูปแบบเทียมในกรณีที่มีการติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ กับฮาร์ดลอคเกิดความไม่ถูกต้อง และเป็นส่วนกำหนดสัญญาณควบคุมจังหวะการทำงานของส่วน MODULE5,MODULE6,MODULE3



รูปที่ 2.24 (a-g) แสดงขาสัญญาณของ MODULE1-MODULE7

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

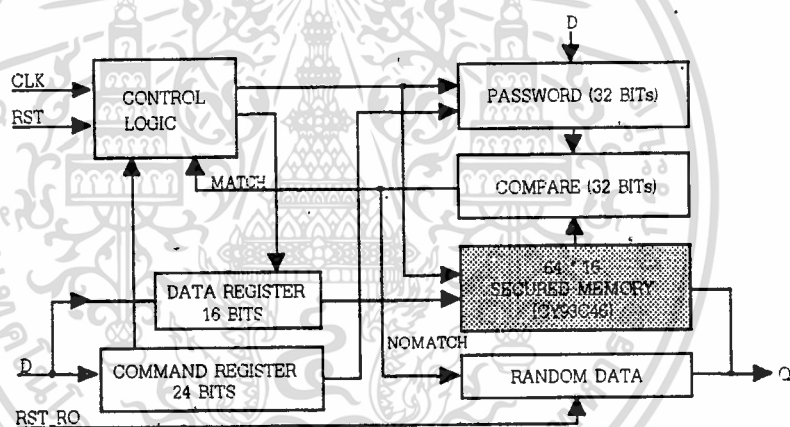
รูปแสดงการเชื่อมต่อของวงจรทุกส่วนและรูปวงจร MODULE1-MODULE7 แสดงไว้ในภาคผนวก ก.

ตารางที่ 2.14 อธิบายการทำงานของขาสัญญาณต่างๆของวงจรทุกส่วน

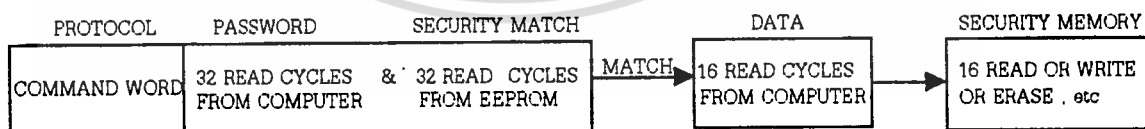
ขาสัญญาณ	การทำงาน
D	รับข้อมูลแบบอนุกรมเข้ามาตามจังหวะสัญญาณนาฬิกา
RST	รีเซ็ตการทำงานของฮาร์ดลอค
CLK	สัญญาณนาฬิกาของฮาร์ดลอค
CM[23:0]	เป็นเอาต์พุตของข้อมูล 24 บิตที่รับเข้า
F1	แอดทิฟ "1" เมื่อข้อมูลคำสั่งเข้ามาครบ
ER,SP,WR,RD	แอดทิฟ "0" เมื่อเป็นคำสั่งลบ พิเศษ เขียน และอ่านข้อมูล
CM_T	แอดทิฟ "0" เมื่อคำสั่งที่รับเข้ามาถูกต้อง
ER_A	แอดทิฟ "0" เมื่อเสร็จสิ้นการอ่านรหัสผ่านขนาด 32 บิตจาก EEPROM
CS1	แอดทิฟ "0" รีเซ็ตการทำงานของ EEPROM เมื่ออ่านรหัสผ่าน
DI1	ป้อนคำสั่งให้กับ EEPROM หลังจาก CS1 แอดทิฟ
CLKC	ให้จังหวะการเปรียบเทียบรหัสผ่านที่ป้อนกับรหัสผ่านที่อ่านจาก EEPROM
MATCH	แอดทิฟ "0" รหัสผ่านกับรหัสผ่านที่อ่านจาก EEPROM เหมือนกัน
Q	เป็นเอาต์พุตของข้อมูลที่อ่านจาก EEPROM หรือข้อมูลสุ่ม
CS2	เป็นการนำสัญญาณ CS1 กับ CCS เข้าด้วยกัน
DI2	ป้อนคำสั่งให้กับ EEPROM หลังจาก CS2 แอดทิฟ
CCS	แอดทิฟ "0" รีเซ็ตการทำงานของ EEPROM เมื่อทำตามคำสั่ง
DATA_C	ทางผ่านของข้อมูล (DATA) 16 บิตที่จะเขียนลง EEPROM
RST_RO	รีเซ็ตการทำงานของวงจรสร้างข้อมูลสุ่ม
CR	ให้จังหวะการได้อัตลักษณ์ DO_EP กับ DO_RO เป็นเอาต์พุตที่ขา Q
TLD	ใช้โหลดคำสั่งเพื่อป้อนให้ EEPROM หลังจาก CS2 แอดทิฟ
TDATA	ให้จังหวะเพื่อนำข้อมูลที่เก็บไว้จากทาง DATA_C
TCSS	ควบคุมการโหลดคำสั่งเพื่อป้อนให้ EEPROM หลังจาก CS2 แอดทิฟ
TCMP	ควบคุมการเปรียบเทียบรหัสผ่านที่ป้อนกับรหัสผ่านที่อ่านจาก EEPROM
DO_RO	ทางออกของข้อมูลสุ่มที่สร้างเพื่อหลอกผู้แอบดูข้อมูล

2.2.2 ลักษณะการทำงานของฮาร์ดล๊อค

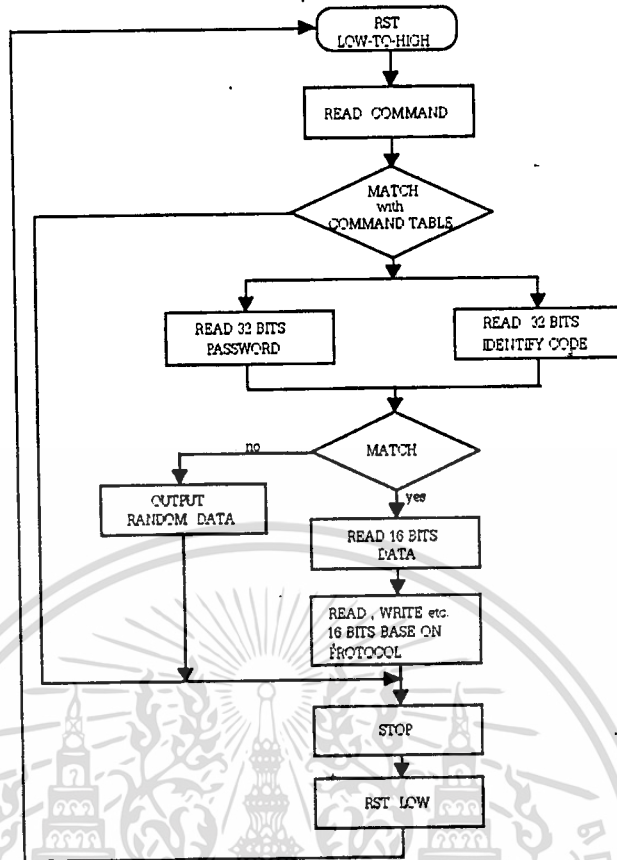
การรับข้อมูลของฮาร์ดล๊อคเริ่มขึ้นเมื่อมีสัญญาณ RST (LOW-TO-HIGH) และสัญญาณ RST_RO (LOW-TO-HIGH) ไปเซตส่วนของ Random Data จากนั้นฮาร์ดล๊อคก็จะรับรหัสคำสั่ง 24 บิตหน้าหน้าด้วย '1' เข้ามาเก็บใน Command Register ทางขา D ตามสัญญาณ CLK ขาขึ้น ซึ่งจะต้องตรงตามตารางที่ 2.3 (ถ้ารหัสคำสั่งไม่ตรงการติดต่อกับฮาร์ดล๊อคจะถูกยกเลิก) เมื่อตัวฮาร์ดล๊อคตรวจสอบรหัสคำสั่งถูกต้องแล้วจึงรับข้อมูลต่ออีก 32 บิต (Password) ตามสัญญาณ CLK ขาขึ้น ในขณะเดียวกันบล็อก Compare ก็จะอ่านข้อมูลออกมาจาก EEPROM มา 2 ชุด ชุดละ 16 บิต ตามสัญญาณ CLK ขาลง เพื่อเปรียบเทียบกัน ถ้าข้อมูลตรงกันตัวฮาร์ดล๊อคก็พร้อมที่จะทำตามคำสั่งดังตารางที่ 2.3 (อ่าน READ เขียน WRITE และ ลบ ERASE ข้อมูลที่ละแอดเดรส อีเนเบิล /ดีสเอบิลการเขียนและลบ EWEN/EWDS เขียน และ ลบข้อมูลทุกแอดเดรส WRAL/ERAL) ที่ป้อนให้ ตรงกันข้ามถ้าเปรียบเทียบกันแล้วข้อมูลไม่ตรงกันตัวฮาร์ดล๊อคจะส่งข้อมูลออกมาที่ขา Q ดังรูปที่ 2.25-2.27 ประกอบ



รูปที่ 2.25 องค์ประกอบหลักของฮาร์ดล๊อค



รูปที่ 2.26 ลำดับการรับข้อมูลคำสั่งของฮาร์ดล๊อค



รูปที่ 2.27 ไฟล์ชาร์ตแสดงการทำงานของฮาร์ดลิค

ดังที่ได้อธิบายลักษณะการทำงานของฮาร์ดลิคไว้แล้วข้างต้น จะเห็นว่าจำนวนข้อมูลที่ใช้ติดต่อกับฮาร์ดลิคทั้งหมด แบ่งออกเป็น 3 ส่วนคือ Command Word (24 บิต) มี Start Bit "1" นำหน้า , Password (32 บิต) และ Data (16 บิต) เรียงตามลำดับ ซึ่งมีรายละเอียดดังนี้

Command Word มีทั้งหมด 24 บิต แบ่งเป็น 5 ส่วนดังนี้

$(K_H K_L)$ = กำหนดส่วนของข้อมูลที่เข้าถึงได้		
$(K_H K_L)_C$ = คอมพ्लीเมนต์ของ $(K_H K_L)$		
$(A_5 A_4 A_3 A_2 A_1 A_0)$ = ส่วนของแอดเดรส		
$(A_5 A_4 A_3 A_2 A_1 A_0)_C$ = คอมพ्लीเมนต์ของ $(A_5 A_4 A_3 A_2 A_1 A_0)$		
$(F_7 F_6 F_5 F_4 F_3 F_2 F_1 F_0)$ = ส่วนรหัสฟังก์ชัน กำหนดลักษณะการทำงานดังรูปที่ 4.3		

[CM ₂₄ -CM ₁₆]	[CM ₁₅ -CM ₈]	[CM ₇ -CM ₀]
$(K_H K_L)_C (A_5 A_4 A_3 A_2 A_1 A_0)_C$	$(K_H K_L) (A_5 A_4 A_3 A_2 A_1 A_0)$	$F_7 F_6 F_5 F_4 F_3 F_2 F_1 F_0$

ตารางที่ 2.15 แสดงฟังก์ชันการทำงานแต่ละคำสั่ง

COMMAND	VALID PARTITION CODE K _H K _L	VALID BYTE ADDRESS A ₅ ... A ₀	VALID FUNCTION CODE F ₇ ... F ₀
READ	11	0-63	0 1 1 0 0 1 1 0
WRITE	11	0-63	1 0 0 1 1 0 0 1
ERASE	11	0-63	0 1 0 1 0 1 0 1
READ	00,01,10	16-63	0 1 1 0 0 1 1 0
WRITE	00,01,10	16-63	1 0 0 1 1 0 0 1
ERASE	00,01,10	16-63	0 1 0 1 0 1 0 1
EWEN	XX	1 1 X X X X	0 1 0 1 1 0 1 0
EWDS	XX	0 0 X X X X	0 1 0 1 1 0 1 0
ERAL	XX	1 0 X X X X	0 1 0 1 1 0 1 0
WRAL	XX	0 1 X X X X	0 1 0 1 1 0 1 0

Password เป็นข้อมูลขนาด 32 บิต มีค่าเป็นอย่างไรขึ้นอยู่กับแอดเดรสที่ต้องการ อ่าน เขียน และลบข้อมูล เช่นถ้าต้องการอ่านข้อมูลที่อยู่ในแอดเดรส 0-15 จะต้องป้อนรหัสผ่าน (Password) ให้ตรงกับค่าคอมพลีเมนต์ของรหัสชี้เฉพาะ (Identify Code) ที่แอดเดรส 0 และ 1 รวมกันมีขนาดเท่ากับ 32 บิต ซึ่งรายละเอียดการเลือกใช้รหัสชี้เฉพาะเป็นไปตามตารางที่ 2.16

ตารางที่ 2.16 แอดเดรสที่ใช้เป็นรหัสผ่าน

คีย์ฟังก์ชัน K _H K _L	แอดเดรสที่มองเห็น	อ่านรหัสชี้เฉพาะจากแอดเดรส
1 1	0- 63 (000000-111111)	0 และ 1
0 0	16-31 (010000-011111)	16 และ 17
0 1	32-47 (100000-101111)	32 และ 33
1 0	48-63 (110000-111111)	47 และ 48

Data เป็นข้อมูลขนาด 16 บิตที่ใช้เขียนลงบน EEPROM ที่อยู่ภายในฮาร์ดล็ค โดยค่าที่เขียนจะเป็นค่าที่เข้ารหัสของข้อมูล ดูรายละเอียดในหัวข้อการเข้าและถอดรหัสลับ

ตัวอย่าง ลำดับการสร้างสัญญาณเพื่ออ่านข้อมูลแอดเดรส 37 D ที่อยู่ใน EEPROM มีองค์ประกอบของข้อมูลคำสั่งเป็นดังนี้

F7 ... F0 เป็น 0110 0110 (CM7-CM0) เนื่องจากต้องการอ่านรหัสลับ
A5 ... A0 เป็น 100101 (CM13-CM8) อ่านที่แอดเดรส 37 D=100101 B
KH KL (CM15 ,CM14) เป็น 0 1 โดยได้ออกแบบให้ฮาร์ดล๊อคอ่านข้อมูลจากแอดเดรส ที่ 32 , 33 (Security Match) ดูรายละเอียดในตารางที่ 5.4
CM24-CM16 เป็น 9A H (ค่าคอมพลิเมนต์ของ CM15-CM8= 65 H)
ดังนั้น Password (EXM31-EXM0) ที่ป้อนเข้าไปจะต้องมีข้อมูลสอดคล้องกับ รหัสชี้เฉพาะของฮาร์ดล๊อคที่อยู่ในแอดเดรส 32 และ 33 โดย EXM31-EXM16 จะเท่ากับ คอมพลิเมนต์ของแอดเดรส 32 EXM15-EXM0 จะเท่ากับ คอมพลิเมนต์ของแอดเดรส 33 ถ้า ให้แอดเดรส 32 ของฮาร์ดล๊อคมีรหัสชี้เฉพาะเป็น 20000D (4E20 H) และ ให้แอดเดรส 33 ของฮาร์ดล๊อคมีรหัสชี้เฉพาะเป็น 10000D (2710 H) ดังนั้น EXM31-EXM0 = B1DFD8EF H
จากขั้นตอนข้างบนจะได้ลำดับข้อมูลคำสั่งที่ป้อนให้ฮาร์ดล๊อคทางขา D ทั้งหมดเป็นดังนี้ ["1"] + [CM23- CM0] + [EXM31- EXM0] + [D15- D0] ["1"] + [9A6566 H] + [B1DFD8EF H] + [0000 H]

ข้อมูลที่ส่งออกมาที่ขา Q หลังจากที่ถูกโอนคำสั่ง จึงเป็นข้อมูลถอดรหัสของรหัสลับที่แอดเดรส 37 D นั่นเอง

ถ้า รหัสลับที่อยู่ในแอดเดรส 37 D เป็น 9B88 H

ข้อมูลถอดรหัสคือ 1388 H ดูรายละเอียดในหัวข้อการเข้าและถอดรหัส

* ถ้าป้อนข้อมูลค่าสิ่งที่ป้อนไม่ถูกต้อง ฮาร์ดดิสก์จะส่งข้อมูลออกมาที่ขา Q เช่นเดียวกัน

2.2.3 การจัดเก็บข้อมูลของฮาร์ดดิสก์

ข้อมูลที่เกิดขึ้นในฮาร์ดลอคเป็นข้อมูลที่ต้องการป้องกันการเข้าถึงของ EEPROM นั่นเอง ซึ่งข้อมูลทั้งหมด 64 แอดเดรสได้แบ่งไว้เก็บข้อมูลสองส่วนคือ เก็บข้อมูลรหัสชี้เฉพาะ (Identify code) และเก็บรหัสลับ (Secret code) จะจัดแบ่งไว้ดังรูปที่ 2.28 โดยให้หนึ่งบล็อกละแทนข้อมูลขนาด 16 บิตในรีจิสเตอร์แต่ละตัว

ในกรณีที่ฮาร์ดดิสก์จะทำการอ่านข้อมูลจากแอดเดรส 7 ฮาร์ดดิสก์จะอ่านข้อมูลที่สล็อตเฉพาะแอดเดรส 0 และ 1 มาเปรียบเทียบกับข้อมูลที่เป็นรหัสผ่านถ้าตรงกันรหัสลับที่อยู่ในที่แอดเดรส 7 จะถูกอ่านออกมาผ่านวงจรถอดรหัสลับ ที่ขา Q

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

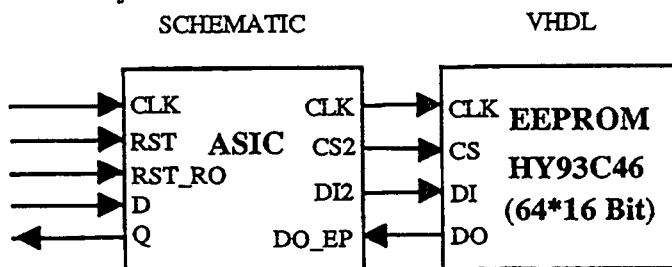
 รหัสชี้เฉพาะ

☐ รหัสลับ

รูปที่ 2.28 การจัดแบ่งข้อมูลที่อยู่ภายในชาร์ดลีด

2.3 การออกแบบหน่วยความจำชนิด EEPROM โดยใช้ VHDL

เพื่อให้ได้วงจรลัดที่สมบูรณ์ หลังจากที่ได้ออกแบบวงจรลัดเป็นแผงวงจรไฟฟ้าแล้ว จะต้องออกแบบหน่วยความจำด้วย จากนั้นจึงนำวงจรลัด และหน่วยความจำที่ออกแบบไว้มาสร้างเป็น Symbol มาเชื่อมต่อกัน ดังแสดงรูปที่ 2.29



รูปที่ 2.29 บล็อกแสดงการเชื่อมต่อ Symbol ของชาร์ดล็ค และหน่วยความจำ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.1 องค์ประกอบพื้นฐานใน VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบใน VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 2.30 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำ ENTITY ตามด้วยชื่อขององค์ประกอบและคำ IS ภายในบรรยายถึงพอร์ตการติดต่อ อินพุต-เอาพุตพอร์ตขององค์ประกอบ ส่วนลักษณะภายนอกอื่นๆ เช่น เวลาและอุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุต-เอาพุตและพารามิเตอร์อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อ การบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังจากคำว่า BEGIN เป็นต้นไป จบด้วย END

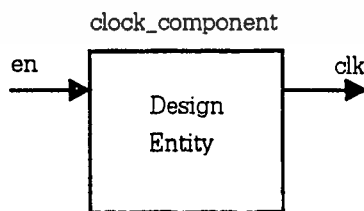
```
ENTITY entity_name IS
    [ PORT (name_list : mode type ; ... ; name_list : mode type ) ; ]
END [ entity_name ] ;
```

```
ARCHITECTURE body_name OF entity_name IS
    -- declarative_statements
BEGIN
    -- activity_statements
END [ body_name ] ;
```

รูปที่ 2.30 แสดงการกำหนดการเชื่อมต่อและสถาปัตยกรรม

2.3.1.1 การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับารติดต่อกับองค์ประกอบภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 2.31 แสดงบล็อกไดอะแกรม และการบรรยายการเชื่อมต่อ



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

รูปที่ 2.31 แสดงบล็อกไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_component

2.3.1.2 การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเข้าพุทในเทอมของอินพุตหรือในรูปขององค์ประกอบอื่นๆหรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.32 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมมี en เป็นอินพุตและ clk เป็นเอาพุต PROCESS เป็นคำเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม ภายในโปรแกรมกำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น '0' ถ้าสัญญาณ en มีค่าเป็น '1' ค่าของ periodic จะถูกคอมพลีเมนต์ (Complement) และส่งค่าให้กับ clk ซึ่งเป็นสัญญาณเอาพุต คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเป็นเวลา 1 ไมโครวินาที

```

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

```

รูปที่ 2.32 แสดงการบรรยายเชิงพฤติกรรมของ clock_component

2.3.1.3 โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมชั้นสูงทั่วไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมี หรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าเราใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรจริง ๆ ในขณะที่ถ้าเราใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 2.33 แสดงการใช้ฟังก์ชันโดยกำหนดให้ x เป็นตัวแปรชนิดบิตแทนการกระทำในสมการบูลีน รูปที่ 2.34 แสดงโพรซีเจอร์ที่เป็นตัวกลับสัญญาณ

```

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

```

รูปที่ 2.33 แสดงตัวอย่างฟังก์ชัน

```

LIBRARY mgc_portable ;
USE mgc_portable.qsim_logic.ALL ;
ENTITY invert_example IS
    PORT ( in_a : IN qsim_state ;
           out_b : OUT qsim_state ) ;
END invert_example ;
ARCHITECTURE alg OF invert_example IS
    PROCEDURE qsim_inverter ( SIGNAL bit_in : IN qsim_state ;
                              SIGNAL bit_out : OUT qsim_state ) IS
    BEGIN
        IF bit_in = '1' THEN
            bit_out <= '0' ;
        ELSIF bit_in = '0' THEN
            bit_out <= '1' ;
        ELSE
            bit_out <= 'X' ;
        ENDIF ;
    END qsim_inverter ;
BEGIN
    qsim_inverter ( in_a, out_b ) ;
END alg ;

```

รูปที่ 2.34 แสดงตัวอย่างโปรซีเจอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.1.4 โอเปอเรเตอร์

การบรรยายเชิงพฤติกรรมใน VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกันกับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 2.35

PREDEFINED OPERATORS	
LOGICAL OPERATORS	: AND, OR, NAND, NOR, XOR, NOT
RELATIONAL OPERATORS	: =, /=, <, <=, >, >=
MISCELLANEOUS OPERATORS	: &, +, -
ARITHMETIC OPERATORS	: **, ABS, */, MOD, REM, +, -

รูปที่ 2.35 แสดงโอเปอเรเตอร์ใน VHDL

2.3.1.5 เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์ทุกตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุกๆเหตุการณ์ที่เกิดขึ้น VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบและการป้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยาย (Architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายๆโปรเซสอยู่ในโครงสร้างเดียวกัน ทุกๆโปรเซสก็จะทำงานไปพร้อมๆกันด้วย

2.3.1.6 สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเหมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ $\leq$ ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น $w \leq a \text{ AFTER } 12 \text{ ns}$. หมายถึงกำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 ns.

ในทางตรงกันข้าม ตัวแปรมีลักษณะเป็นเหมือนตัวกลางซอฟต์แวร์ที่ใช้ในการส่งผ่านข้อมูลและไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ $:=$ ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชัน โปรซีเจอร์ และโปรเซส

2.3.2 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรมเป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสถานะของข้อมูลที่เข้ามาโดยไม่คำนึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในจะ

2.3.2.1 โพรเซส

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติคำสั่งพร้อมๆกันกับโพรเซสอื่นๆที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับสัญญาณที่อยู่ด้านขวามือของสัญลักษณ์การกำหนดค่าให้กับสัญญาณ ($\leq$)

การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS รูปที่ 2.36 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ

Format:
<pre>[label :] PROCESS -- declarative_statements BEGIN -- sequential_activity_statements END PROCESS [label] ;</pre>
Example:
<pre>la_bel : PROCESS BEGIN inter_mediate <= a AND b ; c <= NOT inter_mediate ; WAIT ON a , b ; END PROCESS la_bel ;</pre>

รูปที่ 2.36 แสดงรูปแบบและตัวอย่างของการบรรยายแบบโพรเซส

2.3.2.2 การกำหนดตัวกระทำภายในโพรเซส

ตัวกระทำภายในโพรเซสมี 3 ชนิดคือตัวแปร (variable) ไฟล์ (file) และตัวคงที่ (constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใดก็จะใช้เฉพาะโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซสสามารถทำได้โดยใช้สัญญาณ (signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ Architecture รูปที่ 2.37 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN คำเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียวเท่านั้น แต่ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้โปรแกรมย่อยนั้นๆ

```

PROCESS

    FILE flush : TEXT IS IN "filename.dat" ;

    VARIABLE var : BIT;

    CONSTANT n : INTEGER :=0 ;

BEGIN

    .....

END PROCESS ;

```

รูปที่ 2.37 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส

2.3.2.3 การกำหนดการกระทำภายในโพรเซส

การกระทำใดๆภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequencial) เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการกระทำซ้ำได้เช่น IF-THEN-ELSE, CASE-WHEN, FOR LOOP และ WHILE LOOP ดังตัวอย่างในรูปที่ 2.38

```

ARCHITECTURE demo OF partail_process IS

    .....

BEGIN

    PROCESS

        .....

        BEGIN

            .....

            x <= '1' ;

            IF x = '1' THEN

                perform action_1

            ELSE

                perform action_2

            END IF;

            ....

        END PROCESS;

    END demo;

```

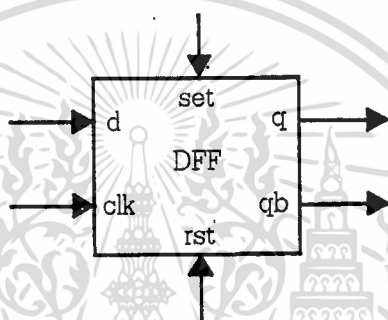
รูปที่ 2.38 แสดงเงื่อนไขการกระทำในโพรเซส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.2.4 การกระตุ้นและยับยั้งการกระทำของโปรเซส

การกระทำภายในโปรเซสจะเตรียมพร้อมและมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ตามเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโปรเซสได้โดยกำหนดรายการของสัญญาณที่ต้องการให้โปรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้น เหตุการณ์ใด ๆ ที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโปรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS

รูปที่ 2.39 แสดง (a) ตัวอย่างโมเดล และ (b) ตัวอย่างการบรรยายการเชื่อมต่อของ D-FlipFlop รูปที่ 2.39 (c) แสดงการบรรยายเชิงพฤติกรรมของ D-FlipFlop โดยมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโปรเซส



รูปที่ 2.39 (a) โมเดล D-FlipFlop

```
ENTITY dff_sr IS
    PORT (d, set, rst, clk : IN BIT ; q, qb : OUT BIT) ;
END dff_sr ;
```

รูปที่ 2.39 (b) การบรรยายการเชื่อมต่อของ D-FlipFlop

รูปที่ 2.39 (c) การบรรยายเชิงพฤติกรรมของ D-FlipFlop

```
ARCHITECTURE behavioral OF dff_sr IS
    SIGNAL state : BIT := '0' ;
BEGIN
    dff : PROCESS (rst, set, clk)
    BEGIN
        IF set = '1' THEN
            state <= '1' AFTER 6 NS ;
```

รูปที่ 2.39 (c) ต่อ

```

        ELSIF rst = '1' THEN
            state <= '0' AFTER 6 NS ;
        ELSIF clk = '1' AND clk'EVENT THEN
            state <= d AFTER 6 NS ;
        END IF ;
    END PROCESS dff ;

    q <= state ;
    qb <= NOT state ;

END behavioral ;

```

2.3.2.5 การหยุดรอ

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณเมื่ออยู่ 4 รูปแบบดังนี้

```

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

```

คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซส และโพรซีเจอร์ที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น WAIT FOR จะหยุดรอเป็นเวลาเท่ากับ waiting_time WAIT ON จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ waiting_sensitivity_list WAIT UNTIL จะหยุดรอจนกว่าเงื่อนไข waiting_condition เปลี่ยนจาก FALSE เป็น TRUE WAIT จะหยุดรอตลอดไป

2.3.2.6 การสร้างองค์ประกอบย่อย

การสร้างองค์ประกอบย่อยเพื่อใช้เป็นไลบรารี เป็นการบรรยายเชิงโครงสร้างซึ่งจะต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่างๆด้วย ประกาศไว้ในส่วนของ Architecture ประกอบด้วย 3 ส่วนตามลำดับ คือส่วนประกาศองค์ประกอบ (Component Declaration) ส่วนที่แสดงรายละเอียดขององค์ประกอบ (Component Specification) ส่วนที่กำหนดการเชื่อมต่อของพอร์ตกับสัญญาณขององค์ประกอบ (Component Instantiation) รูปที่ 2.40 แสดงการบรรยายการเชื่อมต่อของวงจรเลือกสัญญาณ 2 ออก 1 (Multiplexer)

COMPONENT DECLARATION

Format:

```
COMPONENT identifier_name
    PORT (local_port_list);
END COMPONENT;
```

COMPONENT SPECIFICATION

Format:

```
FOR component_instantiation_label : component_name USE binding_indication ;
Binding_indication:
ENTITY library.entity_name [ (architecture_name) ]
```

COMPONENT INSTANTIATION

Format:

```
label : component_declaration_name PORT MAP (association_list) ;
```

รูปที่ 2.40 แสดงการบรรยายเชิงโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1

```
ENTITY mux IS
    PORT (d0, d1, sel : IN bit ; q : OUT bit) ;
END mux ;
ARCHITECTURE struct OF mux IS
    COMPONENT and2
        PORT (a, b : IN bit ; z : OUT bit) ;
    END COMPONENT ;
    COMPONENT or2
        PORT (a, b : IN bit ; z : OUT bit) ;
    END COMPONENT ;
    COMPONENT inv
        PORT (i : IN bit ; z : OUT bit) ;
    END COMPONENT ;
    SIGNAL aa,ab, nsel : bit ;
    FOR u1 :inv USE ENTTTY invrt(behav) ;
    FOR u2,u3 :and2 USE ENTTTY and_gt(dflw) ;
    FOR u4 :or2 USE ENTTTY or_gt(arch1) ;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้ทำซ้ำโดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 2.40 (ต่อ)

```

BEGIN

    u1 : inv    PORT  MAP (sel, nsel) ;

    u2 : and2   PORT  MAP (nsel, d1, ab) ;

    u3 : and2   PORT  MAP (d0, sel, aa) ;

    u4 : or2    PORT  MAP (aa,ab,q) ;

END struct;

```

2.3.2.7 แพคเกจ

แพคเกจคือกลุ่มของชนิดข้อมูล โปรแกรมย่อยหรืออุปกรณ์ย่อยต่างๆที่ผู้ออกแบบได้ออกแบบไว้แล้ว นำมารวบรวมไว้เป็นกลุ่มๆอยู่ในไลบรารีเพื่อให้ผู้ออกแบบสามารถเรียกใช้ได้โดยสะดวกดังรูปที่ 2.41

```

PACKAGE identifier IS
    package_declarative_part
END identifier;

```

```

PACKAGE BODY identifier IS
    package_declarative_part
END identifier;

```

รูปที่ 2.41 แสดงโครงสร้างของแพคเกจ

2.3.3 การบรรยายการทำงานระดับพฤติกรรมของ EEPROM

สร้างองค์ประกอบย่อยชื่อ RAM เพื่อทำหน้าที่เป็นหน่วยความจำใช้เก็บข้อมูล สร้างแพคเกจ basic_utilities ไว้เก็บโปรแกรมที่จะใช้แปลงชนิดข้อมูล และสร้างองค์ประกอบหลัก EEPROM เพื่อทำหน้าที่ควบคุมตามคำสั่งที่รับเข้ามา (ดูตารางที่ 3.2 ประกอบ) โดยจะเชื่อมต่อกับส่วนของ RAM เพื่อให้อ่านหรือเขียนข้อมูล

2.3.3.1 กำหนดไลบรารี

เมื่อเริ่มเขียนคำสั่งของภาษา VHDL จำเป็นต้องบอกให้โปรแกรมรู้ว่าชนิดของข้อมูลที่จะใช้เป็นอย่างใด ซึ่งภาษา VHDL ของ Mentor Graphic ได้สร้างไว้แล้วโดยเก็บอยู่ในไลบรารี mgc_portable แพคเกจชื่อ qsim_logic เขียนเป็นคำสั่งได้ดังรูปที่ 2.42 ตัวอย่างชนิดข้อมูลที่มีอยู่ในแพคเกจ qsim_logic เช่น bit มีชนิดข้อมูลเป็น '0' และ '1' qsim_state มีชนิดข้อมูลเป็นลอจิก '0' , '1' , 'Z' และ 'X'

qsim_state_resolved_x_vector มีชนิดข้อมูลเป็นอะเรย์ของ qsim_state แต่จะใช้กับสัญญาณที่นำเอาพุตมาต่อเข้าด้วยกัน ซึ่งใช้ในกรณีบัสข้อมูลของหน่วยความจำที่มีหลายแอดเดรส ฯลฯ

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
```

รูปที่ 2.42 กำหนดไลบรารี

2.3.3.2 Basic Utilities Package

แพ็คเกจ basic_utilities เป็นแพ็คเกจที่อยู่ในไลบรารี "util" ภายในเก็บโปรซีเจอร์ แปลงข้อมูลจาก qsim_state_vector เป็น integer (vec2int) และ integer เป็น qsim_state_vector (int2vec) ดังแสดงในรูปที่ 2.43 qsim_state เป็นชนิดของข้อมูลที่ประกอบด้วยค่าของลอจิก 4 ค่าคือ '0', '1', 'Z' และ 'X' ในขณะที่ qsim_state_vector เป็นชนิดข้อมูลแบบอะเรย์ของ qsim_state

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
PACKAGE basic_utilities IS
  PROCEDURE vec2int(vec:IN qsim_state_vector,int:OUT INTEGER);
  PROCEDURE int2vec(int:IN INTEGER;vec:OUT qsim_state_vector);
END basic_utilities;
PACKAGE BODY basic_utilities IS
  PROCEDURE vec2int(vec:IN qsim_state_vector,int:OUT INTEGER) IS
    VARIABLE result:INTEGER;
  BEGIN
    result:=0;
    FOR i IN 0 TO (vec'LENGTH-1) LOOP
      IF vec(i)='1' THEN
        result:=result+2**i;
      END IF;
    END LOOP;
    int:=result;
  END vec2int;
  PROCEDURE int2vec(int:IN INTEGER;vec:OUT qsim_state_vector) IS
    VARIABLE tmp:INTEGER;
  BEGIN
    tmp:=int;
    FOR i IN 0 TO (vec'LENGTH-1) LOOP
      IF (tmp MOD 2=1) THEN
        vec(i):='1';
      ELSE vec(i):='0';
      END IF;
      tmp:=tmp/2;
    END LOOP;
  END int2vec;
END basic_utilities;
```

รูปที่ 2.43 แสดงแพ็คเกจ basic_utilities

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3.3.3 การบรรยายพฤติกรรมหน่วยความจำภายในของ EEPROM

หน่วยความจำของ EEPROM มีทั้งหมด 64 แอดเดรส โดยแต่ละแอดเดรสมีขนาดข้อมูล 16 บิต ในการบรรยายพฤติกรรมของหน่วยความจำจะต้องใช้โปรซีเจอร์ `vec2int` เนื่องจากการอ้างอิงตำแหน่งของอะเรย์ต้องเป็นเลขจำนวนเต็มจึงต้องแปลงแอดเดรสจากชนิด `qsim_state_vector` เป็น `integer` เสียก่อนซึ่งมีอยู่ในแพ็คเกจ `basic_utilities` การทำงานจะเริ่มเมื่อมีสัญญาณ `rw` เกิดขึ้นแล้วตรวจสอบว่าเป็นการอ่านหรือเขียนข้อมูล ถ้าเป็น '0' จะอ่านข้อมูลและ '1' จะเขียนข้อมูลลงหน่วยความจำ ดังแสดงในรูปที่ 2.44 บรรยายพฤติกรรมของหน่วยความจำ

```

LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
USE work.basic_utilities.ALL;

ENTITY memory IS
  PORT (addr: IN qsim_state_vector(5 DOWNTO 0);
        data: INOUT qsim_state_resolved_x_vector(15 DOWNTO 0) BUS;
        rw : IN qsim_state);
END memory;
ARCHITECTURE behavioral OF memory IS
  SUBTYPE dbyte IS qsim_state_vector(15 DOWNTO 0);
  TYPE dbyte_memory IS ARRAY(0 TO 63) OF dbyte;
BEGIN
  PROCESS
    VARIABLE mem:dbyte_memory;
    VARIABLE address:INTEGER;
  BEGIN
    WAIT ON rw;
    vec2int(addr,address);
    IF (rw='0') THEN
      data<=qsim_state_resolved_x_vector(mem(address));
    ELSIF (rw='1') THEN
      mem(address):=dbyte(data);
    END IF;
  END PROCESS;
END behavioral;

```

รูปที่ 2.44 แสดงการบรรยายพฤติกรรมของหน่วยความจำภายใน EEPROM

2.3.3.4 องค์ประกอบหลักของ EEPROM

เมื่อมีองค์ประกอบที่ทำหน้าที่เก็บข้อมูล และแพ็คเกจที่เก็บโปรซีเจอร์ไว้แล้ว เขียนบรรยายพฤติกรรมตามตารางคำสั่งของ EEPROM การทำงานจะมีขึ้นทุกๆครั้งที่ป้อนสัญญาณคล็อก

รูปที่ 2.45 แสดงการบรรยายพฤติกรรมองค์ประกอบหลักของ EEPROM

```

LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
USE work.basic_utilities.ALL;
ENTITY eeprom IS
    PORT (SIGNAL sk:IN qsim_state;
          SIGNAL cs:IN qsim_state;
          SIGNAL di:IN qsim_state;
          SIGNAL do:OUT qsim_state);
END eeprom;
ARCHITECTURE behav OF eeprom IS
    COMPONENT ram PORT (addr:IN qsim_state_vector (5 DOWNT0 0);
                        data: INOUT qsim_state_resolved_x_vector (15 DOWNT0 0) BUS;
                        rw :IN qsim_state);
    END component;
    FOR U1: ram USE ENTITY work.memory(behavioral);
    SIGNAL addr: qsim_state_vector (5 DOWNT0 0);
    SIGNAL data: qsim_state_resolved_x_vector (15 DOWNT0 0) BUS;
    SIGNAL rw : qsim_state;
    BEGIN
        U1:ram PORT map (addr,data,rw);
        PROCESS
            VARIABLE OpCode:qsim_state_vector (1 DOWNT0 0);
            VARIABLE EWEN:qsim_state := 'X';
            VARIABLE WR_FFFF:qsim_state_vector (15 DOWNT0 0) := "1111111111111111";
            VARIABLE address:qsim_state_vector (5 DOWNT0 0);
            BEGIN
                WAIT ON sk UNTIL sk = '1' AND sk'event AND (sk'LAST_VALUE = '0');
                IF (cs = '1') THEN
                    IF (di = '1') THEN -- Start bit
                        FOR i IN OpCode'REVERSE_RANGE LOOP
                            IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
                                OpCode(i) := di;
                            ELSE exit;
                            END IF;
                        END LOOP;
                        IF (OpCode = "10") THEN -- Read mode
                            FOR i IN addr'REVERSE_RANGE LOOP
                                IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
                                    addr(i) <= di;
                                ELSE exit;
                                END IF;
                            END LOOP;
                            rw <= '0';
                            FOR i IN data'REVERSE_RANGE LOOP
                                IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
                                    do <= data(i);
                                ELSE exit;
                                END IF;
                            END LOOP;
                        ELSIF OpCode = "01" AND EWEN = '1' THEN -- Write mode
                            FOR i IN addr'REVERSE_RANGE LOOP
                                IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
                                    addr(i) <= di;
                                ELSE exit;
                                END IF;
                            END LOOP;
                        END IF;
                    END IF;
                END IF;
            END PROCESS;
        END IF;
    END IF;
END IF;

```

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 2.45 (ต่อ)

```

FOR i IN data'REVERSE_RANGE LOOP
  IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
    data(i) <= di;
  ELSE exit;
  END IF;
END LOOP;
rw <= '0';
IF (data = "1111111111111111") THEN
  rw <= '1';
END IF;
ELSIF OpCode = "11" AND EWEN = '1' THEN -- Erase
FOR i IN addr'REVERSE_RANGE LOOP
  IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
    addr(i) <= di;
  ELSE exit;
  END IF;
END LOOP;
data <= qsim_state_resolved_x_vector(WR_FFFF);
rw <= '1';
ELSIF (OpCode = "00") THEN
FOR i IN 5 DOWNT0 0 LOOP
  IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
    addr(i) <= di;
  ELSE exit;
  END IF;
END LOOP;
IF (addr(5 DOWNT0 4) = "11") THEN -- Enable
  EWEN := '1';
ELSIF (addr(5 DOWNT0 4) = "00") THEN -- Disable
  EWEN := '0';
ELSIF (addr(5 DOWNT0 4) = "10") THEN -- Erase all
  rw <= '1';
  FOR i IN 0 to 63 LOOP
    data <= qsim_state_resolved_x_vector(WR_FFFF);
    int2vec(i,address); addr<=address;
  END LOOP;
  rw <= '0';
ELSIF (addr(5 DOWNT0 4) = "01") THEN -- Write all
  FOR i IN 15 DOWNT0 0 LOOP
    IF (sk = '1' AND sk'EVENT AND (sk'LAST_VALUE = '0')) THEN
      data(i) <= di;
    ELSE exit;
    END IF;
  END LOOP;
  rw <= '1';
  FOR i IN 63 DOWNT0 0 LOOP
    int2vec(i,address);
    addr<=address;
  END LOOP;
  rw <= '0';
END IF;
END IF;
END IF;
END IF;
END PROCESS;
END behav;

```

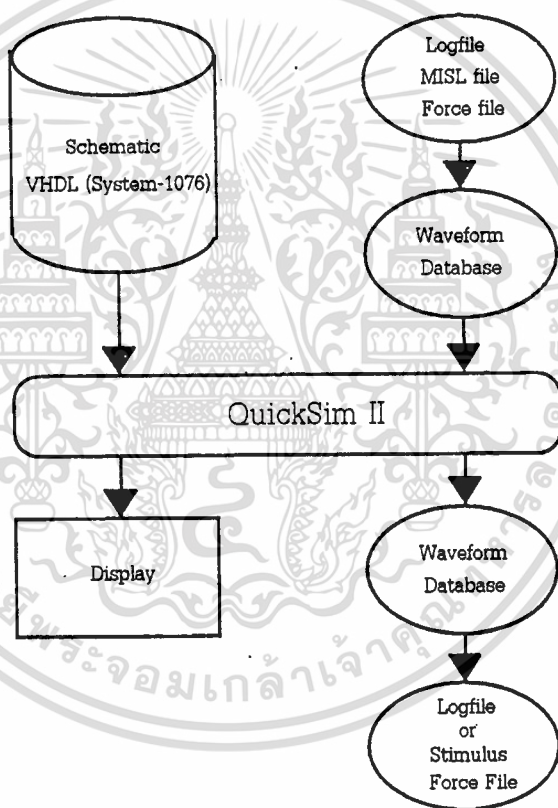
เอกสารนี้เป็นเอกสารสงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.4 การสร้างสัญญาณทดสอบการทำงานของวงจรด้วยภาษา MISL

ภาษา MISL (Mentor Interactive Stimulus Language) เป็นภาษาที่ใช้เขียนเป็นโปรแกรมบรรยายลักษณะของสัญญาณอินพุตที่ต้องการป้อนให้กับวงจรเพื่อจำลองการทำงาน ภาษา MISL เหมาะกับวงจรที่มีสัญญาณอินพุตจำนวนมากๆ วงจรที่ต้องใช้จำนวนสัญญาณต่อกันยาวๆ และวงจรที่ต้องใช้และวงจรที่ต้องการจำลองการทำงานด้วยรูปแบบจำนวนมากๆ

สำหรับในกรณีของการจำลองการทำงานของวงจรฮาร์ดลอคจะต้องป้อนสัญญาณอินพุตต่อกันค่อนข้างยาว ทำให้การสร้างสัญญาณด้วยการใช้คำสั่ง Force ที่ละสัญญาณ หรือจะเขียนเป็น Logfile หรือจะใช้ Waveform Editor ที่ใช้ในโปรแกรม QuickSim II ก็ตาม ไม่เหมาะสมกับวงจรฮาร์ดลอค จึงได้ใช้ภาษา MISL สร้างสัญญาณอินพุตเพื่อป้อนให้กับวงจรฮาร์ดลอคในการจำลองการทำงาน ดังแสดงขั้นตอนในรูปที่ 2.46



รูปที่ 2.46 บล็อกแสดงการจำลองการทำงานด้วย QuickSim II

ลักษณะของโปรแกรมที่เขียนด้วยภาษา MISL มีรูปแบบคำสั่งที่เข้าใจง่าย ดังตัวอย่างในรูปที่ 2.47 แสดงการนำภาษา MISL เขียนโปรแกรมสร้างสัญญาณอินพุต 5 อินพุต X1, X2, X3, X4, CLK ให้กับวงจรชื่อ EXAMPLE มีช่วงเวลาของการเปลี่ยนแปลงสัญญาณเป็น 100 ns. จากโปรแกรมที่เขียนจะสร้างสัญญาณได้ตามตารางท้ายโปรแกรม

```

CIRCUIT EXAMPLE;
CONST testpata='1010' S;
CONST testpath=NOT testpata;
TIMEDEF period=100NS;
INPUT X1,X2,X3,X4,CLK;
OUTPUT X5,X6;
VECTOR Vec1= X1,X2,X3,X4; /* X1 most significant bit */
VAR i,j;
j=4;
FOR i=1 TO j BY 2 /* I takes values of 1,3 */
{
  Vec1=testpata $ /* Assign statestring to Vec1 */
  DO i TIMES
  {
    CLK=HI $
    CLK=LO $
  }
  Vec1=testpath $ /* Assign inverse statestring to Vec1 */
}
END .

```

List window equivalent:

0	1	0	1	0	
100	1	0	1	0	1
200	1	0	1	0	0
300	0	1	0	1	0
400	1	0	1	0	0
500	1	0	1	0	1
600	1	0	1	0	0
700	1	0	1	0	1
800	1	0	1	0	0
900	1	0	1	0	1
1000	1	0	1	0	0
1100	0	1	0	1	0
	^	^	^	^	^
Times	X1	X2	X3	X4	CLK

รูปที่ 2.47 ตัวอย่างโปรแกรมสร้างสัญญาณ และตารางแสดงลักษณะของสัญญาณที่ได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.5 ผลการทดสอบ การจำลองการทำงาน

ในการจำลองการทำงานของวงจรฮาร์ดลอค ได้ใช้ภาษา MISL เขียนเป็นโปรแกรมสร้างสัญญาณอินพุต เพื่อทดสอบการทำงานของวงจรไวดลายแพทเทอน (Pattern) ตามฟังก์ชันการทำงานแต่ละคำสั่งของฮาร์ดลอค เช่นในการทดสอบการอ่านข้อมูลจากแอดเดรส 37 D มีขั้นตอนดังต่อไปนี้

1. ก่อนอื่นต้องเขียนโปรแกรมสร้างสัญญาณอินพุตดังรูปที่ 2.48 ซึ่งจะสังเกตเห็นว่า read_pat คือคำสั่งทั้งหมดที่ป้อนเข้าที่ขาสัญญาณ D นั้นเอง วิธีการสร้างคำสั่งได้อธิบายเป็นตัวอย่างไว้แล้ว
2. หลังจากนั้นเข้าสู่โปรแกรม QuickSim II ของวงจร HARDLOCK
3. เปิดหน้าต่างวงจรทั้งหมด (Sheet) และเปิดโมเดลของหน่วยความจำ EEPROM (VHDL)
4. เช็ตค่าเริ่มต้นให้กับรหัสชี้เฉพาะ DATA (32), DATA (33) และรหัสลับ DATA(37) เป็น 4E20H, 2710H และ 9B88H
5. เปิด Trace ว่างๆขึ้นมาก่อน
6. ป้อนคำสั่งเพื่อคอมไพล์โปรแกรม MISL เป็น dofile MISL ชื่อโปรแกรม' ชื่อของ Wavwform' ที่ Session window
7. ตรวจสอบผลลัพธ์ของเอาพุตหลักที่ขาสัญญาณ CS2, DI2, DO_EP, Q ซึ่งผลลัพธ์ที่ถูกต้องจะเป็นดังรูปที่ 2.35 แสดง Waveform จำลองการทำงานการอ่านรหัสลับในแอดเดรส 37 D

รูปที่ 2.48 โปรแกรมสร้างสัญญาณทดสอบการอ่านรหัสลับที่แอดเดรส 37 D

```

CIRCUIT HARDLOCK;
CONST
read_pat='11001101001100101011001101011000111011111110110001110111100000000000000000'
TIMEDEF PERIOD = 1000 NS;
INPUT  CLK, RST, RST_RO, D;
OUTPUT CS2, DI2, DO_EP, Q;
VECTOR Read_Vec=D;
VAR I;
FOR I=0 TO 1 BY 1
{ RST=LO $ RST_RO=LO $
  D=HI $
  CLK=LO $
  CLK=HI $
}
END.

```

รูปที่ 2.48 (ต่อ)

```

FOR I=0 TO 130 BY 1
{
    Read_Vec=read_pat $
    CLK=LO $
    CLK=HI $
}
END.

```

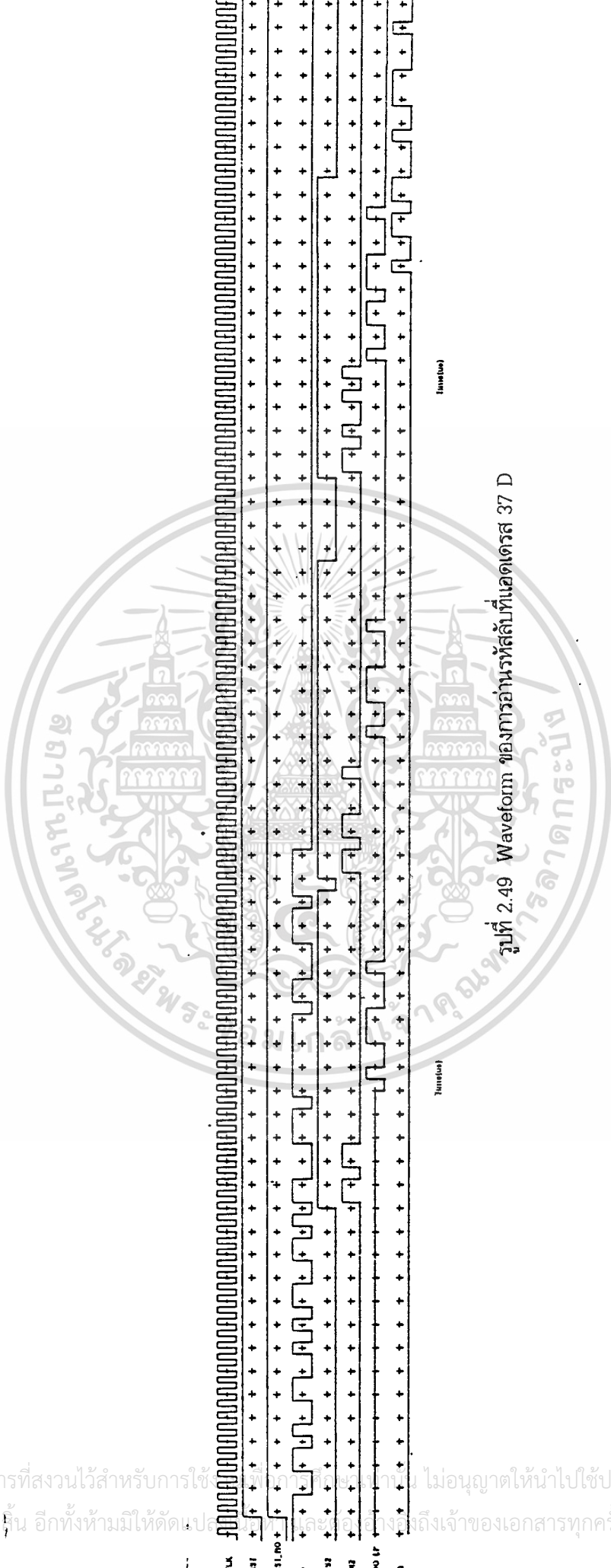
จาก Waveform ผลลัพธ์การอ่านรหัสลับจากแอดเดรส 37 D ที่ได้ดังรูปที่ 2.49 อธิบายได้ว่าเมื่อบริการฮาร์ดแวร์รับข้อมูลคำสั่งเข้ามาที่ขาสัญญาณ D ครบ 1+24 บิต สัญญาณ CS2 ที่ต่ออยู่กับขาสัญญาณ CS ของ EEPROM จะเป็น '0' → '1' แสดงว่า EEPROM พร้อมที่จะรับคำสั่ง หลังจากนั้นวงจรฮาร์ดแวร์จะสร้างสัญญาณ 110110000 ออกมาทางขาสัญญาณ DI2 ป้อนให้กับ EEPROM เพื่อเป็นคำสั่ง (ดูตาราง 3.2 ชุดคำสั่งของ EEPROM ประกอบ) อ่านข้อมูลที่แอดเดรส 110000 เข้าที่ขาสัญญาณ DI จากนั้น EEPROM จะส่งข้อมูลที่อยู่ในแอดเดรส 110000 ออกมาที่ขาสัญญาณ DO เข้าสู่วงจรฮาร์ดแวร์ที่ขาสัญญาณ DO เป็น 4E20H

CS เป็น '0' → '1' อีกครั้ง หลังจากนั้นวงจรฮาร์ดแวร์จะสร้าง สัญญาณ 110110001 ออกมาทางขาสัญญาณ DI2 ป้อนให้กับ EEPROM เพื่อเป็นคำสั่งอ่านข้อมูลที่แอดเดรส 110001 เข้าที่ขาสัญญาณ DI จากนั้น EEPROM จะส่งข้อมูลที่อยู่ในแอดเดรส 110001 ออกมาที่ขาสัญญาณ DO เข้าสู่วงจรฮาร์ดแวร์ที่ขาสัญญาณ DO เป็น 2710H

ในขณะที่วงจรฮาร์ดแวร์อ่านข้อมูลจาก EEPROM ที่อยู่ในแอดเดรส 110000 และ 110001 อยู่นั้น วงจรฮาร์ดแวร์จะทำการเปรียบเทียบ กับรหัสผ่าน (Password) ที่รับเข้ามา (ค่า 1's คอมพลิเมนต์ของข้อมูลที่อ่านจาก EEPROM : รหัสชี้เฉพาะ) ถ้าตรงกันวงจรฮาร์ดแวร์ก็จะทำงานต่อดังนี้คือ

CS เป็น '0' → '1' ครั้งที่สาม วงจรฮาร์ดแวร์จะสร้างสัญญาณ DI2 เป็น เพื่อเป็นคำสั่งอ่านข้อมูลที่แอดเดรส 100101 = 37D ของ EEPROM จากนั้นข้อมูลที่อ่านได้จะออกมาที่ขาสัญญาณ DO เป็น 9B88H เข้าสู่วงจรฮาร์ดแวร์ที่ขาสัญญาณ DO_EP นำไปถอดรหัส ต่อจากนั้นอีก 8 คล็อกข้อมูลก็ถอดรหัสได้ (รหัสลับ) จะออกมาที่ขาสัญญาณ Q เป็น 1388H

ในกรณีที่เป็นการเขียนข้อมูลให้แอดเดรส 37 D เป็น A1B6H การอ่านข้อมูลจากแอดเดรส 110000 และ 110001 ยังคงเหมือนกับการอ่านรหัสลับอยู่ จะต่างกันเมื่อสัญญาณ CS เป็น '0' → '1' ครั้งที่สาม คือแทนที่วงจรฮาร์ดแวร์จะสร้างคำสั่งเพื่ออ่านข้อมูลจาก EEPROM กลับสร้างสัญญาณคำสั่งเพื่อเขียนข้อมูลเป็น 101100101 ต่อด้วยข้อมูล (Data) ที่วงจรฮาร์ดแวร์รับเป็นคำสั่งเข้ามา ซึ่งเป็นค่าเข้ารหัสของ A1B6H เท่ากับ 17B6H ออกมาที่ขาสัญญาณ DI2 ท้ายสุด EEPROM จะถูกเขียนข้อมูลเข้าแอดเดรส 37 D เป็น A1B6H



รูปที่ 2.49 Waveform ของการอ่านรหัสที่แอดเดรส 37 D

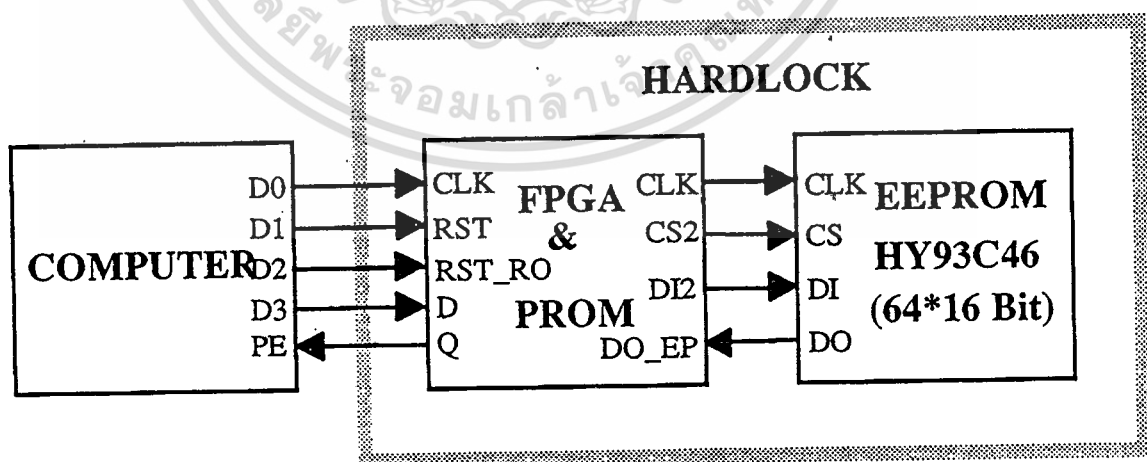
บทที่ 3

การพัฒนาอุปกรณ์ฮาร์ดล็อกโดยใช้เทคโนโลยีของ FPGA

เพื่อให้วงจรฮาร์ดล็อกที่ได้ผ่านการจำลองการทำงานมาแล้วในบทที่ 2 นั้น สามารถทำงานได้จริงตามที่ได้ทดสอบจำลองการทำงาน จึงจำเป็นอย่างยิ่งที่จะต้องนำวงจรฮาร์ดล็อกที่ได้นี้ไปพัฒนาสร้างเป็นอุปกรณ์ที่จะทดสอบการทำงานของวงจรอีกครั้งหนึ่ง (วงจรต้นแบบ) ซึ่งเป็นไปได้ที่อาจจะต้องกลับไปแก้ไขวงจรใหม่ หลังจากทดสอบต้นแบบแล้วเนื่องจากทำงานไม่ตรงกับที่ได้จำลองการทำงานมา แต่อย่างไรก็ตามเมื่อได้วงจรต้นแบบที่ทำงานได้ถูกต้องแล้วถ้าต้องการจะผลิตเป็นเชิงพาณิชย์ จะต้องมีวิธีการผลิตที่ไม่ยุ่งยาก ซึ่งหมายความว่า ผู้ออกแบบไม่ต้องพัฒนาต้นแบบต่อไปอีก ควรจะเป็นหน้าที่ของผู้ผลิตอย่างเดียว และมั่นใจว่าจะทำงานได้ตามวงจรที่สร้างอยู่บนอุปกรณ์ต้นแบบนั้น อีกทั้งระยะเวลาในการผลิตเป็นเชิงพาณิชย์จะต้องไม่นานเกินไป

3.1 องค์ประกอบของต้นแบบฮาร์ดล็อก

ต้นแบบฮาร์ดล็อกประกอบด้วย วงจรรวมเฉพาะกิจที่เป็น FPGA ตระกูล 3000 (XC3042A) , PROM ชนิดอนุกรมความจุขนาด 36,288 บิตเพื่อใช้โหลดโปรแกรมคอนฟิก (Configuration) ให้กับ FPGA และหน่วยความจำขนาดความจุ 64*16 บิต (EEPROM HY93C46) ซึ่งเป็นหน่วยความจำแบบอนุกรมที่สามารถโปรแกรมได้ ใช้เก็บรหัสเฉพาะ (Identify Code) และรหัสลับ (Code) ของฮาร์ดล็อก การเชื่อมต่อสัญญาณระหว่าง FPGA & PROM กับ EEPROM และเชื่อมต่อกับคอมพิวเตอร์ทางพอร์ตแบบขนานดังรูปที่ 3.1 ส่วนลักษณะการเชื่อมต่อระหว่าง FPGA กับ PROM จะกล่าวในหัวข้อต่อไป



รูปที่ 3.1 บล็อกแสดงองค์ประกอบต้นแบบฮาร์ดล็อก และการเชื่อมต่อสัญญาณ

3.1.1 64*16 BIT SERIAL ELECTRICALLY ERASABLE PPROM (HY93C46)

เป็นหน่วยความจำกึ่งการที่สามารถเขียน,ลบ ,อ่านด้วยแรงดันไฟฟ้า (Electrical Erasable Programmable Read Only Memory) ข้อมูลที่อ่านและเขียนเป็นแบบอนุกรม จัดโครงสร้างเป็นรีจิสเตอร์ขนาด 16 บิต จำนวน 64 ชุด ข้อมูลจะคงอยู่จนกว่าจะมีการลบหรือเขียนใหม่ เก็บข้อมูลได้นานถึง 10 ปี ใช้เทคโนโลยี CMOS ซึ่งประหยัดไฟลงถึง 80 % อยู่ในตัวถังแบบ (DIP) ขนาด 8 ขาเท่านั้น

3.1.1.1 คุณสมบัติเฉพาะ

ใช้เทคโนโลยี CMOS

ราคาต่ำ

อินเตอร์เฟสกับไอซี TTL ได้โดยตรง

เขียนและ ลบข้อมูลได้ด้วยแรงดัน 5 โวลต์

โครงสร้าง 64*16 บิต แบบอนุกรม

ใช้งานง่าย มีความเชื่อถือสูง

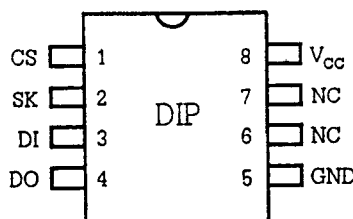
สิ้นเปลืองกำลังไฟขณะสแตนด์บายต่ำ

มีสัญญาณบอกสภาวะการทำงาน

เก็บข้อมูลได้นานถึง 10 ปี

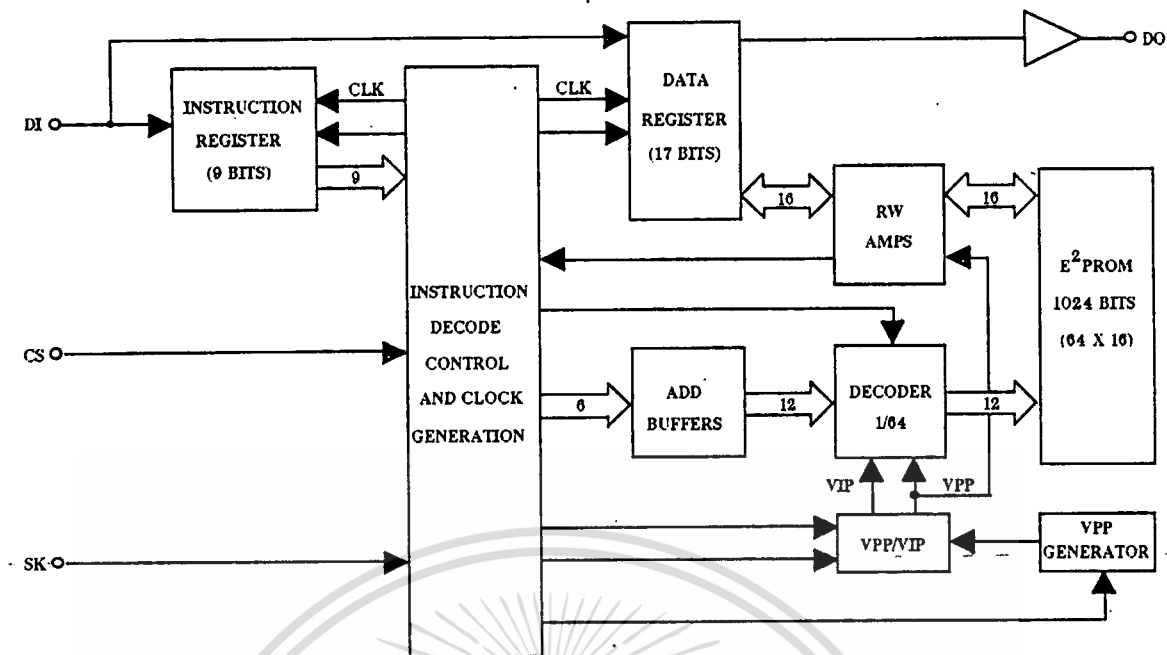
3.1.1.2 รายละเอียดขา

รูปร่างภายนอก การจัดขา และตำแหน่งสัญญาณดังแสดงในรูปที่ 3.2 จะเห็นว่าขาสัญญาณที่ใช้งานมีเพียง 4 ขาเท่านั้นยกเว้นขาไฟเลี้ยง และกราวด์ทำให้ใช้งานง่าย โดยแต่ละสัญญาณมีชื่อเรียกและหน้าที่ดังตารางที่ 3.1 ส่วนโครงสร้างวงจรภายในเป็นดังรูปที่ 3.3



รูปที่ 3.2 โครงสร้างภายนอกและการจัดขาของ EEPROM HY93C46

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.3 โครงสร้างภายในของ EEPROM HY93C46

ตารางที่ 3.1 แสดงขาสัญญาณของ EEPROM

ขา	ชื่อเต็ม	การใช้งาน
SK	Serial Clock	ขาต่อสัญญาณคล็อกจากภายนอกสำหรับเลื่อนข้อมูลเข้าออกจากหน่วยความจำ
CS	Chip Select	เมื่อเป็น "1" จะเป็นการอินเเบิลลงจรรยาใน
DI	Serial Data In	เป็นขารับข้อมูล และคำสั่งภายใต้การควบคุมของขา SK และ CS
DO	Serial Data Out	เป็นขาที่ส่งเอาพุตจากหน่วยความจำ ภายใต้การควบคุมของขา SK และ CS ปกติจะเป็นไฮอิมพีแดนซ์จะแอกทีฟเฉพาะตอนอ่าน (Read) และเช็คสภาวะ
V _{CC}		ไฟเลี้ยง + 5 โวลต์
GND	Ground	กราวด์
NC	No Connection	ไม่มีการต่อใช้งาน

3.1.1.3 รายละเอียดการใช้งาน

EEPROM HY93C46 ออกแบบมาเป็นอุปกรณ์หน่วยย่อยสำหรับเก็บข้อมูลที่ค่อนข้างถาวร แต่สามารถเปลี่ยนแปลงแก้ไขได้ ภายในรีจิสเตอร์ขนาด 16 บิตจำนวน 64 ชุด ควบคุมการทำงานด้วยชุดคำสั่งขนาด 9 บิต ทั้งหมด 7 คำสั่ง ในการอ่าน เขียน และลบข้อมูล ทำงานด้วยแรงดัน 5 โวลต์ ระดับเดียวเท่านั้น เพราะว่าแรงดันสูงสำหรับการใช้ในการโปรแกรมข้อมูลได้จากวงจรสร้างแรงดันภายในตัวไอซีเองดูรูปที่ 3.3 ประกอบ ส่วนขาข้อมูลเอาพุต (DO) นอกจากใช้เป็นขาส่งข้อมูลที่อยู่ในหน่วยความจำออกมาแล้วยังใช้แสดงสภาวะในช่วงของวง

เอกสารนี้เป็นการนำเอกสารเพื่อการศึกษานี้ ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทำงานทุกอย่างของ HY93C46 จะเริ่มต้นด้วยการโหลดคำสั่งเข้าไปที่ไอซีก่อน ซึ่งทุกคำสั่งจะเริ่มต้นด้วยบิตที่เป็น "1" เสมอ ตามด้วย 2 บิตที่เป็นรหัสคำสั่ง และ 6 บิตที่เป็นแอดเดรส ซึ่งรายละเอียดของคำสั่งต่างๆ ดูได้จากตารางที่ 3.2

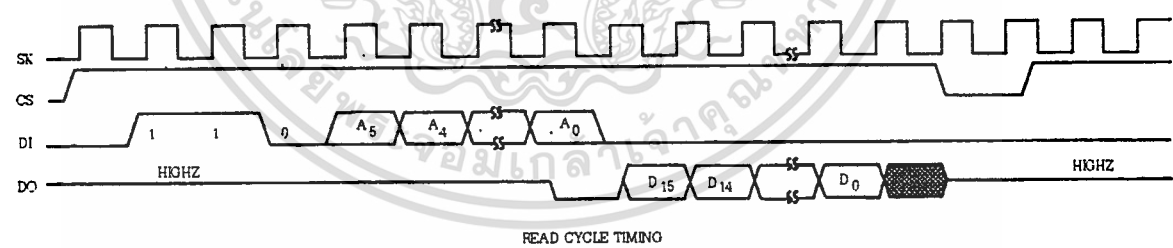
คำสั่งการอ่าน (READ)
เมื่อได้รับคำสั่งอ่านข้อมูลในหน่วยความจำ ณ ตำแหน่งแอดเดรสที่กำหนดในคำสั่งจะถูกถ่ายเทมาไว้ที่รีจิสเตอร์ชนิดอนุกรมขนาด 16 บิต ข้อมูลนี้จะถูกส่งโดยการชิฟต์ออกมาได้โดยการป้อนสัญญาณเข้าที่ขา SK ซึ่งจะมีบิตที่เป็น "0" เป็นตัวมีนำหน้าออกมาก่อนข้อมูลที่ต้องการอ่านจริง โดยข้อมูลจะเปลี่ยนแปลงตามขอบขาขึ้นของสัญญาณนาฬิกา (SK)
คำสั่งการเอนเบิล และดิสเอนเบิล (ENABLE/DISABLE)
เมื่อเริ่มจ่ายกระแสไฟให้ ตัวไอซีจะถูกเซตให้อยู่ในสถานะดิสเอนเบิลจากการโปรแกรม (ไม่สามารถโปรแกรมได้) ซึ่งในการที่จะโปรแกรมข้อมูลใหม่จะต้องป้อนคำสั่ง EWEN เพื่อให้สามารถทำการโปรแกรมได้ หลังจากที่มีการโปรแกรมโดยเขียน หรือ ลบข้อมูลในหน่วยความจำแล้วควรป้อนคำสั่ง EWDS เพื่อให้ไม่สามารถโปรแกรมได้อีกครั้ง ป้องกันการโปรแกรมโดยไม่ตั้งใจ สำหรับคำสั่งการอ่านนั้นจะไม่เกี่ยวข้องกับคำสั่งต่างๆ เหล่านี้
คำสั่งการลบข้อมูล (ERASE)
การที่จะเขียนข้อมูลใหม่ลงไปได้ ข้อมูลที่อยู่ในแอดเดรสนั้นทุกบิตจะต้องถูกลบก่อนซึ่งจะทำให้ทุกบิตเป็น "1" เมื่อคำสั่งลบถูกโหลดเข้าไปแล้วขา CS จะต้องเป็น "0" ช่วงขอบขาของสัญญาณที่ขานี้เป็นจุดเริ่มของวงรอบการลบข้อมูล และเมื่อขา CS กลับเป็น "1" หลังช่วงเวลา t_{CS} ขา DO จะเป็น "0" ตลอดช่วงเวลาของโปรแกรมโหลด และจะเป็น "1" เมื่อพร้อมที่จะรับคำสั่งต่อไป คำสั่ง ERASE ใช้ลบข้อมูลของรีจิสเตอร์ตำแหน่งที่ระบุ ส่วนคำสั่ง ERAL ใช้ลบข้อมูลในรีจิสเตอร์ทุกตัว
คำสั่งการเขียนข้อมูล (WRITE)
เมื่อมีการโหลดคำสั่งเขียนเข้าไปแล้ว จะต้องตามด้วยข้อมูลจำนวน 16 บิต หลังจากหมดบิตสุดท้าย ขา CS จะต้องถูกทำให้เป็น "0" ก่อนขอบขาขึ้นของสัญญาณคล็อกลูกต่อไป ช่วงขอบขาของสัญญาณ CS จะเป็นจุดเริ่มต้นวงรอบการโปรแกรม และเมื่อขา CS กลับเป็น "1" หลังช่วงเวลา t_{CS} ขา DO จะแสดงสถานะความพร้อมเช่นเดียวกัน อย่าลืมว่าก่อนเขียนข้อมูลลงรีจิสเตอร์ใด จะต้องทำการลบข้อมูลที่อยู่ในรีจิสเตอร์นั้นเสียก่อนทุกครั้ง คำสั่งเขียนข้อมูลทุกรีจิสเตอร์ (WRAL) ปกติไม่ได้ใช้งาน แต่จะใช้ทดสอบการเขียนข้อมูลในขั้นตอนการผลิตจากโรงงานเท่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.2 ชุดคำสั่งของ EEPROM (HY93C46)

คำสั่ง	บิต สตาร์ท	รหัส	แอดเดรส	ข้อมูล	การทำงาน
READ	1	10	A ₅ A ₄ A ₃ A ₂ A ₁ A ₀		อ่านรีจิสเตอร์ A ₅ A ₄ A ₃ A ₂ A ₁ A ₀
WRITE	1	01	A ₅ A ₄ A ₃ A ₂ A ₁ A ₀	D ₁₅ - D ₀	เขียนรีจิสเตอร์ A ₅ A ₄ A ₃ A ₂ A ₁ A ₀
ERASE	1	11	A ₅ A ₄ A ₃ A ₂ A ₁ A ₀		ลบรีจิสเตอร์ A ₅ A ₄ A ₃ A ₂ A ₁ A ₀
EWEN	1	00	11 XXXX		อินเพิล การเขียน/ลบข้อมูล
EWDS	1	00	00 XXXX		ดีสเอบิลการเขียน/ลบข้อมูล
ERAL	1	00	10 XXXX		ลบทุกรีจิสเตอร์
WRAL	1	00	01 XXXX	D ₁₅ - D ₀	เขียนทุกรีจิสเตอร์

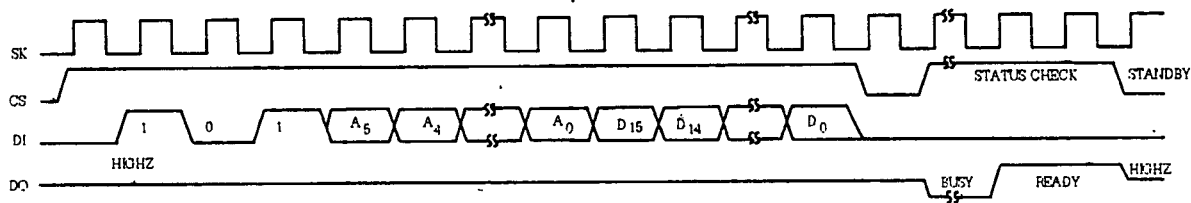
เมื่อแสดงให้เห็นแผนภูมิเวลาของคำสั่งทั้ง 7 คำสั่ง ทำให้เข้าใจลำดับการป้อนคำสั่งเข้าที่ขาสัญญาณทั้งสามขาสัญญาณ และลำดับข้อมูลของสัญญาณที่ออกมาหน่วยความจำทางขาสัญญาณ DO ได้มากยิ่งขึ้น จากรูปที่ 3.4 เป็นแผนภูมิเวลาของคำสั่งอ่านข้อมูล เริ่มจากการป้อนสัญญาณเข้าที่ขา CS “0”→“1” หลังจากนั้นจึงป้อนสัญญาณนาฬิกาเข้าที่ขา SK และป้อนคำสั่งเข้าที่ขา DI ก่อนสัญญาณนาฬิกาขาขึ้น (ทำงานในสัญญาณนาฬิกาขาขึ้น) แล้วข้อมูลที่ต้องการอ่านจะเริ่มออกมาที่ขาสัญญาณ DO โดยมีบิต “0” นำหน้า ในสัญญาณนาฬิกาขาขึ้นหลังจากที่ป้อนแอดเดรสครบทุกบิต ในการป้อนแอดเดรสจะป้อนเป็นลำดับคือ A₅ - A₀ ส่วนข้อมูลที่อ่านออกมาเป็นลำดับคือ D₁₅ - D₀



รูปที่ 3.4 แผนภูมิเวลาคำสั่งของการอ่านข้อมูล

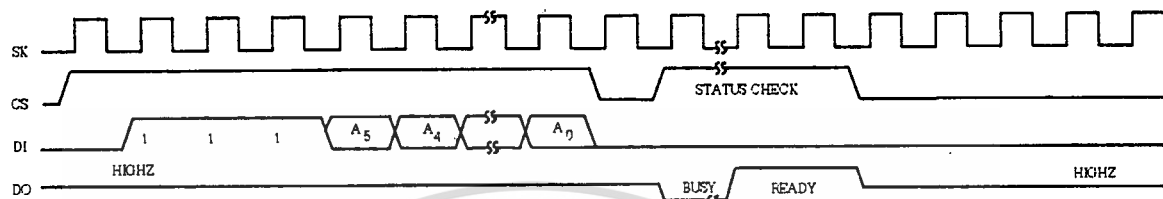
จากรูปที่ 3.5 เป็นแผนภูมิเวลาของคำสั่งเขียนข้อมูล เริ่มจากการป้อนสัญญาณเข้าที่ขา CS “0” → “1” หลังจากนั้นจึงป้อนสัญญาณนาฬิกาเข้าที่ขา SK และป้อนคำสั่งเข้าที่ขา DI ก่อนสัญญาณนาฬิกาขาขึ้น (ทำงานในสัญญาณนาฬิกาขาขึ้น) จะเห็นว่าข้อมูลที่ต้องการจะเขียนลงในหน่วยความจำป้อนเข้าที่ขา DI ต่อจากการป้อนแอดเดรสโดยป้อนเป็นลำดับ หลังจากนั้นสามารถตรวจสอบความพร้อมของไอซีได้โดยป้อนสัญญาณเข้าที่ขา CS “0” → “1” แล้วตรวจสอบที่ขาสัญญาณ DO ถ้าเป็น “1” แสดงว่าพร้อมที่จะทำงานในคำสั่งต่อไป แต่ถ้าเป็น “0” แสดงว่าไม่พร้อมที่จะทำงานในคำสั่งต่อไปได้ ส่วนในคำสั่งที่เหลือแสดงให้เห็นเป็นแผนภูมิเวลา

เอกสารนี้ดังรูปที่ 3.6-3.9 ไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



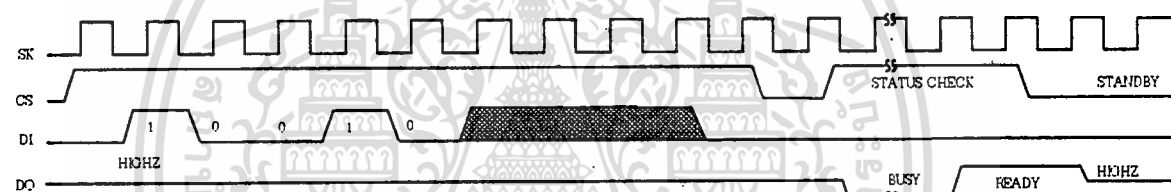
WRITE CYCLE TIMING

รูปที่ 3.5 แผนภูมิเวลาคำสั่งของการเขียนข้อมูล



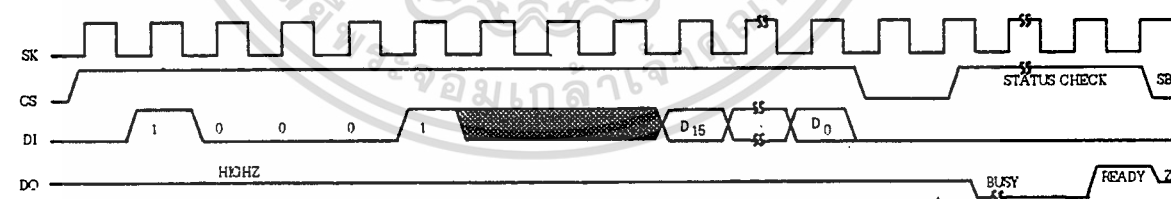
ERASE (REGISTER) CYCLE TIMING

รูปที่ 3.6 แผนภูมิเวลาคำสั่งของการลบข้อมูล



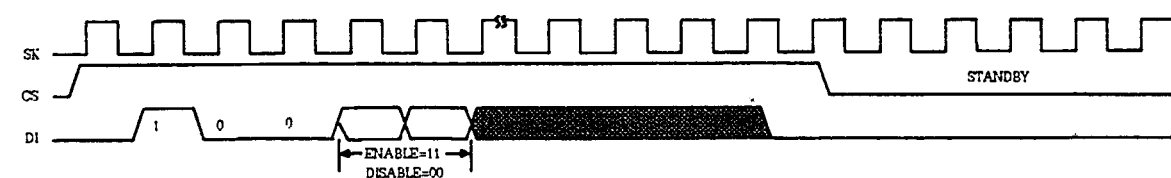
ERASE ALL CYCLE TIMING

รูปที่ 3.7 แผนภูมิเวลาคำสั่งของการลบข้อมูลทั้งหมด



WRITE ALL CYCLE TIMING

รูปที่ 3.8 แผนภูมิเวลาคำสั่งของการเขียนข้อมูลทั้งหมด



ERASE/WRITE ENABLE AND DISABLE CYCLE TIMING

รูปที่ 3.9 แผนภูมิเวลาคำสั่ง อีเนเบิล/ดิสเอเบิล การเขียน และลบข้อมูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.1.2 SERIAL CONFIGURATION PROMs (XC17xxx FAMILY)

เป็นหน่วยความจำถาวรที่สามารถเขียนข้อมูล ด้วยแรงดันไฟฟ้าได้เพียงครั้งเดียว (One-Time Programmable OTP) ข้อมูลที่อ่านเป็นแบบอนุกรมใช้สำหรับเก็บโปรแกรมคอนฟิกที่เก็บเป็น Bitstreams เพื่อใช้โหลดให้กับวงจรรวม FPGA ของ Xilinx โครงสร้างภายในมีวงจรนับแอดเดรส (Address Counter) และเมทริกซ์เซลล์ของ EPROM อยู่ในตัวถังแบบ (DIP) ขนาด 8 ขาเท่านั้น

3.1.2.1 คุณสมบัติเฉพาะ

ใช้เทคโนโลยี CMOS

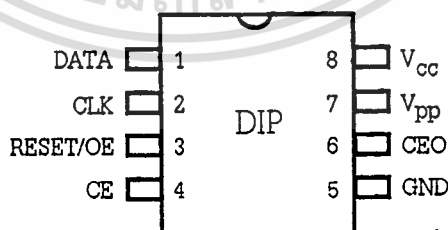
มีวงจรนับแอดเดรสภายใน ซึ่งเพิ่มขึ้นที่สัญญาณนาฬิกาขาขึ้น

อินเทอร์เฟซกับ FPGA ได้ง่าย ซึ่งใช้เพียง 1 ขา I/O เท่านั้น

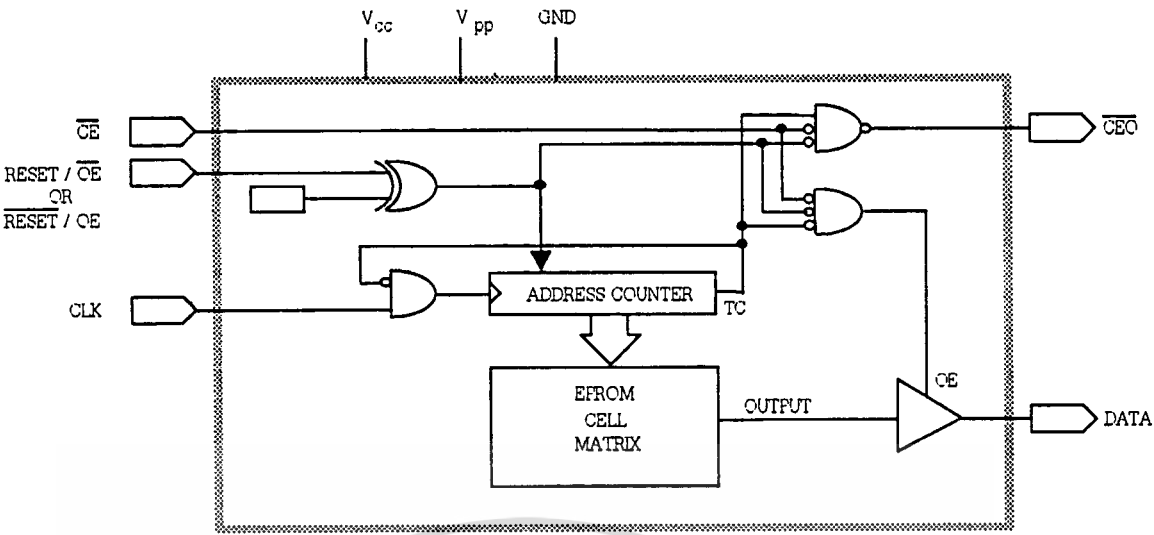
ขา RESET และ OE สามารถโปรแกรมให้แอดดีฟ "0" หรือ "1" ก็ได้ เพื่อให้ตรงกับ FPGA ที่ต่อด้วย
ใช้งานง่าย มีความเชื่อถือสูง

3.1.2.2 รายละเอียดขา

รูปร่างภายนอก , การจัดขา และตำแหน่งสัญญาณดังแสดงในรูปที่ 3.10 โดยแต่ละสัญญาณมีชื่อเรียกและหน้าที่ดังตารางที่ 3.3 เพื่อให้การเลือกขนาดของ PROM (Serial Configuration PROMs : SCPs) เป็นไปอย่างเหมาะสมจึงต้องพิจารณาให้สอดคล้องกับขนาดของโปรแกรมคอนฟิกที่ FPGA ต้องการด้วย ดังแสดงในตารางที่ 3.4 เช่น FPGA XC3042 มีขนาดของข้อมูลที่ใช้ในการคอนฟิก 30,784 บิต เพราะฉะนั้นขนาด SCP ที่เหมาะสมคือ XC1736 ซึ่งมีขนาด 30,824 บิต ส่วนโครงสร้างภายในของ SCP เป็นดังรูปที่ 3.11



รูปที่ 3.10 โครงสร้างภายนอกและการจัดขา



รูปที่ 3.11 โครงสร้างภายในอย่างง่ายของ PROM XC17xxx

ตารางที่ 3.3 แสดงหน้าที่ขาสัญญานต่างๆของ EEPROM

ชื่อขา	การใช้งาน
DATA	สำหรับเลื่อนข้อมูลเข้าออกจากหน่วยความจำ
CLK	ขณะคล็อกขาขึ้นจะไปเพิ่มตัวนับแอดเดรส
RESET/OE(OE/RESET)	ถ้ารีเซ็ตแอดที่ตัวนับแอดเดรสจะถูกรีเซ็ต และควบคุมข้อมูลออกที่ขา DATA
CE	เมื่อเป็น "1" จะดิสเอบิลตัวนับแอดเดรส DATA เป็น "z" และบังคับให้อิซิกิน กระแสไฟน้อยลง อยู่ในโหมด Standby
CEO (Chip Enable Output)	ต่อกับขาอินพุต CE ของ SCPs อีกตัวหนึ่งเพื่อบอกให้อีกตัวทำงานเมื่อตัวเอง สิ้นสุดการนับ (หมดข้อมูล, TC แอ็คทีฟ) ในขณะที่ตัวเองหยุดทำงาน
V_{cc}	ไฟเลี้ยง + 5 โวลต์
V_{pp}	ไฟใช้เขียนข้อมูล (โปรแกรม) ส่วนตอนอ่านต้องต่อกับขา V_{cc} ไม่ปล่อยลอย
GND	กราวด์

ตารางที่ 3.4 แสดงขนาดที่เหมาะสมของ SCPs กับ FPGA เบอร์ต่างๆ

อุปกรณ์	จำนวนบิต	SCP
XC2064	12,038	XC1718
XC2018	17,878	XC1718
XC3020/3120	14,819	XC1718
XC3030/3130	22,216	XC1736

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.4 (ต่อ)

XC3042/3142	30,824	XC1736
XC3064/3164	46,104	XC1765
XC3090/3190	64,200	XC1765
XC3195	94,984	XC17128
XC4002A	31,668	XC1736
XC4003A	45,676	XC1765
XC4003H	53,967	XC1765
XC4004A	62,244	XC1765
XC4005A	81,372	XC17128
XC4005/4005H	95,000	XC17128
XC4006	119,832	XC17128+XC1718
XC4008	147,544	XC17128+XC1765
XC4010	178,136	XC17128+XC17128
XC4013	247,960	XC17128+XC17128
XC4025	422,168	XC17128+XC1736

3.1.2.3 รายละเอียดการใช้งาน

เมื่อ FPGA ใช้งานในโหมด Master Serial จะสร้างคอนฟิกคล็อก (Configuration Clock) ออกมาทางขาสัญญาณ CCLK เพื่ออ่านข้อมูลจาก XC17xxx (Serial Configuration PROMs : SCPs) หลังจากที่ยก XC17xxx รับสัญญาณนาฬิกา ข้อมูลจะส่งออกที่ขาสัญญาณ DATA ซึ่งต่ออยู่กับขาสัญญาณ DIN ของ FPGA FPGA กำหนดสัญญาณนาฬิกาหลายลูกเพื่อให้โหลดคอนฟิกเสร็จสมบูรณ์ แต่ถ้า FPGA อยู่ในโหมด Slave ทั้ง SCPs และ FPGA จะรับสัญญาณนาฬิกา (CCLK) จากภายนอก

SCPs สามารถต่ออนุกรมกันได้ เพื่อให้ขนาดของข้อมูลมีมากขึ้น โดยต่อขาเข้าพุต OCE ของตัวหนึ่งไปกับขาอินพุต CE ของอีกตัวหนึ่ง ต่อขาสัญญาณเข้าพุต DATA ทั้งสองเข้าด้วยกัน

ในการโปรแกรมข้อมูลลง SCPs โปรแกรม XACT [9] (รายละเอียดจะกล่าวในหัวข้อต่อไป) จะคอมไพล์ไฟล์สเกูล .LCA ของวงจรที่ออกแบบเป็นรูปแบบมาตรฐานแบบ Hex แล้วส่งให้เครื่องโปรแกรม SCPs โดยเฉพาะ เพื่อทำการอัดโปรแกรมลงไป

3.1.3 FPGA (XC3042)

เป็น FPGA ในตระกูล XC3000 [10] ที่ใช้สร้างต้นแบบฮาร์ดลลอค ดูในตารางที่ 3.5 ประกอบ FPGA จัดเป็นวงจรรวมเฉพาะกิจชนิดหนึ่งที่สามารถโปรแกรมเป็นวงจรดิจิทัลใดๆก็ได้เช่นเดียวกับ EPLD ต่างกันที่ EPLD โปรแกรมลงบน EPROM ภายใน และสามารถโปรแกรมใหม่ได้หลังจากนำไปลบด้วยแสง UV แต่

FPGA โปรแกรมวงจรลงบนสแตติกแรมภายในด้วยข้อมูลที่อยู่ภายนอก และสามารถโปรแกรมใหม่ได้โดยการรีเซ็ตด้วยสัญญาณไฟฟ้า นอกจากนั้น FPGA ยังประหยัดไฟ และมีความสามารถวงจรที่มีขนาดใหญ่ๆ (จำนวนเกตมากๆ) ได้ดีอีกด้วย

3.1.3.1 คุณสมบัติเฉพาะ

ใช้เทคโนโลยี CMOS
ที่มีโครงสร้างเหมือนเกตอะเรีย
โปรแกรมการทำงานของวงจรด้วยข้อมูลจากหน่วยความจำภายนอก มี 4 ลักษณะ
สามารถโปรแกรมการทำงานของวงจรขนาดความจุสูงสุดถึง 13,000 เกต
ความเร็วในการทำงานสูง 25-40 Mhz สำหรับตระกูล XC3000 , 50-80 Mhz สำหรับตระกูล XC3100
ซึ่งสามารถมีความเร็วสูงกว่า 100 Mhz ได้ขึ้นอยู่กับการนำไปประยุกต์ใช้
สิ้นเปลืองกำลังไฟต่ำ 0.25-2.0 W ขึ้นกับการใช้งาน
อินเทอร์เฟซกับไอซี TTL ได้โดยตรง
ใช้งานง่าย มีความเชื่อถือสูง

3.1.3.2 โครงสร้างภายใน

วงจรรวมชนิดนี้ผลิตโดยบริษัท Xilinx ซึ่งเป็นบริษัทที่ร่วมกันทำการค้นคว้ากับบริษัท MMI สร้างเป็นอะเรียที่ประกอบด้วยเกตจำนวน 600-25,000 เกต ดังแสดงในตารางที่ 3.5 การที่ต้องบอกขนาดของวงจรรวมเป็นจำนวนเกตเพราะจะได้รู้ว่าขนาดของวงจรที่ออกแบบไว้สามารถโปรแกรมลงบนวงจรรวม FPGA ได้หรือไม่ เช่นวงจรฮาร์ดลอคมีจำนวนเกตประมาณ 1,600 เกตเพราะฉะนั้นไอซี FPGA ตระกูล XC3000 ที่สามารถโปรแกรมวงจรฮาร์ดลอคได้ต้องใหญ่กว่า XC3020 ส่วนการประมาณจำนวนเกตเมื่อเทียบกับเกตพื้นฐานเป็นดังตารางที่ 3.6

ตารางที่ 3.5 แสดงคุณสมบัติของ FPGA ตระกูลต่างๆ

FPGAs	Appr. Gate Count	Max I/Os	Flip-Flops	RAM bits	Available CLBs
XC2000 Family					
XC2064	1,000	58	122	0	64
XC2018	1,500	74	174	0	100
XC3000/3100 Family					
XC3020/3120	1,800	64	256	0	64
XC3030/3130	2,700	80	360	0	100
XC3042/3142	3,700	96	480	0	144

เอกสารนี้เป็นเอกสารตัวอย่าง ไม่ควรใช้เพื่อการเรียนการสอน

ไม่ว่ากรณีใดก็ตาม ผู้อ่านควรตรวจสอบเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.5 (ต่อ)

XC3064/3164	5,500	120	688	0	224
XC3090/3190	7,500	144	928	0	320
XC3195	9,000	176	1,320	0	484
XC4000 Family					
XC4002A	2,000	64	256	2,048	64
XC4003/4003A	3,000	80	360	3,200	100
XC4003H	3,000	160	200	3,200	100
XC4004A	4,000	96	480	4,608	144
XC4005/4005A	5,000	112	616	6,272	196
XC4005H	5,000	192	392	6,272	196
XC4006	6,000	128	768	8,192	256
XC4008	8,000	144	936	10,368	324
XC4010	10,000	160	1,120	12,800	400
XC4010D	10,000	160	1,120	0	400
XC4013	13,000	192	1,536	18,432	576
XC4025	25,000	256	2,560	32,768	1,024

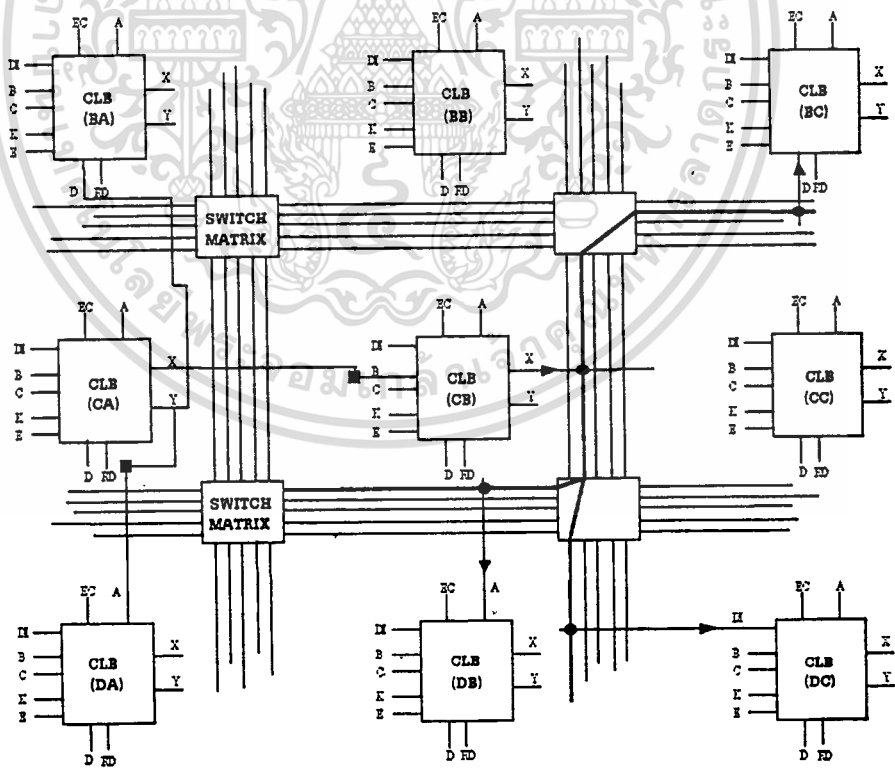
ตารางที่ 3.6 ตารางประมาณการนับเกตของเกตพื้นฐาน
NAIINR2 หมายถึงเกต NAND2 หรือ เกต NOR2

Gate	Equivalent gate count	Gate	Equivalent gate count
INV	1	RS Latch	3
NAIINR2	1	D Latch	4
NA/NR3	2	D Latch with CLR	5
NAIINR4	2	D Latch with PRE	5
NAIINR6	5	D Latch with PRE/CLR	6
NAIINR8	6	DFF	6
NAIINR9	7	DFF with CLR	7
NAIINR12	8	DFF with PRE	7
NAIINR16	11	DFF with PRE/CLR	8
BUFF	2	JKFF	9
ANIIOR2	2	JKFF with CLR	12

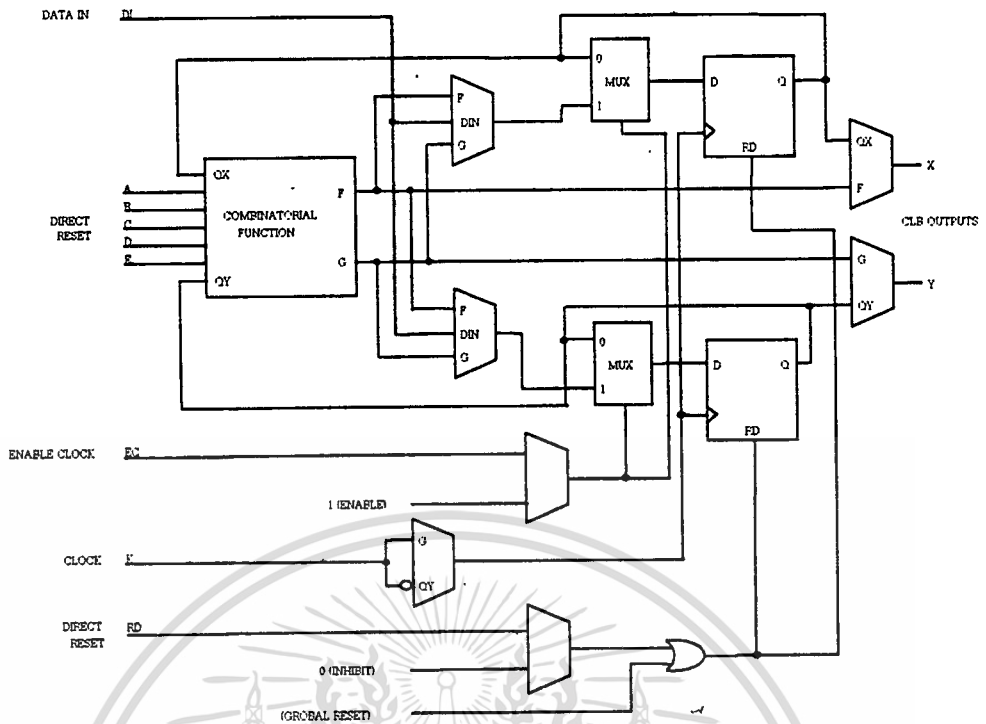
ตารางที่ 3.6 (ต่อ)

ANIIOR3	2	JKFF with PRE/CLR	13
ANIIOR4	3	TFF with CLR	8
XOR2	3	TFF with PRE	8
XNOR2	3	TFF with PRE/CLR	9

FPGA มีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกตอะเรย์ (Gate Array) มาก สามารถโปรแกรมและลบ Configuration สเตติกแรม (Static RAM) ภายในได้โดยใช้กระแสไฟฟ้า ซึ่งจะทำการโปรแกรมได้โดยดึงข้อมูลฐานสิบหกมาจากภายนอกเช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD, PAL ที่มี EPROM อยู่ภายในตัว ภายใน FPGA จัดเรียงเป็น ลอจิกเซลล์ ล้อมรอบภายนอกด้วยอินพุตเ้าพุตเซลล์ FPGA ตัวแรกที่ผลิตโดย Xilinx คือ XC2064 (2000-Family) ประกอบด้วยเซลล์เรียงเป็นเมทริกซ์มีจำนวนเซลล์ 64 เซลล์ หลังจากนั้นผลิตตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างภายใน CLB [10] ขั้วซ้อนขึ้นสามารถจำนวนเกตได้สูงและดีขึ้น แต่ละเซลล์เรียกว่า CLB (Configurable Logic Block) การต่อเซลล์อะเรย์ในเมทริกซ์แสดงในรูปที่ 3.12 ภายในเซลล์ CLB มีโครงสร้างภายในดังรูปที่ 3.13



รูปที่ 3.12 แสดงการจัดเรียงเซลล์เป็นเมทริกซ์ของ FPGA ตระกูล 3000



รูปที่ 3.13 บล็อกแสดงวงจรมายภายในของ CLB

3.1.3.3 รายละเอียดขา

อุปกรณ์ FPGA XC3042PC84 มีจำนวนขาทั้งหมด 84 ขา บรรจุอยู่ในตัวถังแบบ PLCC ดังแสดงในตารางที่ 3.7 โดยหน้าที่ของแต่ละขาเป็นดังนี้

Vcc	มี 2 ขา ทุกขาต้องต่อเข้ากับขั้ว (+) ของแหล่งจ่ายไฟ
GND	มี 2 ขา ทุกขาต้องต่อเข้ากับขั้ว (-) ของแหล่งจ่ายไฟ
PWRDWN	เมื่อขาสัญญาณอินพุตนี้เป็น "0" ฟลิปฟลอปและเล็ดทุกตัวถูกรีเซ็ต เอาพุตทุกขาเป็นไตรสเตจ อินพุตทั้งหมดเป็น "1" ทำการคอนฟิกใหม่ และเมื่อกลับเป็น "1" FPGA พร้อมทำงานเมื่อ DONE เป็น "0" 2 ไสเคลของคล็อกภายใน (1 Mhz) ก่อน ระหว่างการคอนฟิกขานี้ต้องเป็น "1" และถ้าไม่ใช้ขานี้ให้ต่อกับ Vcc
RESET	เป็นขาอินพุตที่ทำงานเมื่อเป็น "0" มีการทำงานสามลักษณะคือ 1. ก่อนที่จะเริ่มคอนฟิก ถ้าสัญญาณเป็น "0" จะหน่วงเวลาขบวนการคอนฟิก 2. ถ้าให้เป็น "0" ระหว่างการคอนฟิก FPGA จะทำการโหลดคอนฟิกใหม่ 3. ถ้าให้เป็น "0" หลังจากการคอนฟิกเสร็จสมบูรณ์จะไปรีเซ็ตขา RESET ทั้งหมดของ IOB และ CLB ที่ประกอบกันเป็น FPGA

CCLK	เป็นสัญญาณคล็อกเข้าพุตออกจากตัว FPGA เพื่อใช้โหลดคอนฟิกในโหมด Master หรือ Peripheral แต่จะเป็นอินพุตในโหมด Slave
DONE/PROG (D/P)	DONE เป็นทั้งสัญญาณอินพุตและเข้าพุตเมื่อเป็นเข้าพุตจะมีลอจิกเป็น "1" เมื่อโหลดคอนฟิกเสร็จสมบูรณ์ เมื่อเป็นอินพุตจะใช้โหลดคอนฟิกใหม่โดยการป้อนสัญญาณ "1" → "0" เข้าไป
M0/RTRIG	เป็นขาสัญญาณที่หน่วงการ Power-on โดยจะหน่วง 2^{14} ไซเคิลเมื่อ M0 เป็น "1" และจะหน่วง 2^{16} ไซเคิลเมื่อ M0 เป็น "0" ก่อนการโหลดคอนฟิกจะเป็นอินพุตร่วมกับ M1 M2 เพื่อใช้เซตโหมดการทำงานของ FPGA ถ้าป้อนสัญญาณ "0" → "1" หลังจากการโหลดคอนฟิกเสร็จจะเริ่มขบวนการ Readback คอนฟิก
M1/RDATA	ถ้าไม่ได้ Readback M1 สามารถต่อตรงกับ GND หรือ Vcc แต่ถ้า ใช้ Readback M1 ต้องใช้รีซิสเตอร์ 5 กิโลโอม์ขึ้นระหว่าง GND หรือ Vcc แอคทีฟ "0" เมื่ออ่านข้อมูล ก่อนการคอนฟิกใช้เซตโหมดร่วมกับ M0 M2 หลังจากโหลดคอนฟิกเสร็จ ขานี้จะเป็นขาเข้าพุตของข้อมูลที่อ่านกลับมา (Readback)
M2	ก่อนการคอนฟิกใช้เซตโหมดร่วมกับ M0 M1 หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
HDC	ระหว่างการโหลดคอนฟิกจะมีเข้าพุตเป็น "0" เพื่อแสดงให้เห็นว่าการโหลดคอนฟิกยังไม่สมบูรณ์ หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
EDC	ระหว่างการโหลดคอนฟิกจะมีเข้าพุตเป็น "0" เพื่อแสดงให้เห็นว่าการโหลดคอนฟิกยังไม่สมบูรณ์ หลังจากโหลดคอนฟิกเสร็จจะเป็น ขา I/O ที่ผู้ใช้สามารถโปรแกรมได้ สัญญาณนี้จำเป็นอย่างยิ่งในโหมด Master จะเป็น "0" เพื่อไปอีเนเบิล EPROM และจะเป็น "1" เมื่อโหลดคอนฟิกเสร็จ
INIT	เข้าพุตเป็นแบบ open-drain แอคทีฟ "0" ใช้เคลียร์หน่วยความจำที่ทำการคอนฟิก และยังใช้กำหนดสถานะให้กับไมโครโปรเซสเซอร์เพื่อใช้ในการโหลดคอนฟิก หลังจากโหลด คอนฟิกเสร็จจะเป็นขา I/O
CS0,CS1,CS2,WS	ควบคุมข้อมูลเข้าต่อนโหลดคอนฟิกในโหมด Peripheral ในโหมด Master-Parallel WS และ CS2 จะเป็นเข้าพุต A0 และ A1 หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
RDY/EUSY	ระหว่างการคอนฟิกในโหมด Peripheral Parallel ขาสัญญาณนี้ใช้เป็นเข้าพุต เพื่อบอกว่า FPGA พร้อมทั้งจะรับคอนฟิกไบต์ต่อไปหลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O

เอกสารนี้เป็นเอกสารของงานวิจัยที่จัดทำขึ้นเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้ทำซ้ำโดยไม่ได้รับอนุญาตจากทางผู้จัดทำ

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RCLK
ระหว่างการคอนฟิกในโหมด Master Parallel ขานี้จะใช้อ่านข้อมูลจากอุปกรณ์หน่วยความจำภายนอก ซึ่งโดยปกติสัญญาณนี้จะไม่ใช่
D0-D7
ทั้ง 8 ขาใช้รับข้อมูลขนาด 1 ไบต์ของคอนฟิกเข้ามาในโหมด Parallel Master และ Peripheral หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
A0-A15
เป็นแอดเดรสเอาต์พุตให้กับ EPROM เพื่อดึงเอาข้อมูลที่อยู่ใน EPROM มาใช้โหลดคอนฟิก หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
DIN
ระหว่างการคอนฟิกในโหมด Slave Serial หรือ Master serial ใช้เป็นอินพุตของข้อมูลที่ละบิต ในโหมด Master หรือ Peripheral ขานี้ก็คือ DATA0 นั่นเอง หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O
DOUT
ระหว่างการคอนฟิกใช้เป็นเอาต์พุตของข้อมูลที่ทำคอนฟิกเพื่อส่งต่อไปยัง FPGA ที่เป็น Slave หลังจากโหลดคอนฟิกเสร็จจะเป็นขา I/O ที่ผู้ใช้สามารถโปรแกรมได้
TCLKIN
เป็นอินพุต (สัญญาณ CMOS) ที่ต่อตรงมายังคล็อกทั้งหมด (Global clock buffer) ขานี้บ่อยครั้งที่ใช้เป็นสัญญาณคล็อกอินพุต และใช้เป็นขา I/O ได้ด้วย
I/O
ผู้ใช้สามารถโปรแกรมให้เป็นอินพุต หรือเอาต์พุตขึ้นอยู่กับคอนฟิก

ตารางที่ 3.7 ตำแหน่งและชื่อของสัญญาณของ XC3042

PIN NO.	XC3042	PIN NO.	XC3042
12	$\overline{\text{PWRDN}}$	67	D2-I/O
13	TCLKIN-I/O	68	I/O
14-21	I/O	69	I/O
22	VCC	70	D1-I/O
23-30	I/O	71	RDY/ $\overline{\text{BUSY}}$ -RCLK-I/O
31	M1-RDATA	72	D0-DIN-I/O
32	M0-RTRIG	73	DOUT-I/O
33	M2-I/O	74	CCLK
34	HDC-I/O	75	A0- $\overline{\text{WS}}$ -I/O

เอกสารนี้เป็นเอกสารของบริษัทไมครอสแตทิสต์ จำกัด เพื่อการใช้งานภายในเท่านั้น ไม่ควรเผยแพร่โดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 3.7 (ต่อ)

35	I/O	76	A1-CS2-I/O
36	$\overline{\text{LDC}}$ -I/O	77	A2-I/O
37-41	I/O	78	A3-I/O
42	INIT-I/O	79	I/O
43	GND	80	I/O
44-52	I/O	81	A15-I/O
53	XTL1(IN)-I/O	82	A4-I/O
54	$\overline{\text{RESET}}$	83	A14-I/O
55	DONE- $\overline{\text{PG}}$	84	A5-I/O
56	D7-I/O	1	GND
57	XTL1(OUT)-BCLKIN-I/O	2	A13-I/O
58	D6-I/O	3	A6-I/O
59	I/O	4	A12-I/O
60	D5-I/O	5	A7-I/O
61	$\overline{\text{CS0}}$ -I/O	6	I/O
62	D4-I/O	7	I/O
63	I/O	8	A11-I/O
64	VCC	9	A8-I/O
65	D3-I/O	10	A10-I/O
66	$\overline{\text{CS1}}$ -I/O	11	A9-I/O

3.1.3.4 รายละเอียดการใช้งาน

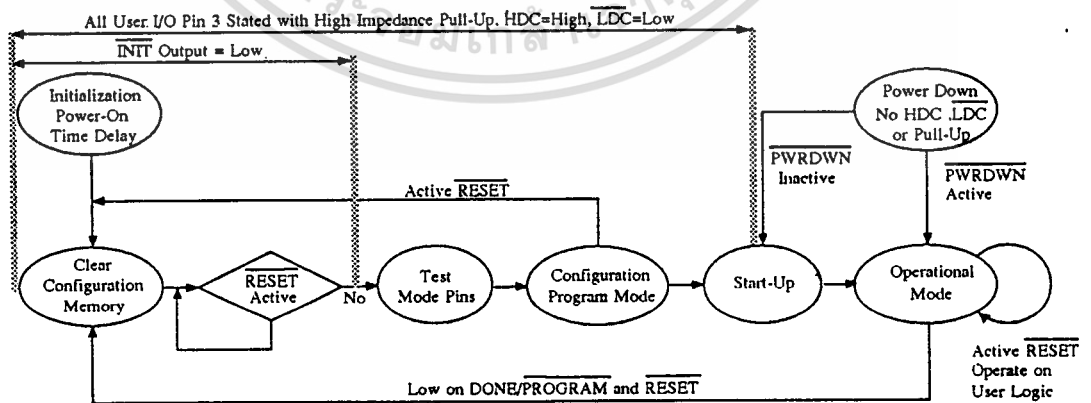
FPGA สามารถทำงานได้หลายโหมดโดยกำหนดได้ที่ขาสัญญาณ M0 M1 M2 ดังแสดงในตารางที่ 3.8 ในโหมด Master Parallel รับโปรแกรมคอนฟิกทีละ 1 ไบต์จากหน่วยความจำจากภายนอกที่เป็นแบบขนาน โดยสามารถรับโปรแกรมคอนฟิกจากแอดเดรสต่ำหรือสูงก่อนก็ได้ โหมด Peripheral รับโปรแกรมคอนฟิกทีละ 1 ไบต์ จากไมโครโปรเซสเซอร์ โดยสามารถโต้ตอบกันได้ว่าพร้อมหรือไม่พร้อมที่จะรับข้อมูลต่อไป โหมด Slave รับโปรแกรมคอนฟิกทีละ 1 บิตจากไมโครโปรเซสเซอร์ตามสัญญาณอินพุต CCLK ส่วนในโหมด Master Serial รับโปรแกรมคอนฟิกทีละ 1 บิต จากหน่วยความจำจากภายนอกที่เป็นแบบอนุกรม

ตารางที่ 3.8 โหมดต่างๆของการคอนฟิก (Configuration)

M0	M1	M2	COLK	MODE	DATA
0	0	0	output	Master Serial	Bit Serial
0	0	1	output	Master Parallel	Byte Wide Addr.=0000 up
0	1	0	----	reserved	----
0	1	1	output	Master Parallel	Byte Wide Addr.=FFFF down
1	0	0	----	reserved	----
1	0	1	output	Peripheral	Byte Wide
1	1	0	----	reserved	----
1	1	1	input	Slave Serial	Bit Serial

จากความต้องการสร้างฮาร์ดลอคให้ใช้กระแสไฟฟ้าต่ำ ในโหมดการต่อใช้งานทั้ง 5 โหมดจึงมีเพียง 2 โหมดเท่านั้นที่เหมาะสม คือ Master Serial Mode และ Slave Serial Mode ส่วนใน Master Parallel Mode ต้องใช้ EPROM 27Cxxx ซึ่งกินกระแสมากกว่า PROM XC17xxx เหมาะในการทดสอบต้นแบบก่อนเมื่อวงจรต้นแบบทำงานได้ถูกต้องแล้วจึงทำการอัดโปรแกรมลง PROM อีกทีหนึ่ง เพราะว่า ในโหมด Parallel EPROM สามารถโปรแกรมได้ใหม่ต่างกับ PROM ที่โปรแกรมได้ครั้งเดียว ในบทนี้จะกล่าวถึง FPGA เบอร์ XC3042 เป็นหลัก เนื่องจากใช้ได้ใช้ FPGA เบอร์นี้ในการออกแบบวงจรรวมฮาร์ดลอค

การใช้งาน FPGA ในโหมด Slave Serial และ Master Serial เมื่อเริ่มจ่ายไฟเข้าตัว FPGA (ตระกูล XC3000) จะทำการเคลียร์หน่วยความจำที่ใช้ในการคอนฟิก (Configuration Memory) ตรวจสอบโหมดการทำการคอนฟิกว่าเป็นโหมดใดในตารางที่ 3.8 (เป็นแบบอนุกรม หรือแบบขนาน) หลังจากนั้นจะเริ่มทำการโปรแกรมคอนฟิก สัญญาณ DONE/PROGRAM เป็น "0" (อยู่ในระหว่างการโปรแกรม) และเมื่อข้อมูลในการคอนฟิกที่รับมาจากภายนอกเต็มหน่วยความจำที่ใช้ในการคอนฟิก และความยาวของข้อมูลตรงกับที่เฮดเดอร์ของข้อมูลคอนฟิก สัญญาณ DONE/PROGRAM เป็น "1" (โปรแกรมคอนฟิกเสร็จสิ้น) รูปที่ 3.14 ประกอบ



Power-On Delay Is

2¹⁴ Cycles for Non-Master Mode--->11 to 33 ms

Clear is 290 Cycles for the XC3042--->195 to 580µs

2¹⁶ Cycles for Master Mode--->43 to 130 ms

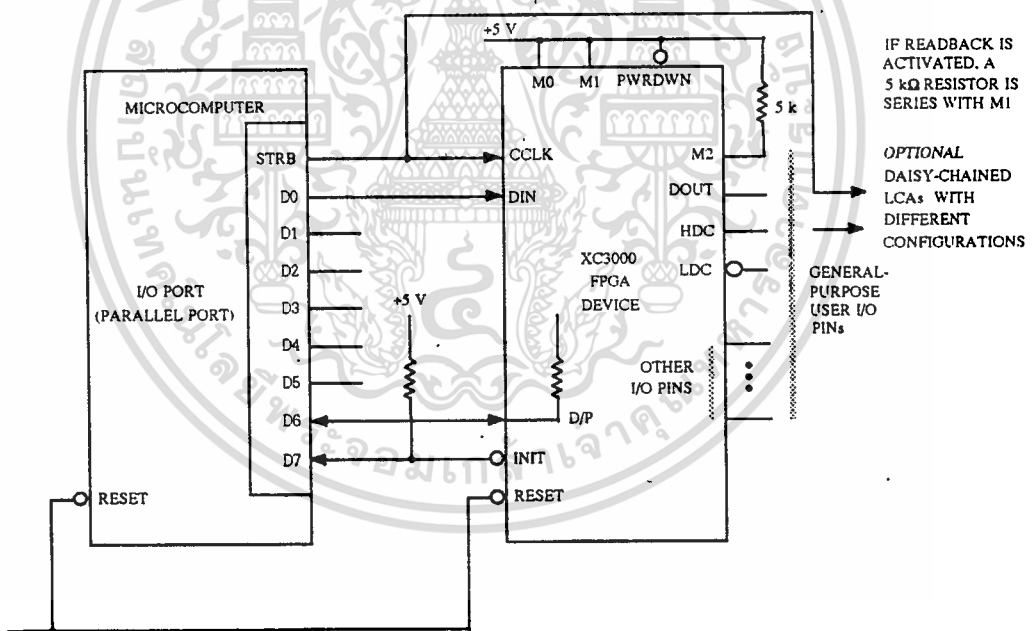
รูปที่ 3.14 ลำดับโต๊ะแกรมในการคอนฟิกเมื่อเริ่มป้อนแหล่งไฟเข้าไอซี และการโปรแกรมใหม่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่นิยมนำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.1.3.4.1 การใช้งานในโหมด Slave Serial

การต่อใช้งานในโหมดนี้เหมาะสมกับวงจรที่ออกแบบมาเพื่อทำงานร่วมกับไมโครคอมพิวเตอร์อยู่แล้ว ทั้งนี้เพราะ FPGA ได้ใช้ความสามารถของไมโครคอมพิวเตอร์ในการเก็บ และส่งข้อมูลคอนฟิกให้ เพียงแต่ต้องเขียนโปรแกรมเพื่อส่งโปรแกรมคอนฟิกให้เพิ่ม ลักษณะการต่อในโหมดนี้เป็นดังรูปที่ 3.15 ซึ่งในโหมดนี้ไมโครคอมพิวเตอร์จะสร้างสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGA ในโหมดนี้โปรแกรมคอนฟิกมีรูปแบบดังแสดงในรูป 3.16 โดยใช้โปรแกรม MakeBits [9] คำสั่ง RawBit เพื่อสร้างไฟล์ RBT ซึ่งมีส่วนที่เป็นหัวข้อมูล บอกขนาดของโปรแกรมคอนฟิก และตามด้วยข้อมูลที่ใช้ในการคอนฟิกเป็นเฟรมๆ จำนวน และขนาดของเฟรมขึ้นกับว่าเป็นโปรแกรมคอนฟิกของ FPGA เบอร์อะไรสำหรับเบอร์ XC3042 มีจำนวนเฟรมทั้งหมด 285 เฟรมแต่ละเฟรมมีขนาด 108 บิต เริ่มต้นด้วย 0 และจบด้วย 111 ทุกๆเฟรมดังแสดงในตัวอย่างตามรูปที่ 3.18 การป้อนโปรแกรมคอนฟิกให้ FPGA ทำได้โดยต่อสัญญาณ Strobe เข้ากับขา CCLK และ พอร์ต D0 เข้ากับขา DIN สร้างสัญญาณคล็อกป้อนให้ที่ขา CCLK และป้อนโปรแกรมคอนฟิกแบบอนุกรมเข้าที่ขา DIN ดังแผนภูมิในรูปที่ 3.17

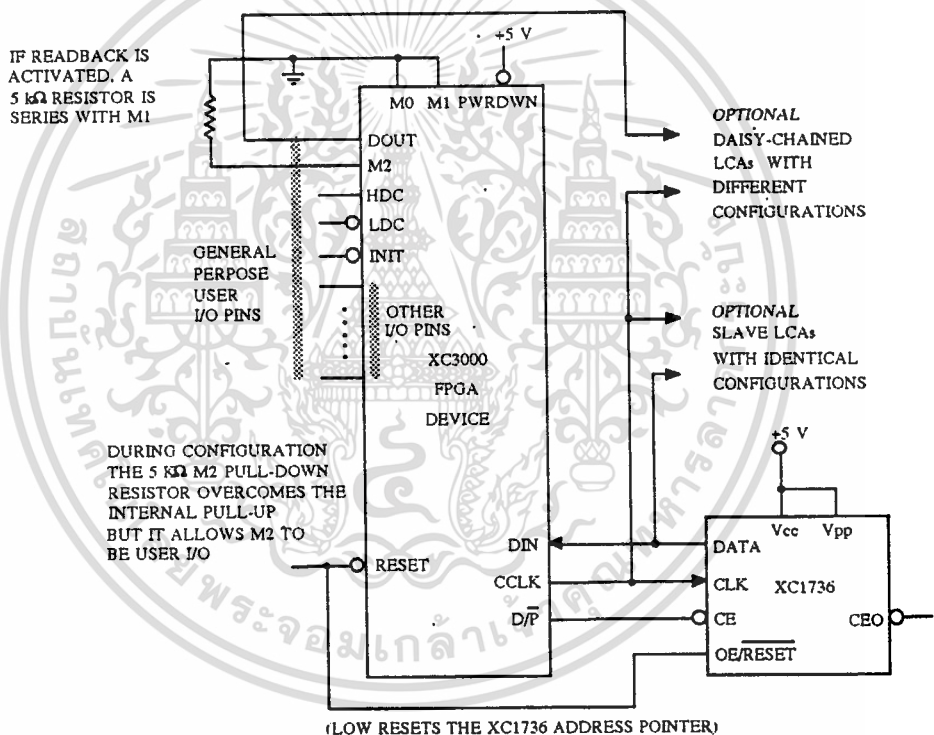


รูปที่ 3.15 แสดงการต่อใช้งานในโหมด Slave Serial

3.1.3.4.2 การใช้งานในโหมด Master Serial

การต่อใช้งานในโหมดนี้ ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากโหมดแรก คือใช้ PROM XC17xxx เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง PROM ทำตามขั้นตอนดังนี้คือ ใช้โปรแกรม MakeBits สร้าง .BIT ไฟล์จากวงจรที่ออกแบบ และใช้โปรแกรม MakePROM สร้าง Hex ไฟล์ แล้วทำการอัดโปรแกรมลง PROM ด้วยอุปกรณ์อัด PROM ที่มาพร้อมกับซอฟต์แวร์ของ Xilinx

โหมดนี้ PROM XC17xxx จะส่งสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGA ดังแสดงในรูปที่ 3.19 DIN เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบอนุกรม ส่วนอุปกรณ์ FPGA จะกำหนดสัญญาณ CCLK ให้กับ PROM เพื่อเป็นจังหวะในการอ่านข้อมูลโปรแกรมคอนฟิกมาไว้ใน Static RAM



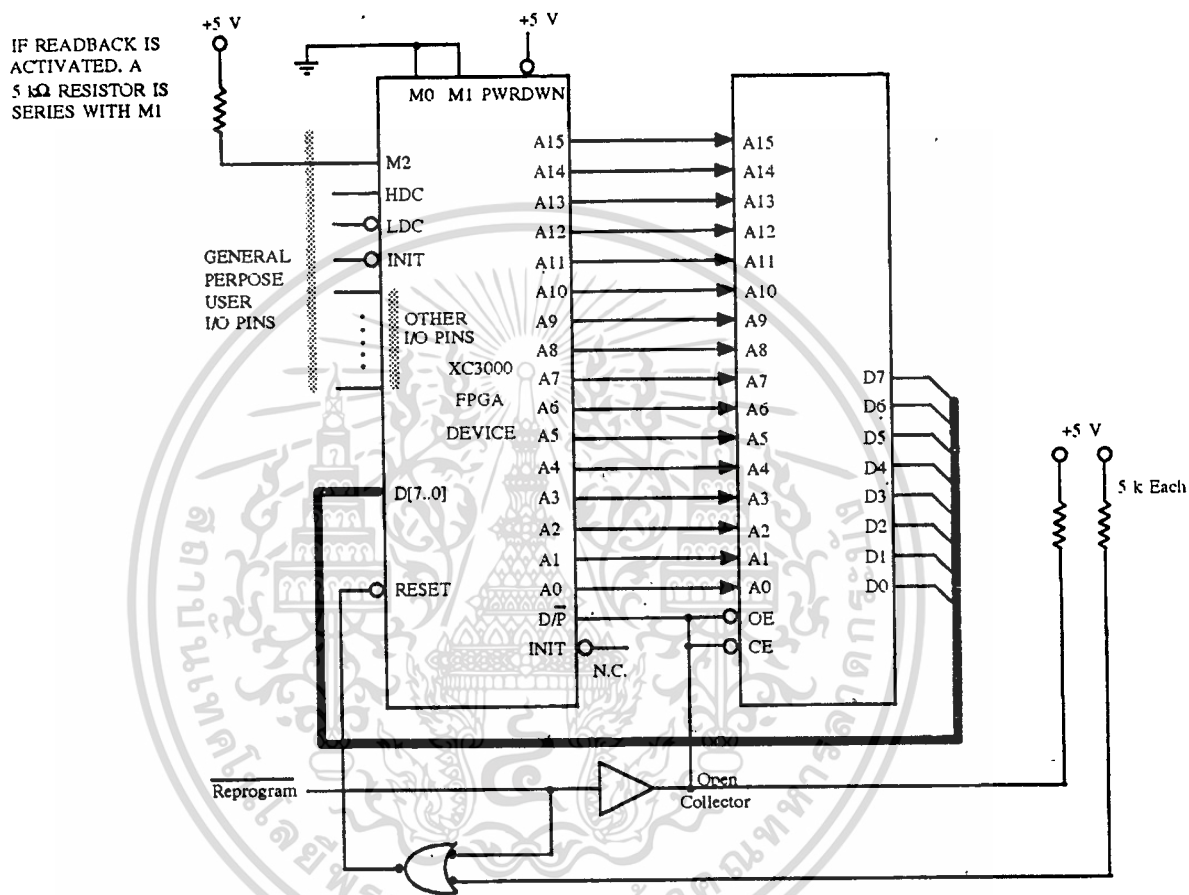
รูปที่ 3.19 แสดงการต่อใช้งานในโหมด Master Serial

3.1.3.4.3 การใช้งานในโหมด Master Parallel

การต่อใช้งานในโหมดนี้ ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากโหมด Master Serial คือใช้ EPROM 27xxx เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง EPROM ทำได้โดยใช้โปรแกรม MakePROM ใน XACT สร้าง Hex ไฟล์ .MCS แล้วทำการอัดโปรแกรมลง EPROM ด้วยอุปกรณ์อัด EPROM ที่มีใช้ทั่วไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหมดนี้ EPROM 27xxx จะส่งสัญญาณที่ละไบต์เพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGA ดังแสดงในรูปที่ 3.20 D0-D7 เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบขนานนี้ A15-A0 เป็นแอดเดรสที่ FPGA สร้างให้กับ EPROM เพื่ออ่านข้อมูลจากหน่วยความจำ (โปรแกรมคอนฟิก) มาเก็บไว้ในสเตติกแรม แอดเดรสทั้ง 16 เส้น ไม่จำเป็นต้องต่อครบก็ได้ขึ้นอยู่กับขนาดหน่วยความจำ EPROM ที่ใช้ และสามารถเช็ตให้นับเป็นแบบขึ้นหรือลงก็ได้



รูปที่ 3.20 แสดงการต่อใช้งานในโหมด Master Parallel

3.1.3.5 ข้อควรระวังในการใช้ FPGA

สิ่งแรกที่สำคัญคือ ไอซีกลัวความร้อนเป็นที่สุด การbakกรีโดยหัวแร้งกำลังสูง หรือbakกรีโดยจี้หัวแร้งที่ขาไอซีนานๆ จะทำให้ไอซีเสียหายได้ง่าย ระยะเวลาในการbakกรีหนึ่งจุดไม่ควรเกิน 5-10 วินาที ควรใช้ช็อกเกตไอซีในการประกอบวงจรลงบนแผ่นปริ้นต์

การป้อนแหล่งจ่ายไฟให้ไอซีไม่ควรผิดชั่ว ถ้าสลับขั้ว (+) หรือ (-) กันจะทำให้ไอซีเสียได้ นอกจากนั้น แรงดันของแหล่งจ่ายไฟต้องอยู่ในช่วงที่โรงงานกำหนดมา สำหรับ FPGA แรงดันที่ไอซีอยู่ในช่วงทำงานได้ $V_{cc} = 4.75-5.25$ โวลต์ และแรงดันที่ทนได้อยู่ในช่วง $-0.5-7$ โวลต์ ดังนั้นก่อนป้อนแรงดันควรตรวจสอบเช็คให้แน่ใจก่อน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การต่อวงจรผิดที่มีผลให้อิซซีเสียหายทันทีได้แก่ การต่อเอาพุตของอิซซีไปเข้าขั้ว (+) หรือ (-) ของแหล่งจ่ายไฟโดยตรง อิซซี TTL จะเสียหายทันที ในขณะที่ CMOS ยังพอทนได้บ้าง เพราะมีการจำกัดกระแสขาออกอยู่ภายในตัว ดังนั้นก่อนบื่อนแรงดันเข้าอิซซีควรตรวจสอบวงจรบริเวณเอาพุตของอิซซีด้วย

อีกประการหนึ่งที่อิซซีประเภท MOS กลัวมากที่สุดคือ ไฟฟ้าสถิต ขาที่เป็นอันตรายได้ง่ายที่สุดคือขาอินพุต ความต้านทานขาเข้าของขาอินพุตของอิซซีประเภทนี้มักจะสูงมากจนมีคุณสมบัติเหมือนกับเป็นฉนวนที่ขาอินพุตนี้อาจรับไฟฟ้าสถิตจากที่ใดมาเก็บไว้ เมื่อสะสมไว้มากๆ ไม่มีทางระบายออกก็จะทำให้วงจรบริเวณอินพุตเสียหายได้ ในปัจจุบันเทคโนโลยีในการผลิตอิซซีเจริญรุดหน้าไปมาก ทางด้านอินพุตของอิซซี MOS จะมีวงจรป้องกันไว้ภายใน อย่างไรก็ตามเพื่อป้องกันเหตุการณ์ดังกล่าวควรเก็บรักษาอิซซีที่ถูกต้อง และควรเก็บไว้ในที่ปลอดภัยจากไฟฟ้าสถิตโดยเก็บไว้ในบนแผ่นเยื่อสังเคราะห์แบบนำกระแสได้ หรือห่อหุ้มด้วยกระดาษอลูมิเนียม

อิซซีประเภท CMOS มีความต้านทานขาเข้าสูงมาก กระแสจึงไม่ไหลจากแหล่งจ่ายไฟเข้าไปที่ขาอินพุตของ CMOS นั้นถึงแม้จะต่อความต้านทานคั่นกลางอยู่ แรงดัน 5 โวลต์จากแหล่งจ่ายไฟก็ยังป้อนเข้าขาอินพุต 5 โวลต์เต็ม โดยที่ไม่มีแรงดันตกคร่อมบนตัวต้านทานเลย ถ้ามีกระแสไหลแสดงว่าอิซซีตัวนั้นเสีย เอาพุตของอิซซี CMOS อยู่ในสภาวะ "1" จะสามารถขับโหลดได้หลายมิลลิแอมป์ และถ้าเอาพุตอยู่ในสภาวะ "0" ก็สามารถจะรับกระแสไหลเข้าได้หลายมิลลิแอมป์

3.2 การใช้ซอฟต์แวร์ ViewLogic และ Xilinx

ในการสร้างต้นแบบวงจรฮาร์ดลอคได้เปลี่ยนมาใช้ซอฟต์แวร์ Viewlogic [8] โปรแกรม Viewdraw เขียนวงจรฮาร์ดลอคเป็นผังวงจรไฟฟ้า (Schematic) โดยใช้ไลบรารีของ FPGA ตระกูล XC3000 ซึ่งต่างจากผังวงจรไฟฟ้าเดิมที่เขียนด้วยซอฟต์แวร์ของ Mentor Graphics โดยใช้ไลบรารีของ LS_LIB แล้วใช้โปรแกรม Viewsim จำลองการทำงานจนได้วงจรที่ทำงานตรงกับความต้องการ เมื่อถึงขั้นตอนการใช้ซอฟต์แวร์ Xilinx [9] จะได้ขนาดของ FPGA ที่ใช้สร้างวงจรรวมฮาร์ดลอคที่ออกแบบไว้เป็น XC3042A จากนั้นสร้าง PROM ไฟล์เพื่อนำไปโปรแกรมลงบน PROM XC1736 ซึ่งเหมาะสมที่จะใช้เป็นตัวป้อนโปรแกรมคอนฟิก (Configuration) ให้กับอุปกรณ์ FPGA XC3042A ท้ายสุดทดสอบอุปกรณ์ FPGA XC3042A ที่มีโปรแกรมคอนฟิกของวงจรรวมฮาร์ดลอคจนกระทั่งสามารถทำงานได้ถูกต้องตามที่ต้องการ กระบวนการต่างๆในการออกแบบวงจรรวมฮาร์ดลอคได้แสดงไว้ในรูปที่ 3.21

3.2.1 ขั้นตอนในการออกแบบ และจำลองการทำงานวงจรโดยใช้ ViewLogic

เริ่มต้นเซตไลบรารีที่ต้องการจะใช้ให้เป็น FPGA ตระกูล XC3000 แล้วใช้โปรแกรม Viewdraw เขียนผังวงจรไฟฟ้าของวงจรรวมฮาร์ดลอค

แปลงไฟล์ที่เป็นผังวงจรไฟฟ้าเป็น Wirelist เพื่อนำไปจำลองการทำงานโดยโปรแกรม VSM

จำลองการทำงานของวงจรโดยโปรแกรม ViewSim แล้วใช้ ViewTrace เพื่อดูรูปสัญญาณทั้งที่ขาอินพุต (Waveform) และเอาพุตโดยรับสัญญาณอินพุตที่ผู้ออกแบบวงจรได้สร้างไว้ เพื่อทดสอบวงจรให้ได้ตรงกับความต้องการ ซึ่งมีหลายวิธีโดยอาจจะป้อนทีละคำสั่ง หรือเขียนเป็นคำสั่งต่อกันเก็บไว้ในรูปของไฟล์ข้อความ (Text File) นามสกุล .cmd

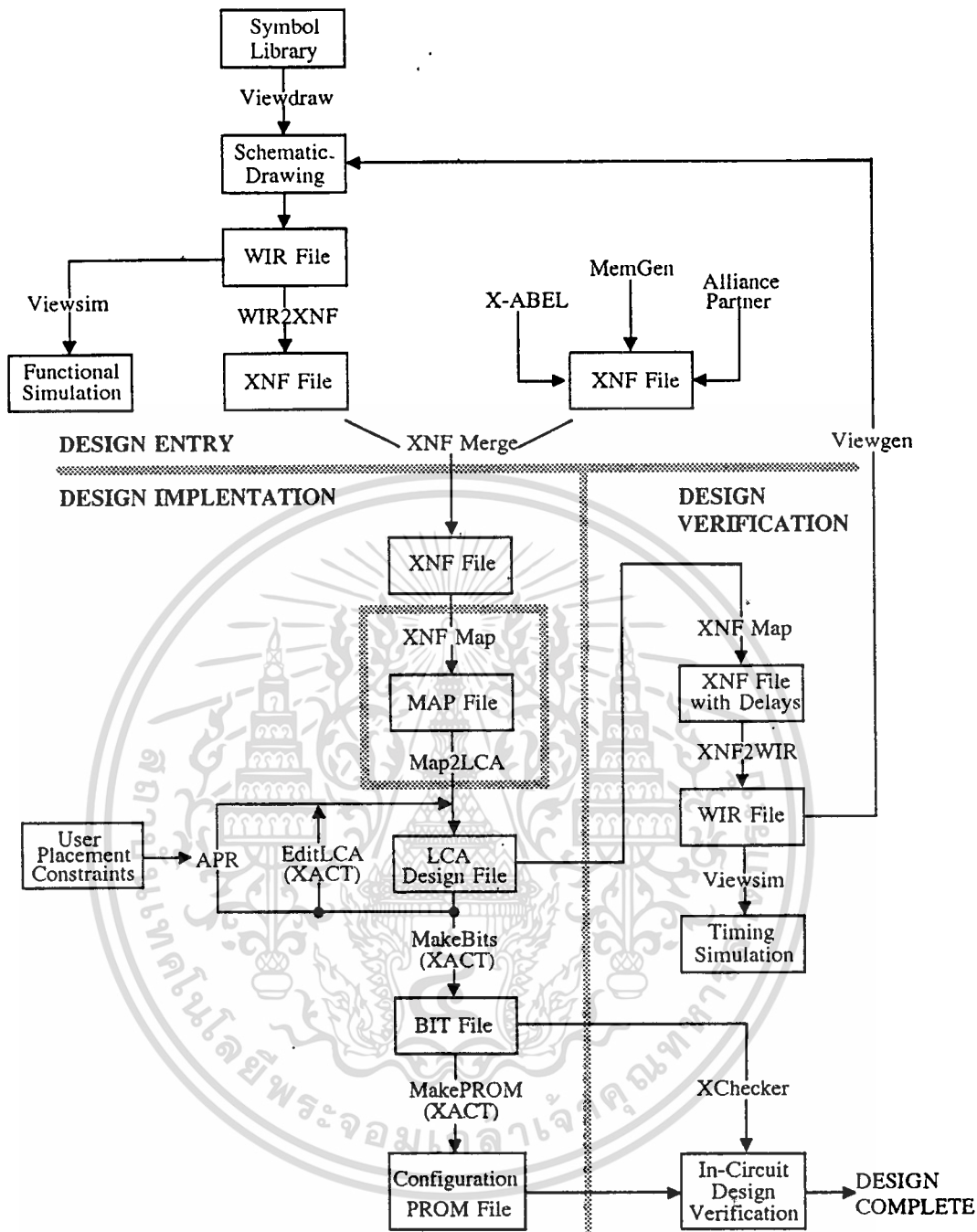
3.2.2 ขั้นตอนในการทดสอบวงจรโดยใช้ Xilinx

สร้าง Map file จาก Netlist โดยใช้โปรแกรม XnfMap และใช้โปรแกรม Map2Lca สร้างไฟล์ .LCA ซึ่งสามารถแก้ไขได้โดยผู้ออกแบบ หรือใช้โปรแกรม APR (Automatic Place and Route) จัดการให้อัตโนมัติ ในขั้นตอนทั้งหมดที่กล่าวมาข้างต้นสามารถใช้โปรแกรม Xmake

แปลงไฟล์ .LCA เป็น .BIT โดยโปรแกรม MakeBit

แปลงเป็น Intel Hex File เพื่อใช้เป็นโปรแกรมคอนฟิกให้กับ FPGA ในกรณีที่ทดสอบวงจรในโหมด Master Parallel หรือแปลงเป็น BitStream เพื่อใช้โปรแกรมลงบน PROM ตระกูล XC17xxx ซึ่งเป็นตัวโหลดโปรแกรมคอนฟิกให้กับอุปกรณ์ FPGA ในกรณีที่ทดสอบวงจรในโหมด Master Serial

เพื่อให้การจำลองการทำงานของวงจรสมบูรณ์มากยิ่งขึ้น หลังจากที่ได้ไฟล์ที่มีโครงสร้างเป็น .LCA แล้ว โดยใช้ซอฟต์แวร์แปลงกลับไปเป็นผังวงจรไฟฟ้า เพื่อจำลองการทำงานใหม่อีกครั้งให้ผลการจำลองตรงกันกับครั้งแรกที่จำลองการทำงาน ซึ่งข้อดีของผังวงจรไฟฟ้าใหม่ที่สร้างขึ้นนี้เปรียบเสมือนการยวบวงจรให้นั่นเอง



รูปที่ 3.21 โฟลว์ชาร์ตการออกแบบวงจรโดยใช้ ViewLogic และ Xilinx

3.3 ฮาร์ดแวร์ของระบบ

หลังจากที่ออกแบบวงจรรวมฮาร์ดลอคด้วยซอฟต์แวร์ของ Mentor Graphic บนเครื่อง Engineering Workstation แล้วนำวงจรนั้นมาเขียนเป็นผังวงจรไฟฟ้าใหม่โดยใช้ซอฟต์แวร์ของ ViewLogic บนเครื่องไมโครคอมพิวเตอร์ โดยทำการปรับเปลี่ยนวงจรบางส่วนเนื่องจากไลบรารีที่มีต่างกัน ทำให้ได้วงจรรวมฮาร์ดลอคใหม่ที่ทำงานได้เหมือนเดิมทุกประการ รูปวงจรทั้งหมดดูในภาคผนวก ก.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

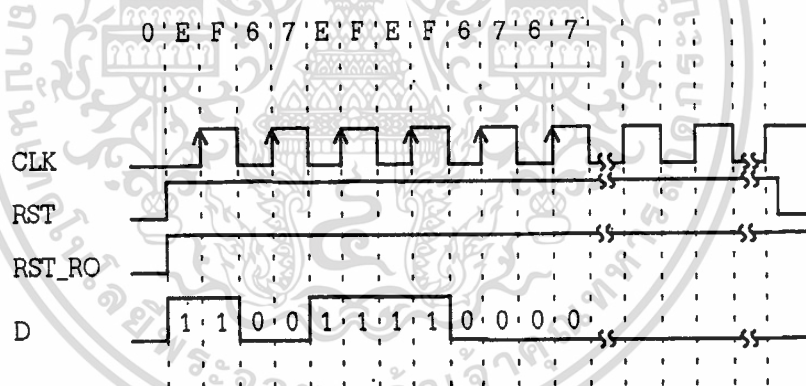
ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.4 ซอฟต์แวร์ของระบบ

ในการเขียนโปรแกรมเพื่อใช้ติดต่อสื่อสารกับฮาร์ดลอค จะต้องส่งข้อมูลคำสั่งทั้งหมดเป็นแบบอนุกรมผ่านพอร์ตขนานออกที่ขา D3 แล้วคอยอ่านรหัสกลับที่ขา PE โดยส่งสัญญาณรีเซ็ตฮาร์ดลอคออกที่ขา D1, รีเซ็ตเฉพาะวงจรสร้างข้อมูลสุ่มที่ขา D2 และสัญญาณคล็อกออกที่ขา D0 ดังนั้นจึงต้องมีวิธีเขียนโปรแกรมเพื่อสร้างสัญญาณส่งคำสั่งให้กับฮาร์ดลอคทางพอร์ตขนานเป็นไปอย่างกระชับ และถูกต้อง ซึ่งวิธีการเขียนโปรแกรมสร้างสัญญาณแบบนี้จะเป็นส่วนประกอบสำคัญของโปรแกรมป้องกันการคัดลอกโปรแกรม .EXE (HL_SHELL)

3.4.1 การสร้างสัญญาณผ่านพอร์ตขนาน

ถ้าต้องการส่งข้อมูลให้กับฮาร์ดลอคที่ขา D เป็น 101100 นำมาเขียนเป็นแผนภูมิสัญญาณที่ป้อนให้กับฮาร์ดลอคอย่างง่าย (ฮาร์ดลอครับคำสั่งด้วยคล็อกขาขึ้น จึงเปลี่ยนสัญญาณคำสั่งทุกๆคล็อกขาลง) ได้ดังรูปที่ 3.22 จะเห็นว่าสัญญาณที่ขา D เท่านั้นที่เปลี่ยนแปลงไม่มีรูปแบบโดยขึ้นอยู่กับสัญญาณข้อมูลคำสั่งที่ป้อน ส่วนขา CLK จะเปลี่ยนแปลงซ้ำแบบเป็นคาบเวลา เพราะฉะนั้นหลังจากการรีเซ็ตแล้วลักษณะของสัญญาณทั้งหมดจะมี 4 รูปแบบดังรูปที่ 3.23



รูปที่ 3.22 ตัวอย่างสัญญาณที่ป้อนให้ฮาร์ดลอค

CLK	D
0	0
0	1
1	0
1	1

รูปที่ 3.23 รูปแบบคำสั่งที่ป้อนตามสัญญาณคล็อก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อนำเอกลักษณ์ของสัญญาณทั้ง 4 รูปแบบตามรูปที่ 3.23 มาเขียนเป็นโปรแกรมสร้างสัญญาณจะได้สองคำสั่งหลัก คือ คำสั่งช่วงที่สัญญาณคล็อกเป็น 0 และ ช่วงที่สัญญาณคล็อกเป็น 1 ดังรูปที่ 3.24

```
MAIN()
{
    CHAR DATA[12]="10011110000";           ;ข้อมูลคำสั่ง
    INT J,P;
    P=3*16+7*16+8;                           ;พอร์ตนาน 378H
    OUTPORT (P,0);                            ;รีเซ็ตฮาร์ดลอค
    FOR (J=0;J<12;J++)
    {
        OUTPORT (P,6+(DATA[J]-'0')*8);       ;ส่งคำสั่งช่วงที่ CLK เป็น 0
        OUTPORT (P,7+(DATA[J]-'0')*8);       ;ส่งคำสั่งช่วงที่ CLK เป็น 1
    }
}
```

รูปที่ 3.24 โปรแกรมสร้างสัญญาณผ่านพอร์ตนาน

3.4.2 โปรแกรมสนับสนุนฮาร์ดลอค

อุปกรณ์ฮาร์ดลอคจะทำงานได้ต้องใช้โปรแกรมควบคุมการทำงาน ซึ่งโปรแกรมหากล่าวมียู่สองส่วน ส่วนแรกจะเป็นโปรแกรมสนับสนุนการทำงานของฮาร์ดลอคซึ่งผู้ผลิตฮาร์ดลอคเป็นผู้เขียนให้ ส่วนที่สองเป็นโปรแกรมที่ฝังอยู่ในซอฟต์แวร์ที่ต้องการสงวนสิทธิ ซึ่งซอฟต์แวร์เฮาส์ (Software house) ที่ซื้อฮาร์ดลอคไปจะเป็นผู้เขียนเพิ่มเข้าไปในซอฟต์แวร์ที่มีอยู่ ภายในบทนี้จะกล่าวถึงซอฟต์แวร์สนับสนุนการทำงานของฮาร์ดลอคซึ่งประกอบด้วยโปรแกรมต่างๆ ดังนี้

ID_GEN.EXE	ทำหน้าที่เขียนรหัสชี้เฉพาะ (Identify Code : Security_ID) ลงฮาร์ดลอค
CODE_GEN.EXE	ทำหน้าที่เขียนรหัสข้อมูลลงฮาร์ดลอค
HL_SHELL.EXE	เป็นโปรแกรมเข้ารหัสโปรแกรมสกุล .EXE หรือ .COM เพื่อให้การเรียกใช้โปรแกรมโปรแกรมครั้งต่อไปต้องมีฮาร์ดลอค

ID_GEN.EXE	เริ่มต้นใช้งานฮาร์ดลอคด้วยการใส่ Security_ID ฮาร์ดลอคทุกตัวจะต้องทำการเขียน Security_ID ของซอฟต์แวร์เฮาส์ลงในฮาร์ดลอคเสียก่อน เมื่อฮาร์ดลอคถูกเขียน Security_ID แล้ว Software house จากที่อื่นๆ จะไม่สามารถติดต่อกับฮาร์ดลอคตัวนั้นได้ และ Security_ID นี้ก็จะไม่สามารถที่จะแก้ไข หรือเขียนทับใหม่ได้อีก วิธีการเขียน Security_ID ทำได้โดยต่อฮาร์ดลอคไว้ที่พอร์ต และเขียน Security_ID ไว้ที่ไฟล์ DATA_WR ก่อน แล้วรันโปรแกรม ID_GEN ศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
------------	---

CODE_GEN.EXE

หลังจากที่ตัวฮาร์ดลอคมี Security_ID แล้ว ต่อไปจะต้องทำการใส่รหัสลับลงไป ซึ่งรหัสลับที่ใส่ไปนี้จะถูกอ่านโดยโปรแกรม HL_SHELL ที่ทำการลอค และถอดลอค

HL_SHELL.EXE

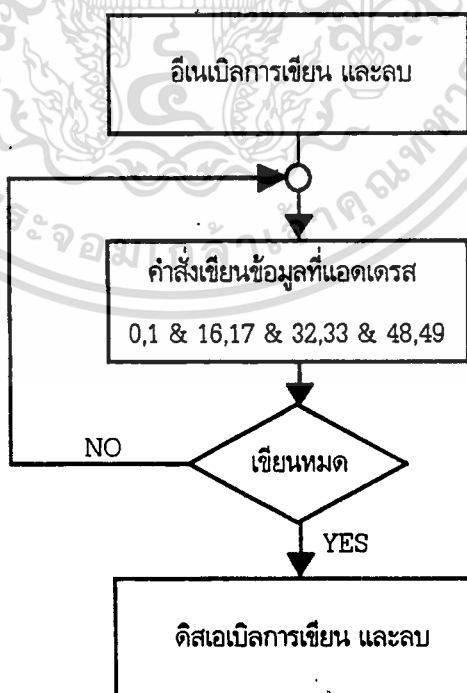
เป็นไฟล์ที่ใช้ในการลอคซอฟต์แวร์อย่างง่ายคือใช้วิธีการที่เรียกว่า HARD-SHELL ซึ่งทำได้โดยใช้โปรแกรม HL_SHELL.EXE ซึ่งเป็นโปรแกรมที่ใช้ลอคโปรแกรมใดๆ ที่ถูกคอมไพล์แล้วเป็นไฟล์ .EXE หรือ .COM มีวิธีใช้ดังตัวอย่างข้างล่าง การลอคไฟล์หมายถึงการเข้ารหัสโปรแกรมสกุล .EXE หรือ .COM เพื่อให้การเรียกใช้ โปรแกรมครั้งต่อไปต้องมีฮาร์ดลอคต่ออยู่ด้วย

HL_SHELL source_file

source_file คือ ไฟล์ที่ต้องการลอค

3.4.2.1 โปรแกรม ID_GEN

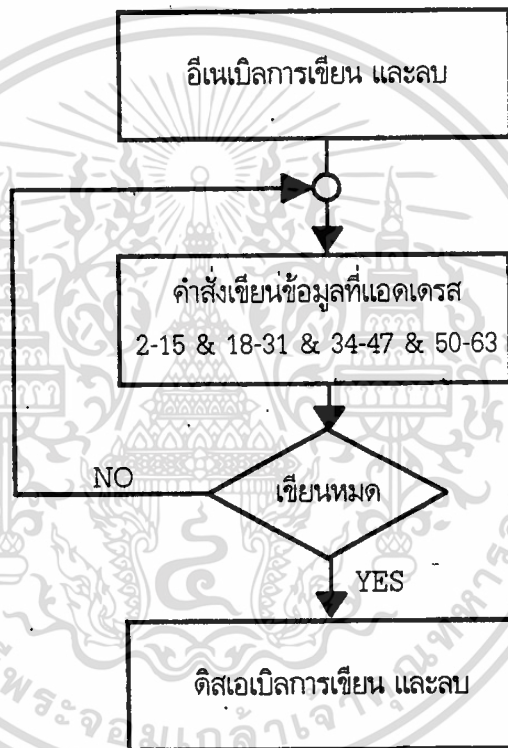
เริ่มต้นโปรแกรมจะทำการอินิเวิลการเขียนและลบข้อมูลก่อนเนื่องจากในการเขียนข้อมูลลง EEPROM ที่นำมาเป็นหน่วยความจำของฮาร์ดลอค จำเป็นต้องอินิเวิลก่อนจึงจะสามารถเขียนข้อมูลได้ หลังจากนั้นทำการเขียนข้อมูลที่ได้จากไฟล์ข้อความ DATA_WR (มีข้อมูล 8 ชุด) ลงแอดเดรส 0,1 & 16,17 & 32,33 & 48,49 (Identify Code) ตามลำดับ เมื่อเขียนครบทุกแอดเดรสจะทำการดิสเวิลการเขียน และลบข้อมูลเพื่อความปลอดภัยของข้อมูล ขั้นตอนการทำงานเขียนเป็นโฟลว์ชาร์ตได้ดังรูปที่ 3.25 ตัวโปรแกรมดูในภาคผนวก ค.



รูปที่ 3.25 โฟลว์ชาร์ตการเขียนรหัสเฉพาะ

3.4.2.2 โปรแกรม CODE_GEN

เริ่มต้นโปรแกรมจะทำการอีเนเบิลการเขียน และลบข้อมูลก่อนเนื่องจากในการเขียนข้อมูลลง EEPROM ที่นำมาเป็นหน่วยความจำของฮาร์ดล็ค จำเป็นต้องอีเนเบิลก่อนจึงจะสามารถเขียนข้อมูลได้ หลังจากนั้นทำการเขียนข้อมูลที่ได้จากไฟล์ข้อความ DATA_WR (มีข้อมูล 64-8 ชุด) ลงแอดเดรส 2-15 & 18-31 & 34-47 & 50-63 (Secret Code) ตามลำดับ เมื่อเขียนครบทุกแอดเดรสจะทำการดิสเอเบิลการเขียน และลบข้อมูลเพื่อความปลอดภัยของข้อมูล ขั้นตอนการทำงานเขียนเป็นโฟลว์ชาร์ตได้ดังรูปที่ 3.26 ซึ่งใช้ในกรณีที่เขียนรหัสลับเป็นครั้งแรกเท่านั้น แต่ถ้าต้องการเขียนรหัสลับใหม่จะต้องทำการลบรหัสที่มีอยู่เดิมด้วยคำสั่งลบก่อน แล้วจึงทำการเขียนรหัสลับใหม่ลงไป ตัวโปรแกรมดูในภาคผนวก ค.



รูปที่ 3.26 โฟลว์ชาร์ตการเขียนรหัสลับครั้งแรก

3.4.2.3 โปรแกรม HL_SHELL

โปรแกรมที่ใช้ป้องกันการคัดลอกมีทั้งหมดสองโปรแกรมคือ โปรแกรม HL_SHELL.CPP เป็นโปรแกรมหลักและ DECODE.CPP เป็นโปรแกรมรอง ซึ่งตัวโปรแกรมทั้งสองแสดงไว้ในภาคผนวก ข.

การทำงานของโปรแกรม คือการอ่านข้อมูลรหัสลับ และข้อมูลสุมจากฮาร์ดล็คครั้งละ 16 บิต มาเข้ารหัสกับภาษาเครื่องของโปรแกรมที่ต้องการล็คครั้งละ 8 บิต ทำให้ได้ข้อมูลใหม่ขนาดที่มีขนาดเท่ากับโปรแกรมที่ถูกล็ค ตัวอย่างเช่น

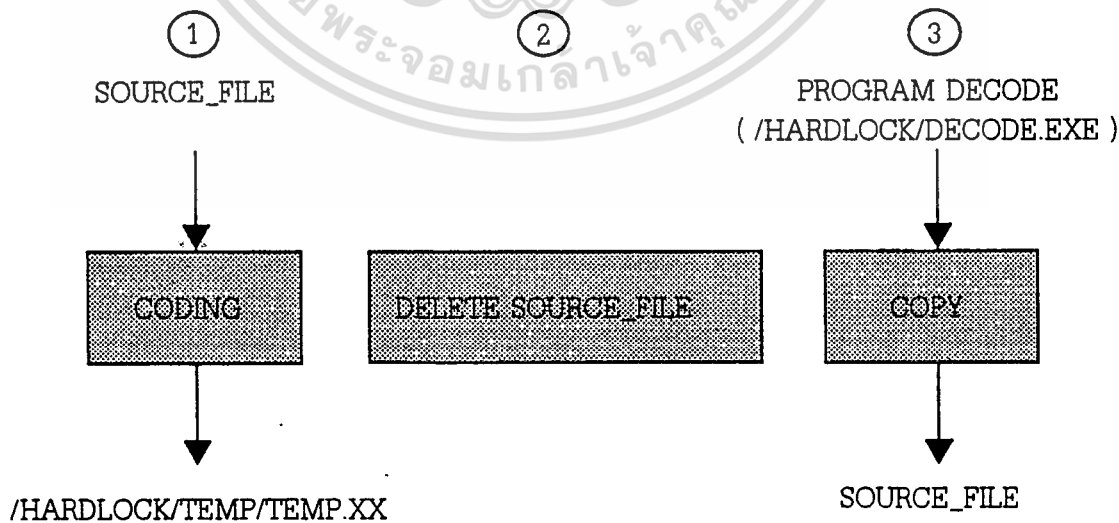
ข้อมูลภาษาเครื่องของโปรแกรมเป็น	1A H
ข้อมูลรหัสลับที่อ่านได้เป็น	23BC H
การเข้ารหัส	[1A H Xor 23 H] Xor [BC H]
ผลลัพธ์ที่ได้จากการเข้ารหัส	85 H

เมื่อเข้ารหัสโปรแกรมทั้งหมดได้เป็นโปรแกรมใหม่แล้ว เรียกใช้โปรแกรมที่ล็อคใหม่อีกครั้งจะต้องถอดรหัสโปรแกรมที่ได้เข้ารหัสไว้ก่อนแล้วให้กลับไปเหมือนกับโปรแกรมที่ไม่ได้ล็อคเพื่อให้เอ็กซีคิวต์ได้ดังเดิมตัวอย่างเช่น

ข้อมูลภาษาเครื่องของโปรแกรมที่ล็อค	85 H
ข้อมูลรหัสลับที่อ่านได้เป็น	23BC H
การถอดรหัส	[85 H Xor 23 H] Xor [BC H]
ผลลัพธ์ที่ได้จากการถอดรหัส	1A H

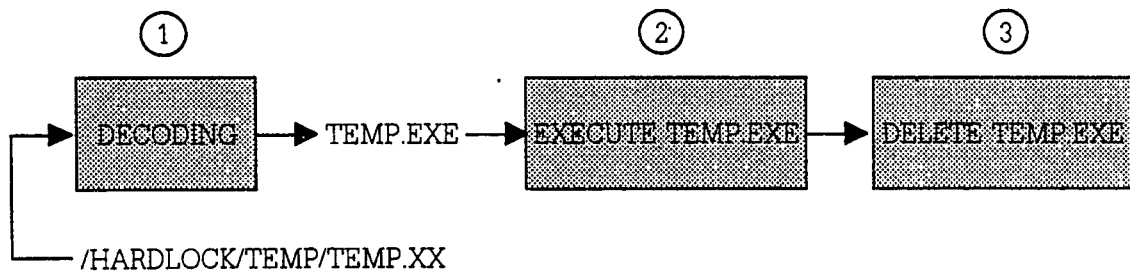
เมื่อใช้คำสั่ง HL_SHELL Source_file ตัวโปรแกรม HL_SHELL จะสร้างไฟล์สำรองที่ได้จากการเข้ารหัสโปรแกรม (ใช้รหัสจากฮาร์ดล็อค) ขึ้นมาชื่อ TEMP.xx อยู่ใน PATH \HARDLOCK\TEMP\ หลังจากนั้นจะลบ Source_file ดังแสดงขั้นตอนไว้ในรูปที่ 3.27 ท้ายสุดจะก๊อปปี้โปรแกรมที่ใช้ถอดรหัสไฟล์ TEMP.xx ชื่อ DECODE.EXE ไปไว้ในชื่อของ Source_file เพื่อให้โปรแกรม Source_file สามารถเอ็กซีคิวต์ได้อีกครั้ง

โปรแกรม DECODE.EXE จะทำการถอดรหัส TEMP.xx ที่สร้างจาก HL_SHELL เป็นชื่อ TEMP.EXE อยู่ใน PATH ของ Source_file หลังจากนั้นจะเอ็กซีคิวต์ TEMP.EXE เมื่อออกจากโปรแกรมจะลบโปรแกรม TEMP.EXE ออกด้วย ดังรูปที่ 3.28



รูปที่ 3.27 การทำงานของโปรแกรม HL_SHELL.EXE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3.28 การทำงานของโปรแกรม DECODE.EXE

ตัวอย่าง สมมติว่าต้องการล๊อคโปรแกรมชื่อ C:\CW\CW.EXE ตัวโปรแกรม HL_SHELL จะทำงานตามขั้นตอนดังต่อไปนี้

การเข้ารหัส
1. ทำการเข้ารหัสโปรแกรม C:\CW\CW.EXE สร้างเป็นไฟล์ชื่อ C:\HARDLOCK\TEMP\TEMP.1
2. เขียนชื่อโปรแกรมที่เข้ารหัสไว้ใน C:\HARDLOCK\HISTORY.INI
3. ลบโปรแกรม C:\CW\CW.EXE
4. ก๊อปปี้โปรแกรม C:\HARDLOCK\DECODE.EXE ไปเป็นโปรแกรม C:\CW\CW.EXE

การเรียกใช้
หลังจากที่เข้ารหัสแล้วครั้งต่อไปเมื่อเรียกเอ็กซีคิวต์โปรแกรม C:\CW\CW.EXE ตัวโปรแกรมจะค้นหาโปรแกรมใน C:\HARDLOCK\HISTORY.INI เพื่อหาว่าโปรแกรมที่เข้ารหัสไว้ลำดับที่เท่าไรถ้าเป็นลำดับที่ 1 จะไปเปิดโปรแกรม C:\HARDLOCK\TEMP\TEMP.1 มาถอดรหัสแล้วเก็บโปรแกรมไว้ใน C:\CW\TEMP.EXE จากนั้นเอ็กซีคิวต์โปรแกรม C:\CW\TEMP.EXE หลังจากthatเอ็กซีคิวต์จะลบโปรแกรมนี้ออก

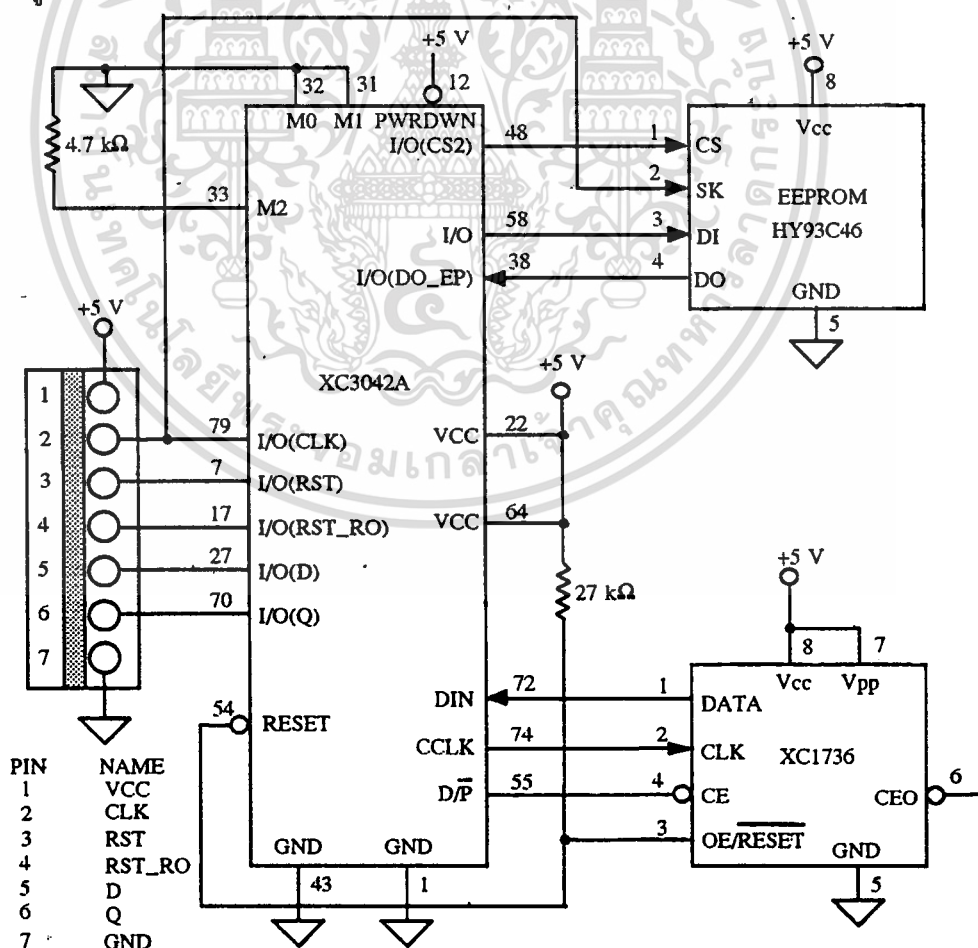
บทที่ 4

การสร้าง และการทดสอบ

หลังจากที่ได้ทำการจำลองการทำงาน (Simulation) ของวงจรฮาร์ดลอคจนกระทั่งทำงานตามที่ต้องการแล้ว จะต้องนำวงจรที่ได้มาทดสอบด้วยเทคโนโลยีวงจรรวมอีกครั้ง เพื่อป้องกันความบกพร่องของผู้ออกแบบวงจร และทดสอบการทำงานให้ได้มากแบบยิ่งขึ้น สมบูรณ์ยิ่งขึ้น

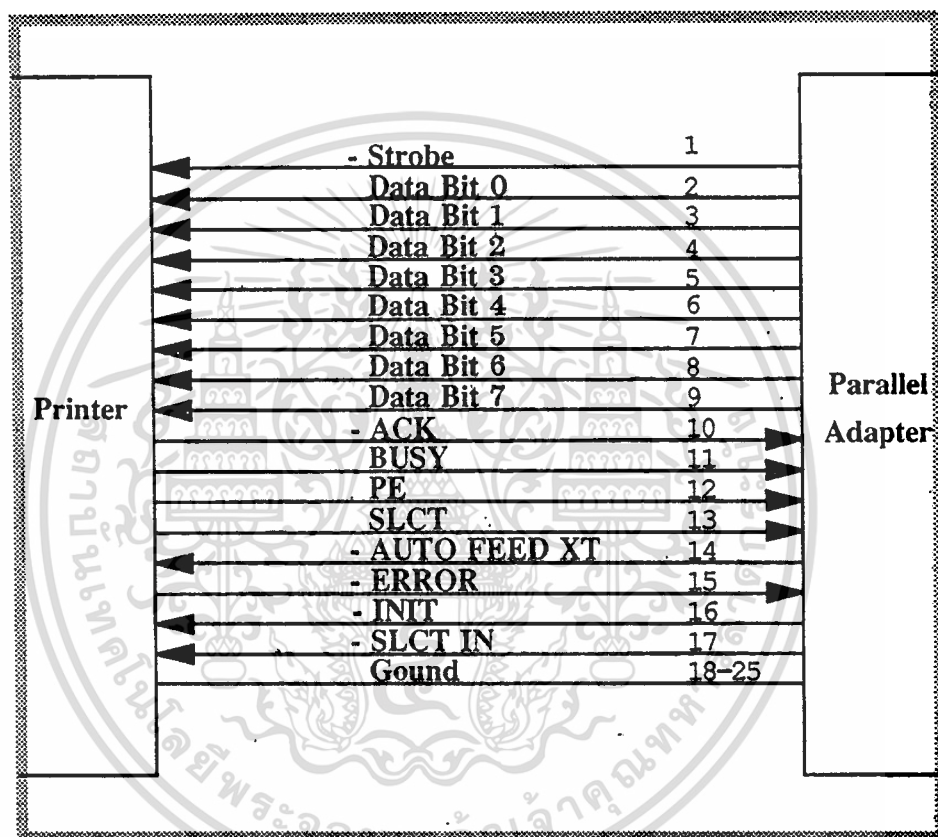
4.1 ต้นแบบฮาร์ดลอค

วิทยานิพนธ์ฉบับนี้ใช้เทคโนโลยีของ FPGA สร้างเป็นวงจรต้นแบบเพื่อทดสอบการทำงานของวงจรที่จำลองไว้แล้ว โดยนำวงจรที่ได้จำลองการทำงานแล้วมาสร้างเป็นโปรแกรมคอนฟิกเพื่ออัดลงบน PROM XC1736 (ดูบทที่ 3 ประกอบ) จากนั้นสร้างวงจรต้นแบบฮาร์ดลอคที่สมบูรณ์โดยเชื่อมต่อ PROM ที่ได้กับ FPGA XC3042A ซึ่งเซตให้เป็นโหมด Master Serial และ EEPROM HY93C46 ที่ใช้เป็นหน่วยความจำ ดังแสดงในรูปที่ 4.1



4.2 วิธีเชื่อมต่อต้นแบบฮาร์ดล็ค

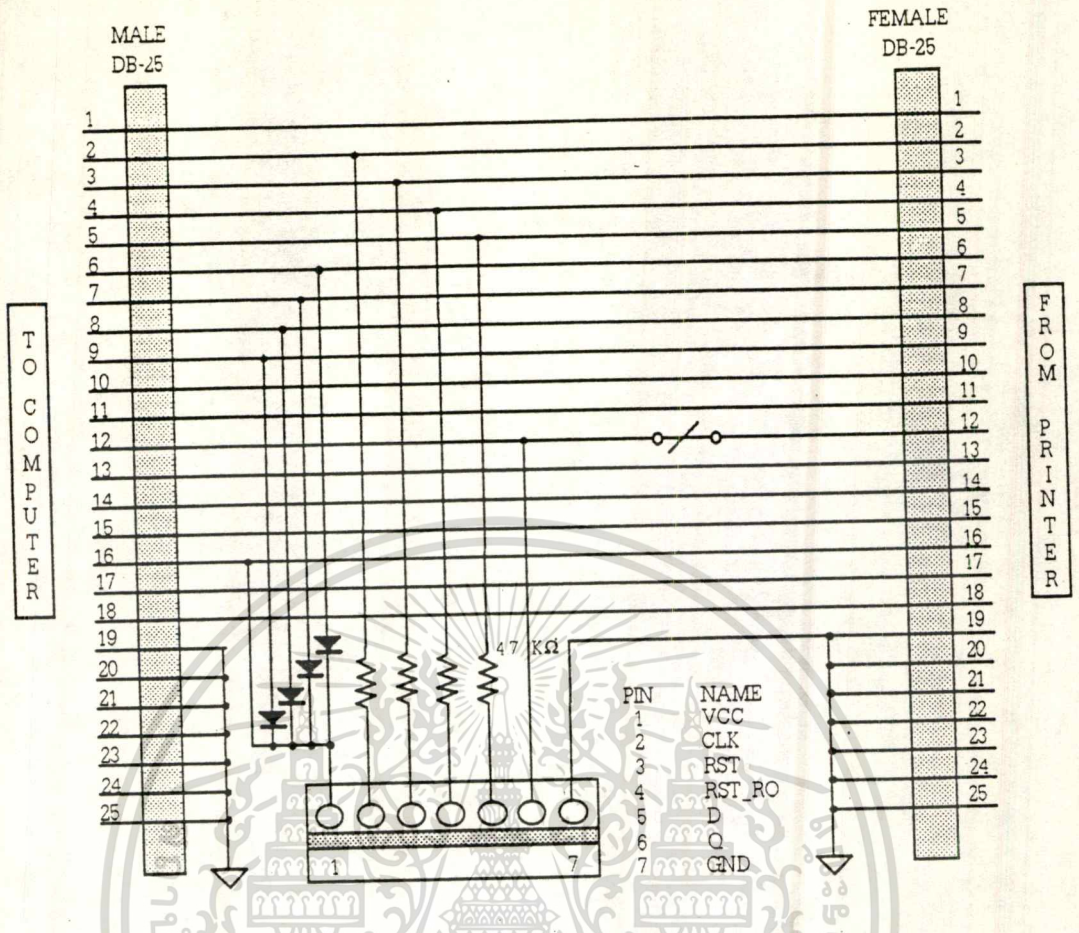
ในการทดสอบได้เชื่อมต่อฮาร์ดล็คเข้ากับพอร์ตขนาน โดยทั่วไปพอร์ตขนานนี้ปกติจะใช้เป็นเป็นเอาต์พุต เพื่อส่งข้อมูลให้กับเครื่องพิมพ์. แต่ก็ยังสามารถรับข้อมูลเป็นอินพุตพอร์ตได้เช่นกัน ซึ่งในวิทยานิพนธ์นี้ใช้เป็นเอาต์พุตพอร์ตส่งข้อมูลคำสั่งให้ฮาร์ดล็คทางขาสัญญาณ D0-D3 ส่วนการรับข้อมูลจากฮาร์ดล็คได้ใช้ขาสัญญาณ PE เป็นสัญญาณที่เครื่องพิมพ์ส่งมาให้คอมพิวเตอร์ว่ากระดาษหมดขอบแล้วดังรูปที่ 4.2 โดยพอร์ตที่ใช้ส่งข้อมูลคำสั่งเป็นพอร์ตหมายเลข 378H ส่วนพอร์ตที่ใช้รับข้อมูลเป็นพอร์ตหมายเลข 379 H



รูปที่ 4.2 ชื่อขาสัญญาณพอร์ตขนาน

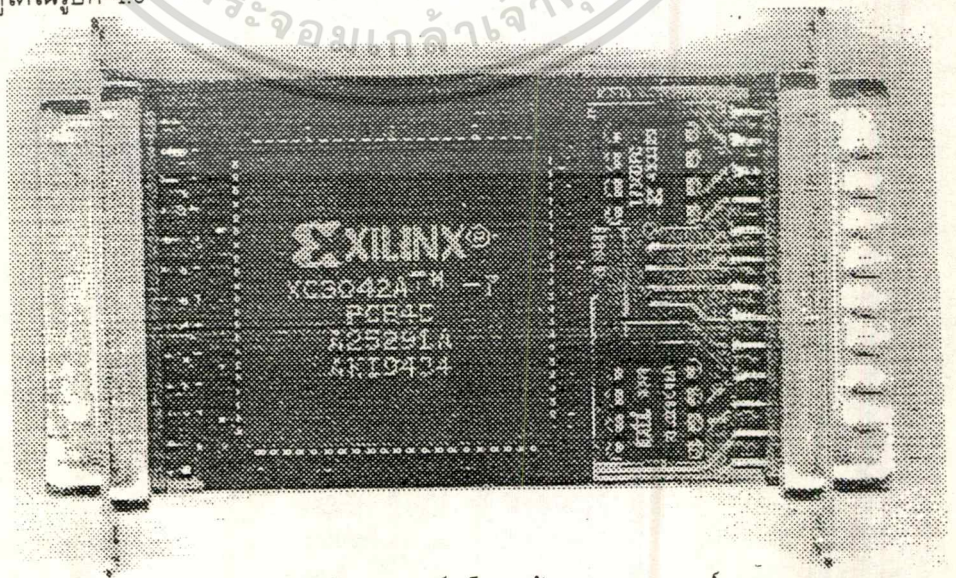
เนื่องจากพอร์ตขนานไม่มีขาสัญญาณไฟเลี้ยง +5 โวลต์ เมื่อต้องการเชื่อมต่อฮาร์ดล็คโดยไม่ใช้แหล่งจ่ายไฟจากภายนอก จึงจำเป็นต้องสร้างแหล่งจ่ายไฟให้เพียงพอความต้องการของฮาร์ดล็คประมาณ 30 mA โดยดึงกระแสของขาสัญญาณ D4-D7 (ทำให้มีลอจิกเป็น "1" ขณะใช้งาน) และ INIT จากพอร์ตขนานมาใช้

นอกจากการสร้างแหล่งจ่ายไฟแล้วยังควบคุมการทำงานผ่านทางขาสัญญาณ D0-D3 ซึ่งต่ออยู่กับ CLK RST RST_RO และ D ตามลำดับ โดยจะอ่านรหัสลับ หรือรหัสหลุดจากฮาร์ดล็คเข้ามาทางขา PE ซึ่งต่ออยู่กับเอาต์พุต Q ของฮาร์ดล็ค (ดูรูปที่ 4.1 ประกอบ) การเชื่อมต่อขาสัญญาณทั้งหมดที่กล่าวมากับพอร์ตขนานเขียนเป็นรูปวงจรได้ดังรูปที่ 4.3



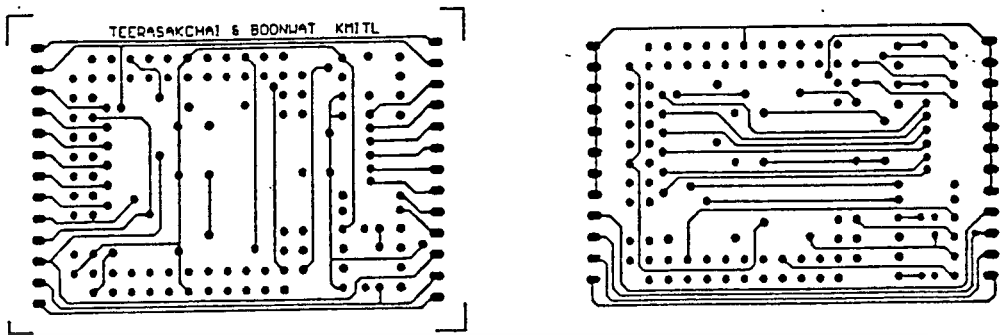
รูปที่ 4.3 การเชื่อมต่อต้นแบบฮาร์ดลอคกับพอร์ตขนาน

นำวงจรในรูปที่ 4.1 และ 4.3 มารวมกันได้ต้นแบบฮาร์ดลอคที่มีคอนเน็คเตอร์ 25 ขาอยู่ด้วยดังแสดงในรูปที่ 4.4 ภายในมีไอซี 3 ตัวคือ FPGA XC3042 , PROM XC1736 และ EEPROM HY93C46 ส่วนลายพิมพ์วงจรดูได้ในรูปที่ 4.5



รูปที่ 4.4 ต้นแบบฮาร์ดลอคพร้อมคอนเน็คเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4.5 ลายพิมพ์ต้นแบบฮาร์ดล๊อค

4.3 การทดสอบต้นแบบ

วิธีการทดสอบต้นแบบฮาร์ดล๊อคเริ่มจากการเชื่อมต่อกับไมโครคอมพิวเตอร์ที่ฟอร์ตชานก่อน แล้วใช้ซอฟต์แวร์ Borland C สร้างโปรแกรมในภาคผนวก ค. ประกอบด้วย โปรแกรมทดสอบการเขียนรหัสฮาร์ดล๊อคซึ่งในโปรแกรมมีการเรียกใช้ฟังก์ชันอินเบิล และดิสเอเบิล ฟังก์ชันการลบทัสด้วยโปรแกรม CODE_GEN.CPP โปรแกรมทดสอบการอ่านรหัสจากฮาร์ดล๊อค CODE_RD.CPP โปรแกรมทดสอบการสร้างข้อมูลสุ่ม RAND_RD.CPP และโปรแกรมทดสอบการนำไปประยุกต์ใช้ป้องกันการคัดลอกโปรแกรม HL_SHELL.CPP จะเห็นว่าโปรแกรมที่กล่าวมาสามารถทดสอบคำสั่งของฮาร์ดล๊อคได้ทั้งหมดทุกคำสั่ง

ก่อนที่จะทดสอบฮาร์ดล๊อคด้วยโปรแกรมทั้งหมดที่กล่าวมา ฮาร์ดล๊อคจะต้องทำการเขียนรหัสเฉพาะไปเก็บไว้ในแอดเดรส 0,1,16,17,32,33,48,49 ตามลำดับ ซึ่งมีข้อมูลเดิมเป็น FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ถ้าต้องการเขียนเป็น 9342 A1B6 C108 5D72 EBA3 64E2 D49C 1AF8 จะต้องใช้โปรแกรมเขียนรหัสเฉพาะ ID_GEN.CPP (ภาคผนวก ค.) ที่มีรหัสผ่านเริ่มต้นอยู่ในไฟล์ C:\HARDLOCK\STANDARD.INI เป็น HOSTID FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF และสร้างไฟล์ข้อความเก็บรหัสเฉพาะที่ต้องการเขียน DATA_WR เป็นดังนี้ 9342 A1B6 C108 5D72 EBA3 64E2 D49C 1AF8 โปรแกรม ID_GEN.CPP มีลักษณะการทำงานเหมือนกับโปรแกรม CODE_GEN.CPP ต่างกันตรงแอดเดรสที่ต้องการเขียนนั่นเองโดยถ้าเป็นโปรแกรม ID_GEN.CPP จะเขียนข้อมูลลงแอดเดรสที่เป็นส่วนของรหัสเฉพาะ ส่วนโปรแกรม CODE_GEN.CPP จะเขียนข้อมูลลงแอดเดรสที่เป็นส่วนของรหัสลับ

4.3.1 การทดสอบด้วยโปรแกรมเขียนรหัส CODE GEN

โปรแกรมนี้จะเขียนรหัสฮาร์ดล๊อค ตามไฟล์ข้อความที่สร้างไว้ดังแสดงในรูปที่ 4.6 โดยแถวแรกเขียนลงในแอดเดรส 2-15 แถวที่สองเขียนลงในแอดเดรส 18-31 แถวที่สามเขียนลงในแอดเดรส 34-47 และเอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แถวที่สี่เขียนลงแอดเดรส 50-63 ตามลำดับ ฮาร์ดล๊อคจะอ่านรหัสผ่านขนาด 128 บิตจากไฟล์ C:\HARDLOCK\STANDARD.INI ซึ่งมีข้อความดังนี้ HOSTID 9342A1B6C1085D72EBA364E2D49C1AF8 เพื่อเปรียบเทียบกับข้อมูลที่เป็นรหัสชี้เฉพาะของฮาร์ดล๊อค

1CB5	789B	ADBF	983C	067D	CD13	ADC5	CE40	C108	64E2	B181	E0D1	CB12	45CD
5B67	1D34	BCDA	D134	ABC6	90B7	D14B	734C	3208	B56C	97AB	67DC	E8FC	5CBC
B809	63D3	5CB6	1504	12CA	7A1B	8C4A	456A	ADCA	0986	C803	B805	F3BA	AC00
C322	609B	F067	B770	C6C5	A345	AC78	EFB0	BA57	8F87	945E	A45A	1BB2	12B4

รูปที่ 4.6 ตัวอย่างรูปแบบข้อมูลที่ใช้เขียนลงฮาร์ดล๊อค

เมื่อสร้างไฟล์ข้อความที่ใช้เป็นข้อมูลที่ต้องการเขียนลงฮาร์ดล๊อคไว้ในชื่อ DATA_WR แล้วเอ็กซีคิวต์โปรแกรม CODE_GEN โปรแกรมจะทำงานตามขั้นตอนดังนี้

อีนเบิ้ลการเขียน และลบข้อมูล
ลบรหัสลับเดิมที่แอดเดรส 2 ออกแล้วเขียนรหัสลับใหม่ 1CB5 ลงแอดเดรสที่ 2 แทน ทำเช่นนี้ซ้ำ ที่แอดเดรส 2-15 , แอดเดรส 18-31 ,แอดเดรส 34-47 และแอดเดรส 50-63 ตามลำดับ
*หมายเหตุ ถ้ารหัสลับยังไม่ได้เขียนมาก่อน (FFFF) ไม่จำเป็นต้องลบรหัสลับเดิม สามารถเขียนรหัสลับได้เลย
ดิสเอเบิลการเขียน และลบข้อมูล เพื่อป้องกันข้อมูลสูญหาย

ผลจากการทดสอบด้วยโปรแกรมนี้สามารถตรวจสอบได้ด้วยโปรแกรม CODE_RD ทำหน้าที่อ่านรหัสลับมาเก็บไว้ในไฟล์ข้อความ DATA_RD ซึ่งจะกล่าวในหัวข้อต่อไป

4.3.2 การทดสอบด้วยโปรแกรมอ่านรหัส CODE_RD

โปรแกรมนี้จะทำการอ่านรหัสจากฮาร์ดล๊อคมาเก็บไว้ในไฟล์ข้อความชื่อ DATA_RD โดยแถวแรกเป็นรหัสลับที่อยู่ในแอดเดรส 2-15 แถวที่สองเป็นรหัสลับที่อยู่ในแอดเดรส 18-31 แถวที่สามเป็นรหัสลับที่อยู่ในแอดเดรส 34-47 และแถวที่สี่เป็นรหัสลับที่อยู่ในแอดเดรส 50-63 โดยอ่านรหัสผ่านขนาด 128 บิต จากไฟล์ C:\HARDLOCK\STANDARD.INI มีข้อความดังนี้ HOSTID 9342A1B6C1085D72EBA364E2D49C1AF8 ผลจากการทดสอบด้วยโปรแกรมนี้ดูที่ไฟล์ข้อความ DATA_RD ซึ่งจะมีข้อมูลตรงกับข้อมูลที่อยู่ในไฟล์ข้อความ DATA_WR หลังจากที่เอ็กซีคิวต์โปรแกรม CODE_GEN มาแล้ว

4.3.3 การทดสอบด้วยโปรแกรมอ่านรหัสสุ่ม RAND_RD

เมื่อเอ็กซีคิวต์โปรแกรม RAND_RD โปรแกรมจะทำการอ่านรหัสสุ่มจากฮาร์ดล๊อคมาเก็บไว้ในไฟล์ข้อความชื่อ DATA_RM ผลบางส่วนของรหัสสุ่มแสดงในรูปที่ 4.7 ซึ่งรหัสสุ่มที่ได้จะกำหนดขึ้นเมื่อป้อนคำสั่ง เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผิดๆให้กับฮาร์ดล็อก เช่นเป็นคำสั่งที่ไม่ตรงตามตารางฟังก์ชันที่ 2.3 หรือรหัสผ่านไม่ตรงกับรหัสเฉพาะ จากที่กล่าวมาจะเห็นว่าคำสั่งที่สร้างรหัสสุมถือเป็นคำสั่งหนึ่งเสริมเข้ามาจากคำสั่งในตารางที่ 2.3 ได้เหมือนกัน

CE40	C5D7	5B8B	4561	ADF0	7654	4414	360D	FC43	EE79	B0CC	A535	E557	521E
576B	53EE	9BCC	4B49	B2BB	9540	9F79	23C0	9A5F	238D	8B7D	458F	5363	C13C
DD40	D257	6309	CF0D	F32B	2D44	43C5	B64F	6794	57FA	DF1E	73A9	FD41	0A4D
616F	C3F2	91BB	44CD	4CFF	104F	AD2A	AEB9	89C1	919D	FC26	0F25	D21F	950B
CB64	0670	8BC7	9B1E	9E0E	7BD7	D126	319D	E65C	95C9	435C	3715	8B48	0204
C969	A346	CC8F	E8D9	2DA6	3629	8742	6CCD	CAE1	76F4	09BA	C6BE	59CA	9090
6C0D	8133	86FF	B148	B444	879D	5300	655F	FC21	2BEF	711A	0033	02B4	D9B8
ABFC	2DEF	A1DE	639C	C528	9992	EAB7	2CF3	5000	0D3D	CD76	4821	D10F	A016
023A	2157	1C2C	A9C0	4987	FE40	03DA	B08F	FC82	C45E	A070	4179	625D	C8D1
99E3	D041	34F5	552C	C330	1993	75A8	4C70	8955	FA4D	7629	9468	48D9	37DC
ACC5	1801	CED3	8ACD	7280	4B14	FEE6	0B4B	038E	908D	8687	E7E1	484C	3DF9
C67C	D1F2	A1E5	6C82	1829	D198	69D9	A9ED	5ED9	98FA	C6BF	383F	7037	30B3
74C3	C267	BC40	3EB9	55B6	E075	D486	C495	4417	9412	DD9A	0ABA	AEDC	689D
F34E	CC18	2116	8671	49A7	0394	03B2	BB59	F883	3AD3	02F2	672F	E29E	F2C4
95FC	04D7	AD1F	E932	13CB	F3E7	EDE9	35AB	9CB0	6C2F	BF0D	47F9	02E3	5BA5
F020	EA93	57F2	D2B4	7710	1EB7	A4CC	45AF	AEB7	C154	D696	D666	9D94	6D8F
D57D	A787	79D0	BA50	B302	CE10	6300	A352	1725	920C	18B4	D617	07D9	8DE8

รูปที่ 4.7 ข้อมูลสุมบางส่วนที่สร้างขึ้น

4.3.4 การทดสอบด้วยโปรแกรมป้องกันการคัดลอก HL_SHELL

ก่อนการทดสอบจะต้องต่อฮาร์ดล็อกกับพอร์ตนานก่อน แล้วติดตั้งสภาวะเริ่มต้นให้กับฮาร์ดล็อกโดยเอ็กซีคิวทีฟโปรแกรม INSTALLH.CPP ซึ่งมีลักษณะการทำงานเป็นไปตามขั้นตอนดังนี้

สร้างไดเรกทอรี C:\HARDLOCK
ก๊อปปี้โปรแกรม HISTORY.INI ,STANDARD.INI ,HL_SHELL.EXE ,DECODE.EXE ,POWER.EXE ลงไปใน C:\HARDLOCK
เพิ่ม PATH=C:\HARDLOCK และ C:\HARDLOCK\POWER.EXE ในแบ็ตไฟล์
เอ็กซีคิวทีฟ AUTOEXEC.BAT ใหม่อีกครั้งเพื่อให้ใช้ฮาร์ดล็อกได้เลยโดยไม่ต้อง BOOT เครื่องใหม่ เพื่อเอ็กซีคิวทีฟโปรแกรม POWER.EXE (ทำให้ D4-D7 มีลอจิกเป็น "1")

โปรแกรมนี้มีการใช้ทั้งคำสั่งการอ่านรหัสลับจริง และรหัสที่เกิดจากวงจรสุม โดยนำการทำงานทั้งสองลักษณะมาประยุกต์ใช้ จากการทดสอบได้ทำการล็อกโปรแกรม CW.EXE (เรียกโปรแกรม HL_SHELL แล้วโปรแกรมจะถามชื่อโปรแกรมที่ต้องการล็อกให้ใส่ CW.EXE แล้วกด Enter) หลังจากนั้นได้เรียกใช้โปรแกรม CW.EXE สังเกตเห็นว่าการเรียกใช้เมื่อล็อกแล้วจะต้องใช้เวลาในการโหลด และเข้าสู่โปรแกรม มากกว่าก่อนล็อก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประมาณ 5 วินาที ซึ่งก็คือเวลาที่ใช้ในการตรวจสอบ และถอดรหัสของโปรแกรมที่ล็อกไว้นั่นเอง เมื่อเข้าสู่โปรแกรม CW.EXE แล้วทดลองใช้โปรแกรมดู ผลปรากฏว่าทำงานได้เหมือนโปรแกรมก่อนล็อกทุกประการ

เมื่อทดสอบการล็อกโปรแกรม และการเรียกใช้งานแล้วจะได้ข้อมูล Hex ของโปรแกรม CW.EXE ก่อนล็อกดังรูปที่ 4.8 กับ ข้อมูล Hex ของโปรแกรม CW.EXE หลังล็อกดังรูปที่ 4.9 อย่าลืมนะว่าโปรแกรมหลังล็อกเกิดจากโปรแกรม DECODE.CPP (ดูโปรแกรม HL_SHELL.CPP ในภาคผนวก ค. ประกอบ)

รูปที่ 4.8 ข้อมูลแบบ Hex บางส่วนของโปรแกรม CW.EXE ก่อนล็อก

4D	5A	F3	01	8A	00	00	00	02	00	A3	17	FF	FF	49	21
80	00	00	00	94	3F	41	C9	20	BB	14	E0	FD	EA	D3	20
27	9E	7C	E8	7C	A1	42	AB	41	6E	55	CB	5A	E6	88	C8
D6	A0	12	08	41	E0	85	A6	27	39	B1	DF	7B	52	6E	F7
C9	0F	2E	C1	C6	FD	C0	07	EB	E5	85	29	9B	FB	04	AA
C2	44	F0	27	3A	68	53	EE	A7	9D	34	81	84	B3	88	9B
71	93	C0	FE	E2	31	83	2B	54	A8	34	81	C4	1B	10	5A
EA	25	C1	47	F8	1D	78	0D	2A	76	FB	46	21	77	FF	FB
4D	0F	E6	19	98	76	BB	26	3C	B9	67	C1	4F	B7	55	EA
51	92	77	2B	B4	60	8C	42	0A	57	0E	88	94	2E	79	E7
5F	80	EE	D2	1D	7B	95	5A	37	8A	12	05	EC	F5	B0	00
09	32	D1	AE	93	DF	41	76	B5	02	97	A9	48	B4	DD	A7
C4	6B	EF	D3	71	9E	64	17	BD	DE	1E	65	DE	70	88	AD
FE	47	2E	07	64	6D	B7	E6	19	45	BF	77	32	01	60	A8
28	E3	21	49	C3	1B	73	00	11	38	30	F2	C9	88	83	31
38	39	6D	F3	84	21	58	0D	C3	8D	65	E0	4D	DF	16	7C
54	9E	3A	6B	FC	69	00	9A	49	02	70	13	1F	FF	2E	C6
06	B8	01	72	FA	A7	01	00	50	9A	3F	30	05	06	CA	B8
C6	C2	08	C2	D8	43	FD	E8	9E	C2	FD	1E	B6	44	FC	BA
3C	0F	FC	BE	44	50	EE	9A	79	F0	C3	D1	08	EC	B4	4C
8A	46	04	62	B9	FF	07	0E	00	BA	2D	00	E9	C1	00	1E
B8	00	35	F1	68	C8	43	59	D9	F1	5B	B6	04	31	E4	F3
5D	F3	5F	F3	05	F3	18	72	61	F3	63	F3	06	F3	65	0C
3F	F3	67	F3	00	25	8C	CA	8E	DA	84	F1	BE	01	EC	1F
C3	BD	25	C5	80	10	16	BF	F4	F5	04	81	23	F5	C1	F5
05	F5	70	04	C3	F5	06	F5	C5	FC	FF	F5	CB	B4	FF	8B
D7	8B	DE	3B	DF	74	14	80	FF	E1	3F	FF	74	0A	38	67
01	77	05	8A	FB	8B	D3	FF	1F	83	C3	06	EB	E8	3B	D7
74	20	8B	DA	1E	07	06	9E	FF	E3	00	C6	07	FF	47	74
07	26	FF	5F	02	43	FC	1F	EB	C5	F9	57	F9	BE	C3	B4
40	F1	D0	BB	02	B4	C3	B9	1E	3C	3B	11	FC	6F	84	E8

รูปที่ 4.8 (ต่อ)

EA	FF	B8	03	F2	F1	1A	1C	FF	C3	1F	04	C0	55	17	56
FA	C0	60	FF	AF	08	1E	FE	FD	06	9A	7B	24	EC	83	C4
06	8B	F0	3D	FF	F0	FF	FF	75	02	EB	1A	F7	46	0A	BF
74	06	F7	30	FC	C6	D2	75	04	D9	EB	09	C7	06	7D	01
8E	00	05	2A	E2	5E	5D	CB	FF	F0	C5	83	3E	B2	43	20
75	05	B8	E1	EB	1C	8B	30	FC	1E	F4	D1	E3	FE	8B	46
08	8B	56	FF	03	06	89	97	CE	B5	89	87	D0	B5	FF	06
EA	F0	F0	C7	D3	56	8B	9E	0B	F6	7C	1F	E0	05	83	FE
14	7C	09	17	7B	E8	FD	03	3C	29	EB	11	1D	CD	C7	87
54	46	1C	FB	AD	56	9A	98	81	59	A6	FD	EC	08	C6	FF
FF	46	FB	00	E8	07	1F	B4	00	89	46	FE	E8	FF	1E	B1
08	E9	1F	D3	E8	F4	FC	E9	E0	00	C4	5E	0A	FF	1E	E0
65	26	8A	07	88	DC	EC	2D	07	7F	F8	00	8B	D8	83	FB
06	77	2E	B5	2E	FF	A7	C7	FF	0F	03	B4	0E	B0	07	E8
0F	22	EB	76	A0	1C	49	E1	CF	E1	3B	C1	7D	6C	FF	4E
FE	EB	67	38	7C	F1	B2	EB	5D	C2	FC	EB	58	A0	25	7C
18	F1	0B	C0	75	2E	39	2B	49	7E	80	19	27	8A	26	20
00	8A	AF	24	70	08	FB	3C	FE	40	50	FB	FC	7F	88	FB
E8	36	3F	52	50	16	8D	ED	50	F0	FF	1B	F2	4E	3F	EB
1E	8A	56	FE	8A	76	FC	B4	E1	79	02	B7	62	AB	21	8A
1E	CC	B4	09	08	87	F2	B9	DF	E8	9A	21	A6	5B	7C	FE
A0	1E	8A	FD	12	8F	FE	B5	03	06	1A	49	9C	7F	43	A0
1F	E4	FC	7D	1C	B0	06	50	84	08	E1	FC	1D	FC	00	87
CF	FC	E7	50	B0	01	9E	A1	10	D5	16	55	FC	BC	F1	FA
08	BE	81	65	74	03	E9	13	FF	90	F8	0A	3B	90	FE	1F
0F	8B	E5	5D	C2	0A	00	E1	02	EA	02	08	03	00	F6	FF
FC	FE	F9	02	CC	F1	B8	A1	7F	21	02	50	33	C0	33	D2
50	52	31	F4	4A	0A	EC	07	B0	A1	39	B0	F5	0A	56	C7
46	FA	5D	F8	E3	A5	06	8B	56	04	89	56	FC	69	0B	3F
FC	D0	74	4C	C4	5E	FC	83	71	04	26	8B	07	26	BF	87
0B	47	02	74	3C	EB	25	EE	FE	FF	77	02	FC	F7	E1	37
9A	0F	3C	6B	F1	59	40	8B	F0	C6	FA	03	FF	E8	C6	3D
7F	00	77	18	01	76	FA	DB	CD	FE	09	77	08	26	C4	1F
40	79	E9	75	5C	04	FA	8F	2C	89	FA	9A	14	49	CE	A1
F6	3E	84	9B	F8	0B	C2	74	79	D5	F6	E7	F0	87	F6	47
FA	04	FE	26	88	07	0A	2B	EE	F8	52	7C	FC	83	C2	79
FD	EB	2C	DD	26	C6	07	28	84	20	D9	83	F8	0D	B2	F8
FD	B7	36	F6	9A	26	1F	69	EA	08	AD	FC	86	F8	13	BF
BF	FC	0D	82	C3	BF	A7	42	F1	F6	2B	92	EB	DF	DD	04
33	D2	33	C0	5E	D4	F2	04	00	AE	EC	56	BE	00	FA	D0
EF	0E	DD	0C	00	FF	31	33	F6	EB	18	0E	F8	15	03	EE

รูปที่ 4.9 ข้อมูลแบบ Hex บางส่วนของโปรแกรม CW.EXE หลังล๊อค

01	58	BB	00	00	CD	21	73	03	E9	6E	01	B4	00	CD	1A
4D	5A	50	00	3D	00	6B	00	40	00	00	00	FF	FF	3D	07
80	00	00	00	00	00	00	00	3E	00	00	00	01	00	FB	50
6A	72	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00
00	00	5E	00	00	00	A3	00	00	00	4F	01	00	00	78	03
14	04	6C	03	14	04	55	03	14	04	3D	03	14	04	2E	03
14	04	26	03	14	04	9E	02	14	04	93	02	14	04	80	02
14	04	F6	01	14	04	A8	01	14	04	96	01	14	04	81	01
14	04	6C	01	14	04	4C	01	14	04	7F	00	14	04	68	00
14	04	D2	05	14	04	A7	05	14	04	89	05	14	04	76	05
14	04	5B	05	14	04	4E	05	14	04	2F	05	14	04	27	05
14	04	E6	04	14	04	D8	04	14	04	CC	04	14	04	C0	04
14	04	A3	04	14	04	8C	04	14	04	15	04	14	04	09	04
14	04	90	0B	14	04	02	0A	14	04	CC	09	14	04	C2	09
14	04	A4	09	14	04	9C	09	14	04	91	09	14	04	86	09
14	04	7A	09	14	04	71	09	14	04	3E	08	14	04	29	0F
14	04	84	0E	14	04	CD	0D	14	04	1C	0C	14	04	04	0C
14	04	BA	0F	1D	05	B6	0F	1D	05	B2	0F	1D	05	34	16
1D	05	EE	2A	00	00	00	14	1D	05	FC	13	1D	05	F8	13
1D	05	F4	13	1D	05	F0	13	1D	05	EC	13	1D	05	E8	13
1D	05	E4	13	1D	05	E0	13	1D	05	DC	13	1D	05	D8	13
1D	05	D4	13	1D	05	D0	13	1D	05	CC	13	1D	05	C8	13
1D	05	C4	13	1D	05	C0	13	1D	05	BC	13	1D	05	B8	13
1D	05	B4	13	1D	05	B0	13	1D	05	AC	13	1D	05	A8	13
1D	05	A4	13	1D	05	A0	13	1D	05	9C	13	1D	05	98	13
1D	05	94	13	1D	05	90	13	1D	05	8C	13	1D	05	88	13
1D	05	84	13	1D	05	80	13	1D	05	7C	13	1D	05	78	13
1D	05	74	13	1D	05	70	13	1D	05	6C	13	1D	05	68	13
1D	05	64	13	1D	05	60	13	1D	05	5C	13	1D	05	58	13
1D	05	54	13	1D	05	50	13	1D	05	4C	13	1D	05	48	13
1D	05	44	13	1D	05	86	3C	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

บทที่ 5

สรุปผลการวิจัย และข้อเสนอแนะ

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์ที่จะออกแบบวงจรรวมป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM โดยประยุกต์ทดลองทำเป็นอุปกรณ์ป้องกันการคัดลอกโปรแกรม .EXE หรือ .COM ซึ่งเรียกอุปกรณ์นี้ว่า ฮาร์ดล๊อค โดยฮาร์ดล๊อคที่สร้างเป็นต้นแบบใช้เทคโนโลยีของ FPGA ในการออกแบบวงจรรวมและจำลองการทำงาน ใช้ซอฟต์แวร์ ViewLogic หลังจากนั้นเปลี่ยนแผงวงจรให้อยู่ในรูปของ LCA (Logic Cell Array) เพื่อแปลงเป็นโปรแกรมคอนฟิกร์บนที่กลอง PROM XC1736 อีกทีหนึ่งโดยใช้ซอฟต์แวร์ Xilinx ซึ่งโปรแกรมคอนฟิกร์นี้จะใช้โหลดให้กับ FPGA XC3042A ทำให้ LCA ภายในเชื่อมต่อกันมีลักษณะการทำงานเหมือนกับวงจรที่ออกแบบเป็นแผงวงจรไฟฟ้า การใช้งานฮาร์ดล๊อคสามารถต่อเชื่อมกับไมโครคอมพิวเตอร์ที่พอร์ตขนานโดยไม่จำเป็นต้องใช้แหล่งจ่ายไฟจากภายนอกทำให้มีความสะดวกในการใช้งาน

การออกแบบวงจรรวมในครั้งนี้ได้ออกแบบเป็นแผงวงจรไฟฟ้าซึ่งใช้เวลาในการออกแบบค่อนข้างมาก และยังเกิดความผิดพลาดจากผู้ออกแบบที่เกิดจากความยุ่งยากของวงจรได้อีก ส่วนวิธีการแก้ปัญหาในเรื่องนี้สามารถใช้วิธีการออกแบบวงจรโดยเขียนบรรยายเชิงพฤติกรรมของวงจรนั้นๆได้ด้วยภาษา VHDL ทำให้สามารถช่วยลดระยะเวลาในการออกแบบได้มาก

ในโปรแกรมตัวอย่างที่ให้มาในภาคผนวก ค. โดยเฉพาะอย่างยิ่งโปรแกรม HL_SHELL ที่ใช้ป้องกันการคัดลอกโปรแกรม .EXE หรือ .COM ที่ให้ไว้ นั้น ยังใช้ประสิทธิภาพของฮาร์ดล๊อคที่ออกแบบไว้ไม่เต็มที่ คือ ใช้คำสั่งอ่านรหัสลับ และอ่านรหัสหลุดจากวงจรลุ่ม เท่านั้น แต่ถ้าโปรแกรมนี้ไปดัดแปลงแก้ไขให้ใช้ประสิทธิภาพของฮาร์ดล๊อคได้อย่างเต็มที่ เช่น โดยการใช้คำสั่งลบข้อมูลเพื่อเปลี่ยนแปลงรหัสลับ หรือรหัสชี้เฉพาะ และใช้คำสั่งลบ อ่าน และเขียนทำงานร่วมกันเพื่อทำการย้ายบล็อกของหน่วยความจำภายในฮาร์ดล๊อค จะทำให้ประสิทธิภาพในการป้องกันการคัดลอกโปรแกรมมีสูงขึ้นมากกว่าเดิมอีก ส่วนของโปรแกรมที่เขียนขึ้นผู้วิจัยเขียนให้สามารถทำความเข้าใจได้ง่าย ไม่ซับซ้อน และเพื่อให้เหมาะสมที่จะนำไปพัฒนาต่อได้

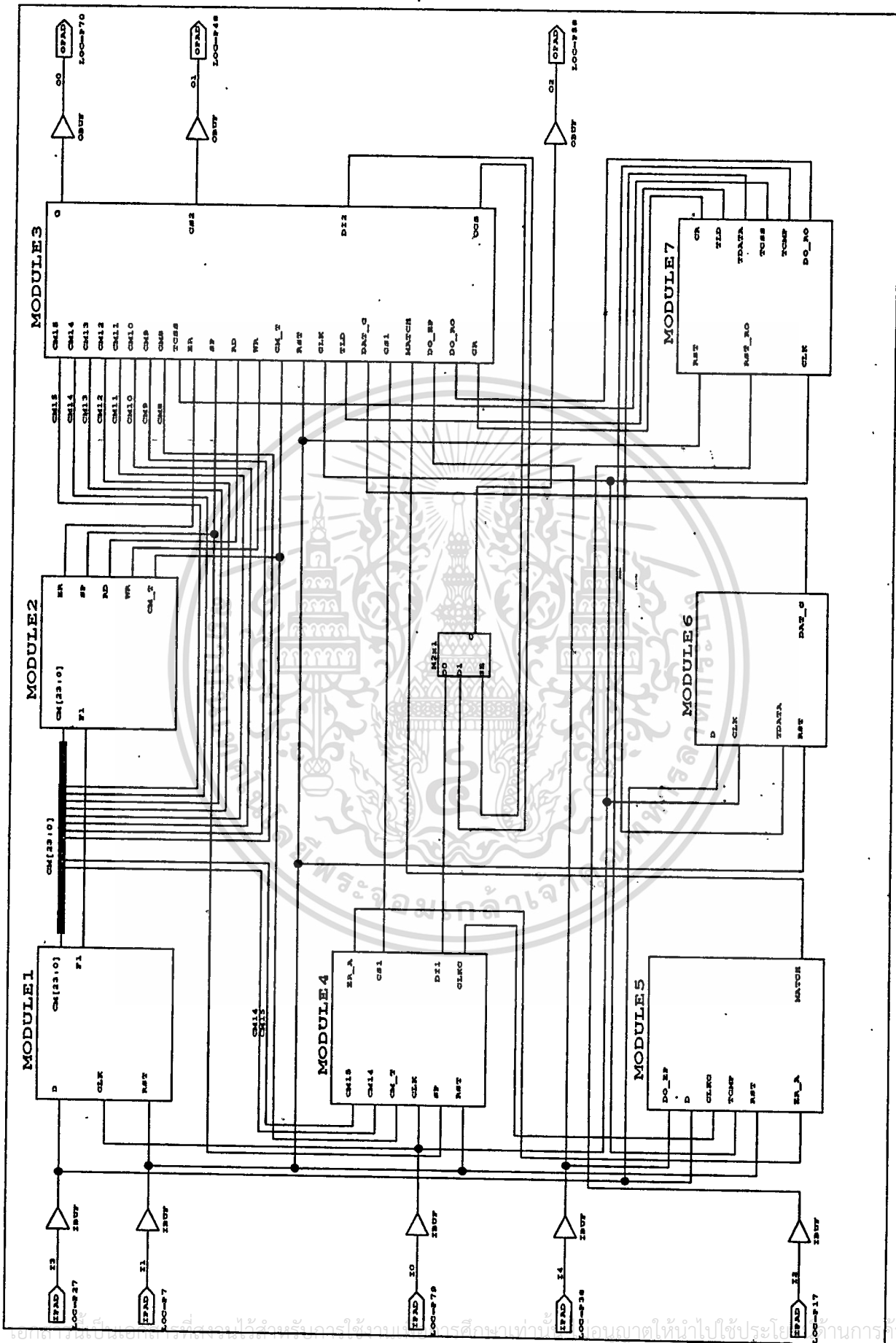
ในการออกแบบฮาร์ดล๊อคได้ออกแบบให้วงจรมีขนาดเล็กที่สุด โดยใช้การทำงานของซอฟต์แวร์เข้ามาช่วย ซึ่งจะทำให้ประสิทธิภาพในการป้องกันการคัดลอกสูงมากขึ้น

เอกสารอ้างอิง

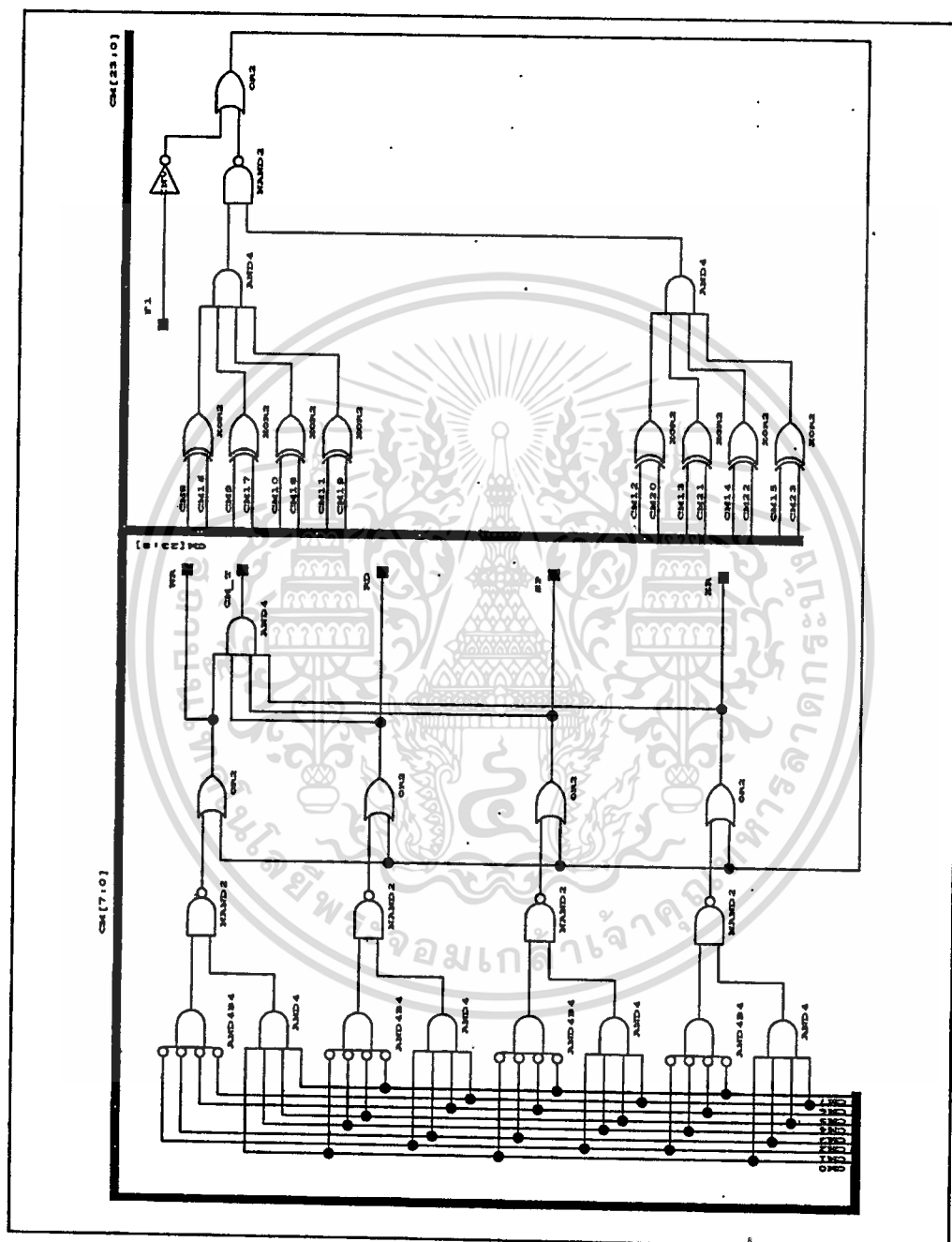
- [1] Dallas Semiconductor, "1992-1993 Production Data Book," Users-Insertable Memory (Secured).
- [2] Charles P. Pfleeger, "Security in computing," University of Tennessee and Trusted Information System, Inc., U.S.A., 1989.
- [3] Mentor Graphics, Co., Ltd., "Getting Started with Design Architect Training Workbook," 1993.
- [4] Mentor Graphics, Co., Ltd., "Getting Started with QuickSim II Training Workbook," 1993.
- [5] Mentor Graphics, Co., Ltd., "An Introduction to Modeling with VHDL," November, 1991.
- [6] Mentor Graphics, Co., Ltd., "Autologic VHDL Reference Manual," 1993.
- [7] Mentor Graphics, Co., Ltd., "MISL User's and Reference Manual," September, 1991.
- [8] Xilinx, Co., Ltd., "User Guide and Tutorials," 1993.
- [9] Xilinx, Co., Ltd., "XACT Development System Reference Guide," January, 1993.
- [10] Xilinx, Co., Ltd., "The Programmable Logic Data Book," 1993.
- [11] Hitachi, Co., Ltd., "HG62E Series Design Manual (High Speed CMOS Gate Array)," Japan., December, 1987.
- [12] New York International Business Machines, Co., Ltd., "Data Security through Cryptography," U.S.A., 1986.
- [13] James Arlin Cooper, "Computer & Communications Security," McGRAW-HILL, 1989.
- [14] Martin R. Smith, "Commonsense Computer Security," McGRAW-HILL, 1989.
- [15] Don Lancaster, "CMOS Cookbook," U.S.A., First Edition, 1977.
- [16] Solomon W. Golomb, "Shift Register Sequences," U.S.A., 1967.
- [17] การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 15 "แอตทิฟฮาร์ดล็คควบคุมการทำงานโดยไมโครคอนโทรลเลอร์แบบชิพเดี่ยวขนาด 4 บิตเบอร์ PD7554 และ การป้องกันโปรแกรมคอมพิวเตอร์โดยใช้ฮาร์ดล็ค" หน้า 5-1 ถึง 5-10 ธันวาคม พ.ศ. 2535.
- [18] บุญวัฒน์ อัฐ และธีรศักดิ์ชัย อังสกุล "การออกแบบวงจรรวมเฉพาะกิจเพื่อช่วยป้องกันการเข้าถึงข้อมูลในหน่วยความจำ EEPROM" วารสารวิศวกรรมสถานแห่งประเทศไทย ฉบับเทคโนโลยี หน้า 65-70 กุมภาพันธ์ พ.ศ. 2538.



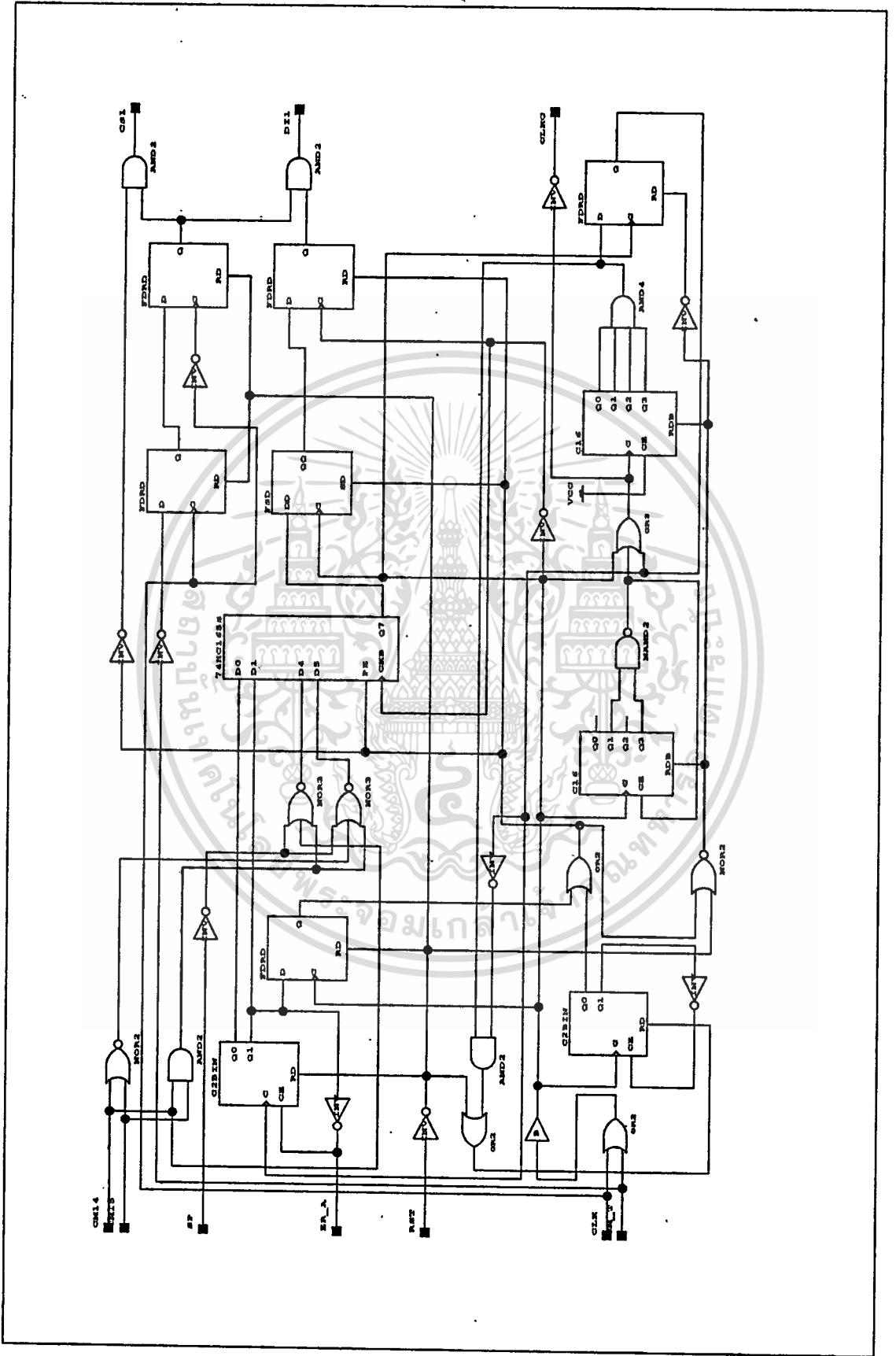
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เลือกแสดงการเชื่อมต่อของโมดูล

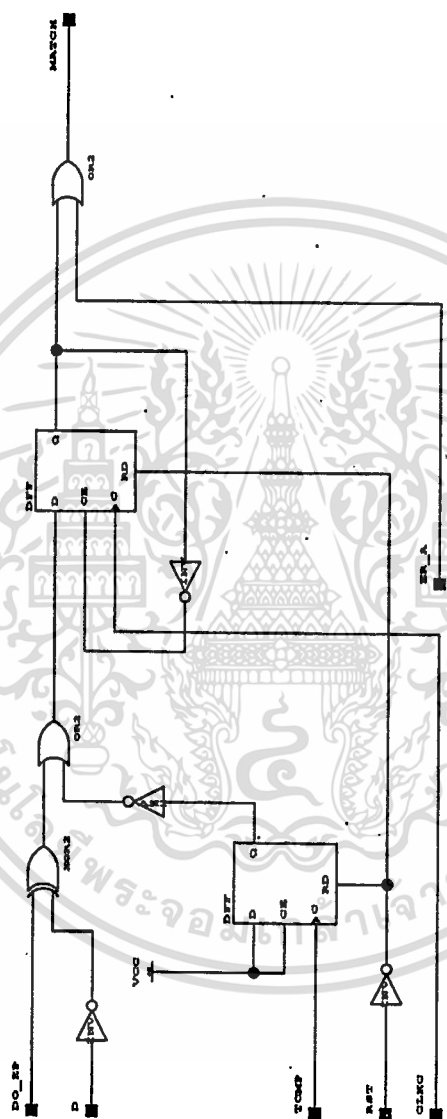


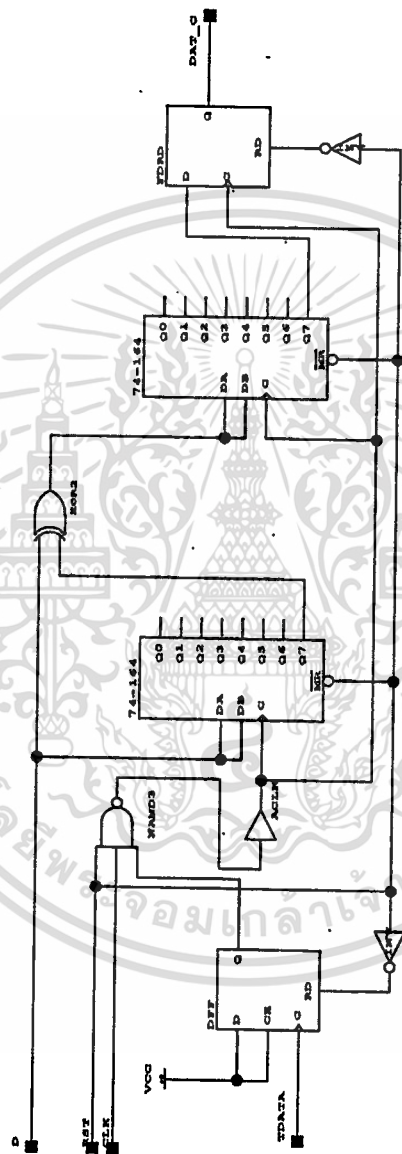
MODULE 2

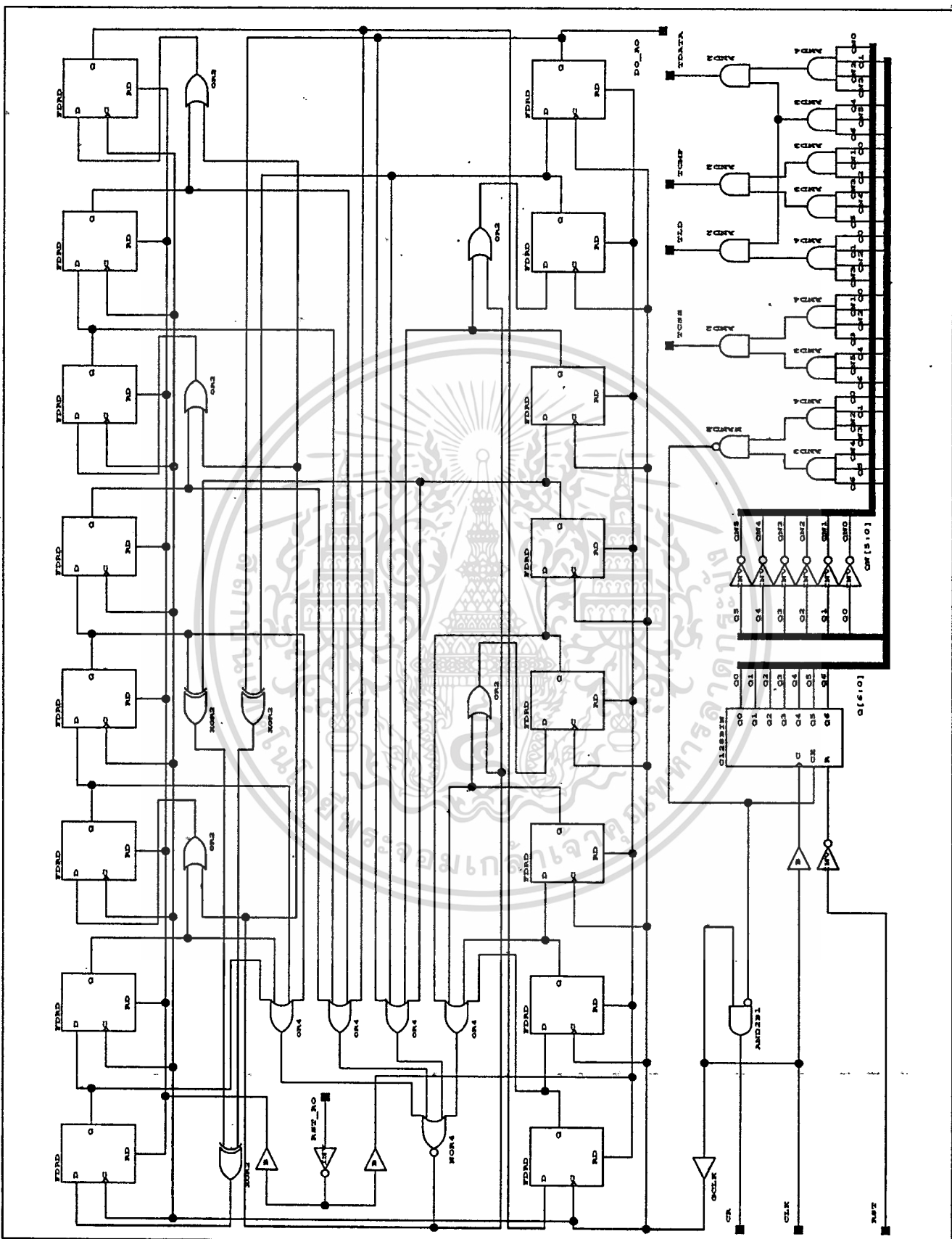


วงจร MODULE 4

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้







เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โปรแกรมติดตั้งสถานะเริ่มต้นให้ฮาร์ดล๊อค INSTALLH.CPP

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <dos.h>
main()
{
FILE *fp0,*fp1;
int ij=0,handle;
char search[250],*ptr,*insted="C:\\HARDLOCK;\n",*path="PATH=C:\\",*FullPath;
char olds[201],pp[220]="";

clrscr();
handle=creatnew("c:\\auto.bat",0);
close(handle);
system("xcopy . c:\\s");
fp0=fopen("c:\\autoexec.bat","rt");
fp1=fopen("c:\\auto.bat","r+t");
do
    { j++;fgets(search,200,fp0);
      ptr = strstr(search,insted);
      if (strcmp(ptr,insted)==0) {goto endpp;};
    }while (j!=100);
fseek(fp0,0,SEEK_SET);
do
    { j++;fgets(search,200,fp0);
      ptr = strstr(search,path);
      if (strcmp(ptr,search)==0)
      {
          strncat(pp,search,strlen(search)-1);
          strcat(pp,insted);strcat(pp,"C:\\HARDLOCK\\POWER.EXE\n");
          strcpy(search,pp);
      }
      if (strcmp(search,olds)==0) {goto endp;};
      fputs(search,fp1);
      strcpy(olds,search);
    } while (j!=100);
endp:fclose(fp0);fclose(fp1);

rename("c:\\autoexec.bat","c:\\autoexec.old");
system("del c:\\autoexec.bat");
rename("c:\\auto.bat","c:\\autoexec.bat");
system("del c:\\installh.exe");
system("c:\\autoexec");
system("cls");system("c:\\");
endpp:fclose(fp0);fclose(fp1);
}
```

; ตรวจสอบว่าแบดไฟล์ถูกแก้ไขแล้วหรือไม่
; โดยตรวจสอบว่าข้อความ C:\HARDLOCK
; เพิ่มข้อความ C:\HARDLOCK; และ
; C:\HARDLOCK\POWER.EXE
; ลงไปในแบดไฟล์ต่อท้ายข้อความที่กำหนด
; PATH
; คำสั่ง DOS

โปรแกรม POWER.ASM

```
code_seg    segment
assume      cs:code_seg
org         100h
power       proc near
            mov dx,0378h
            mov ax,00F0h
            out dx,ax                ; ส่ง "1" ออกไปที่พอร์ตสัญญาณ D4-D7 เพื่อสร้างเป็นแหล่งจ่ายไฟ
            int 21h
            int 20h
            endp
code_seg    ends
end power
```



โปรแกรมเขียนรหัสเฉพาะ ID_GEN.CPP

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

char *hex2bini(char *,char *);
char *hex2bin(char *, char *);
void delay();
void feed_data(char *,char *,char *);
int p=3*16*16+7*16+8;
main()
{
FILE *fp0,*fp1;
int i,j,k,p;
char code_sp [17]="0000000000000000";
char secu_id0[5][17],secu_id1[5][17];
char code_dm[5],code[17],password[5][50];
char secu_d0[5][5],secu_d1[5][5],Title[8],sp[2];
char c_ew_en[40]="1000011111110000",c_ew_ds[40]="10011111111000000";
char func_wr[ 9]="10011001",func_er[9]="01010101",func_pv[9]="01011010";
char d_commd[57][40];
char command[57][40]=
{"10011111111000000","10011111011000001",
"10010111111010000","10010111011010001",
"10001111111100000","10001111011100001",
"10000111111100000","10000111011110001"};
clrscr();
fp0=fopen("C:\\HARDLOCK\\STANDARD.INI","r"); ; เปิดไฟล์ที่เก็บรหัสผ่าน .
fp1=fopen("DATA_WR","r"); ; เปิดไฟล์ที่ใช้เก็บรหัสที่จะเขียนลงฮาร์ดดิสก์
fscanf(fp0,"%s",Title);fgets(sp,2,fp0);
for (i=0;i<4;i++)
{
fgets(secu_d0[i],5,fp0);hex2bini(secu_d0[i],secu_id0[i]); ; อ่านรหัสผ่านจากไฟล์ fp0
fgets(secu_d1[i],5,fp0);hex2bini(secu_d1[i],secu_id1[i]);
}
strcpy(password[0],secu_id0[0]);strcat(password[0],secu_id1[0]); ; สร้างรหัสผ่านให้เข้าถึงข้อมูลได้ทุกแอดเดรส
strcat(c_ew_en,func_pv);
feed_data(c_ew_en,password[0],code_sp); ; ป้อนคำสั่งเพื่อให้เขียนข้อมูลได้ (อีนเนเบิล)
for (j=0;j<8;j++)
{
fscanf(fp1,"%s",code_dm);hex2bin(code_dm,code);
strcpy(d_commd[j],command[j]);
strcat(d_commd[j],func_er);
strcat(command[j],func_wr);
feed_data(command[j],password[0],code); ; ป้อนคำสั่งเพื่อเขียนรหัสเฉพาะ
}
strcat(c_ew_ds,func_pv);
feed_data(c_ew_ds,password[0],code_sp); ; ป้อนคำสั่งเพื่อไม่ให้เขียนข้อมูลได้
fclose(fp0);fclose(fp1);
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

char *hex2bini(char *hex, char *bin_inv)

; เปลี่ยนเลขฐาน 16 เป็นคอมพลีเมนต์เลขฐาน

สอง

```
{
    unsigned int v,dum;
    int len, lcnt,digit=16;
    char h[17]="";
    sscanf(hex, "%x", &v);
    dum=65535-v;
    itoa(dum, bin_inv, 2);
    len = strlen(bin_inv);
    if ( len < digit )
    {
        lcnt = digit - len;
        while (lcnt-- > 0) strcat(h, "0");
        strcat(h, bin_inv); strcpy(bin_inv, h);
    }
    return(bin_inv);
}
```

char *hex2bin(char *hexb, char *bin)

; เปลี่ยนเลขฐาน 16 เป็นเลขฐานสอง

```
{
    unsigned int v;
    int lenb, lcntb,digitb=16;
    char hb[17]="";
    sscanf(hexb, "%x", &v);
    itoa(v, bin, 2);
    lenb = strlen(bin);
    if ( lenb < digitb )
    {
        lcntb = digitb - lenb;
        while (lcntb-- > 0) strcat(hb, "0");
        strcat(hb, bin); strcpy(bin, hb);
    }
    return(bin);
}
```

void feed_data(char *commd,char *pwd,char *dat)

; ป้อนคำสั่งทั้งหมดทีละบิต

```
{
    int j;char add_sig[27]="00000000000000000000000000000000";
    for (j=0;j<25;j++)
    {
        outport(p,240+6+(commd[j]-'0')*8);
        delay();outport(p,240+7+(commd[j]-'0')*8);
    }
    for (j=0;j<32;j++)
    {
        outport(p,240+6+(pwd[j]-'0')*8);
        delay();outport(p,240+7+(pwd[j]-'0')*8);
    }
    for (j=0;j<16;j++)
    {
        outport(p,240+6+(dat[j]-'0')*8);
        delay();outport(p,240+7+(dat[j]-'0')*8);
    }
}
```

void delay()

; หน่วงเวลาเพื่อเขียนข้อมูลลง EEPROM

```
{int i; for(i=0;i<1000;i++){;}}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปเผยแพร่โดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โปรแกรม CODE_GEN.CPP (ส่วนของฟังก์ชันที่เหมือนโปรแกรมข้างบนจะไม่แสดงไว้)

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

char *hex2bini(char *,char *);
char *hex2bin(char *, char *);
void delay();
void feed_data(char *,char *,char *);
int p=3*16*16+7*16+8;
main()
{
FILE *fp0,*fp1;
int i,j,k,p;
char code_sp [17]="0000000000000000";
char secu_id0[5][17],secu_id1[5][17];
char code_dm[5],code[17],password[5][50];
char secu_d0[5][5],secu_d1[5][5],Title[8],sp[2];
char c_ew_en[40]="10000111111110000",c_ew_ds[40]="10011111111000000";
char func_wr[ 9]="10011001",func_er[9]="01010101",func_pv[9]="01011010";
char d_commd[57][40];
char command[57][40]=
{"10011110111000010","10011110011000011","10011101111000100","10011101011000101",
"10011100111000110","10011100011000111","10011011111001000","10011011011001001",
"10011010111001010","10011010011001011","10011001111001100","10011001011001101",
"10011000111001110","10011000011001111","11110110100010010","11110110000010011",
"11110101100010100","11110101000010101","11110100100010110","11110100000010111",
"11110011100011000","11110011000011001","11110010100011010","11110010000011011",
"11110001100011100","11110001000011101","11110000100011110","11110000000011111",
"11001110101100010","11001110001100011","11001101101100100","11001101001100101",
"11001100101100110","11001100001100111","11001011101101000","11001011001101001",
"11001010101101010","11001010001101011","11001001101101100","11001001001101101",
"11001000101101110","11001000001101111","10100110110110010","10100110010110011",
"10100101110110100","10100101010110101","10100100110110110","10100100010110111",
"10100011110111000","10100011010111001","10100010110111010","10100010010111011",
"10100001110111100","10100001010111101","10100000110111110","10100000010111111"};
clrscr();
fp0=fopen("C:\\HARDLOCK\\STANDARD.INI","r"); ; เปิดไฟล์ที่เก็บรหัสผ่าน
fp1=fopen("DATA_WR","r"); ; เปิดไฟล์ที่ใช้เก็บรหัสที่จะเขียนลงฮาร์ดล็อก
fscanf(fp0,"%s",Title);fgets(sp,2,fp0);
for (i=0;i<4;i++)
{
fgets(secu_d0[i],5,fp0);hex2bini(secu_d0[i],secu_id0[i]); ; อ่านรหัสผ่านจากไฟล์ fp0
fgets(secu_d1[i],5,fp0);hex2bini(secu_d1[i],secu_id1[i]);
}
strcpy(password[0],secu_id0[0]);strcat(password[0],secu_id1[0]); ; สร้างรหัสผ่านเข้าหน่วยความจำทั้ง 4 ส่วน
strcpy(password[1],secu_id0[1]);strcat(password[1],secu_id1[1]);
strcpy(password[2],secu_id0[2]);strcat(password[2],secu_id1[2]);
strcpy(password[3],secu_id0[3]);strcat(password[3],secu_id1[3]);

strcat(c_ew_en,func_pv);strcat(c_ew_en,nocare10);
feed_data(c_ew_en,password[0],code_sp);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้เฉพาะเพื่อการศึกษเท่านั้น ไม่อนุญาตให้ทำซ้ำโดยไม่ขออนุญาต

ไม่ว่ากรณีใดทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

for (j=0;j<56;j++)
{
    fscanf (fp1,"%s",code_dm);hex2bin(code_dm,code);
    strcpy(d_commd[j],command[j]);
    strcat(d_commd[j],func_er);
    strcat(command[j],func_wr);
    feed_data(d_commd[j],password[j/14],code_sp);
    feed_data(command[j],password[j/14],code);
}
strcat(c_ew_ds,func_pv);
feed_data(c_ew_ds,password[0],code_sp);
fclose(fp0);fclose(fp1);
}

```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

char *hex2bini (char *,char *);
char *bin2hex (char *bin, char *hex);
void feed_data (char *,char *,char *);
char *decode(char *enc,char *dec);
char *read_data(char *di_hex);
int p=3*16*16+7*16+8;

main()
{
FILE *fp0,*fp1;
int i,j,k;
char code_sp [17]="0000000000000000",datai[17],data_rd[17];
char secu_id0[5][17],secu_id1[5][17],Title[8],sp[2];
char code_dm[5],code[17],password[5][50];
char secu_d0[5][5],secu_d1[5][5];
char func_rd[9]="01100110";
char command[57][40]=
{"1001110111000010","1001110011000011","1001101111000100","1001101011000101",
"10011100111000110","10011100011000111","10011011111001000","10011011011001001",
"10011010111001010","10011010011001011","10011001111001100","10011001011001101",
"10011000111001110","10011000011001111","11110110100010010","11110110000010011",
"11110101100010100","11110101000010101","11110100100010110","11110100000010111",
"11110011100011000","11110011000011001","11110010100011010","11110010000011011",
"11110001100011100","11110001000011101","11110000100011110","11110000000011111",
"11001110101100010","11001110001100011","11001101101100100","11001101001100101",
"11001100101100110","11001100001100111","11001011101101000","11001011001101001",
"11001010101101010","11001010001101011","11001001101101100","11001001001101101",
"11001000101101110","11001000001101111","10100110110110010","10100110010110011",
"10100101110110100","10100101010110101","10100100110110110","10100100010110111",
"10100011110111000","10100011010111001","10100010110111010","10100010010111011",
"10100001110111100","10100001010111101","10100000110111110","10100000010111111"};

clrscr();
fp0=fopen("C:\\HARDLOCK\\STANDARD.INT","r"); ; เปิดไฟล์ที่เก็บรหัสผ่าน
fp1=fopen("DATA_RD","w"); ; เปิดไฟล์ที่ใช้เก็บรหัสลับที่อ่านได้
fscanf(fp0,"%s",Title);fgets(sp,2,fp0);

for (i=0;i<4;i++)
{
fgets(secu_d0[i],5,fp0);hex2bini(secu_d0[i],secu_id0[i]); ; อ่านรหัสผ่านจากไฟล์ fp0
fgets(secu_d1[i],5,fp0);hex2bini(secu_d1[i],secu_id1[i]);
}

strcpy(password[0],secu_id0[0]);strcat(password[0],secu_id1[0]); ; สร้างรหัสผ่านเข้าหน่วยความจำทั้ง 4 ส่วน
strcpy(password[1],secu_id0[1]);strcat(password[1],secu_id1[1]);
strcpy(password[2],secu_id0[2]);strcat(password[2],secu_id1[2]);
strcpy(password[3],secu_id0[3]);strcat(password[3],secu_id1[3]);

```

```

for (j=0;j<56;j++)
{
    outport(p,240+0);
    strcat(command[j],func_rd);
    feed_data(command[j],password[j/14],code_sp);
    read_data(datai);
    fprintf(fp1,"%s ",datai);
    if (j==13 || j==27 || j==41)
    {
        fprintf(fp1,"%s","\n");
    }
}
fclose(fp0);fclose(fp1);
}

void feed_data(char *commd,char *pwd,char *dat)
{
    int j;
    char add_sig[27]="00000000000000000000000000000000";
    for (j=0;j<25;j++)
    {
        outport(p,240+6+(commd[j]-'0')*8);
        outport(p,240+7+(commd[j]-'0')*8);
    }
    for (j=0;j<32;j++)
    {
        outport(p,240+6+(pwd[j]-'0')*8);
        outport(p,240+7+(pwd[j]-'0')*8);
    }
    for (j=0;j<16;j++)
    {
        outport(p,240+6+(dat[j]-'0')*8);
        outport(p,240+7+(dat[j]-'0')*8);
    }
}

```

; สร้างคำสั่งที่ใช้อ่านรหัสลับ
 ; ป้อนคำสั่งที่ใช้อ่านรหัสลับให้ฮาร์ดลิค
 ; อ่านรหัสลับ 16 บิต
 ; เขียนรหัสที่อ่านได้ลงไฟล์ DATA_RD
 ; ป้อนคำสั่งทั้งหมดโดยไม่มีกรท่วง

โปรแกรม RAND_RD.CPP

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

char *hex2bin(char *,char *);
char *bin2hex(char *,char *);
char *decode(char *,char *);
char *read_dat(char *);
void feed_dat(char *,char *,char *);
int p=3*16*16+7*16+8;
main()
{
FILE *fp0,*fp1;
int i,j,l,n;
char password[45]="11101100101111010101111001001001";
char code_dm[5][5],code[5][17],secu_d0[5][5],secu_d1[5][5];
char datai[5],hex_dec[5],datac[5],bin_enc[17],datab[17];
char func_rd[9]="01100110";
char data[17]="0000000000000000";
char command[40]="10011101001100101"; ; เป็นคำสั่งที่ผิด
clrscr();
fp0=fopen("C:\\HARDLOCK\\STANDARD.INT","r");
fp1=fopen("DATA_RM","w");
outport(p,240+0);n=0;
for (j=0;j<686;j++)
{
    strcat(command,func_rd);
    if (j>0) outport(p,240+4);
    feed_dat(command,password,data);
    read_dat(datai);
    hex2bin(datai,datab);
    decode(datab,datac);
    fprintf(fp1,"%s ",datac);n++;
    if (n==14)
    {
        n=0;fprintf(fp1,"%s","\n");
    }
}
fclose(fp0);fclose(fp1);
}
```

โปรแกรม HL_SHELL.CPP

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <iostream>
#include <fcntl.h>
#include <string.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>
int p=3*16*16+7*16+8;
char secu_id0[5][17],secu_id1[5][17],curdir[MAXPATH];
char secu_d0 [5][5 ],secu_d1 [5][5 ];
char code_dm[5],code[17],password[5][50];
char code_sp [17]="0000000000000000",func_rd[9]="01100110";
char *hex2bini(char *hex, char *bin_inv);
char *bin2hex(char *bin, char *hex);
char *read_data(char *di_hex);
char *decode(char *enc,char *dec);
char *current_directory(char *path);
long coding(long numberB,long Coded_D);
void ch_passw(void);
void cr_dt_ee(long);
void feed_data(char *,char *,char *);
int main(void)
{
FILE *temp,*TPF,*EXF,*History;
struct fblk fblk;
long PRcs_ip,FiSizeS,FiSizeA,FiSizeN,ParaB,i,Cdum,Coded_D;
int HPLo,HPHi,done,pf,handhist,handle,ptr,endf,handle1,ih=0,HistN=0;
unsigned cdum1;
char ch[13],ch1[13],ExeFile[128],column1[13],non[13],N_Tfile[3];
char HistPath[128]="",*ExeFileU;
char HistNS[4],HistFile[128]="",ret[1]="\n",HistOld[128]="",HistNew[128]="";
char TempFile[23]="C:\\HARDLOCK\\TEMP\\TEMP.";
char DelTemp[140]="DEL ",DelExe[140]="DEL ",CopyExe[140]="COPY ";
char CopyDec[140]="COPY C:\\HARDLOCK\\DECODE.EXE ";

clrscr();ch_passw(); ; ตรวจสอบว่ามีฮาร์ดล็อกติดตั้งอยู่หรือไม่
printf("Enter file locked -> ");gets(ExeFile);
system("mkdir \\HARDLOCK\\TEMP");clrscr();
handhist=creatnew("C:\\HARDLOCK\\HISTORY.INI",0);
close(handhist);
curdir[MAXPATH]; ; รับไดเรกทอรีที่อยู่ของไฟล์ที่ต้องการล็อก

current_directory(curdir);

ExeFileU=strupr(ExeFile);
strcpy(HistPath,curdir);strcat(HistPath,"\\");
strcat(HistPath,ExeFileU);
History=fopen("C:\\HARDLOCK\\HISTORY.INI","rt+");
fseek (History,0,SEEK_SET); ; ตรวจสอบจำนวนโปรแกรมที่ล็อกไว้เพื่อ
do { strcpy(HistOld,HistFile); ; ทาลำดับที่ใช้สร้างโปรแกรมสำรอง
fgets(HistFile,128,History);
```

เอกสารนี้เป็นเอกสารที่จัดทำขึ้นเพื่อใช้ในการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

strcpy(HistNew,HistFile);
if (strcmp(HistOld,HistNew)==0) break;
HistN++;
} while (HistN!=61);
HistN=HistN/2;
fseek (History,0,SEEK_SET);
do { ih++; ; ตรวจสอบว่าโปรแกรมได้ถูกล็อกแล้วหรือไม่
    fgets(HistFile,128,History);
    if (strcmp(HistFile,HistPath)==0)
    {
        printf("File has already locked");
        goto endp;
    }
} while (ih!=61);
if (ih==61)
{ fseek (History,0,SEEK_END);
  fwrite(&HistPath,128,1,History); ; ถ้าโปรแกรมนั้นไม่ได้ล็อกจะบันทึกชื่อโปรแกรม
  fwrite(&ret,1,1,History); ; ลงในไฟล์ C:\HARDLOCK\HISTORY.INI
}
fclose(History);
itoa(HistN+1,HistNS,10);
strcat(TempFile,HistNS);
strcpy(ExeFile,HistPath);
strcat(CopyExe,ExeFile);strcat(CopyExe," ");
strcat(CopyExe,TempFile);strcat(DelTemp,TempFile); ; สำรองโปรแกรมที่ถูกล็อกไว้ที่
system(DelTemp);system(CopyExe); ; C:\HARDLOCK\TEMP\TEMP.ลำดับที่ล็อก
TPF=fopen(TempFile,"rb+");EXF=fopen(ExeFile,"rb");
fseek(EXF,0,SEEK_END);FiSizeS=ftell(EXF);
for (i=0;i<(FiSizeS);i++) ; เลือกไบท์ที่จะเข้ารหัสตลอดโปรแกรม
{
    if ((i>19 && i<256) || (i>255 && (i%256==0) || (i%257==0) || (i%258==0)))
    { if (il=0)
        {
            fseek(EXF,i,SEEK_SET);
            Cdum=fgetc(EXF); ; อ่านข้อมูล Hex File ของโปรแกรมที่ถูกล็อก
            if (i<(FiSizeS/2))
            {
                Coded_D=coding((i%56),Cdum); ; เข้ารหัสข้อมูลที่ละไบท์
            }
            else
            {
                Coded_D=coding(56,Cdum);
            }
            fseek(TPF,i,SEEK_SET);
            fwrite(&Coded_D,1,1,TPF);
        }
    }
}
fclose(EXF);fclose(TPF);
strcat(DelExe,HistPath);system(DelExe);clrscr();
strcat(CopyDec,HistPath);system(CopyDec);clrscr();
endp:fclose(History);return(0);
}

```

```

char *current_directory(char *path)                                ; หาไดเรกทอรีปัจจุบันของโปรแกรมที่จะล็อก
{
    strcpy(path, "X:\\");
    path[0] = 'A' + getdisk();
    getcurdir(0, path+3);
    return(path);
}

void ch_passw(void)
{
    FILE *fid;
    int il,iid,equ0,equ1;
    char Title[8],sp[2];
    char id_datac[5],id_datad[9][5];
    char cdc_id[9][40] =
    {"10011111111000000","10011111011000001",
     "10010111111010000","10010111011010001",
     "10001111111100000","10001111011100001",
     "10000111111100000","10000111011110001"};
    fid=fopen("C:\\HARDLOCK\\STANDARD.INT","r");
    fscanf(fid,"%s",Title);fgets(sp,2,fid);
    for (il=0;il<4;il++)
    {
        fgets(secu_d0[il],5,fid);hex2bini(secu_d0[il],secu_id0[il]);
        fgets(secu_d1[il],5,fid);hex2bini(secu_d1[il],secu_id1[il]);
    }
    strcpy(password[0],secu_id0[0]); strcat(password[0],secu_id1[0]);
    strcpy(password[1],secu_id0[1]); strcat(password[1],secu_id1[1]);
    strcpy(password[2],secu_id0[2]); strcat(password[2],secu_id1[2]);
    strcpy(password[3],secu_id0[3]); strcat(password[3],secu_id1[3]);
    fclose(fid);
    for (iid=0;iid<8;iid++)                                ; อ่านรหัสเฉพาะจากหน่วยความจำ
    {
        outport(p,240+0);
        strcat(cdc_id[iid],func_rd);
        feed_data(cdc_id[iid],password[0],code_sp);
        read_data(id_datac);decode(id_datac,id_datad[iid]);
    }
    for (iid=0;iid<4;iid++)                                ; ตรวจสอบรหัสเฉพาะจากหน่วยความจำ
    {
                                                ; กับรหัสผ่านที่อ่านจากไฟล์ fid
        equ0=strcmp(secu_d0[iid],id_datad[iid*2]);
        equ1=strcmp(secu_d1[iid],id_datad[(iid*2)+1]);
        if (equ0!=0 || equ1!=0)
        {
            clrscr();
            printf("Hardlock doesn't install....\n");
            printf("Please install and re-execute\n");
            sound(900);delay(200);nosound();getchar();exit(0);
        }
    }
}

```

```
void cr_dt_ee(long numberA)
```

; เลือกคำสั่งที่จะป้อนให้ฮาร์ดลอจิกมี 56 คำสั่ง

```
{
```

; จมิม กับอีก 1 คำสั่งหลอก

```
char command[58][40]=
```

```
{"10011110111000010","10011110011000011","10011101111000100","10011101011000101",  
"10011100111000110","10011100011000111","10011011111001000","10011011011001001",  
"10011010111001010","10011010011001011","10011001111001100","10011001011001101",  
"10011000111001110","10011000011001111","11110110100010010","11110110000010011",  
"11110101100010100","11110101000010101","11110100100010110","11110100000010111",  
"11110011100011000","11110011000011001","11110010100011010","11110010000011011",  
"11110001100011100","11110001000011101","11110000100011110","11110000000011111",  
"11001110101100010","11001110001100011","11001101101100100","11001101001100101",  
"11001100101100110","11001100001100111","11001011101101000","11001011001101001",  
"11001010101101010","11001010001101011","11001001101101100","11001001001101101",  
"11001000101101110","11001000001101111","10100110110110010","10100110010110011",  
"10100101110110100","10100101010110101","10100100110110110","10100100010110111",  
"10100011110111000","10100011010111001","10100010110111010","10100010010111011",  
"10100001110111100","10100001010111101","10100000110111110","10100000010111111",  
"10011101001100101"};
```

```
if (numberA!=56)
```

```
{
```

```
    output(p,240+0);
```

```
    strcat(command[numberA],func_rd);
```

```
    feed_data(command[numberA],password[numberA/14],code_sp);
```

```
}
```

```
else
```

```
{
```

```
    output(p,240+4);
```

```
    strcat(command[56],func_rd);
```

```
    feed_data(command[56],password[0],code_sp);
```

```
}
```

```
long coding(long numberB,long Coded_D)
```

; เก็บรหัสไฟล์ที่ต้องการล๊อคทีละ 1 ไบท์

```
{
```

; ระหว่างข้อมูล 16 ไบท์ที่อ่านจากฮาร์ดล๊อคกับ

```
long ByteHi,ByteLo,bc,c1;
```

; ข้อมูล 8 ไบท์ที่อ่านจากไฟล์ที่ถูกล๊อค

```
char C_Data[17];
```

```
cr_dt_ee(numberB);
```

```
read_data(C_Data);
```

```
sscanf(C_Data,"%x",&bc);
```

```
bc=bc&0x000FFFFF;
```

```
ByteHi=bc/256;ByteLo=bc&0x00FF;
```

```
c1=Coded_D^ByteHi;Coded_D=c1^ByteLo;
```

```
return(Coded_D);
```

```
}
```

```
char *read_data(char *di_hex)
```

```
{
```

```
int di,m;
```

```
char di_dum[17]="",dis[2];
```

```
for (m=0;m<16;m++)
```

```
{ output(p,240+6);
```

```
    di=inport(p+1);
```

```
    output(p,240+7);
```

```
    di=di>>5;di=di&0x0001;itoa(di,dis,10);
```

```
    strcat(di_dum,dis);
```

```
}
```

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์สำหรับการใช้เพื่อการศึกษาเท่านั้น ไม่อนุญาตให้ทำซ้ำโดยไม่ได้รับอนุญาต

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ตัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

bin2hex(di_dum,di_hex);
return(di_hex);
}

char *bin2hex(char *bin, char *hex)
{
int ib,result=0,prod=1,len,lcnt,nibble=4;
char h[17]="";
for (ib=15;ib>=1;ib--)
{
if (bin[ib]=='1')
{
result=result+prod;
}
prod=prod*2;
}
itoa(result,hex,16);
len = strlen(hex);
if (len < nibble )
{
lcnt=nibble-len;
while (lcnt-- > 0) strcat(h,"0");
strcat(h, hex);
strcpy(hex, h);
}
strupr(hex); return(hex);
}

char *decode(char *enc,char *dec)
{
unsigned iht vd,vdum;
char b[17]="",c[17]="",enc_b[17];
int jd;
hex2bini(enc,enc_b);
for (jd=0;jd<16;jd++)
{
if (enc_b[jd]=='0') strcat(c,"1");
if (enc_b[jd]=='1') strcat(c,"0");
}
for (jd=0;jd<16;jd++)
{
if (jd==0 || jd<=7)
{
if (c[jd]==c[jd+8]) strcat(b,"0");
if (c[jd]!= c[jd+8]) strcat(b,"1");
}
else
{
if (c[jd]==c[jd+8]) strcat(b,"0");
if (c[jd]!=c[jd+8]) strcat(b,"1");
}
}
}
bin2hex(b,dec);
return(dec);
}

```

; เปลี่ยนเลขฐานสองเป็นเลขฐาน 16

; ถอดรหัส DATA

โปรแกรม DECODE.CPP

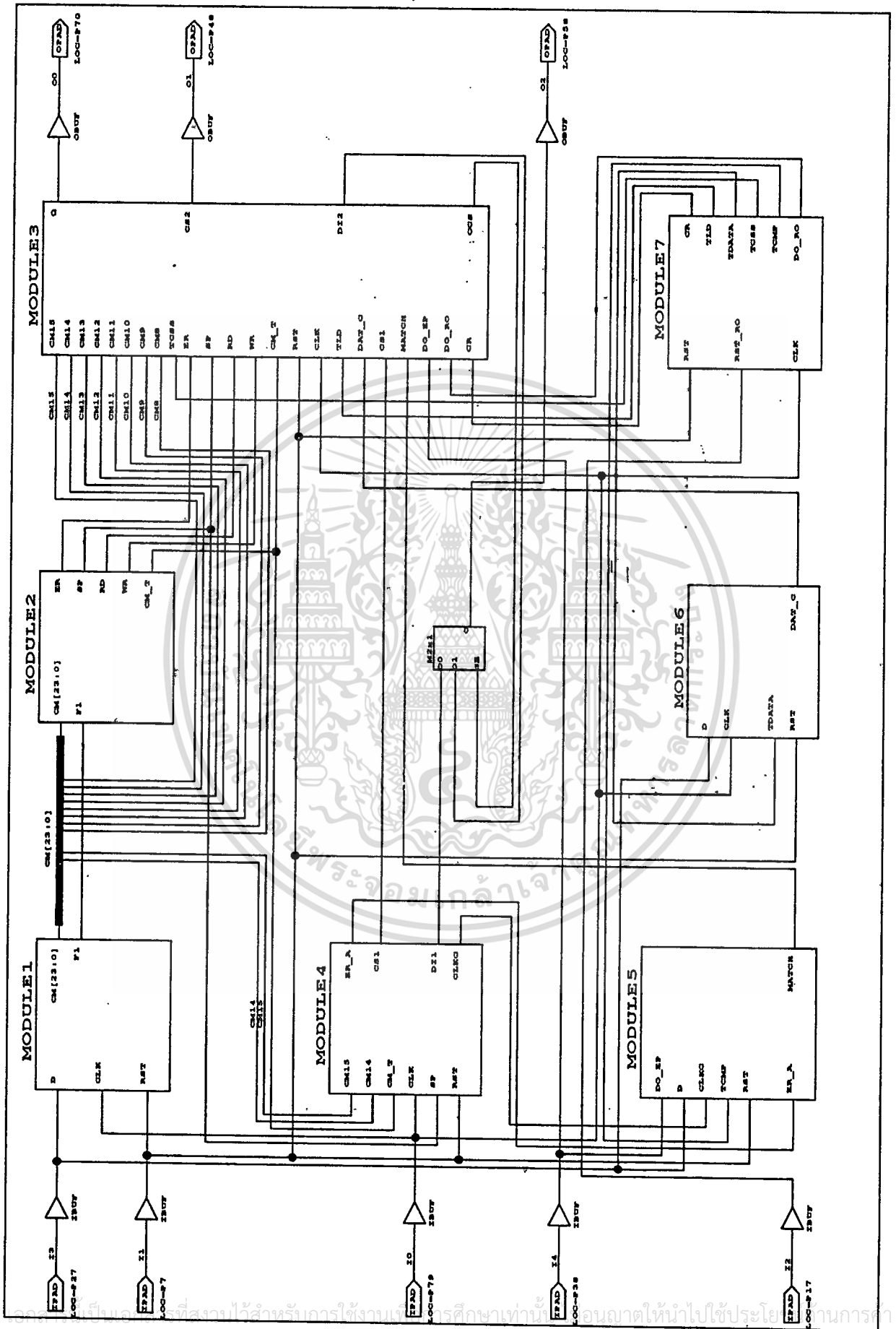
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <conio.h>
#include <dir.h>
#include <dos.h>

int p=3*16*16+7*16+8;
char secu_id0[5][17],secu_id1[5][17],secu_d0[5][5],secu_d1[5][5];
char code_dm[5],code[17],password[5][50],d_commd[57][40];
char code_sp [17]="0000000000000000",func_rd[9]="01100110";
char *read_data(char *di_hex);
char *hex2bini(char *hex, char *bin_inv);
char *bin2hex(char *bin, char *hex);
char *decode(char *enc,char *dec);
char NameBuff[_MAX_PATH];
long coding(long numberB,long Coded_D);
void cr_dt_ee(long);
void feed_data(char *,char *,char *);
void ch_passw(void);
void main(int argc, char *argv[])
{
FILE *TPFD,*TPFC,*temp,*TPF,*fi_s,*hist;
unsigned cdum1;
struct ffbk ffbk;
long PRcs_ip,FiSizeS,FiSizeA,FiSizeN,ParaB,i,C_Period,Cdum,Coded_D;
char ch[13],ch1[13],ExeFile[20],column1[13],non[13],N_Tfile[3];
int HPLo,HPHi,done,handle,ptr,endf,handle1,c,ExeN=0,pf;
char TempNoS[4],FileName[13],LockFile[128],FullN[128],ExeNS[3];
char FileC[40]="\\HARDLOCK\\TEMP\\TEMP.",ExePath[114]="",*FileN,TempFile[128]="";
char DelTemp[128]="DEL ",a='N',ch_hist[2],CopyFiC[128]="COPY ";

ch_passw();
for ( ; argc; argv++, argc--)
{
if (_fullpath(NameBuff, argv[0], _MAX_PATH) == NULL)
{
printf("Unable to obtain full path of %s\n",argv[0]);
}
else
for (i=0;i<(strlen(NameBuff));i++)
{
if (NameBuff[i]=='\\') pf=i;
}
for (i=0;i<pf+1;i++)
{
ExePath[i]=NameBuff[i];
}
strcpy(FullN,NameBuff); strnset(NameBuff,a,pf);
strtok(NameBuff,"\\");
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้คัดลอกเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```

hist=fopen("C:\\HARDLOCK\\HISTORY.INI","rt");
fseek(hist,0,SEEK_SET);

do { fgets(LockFile,128,hist);                                     ; ลำดับของโปรแกรมที่ล๊อคไว้
    strtok(LockFile,"\\n");
    if (strcmp(LockFile,FullN)==0) break;
    ExeN++;
} while (ExeN!=60);

itoa((ExeN/2)+1,ExeNS,10);strcat(FileC,ExeNS);

strcat(TempFile,ExePath);strcat(TempFile,"TEMP.EXE");
if ((handle = _creat(TempFile, 0)) < 0) {                           ; สร้างโปรแกรมเพื่อใช้เอ็กซีคิวต์ชั่วคราว
    perror("Unable to Execute this File"); }
_close(handle);

strcat(CopyFiC,FileC);strcat(CopyFiC," ");
strcat(CopyFiC,TempFile); system(CopyFiC);clrscr();

TPFC=fopen(FileC,"rb");                                           ; โปรแกรมสำรองที่สร้างไว้ตอนใช้ HL_SHELL
TPFD=fopen(TempFile,"rb+");                                       ; โปรแกรมเพื่อใช้เอ็กซีคิวต์ชั่วคราว
fseek(TPFC,0,SEEK_END);
FiSizeS=ftell(TPFC);
for (i=0;i<(FiSizeS);i++)
{
    if ((i>19 && i<256) || (i>255 && (i%256==0) || (i%257==0) || (i%258==0)))
    { if (i!=0)
        {
            fseek(TPFC,i, SEEK_SET);
            Cdum=fgetc(TPFC);
            if (i<(FiSizeS/2))
            {
                Coded_D=coding((i%56),Cdum);                     ; ถอดรหัสไฟล์สำรองที่อยู่ใน
                                                                    ; C:\\HARDLOCK\\TEMP\\TEMP.ลำดับที่ล๊อค
            }
        }
        else
        {
            Coded_D=coding(56,Cdum);
        }
        fseek(TPFD,i, SEEK_SET);
        fwrite(&Coded_D, 1, 1, TPFD);
    }
}

fclose(TPFC);fclose(TPFD);fclose(hist);
system(TempFile);
strcat(DelTemp,TempFile);system(DelTemp);                         ; ลบโปรแกรมที่ใช้เอ็กซีคิวต์ชั่วคราวออก
clrscr();
}

```



ภาคผนวก ค.

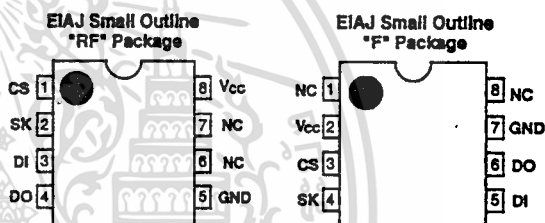
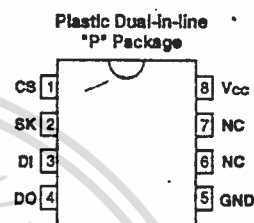
ข้อมูลไอซี EEPROM HY93C46 , PROM XC17000 และ FPGA XC3000

1,024-Bit Serial Electrically Erasable PROM with 2V Read Capability

FEATURES

- 2.7 to 5.5V Operation (XL93LC46)
4.5 to 5.5V Operation (XL93LC46A)
- Extended Temperature Range: -40°C to +85°C
- State-of-the-Art Architecture
 - Nonvolatile data storage
 - Fully TTL compatible inputs and outputs
 - Auto increment for efficient data dump
- Hardware and Software Write Protection
 - Defaults to write-disabled state at power up
 - Software instructions for write-enable/disable
 - Vcc lockout inadvertent write protection (XL93LC46A)
- Low Power Consumption
 - 1mA active (typical)
 - 1µA standby (typical)
- Low Voltage Read Operations
 - Reliable read operations down to 2.0 volts
- Advanced Low Voltage CMOS E²PROM Technology
- Versatile, Easy-to-Use Interface
 - Self-timed programming cycle
 - Automatic erase-before-write
 - Programming Status Indicator
 - Word and chip erasable
- Durable and Reliable
 - 10-year data retention after 100K write cycles
 - Minimum of 100,000 write cycles per word
 - Unlimited read cycles
 - ESD protection (EIAJ and JEDEC standard)

PIN CONFIGURATIONS



PIN NAMES

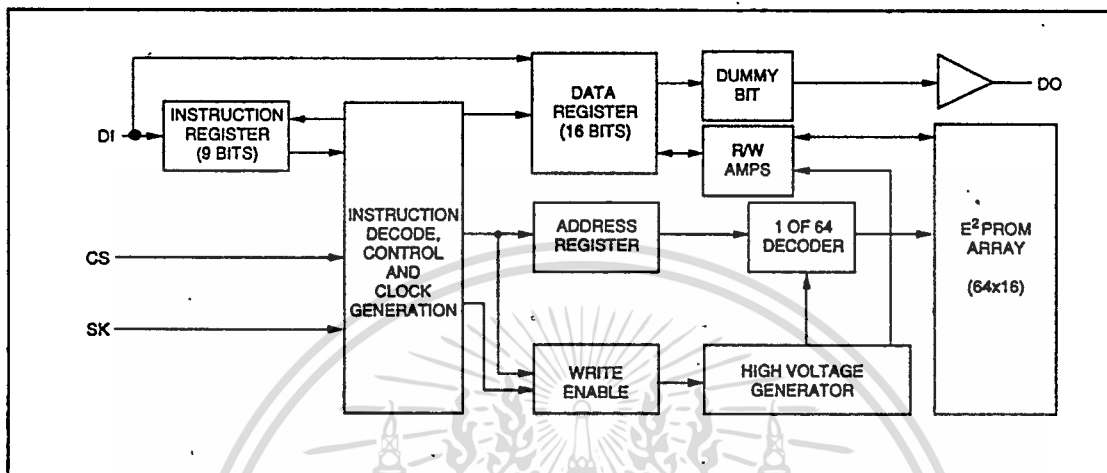
CS	Chip Select
SK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
GND	Ground
Vcc	Power Supply
NC	Not Connected

OVERVIEW

The XL93LC46/46A is a cost effective 1,024-bit, nonvolatile, serial E²PROM. It is fabricated using EXEL's advanced CMOS E²PROM technology. The XL93LC46/46A provides efficient nonvolatile read/write memory arranged as 64 registers of 16 bits each. Seven 9-bit instructions control the operation of the device, which include read, write, and mode enable functions. The data output pin (DO) indicates the status of the device during the self-timed nonvolatile programming cycle.

The self-timed write cycle includes an automatic erase-before-write capability. To protect against inadvertent writes, the WRITE instruction is accepted only while the chip is in the write enabled state. Data is written in 16 bits per write instruction into the selected register. If Chip Select (CS) is brought HIGH after initiation of the write cycle, the Data Output (DO) pin will indicate the READY/BUSY status of the chip.

BLOCK DIAGRAM



APPLICATIONS

The XL93LC46/46A is ideal for high volume applications requiring low power and low density storage. This device uses a cost effective, space saving 8-pin package. Candidate applications include robotics, alarm devices, electronic locks, meters and instrumentation settings.

ENDURANCE AND DATA RETENTION

The XL93LC46/46A is designed for applications requiring up to 100,000 write cycles per bit. It provides 10 years of secure data retention without power after the execution of 100,000 write cycles for each location.

DEVICE OPERATION

The XL93LC46/46A is controlled by seven 9-bit instructions. Instructions are clocked in (serially) on the DI pin. Each instruction begins with a logical "1" (the start bit). This is followed by the opcode (2 bits), the address field (6 bits), and data, if appropriate. The clock signal (SK) may be halted at any time and the XL93LC46/46A will remain in its last state. This allows full static flexibility and maximum power conservation.

Read (READ)

The READ instruction is the only instruction that results in serial data on the DO pin. After the read instruction and address have been decoded, data is transferred from the selected memory register into a 16-bit serial shift register. (Please note that one logical "0" bit precedes the actual 16-bit output data string.) The output on DO changes during the LOW-TO-HIGH transitions of SK. (See Figure 2.)

Low Voltage Read

The XL93LC46/46A has been designed to ensure that data read operations are reliable in low voltage environments. The XL93LC46/46A is guaranteed to provide accurate data during read operations with V_{CC} as low as 2.0V.

Auto Increment Read Operations

In order to facilitate memory transfer operations, the XL93LC46/46A has been designed to output a continuous stream of memory content in response to a single read operation instruction. To utilize this function, the system asserts a read instruction specifying a start location address. Once the 16 bits of the addressed word have been clocked out, the data in consecutively higher address locations is output. The address will wrap around continuously with CS HIGH until the Chip Select control pin is brought LOW. This allows for single instruction data dumps to be executed with a minimum of firmware overhead.

Write Enable (WEN)

The write enable (WEN) instruction must be executed before any device programming can be done. When V_{CC} is applied, this device powers up in the write disabled state. The device then remains in a write disabled state until a WEN instruction is executed. Thereafter the device remains enabled until a WDS instruction is executed or until V_{CC} is removed. (NOTE: Neither the WEN nor the WDS instruction has any effect on the READ instruction. See Figure 3.)

Write (WRITE)

The WRITE instruction includes 16 bits of data to be written into the specified register. After the last data bit has been clocked into DI, and before the next rising edge of SK, CS must be brought LOW. The falling edge of CS initiates the self-timed programming cycle.

After a minimum wait of 250ns from the falling edge of CS (t_{cs}), if CS is brought HIGH, DO will indicate the READY/BUSY status of the chip: logical "0" means programming is still in progress; logical "1" means the selected register has been written, and the part is ready for another instruction. (See Figure 4.) (NOTE: The combination of CS HIGH, DI HIGH and the rising edge of the SK clock, resets the READY/BUSY flag. Therefore, it is important not to reset the READY/BUSY flag through this combination of control signals, if you want to access it.) Before a WRITE instruction can be executed, the device must be write enabled (see WEN).

Write All (WRALL)

The write all (WRALL) instruction programs all registers with the data pattern specified in the instruction.

As with the WRITE instruction, if CS is brought HIGH after a minimum wait of 250ns (t_{cs}), the DO pin indicates the READY/BUSY status of the chip. (See Figure 5.)

Write Disable (WDS)

The write disable (WDS) instruction disables all programming capabilities. This protects the entire memory array against accidental modification of data until a WEN in-

struction is executed. (When V_{cc} is applied, the part powers up in the write disabled state.) To protect data, a WDS instruction should be executed upon completion of each programming operation. (NOTE: Neither the WEN nor the WDS instruction has any effect on the READ instruction. See Figure 6.)

Erase Register

After the erase instruction is entered, CS must be brought LOW. The falling edge of CS initiates the self-timed internal programming cycle. Bringing CS HIGH after a minimum of t_{cs} will cause DO to indicate the READY/BUSY status of the chip: a logical "0" indicates programming is still in progress; a logical "1" indicates the erase cycle is complete and the part is ready for another instruction. (See Figure 7.) Erase of a register sets all of the bits of the register to a logical "1".

Erase All (ERALL)

Full chip erase is provided for ease of programming. Erasing the entire chip involves setting all bits in the entire memory array to a logical "1". (See Figure 8.)

Vcc Lockout - Inadvertent Write Protection (XL93LC46A only)

To ensure against inadvertent write operation, the XL93LC46A has been equipped with an internal V_{cc} sensor circuit which inhibits data alteration when the supply voltage (V_{cc}) falls below V_{wl} . If the applied V_{cc} is below 3.75V (typical), the XL93LC46A is inhibited from executing write operations thereby protecting the non-volatile data from inadvertent write operations.

XL93LC46/46A INSTRUCTION SET

Instruction	Start Bit	OP Code	Address	Input Data
READ	1	10	(A5-A0)	
WEN (Write Enable)	1	00	11XXXX	
WRITE	1	01	(A5-A0)	D15-D0
WRALL (Write All Registers)	1	00	01XXXX	D15-D0
WDS (Write Disable)	1	00	00XXXX	
ERASE	1	11	(A5-A0)	
ERALL (Erase All Registers)	1	00	10XXXX	

ABSOLUTE MAXIMUM RATINGS

Temperature under bias:-40°C to +85°C
Storage Temperature-65°C to +150°C
Lead Soldering Temperature (less than 10 seconds) 300°C
Supply Voltage 0 to 6.5V
Voltage on Any Pin-0.3 to Vcc + 0.3V
ESD Rating 2000V

NOTE: These are STRESS ratings only. Appropriate conditions for operating these devices are given elsewhere in this specification. Stresses beyond those listed here may permanently damage the part. Prolonged exposure to maximum ratings may adversely affect device reliability.

DC ELECTRICAL CHARACTERISTICS

TA = -40°C to +85°C

Symbol	Parameter	Conditions	XL93LC46		XL93LC46/46A		XL93LC46/46A		Units
			Vcc = 3.0V±10%		Vcc = 5.0V±10%		Vcc = 2.0V (Read Only)		
			Min	Max	Min	Max	Min	Max	
Icc1	Operating Current CMOS Input Levels	CS = VCC, SK = 250KHz		2		2		2	mA
Icc2	Operating Current TTL Input Levels	CS = VIH, SK = 1MHz		n/a		5		n/a	mA
ISS	Standby Current	CS = DI = SK = 0V		2		2		2	µA
ILI	Input Leakage	VIN = 0V to VCC (CS, SK, DI)	-1	1	-1	1	-1	1	µA
ILO	Output Leakage	VOUT = 0V to Vcc, CS = 0V	-1	1	-1	1	-1	1	µA
VIL	Input Low Voltage		-0.1	0.15VCC	-0.1	0.8	-0.1	0.1 VCC	V
VIH	Input High Voltage		0.8 VCC	VCC+0.2	2	VCC+0.2	0.9 VCC	VCC+0.2	V
VOL1	Output Low Voltage	IOL = 2.1mA TTL		n/a		0.4		n/a	V
VOH1	Output High Voltage	IOH = -400µA TTL	n/a		2.4		n/a		V
VOL2	Output Low Voltage	IOL = 10µA CMOS		0.2		0.2		0.2	V
VOH2	Output High Voltage	IOH = -10µA CMOS	Vcc-0.2		Vcc-0.2		Vcc-0.2		V
VWI	Write Inhibit Threshold		n/a	n/a	2.7	4.4	n/a	n/a	V

AC ELECTRICAL CHARACTERISTICS

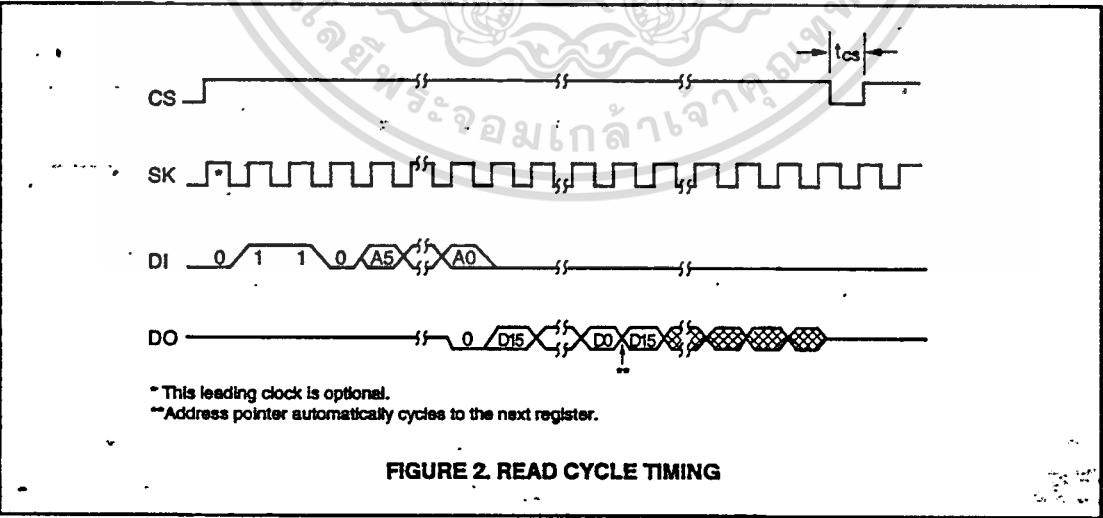
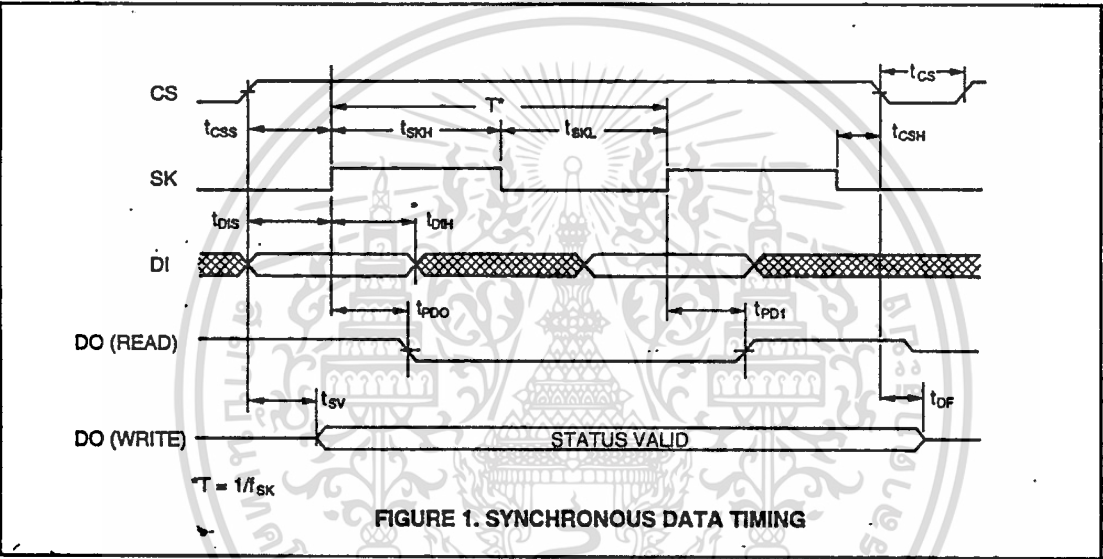
TA = -40°C to +85°C

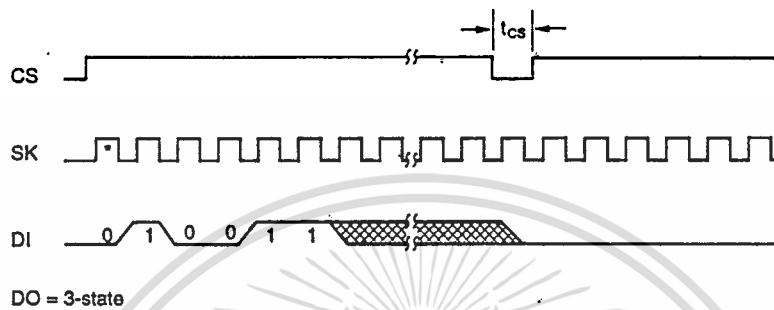
Symbol	Parameter	Conditions	XL93LC46		XL93LC46/46A		XL93LC46/46A		Units
			Vcc = 3.0V±10%		Vcc = 5.0V±10%		Vcc = 2.0V (Read Only)		
			Min	Max	Min	Max	Min	Max	
fSK	SK Clock Frequency		0	250	0	1000	0	250	KHz
tSKH	SK High Time		1000		400		2000		ns
tSKL	SK Low Time		1000		250		2000		ns
tCS	Minimum CS Low Time		1000		250		1000		ns
tCSS	CS Setup Time	Relative to SK ⌋	200		50		200		ns
tDIS	DI Setup Time	Relative to SK ⌋	400		100		400		ns
tCSH	CS Hold Time	Relative to SK ⌋	0		0		0		ns
tDIH	DI Hold Time	Relative to SK ⌋	400		100		400		ns
tPO1	Output Delay to "1"	AC Test		2000		500		2000	ns
tPO0	Output Delay to "0"	AC Test		2000		500		2000	ns
tSV	CS to Status Valid	AC Test CL = 100pF		2000		500		2000	ns
tDP	CS to DO In-Status	CS = Low to DO = Hi-Z		400		100		400	ns
tWC	Write Cycle Time	CS = Low to DO = Ready		25		10		n/a	µs

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CAPACITANCE
TA = 25°C, f = 250KHz

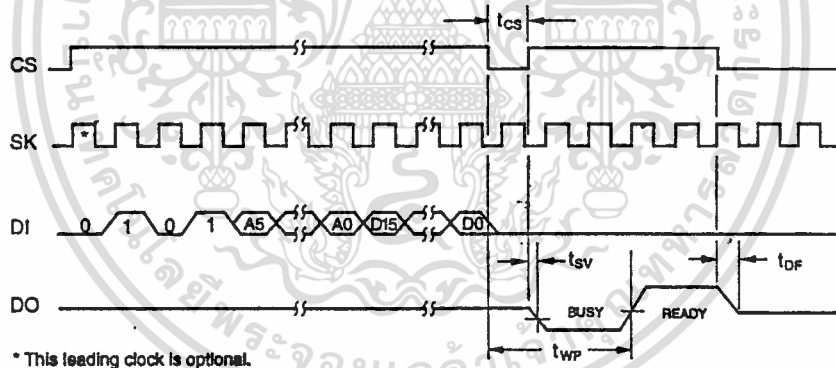
Symbol	Parameter	Max	Units
CIN	Input Capacitance	5	pF
COUT	Output Capacitance	5	pF





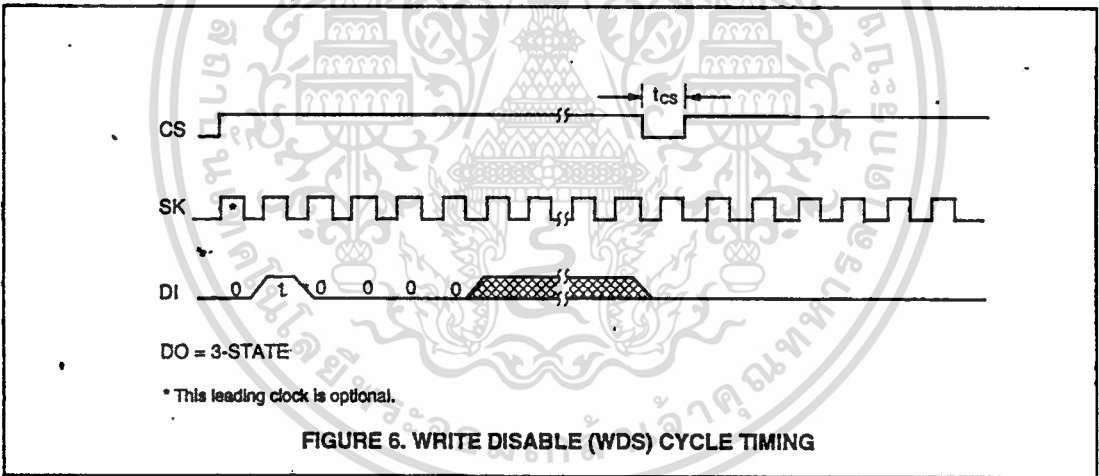
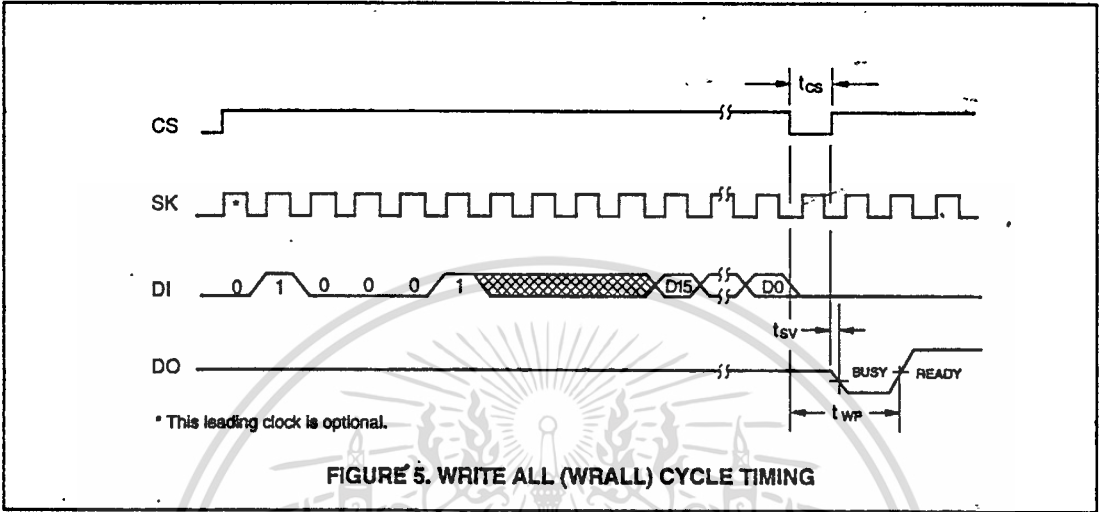
* This leading clock is optional.

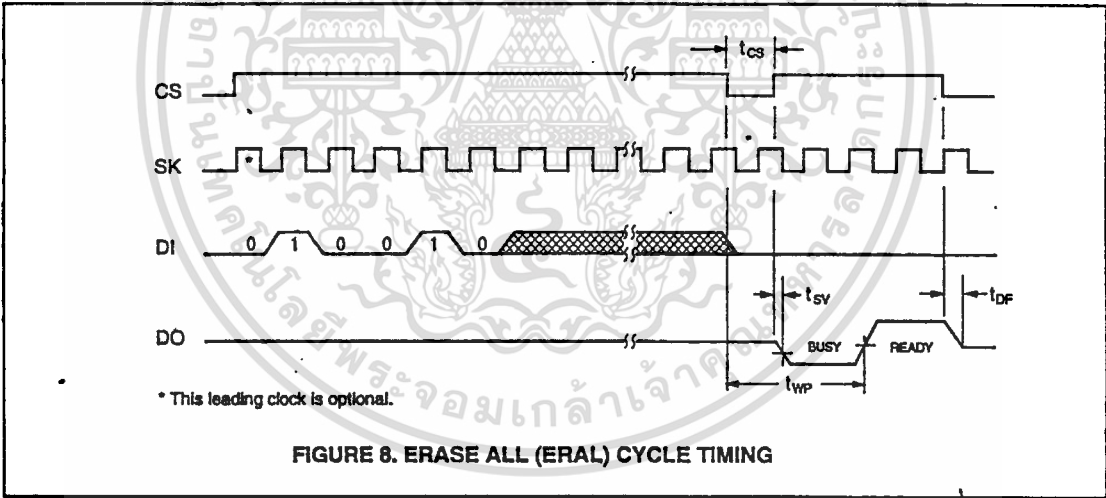
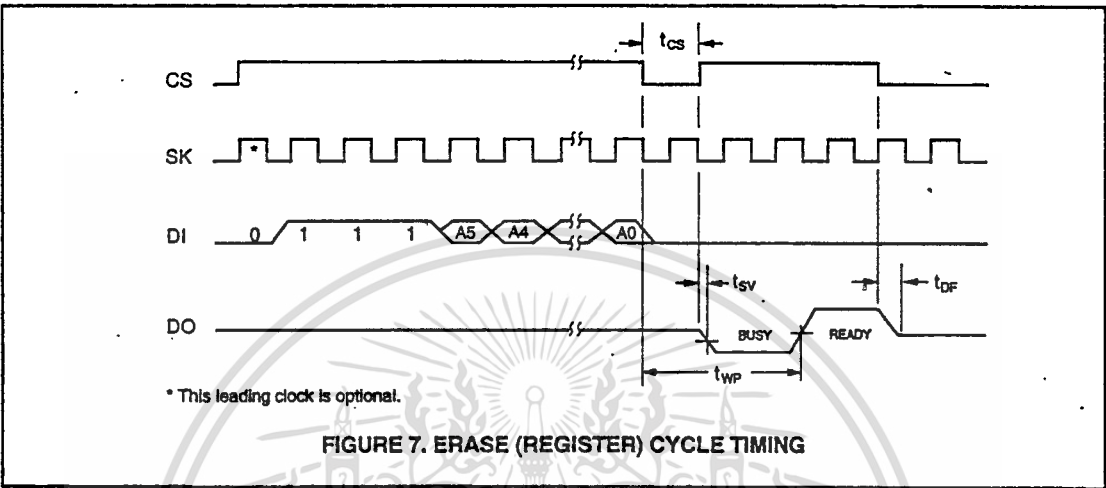
FIGURE 3. WRITE ENABLE (WEN) CYCLE TIMING



* This leading clock is optional.

FIGURE 4. WRITE CYCLE TIMING







XC17000 Family of Serial Configuration PROMs

Preliminary Product Specifications

Features

- Extended family of one-time programmable (OTP) bit-serial read-only memories used for storing the configuration bitstreams of Xilinx FPGAs
- On-chip address counter, incremented by each rising edge on the clock input
- Simple interface to the FPGA requires only one user I/O pin
- Cascadable for storing longer or multiple bitstreams
- Programmable reset polarity (active High or active Low) for compatibility with different FPGA solutions, (the older XC1736A has active-High reset only)
- XC17128 supports XC4000 fast configuration mode (10 MHz)
- Low-power CMOS EPROM process
- Available in 5 V and 3.3 V versions
- Available in plastic and ceramic packages, and commercial, industrial and military temperature ranges
- Space efficient 8-pin DIP, 8-pin SOIC or 20-pin surface-mount packages.
- Programming support by leading programmer manufacturers.

Description

The XC17000 family of serial configuration PROMs (SCPs) provides an easy-to-use, cost-effective method for storing Xilinx FPGA configuration bitstreams.

When the FPGA is in master serial mode, it generates a configuration clock that drives the SCP. A short access time after the rising clock edge, data appears on the SCP DATA output pin that is connected to the FPGA DIN pin. The FPGA generates the appropriate number of clock pulses to complete the configuration. Once configured, it disables the SCP. When the FPGA is in slave mode, the SCP and the FPGA must both be clocked by an incoming signal.

Multiple devices can be concatenated by using the $\overline{CE}$ output to drive the $\overline{CE}$ input of the following device. The clock inputs and the DATA outputs of all SCPs in this chain are interconnected. All devices are compatible and can be cascaded with other members of the family.

For device programming, the XACT development system compiles the LCA design file into a standard Hex format, which is then transferred to the programmer.

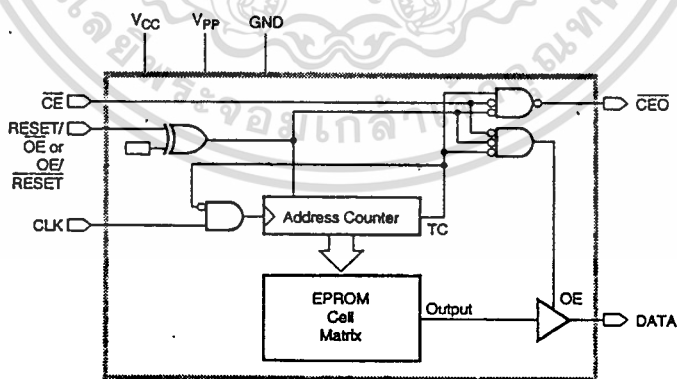


Figure 1. Simplified Block Diagram (does not show programming circuit)

Pin Assignments

DATA

Data output, 3-stated when either $\overline{CE}$ or $\overline{OE}$ are inactive. During programming, the DATA pin is I/O. Note that $\overline{OE}$ can be programmed to be either active High or active Low.

CLK

Each rising edge on the CLK input increments the internal address counter, if both $\overline{CE}$ and $\overline{OE}$ are active. Note that $\overline{OE}$ can be programmed to be either active High or active Low.

RESET/ $\overline{OE}$

When High, this input holds the address counter reset and 3-states the DATA output. The polarity of this input pin is programmable as either RESET/ $\overline{OE}$ or OE/RESET. To avoid confusion, this document describes the pin as RESET/ $\overline{OE}$, although the opposite polarity is possible on all devices except the older XC1736A.

$\overline{CE}$

When High, this pin disables the internal address counter, 3-states the DATA output, and forces the device into low- I_{CC} standby mode.

$\overline{CEO}$

Chip Enable output, to be connected to the CE input of the next SCP in the daisy chain. This output is Low when the $\overline{CE}$ and $\overline{OE}$ inputs are both active AND the internal address counter has been incremented beyond its Terminal Count (TC) value. In other words: when the PROM has been read, $\overline{CEO}$ will follow $\overline{CE}$ as long as $\overline{OE}$ is active. When $\overline{OE}$ goes inactive, $\overline{CEO}$ stays High until the PROM is reset. Note that $\overline{OE}$ can be programmed to be either active High or active Low.

V_{PP}

Programming voltage. No overshoot above the specified max voltage is permitted on this pin. For normal read operation, this pin *must* be connected to V_{CC} . Failure to do so may lead to unpredictable, temperature-dependent operation and severe problems in circuit debugging. *Do not leave V_{PP} floating!*

V_{CC}

Positive supply pin.

GND

Ground pin

Serial PROM Pinouts

Pin Name	8-Pin	20-Pin
DATA	1	2
CLK	2	4
RESET/ $\overline{OE}$ (OE/RESET)	3	6
$\overline{CE}$	4	8
GND	5	10
$\overline{CEO}$	6	14
V_{PP}	7	17
V_{CC}	8	20

Number of Configuration Bits, Including Header for all Xilinx FPGAs

Device	Configuration Bits
XC2064	12,038
XC2018	17,878
XC3020/3120	14,819
XC3030/3130	22,216
XC3042/3142	30,824
XC3064/3164	46,104
XC3090/3190	64,200
XC3195	94,984
XC4002A	31,668
XC4003A	45,676
XC4003H	53,967
XC4004A	62,244
XC4005A	81,372
XC4005/4005H	95,000
XC4006	119,832
XC4008	147,544
XC4010	178,136
XC4013	247,960

Controlling Serial PROMS

Most connections between the LCA device and the Serial PROM are simple and self-explanatory.

- The DATA output of the PROM drives DIN of the LCA devices.
- The master LCA CCLK output drives the CLK input of the Serial PROM.
- The $\overline{CEO}$ output of any Serial PROM can be used to drive the $\overline{CE}$ input of the next serial PROM in a cascade chain of PROMs.
- V_{PP} *must* be connected to V_{CC} . Leaving V_{PP} open can lead to unreliable, temperature-dependent operation.

There are, however, two different ways to use the inputs $\overline{CE}$ and $\overline{OE}$.

1. The LCA $D/\overline{P}$ or LDC output drives both $\overline{CE}$ and $\overline{OE}$ in parallel. This is the simplest connection, but it fails if a user applies $\overline{RESET}$ during the LCA configuration process. The LCA device aborts the configuration and then restarts a new configuration, as intended, but the Serial PROM does not reset its address counter, since it never saw a High level on its $\overline{OE}$ input. The new configuration, therefore, reads the remaining data in the PROM and interprets it as preamble, length count etc. Since the LCA device is the master, it issues the necessary number of CCLK pulses, up to 16 million (2^{24}) and $D/\overline{P}$ goes High. However, the LCA configuration will be completely wrong, with potential contentions inside the LCA device and on its output pins. *This method must, therefore, never be used when there is any chance of external reset during configuration.*
2. The LCA $D/\overline{P}$ or LDC output drives only the $\overline{CE}$ input of the Serial PROM while its $\overline{OE}$ input is driven by the LCA $\overline{RESET}$ input. This connection works under all normal circumstances, even when the user aborts a configuration before $D/\overline{P}$ has gone High. The Low level on the $\overline{OE}$ input during reset clears the PROM internal address pointer, so that the reconfiguration starts at the beginning. The reset polarity should be inverted for this mode to be used. It is strongly recommended that this method, shown in Figure 2, be used when-ever possible.

LCA Master Serial Mode Summary

The I/O and logic functions of the Logic Cell Array and their associated interconnections are established by a configuration program. The program is loaded either automatically upon power up, or on command, depending on the state of the three LCA mode pins. In Master Mode, the Logic Cell Array automatically loads the configuration program from an external memory. The Serial Configuration PROM has been designed for compatibility with the Master Serial Mode.

Upon power-up or reconfiguration, an LCA device enters the Master Serial Mode whenever all three of the LCA

mode-select pins are Low ($M0=0$, $M1=0$, $M2=0$). Data is read from the Serial Configuration PROM sequentially on a single data line. Synchronization is provided by the rising edge of the temporary signal CCLK, which is generated during configuration.

Master Serial Mode provides a simple configuration interface. Only a serial data line and two control lines are required to configure an LCA device. Data from the Serial Configuration PROM is read sequentially, accessed via the internal address and bit counters which are incremented on every valid rising edge of CCLK.

If the user-programmable, dual-function DIN pin on the LCA device is used only for configuration, it must still be held at a defined level during normal operation. The XC3000 and XC4000 families take care of this automatically with an on-chip default pull-up resistor. With XC2000-family devices, the user must either configure DIN as an active output, or provide a defined level, e.g., by using an external pull-up resistor, if DIN is configured as an input.

Programming the LCA With Counters Unchanged Upon Completion

When multiple LCA-configurations for a single LCA are stored in a Serial Configuration PROM, the $\overline{OE}$ pin should be tied Low as shown in Figure 3. Upon power-up, the internal address counters are reset and configuration begins with the first program stored in memory. Since the $\overline{OE}$ pin is held Low, the address counters are left unchanged after configuration is complete. Therefore, to reprogram the LCA with another program, the $D/\overline{P}$ line is pulled Low and configuration begins at the last value of the address counters.

Cascading Serial Configuration PROMs

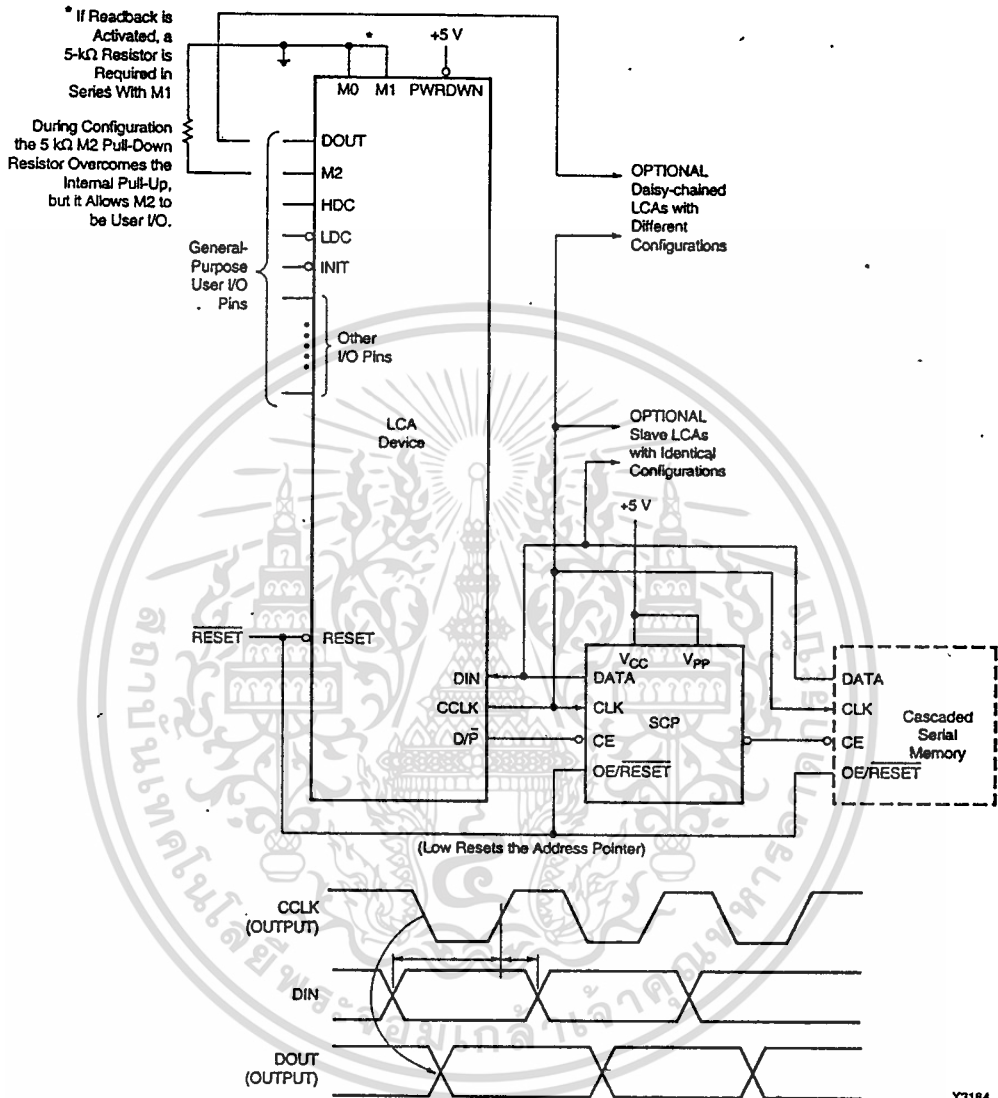
For multiple LCAs configured as a daisy-chain, or for future LCAs requiring larger configuration memories, cascaded SCPs provide additional memory. After the last bit from the first SCP is read, the next clock signal to the SCP asserts its $\overline{CEO}$ output Low and disables its DATA line. The second SCP recognizes the Low level on its $\overline{CE}$ input and enables its DATA output. See Figure 2.

After configuration is complete, the address counters of all cascaded SCPs are reset if the LCA $\overline{RESET}$ pin goes Low, assuming the SCP reset polarity option has been inverted.

If the address counters are not to be reset upon completion, then the $\overline{RESET}/\overline{OE}$ inputs can be tied to ground, as shown in Figure 3. To reprogram the LCA device with another program, the $D/\overline{P}$ line goes Low and configuration begins where the address counters had stopped. In this case, avoid contention between DATA and the configured I/O use of DIN.

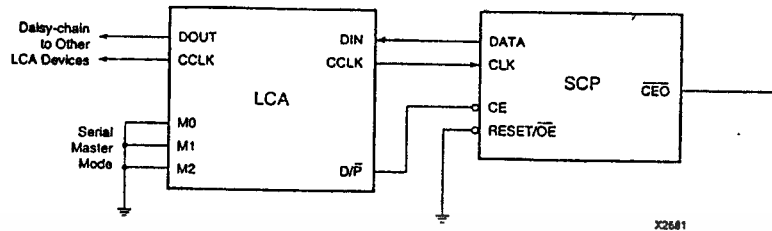
Extremely large, cascaded memories in some systems may require additional logic if the rippled chip enable is too slow to activate successive SCPs.

XC17000 Serial Configuration PROM



X3184

Figure 2. Master Serial Mode. The one-time-programmable Serial Configuration PROM supports automatic loading of configuration programs. Multiple devices can be cascaded to support additional LCA devices. An early D/P inhibits the PROM data output one CCLK cycle before the LCA I/Os become active.



- Notes:**
1. If M2 is tied directly to ground, it should be programmed as an input during operation.
 2. If the LCA is reset during configuration, it will abort back to initialization state. D/P will not go High, so an external signal is required to reset the XC17XX counters.

Figure 3. Address Counters Not Reset at the End of Configuration

Standby Mode

The PROM enters a low-power standby mode whenever $\overline{CE}$ is asserted High. In this mode, the SCP consumes less than 0.5 mA of current. The output remains in a high-impedance state regardless of the state of the $\overline{OE}$ input.

Programming the XC17000 Family Serial PROMs

The devices can be programmed on programmers supplied by Xilinx or other qualified third-party vendors. The user must ensure that the appropriate programming algorithm and voltage are used. Different product types use different algorithms and voltages, and the wrong choice can permanently damage the device.

XC17000 Serial Configuration PROM

XC1718D, XC1736D, XC1765D, XC17128

Absolute Maximum Ratings

Symbol	Description		Units
V _{CC}	Supply voltage relative to GND	-0.5 to +7.0	V
V _{PP}	Supply voltage relative to GND: XC1718D, XC1736D, XC1765D	-0.5 to +12.5	V
	Supply voltage relative to GND: XC17128	-0.5 to +15.5	V
V _{IN}	Input voltage relative to GND	-0.5 to V _{CC} +0.5	V
V _{TS}	Voltage applied to 3-state output	-0.5 to V _{CC} +0.5	V
T _{STG}	Storage temperature (ambient)	-65 to +125	°C
T _{SOL}	Maximum soldering temperature (10 s @ 1/16 in.)	+260	°C

Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

Symbol	Description		Min	Max	Units
V _{CC}	Commercial	Supply voltage relative to GND -0 °C to +70°C	4.75	5.25	V
	Industrial	Supply voltage relative to GND -40°C to +85°C	4.5	5.5	V
	Military	Supply voltage relative to GND -55°C to +125°C	4.5	5.5	V

DC Characteristics Over Operating Condition

Symbol	Description		Min	Max	Units
V _{IH}	High-level input voltage		2.0	V _{CC}	V
V _{IL}	Low-level input voltage		0	0.8	V
V _{OH}	High-level output voltage (I _{OH} = -4 mA)	Commercial	3.86		V
V _{OL}	Low-level output voltage (I _{OL} = +4 mA)			0.32	V
V _{OH}	High-level output voltage (I _{OH} = -4 mA)	Industrial	3.76		V
V _{OL}	Low-level output voltage (I _{OL} = +4 mA)			0.37	V
V _{OH}	High-level output voltage (I _{OH} = -4 mA)	Military	3.7		V
V _{OL}	Low-level output voltage (I _{OL} = +4 mA)			0.4	V
I _{CCA}	Supply current, active mode			10	mA
I _{CCS}	Supply current, standby mode			0.5	mA
I _L	Input or output leakage current		-10	10	μA

Note: During normal read operation V_{PP} *must* be connected to V_{CC}

XC1718L and XC1765L

Absolute Maximum Ratings

Symbol	Description		Units
V_{CC}	Supply voltage relative to GND	-0.5 to +6.0	V
V_{PP}	Supply voltage relative to GND	-0.5 to +12.5	V
V_{IN}	Input voltage with respect to GND	-0.5 to $V_{CC} + 0.5$	V
V_{TS}	Voltage applied to 3-state output	-0.5 to $V_{CC} + 0.5$	V
T_{STG}	Storage temperature (ambient)	-65 to +125	°C
T_{SOL}	Maximum soldering temperature (10 s @ 1/16 in.)	+260	°C

Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

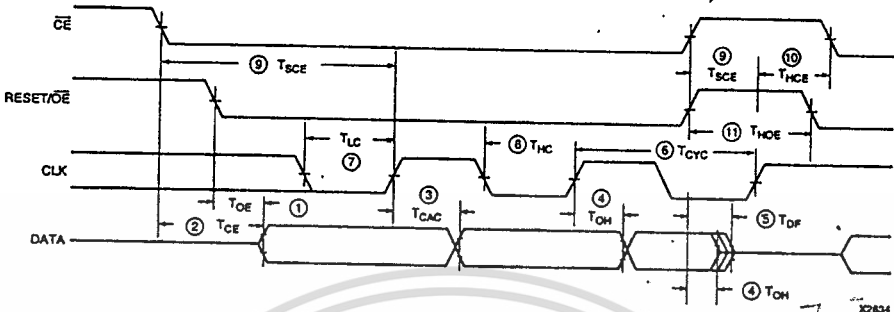
Symbol	Description		Min	Max	Units
V_{CC}	Commercial	Supply voltage relative to GND -0 °C to +70°C	3.0	3.6	V
	Industrial	Supply voltage relative to GND -40°C to +85°C	3.0	3.6	V
	Military	Supply voltage relative to GND -55°C to +125°C	3.0	3.6	V

DC Characteristics Over Operating Condition

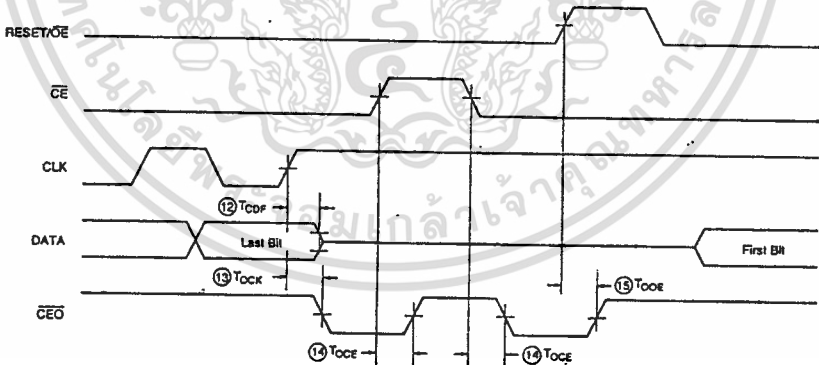
Symbol	Description	Min	Max	Units
V_{IH}	High-level input voltage	2.0	V_{CC}	V
V_{IL}	Low-level input voltage	0	0.8	V
V_{OH}	High-level output voltage ($I_{OH} = -4$ mA)	2.4		V
V_{OL}	Low-level output voltage ($I_{OL} = +4$ mA)		0.4	V
I_{CCA}	Supply current, active mode		5	mA
I_{CCS}	Supply current, standby mode		0.5	mA
I_L	Input or output leakage current	-10	10	μA

Note: During normal read operation V_{PP} must be connected to V_{CC}

AC Characteristics Over Operating Conditions



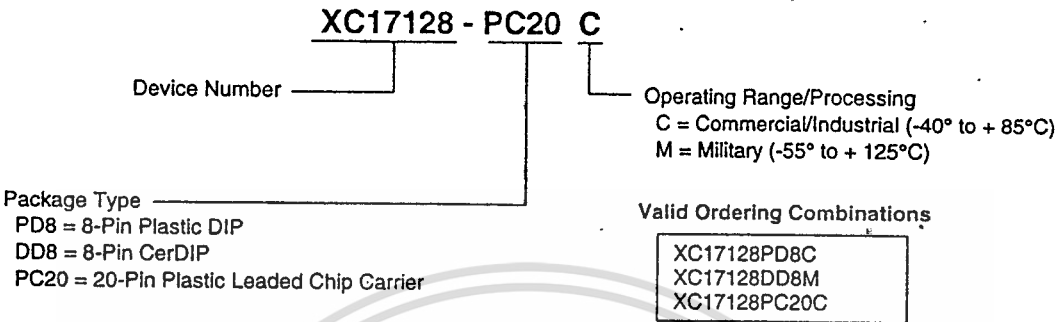
Symbol	Description	XC1718D, XC1736D, XC1765D		XC1718L, XC1765L		XC17128		Units
		Min	Max	Min	Max	Min	Max	
1 T_{HOE}	OE to Data Delay		45		45		50	ns
2 T_{CE}	$\overline{CE}$ to Data Delay		60		60		50	ns
3 T_{CAC}	CLK to Data Delay		150		200		60	ns
4 T_{OH}	Data Hold From $\overline{CE}$, $\overline{OE}$, or CLK	0		0		0		ns
5 T_{DF}	$\overline{CE}$ or $\overline{OE}$ to Data Float Delay ²		50		50		50	ns
6 T_{CYC}	Clock Periods	200		400		100		ns
7 T_{LC}	CLK Low Time ³	100		100		25		ns
8 T_{HC}	CLK High Time ³	100		100		25		ns
9 T_{SCE}	$\overline{CE}$ Setup Time to CLK (to guarantee proper counting)	25		40		25		ns
10 T_{HCE}	$\overline{CE}$ Hold Time to CLK (to guarantee proper counting)	0		0		0		ns
11 T_{HOE}	$\overline{OE}$ High Time (guarantees counters are reset)	100		100		20		ns



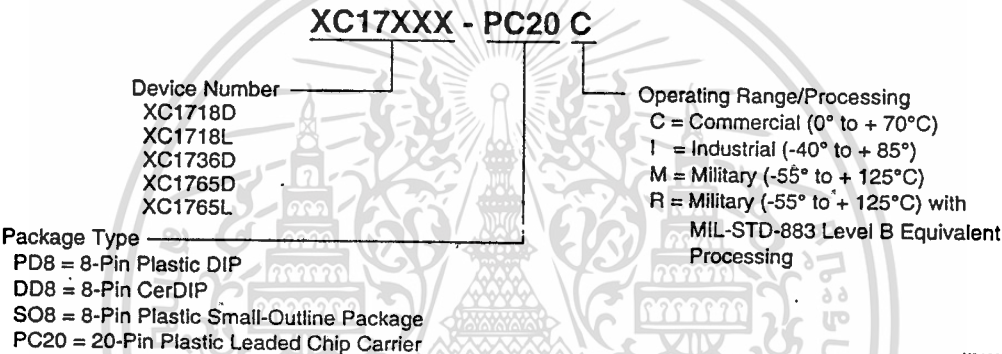
Symbol	Description	XC1718D, XC1736D, XC1765D		XC1718L, XC1765L		XC17128		Units
		Min	Max	Min	Max	Min	Max	
12 T_{CDF}	CLK to Data Float Delay ²		50		50		50	ns
13 T_{OCK}	CLK to $\overline{CEO}$ Delay		65		65		40	ns
14 T_{OCE}	$\overline{CE}$ to $\overline{CEO}$ Delay		45		45		40	ns
15 T_{OOE}	RESET/ $\overline{OE}$ to $\overline{CEO}$ Delay		40		40		45	ns

- Notes:
1. AC test load = 50 pF
 2. Float delays are measured with minimum tester ac load and maximum dc load.
 3. Guaranteed by design, not tested.
 4. All ac parameters are measured with $V_{IL} = 0.0$ V and $V_{IH} = 3.0$ V.

Ordering Information



X3179



X3180

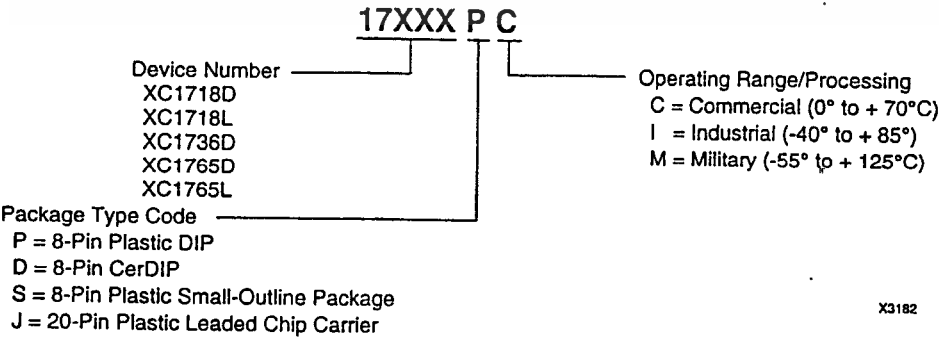
Valid Ordering Combinations

XC1718DPD8C XC1718DPD8I XC1718DSO8C XC1718DSO8I XC1718DPC20C XC1718DPC20I	XC1736DPD8C XC1736DPD8I XC1736DSO8C XC1736DSO8I XC1736DPC20C XC1736DPC20I XC1736DDD8M	XC1765DPD8C XC1765DPD8I XC1765DSO8C XC1765DSO8I XC1765DPC20C XC1765DPC20I XC1765DDD8M XC1765DDD8R	XC1718LPD8C XC1718LPD8I XC1718LSO8C XC1718LSO8I XC1718LPC20C XC1718LPC20I	XC1765LPD8C XC1765LPD8I XC1765LSO8C XC1765LSO8I XC1765LPC20C XC1765LPC20I XC1765LDD8M
--	---	--	--	---

X3181

Marking Information

Due to the small size of the serial PROM package, the complete ordering part number cannot be marked on the package. The XC prefix is deleted and the package code is simplified. Device marking is as follows.



X3182



XC3000, XC3000A, XC3000L, XC3100 Logic Cell Array Families

Product Description

Features

- Complete line of four related Field Programmable Gate Array product families.
 - XC3000, XC3000A, XC3000L, XC3100
- Ideal for a wide range of custom VLSI design tasks
 - Replaces TTL, MSI, and other PLD logic
 - Integrates complete sub-systems into a single package
 - Avoids the NRE, time delay, and risk of conventional masked gate arrays
- High-performance CMOS static memory technology
 - Guaranteed toggle rates of 70 to 270 MHz, logic delays from 9 to 3 ns
 - System clock speeds of up to 80 MHz
 - Low quiescent and active power consumption
- Flexible FPGA architecture
 - Compatible arrays ranging from 1,300 to 9,000 gate complexity
 - Extensive register, combinatorial, and I/O capabilities
 - High fan-out signal distribution, low-skew clock nets
 - Internal 3-state bus capabilities
 - TTL or CMOS input thresholds
 - On-chip crystal oscillator amplifier
- Unlimited reprogrammability
 - Easy design iteration
 - In-system logic changes
- Extensive Packaging Options
 - Over 20 different packages
 - Plastic and ceramic surface-mount and pin-grid-array packages
 - Thin and Very Thin Quad Flat Pack (TQFP and VQFP) options
- Ready for volume production
 - Standard, off-the-shelf product availability
 - 100% factory pre-tested devices
 - Excellent reliability record

- Complete XACT Development System
 - Schematic capture, automatic place and route
 - Logic and timing simulation
 - Interactive design editor for design optimization
 - Timing calculator
 - Interfaces to popular design environments like Viewlogic, Cadence, Mentor Graphics, and others

Description

The CMOS XC3000 Class of Logic Cell Array (LCA) families provide a group of high-performance, high-density, digital integrated circuits. Their regular, extendable, flexible, user-programmable array architecture is composed of a configuration program store plus three types of configurable elements: a perimeter of I/O Blocks (IOBs), a core array of Configurable Logic Blocks (CLBs) and resources for interconnection. The general structure of an LCA device is shown in Figure 1 on the next page. The XACT development system provides schematic capture and auto place-and-route for design entry. Logic and timing simulation, and in-circuit emulation are available as design verification alternatives. The design editor is used for interactive design optimization, and to compile the data pattern that represents the configuration program.

The LCA user logic functions and interconnections are determined by the configuration program data stored in internal static memory cells. The program can be loaded in any of several modes to accommodate various system requirements. The program data resides externally in an EEPROM, EPROM or ROM on the application circuit board, or on a floppy disk or hard disk. On-chip initialization logic provides for optional automatic loading of program data at power-up. The companion XC17XX Serial Configuration PROMs provide a very simple serial configuration program storage in a one-time programmable package.

Device	CLBs	Array	User I/Os Max	Flip-Flops	Horizontal Longlines	Configuration Data Bits
XC3020, 3020A, 3020L, 3120	64	8 x 8	64	256	16	14,779
XC3030, 3030A, 3030L, 3130	100	10 x 10	80	360	20	22,176
XC3042, 3042A, 3042L, 3142	144	12 x 12	96	480	24	30,784
XC3064, 3064A, 3064L, 3164	224	16 x 14	120	688	32	46,064
XC3090, 3090A, 3090L, 3190	320	16 x 20	144	928	40	64,160
XC3195	484	22 x 22	176	1,320	44	94,984

the length count compares, the user I/O pins become active. Options in the MakeBits program allow timing choices of one clock earlier or later for the timing of the end of the internal logic RESET and the assertion of the DONE signal. The open-drain DONE/ $\overline{\text{PROG}}$ output can be AND-tied with multiple LCA devices and used as an active-High READY, an active-Low PROM enable or a RESET to other portions of the system. The state diagram of Figure 18 illustrates the configuration process.

Master Mode

In Master mode, the LCA device automatically loads configuration data from an external memory device. There are three Master modes that use the internal timing source to supply the configuration clock (CCLK) to time the incoming data. Master Serial mode uses serial configuration data supplied to Data-in (DIN) from a synchronous serial source such as the Xilinx Serial Configuration PROM shown in Figure 21. Master Parallel Low and High modes automatically use parallel data supplied to the D0–D7 pins in response to the 16-bit address generated by the LCA device. Figure 22 shows an example of the parallel Master mode connections required. The LCA HEX starting address is 0000 and increments for Master Low mode and it is FFFF and decrements for Master High mode. These two modes provide address compatibility with microprocessors which begin execution from opposite ends of memory.

Peripheral Mode

Peripheral mode provides a simplified interface through which the device may be loaded byte-wide, as a processor peripheral. Figure 23 shows the peripheral mode connections. Processor write cycles are decoded from the common assertion of the active low Write Strobe (WS), and two active low and one active high Chip Selects ($\overline{\text{CS0}}$, $\overline{\text{CS1}}$, CS2). The LCA device generates a configuration clock from the internal timing generator and serializes the parallel input data for internal framing or for succeeding slaves on Data Out (DOUT). A output High on READY/BUSY pin indicates the completion of loading for each byte when the input register is ready for a new byte. As with Master modes, Peripheral mode may also be used as a lead device for a daisy-chain of slave devices.

Slave Serial Mode

Slave Serial mode provides a simple interface for loading the Logic Cell Array configuration as shown in Figure 24. Serial data is supplied in conjunction with a synchronizing input clock. Most Slave mode applications are in daisy-chain configurations in which the data input is driven from the previous Logic Cell Array's data out, while the clock is supplied by a lead device in Master or Peripheral mode. Data may also be supplied by a processor or other special circuits.

Daisy Chain

The XACT development system is used to create a composite configuration for selected LCA devices including: a preamble, a length count for the total bitstream, multiple concatenated data programs and a postamble plus an additional fill bit per device in the serial chain. After loading and passing-on the preamble and length count to a possible daisy-chain, a lead device will load its configuration data frames while providing a High DOUT to possible down-stream devices as shown in Figure 22. Loading continues while the lead device has received its configuration program and the current length count has not reached the full value. The additional data is passed through the lead device and appears on the Data Out (DOUT) pin in serial form. The lead device also generates the Configuration Clock (CCLK) to synchronize the serial output data and data in of down-stream LCA devices. Data is read in on DIN of slave devices by the positive edge of CCLK and shifted out the DOUT on the negative edge of CCLK. A parallel Master mode device uses its internal timing generator to produce an internal CCLK of 8 times its EPROM address rate, while a Peripheral mode device produces a burst of 8 CCLKs for each chip select and write-strobe cycle. The internal timing generator continues to operate for general timing and synchronization of inputs in all modes.

Special Configuration Functions

The configuration data includes control over several special functions in addition to the normal user logic functions and interconnect.

- Input thresholds
- Readback disable
- DONE pull-up resistor
- DONE timing
- RESET timing
- Oscillator frequency divided by two

Each of these functions is controlled by configuration data bits which are selected as part of the normal XACT development system bitstream generation process.

Input Thresholds

Prior to the completion of configuration all LCA device input thresholds are TTL compatible. Upon completion of configuration, the input thresholds become either TTL or CMOS compatible as programmed. The use of the TTL threshold option requires some additional supply current for threshold shifting. The exception is the threshold of the $\overline{\text{PWRDWN}}$ input and direct clocks which always have a CMOS input. Prior to the completion of configuration the user I/O pins each have a high impedance pull-up. The configuration program can be used to enable the IOB pull-

up resistors in the Operational mode to act either as an input load or to avoid a floating input on an otherwise unused pin.

Readback

The contents of a Logic Cell Array may be read back if it has been programmed with a bitstream in which the Readback option has been enabled. Readback may be used for verification of configuration and as a method of determining the state of internal logic nodes during debugging. There are three options in generating the configuration bitstream.

- "Never" inhibits the Readback capability.
- "One-time," inhibits Readback after one Readback has been executed to verify the configuration.
- "On-command" allows unrestricted use of Readback.

Readback is accomplished without the use of any of the user I/O pins; only M0, M1 and CCLK are used. The initiation of Readback is produced by a Low to High transition of the M0/RTRIG (Read Trigger) pin. The CCLK input must then be driven by external logic to read back the configuration data. The first three Low-to-High CCLK transitions clock out dummy data. The subsequent Low-to-High CCLK transitions shift the data frame information out on the M1/RDATA (Read Data) pin. Note that the logic polarity is always inverted, a zero in configuration becomes a one in Readback, and vice versa. Note also that each Readback frame has one Start bit (read back as a one) but, unlike in configuration, each Readback frame has only one Stop bit (read back as a zero). The third leading dummy bit mentioned above can be considered the Start bit of the first frame. All data frames must be read back to complete the process and return the Mode Select and CCLK pins to their normal functions.

Readback data includes the current state of each CLB flip-flop, each input flip-flop or latch, and each device pad. These data are imbedded into unused configuration bit positions during Readback. This state information is used by the XACT development system In-Circuit Verifier to provide visibility into the internal operation of the logic while the system is operating. To readback a uniform time-sample of all storage elements, it may be necessary to inhibit the system clock.

Reprogram

To initiate a re-programming cycle, the dual-function pin DONE/PROG must be given a High-to-Low transition. To reduce sensitivity to noise, the input signal is filtered for two

cycles of the LCA device internal timing generator. When reprogram begins, the user-programmable I/O output buffers are disabled and high-impedance pull-ups are provided for the package pins. The device returns to the Clear state and clears the configuration memory before it indicates 'initialized'. Since this Clear operation uses chip-individual internal timing, the master might complete the Clear operation and then start configuration before the slave has completed the Clear operation. To avoid this problem, the slave INIT pins must be AND-wired and used to force a RESET on the master (see Figure 22). Reprogram control is often implemented using an external open-collector driver which pulls DONE/PROG Low. Once a stable request is recognized, the DONE/PROG pin is held Low until the new configuration has been completed. Even if the re-program request is externally held Low beyond the configuration period, the LCA device will begin operation upon completion of configuration.

DONE Pull-up

DONE/PROG is an open-drain I/O pin that indicates the LCA device is in the operational state. An optional internal pull-up resistor can be enabled by the user of the XACT development system when MAKEBITS is executed. The DONE/PROG pins of multiple LCA devices in a daisy-chain may be connected together to indicate all are DONE or to direct them all to reprogram.

DONE Timing

The timing of the DONE status signal can be controlled by a selection in the MakeBits program to occur either a CCLK cycle before, or after, the outputs going active. See Figure 20. This facilitates control of external functions such as a PROM enable or holding a system in a wait state.

RESET Timing

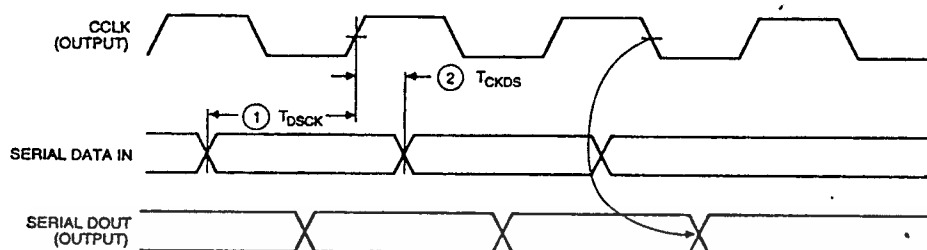
As with DONE timing, the timing of the release of the internal reset can be controlled by a selection in the MakeBits program to occur either a CCLK cycle before, or after, the outputs going active. See Figure 20. This reset keeps all user programmable flip-flops and latches in a zero state during configuration.

Crystal Oscillator Division

A selection in the MakeBits program allows the user to incorporate a dedicated divide-by-two flip-flop between the crystal oscillator and the alternate clock line. This guarantees a symmetrical clock signal. Although the frequency stability of a crystal oscillator is very good, the symmetry of its waveform can be affected by bias or feedback drive.

The following seven pages describe the different configuration modes in detail

Master Serial Mode Programming Switching Characteristics



	Speed Grade		Min	Max	Units
	Description	Symbol			
CCLK3	Data In setup	1 T_{DSCK}	60		ns
	Data In hold	2 T_{CKDS}	0		ns

- Notes:
1. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min in less than 25 ms. If this is not possible, configuration can be delayed by holding RESET Low until V_{CC} has reached 4.0 V (2.5 V for the XC3000L). A very long V_{CC} rise time of >100 ms, or a non-monotonically rising V_{CC} may require >1- μ s High level on RESET, followed by a >6- μ s Low level on RESET and D/P after V_{CC} has reached 4.0 V (2.5 V for the XC3000L).
 2. Configuration can be controlled by holding RESET Low with or until after the INIT of all daisy-chain slave-mode devices is High.
 3. Master-serial-mode timing is based on slave-mode testing.

Master Parallel Mode

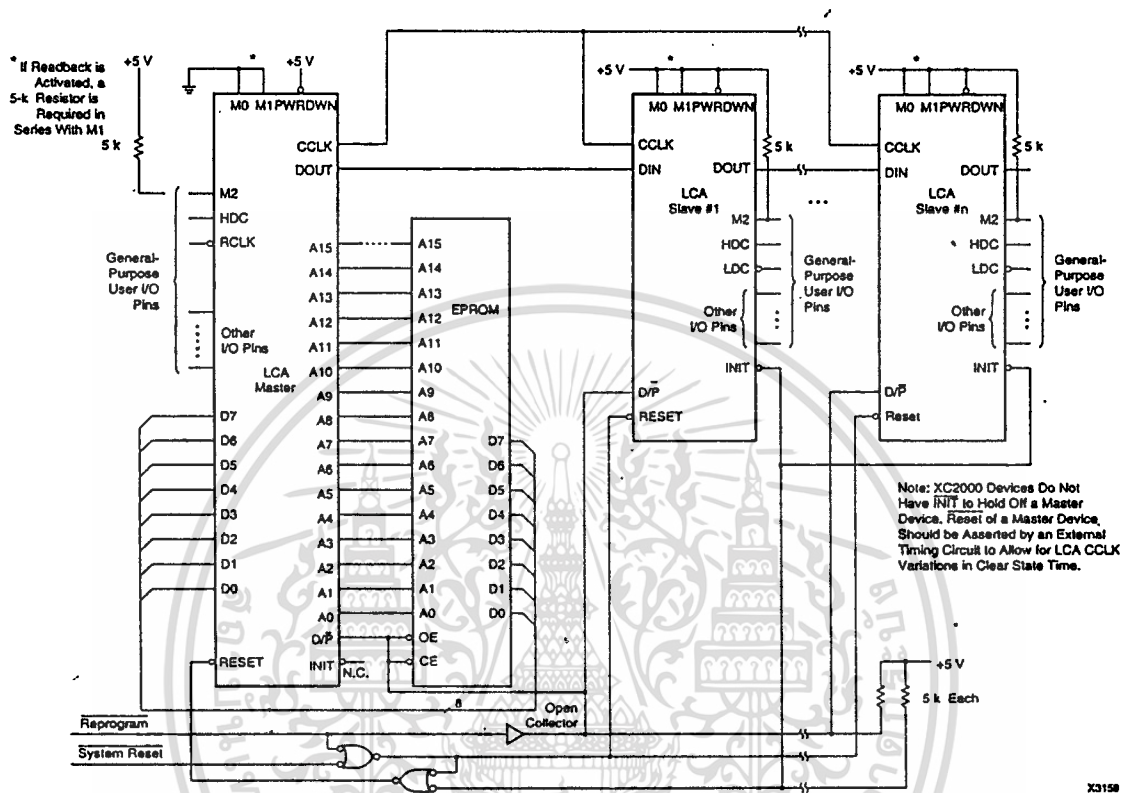


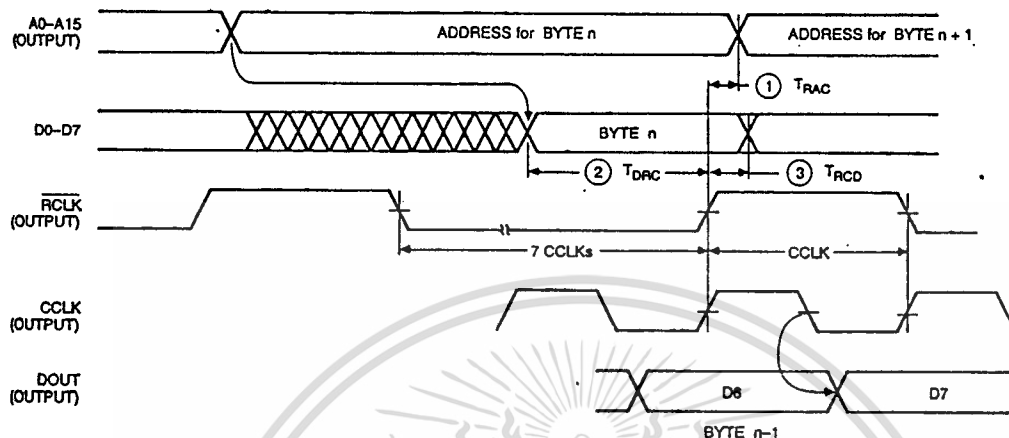
Figure 22. Master Parallel Mode

In Master Parallel mode, the lead LCA device directly addresses an industry-standard byte-wide EPROM and accepts eight data bits right before incrementing (or decrementing) the address outputs.

The eight data bits are serialized in the lead LCA device, which then presents the preamble data (and all data that overflows the lead device) on the DOUT pin. There is an

internal delay of 1.5 CCLK periods, after the rising CCLK edge that accepts a byte of data, and also changes the EPROM address, until the falling CCLK edge that makes the LSB (D0) of this byte appear at DOUT. This means that DOUT changes on the falling CCLK edge, and the next LCA device in the daisy chain accepts data on the subsequent rising CCLK edge.

Master Parallel Mode Programming Switching Characteristics

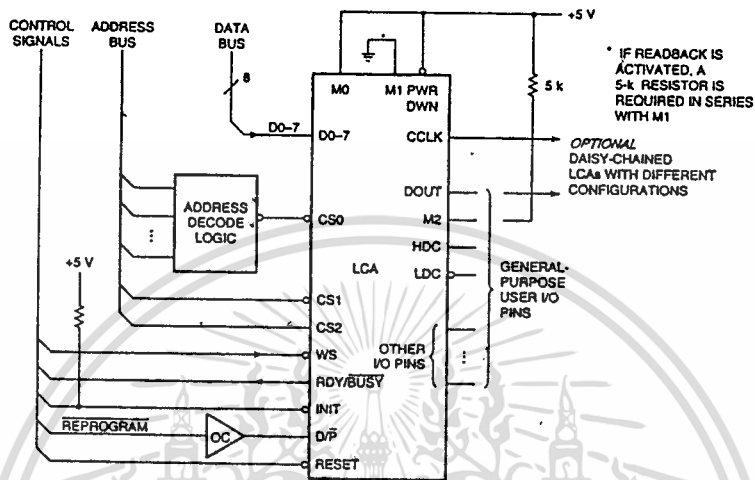


	Description	Symbol		Min	Max	Units
RCLK	To address valid	1	T_{RAC}	0	200	ns
	To data setup	2	T_{DRG}	60		ns
	To data hold	3	T_{RCD}	0		ns
	RCLK High		T_{RCH}	600		ns
	RCLK Low		T_{RCL}	4.0		μ s

- Notes: 1. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min in less than 25 ms. If this is not possible, configuration can be delayed by holding $\overline{RESET}$ Low until V_{CC} has reached 4.0 V (2.5 V for the XC3000L). A very long V_{CC} rise time of >100 ms, or a non-monotonically rising V_{CC} may require a >1- μ s High level on $\overline{RESET}$, followed by a >6- μ s Low level on $\overline{RESET}$ and D/P after V_{CC} has reached 4.0 V (2.5 V for the XC3000L).
2. Configuration can be controlled by holding $\overline{RESET}$ Low with or until after the INIT of all daisy-chain slave-mode devices is High.

*This timing diagram shows that the EPROM requirements are extremely relaxed:
EPROM access time can be longer than 4000 ns, EPROM data output has no hold time requirements.*

Peripheral Mode



XC3031

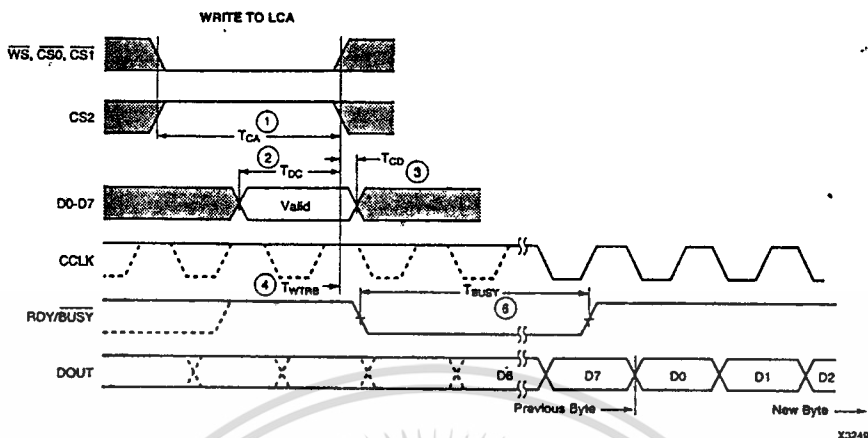
Peripheral mode uses the trailing edge of the logic AND condition of the CS0, CS1, CS2, and WS inputs to accept byte-wide data from a microprocessor bus. In the lead LCA device, this data is loaded into a double-buffered UART-like parallel-to-serial converter and is serially shifted into the internal logic. The lead LCA device presents the preamble data (and all data that overflows the lead device) on the DOUT pin.

The Ready/Busy output from the lead LCA device acts as a handshake signal to the microprocessor. RDY/BUSY goes Low when a byte has been received, and goes High

again when the byte-wide input buffer has transferred its information into the shift register, and the buffer is ready to receive new data. The length of the BUSY signal depends on the activity in the UART. If the shift register had been empty when the new byte was received, the BUSY signal lasts for only two CCLK periods. If the shift register was still full when the new byte was received, the BUSY signal can be as long as nine CCLK periods.

Note that after the last byte has been entered, only seven of its bits are shifted out. CCLK remains High with DOUT equal to bit 6 (the next-to-last bit) of the last byte entered.

Peripheral Mode Programming Switching Characteristics



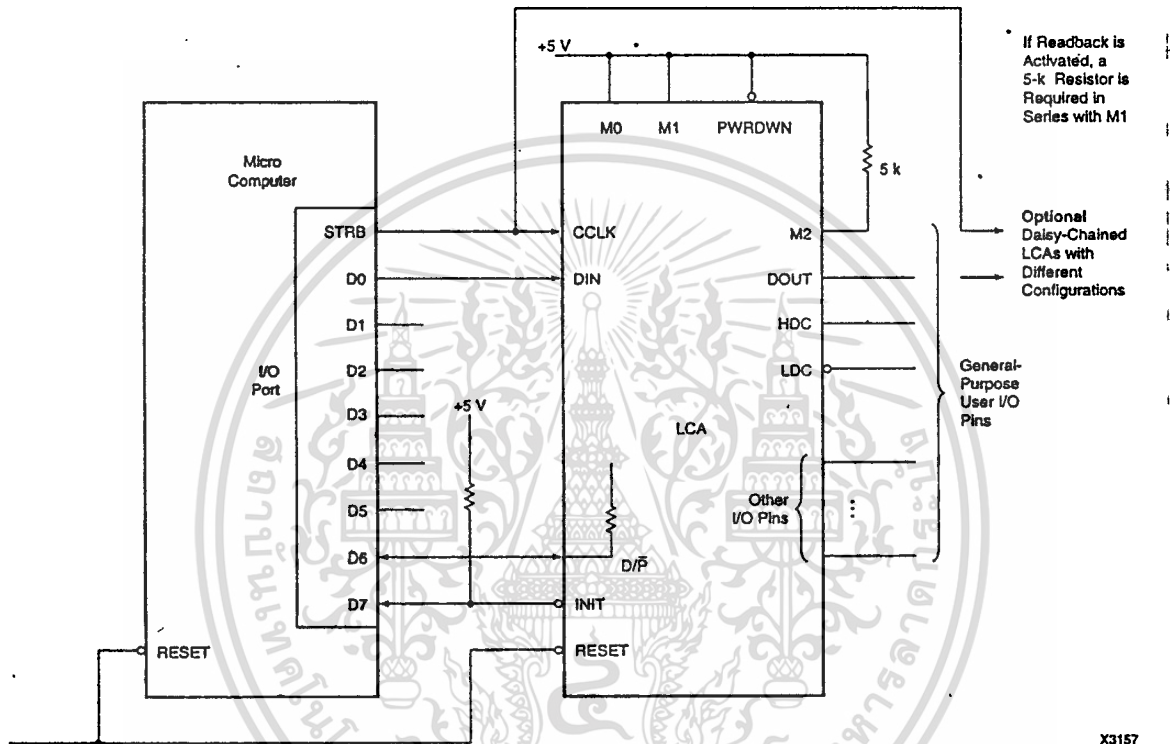
	Description	Symbol	Min	Max	Units
Write	Effective Write time required (Assertion of $\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, $\overline{WS}$)	1 T_{CA}	100		ns
	DIN Setup time required	2 T_{DC}	60		ns
	DIN Hold time required	3 T_{CD}	0		ns
	RDY/BUSY delay after end of $\overline{WS}$	4 T_{WTRB}		60	ns
RDY	Earliest next $\overline{WS}$ after end of $\overline{BUSY}$	5 T_{RBWT}	0		ns
	$\overline{BUSY}$ Low time generated	6 T_{BUSY}	2.5	9	CCLK Periods

Notes:

- At power-up, V_{CC} must rise from 2.0 V to V_{CC} min in less than 25 ms. If this is not possible, configuration can be delayed by holding RESET Low until V_{CC} has reached 4.0 V (2.5 V for the XC3000L). A very long V_{CC} rise time of >100 ms, or a non-monotonically rising V_{CC} may require a >1- μ s High level on RESET, followed by a >6- μ s Low level on RESET and D/P after V_{CC} has reached 4.0 V (2.5 V for the XC3000L).
- Configuration must be delayed until the INIT of all LCAs is High.
- Time from end of $\overline{WS}$ to CCLK cycle for the new byte of data depends on completion of previous byte processing and the phase of the internal timing generator for CCLK.
- CCLK and DOUT timing is tested in slave mode.
- T_{BUSY} indicates that the double-buffered parallel-to-serial converter is not yet ready to receive new data. The shortest T_{BUSY} occurs when a byte is loaded into an empty parallel-to-serial converter. The longest T_{BUSY} occurs when a new word is loaded into the input register before the second-level buffer has started shifting out data.

This timing diagram shows very relaxed requirements: Data need not be held beyond the rising edge of $\overline{WS}$. $\overline{BUSY}$ will go active within 60 ns after the end of $\overline{WS}$. $\overline{BUSY}$ will stay active for several microseconds. $\overline{WS}$ may be asserted immediately after the end of $\overline{BUSY}$.

Slave Serial Mode



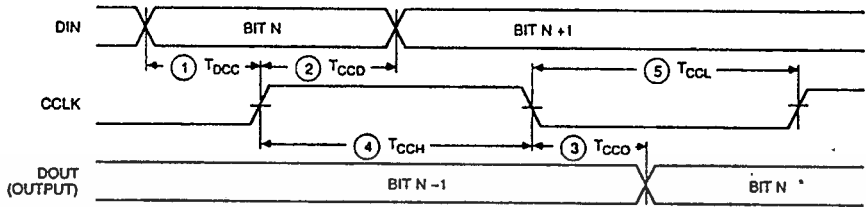
X3157

Figure 24. Slave Serial Mode.

In Slave Serial mode, an external signal drives the CCLK input(s) of the LCA device(s). The serial configuration bitstream must be available at the DIN input of the lead LCA device a short set-up time before each rising CCLK edge. The lead LCA device then presents the preamble

data (and all data that overflows the lead device) on its DOUT pin. There is an internal delay of 1.5 CCLK periods, which means that DOUT changes on the falling CCLK edge, and the next LCA device in the daisy-chain accepts data on the subsequent rising CCLK edge.

Slave Serial Mode Programming Switching Characteristics

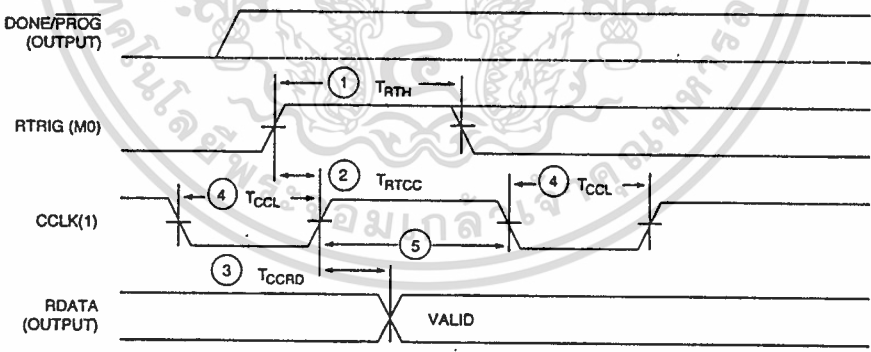


1106 31

	Description	Symbol	Min	Max	Units
CCLK	To DOUT	3 T_{CCO}		100	ns
	DIN setup	1 T_{DCC}	60		ns
	DIN hold	2 T_{CCD}	0		ns
	High time	4 T_{CCH}	0.05		μ s
	Low time (Note 1)	5 T_{CCL}	0.05	5.0	μ s
	Frequency	F_{CC}		10	MHz

- Notes:
1. The max limit of CCLK Low time is caused by dynamic circuitry inside the LCA device.
 2. Configuration must be delayed until the INIT of all LCA devices is High.
 3. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min in less than 25 ms. If this is not possible, configuration can be delayed by holding RESET Low until V_{CC} has reached 4.0 V (2.5 V for the XC3000L). A very long V_{CC} rise time of >100 ms, or a non-monotonically rising V_{CC} may require a >1- μ s High level on RESET, followed by a >6- μ s Low level on RESET and D/P after V_{CC} has reached 4.0 V (2.5 V for the XC3000L).

Program Readback Switching Characteristics

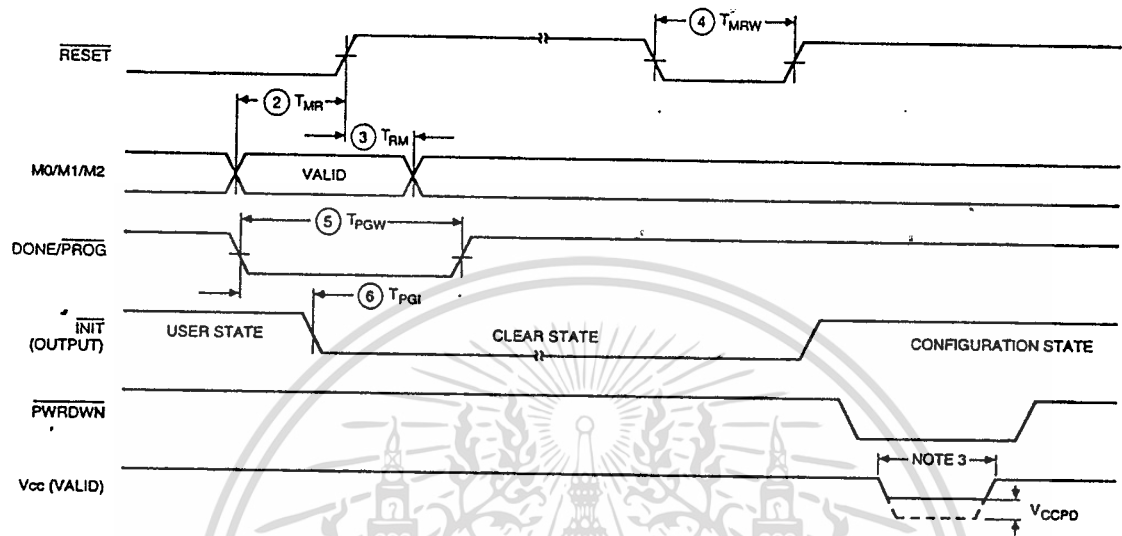


X3028

	Description	Symbol	Min	Max	Units
RTRIG	RTRIG High	1 T_{RTH}	250		ns
CCLK	RTRIG setup	2 T_{RTCC}	200		ns
	RDATA delay	3 T_{CCRD}		100	ns
	High time	5 T_{CCHR}	0.5		μ s
	Low time	4 T_{CCLR}	0.5	5	μ s

- Notes:
1. During Readback, CCLK frequency may not exceed 1 MHz.
 2. RETRIG (M0 positive transition) shall not be done until after one clock following active I/O pins.
 3. Readback should not be initiated until configuration is complete.
 4. T_{CCLR} is 5 μ s min to 15 μ s max for XC3000L.

General LCA Switching Characteristics



1105 20

Description		Symbol	XC3000 XC3100		XC3000A XC3000L		Units
			Min	Max	Min	Max	
RESET (2)	M0, M1, M2 setup time required	2	T_{MR}	1	1		μs
	M0, M1, M2 hold time required	3	T_{RM}	1	3		μs
	RESET Width (Low) req. for Abort	4	T_{MRW}	6	6		μs
DONE/PROG	Width (Low) required for Re-config.	5	T_{PGW}	6	6		μs
	INIT response after D/P is pulled Low	6	T_{PGI}	7	7		μs
PWRDWN (3)	Power Down V_{CC}		V_{CCPD}	2.3	2.3		V

- Notes: 1. At power-up, V_{CC} must rise from 2.0 V to $V_{CC\ min}$ in less than 25 ms. If this is not possible, configuration can be delayed by holding RESET Low until V_{CC} has reached 4.0 V (2.5 V for XC3000L). A very long V_{CC} rise time of >100 ms, or a non-monotonically rising V_{CC} may require a >1- μs High level on RESET, followed by a >6- μs Low level on RESET and D/P after V_{CC} has reached 4.0 V (2.5 V for XC3000L).
2. RESET timing relative to valid mode lines (M0, M1, M2) is relevant when RESET is used to delay configuration.
3. PWRDWN transitions must occur while $V_{CC} > 4.0$ V (2.5 V for XC3000L).

Performance

Device Performance

The XC3000 families of FPGAs can achieve very high performance. This is the result of

- A sub-micron manufacturing process, developed and continuously being enhanced for the production of state-of-the-art CMOS SRAMs.
- Careful optimization of transistor geometries, circuit design, and lay-out, based on years of experience with the XC3000 family.
- A look-up table based, coarse-grained architecture that can collapse multiple-layer combinatorial logic into a single function generator. One CLB can implement up to four layers of conventional logic in as little as 2.7 ns.

Actual system performance is determined by the timing of critical paths, including the delay through the combinatorial and sequential logic elements within CLBs and IOBs, plus the delay in the interconnect routing. The ac-timing specifications state the worst-case timing parameters for the various logic resources available in the XC3000-families architecture. Figure 25 shows a variety of elements involved in determining system performance.

Logic block performance is expressed as the propagation time from the interconnect point at the input to the block to the output of the block in the interconnect area. Since combinatorial logic is implemented with a memory lookup table within a CLB, the combinatorial delay through the CLB, called T_{ILO} , is always the same, regardless of the function being implemented. For the combinatorial logic function driving the data input of the storage element, the critical timing is data set-up relative to the clock edge provided to the flip-flop element. The delay from the clock source to the output of the logic block is critical in the timing signals produced by storage elements. Loading of a logic-

block output is limited only by the resulting propagation delay of the larger interconnect network. Speed performance of the logic block is a function of supply voltage and temperature. See Figure 26.

Interconnect performance depends on the routing resources used to implement the signal path. Direct interconnects to the neighboring CLB provide an extremely fast path. Local interconnects go through switch matrices (magic boxes) and suffer an RC delay, equal to the resistance of the pass transistor multiplied by the capacitance of the driven metal line. Longlines carry the signal across the length or breadth of the chip with only one access delay. Generous on-chip signal buffering makes performance relatively insensitive to signal fan-out; increasing fan-out from 1 to 8 changes the CLB delay by only 10%. Clocks can be distributed with two low-skew clock distribution networks.

The tools in the XACT Development System used to place and route a design in an XC3000 FPGA (the Automatic Place and Route [APR] program and the XACT Design Editor) automatically calculate the actual maximum worst-case delays along each signal path. This timing information can be back-annotated to the design's netlist for use in timing simulation or examined with X-DELAY, a static timing analyzer.

Actual system performance is applications dependent. The maximum clock rate that can be used in a system is determined by the critical path delays within that system. These delays are combinations of incremental logic and routing delays, and vary from design to design. In a synchronous system, the maximum clock rate depends on the number of combinatorial logic layers between re-synchronizing flip-flops. Figure 27 shows the achievable clock rate as a function of the number of CLB layers.

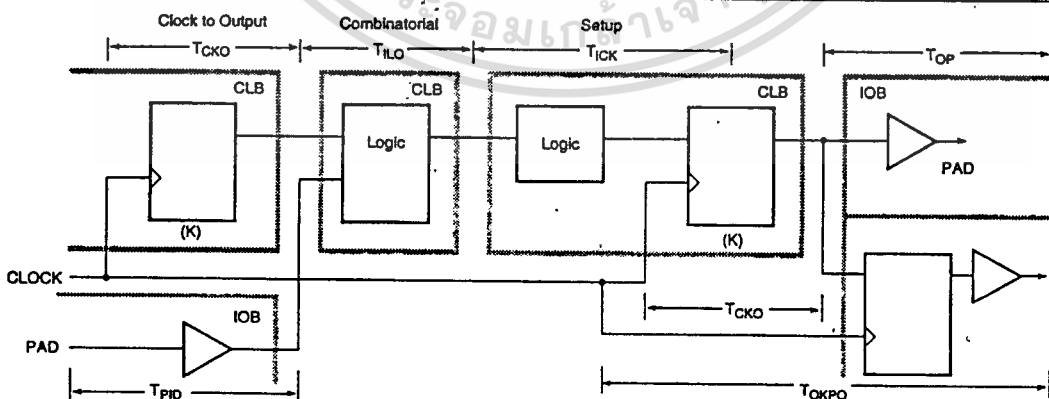


Figure 25. Primary Block Speed Factors. Actual timing is a function of various block factors combined with routing factors. Overall performance can be evaluated with the XACT timing calculator or by an optional simulation.

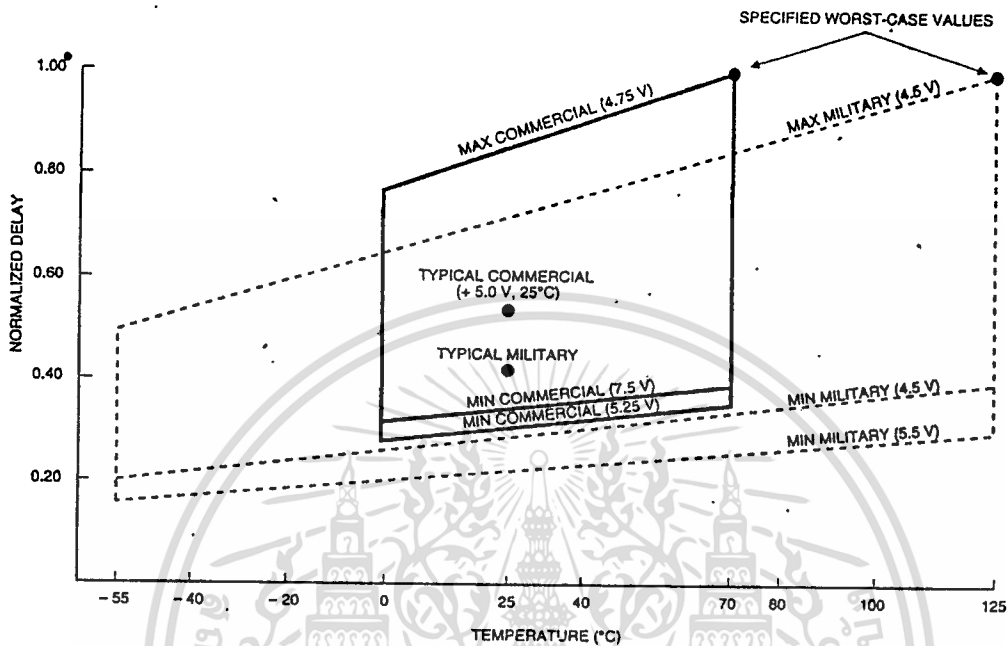
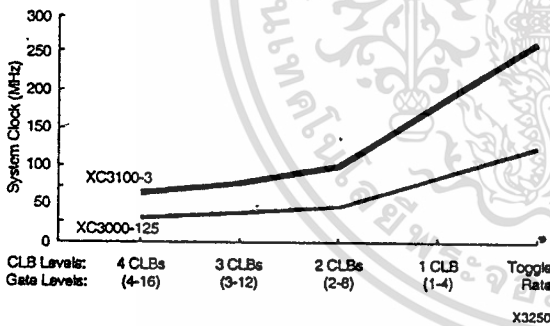


Figure 26. Relative Delay as a Function of Temperature, Supply Voltage and Processing Variations

X1045



Power

Power Distribution

Power for the LCA device is distributed through a grid to achieve high noise immunity and isolation between logic and I/O. Inside the LCA device, a dedicated V_{CC} and ground ring surrounding the logic array provides power to the I/O drivers. An independent matrix of V_{CC} and groundlines supplies the interior logic of the device. This power distribution grid provides a stable supply and ground for all internal logic, providing the external package power pins are all connected and appropriately decoupled. Typically a 0.1- μ F capacitor connected near the V_{CC} and ground pins will provide adequate decoupling.

Output buffers capable of driving the specified 4- or 8-mA loads under worst-case conditions may be capable of driving as much as 25 to 30 times that current in a best case. Noise can be reduced by minimizing external load capacitance and reducing simultaneous output transitions in the same direction. It may also be beneficial to locate heavily loaded output buffers near the ground pads. The I/O Block output buffers have a slew-limited mode which should be used where output rise and fall times are not speed critical. Slew-limited outputs maintain their dc drive capability, but generate less external reflections and internal noise.

Figure 27. Clock Rate as a Function of Logic Complexity (Number of Combinational Levels between Flip-Flops)

X3250

Dynamic Power Consumption

	XC3042	XC3042A	XC3042L	
One CLB driving 3 local interconnects	0.25	0.17	0.07	mW per MHz
One global clock buffer and clock line	2.25	1.40	0.50	mW per MHz
One device output with a 50 pF load	1.40	1.40	0.70	mW per MHz

Power Consumption

The Logic Cell Array exhibits the low power consumption characteristic of CMOS ICs. For any design, the configuration option of TTL chip input threshold requires power for the threshold reference. The power required by the static memory cells that hold the configuration data is very low and may be maintained in a power-down mode.

Typically, most of power dissipation is produced by external capacitive loads on the output buffers. This load and frequency dependent power is $25 \mu\text{W/pF/MHz}$ per output. Another component of I/O power is the external dc loading on all output pins.

Internal power dissipation is a function of the number and size of the nodes, and the frequency at which they change. In an LCA device, the fraction of nodes changing on a given clock is typically low (10-20%). For example, in a long binary counter, the total activity of all counter flip-flops is equivalent to that of only two CLB outputs toggling at the clock frequency. Typical global clock-buffer power is between 2.0 mW/MHz for the XC3020 and 3.5 mW/MHz for the XC3090. The internal capacitive load is more a function of interconnect than fan-out. With a typical load of three general interconnect segments, each CLB output requires about 0.25 mW per MHz of its output frequency.

Because the control storage of the Logic Cell Array is CMOS static memory, its cells require a very low standby current for data retention. In some systems, this low data retention current characteristic can be used as a method of preserving configurations in the event of a primary

power loss. The Logic Cell Array has built in power-down logic which, when activated, will disable normal operation of the device and retain only the configuration data. All internal operation is suspended and output buffers are placed in their high-impedance state with no pull-ups. Different from the XC3000 family which can be powered down to a current consumption of a few microamps, the XC3100 draws 5 mA, even in power-down. This makes power-down operation less meaningful. In contrast, I_{CCPD} for the XC3000L is only 10 μA .

To force the Logic Cell Array into the Power-Down state, the user must pull the PWRDWN pin Low and continue to supply a retention voltage to the V_{CC} pins. When normal power is restored, V_{CC} is elevated to its normal operating voltage and PWRDWN is returned to a High. The Logic Cell Array resumes operation with the same internal sequence that occurs at the conclusion of configuration. Internal-I/O and logic-block storage elements will be reset, the outputs will become enabled and the DONE/PROG pin will be released.

When V_{CC} is shut down or disconnected, some power might unintentionally be supplied from an incoming signal driving an I/O pin. The conventional electro-static input protection is implemented with diodes to the supply and ground. A positive voltage applied to an input (or output) will cause the positive protection diode to conduct and drive the V_{CC} connection. This condition can produce invalid power conditions and should be avoided. A large series resistor might be used to limit the current or a bipolar buffer may be used to isolate the input signal.

Pin Descriptions

Permanently Dedicated Pins.

V_{CC}

Two to eight (depending on package type) connections to the positive V supply voltage. All must be connected.

GND

Two to eight (depending on package type) connections to ground. All must be connected.

PWRDWN

A Low on this CMOS-compatible input stops all internal activity, but retains configuration. All flip-flops and latches are reset, all outputs are 3-stated, and all inputs are interpreted as High, independent of their actual level. When PWRDWN returns High, the LCA device becomes operational with DONE Low for two cycles of the internal 1-MHz clock. During configuration, PWRDWN must be High. If not used, PWRDWN must be tied to V_{CC} .

RESET

This is an active Low input which has three functions.

Prior to the start of configuration, a Low input will delay the start of the configuration process. An internal circuit senses the application of power and begins a minimal time-out cycle. When the time-out and RESET are complete, the levels of the M lines are sampled and configuration begins.

If RESET is asserted during a configuration, the LCA device is re-initialized and restarts the configuration at the termination of RESET.

If RESET is asserted after configuration is complete, it provides a global asynchronous RESET of all IOB and CLB storage elements of the LCA device.

CCLK

During configuration, Configuration Clock is an output of an LCA device in Master mode or Peripheral mode, but an input in Slave mode. During Readback, CCLK is a clock input for shifting configuration data out of the LCA device.

CCLK drives dynamic circuitry inside the LCA device. The Low time may, therefore, not exceed a few microseconds. When used as an input, CCLK must be "parked High". An internal pull-up resistor maintains High when the pin is not being driven.

DONE/PROG (D/P)

DONE is an open-drain output, configurable with or without an internal pull-up resistor. At the completion of configuration, the LCA device circuitry becomes active in a synchronous order; DONE is programmed to go active High one cycle either before or after the outputs go active.

Once configuration is done, a High-to-Low transition of this pin will cause an initialization of the LCA device and start a reconfiguration.

M0/RTRIG

As Mode 0, this input is sampled on power-on to determine the power-on delay (2^{14} cycles if M0 is Low, 2^{16} cycles if M0 is High). Before the start of configuration, this input is again sampled together with M1, M2 to determine the configuration mode to be used.

A Low-to-High input transition, after configuration is complete, acts as a Read Trigger and initiates a Readback of configuration and storage-element data clocked by CCLK. By selecting the appropriate Readback option when generating the bitstream, this operation may be limited to a single Readback, or be inhibited altogether.

M1/RDATA

As Mode 1, this input and M0, M2 are sampled before the start of configuration to establish the configuration mode to be used. If Readback is never used, M1 can be tied directly to ground or V_{CC} . If Readback is ever used, M1 must use a 5-k Ω resistor to ground or V_{CC} to accommodate the RDATA output.

As an active-Low Read Data, after configuration is complete, this pin is the output of the Readback data.

User I/O Pins that can have special functions.

M2

During configuration, this Input has a weak pull-up resistor. Together with M0 and M1, it is sampled before the start of configuration to establish the configuration mode to be used. After configuration, this pin is a user-programmable I/O pin.

HDC

During configuration, this output is held at a High level to indicate that configuration is not yet complete. After configuration, this pin is a user-programmable I/O pin.

LDC

During Configuration, this output is held at a Low level to indicate that the configuration is not yet complete. After configuration, this pin is a user-programmable I/O pin. LDC is particularly useful in Master mode as a Low enable for an EPROM, but it must then be programmed as a High after configuration.

INIT

This is an active Low open-drain output with a weak pull-up and is held Low during the power stabilization and internal clearing of the configuration memory. It can be used to indicate status to a configuring microprocessor or, as a wired AND of several slave mode devices, a hold-off signal for a master mode device. After configuration this pin becomes a user-programmable I/O pin.

BCLKIN

This is a direct CMOS level input to the alternate clock buffer (Auxiliary Buffer) in the lower right corner.

XTL1

This user I/O pin can be used to operate as the output of an amplifier driving an external crystal and bias circuitry.

XTL2

This user I/O pin can be used as the input of an amplifier connected to an external crystal and bias circuitry. The I/O Block is left unconfigured. The oscillator configuration is activated by routing a net from the oscillator buffer symbol output and by the MakeBits program.

CS0, CS1, CS2, WS

These four inputs represent a set of signals, three active Low and one active High, that are used to control configuration-data entry in the Peripheral mode. Simultaneous assertion of all four inputs generates a Write to the internal data buffer. The removal of any assertion clocks in the D0-D7 data. In Master-Parallel mode, WS and CS2 are the A0 and A1 outputs. After configuration, these pins are user-programmable I/O pins.

RCLK

During Master parallel mode configuration RCLK represents a "read" of an external dynamic memory device (normally not used). After configuration is complete, this pin becomes a user-programmed I/O pin.

RDY/BUSY

During Peripheral parallel mode configuration this pin indicates when the chip is ready for another byte of data to be written to it. After configuration is complete, this pin becomes a user-programmed I/O pin.

D0-D7

This set of eight pins represents the parallel configuration byte for the parallel Master and Peripheral modes. After configuration is complete, they are user-programmed I/O pins.

A0-A15

During Master Parallel mode, these 16 pins present an address output for a configuration EPROM. After configuration, they are user-programmable I/O pins.

DIN

During Slave or Master Serial configuration, this pin is used as a serial-data input. In the Master or Peripheral configuration, this is the Data 0 input. After configuration is complete, this pin becomes a user-programmed I/O pin.

DOUT

During configuration this pin is used to output serial-configuration data to the DIN pin of a daisy-chained slave. After configuration is complete, this pin becomes a user-programmed I/O pin.

TCLKIN

This is a direct CMOS-level input to the global clock buffer. This pin can also be configured as a user programmable I/O pin. However, since TCLKIN is the preferred input to the global clock net, and the global clock net should be used as the primary clock source, this pin is usually the clock input to the chip.

Unrestricted User I/O Pins.

I/O

An I/O pin may be programmed by the user to be an Input or an Output pin following configuration. All unrestricted I/O pins, plus the special pins mentioned on the following page, have a weak pull-up resistor of 50 kΩ to 100 kΩ that becomes active as soon as the device powers up, and stays active until the end of configuration.

Configuration Pin Assignments

Configuration Mode <M2:M1:M0>					---	68	84	84	100	100	132	160	175	208	----	User
SLAVE <1:1:1>	MASTER-SER <0:0:0>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:0:1>	MASTER-LOW <1:0:0>	44 PLCC	PLCC	PLCC	PGA	POFP	TOFP	PGA	POFP	PGA	POFP	----	Operation
PWRDOWN (I)	PWRDOWN (I)	PWRDOWN (I)	PWRDOWN (I)	PWRDOWN (I)	7	10	12	B2	29	26	A1	159	B2	3		PWRDOWN (I)
VCC	VCC	VCC	VCC	VCC	12	18	22	F3	41	38	C8	20	D9	28		VCC
M1 (HIGH) (I)	M1 (LOW) (I)	M1 (LOW) (I)	M1 (HIGH) (I)	M1 (LOW) (I)	16	25	31	J2	52	49	B13	40	B14	48		RDATA
M0 (HIGH) (I)	M0 (LOW) (I)	M0 (HIGH) (I)	M0 (LOW) (I)	M0 (HIGH) (I)	17	26	32	L1	54	51	A14	42	B15	50		RTRIG (I)
M2 (HIGH) (I)	M2 (LOW) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	18	27	33	K2	56	53	C13	44	C15	56		IO
HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	19	28	34	K3	57	54	B14	45	E14	57		IO
LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	20	30	36	L3	59	56	D14	49	D16	81		IO
INIT*	INIT*	INIT*	INIT*	INIT*	22	34	42	K8	65	62	G14	59	H15	77		IO
GND	GND	GND	GND	GND	23	35	43	J8	66	63	H12	19	J14	79		GND
RESET (I)	RESET (I)	RESET (I)	RESET (I)	RESET (I)	26	43	53	L11	76	73	M13	76	P15	100		XTL2 OR VO
DONE	DONE	DONE	DONE	DONE	27	44	54	K10	78	75	P14	78	R15	102		RESET (I)
		DATA 7 (I)	DATA 7 (I)	DATA 7 (I)	28	45	55	J10	80	77	N13	80	R14	107		PROGRAM (I)
					46	56	K11	81	78	M12	81	N13	109			IO
		DATA 6 (I)	DATA 6 (I)	DATA 6 (I)	30	47	57	J11	82	79	P13	82	T14	110		XTL1 OR VO
		DATA 5 (I)	DATA 5 (I)	DATA 5 (I)	48	58	H10	83	80	N11	86	P12	115			IO
		DATA 4 (I)	DATA 4 (I)	DATA 4 (I)	49	60	F10	87	84	M9	92	T11	122			IO
		CS0 (I)			50	61	G10	88	85	N9	93	R10	123			IO
		DATA 3 (I)	DATA 3 (I)	DATA 3 (I)	51	62	G11	89	86	N8	96	R9	126			IO
VCC	VCC	VCC	VCC	VCC	34	52	F9	01	88	M8	100	N9	130			Vcc
		DATA 2 (I)	DATA 2 (I)	DATA 2 (I)	53	65	F11	82	89	N7	102	P8	132			IO
		CS1 (I)			54	66	E11	93	90	P6	103	R8	133			IO
		DATA 1 (I)	DATA 1 (I)	DATA 1 (I)	55	67	E10	94	91	M8	108	R7	138			IO
		RDY/BUSY	RCLK	RCLK	56	70	D10	96	95	M5	114	R5	145			IO
DIN (I)	DIN (I)	DATA 0 (I)	DATA 0 (I)	DATA 0 (I)	57	71	C11	99	96	N4	115	P5	146			IO
DOUT	DOUT	DOUT	DOUT	DOUT	38	58	72	B11	100	97	N2	119	R3	151		IO
CCLK (I)	CCLK (I)	CCLK (I)	CCLK (I)	CCLK (I)	39	59	73	C10	1	98	M3	120	N4	152		IO
		WS (I)	A0	A0	40	60	74	A11	2	99	P1	121	R2	153		CCLK (I)
		CS2 (I)	A1	A1	61	75	B10	5	2	M2	124	P2	161			IO
			A2	A2	62	76	B9	6	3	N1	125	M3	162			IO
			A3	A3	63	77	A10	8	5	L2	128	P1	165			IO
			A4	A4	64	78	A9	9	6	L1	129	N1	166			IO
			A15	A15	65	81	B6	12	9	K1	132	M1	172			IO
			A4	A4	66	82	B7	13	10	J2	133	L2	173			IO
			A14	A14	67	83	A7	14	11	H1	136	K2	178			IO
			A5	A5	68	84	C7	15	12	H2	137	K1	179			IO
GND	GND	GND	GND	GND	1	1	1	C6	16	13	H3	139	J3	182		GND
			A13	A13	2	2	A6	17	14	G2	141	H2	184			IO
			A6	A6	3	3	A5	18	15	G1	142	H1	185			IO
			A12	A12	4	4	B5	19	16	F2	147	F2	192			IO
			A7	A7	5	5	C5	20	17	E1	148	E1	193			IO
			A11	A11	6	8	A3	23	20	D1	151	D1	199			IO
			A8	A8	7	9	A2	24	21	D2	152	C1	200			IO
			A10	A10	8	10	B3	25	22	B1	155	E3	203			IO
			A9	A9	9	11	A1	26	28	C2	156	C2	204			IO
					X	X	X	X	X							XC3020 etc.
					X	X	X	X	X	X						XC3030 etc.
						X	X	X	X	X						XC3042 etc.
						X					X					XC3084 etc.
						X						X	X	X		XC3090 etc.
						X						X	X	X		XC3195

Represents a 58-kΩ to 100-kΩ pull-up

INIT* is an open drain output during configuration

(I) Represents an input

Pin assignment for the XC3064/XC3090 and XC3195 differ from those shown. See pages 2-135.

Paraphernal mode and master parallel mode are not supported in the PC44 package. See page 2-133.

Pin assignments for the XC3195 PQ208 differ from those shown. See page 2-142.

Note: Pin assignments of PGA Footprint PLCC sockets and PGA packages are not electrically identical. Generic I/O pins are not shown.

XC3000, XC3000A, XC3000L, XC3100 Logic Cell Array Families

XC3000 Families 68-Pin PLCC, 84-Pin PLCC and PGA Pinouts

XC3000, XC3000A, XC3000L and XC3100 families have identical pinouts

68 PLCC		XC3020 XC3030, XC3042	84 PLCC	84 PGA
XC3030	XC3020			
10	10	PWRDN	12	B2
11	11	TCLKIN-I/O	13	C2
12	—	I/O*	14	B1
13	12	I/O	15	C1
14	13	I/O	16	D2
—	—	I/O	17	D1
15	14	I/O	18	E3
16	15	I/O	19	E2
—	16	I/O	20	E1
17	17	I/O	21	F2
18	18	VCC	22	F3
19	19	I/O	23	G3
—	—	I/O	24	G1
20	20	I/O	25	G2
—	21	I/O	26	F1
21	22	I/O	27	H1
22	—	I/O	28	H2
23	23	I/O	29	J1
24	24	I/O	30	K1
25	25	M1-RDATA	31	J2
26	26	M0-RTRIG	32	L1
27	27	M2-I/O	33	K2
28	28	HDC-I/O	34	K3
29	29	I/O	35	L2
30	30	LDC-I/O	36	L3
—	31	I/O	37	K4
—	—	I/O*	38	L4
31	32	I/O	39	J5
32	33	I/O	40	K5
33	—	I/O*	41	L5
34	34	INIT-I/O	42	K6
35	35	GND	43	J6
36	36	I/O	44	J7
37	37	I/O	45	L7
38	38	I/O	46	K7
39	39	I/O	47	L6
—	40	I/O	48	L8
—	41	I/O	49	K8
40	—	I/O*	50	L9
41	—	I/O*	51	L10
42	42	I/O	52	K9
43	43	XTL2(IN)-I/O	53	L11

68 PLCC XC3030 XC3020	XC3020 XC3030, XC3042	84 PLCC	84 PGA
44	RESET	54	K10
45	DONE-PG	55	J10
46	D7-I/O	56	K11
47	XTL1(OUT)-BCLKIN-I/O	57	J11
48	D6-I/O	58	H10
—	I/O	59	H11
49	D5-I/O	60	F10
50	CS0-I/O	61	G10
51	D4-I/O	62	G11
—	I/O	63	G9
52	VCC	64	F9
53	D3-I/O	65	F11
54	CS1-I/O	66	E11
55	D2-I/O	67	E10
—	I/O	68	E9
—	I/O*	69	D11
56	D1-I/O	70	D10
57	RDY/BUSY-CLK-I/O	71	C11
58	D0-DIN-I/O	72	B11
59	DOUT-I/O	73	C10
60	CLK	74	A11
61	A0-WS-I/O	75	B10
62	A1-CS2-I/O	76	B9
63	A2-I/O	77	A10
64	A3-I/O	78	A9
—	I/O*	79	B8
—	I/O*	80	A8
65	A15-I/O	81	B6
66	A4-I/O	82	B7
67	A14-I/O	83	A7
68	A5-I/O	84	C7
1	GND	1	C6
2	A13-I/O	2	A6
3	A6-I/O	3	A5
4	A12-I/O	4	B5
5	A7-I/O	5	C5
—	I/O*	6	A4
—	I/O*	7	B4
6	A11-I/O	8	A3
7	A8-I/O	9	A2
8	A10-I/O	10	B3
9	A9-I/O	11	A1

Unprogrammed IOBs have a default pull-up. This prevents an undefined pad level for unbonded or unused IOBs. Programmed outputs are default slew-rate limited.

This table describes the pinouts of three different chips in three different packages. The second column lists 84 of the 118 pads on the XC3042 (and 84 of the 98 pads on the XC3030) that are connected to the 84 package pins. Ten pads, indicated by an asterisk, do not exist on the XC3020, which has 74 pads; therefore the corresponding pins on the 84-pin packages have no connections to an XC3020. Six pads on the XC3020 and 16 pads on the XC3030, indicated by a dash (—) in the 68 PLCC column, have no connection to the 68 PLCC, but are connected to the 84-pin packages.



XC3000A Logic Cell Array Family

Preliminary Product Specifications

Features

- Enhanced, high performance FPGA family with five device types
 - Improved redesign of the basic XC3000 LCA Family
 - Logic densities from 1,300 to 7,500 gates
 - Up to 144 user-definable I/Os
- Superset of the industry-leading XC3000 family
 - Identical to the basic XC3000 in structure, pin out, design methodology, and software tools
 - 100% compatible with all XC3000, XC3000L, and XC3100 bitstreams
 - Improved routing and additional features
- Additional programmable interconnection points (PIPs)
 - Improved access to longlines and CLB clock enable inputs
 - Most efficient XC3000-class solution to bus-oriented designs
- Advanced 0.8 μ CMOS static memory technology
 - Low quiescent and active power consumption
- Performance specified by logic delays, faster than corresponding XC3000 versions
- XC3000A-specific features
 - 4 mA output sink and source current
 - Error checking of the configuration bitstream
 - Soft startup starts all outputs in slew-limited mode upon power-up
 - Easy migration to the XC3400 series of HardWire mask programmed devices for high-volume production.

Description

The XC3000A family offers the following enhancements over the popular XC3000 family:

The XC3000A family has additional interconnect resources to drive the I-inputs of TBUFs driving horizontal Longlines. The CLB Clock Enable input can be driven from a second vertical Longline. These two additions result in more efficient and faster designs when horizontal Longlines are used for data bussing.

During configuration, the XC3000A devices check the bitstream format for stop bits in the appropriate positions. Any error terminates the configuration and pulls INIT Low.

When the configuration process is finished and the device starts up in user mode, the first activation of the outputs is automatically slew-rate limited. This feature, called Soft Startup, avoids the potential ground bounce when all outputs are turned on simultaneously. After start-up, the slew rate of the individual outputs is, as in the XC3000 family, determined by the individual configuration option.

The XC3000A family is a superset of the XC3000 family. Any bitstream used to configure an XC3000 or XC3100 device configures an XC3000A device exactly the same way.

Device	CLBs	Array	User I/Os Max	Flip-Flops	Horizontal Longlines	Configurable Data Bits
XC3020A	64	8 x 8	64	256	16	14,779
XC3030A	100	10 x 10	80	360	20	22,176
XC3042A	144	12 x 12	96	480	24	30,784
XC3064A	224	16 x 14	120	688	32	46,064
XC3090A	320	16 x 20	144	928	40	64,160

Xilinx maintains test specifications for each product as controlled documents. To insure the use of the most recently released device performance parameters, please request a copy of the current test-specification revision.

Absolute Maximum Ratings

Symbol	Description		Units
V _{CC}	Supply voltage relative to GND	−0.5 to +7.0	V
V _{IN}	Input voltage with respect to GND	−0.5 to V _{CC} +0.5	V
V _{TS}	Voltage applied to 3-state output	−0.5 to V _{CC} +0.5	V
T _{STG}	Storage temperature (ambient)	−65 to +150	°C
T _{SOL}	Maximum soldering temperature (10 s @ 1/16 in.)	+260	°C
T _J	Junction temperature plastic	+125	°C
	Junction temperature ceramic	+150	°C

Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

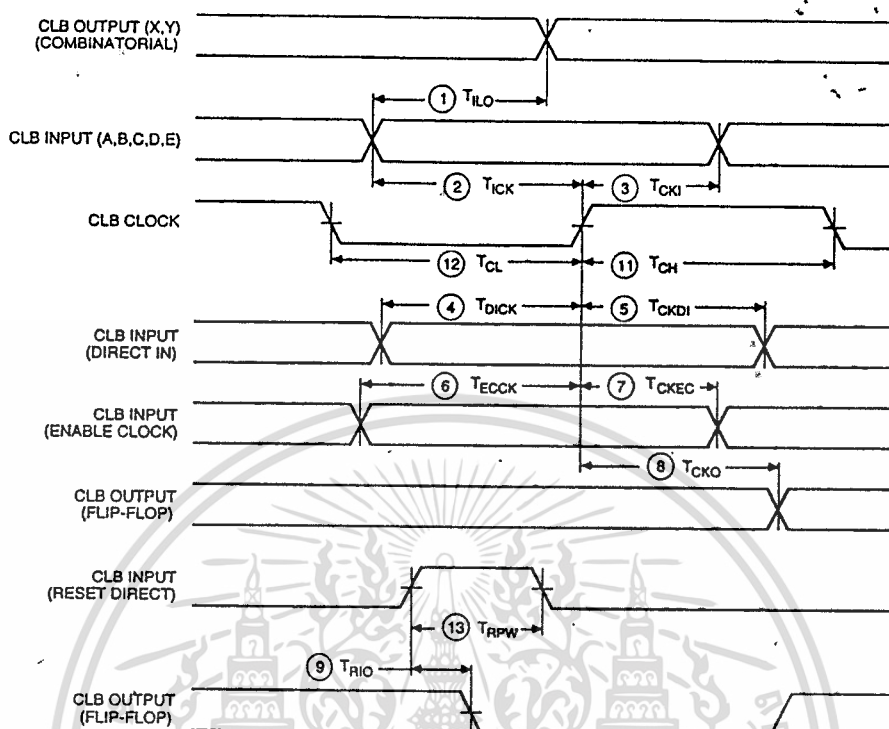
Symbol	Description	Min	Max	Units
V _{CC}	Supply voltage relative to GND Commercial 0°C to +70°C	4.75	5.25	V
	Supply voltage relative to GND Industrial −40°C to +85°C	4.5	5.5	V
V _{IHT}	High-level input voltage — TTL configuration	2.0	V _{CC}	V
V _{ILT}	Low-level input voltage — TTL configuration	0	0.8	V
V _{IHC}	High-level input voltage — CMOS configuration	70%	100%	V _{CC}
V _{ILC}	Low-level input voltage — CMOS configuration	0	20%	V _{CC}
T _{IN}	Input signal transition time		250	ns

DC Characteristics Over Operating Conditions

Symbol	Description		Min	Max	Units
V _{OH}	High-level output voltage (@ I _{OH} = -4.0 mA, V _{CC} min)	Commercial	3.86		V
V _{OL}	Low-level output voltage (@ I _{OL} = 4.0 mA, V _{CC} max)			0.40	V
V _{OH}	High-level output voltage (@ I _{OH} = -4.0 mA, V _{CC} min)	Industrial	3.76		V
V _{OL}	Low-level output voltage (@ I _{OL} = 4.0 mA, V _{CC} max)			0.40	V
V _{CCPD}	Power-down supply voltage (PWRDWN must be Low)		2.30		V
I _{CCPD}	Power-down supply current (V _{CC(MAX)} @ T _{MAX})	XC3020A		50	μA
		XC3030A		80	μA
		XC3042A		120	μA
		XC3064A		170	μA
		XC3090A		250	μA
I _{CCO}	Quiescent LCA supply current in addition to I _{CCPD} Chip thresholds programmed as CMOS levels			500	μA
	Chip thresholds programmed as TTL levels			10	μA
I _{IL}	Input Leakage Current		-10	+10	μA
C _{IN}	Input capacitance, all packages except PGA175 (sample tested) All Pins except XTL1 and XTL2 XTL1 and XTL2			10 15	pF pF
	Input capacitance, PGA 175 (sample tested) All Pins except XTL1 and XTL2 XTL1 and XTL2			15 20	pF pF
I _{RIN}	Pad pull-up (when selected) @ V _{IN} = 0 V (sample tested)		0.02	0.17	mA
I _{RL}	Horizontal Longline pull-up (when selected) @ logic Low			3.4	mA

* With no output current loads, no active input or Longline pull-up resistors, all package pins at Vcc or GND, and the LCA device configured with a MakeBits tie option.

CLB Switching Characteristic Guidelines



Buffer (Internal) Switching Characteristic Guidelines

1105 28

Speed Grade					
Description	Symbol				Units
Global and Alternate Clock Distribution* Either: Normal IOB input pad through clock buffer to any CLB or IOB clock input Or: Fast (CMOS only) input pad through clock buffer to any CLB or IOB clock input	T_{PID}				ns
	T_{PIDC}				ns
TBUF driving a Horizontal Longline (L.L.)* I to L.L. while T is Low (buffer active) T↓ to L.L. active and valid with single pull-up resistor T↓ to L.L. active and valid with pair of pull-up resistors T↑ to L.L. High with single pull-up resistor T↑ to L.L. High with pair of pull-up resistors	T_{IO}				ns
	T_{ON}				ns
	T_{ON}				ns
	T_{PUS}				ns
	T_{PUF}				ns
BIDI Bidirectional buffer delay	T_{BIDI}				ns

* Timing is based on the XC3042A, for other devices see XACT timing calculator.

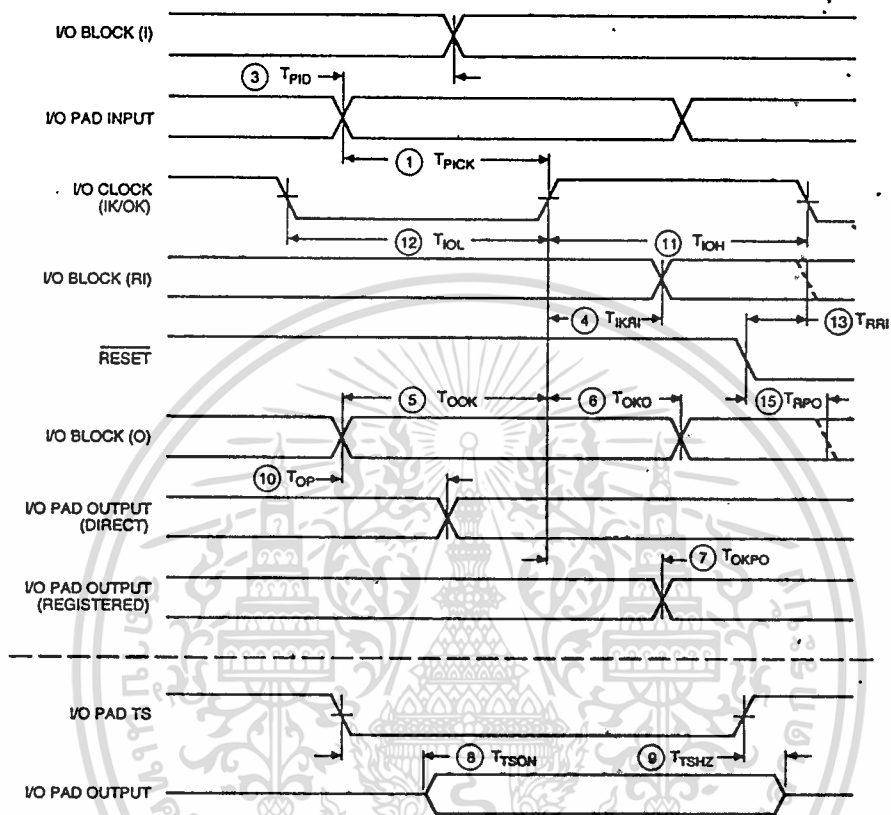
Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description		Symbol	Speed Grade							Units
Combinatorial Delay Logic Variables A, B, C, D, E, to outputs X or Y		1	T _{IL0}							ns
Sequential delay Clock k to outputs X or Y		8	T _{CK0}							ns
Clock k to outputs X or Y when Q is returned through function generators F or G to drive X or Y			T _{OL0}							ns
Set-up time before clock K Logic Variables A, B, C, D, E		2	T _{ICK}							ns
Data In DI		4	T _{DICK}							ns
Enable Clock EC		6	T _{ECCK}							ns
Reset Direct inactive RD										ns
Hold Time after clock K Logic Variables A, B, C, D, E		3	T _{CKI}							ns
Data In DI		5	T _{CKDI}							ns
Enable Clock EC		7	T _{CKEC}							ns
Clock Clock High time		11	T _{CH}							ns
Clock Low time		12	T _{CL}							ns
Max. flip-flop toggle rate			F _{CLK}							MHz
Reset Direct (RD) RD width		13	T _{RPW}							ns
delay from RD to outputs X or Y		9	T _{RIO}							ns
Global Reset (RESET Pad)* RESET width (Low)			T _{MW}							ns
delay from RESET pad to outputs X or Y			T _{MRO}							ns

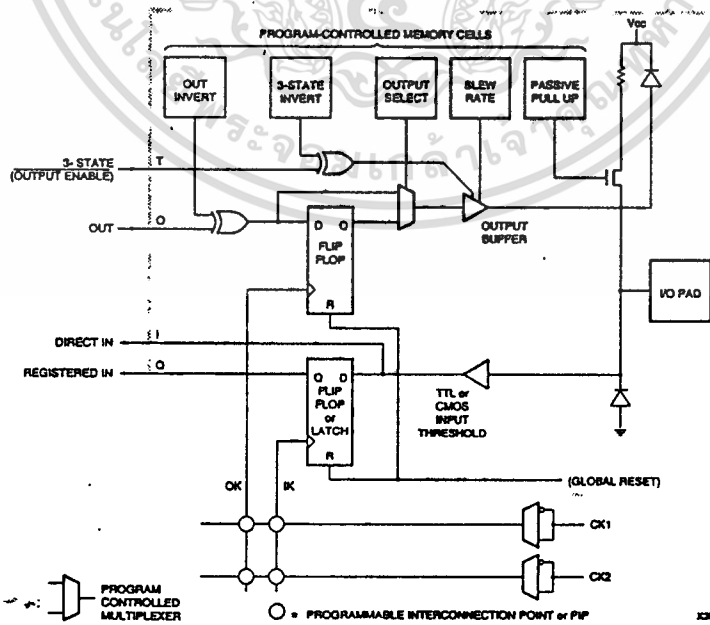
*Timing is based on the XC3042A, for other devices see XACT timing calculator.

Notes: The CLB K to Q output delay (T_{CKO} , #8) of any CLB, plus the shortest possible interconnect delay, is always longer than the Data In hold time requirement (T_{CKD} , #5) of any CLB on the same die. T_{LO} , T_{OLO} and T_{ICK} are specified for 4-input functions. For 5-input functions or base FGM functions, each specification increases by 0.8 ns (-5), 0.6 ns (-4) and 0.5 ns (-3).

IOB Switching Characteristic Guidelines



1105 57C



XC3000

- Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

XC3000A Logic Cell Array Family

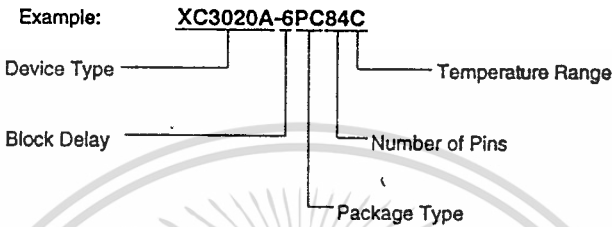
For a detailed description of the device architecture, see pages 2-100 through 2-117.

For a detailed description of the configuration modes and their timing, see pages 2-118 through 2-126.

For detailed lists of package pin-outs, see pages 2-132 through 2-142.

For package physical dimensions, see Section 4.

Ordering Information



Component Availability

PINS	44	64	68	84	100					132		144	160	164	175		176	208	223
TYPE	PLAST. PLCC	PLAST. VOFP	PLAST. PLCC	PLAST. PLCC	CERAM. PGA	PLAST. POFP	PLAST. TOFP	PLAST. VOFP	TOP. BRAZED COFP	PLAST. PGA	CERAM. PGA	PLAST. TOFP	PLAST. POFP	TOP. BRAZED COFP	PLAST. PGA	CERAM. PGA	PLAST. TOFP	PLAST. POFP	CERAM. PGA
CODE	PC44	VO64	PC68	PC84	PG84	PO100	TQ100	VQ100	CB100	PP132	PG132	TQ144	PQ160	CB164	PP175	PG175	TQ176	PQ208	PG223
XC3020A	-7		C1	C1	C1	C1													
	-6		C	C	C	C													
XC3030A	-7	C1	C1	C1	C1	C1		C1											
	-6	C	C	C	C	C		C											
XC3042A	-7			C1	C1	C1		C1		C1	C1	C1							
	-6			C	C	C		C		C	C	C							
XC3064A	-7			C1						C1	C1	C1	C1						
	-6			C						C	C	C	C						
XC3080A	-7			C1										C1		C1	C1	C1	
	-6			C										C		C	C	C	

C = Commercial = 0° to +70° C I = Industrial = -40° to +85° C M = Mil Temp = -55° to +125° C B = MIL-STD-883C Class B

Parentheses indicate future product plans

ประวัติผู้เขียน

ชื่อผู้เขียน	นายธีรศักดิ์ชัย อังสกุล
วันเดือนปีเกิด	วันที่ 13 เมษายน พ.ศ. 2511
สถานที่เกิด	นราธิวาส
วุฒิการศึกษาระดับปริญญาตรี	มหาวิทยาลัยศรีนครินทรวิโรฒ วิทยาเขตพิษณุโลก
ปีที่สำเร็จการศึกษา	2532
ผลงานทางวิชาการที่ได้รับการตีพิมพ์	การออกแบบวงจรรวมเฉพาะกิจเพื่อช่วยป้องกันการเข้าถึงข้อมูลในหน่วยความจำชนิด EEPROM
รางวัลหรือทุนที่เคยได้รับ	มูลนิธิคอมพิวเตอร์และการสื่อสาร



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้