สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส

CRYPTOCHIP APPLICATION PROTOTYPE

นาย นพดล ศิริขำ

นางสาว ปิยวรรณ นวมรัตน์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เลขหมู่.....

เลขทะเบียน 49997

วัน,เดือน,ปี 16 เม.ย. 2547

.b.....

.i.....

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส

CRYPTOCHIP APPLICATION PROTOTYPE

โดย

นาย นพดล ศิริขำ

นางสาว ปิยวรรณ นวมรัตน์

อาจารย์ที่ปรึกษา

อาจารย์ อัครเดช วัชระภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส

CryptoChip Application Prototype

ผู้จัดทำ

1. นาย นพดล ศิริขำ รหัสประจำตัว 42010159
2. นางสาว ปิยวรรณ นวมรัตน์ รหัสประจำตัว 42010203

_____ อาจารย์ที่ปรึกษา

(อาจารย์ อัครเดช วัชระภูพงษ์)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส

นาย นพดล ศิริขำ 42010159
นางสาว ปิยวรรณ นวมรัตน์ 42010203

อาจารย์ อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2545

บทคัดย่อ

การสื่อสารข้อมูลในปัจจุบันจำต้องพึ่งพากรรมวิธีการเข้าและถอดรหัสอันสลับซับซ้อนยิ่งขึ้นเรื่อยๆ ฉะนั้น การเข้าและถอดรหัสด้วยฮาร์ดแวร์จึงเป็นสิ่งที่เลี่ยงไม่ได้ แม้ว่ากรรมวิธีการเข้าและถอดรหัสจะเป็นที่ทราบกันดีและมีซอร์สโค้ดทั่วไปก็ตาม หากแต่เมื่อมีการเปลี่ยนแปลงมาใช้งานกับฮาร์ดแวร์ ต้องพิจารณาถึงข้อเด่นและข้อด้อยของแพลตฟอร์มฮาร์ดแวร์นั้นด้วย เพื่อให้เกิดประสิทธิภาพการทำงานสูงสุด

โครงงานนี้ มุ่งหวังที่ให้เป็นจุดเริ่มต้นให้เกิดต้นแบบการใช้งานชิปเข้าและถอดรหัส โดยใช้วิธีการของภาษาเอชดีและอัลกอริทึม XOR ซึ่งเข้าและถอดรหัสพื้นฐานบนฮาร์ดแวร์ที่กำหนด พร้อมพัฒนาซอฟต์แวร์ไดรเวอร์หรือไลบรารีเท่าที่จำเป็น จากนั้น จึงพัฒนางานตัวอย่างเพื่อพิสูจน์ความสามารถในการใช้งานได้จริง ซึ่งใช้งานวีพีเอ็นเป็นงานตัวอย่าง โดยทำเป็นการส่งข้อมูลระหว่าง 2 เน็ตเวิร์ก แล้วดักจับข้อมูลเพื่อตรวจสอบความสามารถของตัวต้นแบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CryptoChip Application Prototype

Noppadol Sirikhum

Piyawan Nawamarat

Akkradach Watcharapupong Advisor

ABSTRACT

Nowadays, the security of data communication needs to use more complex cryptographic algorithms for preventing the attacker to get data. Sometime cryptographic software cannot immediately support these complex cryptographic algorithms. This is the reason why we should use cryptographic hardware. Although cryptographic algorithms are well known and those source codes are opened but when we use cryptographic hardware, we have to consider in advantage and disadvantage of these hardware platforms for getting the most effective result.

We hope this project will be the beginning of CryptoChip Application Prototype in which using HD language and XOR algorithm to encrypt and decrypt information. Then we developed software drivers or necessary libraries.

Finally, we created test case with VPN that transfers data between two networks and then capture data to prove ability of prototype.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้ เสร็จสมบูรณ์ได้ด้วยดี จากความช่วยเหลือทั้งแรงกาย แรงวาจา และแรงใจของบุคคลหลายฝ่าย ได้แก่

อาจารย์อัครเดช หรือ พี่เด็ก (Adek) ของน้อง ๆ ซึ่งเป็นอาจารย์ที่ปรึกษา ที่คอยให้คำแนะนำ คำปลอบใจ และความสนุกสนานเฮฮาทั้งหลายที่กระหน่ำปล่อยออกมา เพื่อสร้างบรรยากาศของห้องให้น่าอยู่มากขึ้น

อาจารย์เจริญ หรือ พี่ต๋อง ผู้เป็นขุมความรู้ให้แก่ปริญญานิพนธ์ฉบับนี้ ที่คอยให้คำชี้แนะ คำปรึกษา และความห่วงใยตลอดมา

ห้องวิจัยอีเอสแอล (ESL) ที่ช่วยอุปถัมภ์ค้ำจุนอุปกรณ์เครื่องมือเครื่องใช้ต่าง ๆ ที่ทำให้การทำงานผ่านพ้นไปได้อย่างลุล่วงด้วยดี

เพื่อน ๆ จำนวนมากทั้งหลาย ที่คอยเป็นกำลังใจและ/หรือเป็นนักแสดงตลกคาเฟ่มืออาชีพที่ช่วยให้มีสีสันในการทำงาน และทำงานได้อย่างสนุกสนาน

รุ่นพี่และรุ่นน้องทุกท่านทุกคน ที่คอยกระตุ้นให้ตื่นตัวอยู่เสมอ ถึงวันและเวลาที่ใกล้จะส่งปริญญานิพนธ์ พร้อมทั้งช่วยนับถอยหลังให้อย่างเต็มใจ

บุคคลทางบ้าน ที่ช่วยเสริมสร้างพลังกาย พลังใจ และพลังชีวิตให้แก่การทำงาน

และที่สำคัญที่สุด คือ สมาชิกในกลุ่มผู้จัดทำ ที่สามารถทำงานที่ใครต่อใครคาดว่าจะไม่เสร็จ ให้สามารถสำเร็จลุล่วงไปได้ด้วยดี

ณ ที่นี้ จึงขอขอบคุณและขอบพระคุณทุก ๆ ท่าน ในทุก ๆ ความปรารถนาดีที่มีให้มา

นพดล ศิริขำ

ปิยวรรณ นวมรัตน์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

	หน้าที่
บทคัดย่อ	I
ABSTRACT	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VIII
สารบัญรูปภาพ	IX
บทที่ 1 บทนำ	1
1.1 ที่มาของโครงงาน	1
1.2 วัตถุประสงค์ของโครงงาน	1
1.3 ขอบเขตของโครงงาน	1
1.4 ประโยชน์ที่คาดว่าจะได้รับ	1
บทที่ 2 การเข้ารหัสและการคำนวณ	2
2.1 ระบบของการเข้ารหัสข้อมูล (Cryptography System)	2
2.1.1 ระบบการเข้ารหัสโดยใช้กุญแจเดียว	2
2.1.2 ระบบการเข้ารหัสโดยใช้กุญแจสาธารณะ	3
2.2 การเข้ารหัสแบบ XOR	3
2.3 การเข้ารหัสแบบ DES (Data Encryption Standard)	4
2.4 การเข้ารหัสแบบ 3DES (Triple Data Encryption Standard)	5
2.5 การเข้ารหัสแบบ RSA	5
2.6 การแปลงข้อมูลโดยใช้การคำนวณแบบ MD5 (Message Digest Standard)	7
2.7 การเข้ารหัสข้อมูลสำหรับการสื่อสารในโพรโตคอลทีซีพี/ไอพี (TCP/IP)	9
2.7.1 การเข้ารหัสระดับการเชื่อมโยงข้อมูล	9
2.7.2 การเข้ารหัสระดับเครือข่าย	9
2.7.3 การเข้ารหัสระดับทรานสปอร์ต	10
2.7.4 การเข้ารหัสระดับโปรแกรมประยุกต์	10
บทที่ 3 เอฟพีจีเอ	11
3.1 Field Programmable	11
3.1.1 พีแอลดี (PLD: Programmable Logic Device	12

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้าที่

3.1.2 แอลซีเอ (LCA: Logic Cell Array 12

3.1.3 เอฟพีจีเอ (FPGA: Field Programmable Gate Array) 12

3.2 Mark Programmable 13

3.3 เทคโนโลยีของเอฟพีจีเอ 13

3.4 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000 14

3.5 คุณสมบัติของเอฟพีจีเอตระกูล XC4000 17

3.6 การโปรแกรมเอฟพีจีเอตระกูล XC4000 18

3.7 ความสามารถของแรมในเอฟพีจีเอตระกูล XC4000 21

บทที่ 4 การ์ดไอเอสเอและสตับ 22

4.1 การ์ดไอเอสเอ (ISA Card) 22

4.1.1 ไอเอสเอบัส 22

4.1.2 ขาสัญญาณต่าง ๆ ของการ์ดไอเอสเอ 23

4.1.3 ดีเอ็มเอ (DMA: Direct Memory Access) 27

4.2 สตับ (Stub) 28

บทที่ 5 การเขียนดีไวซ์ไดรเวอร์ 30

5.1 ความหมายของดีไวซ์ไดรเวอร์ 30

5.2 การสร้างไฟล์อุปกรณ์ 30

5.3 ชนิดของอุปกรณ์ 31

5.3.1 คาแร็กเตอร์ดีไวซ์ (Character Device) 31

5.3.2 บล็อกดีไวซ์ (Block Device) 32

5.3.3 เน็ตเวิร์กดีไวซ์ (Network Device) 32

5.4 การเขียนโปรแกรมดีไวซ์ไดรเวอร์บนระบบปฏิบัติการลีนุกซ์ 32

5.4.1 ขั้นตอนการเขียนโปรแกรมดีไวซ์ไดรเวอร์ 33

5.4.2 การเขียนโปรแกรมโมดูล 33

5.4.3 การคอมไพล์ (Compliment) 34

5.5 คาแร็กเตอร์ดีไวซ์ไดรเวอร์ 34

5.5.1 การเพิ่มและลบโมดูลไดรเวอร์ออกจากระบบ 34

5.5.2 ฟังก์ชันที่ใช้ทำงานกับอุปกรณ์ (ไฟล์) 35

5.5.3 การเปิดและปิดไฟล์ดีไวซ์ไดรเวอร์ 36

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้าที่

5.5.3.1 ขั้นตอนการเปิดไฟล์ดีไวซ์ไดรเวอร์ 36
5.5.3.2 ขั้นตอนการปิดไฟล์ดีไวซ์ไดรเวอร์ 38
5.5.4 การอ่านและเขียนข้อมูลบนอุปกรณ์ 38
5.6 บล็อกดีไวซ์ไดรเวอร์ 41
5.6.1 การเพิ่มหรือลบไดรเวอร์ออกไปจากเคอร์เนล 41
5.6.2 ฟังก์ชันการอ่านและเขียนดีไวซ์ไดรเวอร์ 41
5.6.3 การยกเลิกการโหลดข้อมูลของไดรเวอร์ 42
5.7 เน็ตเวิร์กดีไวซ์ไดรเวอร์ 43
5.7.1 การส่งแพ็กเกตข้อมูล 44
5.7.2 การรับแพ็กเกตข้อมูล 45

บทที่ 6 วีพีเอ็น 47
6.1 ความหมายของวีพีเอ็น (VPN: Virtual Private Network) 48
6.2 คุณสมบัติของวีพีเอ็น 48
6.3 การทำงานของวีพีเอ็น 48
6.4 การสร้างทันเนลในวีพีเอ็น 49
6.5 รูปแบบบริการของวีพีเอ็น 50
6.6 อุปกรณ์ทียูเอ็นและทีเอพี (TUN/TAP) 51
6.6.1 ความหมายของทียูเอ็น 51
6.6.2 ความหมายของทีเอพี 51
6.6.3 การทำงานของอุปกรณ์เน็ตเวิร์กแบบเสมือน (Virtual Network Device) 52
6.6.4 ไดรเวอร์ของทียูเอ็นและทีเอพี 52
6.6.5 การตั้งค่าคอนฟิกกูเรชัน 52
6.6.6 การเขียนโปรแกรมส่วนติดต่อ 52
6.7 การนำวีพีเอ็นมาใช้ในชีวิตจริง 53
6.8 ประโยชน์ที่ได้รับจากวีพีเอ็น 56

บทที่ 7 การออกแบบ 57
7.1 แนวคิดในการออกแบบ 57
7.2 ภาพรวมในการออกแบบ 57
7.3 การทำงานร่วมกันในแต่ละส่วน 58

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ (ต่อ)

หน้าที่

7.4 การทำงานในแต่ละส่วนย่อย 58

7.4.1 การออกแบบส่วนของอุปกรณ์ฮาร์ดแวร์ 58

7.4.1.1 ส่วนของการ์ดไอเอสเอ 59

7.4.1.2 ส่วนของบอร์ดเอฟพีจีเอ 59

7.4.1.3ส่วนของโปรแกรมที่โหลดลงบอร์ดเอฟพีจีเอ 59

7.4.2 ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์ 61

7.4.3 ส่วนของโปรแกรมวีพีเอ็น 63

7.5 แนวคิดในการออกแบบชุดทดสอบ 64

7.6 ชุดทดสอบส่วนของอุปกรณ์ฮาร์ดแวร์ 64

7.7 ชุดทดสอบส่วนของระบบดีไวซ์ไดรเวอร์ 65

7.8 ชุดทดสอบโปรแกรมวีพีเอ็น 65

บทที่ 8 การทดลองและผลการทดลอง 67

8.1 การตรวจสอบความถูกต้องของโปรแกรม 67

8.2 การโหลดโปรแกรมลงบอร์ด 69

8.3 การตรวจสอบการทำงานของโปรแกรม บอร์ดเอฟพีจีเอ และการ์ดไอเอสเอ 72

8.4 การทดสอบการทำงานของตัวต้นแบบ 73

บทที่ 9 บทวิจารณ์และสรุป 75

9.1 การวิจารณ์โครงงาน 75

9.1.1 ส่วนที่เกี่ยวข้องกับฮาร์ดแวร์ 75

9.1.2 ส่วนที่เกี่ยวข้องกับซอฟต์แวร์ 75

9.2 สรุปผลของโครงงาน 76

9.3 แนวทางการพัฒนาต่อไปในอนาคต 76

บรรณานุกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญตาราง

หน้าที่

ตารางที่ 2-1: ตารางแสดงการทำงานของ XOR 3
ตารางที่ 3-1: รายละเอียดของอุปกรณ์ภายในของเอฟพีจีเอตระกูล XC4000 18
ตารางที่ 3-2: โหมดการโปรแกรม 6 โหมดของเอฟพีจีเอตระกูล XC4000 19
ตารางที่ 3-3: จำนวนของแรมภายในเอฟพีจีเอตระกูล XC4000 21
ตารางที่ 4-1: ชนิดของการใช้ขาสัญญาณโดยระบบบอร์ดไอเอสเอ (ISA System Board) 27
ตารางที่ 8-1: ตารางการเชื่อมต่อระหว่างบอร์ดเอฟพีจีเอกับการ์ดไอเอสเอ 83

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูปภาพ

หน้าที่

รูปที่ 2-1: การเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว 3
รูปที่ 2-2: การเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ 3
รูปที่ 2-3: กระบวนการเข้ารหัสด้วย DES 4
รูปที่ 2-4: กระบวนการเข้าและถอดรหัสด้วย 3DES 5
รูปที่ 2-5: ขั้นตอนการเข้ารหัสแบบ RSA 6
รูปที่ 2-6: การทำงานของ MD5 8
รูปที่ 2-7: การเข้ารหัสในระดับการเชื่อมโยงข้อมูล 9
รูปที่ 2-8: การเข้ารหัสระดับเครือข่าย 9
รูปที่ 2-9: การเข้ารหัสระดับทรานสปอร์ต 10
รูปที่ 2-10: การเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดล (OSI Model) ของทีซีพี/ไอพี 10
รูปที่ 3-1: ผังการแบ่งกลุ่มของวงจรรวม ASIC 11
รูปที่ 3-2: วงจรพื้นฐานของอุปกรณ์ที่แอลดีซึ่งอยู่ในรูปผลคูณร่วมบวก 12
รูปที่ 3-3: วงจรพื้นฐานภายในเอฟพีจีเอ 14
รูปที่ 3-4: ผังวงจรภายในของซีแอลบีของเอฟพีจีเอตระกูล XC4000 15
รูปที่ 3-5: ผังวงจรภายในของไอโอบีของเอฟพีจีเอตระกูล XC4000 16
รูปที่ 3-6: ผังการเชื่อมต่อภายในของเอฟพีจีเอตระกูล XC4000 16
รูปที่ 3-7: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดหลักแบบขนาน (Master Parallel Mode) 19
รูปที่ 3-8: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดเพอริเฟอรัลแบบอะซิงโครนัส (Peripheral Asynchronous Mode) 20
รูปที่ 3-9: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดรองแบบอนุกรม (Slave Serial mode) 20
รูปที่ 4-1: ลักษณะของการ์ดและสล็อตของการ์ด 8 บิตและ 16 บิต 22
รูปที่ 4-2: ขาสัญญาณต่าง ๆ ของการ์ดไอเอสเอ 23
รูปที่ 4-3: ไดอะแกรมช่วงเวลาของสัญญาณที่สำคัญของสตับ 29
รูปที่ 5-1: โครงสร้างข้อมูลของ Character Device 31
รูปที่ 5-2: โครงสร้างข้อมูลของ Block Device 32
รูปที่ 5-3: โครงสร้างในการรับและส่งแพ็กเกตข้อมูลของลีนุกซ์ 46
รูปที่ 6-1: การนำเครือข่ายวีพีเอ็นมาใช้ในระบบ 47
รูปที่ 6-2: การทำงานของวีพีเอ็น 49
รูปที่ 6-3: การนำวีพีเอ็นมาใช้ในรูปแบบ Site-to-Site 54
รูปที่ 6-4: การนำวีพีเอ็นมาใช้ในรูปแบบ Client-initiated 55

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูปภาพ (ต่อ)

หน้าที่

รูปที่ 6-5: การนำวีพีเอ็นมาใช้ในรูปแบบ Network Access Network 55
รูปที่ 7-1: การทำงานร่วมกันระหว่างแต่ละส่วนของระบบ 58
รูปที่ 7-2: ส่วนเชื่อมต่อและโครงสร้างของสตับ 60
รูปที่ 7-3: รูปแบบการเชื่อมต่อเพื่อทดสอบโปรแกรมวีพีเอ็น 65
รูปที่ 8-1: การเลือกตัวโปรแกรมมาตรวจสอบความถูกต้อง 67
รูปที่ 8-2: ผลที่ได้จากการตรวจสอบโปรแกรม 68
รูปที่ 8-3: การสังเคราะห์โปรแกรมภาษาวีเอชดีแอลและผลของการสังเคราะห์โปรแกรม 68
รูปที่ 8-4: การแปลงเป็นไฟล์ที่โปรแกรมลงอุปกรณ์ได้และผลของการแปลง 69
รูปที่ 8-5: โปรแกรมทดสอบบอร์ด 70
รูปที่ 8-6: โปรแกรมตั้งค่าให้คล็อก 70
รูปที่ 8-7: การแจ้งเตือนของการตั้งค่าจัมเปอร์ก่อนและหลังจากการเซตคล็อก 70
รูปที่ 8-8: โปรแกรมโหลดไฟล์ลงบอร์ด 71
รูปที่ 8-9: อุปกรณ์ฮาร์ดแวร์ที่ทำการเชื่อมต่อและการเชื่อมต่อกับบอร์ดเอฟพีจีเอ 71
รูปที่ 8-10: การตรวจสอบการทำงานของตัวต้นแบบ 73
รูปที่ 8-11: แสดงข้อมูลที่ทำการดักจับได้ แบบที่ไม่ได้ทำการเข้ารหัส 74
รูปที่ 8-12: แสดงข้อมูลที่ทำการดักจับได้ แบบที่ได้ทำการเข้ารหัส 74

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ที่มาของโครงงาน

ในโลกของความเป็นจริงที่ต้องการความปลอดภัยของข้อมูลและความเร็วในการรับส่งข้อมูลสูงมากขึ้น ระบบที่ต้องการความปลอดภัยต่างๆจึงจำเป็นที่จะต้องใช้การเข้ารหัสและถอดรหัสที่มีความถูกต้อง แม่นยำ และมีความรวดเร็วมากยิ่งขึ้น การเข้าและถอดรหัสด้วยซอฟต์แวร์มีความง่ายในการใช้งานและมีความยืดหยุ่นสูง แต่มีปัญหาในเรื่องของความเร็วโดยรวมที่มีขีดจำกัด ทำให้เราคิดที่จะลองทำชุดต้นแบบการเข้ารหัสด้วยฮาร์ดแวร์ขึ้นมา

1.2 วัตถุประสงค์ของโครงงาน

1. ศึกษาวิธีการเข้าและถอดรหัสอย่างง่าย
2. ศึกษาและพัฒนาต้นแบบชิปเข้าและถอดรหัสร่วมกับอุปกรณ์ FPGA พัฒนา
3. ศึกษาการเขียนโปรแกรมดีไวซ์ไดรเวอร์บนระบบปฏิบัติการลีนุกซ์
4. เพื่อศึกษาแนวทางการประยุกต์ใช้งานชิปเข้าและถอดรหัส
5. เพื่อสร้างต้นแบบระบบงานที่ใช้งานอุปกรณ์เข้าและถอดรหัส

1.3 ขอบเขตของโครงงาน

1. สามารถทำการเข้าและถอดรหัสข้อมูลผ่านฮาร์ดแวร์ได้อย่างถูกต้อง
2. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบคอมพิวเตอร์บุคคลได้
3. สามารถทำงานกับส่วนติดต่อมาตรฐานของระบบปฏิบัติการได้
4. สามารถใช้ทดแทนระบบการเข้ารหัสแบบเก่าได้บางส่วน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. เป็นต้นแบบในการใช้งานชิปเข้าและถอดรหัสผ่านฮาร์ดแวร์
2. สามารถทำการเข้าและถอดรหัสข้อมูลได้อย่างถูกต้อง
3. สามารถส่งข้อมูลถึงกันได้เสมือนเป็นการส่งผ่านเครือข่ายส่วนตัวเสมือน (Virtual Private Network)
4. รักษาข้อมูลให้เป็นความลับต่อผู้อื่นที่ไม่เกี่ยวข้องได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
การเข้ารหัสและการคำนวณ

ในการส่งข้อมูลที่เป็นความลับไปยังผู้รับ บางครั้งอาจจะมีการลักลอบอ่านและปลอมแปลงข้อมูลในระหว่างทางที่ส่งไป การเข้ารหัสเป็นวิธีที่จะช่วยไม่ให้มีบุคคลอื่นใดสามารถอ่านข้อมูลที่เราส่งไปได้ จึงทำให้ไม่สามารถที่จะปลอมแปลงข้อมูลของเราได้ การเข้ารหัสเป็นการรักษาความปลอดภัยของข้อมูลอย่างหนึ่ง โดยการเปลี่ยนแปลงข้อมูลเป็นที่รู้กันระหว่างผู้รับและผู้ส่ง

2.1 ระบบของการเข้ารหัสข้อมูล (Cryptography System)

การเข้ารหัสข้อมูล คือ การทำให้ข้อมูลเป็นความลับ ซึ่งเป็นส่วนสำคัญในระบบข้อมูลในปัจจุบันนี้ที่ต้องการความปลอดภัยของข้อมูล โดยอาศัยหลักการของการเข้ารหัสและการถอดรหัส

- การเข้ารหัส (Encryption) เป็นการเปลี่ยนรูปข้อมูล โดยผ่านกระบวนการต่างๆของการแปรรูปข้อมูล ซึ่งทำให้ข้อมูลไม่สามารถอ่านหรือตีความหมายได้ เพื่อทำให้ข้อมูลเป็นความลับ การเข้ารหัสมีพื้นฐานอยู่ 2 อย่าง คือ
  1. การแทนที่ (Substitution) เป็นการแทนที่บิตนั้นๆด้วยข้อมูลอื่น ทำให้ข้อมูลมีความสับสนยากต่อการถอดรหัส (โดยตำแหน่งบิตนั้นๆ ยังคงเรียงกันเหมือนเดิม)
  2. การสับเปลี่ยนตำแหน่ง (Permutation) เป็นการสับเปลี่ยนตำแหน่งใดๆของข้อมูล เมื่อมีการสับเปลี่ยนตำแหน่งมากๆ ทำให้ข้อมูลมีความซับซ้อนยากต่อการถอดรหัส
- การถอดรหัส (Decryption) เป็นการแปลงข้อมูลที่ได้ผ่านการเข้ารหัสให้กลับมาอยู่ในสภาพเดิมที่สามารถอ่านและตีความหมายได้

ซึ่งการเข้าและถอดรหัสนั้น ได้ถูกควบคุมโดยกุญแจ (Key) โดยได้แบ่งระบบของการเข้ารหัสตามลักษณะของกุญแจที่ใช้ เป็น 2 แบบ คือ

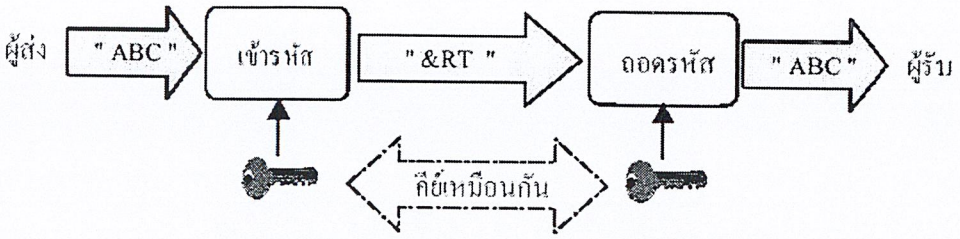
- ระบบการเข้ารหัสโดยใช้กุญแจเดียว (Single Key Encryption)
- ระบบการเข้ารหัสโดยใช้กุญแจสาธารณะ (Public Key Encryption)

2.1.1 ระบบการเข้ารหัสโดยใช้กุญแจเดียว

การเข้ารหัสข้อมูลโดยใช้กุญแจเดียว ผู้รับและผู้ส่งจะใช้กุญแจที่เป็นความลับ เนื่องจากการเข้าและถอดรหัสด้วยระบบนี้นั้น จะใช้กุญแจที่เหมือนกัน หากมีผู้อื่นทราบกุญแจตัวนี้แล้วจะทำให้ข้อมูลไม่เป็นความลับ และอาจทำให้การสื่อสารข้อมูลมีความผิดพลาดขึ้นได้ ถ้าผู้นั้นได้มีการเปลี่ยนแปลงข้อมูลภายใน

ตัวอย่างของการเข้ารหัสด้วยระบบนี้ ได้แก่ DES, 3DES, IDEA เป็นต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



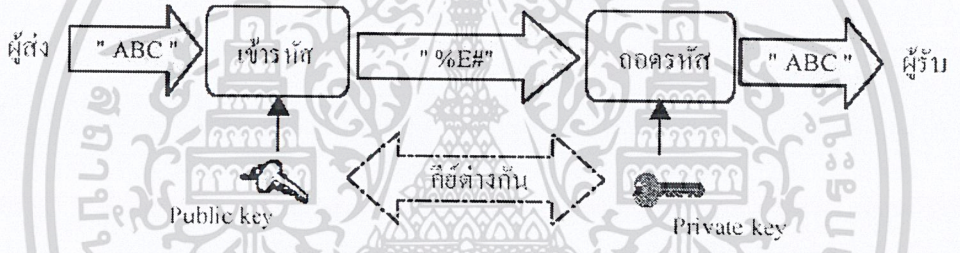
รูปที่ 2-1: การเข้ารหัสและถอดรหัสโดยใช้กุญแจเดียว

2.1.2 ระบบการเข้ารหัสโดยใช้กุญแจสาธารณะ

การเข้ารหัสโดยใช้กุญแจสาธารณะ ได้นำกุญแจมาประกอบการเข้ารหัส 2 อัน คือ

1. กุญแจส่วนตัว (Private Key) เป็น กุญแจที่จะต้องเก็บเป็นความลับ
2. กุญแจสาธารณะ (Public Key) เป็น กุญแจที่สามารถเปิดเผยให้ผู้อื่นทราบได้

ซึ่งกุญแจทั้งสองอันนี้ เป็นกุญแจที่ต่างกัน โดยมีวิธีการเข้าและถอดรหัส ดังรูปนี้



รูปที่ 2-2: การเข้ารหัสและถอดรหัสโดยใช้กุญแจสาธารณะ

2.2 การเข้ารหัสแบบ XOR

การเข้ารหัสแบบ XOR นี้ ถือว่าเป็นพื้นฐานของการเข้ารหัส โดยเมื่อต้องการจะเข้ารหัสข้อมูล จะนำเอาข้อมูลที่ต้องการเข้ารหัส (Plain Text) มา XOR ด้วยกุญแจ ซึ่งจะได้ข้อมูลที่ถูกเข้ารหัสออกมา (Cipher Text) และเมื่อต้องการจะถอดรหัสข้อมูล จะนำเอาข้อมูลที่ถูกเข้ารหัสแล้วมา XOR ด้วยกุญแจ ซึ่งผลที่ได้ออกมาคือ ข้อมูลที่นำมาเข้ารหัสในตอนต้น

Plain Text	XOR	Key	=>	Cipher Text
Cipher Text	XOR	Key	=>	Plain Text

XOR	0	1
0	0	1
1	1	0

ตารางที่ 2-1: ตารางแสดงการทำงานของ XOR

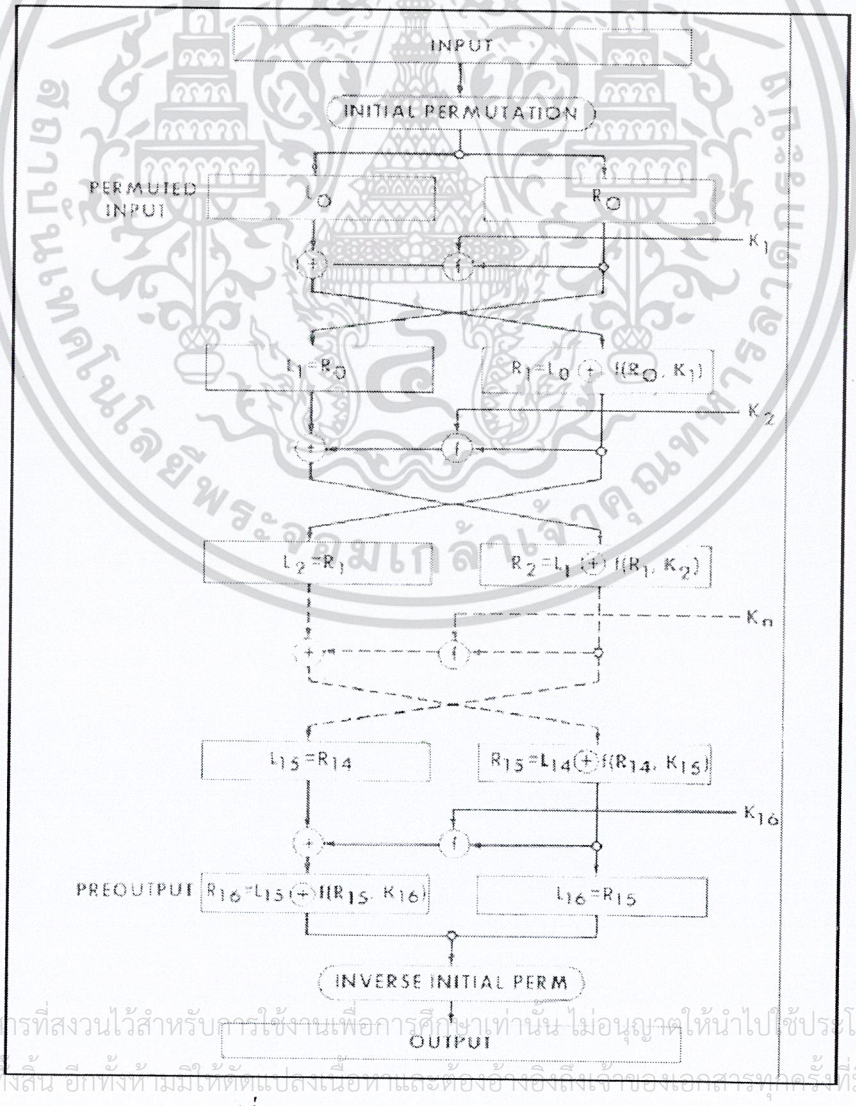
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.3 การเข้ารหัสแบบ DES (Data Encryption Standard)

DES คือ การเข้ารหัสข้อมูลโดยใช้อัลกอริทึมทางคณิตศาสตร์ โดยจะใช้ข้อมูลขนาด 64 บิต และกุญแจขนาด 64 บิต (โดย 8 บิตที่เพิ่มเข้ามา เพื่อนำมาใช้ตรวจสอบความผิดพลาด) ซึ่งในการเข้าและถอดรหัสนั้น ได้ใช้ระบบการเข้ารหัสแบบสมมาตร (Symmetric Cryptosystem) คือ ใช้กุญแจตัวเดียวกันในการเข้าและถอดรหัส ซึ่งในกระบวนการถอดรหัส จะทำกระบวนการย้อนกลับกับกระบวนการเข้ารหัส

การทำงานของอัลกอริทึม DES ได้แบ่งการทำงานออกเป็น 3 ส่วนที่สำคัญ คือ

1. **การเรียงบิตใหม่ (Initial Permutation)** เพื่อผลิต "permuted input" ซึ่งเป็นข้อมูลที่มีการสลับตำแหน่งจากเดิมไป
2. **การเข้าฟังก์ชันการคำนวณ** โดยมีกระบวนการทำงานแบบเดิมซ้ำกัน 16 ครั้ง ซึ่งผลลัพธ์ที่ได้จะได้เป็นข้อมูลขนาด 64 บิตอยู่ชุดหนึ่งที่เรียกว่า PREOUTPUT
3. **การเรียงบิตกลับ (Inverse Initial Permutation)** ซึ่งจะได้ผลลัพธ์ออกมาเป็นข้อมูลที่เข้ารหัสแล้ว



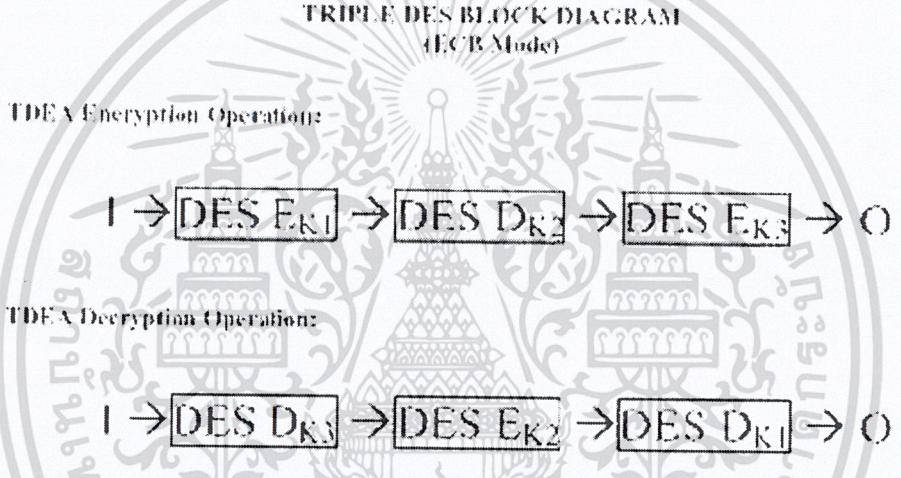
รูปที่ 2-3: กระบวนการเข้ารหัสด้วย DES

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำหรับกระบวนการถอดรหัสนั้น ทำเช่นเดียวกันกับกระบวนการเข้ารหัส คือ ใส่ข้อมูลที่ถูกเข้ารหัสข้อมูลมาแล้วที่มีขนาด 64 บิต พร้อมกับกุญแจตัวเดิมที่ใช้ในการเข้ารหัส แล้วทำตามขั้นตอนต่างๆ เหมือนกับการเข้ารหัสข้อมูล แต่ต้องทำการเปลี่ยนลำดับของกุญแจใหม่ โดยใช้กุญแจจากลำดับท้ายสุดเรียงมาจนถึงลำดับแรกสุด คือ K16, K15,..., K1

2.4 การเข้ารหัสแบบ 3DES (Triple Data Encryption Standard)

การเข้ารหัสแบบ 3DES นั้น เป็นการเข้ารหัสอีกแบบหนึ่ง โดยอิงมาตรฐานมาจากการเข้ารหัสแบบ DES ซึ่งเข้าและถอดรหัสแบบ DES เป็นจำนวน 3 ครั้ง และใช้กุญแจเป็นจำนวน 3 ตัวในการทำงาน



รูปที่ 2-4: กระบวนการเข้าและถอดรหัสด้วย 3DES

โดยมีขั้นตอนในการเข้ารหัสข้อมูล ดังนี้คือ

1. เข้ารหัสข้อมูลโดยใช้กุญแจตัวแรก
2. ถอดรหัสข้อมูลโดยใช้กุญแจตัวที่สอง และ
3. เข้ารหัสข้อมูลโดยใช้กุญแจตัวที่สาม

ในส่วนของขั้นตอนในการถอดรหัสข้อมูล มีดังนี้คือ

1. ถอดรหัสข้อมูลโดยใช้กุญแจตัวแรก
2. เข้ารหัสข้อมูลโดยใช้กุญแจตัวที่สอง และ
3. ถอดรหัสข้อมูลโดยใช้กุญแจตัวที่สาม

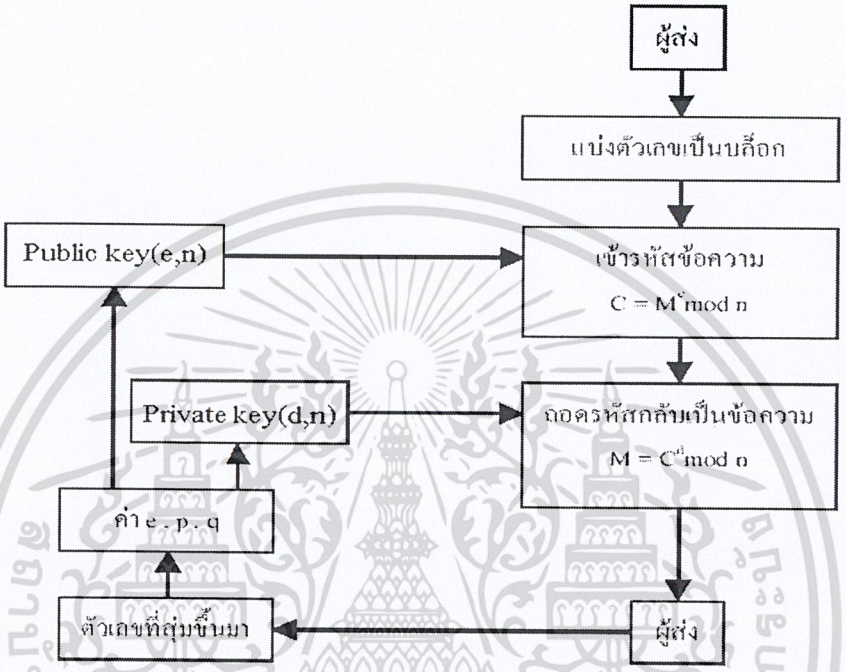
2.5 การเข้ารหัสแบบ RSA

เป็นการเข้ารหัสแบบกุญแจสาธารณะ (Public-Key Cryptosystem) เพื่อแก้ไขปัญหาการทำให้กุญแจเป็นความลับ (Secret-Key Cryptosystem) โดยแต่ละบุคคลมีกุญแจอยู่ 2 ชนิด คือ กุญแจส่วนตัว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(Private Key) และกุญแจสาธารณะ (Public Key) โดยกุญแจส่วนตัวจะเก็บเป็นความลับ ส่วนกุญแจสาธารณะนั้นได้เปิดเผยแก่บุคคลทั่วไป บุคคลใดก็ได้ที่ต้องการส่งเอกสารให้แก่ตน

หลักการทำงานของการเข้ารหัสแบบกุญแจสาธารณะ คือ ข้อมูลที่ถูกเข้ารหัสด้วยกุญแจสาธารณะของบุคคลใด ย่อมสามารถถอดรหัสได้ด้วยกุญแจสาธารณะของบุคคลนั้นเท่านั้น



รูปที่ 2-5: ขั้นตอนการเข้ารหัสแบบ RSA

ในการเข้ารหัสแบบ RSA มีวิธีการหากุญแจ ดังนี้

1. เลือกจำนวนเฉพาะ 2 จำนวน เช่น p = 7, q = 17
2. คำนวณ n = p * q = 119
3. คำนวณ (p-1)(q-1) = 96
4. เลือก e ซึ่งมีความสัมพันธ์กับค่า (p-1)(q-1) โดย e ที่มีค่าน้อยกว่า n และไม่สามารถหาร (p-1)(q-1) ได้ลงตัว ซึ่งได้ค่าของ e = 5
5. คำนวณค่า d ซึ่งสัมพันธ์กับสมการ e * d mod (p-1)(q-1) = 1 ซึ่งค่าที่ได้คือ d = 77 เนื่องจาก 77 * 5 = 385 = (4 * 96) + 1

ดังนั้น กุญแจสาธารณะ คือ {5, 119} และกุญแจส่วนตัว คือ {77, 119}

กำหนดให้ m เป็นข้อมูลที่ต้องการเข้ารหัส โดยสร้างข้อมูลใหม่ให้เป็น c

$c = m^e \bmod n$ เมื่อ (e, n) คือ กุญแจสาธารณะ

ทำการถอดรหัสข้อมูล โดย

$m = c^d \bmod n$ เมื่อ (d, n) คือ กุญแจส่วนตัว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.6 การแปลงข้อมูลโดยใช้การคำนวณแบบ MD5 (Message Digest Standard)

เป็นอัลกอริทึมที่รับข้อความที่มีความยาวไม่จำกัดเข้ามาแล้วได้ผลลัพธ์ออกมาเป็นแมสเซจไดเจสต์ (Message Digest) ที่มีขนาด 128 บิต โดยอาศัยฟังก์ชันทางคณิตศาสตร์ที่เป็นฟังก์ชันทางเดียว (Hash Function) มาใช้ในการทำงานของอัลกอริทึมนี้ ซึ่งในการที่ใช้ฟังก์ชันทางเดียวนั้น หมายถึงว่า มีความเป็นไปได้ยากมากที่จะมีข้อความ 2 ข้อความใด ที่สามารถสร้างออกมาเป็นแมสเซจไดเจสต์ได้เหมือนกัน และไม่สามารถสร้างข้อความใด ๆ จากแมสเซจไดเจสต์นี้ได้เลย

โดยเริ่มต้นที่สมมติว่า มีข้อมูลขนาด b บิตเป็นอินพุต (input) และต้องการที่จะหาแมสเซจไดเจสต์จากข้อความนี้ ซึ่ง b นั้นเป็นเลขจำนวนเต็มที่ไม่ติดลบและมีค่าไม่จำกัด

ขั้นตอนในการคำนวณ คือ

1. **การเติมเต็มบิตต่อท้าย (Padding Bit)**

เนื่องจากมีการแบ่งข้อความที่รับเข้ามาเป็นบล็อกๆ โดยมีขนาดบล็อกละ 512 บิต และในบล็อกสุดท้ายจะทำการเติมจำนวนบิตให้ครบ 448 บิต ซึ่ง 64 บิตที่เหลือนั้น ใช้สำหรับเติมความยาวบิตของข้อความ ในการเติมบิตต่อท้ายนั้น เริ่มต้นด้วยการเติมบิต “1” ก่อน แล้วจึงตามด้วยบิต “0” ต่อท้ายไปจนกว่าความยาวของบิตของข้อความในบล็อกนั้นครบ 448 บิต

2. **การเติมความยาวบิตของข้อความ**

ส่วนของ 64 บิตที่เหลือนั้น ไว้ใช้สำหรับเก็บค่าความยาวของข้อความที่เป็นอินพุต และเนื่องจากอินพุตไม่น่าจะมีขนาดเกิน 2^{64} บิต ดังนั้น 64 บิตจึงเพียงพอที่จะนำมาใช้ในการเก็บค่าความยาวบิตของข้อความ โดยเติมเวิร์ด (word) 32 บิตที่มีลำดับต่ำก่อนแล้วจึงตามด้วยเวิร์ดที่มีลำดับสูง

ดังนั้น ข้อความที่ได้ออกมาจะมีขนาดเป็นหลายเท่าของ 512 บิต หรือมีขนาดเป็นหลายเท่าของ 16 เวิร์ด (1 เวิร์ด มีขนาด 32 บิต) ให้ M[0 … N-1] แทนเวิร์ดของข้อความที่ผ่านกระบวนการต่อเติมแล้ว โดยที่ N คือ จำนวนเท่าของ 16

3. **การกำหนดค่าเริ่มต้นของบัฟเฟอร์ของแมสเซจไดเจสต์**

บัฟเฟอร์มีขนาด 128 บิต เพื่อเก็บผลลัพธ์ของการแฮช (Hash) โดยจะใช้บัฟเฟอร์ที่มีขนาด 32 บิต อยู่ 4 ตัว คือ A B C และ D โดยบัฟเฟอร์เหล่านี้ มีค่าเริ่มต้นในรูปของเลขฐาน 16 ดังนี้ ซึ่งเรียงลำดับจากไบต์ที่มีลำดับต่ำสุดก่อน

4. **การประมวลข้อความเป็นบล็อกขนาด 16 บิต**

เริ่มด้วยการนิยามฟังก์ชันช่วยที่รับอินพุตที่มีขนาด 32 บิต จำนวน 3 ตัวเข้ามา แล้วทำการสร้างผลลัพธ์ออกมาเป็นเวิร์ดขนาด 32 บิต

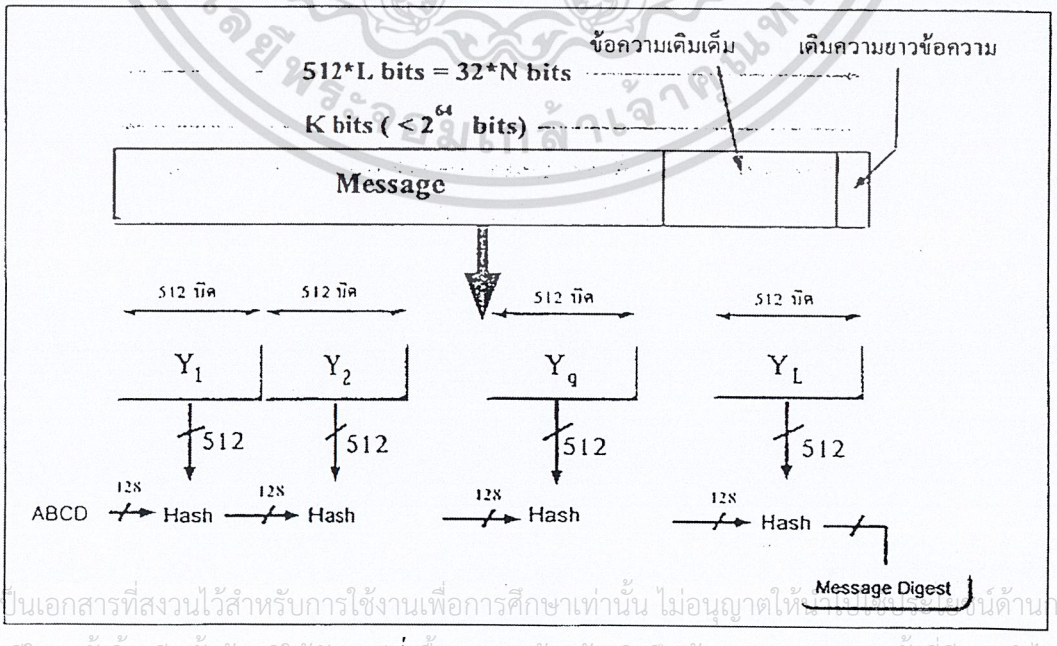
5. **กระบวนการหาแมสเซจไดเจสต์**

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* การประมวลผลของแต่ละบล็อก 16 เวิร์ด */
FOR i = 0 TO (N/16) - 1 DO
```

```
/* Copy บล็อก i ลงใน X */
FOR j = 0 TO 15 DO
        X[j] = M[ i * 16 + j ];
END /* ของ LOOP j */
/* บันทึกค่า A เก็บไว้ชั่วคราวที่ AA, B เก็บไว้ที่ BB, C เก็บไว้ที่ CC, D เก็บไว้ที่ DD */
        AA = A;          BB = B;          CC = C;          DD = D;
/* รอบที่ 1 * ให้ [ABCD k s i] แทนโอเปอเรชันต่อไปนี้ */
        A = B + ( (A + F(B, C, D) + X[k] + T[i]) <<< s) ทำ 16 โอเปอเรชัน
/* รอบที่ 2 * ให้ [ABCD k s i] แทนโอเปอเรชันต่อไปนี้ */
        A = B + ( (A + G(B, C, D) + X[k] + T[i]) <<< s) ทำ 16 โอเปอเรชัน
/* รอบที่ 3 * ให้ [ABCD k s i] แทนโอเปอเรชันต่อไปนี้ */
        A = B + ( (A + H(B, C, D) + X[k] + T[i]) <<< s) ทำ 16 โอเปอเรชัน
/* รอบที่ 4 * ให้ [ABCD k s i] แทนโอเปอเรชันต่อไปนี้ */
        A = B + ( (A + I(B, C, D) + X[k] + T[i]) <<< s) ทำ 16 โอเปอเรชัน
/* ทำการบวกค่าดังต่อไปนี้ (เพิ่มค่าในรีจิสเตอร์แต่ละตัวจากค่าที่มีอยู่ก่อนบล็อกนี้จะถูกประมวล) */
        A = A + AA;      B = B + BB;      C = C + CC;      D = D + DD;
END /* ของ LOOP i */
```

แมสเซจไดเจสต์ที่ได้คือ A B C และ D โดยเรียงลำดับเริ่มต้นที่ลำดับไบต์ต่ำสุดของ A (Low-Order Byte) และสิ้นสุดที่ลำดับไบต์สูงสุดของ D (High-Order Byte)



รูปที่ 2-6: การทำงานของ MD5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2.7 การเข้ารหัสข้อมูลสำหรับการสื่อสารในโพรโตคอลทีซีพี/ไอพี (TCP/IP)

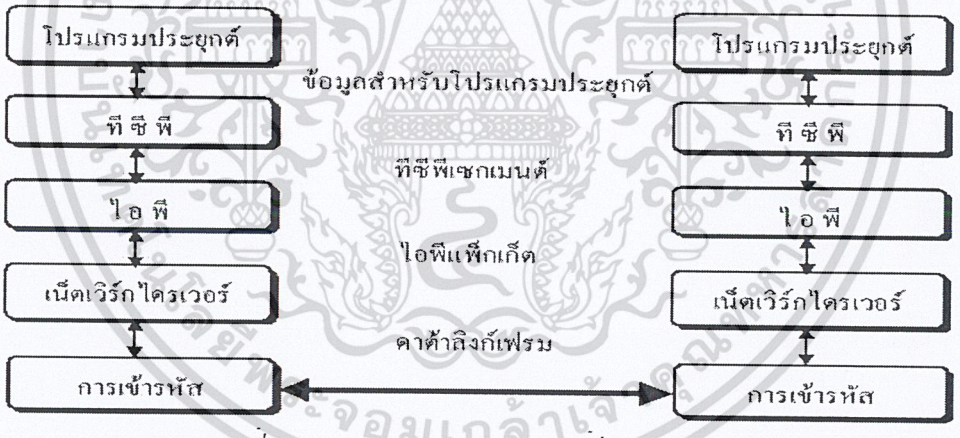
รูปแบบการเข้ารหัสสำหรับการสื่อสารข้อมูลในโพรโตคอลทีซีพี/ไอพีแบ่งออกเป็น 4 ระดับ คือ

- การเข้ารหัสระดับการเชื่อมโยงข้อมูล (Data Link Level Encryption)
- การเข้ารหัสระดับเครือข่าย (Network Level Encryption)
- การเข้ารหัสระดับทรานสปอร์ต (Transport Level Encryption)
- การเข้ารหัสระดับโปรแกรมประยุกต์ (Application Level Encryption)

2.7.1 การเข้ารหัสระดับการเชื่อมโยงข้อมูล

เป็นการเข้ารหัสในระดับดาต้าลิงค์ (Data Link) โดยส่วนมากจะใช้กล่องเข้ารหัส (Encryption Box) มาทำในส่วนของการเข้ารหัสข้อมูล ซึ่งวิธีการนี้ได้ถูกนำมาใช้ในระบบไซเฟอร์ (Cipher) หรือการใช้ดีไวซ์ไดรเวอร์ (Device Driver)

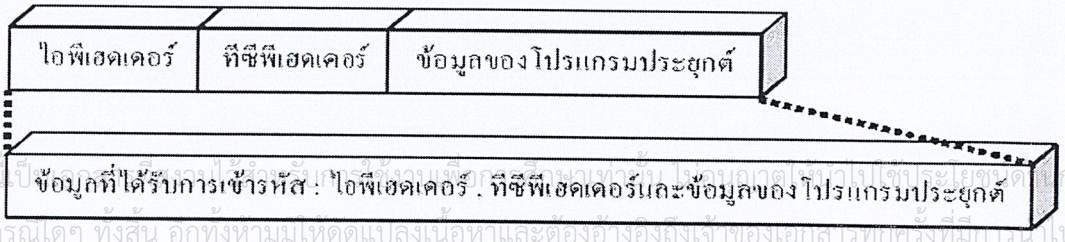
วิธีการนี้เป็นวิธีการที่ปลอดภัยที่สุด เนื่องจากข้อมูลที่ใช้งานและข้อมูลที่เป็นโครงสร้างของโพรโตคอลถูกเข้ารหัสทั้งหมด แต่วิธีการนี้ค่อนข้างมีค่าใช้จ่ายสูง เนื่องจากอุปกรณ์สำหรับการสื่อสารข้อมูลทั้งหมด จะต้องสนับสนุนการเข้ารหัสแบบนี้ด้วย



รูปที่ 2-7: การเข้ารหัสในระดับการเชื่อมโยงข้อมูล

2.7.2 การเข้ารหัสระดับเครือข่าย

เป็นการเข้ารหัสในส่วนของไอพีแพ็กเกจ (IP Packet) ซึ่งมีลักษณะดังรูป



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

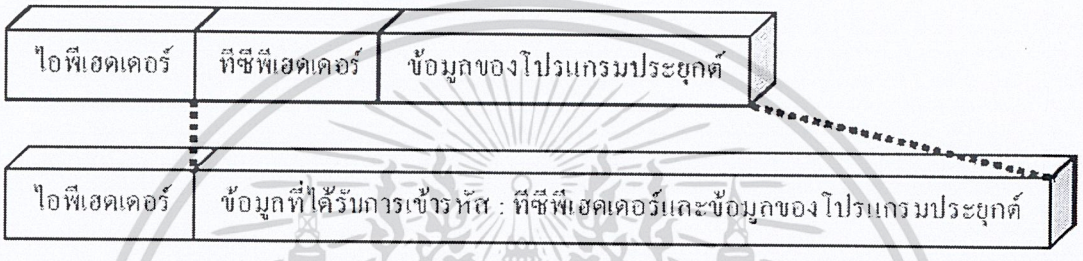
รูปที่ 2-8: การเข้ารหัสระดับเครือข่าย

2.7.3 การเข้ารหัสระดับทรานสปอร์ต

เป็นการเข้ารหัสในส่วนของทีซีพีเซ็กเมนต์ (TCP Segment) ได้แก่

- ทีซีพีเฮดเดอร์ (TCP Header) และ
- ข้อมูลของโปรแกรมประยุกต์

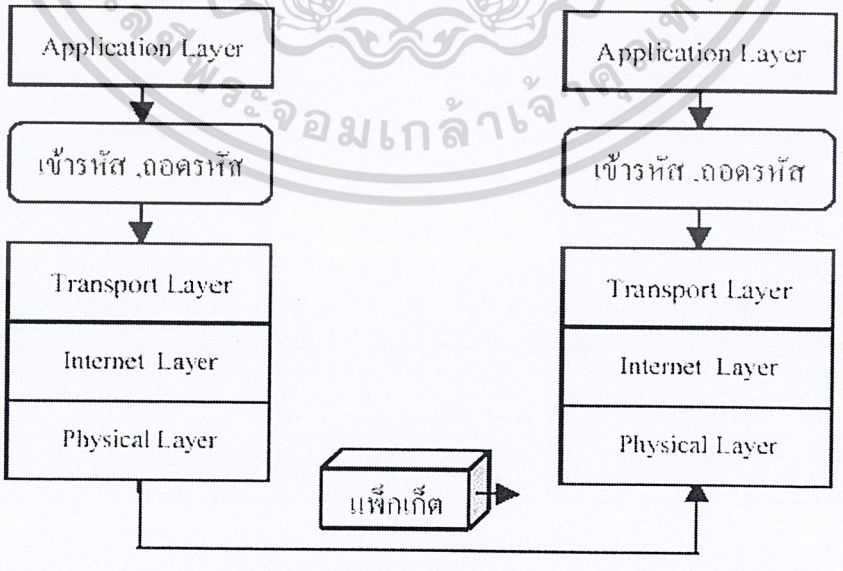
การเข้ารหัสในรูปแบบนี้ ทำให้การส่งข้อมูลไปตามเครือข่ายสามารถทำได้โดยไม่จำเป็นต้องเปลี่ยนแปลงอุปกรณ์ใดๆ เนื่องจากโครงสร้างในส่วนของไอพีไม่มีการเปลี่ยนแปลงใดๆ



รูปที่ 2-9: การเข้ารหัสระดับทรานสปอร์ต

2.7.4 การเข้ารหัสระดับโปรแกรมประยุกต์

เป็นการเข้ารหัสแบบจุดต่อจุด (End-to-End Encryption) คือ โปรแกรมประยุกต์จะเข้ารหัสในส่วนข้อมูลของตัวเอง ก่อนที่จะส่งผ่านไปยังระดับล่าง ซึ่งทำให้กระบวนการในระดับล่างไม่จำเป็นต้องมีการเปลี่ยนแปลงใดๆ



รูปที่ 2-10: การเข้ารหัสข้อมูลบนระบบโอเอสไอโมเดล (OSI Model) ของทีซีพี/ไอพี

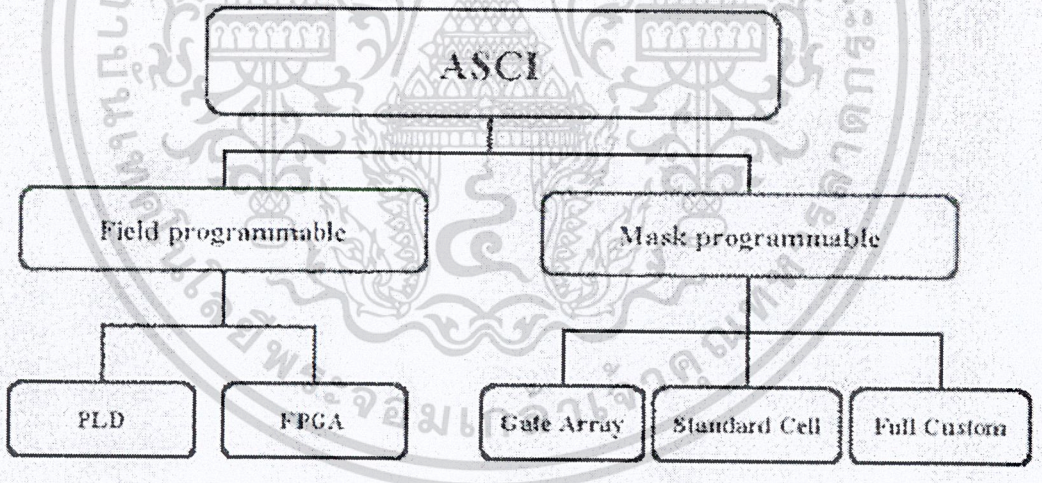
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
เอฟพีจีเอ

ในด้านของอุตสาหกรรมอิเล็กทรอนิกส์มีความก้าวหน้ามากขึ้นจากเดิม เป็นผลให้เกิดการพัฒนาความสามารถของอุปกรณ์ต่างๆมากกมาย ซึ่งสามารถลดค่าใช้จ่ายต่างๆ ลดการสิ้นเปลืองพลังงาน และลดขนาดของอุปกรณ์ลง แต่ในขณะเดียวกันก็ได้ทำการเพิ่มประสิทธิภาพและระดับความน่าเชื่อถือของอุปกรณ์ให้สูงขึ้นด้วย

นักออกแบบอุปกรณ์ทางด้านดิจิตอลได้พิจารณาถึงการผลิตโดยทำการเพิ่มความหนาแน่นและจำนวนของฟังก์ชันโลจิก (Logic Function) ให้เหมาะสม ซึ่งผลิตเป็นปริมาณมาก และทำการผลิตวงจรรวม (ASIC: Application Specific Integrated Circuit) ขึ้น โดยแบ่งวงจรรวมตามการสร้างออกได้เป็น 2 ชนิด คือ

- Field Programmable
- Mark Programmable



รูปที่ 3-1: ผังการแบ่งกลุ่มของวงจรรวม ASIC

3.1 Field Programmable

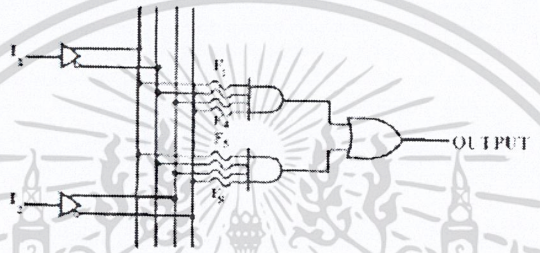
ผู้ใช้สามารถออกแบบและสร้างวงจรที่ต้องการใช้ได้เอง โดยไม่ต้องไปโรงงานเพื่อผลิต และในปัจจุบันนี้มีเครื่องมือที่ใช้ช่วยในการออกแบบและสร้างวงจรรวมที่มีความสามารถสูง ตั้งแต่ขั้นตอนการออกแบบ การจำลองการทำงาน จนถึงการจัดสร้างวงจรลงในอุปกรณ์ ซึ่งอุปกรณ์ Field programmable สามารถหาซื้อได้ง่าย เป็นผลให้การสร้างวงจรอิเล็กทรอนิกส์จนถึงระบบไมโครโพรเซสเซอร์ หันมาใช้อุปกรณ์จำพวกนี้เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อยๆแยกชิ้น (Discrete component) กันมากขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.1.1 พีแอลดี (PLD: Programmable Logic Device)

ภายในอุปกรณ์พีแอลดีถูกเตรียมเป็นวงจรพื้นฐานทางด้านโลจิกต่อกันอยู่เป็นกลุ่มมีทั้งวงจรคอมบิเนชัน (Combination) และซีเควนต์เชียล (Sequential) อุปกรณ์พีแอลดีทุกชนิดล้วนมีหลักการพื้นฐานของวงจรภายในที่เหมือนกัน โดยมีวงจรหลักเป็นวงจรคอมบิเนชันที่ให้ผลเป็นผลคูณร่วมบวก (Sum of product) ซึ่งประกอบไปด้วยชุดของแอนด์เกตที่ต่อร่วมกับออเกต

การโปรแกรม คือ การเลือกว่าจะให้มีการต่ออินพุตภายในของแอนด์เกตกับสัญญาณอินพุตใดบ้าง อินพุตต่างๆของอุปกรณ์ทุกตัว จะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าไม่ต้องการใช้สัญญาณใด จะตัดฟิวส์ทำให้สามารถโปรแกรมได้ครั้งเดียว



รูปที่ 3-2: วงจรพื้นฐานของอุปกรณ์พีแอลดีซึ่งอยู่ในรูปผลคูณร่วมบวก

3.1.2 แอลซีเอ (LCA: Logic Cell Array)

อุปกรณ์ชนิดนี้ถูกสร้างเป็นอาร์เรย์ที่ประกอบด้วยเกตจำนวน 1,200 – 1,800 เกต มีลักษณะของสถาปัตยกรรมที่ใกล้เคียงกับเกตอาร์เรย์ (Gate Array) โดยใช้โปรเซสแบบซีมอส 1.6 ไมครอนขึ้นโลหะคู่ (CMOS 1.6 microns double-layer metal) สามารถโปรแกรมและลบได้โดยใช้กระแสไฟฟ้า (Static RAM based) ภายในจัดเรียงเป็นเมทริกซ์ (Matrix) ของโลจิกเซลล์ (Logic Cell) ล้อมรอบด้วยอินพุต เอาต์พุตเซลล์ แต่ปัจจุบันได้พัฒนามาเป็น เอฟพีจีเอ (FPGA: Filed Programmable Gate Array) ซึ่งมีประสิทธิภาพความจุของเกตสูงมากขึ้น โดยสร้างออกมาเป็นอนุกรม (Series) ต่างๆ เช่น ตระกูล XC3000 และตระกูล XC4000 เป็นต้น

3.1.3 เอฟพีจีเอ (FPGA: Field Programmable Gate Array)

เป็นอุปกรณ์ที่พัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทไซรินซ์ (XILINX Inc.) ซึ่งสามารถจะกำหนดฟังก์ชันการทำงานได้ตามความต้องการของผู้ใช้โดยผ่านการกำหนดฟังก์ชันการทำงานของวงจร และเนื่องจากเอฟพีจีเอมีประสิทธิภาพการทำงานและปริมาณความหนาแน่นของเกตสูง ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอจึงสามารถออกแบบและทดสอบเสร็จภายในเวลาเพียง 2-3 วันเท่านั้น และการเปลี่ยนแปลงแก้ไข ก็ใช้เวลาไม่นาน ซึ่งประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขตัววงจร การเลื่อนเวลาการออกผลิตภัณฑ์ และลดค่าเอ็นอาร์อี (NRE: Non-Recurring Engineering cost) ลงไปด้วย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2 Mark Programmable

สำหรับการใช้งานวงจรรวม ASIC ในเชิงพาณิชย์นั้น จำเป็นต้องใช้วงจรรวม ASIC แบบ Mark programmable เนื่องจากว่า หากมีปริมาณการผลิตที่สูงเป็นหมื่นตัวนั้น จะมีราคาต้นทุนต่ำกว่าแบบ Field Programmable ดังนั้น การใช้งานวงจรรวมแบบ Mark Programmable จึงมีบทบาทสำคัญในการผลิตสินค้าอิเล็กทรอนิกส์ในเชิงพาณิชย์อย่างมากในปัจจุบัน

3.3 เทคโนโลยีของเอฟพีจีเอ

เป็นชิปที่สามารถทำการโปรแกรมได้ โดยการกำหนดุดเชื่อมต่อเพื่อสร้างจุดเชื่อมต่อภายในเพื่อประกอบเป็นวงจรได้ตามต้องการ ซึ่งสามารถแบ่งลักษณะของจุดเชื่อมต่อได้ดังนี้

1. Physical Changing
   - Fused คือ สามารถโปรแกรมได้เพียงครั้งเดียว หลังจากโปรแกรมแล้ว จุดเชื่อมต่อจะขาดจากกัน
   - Anti-fused คือ สามารถโปรแกรมได้เพียงครั้งเดียว หลังจากโปรแกรมแล้ว จุดเชื่อมต่อจะต่อถึงกัน
2. Memory-Based เป็นการใช้หน่วยความจำเป็นตัวเก็บข้อมูล เพื่อควบคุมจุดเชื่อมต่อต่างๆ
   - ROM based คือ สามารถโปรแกรมได้หลายครั้ง ซึ่งข้อมูลจะคงอยู่ได้โดยไม่ต้องใช้ไฟเลี้ยง มีความจุต่ำ ส่วนใหญ่ไม่เกิน 20,000 เกต
   - RAM based คือ สามารถโปรแกรมได้ตลอดไม่จำกัด แต่จำเป็นที่จะต้องใช้ไฟเลี้ยง มีความจุสูง ประมาณ 10,000 – 1,000,000 เกต

เหตุผลที่สามารถทำการออกแบบเอฟพีจีเอได้ง่ายและรวดเร็ว เพราะ

- ไม่ต้องรู้โครงสร้างของชิป เพียงแค่ทำการออกแบบโลจิกที่ต้องการเท่านั้น
- ใช้ภาษาเอชดีแอล (HDL) ในการออกแบบ หรือใช้ซอฟต์แวร์ มากำหนดลักษณะการทำงาน โดยภาษาที่เป็นที่นิยมใช้กันมากที่สุด คือ ภาษาวีเอชดีแอล ซึ่งสามารถทำให้ใช้ได้กับชิปทุกค่าย
- สามารถโปรแกรมให้ชิปทำงานได้โดยตรง โดยไม่จำเป็นต้องถอดตัวชิปออกมาข้างนอก หรือจะใช้เป็นบอร์ดโปรแกรมมาทำงานก็ได้ และสามารถทำการโปรแกรมได้หลายครั้ง ทำให้สามารถพัฒนาและปรับปรุงความมีประสิทธิภาพได้ง่าย
- IC FPGA ที่นิยมใช้และพอหาได้ ในบ้านเรา มีอยู่ 2 ค่ายใหญ่ด้วยกัน คือ ALTERA กับ XILINX

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เนื่องจากโครงงานชิ้นนี้ ได้ใช้เอฟพีจีเอตระกูล XC4000 มาเป็นอุปกรณ์ในส่วนของฮาร์ดแวร์ (Hardware) ดังนั้น จึงจะขอกล่าวถึงเอฟพีจีเอตระกูล XC4000 ให้ทราบ

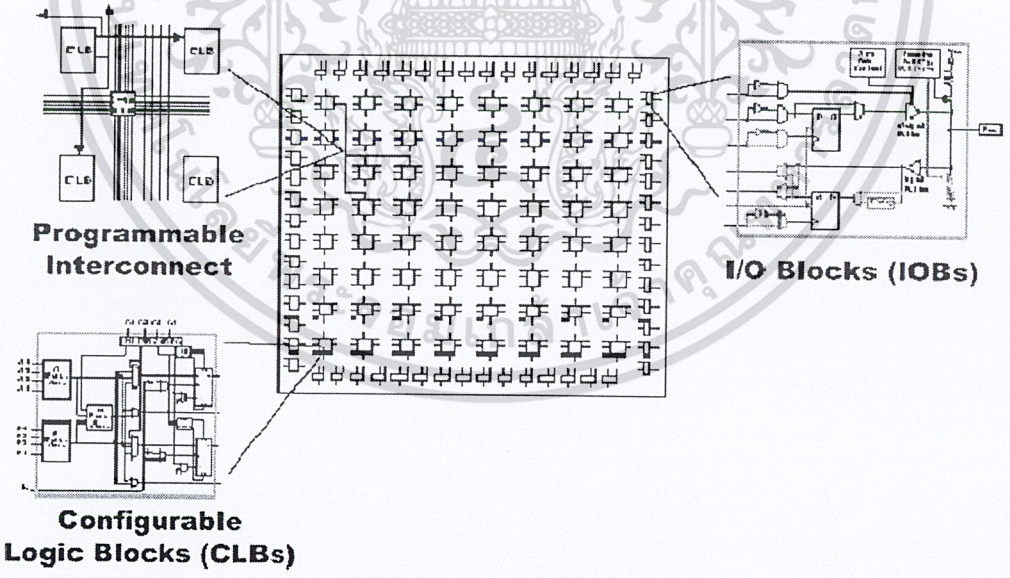
3.4 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000

เอฟพีจีเอเป็นชิปที่มีสนามวงจรขนาดใหญ่อยู่ภายใน มีความเร็วสูง และมีสถาปัตยกรรมการออกแบบคล้ายซีพีแอลดี (CPLD: Complex Programmable Logic Device) แต่มีส่วนประกอบที่ซับซ้อนและมีประสิทธิภาพมากกว่า โดยสามารถนำเอฟพีจีเอมาใช้และออกแบบวงจรต่างๆได้ตามที่เราต้องการโดยกระทำผ่านโปรแกรมที่ใช้ในการออกแบบ ซึ่งสามารถทำได้ง่าย ดังนั้น เอฟพีจีเอจึงเหมาะสมสำหรับการออกแบบวงจรเป็นอย่างมาก

สถาปัตยกรรมภายในของเอฟพีจีเอ สามารถแบ่งออกเป็น 3 ส่วน คือ

- ซีแอลบี (CLB: Configuration Logic Block)
- ไอโอบี (IOB: Input/Output Block) และ
- การเชื่อมต่อภายใน (Interconnect)

ภายในสถาปัตยกรรมของเอฟพีจีเอ จะมีลักษณะเป็นตารางโลจิกบล็อก (Logic Block) และล้อมรอบไปด้วยบล็อกของไอโอ (I/O Block) และการเชื่อมต่อระหว่างซีแอลบี กระทำโดยผ่านช่องว่างระหว่างแถวและคอลัมน์ (Column)



รูปที่ 3-3: วงจรพื้นฐานภายในเอฟพีจีเอ

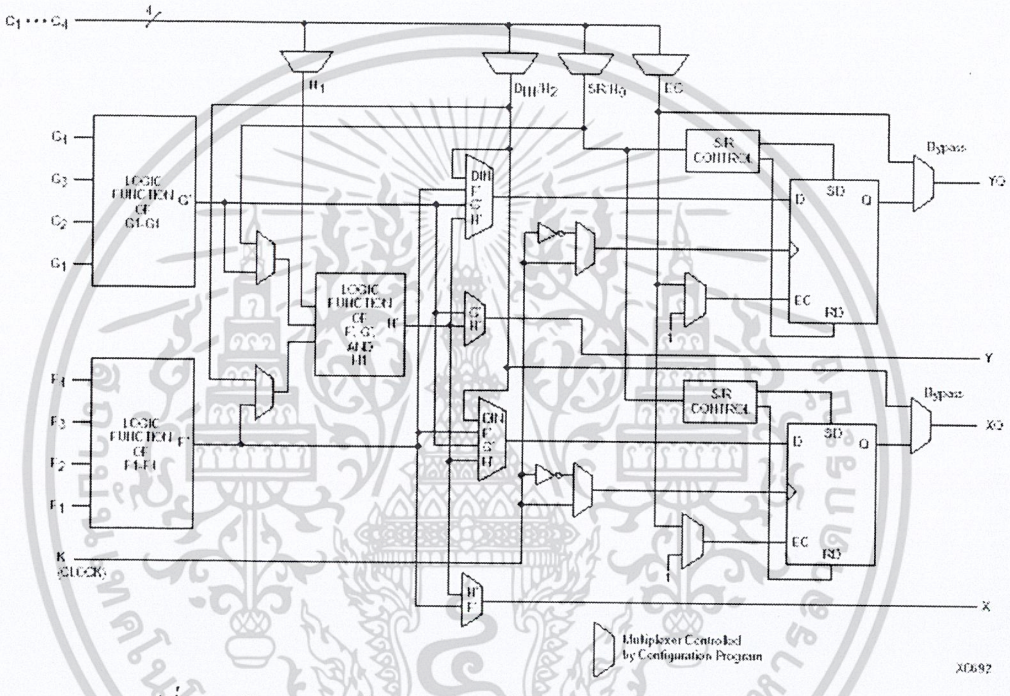
อุปกรณ์แอลซีเอทำงานได้โดยใช้ Program-driven logic device ซึ่งทำหน้าที่เป็นตัวระบุการทำงานของทั้ง 3 ส่วน ซึ่งค่าต่างๆได้ถูกกำหนดไว้ในโปรแกรมคอนฟิกกูเรชัน (Configuration Program) หรือเก็บไว้ในอีพรอม ซึ่งอยู่ภายในแอลซีเอ โดยแอลซีเอจะเริ่มต้นทำงานเมื่อตัวโปรแกรมถูกโหลดเข้าสู่แอลซีเอแล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประสิทธิภาพของแอลซีเอถูกกำหนดโดยความเร็วของโลจิก ส่วนประกอบของหน่วยความจำและการโปรแกรมการเชื่อมต่อต่างๆ

- **ซีแอลบี**

แต่ละตัวประกอบด้วยหน่วยของคอมบิเนชันโลจิก (Combination Logic) ที่สามารถโปรแกรมได้ โดยสามารถนำมาใช้สร้างวงจรทางด้านฟังก์ชันบูลีน (Boolean Function) ของอินพุตได้ และรีจีสเตอร์ (Register) ที่ทำหน้าที่เก็บข้อมูล ซึ่งรับค่าจากส่วนคอมบิเนชัน (Combination) หรือรับค่าโดยตรงจากเอาต์พุตของซีแอลบี

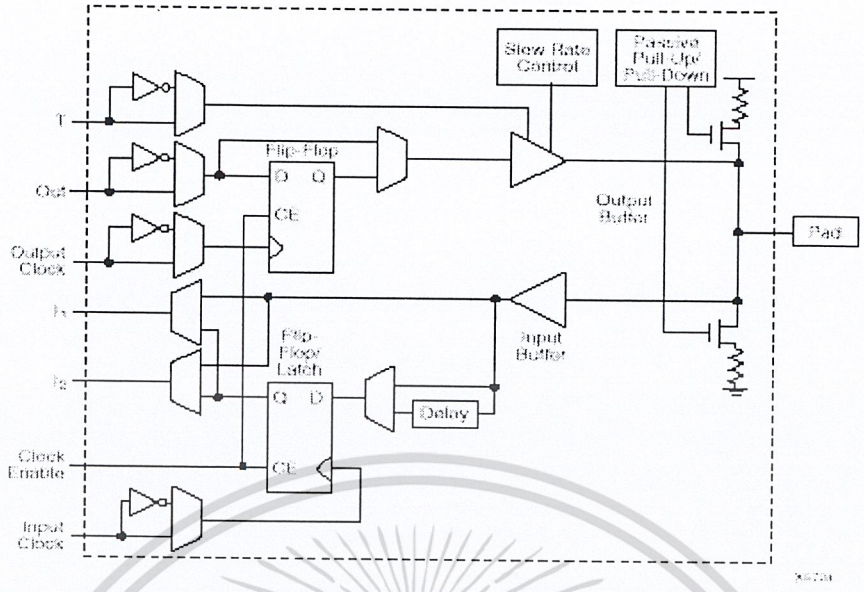


รูปที่ 3-4: ผังวงจรภายในของซีแอลบีของเอฟพีจีเอตระกูล XC4000

- **ไอโอบี**

เป็นส่วนที่ติดต่อกับวงจรภายนอกของแอลซีเอ ซึ่งมาจากส่วนของอุปกรณ์อินพุต และเอาต์พุตที่สามารถโปรแกรมได้ (Programmable I/O device) โดยแต่ละตัวโปรแกรมได้อย่างอิสระ สามารถให้เป็นอินพุตและเอาต์พุตแบบ 3 สถานะ หรือไอโอ (I/O: Input/Output) แบบสองทิศทางก็ได้ โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอล (TTL) และซีมอสเทรดโฮลของไอโอบี ซึ่งแต่ละตัวล้วนมีฟลิปฟลอป (flip-flop) ที่สามารถใช้เป็นบัฟเฟอร์ (Buffer) สำหรับอินพุตและเอาต์พุตได้

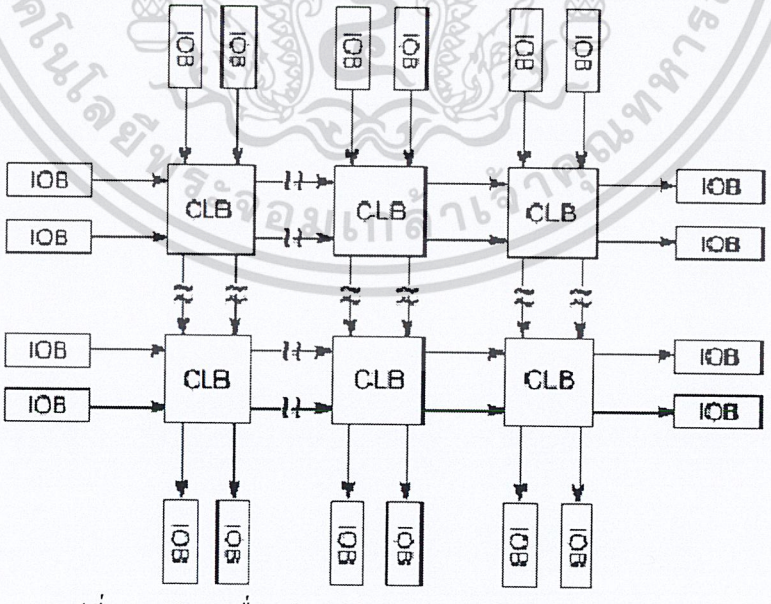
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 3-5: ผังวงจรภายในของไอโอบีของเอฟพีจีเอตระกูล XC4000

- **การเชื่อมต่อภายใน**

การเชื่อมต่อภายในแอลซีเอ ประกอบด้วยเน็ตเวิร์ค (Network) 2 ทิศทาง คือ ทางแถว และทางคอลัมน์ ซึ่งวางอยู่ระหว่าง CLB programmable switch และจะทำการเชื่อมต่ออินพุตและเอาต์พุตของไอโอบี และซีแอลบีที่จุดเชื่อมต่อร่วมระหว่างแถวกับคอลัมน์ ซึ่งสามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่างๆได้



รูปที่ 3-6: ผังการเชื่อมต่อภายในของเอฟพีจีเอตระกูล XC4000

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

3.5 คุณสมบัติของเอฟพีจีเอตระกูล XC4000

1. เป็นอุปกรณ์รุ่นที่ 3 ของเอฟพีจีเอ
   - มีฟลิปฟลอปจำนวนมาก
   - การผลิตฟังก์ชันการทำงานมีความยืดหยุ่นสูง
   - มีจำนวนเกตภายใน 2,000 – 10,000 เกต
   - เพิ่มความสามารถพิเศษของรีจีสเตอร์ และไอโอ
   - มีค่าแฟนเอาต์ (fan-out) สูง
   - มีบัส (Bus) ภายใน 3 สถานะ
   - สามารถทำงานได้กับสัญญาณทีทีแอล และซีมอส
   - มีออสซิลเลเตอร์แอมพลิฟายเออร์ (Oscillator Amplifier) ภายใน
   - มีแรมความเร็วสูงอยู่ภายใน (< 25 ns)
   - ใช้กับงานที่มีความเร็วสูง (ใช้งานได้ที่ความถี่ 70/ 100/ 125 MHz)
   - มี Wide edge decoder
   - เส้นทางการเชื่อมต่อ (Interconnect line) เป็นแบบลำดับชั้น
   - มีการกระจายกำลังของสัญญาณต่ำ
2. มีสถาปัตยกรรมภายในที่ยืดหยุ่น
   - มีโลจิกบล็อกและไอโอบล็อกที่สามารถโปรแกรมได้
   - มีการเชื่อมต่อภายในและ Wide decoder ที่โปรแกรมได้
3. ทำกระบวนการซับไมครอน (Sub-Micron) ชนิดซีมอสได้
   - มีโลจิกและการเชื่อมต่อภายในที่มีความเร็วสูง
   - ใช้พลังงานต่ำ
4. คุณลักษณะทาง System-Orient
   - รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary-scan logic
   - สามารถโปรแกรมค่า Output slew rate ได้
   - สามารถโปรแกรมอินพุตให้มีลักษณะพลูอัพ (Pull-up) หรือพลูดาวน์ (Pull-down) รีจีสเตอร์ได้
   - ให้กระแสเอาต์พุตได้ตั้งแต่ 12 – 24 มิลลิแอมป์ (milli - amp) ซึ่งขึ้นอยู่กับแต่ละรุ่น
5. สามารถทำการโหลดเอาแฟ้มข้อมูลประเภทไบนารี (Binary) ได้
   - ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
   - มีโหมดในการโปรแกรมให้เลือก 6 โหมด
6. มีโปรแกรมช่วยพัฒนา คือ Foundation series
   - สามารถติดต่อกับโปรแกรมอื่นๆได้ เช่น Viewlogic ,Mentor Graphic และOrCAD เป็นต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

49997

- มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ (Automatic place and routing) ที่ครบสมบูรณ์
- มี Interactive Design Editor ที่ใช้สำหรับการทำ Optimization
- มี 288 มาโคร (macro) 34 ฮาร์ดมาโคร (hard-macro) และแรม/ รอมคอมไพเลอร์ (RAM/ ROM Compiler)

Device	Max Logic Gate (No RAM)	Max RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total Logic Blocks	Number of Flip-Flops	Max Decode Input per Side	Max User I/O
XC4003L	3,000	3,200	2,000 - 5,000	10x10	100	360	30	80
XC4005E/L	5,000	6,272	3,000 - 9,000	14x14	196	616	42	112
XC4006L	6,000	8,192	4,000 - 12,000	16x16	256	768	48	128
XC4008L	8,000	10,368	6,000 - 15,000	18x18	324	936	54	144
XC4010E/L	10,000	12,800	7,000 - 20,000	20x20	400	1,120	60	160
XC4013E/L	13,000	18,432	10,000 - 30,000	24x24	576	1,536	72	192
XC4020L	20,000	25,088	13,000 - 40,000	28x28	784	2,016	84	224
XC4025L	25,000	32,768	15,000 - 45,000	32x32	1,024	2,560	96	256
XC4028EX/XL	28,000	32,768	18,000 - 50,000	32x32	1,024	2,560	96	256
XC4036EX/XL	36,000	41,472	22,000 - 65,000	36x36	1,296	3,168	106	288
XC4044EX/XL	44,000	51,200	27,000 - 80,000	40x40	1,600	3,840	120	320
XC4052XL	52,000	61,952	33,000 - 100,000	44x44	1,936	4,576	132	352
XC4062XL	62,000	73,728	40,000 - 130,000	48x48	2,304	5,376	144	384
Larger Devices Available in the First Half of 1997								

* Max value of Typical Gate Range include 20 - 30% of CLBs used as RAM

ตารางที่ 3-1: รายละเอียดของอุปกรณ์ภายในของเอฟพีจีเอตระกูล XC4000

3.6 การโปรแกรมเอฟพีจีเอตระกูล XC4000

คือ กระบวนการในการโหลดข้อมูลในโปรแกรมไปยังแอลซีเอ เพื่อกำหนดหน้าที่การทำงานในแต่ละบล็อกภายใน และการเชื่อมต่อต่างๆ ซึ่งเอฟพีจีเอตระกูล XC4000 จะต้องใช้ข้อมูลเกี่ยวกับการโปรแกรมประมาณ 350 บิตต่อซีแอลบี โดยแต่ละบิตจะบอกถึงสถานะของหน่วยความจำสแตติก (Static) ที่ควบคุมตารางฟังก์ชัน (Function table bit) และลักษณะการมัลติเพล็กซ์ (Multiplex) อินพุต หรือการเชื่อมต่อกันระหว่างทรานซิสเตอร์ (Transistor)

เอฟพีจีเอ ในตระกูล XC4000 มีโหมดการโปรแกรม 6 โหมด ซึ่งแต่ละโหมดจะถูกกำหนดโดยบิต M0 M1 และ M2

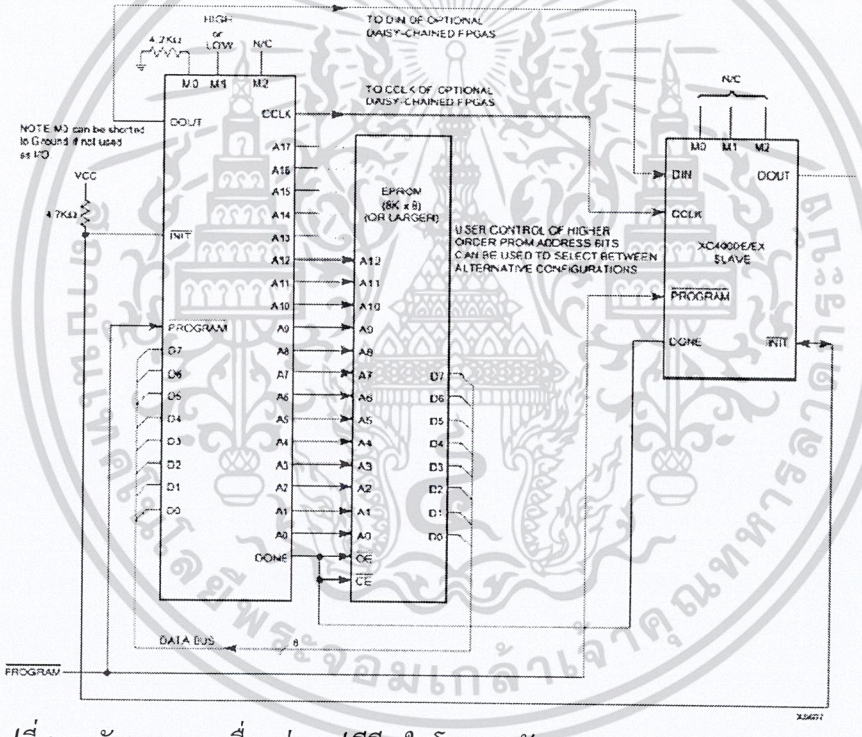
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Mode	M2	M1	M0	Clock	Data
Master Serial	0	0	0	Output	Bit-serial
Slave Serial	1	1	1	Input	Bit-serial
Master Parallel up	1	0	0	Output	Byte-Wide, 00000
Master Parallel down	1	1	0	Output	Byte-Wide, 3FFFF
Peripheral Synch.	0	1	1	Input	Byte-Wide
Peripheral Asynch.	1	0	1	Output	Byte-Wide

ตารางที่ 3-2: โหมดการโปรแกรม 6 โหมดของเอฟพีจีเอ ตระกูล XC4000

1. โหมดหลัก (Master modes)

ตัวแอลซีเอ จะทำการโหลดค่าโครงสร้างที่กำหนดไว้ (Configuration data) จากหน่วยความจำภายนอกเข้ามาโดยอัตโนมัติ



รูปที่ 3-7: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดหลักแบบขนาน (Master Parallel mode)

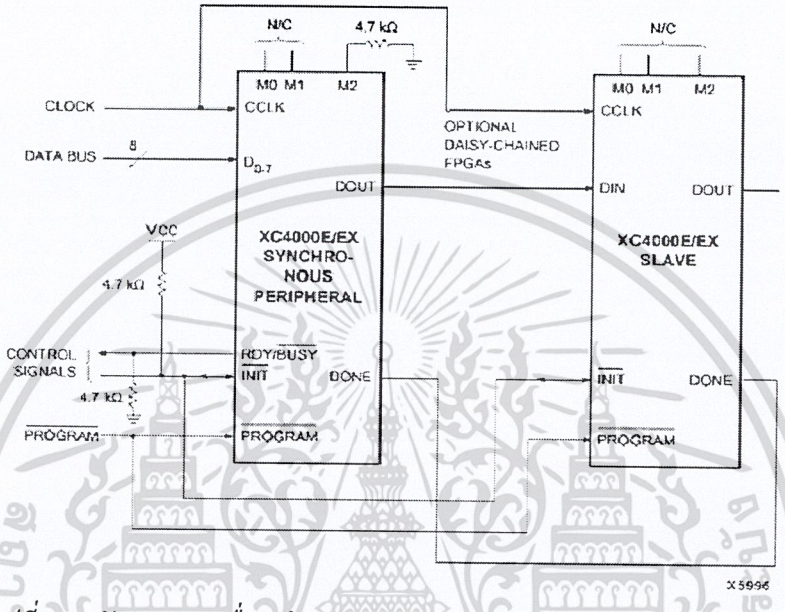
ในโหมดหลักแบบขนาน ไบต์ข้อมูล (Data byte) จะถูกอ่านเข้ามาแบบขนานทุกๆ สัญญาณอาร์คล็อก (RCLK: Read Clock) และส่งเข้าไปภายในแบบอนุกรม โดยอาศัยสัญญาณนาฬิกาของโครงสร้างที่กำหนดไว้ (Configuration Clock)

2. เพอริเฟอรัลโหมด (Peripheral modes)

โดยจะรับข้อมูลสถานะ (Ready/ Busy) เป็นไบต์จากบัส เพื่อทำให้สัญญาณต่างๆ สื่อสารกันได้อย่างมีประสิทธิภาพมากขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

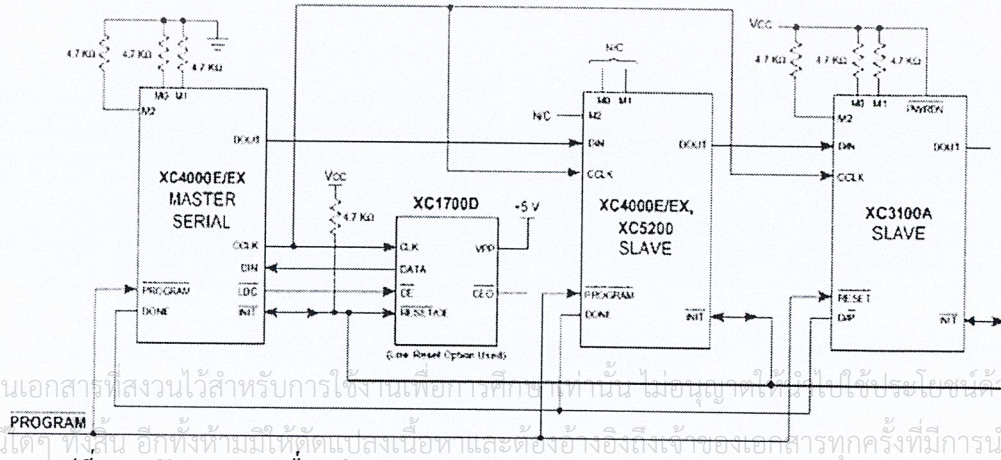
- อะซิงโครนัสโหมด (Asynchronous mode) นั้น ออสซิลเลเตอร์ภายในจะทำการสร้างสัญญาณ CCLK เพื่อรับข้อมูลที่เป็นไบต์ให้อยู่ในรูปแบบที่เหมาะสม
- ซิงโครนัสโหมด (Synchronous mode) นั้น สัญญาณนาฬิกาภายนอกจะเป็นตัวควบคุมการรับข้อมูลที่เป็นไบต์



รูปที่ 3-8: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดเพอริเฟอรัลแบบอะซิงโครนัส (Peripheral Asynchronous mode)

3. โหมดรองแบบอนุกรม (Slave Serial modes)

โหมดนี้ แอลซีเอจะทำการรับค่าโครงสร้างที่กำหนดไว้ที่เป็นแบบอนุกรมในขาขึ้นของสัญญาณ CCLK และหลังจากนั้น จะทำการส่งข้อมูลเพิ่มเติมออกไปด้วย และตัวแอลซีเอจะถูกทำการซิงโครนัสในขาถัดลงไปของสัญญาณ CCLK



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 3-9: ผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดรองแบบอนุกรม (Slave Serial mode)

3.7 ความสามารถของแรม (RAM) ในเอฟพีจีเอตระกูล XC4000

แอลซีเอทำงานโดยใช้ตารางการค้นหา (Look-up Table) ซึ่งจะเก็บตารางนี้ไว้ใน สแตติกแรม (Static RAM) ซึ่งจะถูกเขียนในระหว่างการโปรแกรม และจะถูกนำมาใช้ในระหว่างการทำงาน ดังนั้นแรมจึงถูกรวมไว้ในส่วนที่ผู้ใช้ควรจะออกแบบด้วย

หน้าที่ของแรมในเอฟพีจีเอตระกูล XC4000 มีหน้าที่คล้ายกับแรมโดยทั่วๆไป แต่จะมีความเร็วสูงเหมือนกับเอสแรม (SRAM) จึงไม่จำเป็นต้องคำนึงถึงเวลาหน่วงของการเชื่อมต่อ (Interconnection delay) ในการออกแบบ

RAM Module	Equivalent Logic	XC4003	XC4005	XC4010
16x1	4 Input Function Generators (F or G)	200	392	800
32x1	Two 4 Input Function Generators and One 3 Input Function Generators (F + G + H)	100	196	400

ตารางที่ 3-3: จำนวนของแรมภายในเอฟพีจีเอตระกูล XC4000

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4
ไอเอสเอและสตับ

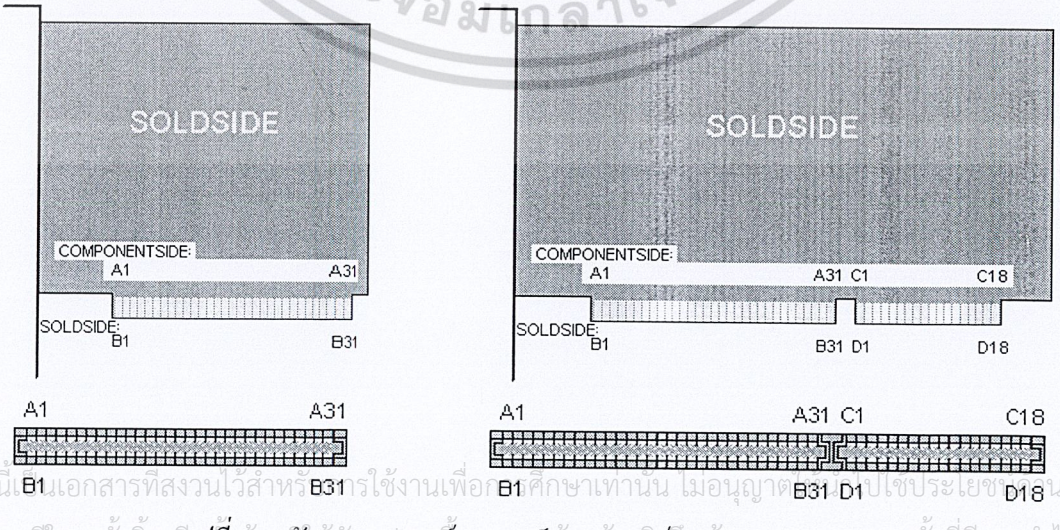
อุปกรณ์ที่เป็นฮาร์ดแวร์ในโครงงานนี้ที่ใช้ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์นั้น จะเชื่อมต่อกันผ่านทางสล็อต (Slot) ของไอเอสเอ (ISA) โดยผ่านทางไอเอสเอบัส (ISA Bus หรือ เรียกว่า ไอซ่าบัส) เพราะไอเอสเอถือเป็นพื้นฐานของการเชื่อมต่อข้อมูลภายใน (Internal Bus) เครื่องคอมพิวเตอร์ ซึ่งในส่วนของโปรแกรมการเข้ารหัสและในส่วนของไอเอสเอบัส จะมีปัญหาในการเชื่อมต่อการทำงานร่วมกัน จึงได้ทำการสร้างในส่วนของสตับ (Stub) ขึ้น เพื่อให้ทั้งสองส่วนสามารถทำงานร่วมกันได้

4.1 การ์ดไอเอสเอ (ISA Card)

หมายถึง การ์ดที่ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์โดยผ่านทางไอเอสเอบัส ซึ่งคำว่าไอเอสเอย่อมาจาก Industry Standard Architecture

4.1.1 ไอเอสเอบัส

ไอเอสเอบัส หรือเรียกอีกชื่อหนึ่งว่า เอทีบัส (AT Bus) เป็นระบบขนถ่ายข้อมูลอุปกรณ์ภายใน ซึ่งไม่ต้องการความเร็วสูงมากนัก ซึ่งทั้งพีซีไอ (PCI) ไอเอสเอ หรือ เอจีพี (AGP) ล้วนหมายถึง การ์ดเพิ่มขยาย (Expansion Card) ที่นำมาต่อกับบัสเพิ่มขยาย (Expansion Bus) เพื่อช่วยเพิ่มประสิทธิภาพและเพิ่มขีดความสามารถให้กับเครื่องคอมพิวเตอร์ เนื่องจากระบบบัสเพิ่มขยายสามารถปรับแต่งหรือเพิ่มขยายความสามารถของระบบได้ โดยผ่านทางปลั๊กอินบอร์ด (Plug-in Board) หรือเรียกว่าบัสเพิ่มขยาย เช่น เมื่ออยากให้คอมพิวเตอร์เล่นเพลงได้ ก็ต้องซื้อซาวนด์การ์ด (Soundcard) และลำโพง โดยแค่นำมาปลั๊กลงในสล็อตเพิ่มขยาย (Expansion Slot) บนเมนบอร์ด (Mainboard) และทำการติดตั้งก็สามารถใช้งานได้ทันทีโดยไม่จำเป็นต้องรื้อเมนบอร์ดใหม่

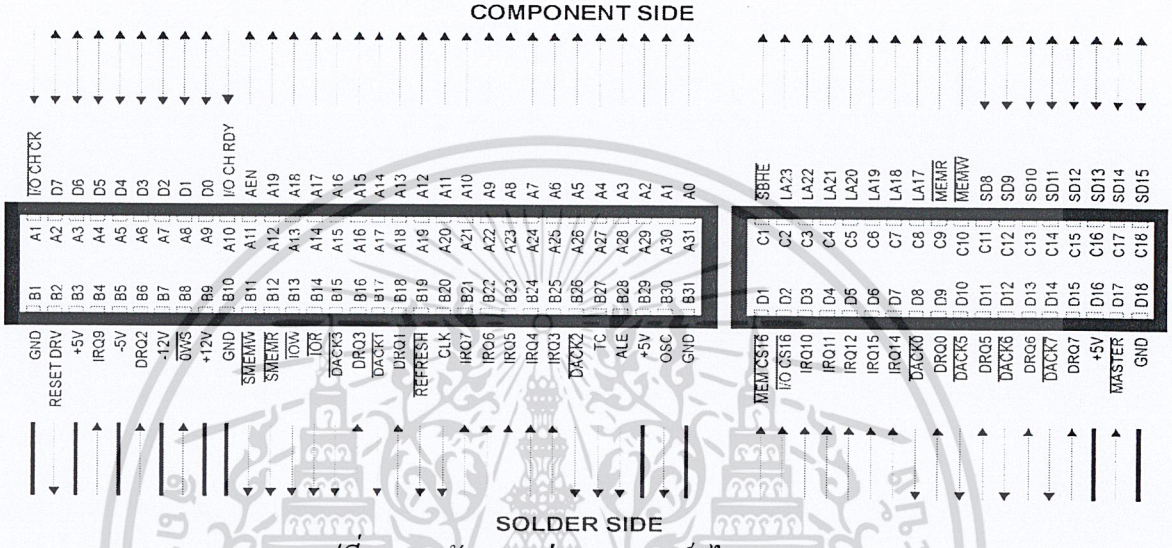


รูปที่ 4-1: ลักษณะของการ์ดและสล็อตของการ์ด 8 บิต และ 16 บิต

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การ์ดไอเอสเอนั้นมีอยู่ 2 ขนาด คือ การ์ดขนาด 8 บิต (Bit) และ 16 บิต ซึ่งเราสามารถนำการ์ดขนาด 8 บิตมาเสียบบนช่อง 16 บิตได้ เพราะใช้สถาปัตยกรรมพื้นฐานเหมือนกัน จะต่างกันตรงส่วนที่เพิ่มมาสำหรับ 16 บิตเท่านั้น

4.1.2 ขาสัญญาณต่าง ๆ ของการ์ดไอเอสเอ



รูปที่ 4-2: ขาสัญญาณต่าง ๆ ของการ์ดไอเอสเอ

จากรูปนั้น จะเห็นถึงขาสัญญาณ (Signal) ต่าง ๆ ของการ์ดไอเอสเอ ซึ่งจะแบ่งออกเป็น 2 ส่วน คือ

1. ส่วนคอมโพเน็นต์ (Component Side)
2. ส่วนโซลเดอร์ (Solder Side)

$\overline{\text{0WS}}$

Zero Wait State จะถูกเป็นเซตเป็น low โดยอุปกรณ์รองที่มีการใช้บัส (Bus Slave Device) เพื่อบอกว่า มันสามารถทำงานได้โดยที่ไม่ต้องมีเพิ่มสถานะ wait (Wait state) ซึ่งจะทำงานที่ไซเกิล (Cycle) ของหน่วยความจำเป็น 16 บิต โดยขาสัญญาณ –0WS ถูก derive มาจากการ decode แอดเดรส

AEN

Address Enable ถูกใช้เพื่อ degate ระบบไมโครโพรเซสเซอร์ (Microprocessor) และอุปกรณ์อื่น ๆ ผ่านบัสในระหว่างที่ DMA ส่งข้อมูล และเมื่อสัญญาณนี้ active ระบบตัวควบคุม DMA (System DMA Controller) จะสามารถควบคุมขา Address ขา Data และควบคุมขา Read และ Write ได้ ซึ่งควรที่จะรวมขาสัญญาณนี้ไปกับการเลือกบอร์ด เพื่อป้องกันการเลือกบอร์ดผิดระหว่างที่ DMA ทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใด ๆ ทั้งสิ้น ไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

BALE

Buffered Address Latch Enable ถูกใช้เพื่อ latch ขาสัญญาณ LA23 ถึง LA17 หรือ decode ของขาสัญญาณเหล่านี้ โดยขา Address จะถูก latch ที่ขอบขาลงของขาสัญญาณ BALE ซึ่งมันจะถูกเซตเป็น high ตลอดการทำงานของ DMA ซึ่งเมื่อนำมาใช้กับขาสัญญาณ AEN แล้วมันจะบ่งบอกตัวไมโครโพรเซสเซอร์หรือแอดเดรสของ DMA

CLK

System Clock เป็น คล็อก (Clock) ที่วิ่งโดยอิสระอยู่ในช่วง 8 ถึง 10 เมกะเฮิรตซ์ ซึ่งมันจะถูกใช้กับบางแอพพลิเคชันของบอร์ดไอเอสเอที่อนุญาตให้ทำงานเป็นซิงโครนัส (Synchronous) โดยระบบไมโครโพรเซสเซอร์

$\overline{\text{DACK0}}$ to $\overline{\text{DACK3}}$ and $\overline{\text{DACK5}}$ to $\overline{\text{DACK7}}$

DMA Acknowledge 0 ถึง 3 และ 5 ถึง 7 ถูกใช้เป็นสัญญาณ acknowledge ตอบรับการร้องขอขาสัญญาณ DRQ0 ถึง DRQ3 และ DRQ5 ถึง DRQ7 ของ DMA

DRQ0 to DRQ3 and DRQ5 to DRQ7

DMA Request ถูกใช้โดยบอร์ดไอเอสเอเพื่อทำการร้องขอบริการ (Request Service) จากระบบตัวควบคุม DMA หรือทำการร้องขอการเป็นเจ้าของบัสเพื่อนำบัสมาใช้งาน (สัญญาณเหล่านี้อาจจะถูกเซตให้ทำงานเป็นแบบอะซิงโครนัส (Asynchronous)) โดยอุปกรณ์ที่มาทำการร้องขอนั้น จะถูกเซตขาสัญญาณการร้องรอเป็น active ไปจนกว่าไอเอสเอบอร์ดจะตอบรับสัญญาณ DACK

$\overline{\text{I/O CS16}}$

I/O Chip Select 16 จะถูกเซตเป็น low โดยอุปกรณ์รองที่มีการติดต่อกับอินพุต-เอาต์พุต (I/O slave device) เพื่อบอกว่า มันสามารถทำการขนถ่ายข้อมูลที่เป็นอินพุต-เอาต์พุต 16 บิตได้ ซึ่งสัญญาณนี้ถูกเซตจากการ decode แอดเดรส SA15 ถึง SA0

$\overline{\text{I/O CH CK}}$

สัญญาณ I/O Channel Check อาจจะถูกกระตุ้น (โดยการร้องขอของบอร์ดไอเอสเอ) มากกว่าสัญญาณเอ็นเอ็มไอ (NMI: Non-Maskable Interrupt) ซึ่งถูก generate โดยระบบไมโครโพรเซสเซอร์ ซึ่งสัญญาณจะถูกเซตเป็น active เพื่อบอกถึงความผิดพลาดที่เกิดขึ้นเพื่อให้ทำการตรวจสอบ

I/O CH RDY

ในขณะที่มีการใช้ไอโอหรือหน่วยความจำอยู่ I/O Channel Ready จะเซตให้ตัวที่ต้องการจะใช้ไอโอหรือใช้หน่วยความจำเช่นกัน แต่ว่ามาทีหลังให้รอก่อนโดยการแทรกสถานะ wait เข้าไป ขาสัญญาณนี้จะถูกเซตเป็น high อยู่เสมอ (หมายถึงพร้อมทำงาน) ซึ่งบอร์ดไอเอสเอจะเซตขาสัญญาณนี้เป็น low เมื่อไม่พร้อมทำงานหรือทำงานไม่เสร็จและทำการเพิ่มสถานะ wait เข้าไป โดยตัวอุปกรณ์ที่มีการใช้ขาสัญญาณนี้ที่มีการแทรกสถานะ wait ควรที่จะเซตขาสัญญาณนี้เป็น low ทันทีที่ทำการตรวจพบว่ามีการใช้สัญญาณขา Address และมีการทำคำสั่งอ่านและ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เขียนอยู่ และจะเซตกลับมาเป็น high ใหม่เมื่ออุปกรณ์ทำไซเกิลของไอโอหรือหน่วยความจำที่เพิ่มสถานะ wait เสร็จแล้ว

$\overline{\text{IOR}}$

I/O Read จะถูกสั่งงานจากเจ้าของบัสขณะนั้น โดยจะสั่งให้อุปกรณ์อินพุต-เอาต์พุตที่ถูกเลือกทำการอ่านข้อมูลจาก Data bus

$\overline{\text{IOW}}$

I/O Read จะถูกสั่งงานจากเจ้าของบัสขณะนั้น โดยจะสั่งให้อุปกรณ์อินพุต-เอาต์พุตที่ถูกเลือกทำการเขียนข้อมูลลงบน Data bus

IRQ3 to IRQ7 and IRQ9 to IRQ12 and IRQ14 to IRQ15

Interrupt Request จะถูกใช้เป็นตัวให้สัญญาณแก่ตัวไมโครโพรเซสเซอร์ เมื่อบอร์ดไอเอสเอต้องการที่จะเรียกใช้งาน ซึ่งสัญญาณ IRQ จะทำการเปลี่ยนจาก low เป็น high จนกว่าไมโครโพรเซสเซอร์จะมีสัญญาณ ACK ตอบกลับการร้องขอ interrupt service routine กลับมา โดยจะใช้ความสำคัญของ IRQ9 ถึง IRQ12 และ IRQ14 ถึง IRQ15 เป็น highest priority และ IRQ3 ถึง IRQ7 เป็น lowest priority

LA23 to LA17

Unlatched Address bits 23:17 ใช้เป็นแอดเดรสเพื่อเข้าถึงหน่วยความจำภายในระบบ ซึ่งจะใช้งานร่วมกับขาสัญญาณ SA19 ถึง SA0 เพื่อให้สามารถอ้างอิงหน่วยความจำได้ 16 เมกะไบต์ โดยจะทำงานเมื่อขาสัญญาณ BALE เป็น high ซึ่งมันจะไม่ถูก latch และไม่นำมาใช้ในบัสไซเกิล โดย decode ของขาสัญญาณเหล่านี้ ควรที่จะถูก latch ที่ขอบขาลงของขาสัญญาณ BALE

$\overline{\text{MASTER}}$

Master ถูกใช้โดยบอร์ดไอเอสเอร่วมกับขาสัญญาณ DRQ เพื่อขอเป็นเจ้าของบัสเพื่อนำบัสมาใช้งาน โดยเมื่ออุปกรณ์รับสัญญาณ –DACK มา อุปกรณ์จะทำการเซตสัญญาณ –MASTER ให้เป็น low ซึ่งจะเป็นการอนุญาตให้ควบคุมขา Address ขา Data และขาควบคุมต่าง ๆ และหลังจากที่เซตสัญญาณ –MASTER ให้เป็น low แล้ว อุปกรณ์ควรที่จะรออีก 1 ช่วงคล็อก (Clock Period) ก่อนที่จะทำการใช้ขา Address และขา Data และรออีก 2 ช่วงคล็อกก่อนที่จะทำคำสั่งอ่านหรือเขียน

$\overline{\text{MEM CS16}}$

Memory Chip Select 16 จะถูกเซตเป็น low โดยอุปกรณ์รองที่มีหน่วยความจำ (Memory Slave Device) เพื่อบอกว่า มันสามารถทำการขนถ่ายข้อมูลที่เป็นหน่วยความจำ 16 บิตได้ ซึ่งสัญญาณนี้ถูกเซตจากการ decode แอดเดรส LA23 ถึง LA17

$\overline{\text{MEMR}}$

Memory Read ใช้ในการเลือกอุปกรณ์ที่มีหน่วยความจำ (Memory Device) ซึ่งจะเป็นการนำข้อมูลลงใน Data bus โดยขาสัญญาณจะ active อยู่ตลอดการทำงานของไซเกิลของการอ่านหน่วยความจำ (Memory Read Cycle)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$\overline{\text{MEMW}}$

Memory Write ใช้ในการเลือกอุปกรณ์ที่มีหน่วยความจำ ซึ่งจะเป็นการนำข้อมูลขึ้นมาจาก Data bus โดยขาสัญญาณจะ active อยู่ตลอดการทำงานของไซเกิลของการเขียนหน่วยความจำ (Memory Write Cycle)

OSC

Oscillator เป็นคล็อกที่มีช่วงเวลา 70 นาโนเซ็ก (ns) ซึ่งสัญญาณนี้ไม่ซิงโครนัสกับคล็อกของระบบ (System Clock: CLK)

$\overline{\text{REFRESH}}$

Memory Refresh จะถูกเซตเป็น low เพื่อบอกให้ทำการ refresh หน่วยความจำ

RESET DRV

Reset Drive จะถูกเซตเป็น high เพื่อทำการ reset หรือ ตั้งค่าโลจิกใหม่ (Initialize) เมื่อทำการเปิดเครื่องขึ้นมาใหม่ หรือมีการกด reset

SA19 to SA0

System Address bits 19:0 ใช้เป็นแอดเดรสที่เข้าถึงหน่วยความจำและอุปกรณ์อินพุต-เอาต์พุตภายในระบบ ซึ่งสัญญาณเหล่านี้อาจจะนำมาใช้ร่วมกับขาสัญญาณ LA23 ถึง LA17 เพื่อให้สามารถอ้างแอดเดรสของหน่วยความจำได้มากขึ้นถึง 16 เมกะไบต์ แต่มีเพียง 16 บิตล่างเท่านั้นที่ใช้เกี่ยวข้องกับตัวอินพุตและเอาต์พุต ซึ่งจะใช้อ้างแอดเดรสได้ 64K

ขา SA19 เป็นบิตที่มีสำคัญมากที่สุด (The Most Significant Bit) และขา SA0 เป็นบิตที่มีความสำคัญน้อยที่สุด (The Least Significant Bit) โดยสัญญาณเหล่านี้จะถูกนำเข้าสู่ระบบบัสเมื่อขาสัญญาณ BALE เป็น high และจะถูก latch ที่ขอบขาลงของขาสัญญาณ BALE ส่วนสัญญาณที่เหลือจะใช้สำหรับการ read และ write ซึ่งสัญญาณทั้งหมดจะถูกควบคุมโดยระบบไมโครโพรเซสเซอร์หรือ DMA หรือไม่ก็อาจจะโดยเจ้าของบัสขณะนั้นบนไอเอสเอบอร์ด

$\overline{\text{SBHE}}$

System Byte High Enable จะถูกเซตเป็น low เพื่อบอกถึงการขนถ่ายข้อมูลบน Data bus D15 ถึง D8

SD15 to SD0

System Data เปรียบเสมือนเป็น Data bus สำหรับอุปกรณ์บนไอเอสเอบัส ซึ่งขา SD15 เป็นบิตที่มีความสำคัญมากที่สุด และขา SD0 เป็นบิตที่มีความสำคัญน้อยที่สุด โดยที่ขาสัญญาณ SD7 ถึง SD0 ใช้ในการขนถ่ายข้อมูลของอุปกรณ์ที่มีขนาด 8 บิต ส่วนขาสัญญาณ SD15 ถึง SD0 จะใช้ในการขนถ่ายข้อมูลของอุปกรณ์ที่มีขนาด 16 บิต หากต้องการขนถ่ายข้อมูลของอุปกรณ์ที่มีขนาด 16 บิตโดยใช้ขาสัญญาณ SD7 ถึง SD0 ก็จะใช้ 2 ไซเกิล ซึ่งไซเกิลละ 8 บิต

$\overline{\text{SMEMR}}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

System Memory Read ใช้ในการเลือกอุปกรณ์ที่ใช้หน่วยความจำ ซึ่งจะเป็นการนำข้อมูลลงใน Data bus โดยขาสัญญาณจะ active เฉพาะตอนที่การ decode หน่วยความจำอยู่ภายใน

พื้นที่ของหน่วยความจำที่มีค่าต่ำกว่า 1 เมกะไบต์ ขาสัญญาณ SMEMR ถูก derive มาจาก MEMR และการ decode หน่วยความจำที่ค่าต่ำกว่า 1 เมกะไบต์

$\overline{\text{SMEMW}}$

System Memory Write ใช้ในการเลือกอุปกรณ์ที่ใช้หน่วยความจำ ซึ่งเป็นการนำข้อมูลมาจาก Data bus โดยขาสัญญาณจะ active เฉพาะตอนที่การ decode หน่วยความจำอยู่ภายในพื้นที่ของหน่วยความจำที่มีค่าต่ำกว่า 1 เมกะไบต์ ขาสัญญาณ SMEMW ถูก derive มาจาก MEMW และการ decode หน่วยความจำที่ค่าต่ำกว่า 1 เมกะไบต์

TC

Terminal Count เป็นตัวสร้าง pulse สัญญาณ โดย terminal count จะถูกส่งไปยัง DMA

4.1.3 ดีเอ็มเอ (DMA: Direct Memory Access)

การรับส่งข้อมูลกับหน่วยความจำโดยตรง โดยไม่มีการส่งข้อมูลผ่านทางซีพียู (CPU) เลย ซึ่งเป็นผลให้การรับส่งข้อมูลเร็วขึ้นและทำให้ซีพียูสามารถทำงานโปรเซสอื่น ๆ ต่อไปได้ ในการรับส่งข้อมูลแบบ DMA นั้น จำเป็นต้องมีการใช้แชนแนล หรือตัวควบคุม DMA (DMA Controller) โดยแชนแนลจะทำหน้าที่แทนซีพียู คือ เมื่อต้องการจะรับส่งข้อมูลแบบ DMA ตัวแชนแนลจะทำการส่งสัญญาณไปบอกให้ซีพียูรับทราบ ซึ่งซีพียูจะสั่งให้แชนแนลทำรูทีน (Routine) ควบคุมการส่งข้อมูล จากนั้นซีพียูจะไปทำงานอื่น และเมื่อการทำ DMA เสร็จแล้ว แชนแนลจะทำการส่งสัญญาณไปบอกให้ซีพียูรับทราบอีกครั้งว่า การทำ DMA เสร็จสิ้นแล้ว

ในขณะที่ซีพียูกำลังทำงานอื่นอยู่ จะไม่เกิดการทำ DMA ขึ้น เนื่องจากแชนแนลจำเป็นต้องใช้ดาต้าบัส (Data Bus) และแอดเดรสบัส (Address Bus) ของหน่วยความจำ และซีพียูเองก็จำเป็นต้องใช้บัสทั้งสองนี้เช่นกัน ดังนั้น DMA จะเกิดขึ้นเฉพาะในช่วงที่ซีพียูไม่ต้องการใช้หน่วยความจำ (ดาต้าบัสมีแอดเดรสว่างอยู่) เท่านั้น เช่น ซีพียูกำลังคำนวณฟังก์ชันทางคณิตศาสตร์อยู่ ซึ่งไม่มีความจำเป็นต้องใช้หน่วยความจำในช่วงเวลานี้ ดังนั้นดาต้าบัสและแอดเดรสบัสจึงว่างอยู่ ทำให้เกิดการทำ DMA ขึ้น ถ้าเมื่อใดซีพียูต้องการใช้หน่วยความจำ DMA จะต้องหยุดการทำงานลงชั่วคราว จนกว่าซีพียูจะใช้งานหน่วยความจำเสร็จ DMA ถึงจะสามารถทำงานต่อได้ ช่วงเวลาขณะที่ DMA ใช้บัสทั้งสอง ในขณะที่ซีพียูไม่ได้ใช้งาน เรียกว่า "การขโมยรอบเวลา (Cycle Stealing) "

<u>ชนิดของการใช้ขาสัญญาณของระบบบอร์ดไอเอสเอ (ISA System Board)</u>

Signal Name	System Board Usage	Signal Name	System Board Usage
AEN	Output	**$\overline{\text{MEM CS16}}$**	I/O
BALE	Output	**$\overline{\text{MEMR}}$**	I/O
CLK	Output	**$\overline{\text{MEMW}}$**	I/O

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$\overline{\text{DACK}}$	Output	**OSC**	Output
DRQ	Input	**$\overline{\text{REFRESH}}$**	I/O
$\overline{\text{I/O CS16}}$	Input	**RESET DEV**	Output
$\overline{\text{I/O CH CK}}$	Input	**SA**	I/O
I/O CH RDY	I/O	**SD**	I/O
$\overline{\text{IOR}}$	I/O	**$\overline{\text{SBHE}}$**	I/O
$\overline{\text{IOW}}$	I/O	**$\overline{\text{SMEMR}}$**	I/O
IRQ	Input	**$\overline{\text{SMEMW}}$**	I/O
LA	I/O	**TC**	I/O
$\overline{\text{MASTER}}$	Input	**$\overline{\text{0WS}}$**	Input

ตารางที่ 4-1: ชนิดของการใช้ขาสัญญาณ โดยระบบบอร์ดไอเอสเอ (ISA System Board)

4.2 สตับ (Stub)

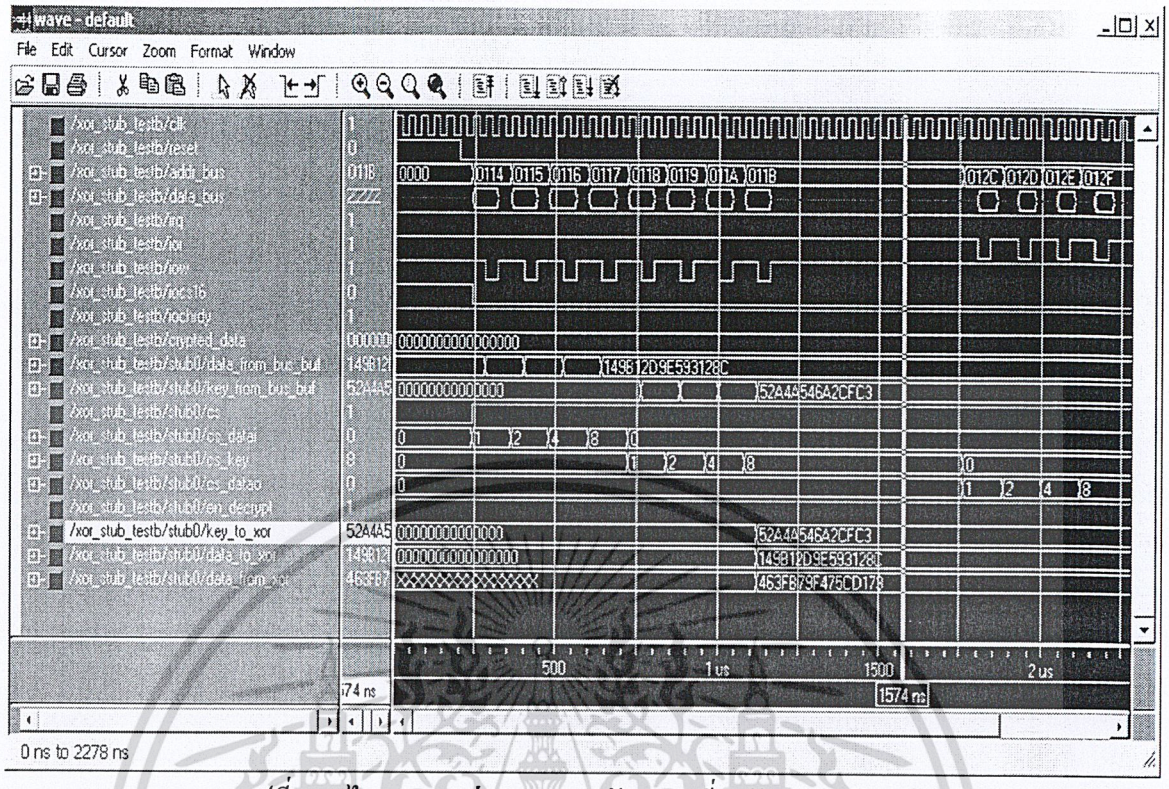
เป็นส่วนที่ทำหน้าที่เป็นตัวเชื่อมระหว่างกลางของสองฝั่ง เพื่อให้สามารถนำมาทำงานร่วมกันได้ โดยในขั้นตอนของการออกแบบสตับ เราได้แบ่งส่วนติดต่อออกเป็น 2 ฝั่งคือ

1) ฝั่งติดต่อกับส่วนของโปรแกรมการเข้ารหัส ในส่วนนี้จะทำการติดต่อกันภายในโปรแกรมภาษาวีเอชดีแอล (VHDL) เอง
2) ฝั่งติดต่อไอเอสเอบัสฝั่งนี้จะมีการเชื่อมต่อผ่านอุปกรณ์ทางกายภาพ(Physically) แต่เราจะเชื่อมต่อผ่านบัฟเฟอร์ (buffer) อีกทีนึง

สำหรับฝั่งติดต่อไอเอสเอบัส เราจะไม่เชื่อมต่อกับทุกเส้นแต่จะเลือกเฉพาะข้อมูลที่สำคัญต่อการเชื่อมต่อข้อมูล ซึ่งมีเส้นต่างๆดังนี้ Address(11:0), Data(15-0), I/O CHRDY, I/O CHCK, IOW, IOR, RESET DRV, IRQ, I/O CS16, AEN และอาจจะมีการเชื่อมต่อ เส้นข้อมูลอื่นๆ เท่าที่จำเป็นต่อไปในอนาคต

นอกจากส่วนติดต่อแล้ว ส่วนที่สำคัญที่สุดของสตับ คือส่วนขอ งบัฟเฟอร์ เพราะ สตับต้องควบคุมการทำงานระหว่างโปรแกรมการเข้ารหัสจากบอร์ดเอฟพีจีเอกับไอเอสเอบัส ที่มีความกว้างของข้อมูลไม่เท่ากัน ให้สามารถทำงานร่วมกันได้ด้วยดี ซึ่งข้อมูลที่สำคัญทั้ง 2 ฝั่งจะถูกเก็บไว้นานเท่าที่จำเป็น การติดต่อข้อมูลจากไอเอสเอบัสไปยังโปรแกรมการเข้ารหัส จะเป็นการติดต่อข้อมูลระหว่างบัฟเฟอร์ของทั้ง 2 ฝั่ง และตัวบัฟเฟอร์ถึงจะส่งข้อมูลไปยังส่วนติดต่อของตัวเองด้วย ช่วงเวลาที่สอดคล้องกับขนาดความกว้างของส่วนติดต่อเอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 4-3: ไดอะแกรมช่วงเวลาของสัญญาณที่สำคัญของสตับ

สัญญาณด้านที่ติดกับส่วนของบัส

clk, reset, addr_bus[], data_bus[], ior, iow, iocs16 และ iochrdy เป็นสัญญาณด้านที่ติดกับบัส ซึ่งความกว้างของขาสัญญาณจะเท่ากับความกว้างที่ใช้ในบัสนั้น ๆ ซึ่งในกรณีนี้ data_bus[] จะกว้างเป็น 16 บิต และ addr_bus[] จะกว้างเป็น 12 บิต และช่วงเวลาของสัญญาณจะขึ้นอยู่กับชนิดของบัสเช่นกัน สำหรับในกรณีของเรา จะเลือกใช้เฉพาะสัญญาณพื้นฐานที่จำเป็นของไอเอสเอบัส และการทำงานของช่วงสัญญาณจะเป็นไปตามมาตรฐานของไอเอสเอบัส ดังที่ได้กล่าวไว้ก่อนแล้ว

สัญญาณด้านที่ติดกับส่วนของการเข้ารหัส

data_from_bus_buf[], key_from_bus_buf[], cs_datai[], cs_key[] และ cs_data_out[] เป็นสัญญาณด้านที่ติดกับส่วนเข้ารหัส ซึ่งสัญญาณจะมีความกว้าง 64 บิต เท่ากับขนาดของบล็อกของการเข้ารหัสแบบ DES โดยช่วงของสัญญาณจะเป็นการส่งและรับข้อมูลขนาด 64 บิต และมีสัญญาณที่ใช้ในการบอกฝั่งปลายทางว่าข้อมูลนั้นเป็นข้อมูลที่ถูกต้องหรือไม่ เช่นสัญญาณ data_from_bus_buf[] ซึ่งจะใช้คู่กับสัญญาณ data_from_bus_buf_valid ถ้าสัญญาณ data_from_bus_buf_valid เป็น 1 แล้ว ฝั่งของส่วนการเข้ารหัส จะทำการบัฟเฟอร์ plain data อันนี้ไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
การเขียนดีไวซ์ไดรเวอร์

อุปกรณ์แต่ละชนิดย่อมมีลักษณะการทำงานที่แตกต่างกันออกไป ซึ่งโอเอส (OS) มีหน้าที่ควบคุมและติดต่อกับอุปกรณ์ต่าง ๆ ทุกประเภท ดังนั้นโอเอสจึงจำเป็นต้องรู้ลักษณะการทำงานของอุปกรณ์ทุกประเภท เพื่อจะได้สามารถควบคุมการทำงานของอุปกรณ์แต่ละประเภทได้ถูกต้อง แต่หากเรานำเอาโปรแกรมที่เป็นส่วนควบคุมอุปกรณ์ทั้งหมดมาใส่รวมไว้ในตัวโอเอส จะทำให้โอเอส มีขนาดใหญ่มาก ผู้ออกแบบโอเอสจึงมีความคิดที่จะแยกเอาส่วนที่ควบคุมอุปกรณ์ต่าง ๆ ออกจากตัวโอเอส ซึ่งโปรแกรมที่แยกออกมานี้ เรียกว่า ดีไวซ์ไดรเวอร์ (Device Driver) เมื่อโอเอสต้องการที่จะติดต่อกับอุปกรณ์ใด โอเอสจะทำการติดต่อผ่านทางดีไวซ์ไดรเวอร์ชนิดนั้นแล้วดีไวซ์ไดรเวอร์จะทำการติดต่อกับอุปกรณ์จริงเอง

5.1 ความหมายของดีไวซ์ไดรเวอร์

ดีไวซ์ไดรเวอร์เป็นโปรแกรมพิเศษที่ถูกโหลดเมื่อระบบทำงาน มีหน้าที่ในการเชื่อมต่อการทำงานระหว่างแอพพลิเคชัน (Application) ของผู้ใช้งานกับอุปกรณ์ฮาร์ดแวร์ (Hardware) ซึ่งควรออกแบบให้มีความยืดหยุ่นในการใช้งาน ไม่จำกัดการใช้งานฮาร์ดแวร์ โดยให้ตัวแอพพลิเคชันเป็นตัวเลือกการทำงานเอง ส่วนดีไวซ์ไดรเวอร์ทำหน้าที่เพียงจัดการทรัพยากร (Resource) ภายในของคอมพิวเตอร์ให้เหมาะสมแก่การใช้งาน

5.2 การสร้างไฟล์ของอุปกรณ์

ลีนุกซ์เคอร์เนล (Linux Kernel) จะทำการเรียกตัวดีไวซ์ไดรเวอร์ด้วยเลขเมเจอร์ (Major Number) และ เลขไมเนอร์ (Minor Number) โดยอุปกรณ์ที่ถูกควบคุมด้วยดีไวซ์ไดรเวอร์ตัวเดียวกัน อุปกรณ์เหล่านั้นก็จะมีเลขเมเจอร์เท่ากัน และเลขไมเนอร์จะใช้ในการจำแนกอุปกรณ์ภายในเลขเมเจอร์นั้น ๆ เช่น มีฟลอปปี้ดิสก์ (Floppy disk) 2 ตัวในเครื่อง ก็จะมีเลขเมเจอร์เป็น 2 โดยที่ฟลอปปี้ดิสก์ตัวแรกจะมีเลขไมเนอร์เป็น 0 และ อีกตัวหนึ่งเป็น 1 ซึ่งเคอร์เนลจะต้องทราบก่อนว่า มีอุปกรณ์นั้นมีอยู่จริงก่อนที่จะนำไดรเวอร์ของตัวนั้นมาใช้งาน โดยเมื่อทำการบูตระบบขึ้นมาแล้ว เคอร์เนลจะทำการสร้างไฟล์พิเศษขึ้นมาสำหรับแต่ละอุปกรณ์ในระบบโดยเก็บไว้ที่ไดเร็กทอรี (Directory) / dev ซึ่งไฟล์พิเศษนี้ จะถูกสร้างขึ้นด้วยคำสั่ง mknod

 # mknod /dev/device_name device_type major_number minor_number

- device_name คือ ชื่อของไฟล์
- device_type คือ ชนิดของอุปกรณ์ ว่าเป็น คาแร็กเตอร์ดีไวซ์ (c) หรือ เป็นบล็อกดีไวซ์ (b)
- เลขเมเจอร์ซึ่งสามารถดูได้จากไฟล์ชื่อ Documentation/devices.txt ในซอสโค้ดของเคอร์เนลที่ใช้อยู่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.3 ชนิดของอุปกรณ์

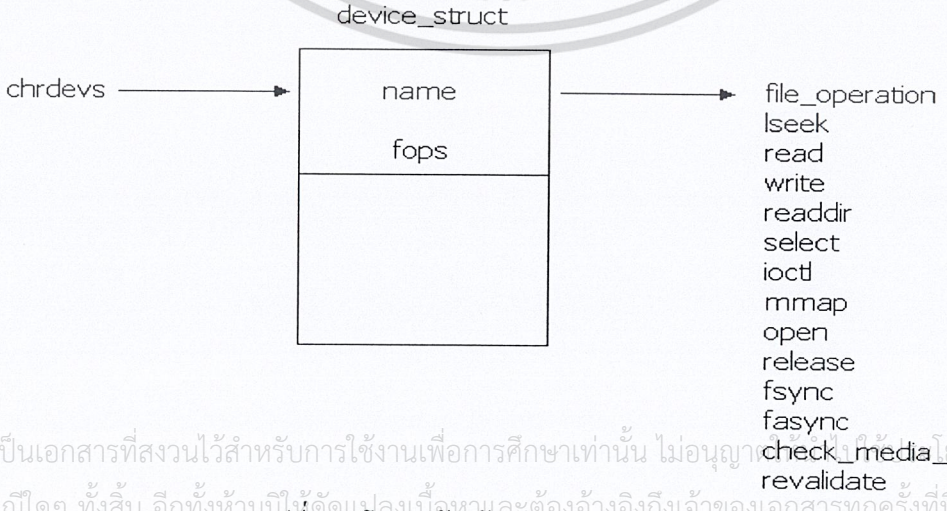
5.3.1 คาแร็กเตอร์ดีไวซ์ (Character Device)

อุปกรณ์ที่กระทำกับข้อมูลอินพุตและเอาต์พุต โดยไม่ใช้บัฟเฟอร์ (Buffer) ซึ่งอีกนัยหนึ่งคือทำการแบบสตีมมิ่ง ซึ่งรายชื่อของอุปกรณ์เหล่านี้ จะอยู่ในโครงสร้างข้อมูล device_struct ซึ่งถูกชี้ด้วยเวกเตอร์ (Vector) ที่ชื่อว่า chrdev ที่มีดัชนี (Index) เป็นเลขเมเจอร์ และอีกฟิลด์ (Field) หนึ่งของโครงสร้างข้อมูลนี้ คือ พอยน์เตอร์ (Pointer) ที่ชี้ไปยังฟังก์ชันการทำงานต่าง ๆ กับไฟล์ ซึ่งอยู่ในโครงสร้างข้อมูล file_operations โดยที่ใช้ฟังก์ชันพื้นฐานทำงานกับอุปกรณ์ได้เหมือนทำงานกับไฟล์ทั่วไป (มองอุปกรณ์เป็นไฟล์) ฟังก์ชันพื้นฐาน ได้แก่ open(), read(), write () และ close()

ซึ่งรูทีน (Routine) การทำงานของอุปกรณ์ที่ระบุในโครงสร้างข้อมูล file_operations เป็นดังนี้

```
struct file_operations {
        int (*lseek)    (struct inode *, struct file *, off_t, int);
        int (*read)     (struct inode *, struct file *, char *, int);
        int (*write)    (struct inode *, struct file *, const char *, int);
        int (*readdir)  (struct inode *, struct file *, struct dirent *, filldir_t);
        int (*select)   (struct inode *, struct file *, int, select_table *);
        int (*ioctl)    (struct inode *, struct file *, unsigned int, unsigned
long);
        int (*mmap)     (struct inode *, struct file *, struct vm_area_struct);
        int (*open)     (struct inode *, struct file *);
        int (*release)  (struct inode *, struct file *);
};
```

เมื่อแอพพลิเคชันต้องการใช้อุปกรณ์ จะต้องเปิดไฟล์ของอุปกรณ์โดยกระทำ ดังนี้

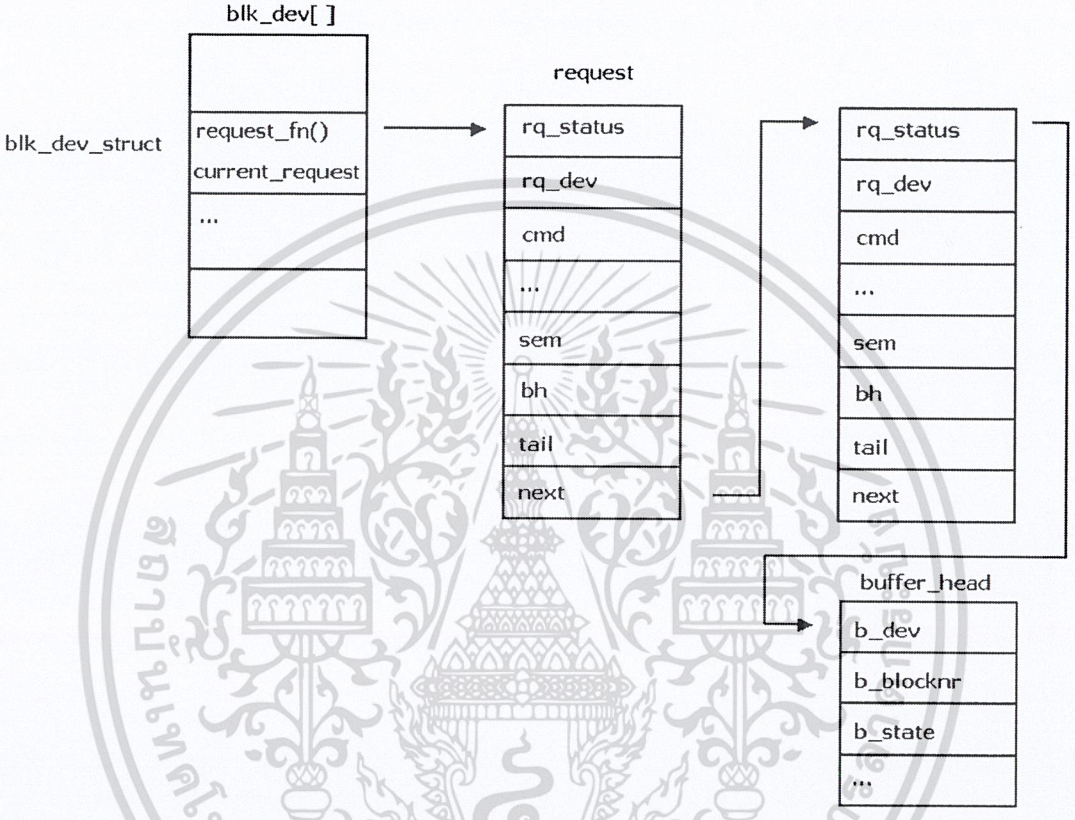


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 5-1: โครงสร้างข้อมูลของ Character Device

5.3.2 บล็อกดีไวซ์ (Block Device)

อุปกรณ์ที่กระทำกับข้อมูลอินพุต (Input) และเอาต์พุต (Output) เป็นบล็อก โดยกระทำผ่านบัฟเฟอร์ เพื่อให้สามารถเข้าถึงข้อมูลได้เร็วขึ้น ซึ่งส่วนใหญ่จะใช้เป็นสื่อในการส่งข้อมูล



รูปที่ 5-2: โครงสร้างข้อมูลของ Block Device

5.3.3 เน็ตเวิร์กดีไวซ์ (Network Device)

อุปกรณ์ที่ทำหน้าที่รับหรือส่งแพ็กเกตข้อมูล ซึ่งทำงานร่วมกับบีเอสดีซ็อกเก็ต(BSD Socket) ส่งผ่านข้อมูลโดยใช้ซ็อกเก็ตบัฟเฟอร์ (Socket Buffer) เพื่อประมวลผลตามมาตรฐานของโพรโตคอล (Protocol) ชั้นต่าง ๆ จนถึงการติดต่อกับอุปกรณ์ฮาร์ดแวร์จริง

5.4 การเขียนโปรแกรมดีไวซ์ไดรเวอร์บนระบบปฏิบัติการลีนุกซ์

การเขียนดีไวซ์ไดรเวอร์ คือ การเขียนโปรแกรมเพื่อรันในส่วนของลีนุกซ์เคอร์เนล ซึ่งไดรเวอร์จะรันบนเคอร์เนลโหมด (Kernel mode) โดยไดรเวอร์บนลีนุกซ์จะถูกแบ่งออกเป็น ๒ แบบคือแบบบิวลท์อิน(Built-in) และแบบโมดูล (Module) ซึ่งจะต่างกันที่ช่วงเวลาที่ใช้เรียกการทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลีนุกซ์โมดูลเป็นเซตของฟังก์ชันและชนิดของข้อมูล ซึ่งสามารถจะคอมไพล์ (Comply) แยกจากตัวเคอร์เนลได้ และจะถูกเรียกใช้งานหลังจากที่บูตเคอร์เนลแล้ว โมดูลจะใช้ในการอิมพลีเมนต์ (Implement) ดีไวซ์ไดรเวอร์ ซึ่งจะใช้ติดต่อกับอุปกรณ์เฉพาะของไดรเวอร์นั้นเท่านั้น

5.4.1 ขั้นตอนการเขียนโปรแกรมดีไวซ์ไดรเวอร์

1. เรียนรู้การทำงานภายในของโอเอสที่ใช้งาน เช่น วินโดวส์ หรือ ลีนุกซ์ เป็นต้น
2. เรียนรู้ว่าจะเขียนดีไวซ์ไดรเวอร์อย่างไรบนโอเอสที่ใช้งานอยู่ เช่น DDK, DDI หรือ DKI เป็นต้น
3. หาทูล (Tool) มาใช้สำหรับการพัฒนาและการดีบักในเคอร์เนลโหมด
4. ทำการเขียนดีไวซ์ไดรเวอร์บนเคอร์เนลโหมด ซึ่งเป็นไดรเวอร์พื้นฐานของฮาร์ดแวร์ที่เป็นส่วนติดต่อกับอินพุตและเอาต์พุต
5. ทำการเขียนแอพพลิเคชันในยูสเซอร์โหมด (User mode) ซึ่งสามารถจะใช้ฮาร์ดแวร์ผ่านทางดีไวซ์ไดรเวอร์ที่เขียนอยู่ในเคอร์เนลโหมดได้
6. ทำขั้นตอนที่ 1-4 ใหม่ สำหรับเมื่อมีการเปลี่ยนเป็นโอเอสตัวใหม่ ซึ่งควรที่จะสามารถรันโค้ดในข้อ 5 ได้

5.4.2 โปรแกรมโมดูล

แบ่งเป็น 2 ส่วนหลัก ๆ คือ

1. int init_modules() : เป็นฟังก์ชันที่ถูกเรียกใช้งานเมื่อมีการติดตั้งโมดูลเข้าไปในเคอร์เนล ใช้ในการติดตั้งค่าหรือเตรียมพร้อมการทำงานของโมดูลนั้น
   - การเรียกใช้งาน : insmod modules.o
2. void cleanup_modules() : เป็นฟังก์ชันที่ถูกเรียกใช้งานเมื่อต้องการกำจัดโมดูลออกจากเคอร์เนลของระบบ ใช้ในการจัดการให้ระบบมีสภาพเหมือนเดิม
   - การเรียกใช้งาน : rmmod modules

โครงสร้างของซอสโค้ดของโมดุลที่สามารถเรียกใช้งานได้จะต้องประกอบด้วย 2 ฟังก์ชันดังนี้

```
#include <linux/modules.h>
…
int init_modules() {
        /* Code to init the module */
}
…
void cleanup_modules() {
        /* Code to close the module */
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.4.3 การคอมไพล์ (Compliment)

 # gcc –c modules.c

ที่ต้องเป็น –c นั้น เนื่องจากต้องการทำให้เป็นออบเจ็กต์ไฟล์ (Object file) เพื่อที่จะสามารถเพิ่มหรือลบโมดูลไดรเวอร์ได้ เพราะเคอร์เนลไม่ได้รันเป็นโปรแกรมอิสระ และไม่ต้องทำการลิงก์ฟังก์ชันภายนอก

5.5 คาแร็กเตอร์ดีไวซ์ไดรเวอร์

การเขียนโปรแกรมไดรเวอร์ของคาแร็กเตอร์ดีไวซ์ เป็นแบบที่ง่ายที่สุด เนื่องจากไม่มีการเกี่ยวข้องกับบัฟเฟอร์ โดยไดรเวอร์จะทำงานในลักษณะของโมดูล คือ สามารถโหลดได้เมื่อเคอร์เนลต้องการใช้งาน ซึ่งทำงานเหมือนเป็นอุปกรณ์จริง

เคอร์เนลจะใช้เลขเมเจอร์เพื่อกำหนดไดรเวอร์ที่จะใช้กับอุปกรณ์ และดีไวซ์ไดรเวอร์จะใช้เลขไมเนอร์ในการแยกแยะอุปกรณ์แต่ละชิ้น ซึ่งการจะเข้าถึงเลขเมเจอร์และเลขไมเนอร์ได้นั้น ต้องใช้คำสั่ง inode -> i_rdev เพราะเลขทั้ง 2 ตัวนั้น เก็บไว้ในฟิลด์ i_rdev ที่อยู่ในโครงสร้าง inode โดยไดรเวอร์จะมีอาร์กิวเมนต์ (Argument) รับพอยน์เตอร์ที่ชี้ไปยังโครงสร้างข้อมูล inode

5.5.1 การเพิ่มและลบโมดูลไดรเวอร์ออกจากระบบ

การเพิ่มโมดูลไดรเวอร์ทำได้โดยใช้ฟังก์ชัน ซึ่งกำหนดไว้ในไฟล์ < linux/fs.h > ฟังก์ชันคือ

int register_chrdevs (unsigned int major, const char *name, struct file_operations *fops);

โดย name คือ ชื่อของอุปกรณ์

fops คือ พอยน์เตอร์ชี้ไปยังโครงสร้างข้อมูล file_opeartions

ซึ่งฟังก์ชันจะส่งค่ากลับเป็นจำนวนเต็มลบ ถ้ามี ERROR เกิดขึ้น เป็นศูนย์หรือจำนวนเต็มบวก ถ้าทำงานสำเร็จ

การลบโมดูลไดรเวอร์ออกจากระบบ จะเป็นการยกเลิกเลขเมเจอร์ที่ใช้งาน เพื่อที่ไดรเวอร์อื่นสามารถนำเลขเมเจอร์ไปใช้งานได้อีก ฟังก์ชัน คือ

int unregister_chrdev (unsigned int major, const char *name);

เคอร์เนลจะทำการเปรียบเทียบเลขเมเจอร์กับชื่อของอุปกรณ์ ในกรณีที่ไม่ตรงกันหรือเลขเมเจอร์มีค่ามากกว่าค่าที่กำหนดได้ จะส่งค่ากลับเป็น –EINVAL ในกรณีที่ไม่สามารถจะ unregister โมดูลได้ ตัวโปรเซส (Process) /proc/devices จะแสดงข้อผิดพลาดเมื่อทำการอ่านครั้งต่อไป เรียกว่า “Oops” ซึ่งเป็นข้อความที่แสดงเมื่อเคอร์เนลพยายามจะแอ็กเซส (Access) แอดเดรส (Address) ที่ไม่ถูกต้อง โดยต้องไม่ลืมลบโหนด (node) ที่เป็นชื่อของอุปกรณ์ออกจากระบบด้วย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.5.2 ฟังก์ชันที่ใช้ทำงานกับอุปกรณ์ (ไฟล์)

เคอร์เนลจะทำงานกับอุปกรณ์ผ่านรูทีนของไดรเวอร์ที่เก็บไว้ในโครงสร้างข้อมูล file_operations โดยในกรณีที่มีฟังก์ชันใดไม่ได้ทำงานกับอุปกรณ์ จะถูกเซตค่าเป็น NULL และจะส่งค่ากลับเป็น 0 เมื่อทำงานสำเร็จ หรือจำนวนเต็มลบเมื่อเกิด ERROR ขึ้น

สรุปการทำงานของฟังก์ชัน ดังต่อไปนี้

int (*lseek) (struct inode *, struct file *, off_t, int);

ใช้เปลี่ยนตำแหน่งในการอ่านหรือเขียนข้อมูลลงไฟล์ โดยค่าที่ส่งกลับจะเป็นจำนวนเต็มบวก หรือจุดออฟเซต (Offset) จากจุดเริ่มต้นไฟล์ ถ้าไม่มีการระบุฟังก์ชันนี้ จะทำให้ไดรเวอร์ไม่สามารถกำหนดตำแหน่งบนข้อมูลของอุปกรณ์ได้

int (*read) (struct inode *, struct file *, char *, int);

ใช้แอ็กเซสข้อมูลจากอุปกรณ์ ถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL จะทำให้ฟังก์ชันของระบบ read ไม่สามารถทำงานได้ จะเกิด ERROR –EINVAL โดยค่าที่ส่งกลับจะเป็นจำนวนไบต์ที่อ่านได้ เมื่อทำงานสำเร็จ

int (*write) (struct inode *, struct file *, const char *, int);

ฟังก์ชันจะส่งข้อมูลไปยังอุปกรณ์ โดยค่าที่จะส่งกลับจะเป็นจำนวนไบต์ที่เขียนได้

int (*readdir) (struct inode *, struct file *, void *, filldir_t);

ฟังก์ชันนี้ ควรจะกำหนดให้เป็น NULL สำหรับไฟล์ node ของอุปกรณ์ เนื่องจากปกติจะใช้อ่านข้อมูลในไดเรกทรอรี (Directory)

int (*select) (struct inode *, struct file *, int, select_table *);

ใช้โดยโปรแกรมเพื่อสอบถามว่า อุปกรณ์สามารถจะอ่านหรือเขียนได้หรือไม่ โดยถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL จะหมายถึง สามารถทำได้ทั้ง 2 อย่าง ซึ่งขึ้นอยู่กับชนิดของอุปกรณ์ด้วย โดยค่าที่ส่งกลับจะเป็นการบอกว่า สามารถอ่านหรือเขียนได้ตามที่กำหนดหรือไม่

int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);

ใช้สำหรับการออกคำสั่งเฉพาะของอุปกรณ์ เช่น การฟอร์แมต (Format) ดิสก์ โดยไม่มีการอ้างอิงไปยังตาราง fops โดยค่าที่ส่งกลับจะเป็นจำนวนเต็มบวก

int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);

ใช้แม็พ (Map) พื้นที่หน่วยความจำของอุปกรณ์และหน่วยความจำของโปรเซส

int (*open) (struct inode *, struct file *);

ใช้เมื่อต้องการจะเริ่มการทำงานกับอุปกรณ์ โดยถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL จะทำให้การเรียกใช้งานฟังก์ชันสำเร็จเสมอ แต่ไม่มีค่าส่งกลับ

int (*release) (struct inode *m struct file *);

ใช้งานเมื่อไฟล์ดีไวซ์ไดรเวอร์ (/dev/…) ของอุปกรณ์ถูกปิด

int (*fsync) (struct inode *, struct file *);

ใช้รีเซต (Reset) อุปกรณ์ให้อยู่ในสถานะพร้อมทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

int (*fasync) (struct inode *, struct file *, int);

ใช้แจ้งให้อุปกรณ์ทราบถึงสถานะแฟล็ก (Flag) FASYNC โดยถ้ามีการกำหนดฟังก์ชันนี้เป็น NULL หมายถึงอุปกรณ์ไม่สนับสนุนการทำงาน

int (*check_media_change) (kdev_t dev);

ใช้งานกับอุปกรณ์ที่เป็นบล็อกดีไวซ์ซึ่งฟังก์ชันนี้จะใช้งานโดยเคอร์เนลเพื่อตัดสินว่า มีการเปลี่ยนสื่อเก็บข้อมูลหลังจากการทำงานครั้งสุดท้ายหรือไม่

int (*revalidate) (kdev_t dev);

ใช้งานกับอุปกรณ์ที่เป็นบล็อกดีไวซ์ เพื่อจัดการกับหน่วยความจำแคช (Cache)

โค้ดด้านล่างเป็นการประกาศโครงสร้างข้อมูลระบุรูทีนสำหรับทำงานกับอุปกรณ์ schd (Special Character Device)

```
static struct file_operations schd_ops=      {
    schd_lseek,
    schd_read,
    schd_write,
    NULL,              // schd_readdir
    NULL,              // schd_select
    schd_ioctl,
    NULL,              // schd_mmap
    schd_open,
    schd_release,      // Nothing more, fill with NULLS
};
```

5.5.3 การเปิดและปิดไฟล์ดีไวซ์ไดรเวอร์

5.5.3.1 ขั้นตอนการเปิดไฟล์ดีไวซ์ไดรเวอร์ เพื่อเริ่มต้นทำงาน

ฟังก์ชัน open จะทำการเซตอัพ (Setup) ทรัพยากรต่าง ๆ เพื่อเตรียมติดต่อกับอุปกรณ์ โดยจะใช้มาโคร (Macro) MOD_INC_USE_COUNT เป็นดัชนีระบุว่ายังมีการใช้งานเพื่อป้องกันการลบโครงสร้างข้อมูลก่อนปิดไฟล์ โดยฟังก์ชัน open จะทำงานดังนี้

- ตรวจสอบอุปกรณ์ว่าอยู่ในสภาพพร้อมที่จะทำงานหรือไม่
- เซตอัพอุปกรณ์ถ้าเพิ่งจะเริ่มเปิดเป็นครั้งแรก
- หาค่าของเลขเมเจอร์และอัพเดต (Update) โครงสร้างข้อมูล f_op
- กำหนดโครงสร้างข้อมูลที่จำเป็นขึ้นมา
- เพิ่มค่าตัวแปรเก็บจำนวนครั้งที่ใช้งาน

สิ่งแรกที่ไดรเวอร์ทำคือ การแอ็กเซสเลขไมเนอร์ที่ฟิลด์ inode->i_rdev โดยอุปกรณ์แต่ละชนิดจะทำงานไม่เหมือนกันถึงแม้จะใช้ไดรเวอร์ตัวเดียวกัน ซึ่งที่จริงแล้วไดรเวอร์ไม่รู้จักอุปกรณ์ที่ต้องทำงานด้วย เพียงแต่ใช้เลขเมเจอร์และเลขไมเนอร์เป็นตัว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แบ่งอุปกรณ์ โดยผู้ใช้งานสามารถกำหนดการทำงานของแต่ละอุปกรณ์ให้ทำงานแตกต่างกันจากการกำหนดชื่อให้ต่างกัน

ตัวอย่างโค้ดของฟังก์ชัน open ต่อไปนี้ แสดงขั้นตอนในการเซตอัพข้อมูลของอุปกรณ์ที่จำเป็น โดยมีการใช้มาโคร TYPE และ NUM แอ็กเซสบิตข้อมูลจากเลขไมเนอร์

```
int schd_open (struct inode *inode, struct file *filp);
{
        int type = TYPE(inode -> i_rdev);
        int num = NUM(inode -> i_rdev);
        Schd_Dev *dev;          /* Device information */

        /* Manage peculiar types first */
        if (type) {
                if (type > SCHD_MAX_TYPE) return –ENODEV;
                filp -> f_op = schd_fop_array[type];

                /* Dispatch to specific open */
                return filp -> f_op -> open(inode, filp);
        }

        /* Type 0, Check the device number */
        if (num >= schd_nr_devs) return –ENODEV;
        dev = &schd_devices[num];

        /* Trim to 0 the length of the device if open was write-only */
        if ( (filp -> f_flags & O_ACCMODE) == O_WRONLY)
                schd_trim(dev);         /* Ignore errors */

        /* And use filp -> private_data to point to the device data */
        filp -> private_data = dev;

        MOD_INC_USE_COUNT;
        Return 0;                       /* Success */
}
```

ข้อมูลเกี่ยวกับอุปกรณ์จะเก็บไว้ที่โครงสร้างข้อมูล Schd_Dev จำนวนของอุปกรณ์ทั้งหมดที่มีชนิดเดียวกัน คือ schd_nr_devs และอาร์เรย์ (Array) ของพอยน์เตอร์ที่ชี้ไปยังพื้นที่หน่วยความจำ Schd_Dev คือ schd_devices[ ]

จากโค้ดตัวอย่างข้างต้นนี้ ไม่มีส่วนสำหรับทำงานบนอุปกรณ์จริง สิ่งที่โค้ดทำงานเป็นเพียงการรีเซตข้อมูลของอุปกรณ์ให้เหลือ 0 ไบต์ เมื่อเริ่มมีการใช้ฟังก์ชัน open เท่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.5.3.2 ขั้นตอนการปิดไฟล์ดีไวซ์ไดรเวอร์ เมื่อเลิกใช้งาน

- ลดจำนวนการใช้งานลง
- ยกเลิกการใช้งานโครงสร้างข้อมูลต่าง ๆ ที่กำหนดขึ้นมาเมื่อใช้ฟังก์ชัน open
- ปิดอุปกรณ์

ตัวอย่างโค้ดของการปิดการใช้งานอุปกรณ์

```
void schd_release (struct inode *inode, struct file *filp);
{
        MOD_DEC_USE_COUNT;
}
```

การลดจำนวนครั้งการใช้งานลงเป็นสิ่งสำคัญมาก เนื่องจากเคอร์เนลไม่สามารถจะลบหรือปิดการใช้งานไฟล์ดีไวซ์ไดรเวอร์ออกไปจากพื้นที่ของเคอร์เนลได้ ถ้าจำนวนการใช้งานไม่เท่ากับ 0 โดยโครงสร้างข้อมูล file จะถูกสร้างขึ้นเมื่อมีการใช้ฟังก์ชัน open เท่านั้น ถ้าจำนวนการใช้งานเป็น 0 ซึ่งจะเกิดขึ้นเมื่อมีการทำลายโครงสร้างข้อมูล file จึงจะสามารถใช้ฟังก์ชัน release เพื่อปิดการใช้งานอุปกรณ์ได้ (ปิดไฟล์ไดรเวอร์)

5.5.4 การอ่านและเขียนข้อมูลบนอุปกรณ์

การอ่านและเขียนข้อมูลบนไดรเวอร์เป็นการส่งผ่านข้อมูลระหว่างพื้นที่บนหน่วยความจำ 2 ส่วน คือ

1. พื้นที่ของเคอร์เนล และ
2. พื้นที่ของแอพพลิเคชัน

ฟังก์ชัน read และ write ในการอ่านข้อมูลบนอุปกรณ์จะใช้ฟังก์ชันต่อไปนี้สำหรับการอ่านและเขียนข้อมูลไปยังพื้นที่การทำงานของแอพพลิเคชัน

- ฟังก์ชันเขียนข้อมูลไปยังอุปกรณ์

void memcpy_fromfs(void *to, const void *from, unsigned long count);

- ฟังก์ชันอ่านข้อมูลจากอุปกรณ์

void memcpy_tofs (void *to, const void *from, unsigned long count);

ฟังก์ชัน read และ write จะส่งค่ากลับเป็นจำนวนเต็มลบถ้ามี ERROR เกิดขึ้น หรือเป็นจำนวนไบต์ที่อ่านได้หรือเขียนได้สำเร็จ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างโค้ดของฟังก์ชันอ่านข้อมูลของอุปกรณ์จากพื้นที่ของเคอร์เนล

ฟังก์ชัน read จะส่งค่ากลับดังนี้

- ถ้าส่งค่ากลับเท่ากับค่าที่ระบุ count แสดงว่าข้อมูลถูกอ่านหมดทั้งหมด
- ถ้าเป็นจำนวนเต็มบวกน้อยกว่าค่าที่ระบุ count แสดงว่าส่งผ่านข้อมูลได้เพียงบางส่วน ปกติแล้วแอพพลิเคชันจะพยายามอ่านให้ครบตามจำนวนที่กำหนด
- ถ้าค่าที่ส่งกลับเป็น 0 แสดงว่าถึงจุดสิ้นสุดไฟล์
- ถ้าส่งค่ากลับเป็นจำนวนเต็มลบ แสดงว่ามี ERROR เกิดขึ้น ชนิดของ ERROR จะระบุไว้ในไฟล์ <linux/errno.h>

```
read_write_t schd_read (struct inode *inode, struct file *filp, char *buf,
count_t count);
{
        Schd_Dev *dev = filp -> private_data; /* The first list item */

        int quantum = dev -> quantum;
        int qset = dev -> qset;
        int itemsize = quantum * qset; /* How many bytes in the list */

        unsigned long f_pos = (unsigned long) (filp -> f_pos);
        int item, s_pos, q_pos, rest;

        if (f_pos > dev -> size) return 0;
        if (f_pos + count > dev -> size) count = dev -> size – f_pos;

        /* Find list item, qset index, and offset in the quantum */
        item    = f_pos / itemsize;
        rest=     f_pos % itemsize;
        s_pos   = rest / quantum;
        q_pos   = rest % quantum

        /* Follow the list up to right position (defined elsewhere) */
        dev = schd_follow(dev, item);

        if (!dev -> data) return 0;        /* Don't fill holes */
        if (!dev -> data[s_pos])           return 0;
        if (count > quantum – q_pos) count = quantum – q_pos;
        dev -> usage++;            /* The following call may sleep */
        memcpy_tofs(buf, dev -> data[s_pos] + q_pos, count);
        dev -> usage--;

        filp -> f_pos += count;
        return count;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวอย่างโค้ดของฟังก์ชันเขียนข้อมูลของอุปกรณ์ไปยังพื้นที่ของเคอร์เนล

ฟังก์ชัน write จะส่งค่ากลับดังนี้

- ถ้าส่งค่ากลับเท่ากับค่าที่กำหนด count แสดงว่าการเขียนข้อมูลทำได้สำเร็จ
- ถ้าเป็นจำนวนเต็มบวกน้อยกว่าค่าที่ระบุ count แสดงว่าเขียนข้อมูลได้เพียงบางส่วน
- ถ้าค่าที่ส่งกลับเป็น 0 แสดงว่าไม่มีการเขียนข้อมูล อาจจะเนื่องจากอุปกรณ์ถูกบล็อกไว้ (ไม่ถือว่าเป็น ERROR)
- ถ้าส่งค่ากลับเป็นจำนวนเต็มลบ แสดงว่ามี ERROR เกิดขึ้น เช่นเดียวกับฟังก์ชัน read

```
read_write_t schd_write(struct inode *inode, struct file *filp, const
char *buf, count_t count);
{
        Schd_Dev *dev = filp -> private_data;
        Schd_Dev *dptr;
        int quantum = dev -> quantum;
        int qset = dev -> qset;
        int itemsize = quantum * qset;
        unsigned long f_pos = (unsigned long) (filp -> f_pos);
        int item, s_pos, q_pos, rest;

        /* Find list item, qset index and offset in the quantum */
        item    = f_pos / itemsize;
        rest=    f_pos % itemsize;
        s_pos  = rest / quantum;
        q_pos  = rest % quantum;

        /* Follow the list up to the right position */
        dptr = schd_follow(dev, item);
        if (!dptr -> data) {
                dptr -> data = kmalloc(qset * sizeof(char *),
                GFP_KERNEL);
                if (!dptr -> data) return –ENOMEM;
                memset(dptr ->data, 0, qset * sizeof(char *) );
        }
        if (!dptr -> data[s_pos]) {
                dptr -> data[s_pos] = kmalloc(quantum,
                GFP_KERNEL);
                if (!dptr -> data[s_pos]) return –ENOMEM;
        }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        /* Write to the end of quantum */
        if (count > quantum – q_pos) count = quantum – q_pos;
        dev -> usage++;          /* The following call may sleep */
        memcpy_fromfs(dptr -> data[s_pos] + q_pos, buf, count);
        dev -> usage--;

        /* Update the size */
        if (dev -> size < f_pos + count) (dev -> size = f_pos + count);
        filp -> f_pos += count;
        return count;
}
```

5.6 บล็อกดีไวซ์ไดรเวอร์

5.6.1 การเพิ่มหรือลบไดรเวอร์ออกไปจากเคอร์เนล

จะทำเช่นเดียวกับคาแร็กเตอร์ดีไวซ์ไดรเวอร์ โดยเคอร์เนลจะใช้เลขเมเจอร์ในการจำแนกโมดูลไดรเวอร์และใช้ฟังก์ชันต่อไปนี้ในการเพิ่มหรือลบไดรเวอร์ออกจากเคอร์เนล

int register_blkdev(unsigned int major, const char *name, struct file_operations *fops);

int unregister_blkdev(unsigned int major, const char *name);

5.6.2 ฟังก์ชันการอ่านและเขียนดีไวซ์ไดรเวอร์

ฟังก์ชันที่ใช้สำหรับการอ่านและเขียนดีไวซ์ไดรเวอร์ จะเป็นฟังก์ชันพื้นฐานที่ใช้งานโดยทั่วไปเรียกว่า "Strategic Routine" ซึ่งจะได้โครงสร้างข้อมูล file_operations ของอุปกรณ์ที่ชื่อว่า sbkd (Special block device) เป็นดังนี้

```
struct file_operations sbkd_fops = {
        NULL,           /* lseek: default */
        block_read,             /* Strategy routine */
        block_write,            /* Strategy routine */
        NULL,           /* sbkd_readdir */
        NULL,           /* sbkd_select */
        sbkd_ioctl,
        NULL,
        sbkd_open,
        sbkd_release,
        block_fsync,            /* Strategy routine */
        NULL,           /* sbkd_fasync */
        sbkd_check_change,
        sbkd_revalidate
};
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

และบล็อกดีไวซ์ยังมีโครงสร้างข้อมูลที่ทำหน้าที่เก็บลักษณะทางกายภาพไว้ด้วย คือ โครงสร้างข้อมูล blk_dev_struct ซึ่งเก็บไว้ในไฟล์ <linux/blkdev.h> โดยโครงสร้างเป็นดังนี้

```
struct blk_dev_struct {
        void    (*request_fn) (void);
        struct  request current_request;
        struct  request plug;
        struct  tq_struct plug_tq;
};
extern struct blk_dev_struct blk_dev[MAX_BLKDEV];
```

จากในโค้ดนั้น ยังมีอาร์เรย์ข้อมูลเก็บลักษณะของอุปกรณ์ นอกจาก blk_dev เก็บไว้ในไฟล์ drivers/block/ll_rw_block.c

ข้อแตกต่างที่สำคัญระหว่างคาแร็กเตอร์ดีไวซ์และบล็อกดีไวซ์ คือ คาแร็กเตอร์ดีไวซ์จะมีฟังก์ชันสำหรับอ่านหรือเขียนข้อมูลโดยตรงกับอุปกรณ์ ในขณะที่บล็อกดีไวซ์จะทำงานผ่านหน่วยความจำแคช (Cache) เมื่อโปรแกรมต่าง ๆ ต้องการข้อมูล ตัวโปรแกรมจะต้องทำการร้องขอไปยังหน่วยความจำแคช ถ้าไม่มีข้อมูลนั้นในแคช บล็อกดีไวซ์จะทำการเพิ่มโครงสร้างข้อมูล request ไปยังลิสต์ (List) ของโครงสร้างข้อมูล blk_dev_struct เพื่อทำการค้นหาข้อมูลที่โปรแกรมต้องการจากสื่อเก็บข้อมูล ดังนั้น ในโมดูล init จะต้องทำการกำหนดค่าของฟิลด์ request_fn ในโครงสร้างข้อมูล blk_dev_struct ด้วย ส่วนฟิลด์อื่น ๆ จะใช้งานภายในเคอร์เนล ซึ่งผู้เขียนไดรเวอร์ไม่จำเป็นต้องใช้งาน

5.6.3 การยกเลิกการโหลดข้อมูลของไดรเวอร์

การยกเลิกการโหลดข้อมูลจะทำเมื่อค่าของ counter เป็น 0 ซึ่งพื้นที่ของหน่วยความจำที่เก็บข้อมูลของอุปกรณ์ตัวนั้นจะถูกนำไปใช้ในงานอื่น โดยมีฟังก์ชันที่สนับสนุนการนำพื้นที่ไปใช้ คือ

1. ฟังก์ชัน check_media_change ทำการตรวจสอบว่า มีการเปลี่ยนอุปกรณ์หรือไม่ และ
2. ฟังก์ชัน revalidate ทำการอัพเดตแฟล็กสถานะหลังจากเปลี่ยนอุปกรณ์แล้ว

ตัวอย่างโค้ดต่อไปนี้ เป็นของฟังก์ชัน check_media_change โดยถ้ามีการยกเลิกการโหลดข้อมูลของดีไวซ์ไดรเวอร์ จะได้ค่าที่ส่งกลับเป็น 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
int sbkd_check_change(kdev_t i_rdev)
{
        int minor = MINOR(i_rdev);
```

```
        Sbkd_Dev *dev = sbkd_devices + minor;
        if (minor >= sbkd_devs)        /* Paranoid */
                return 0;

        PDEBUG("check_change for dev %I\n", minor);

        if (dev -> data)
                return 0;              /* Still valid */
        return 1;                      /* Expired */
}
```

และเมื่อมีการเปลี่ยนแปลงอุปกรณ์ ก็จำเป็นที่จะต้องอัพเดตแฟล็กสถานะของอุปกรณ์ด้วย ซึ่งฟังก์ชันที่ใช้ในการอัพเดตคือ ฟังก์ชัน revalidate โดยฟังก์ชันนี้จะทำการจองพื้นที่หน่วยความจำเพื่อเก็บแฟล็กสถานะของอุปกรณ์ หากทำการจองพื้นที่ได้สำเร็จ จะได้ค่าที่ส่งกลับเป็น 0 ซึ่งตัวอย่างโค้ดต่อไปนี้ เป็นของฟังก์ชัน revalidate

```
int sbkd_revalidate(kdev_t i_rdev)
{
        Sbkd_Dev *dev = sbkd_devices + MINOR(i_rdev);

        PDEBUG("revalidate for dev &I\n", MINOR(i_rdev);

        if (dev -> data)
                return 0;
        dev -> data = vmalloc(dev ->size);
        if (!dev -> data)
                return –ENOMEM;
        return 0;
}
```

5.7 เน็ตเวิร์กดีไวซ์ไดรเวอร์

เน็ตเวิร์กดีไวซ์ไดรเวอร์มีการทำงานคล้ายกับบล็อกดีไวซ์ไดรเวอร์ คือ ทำการลงทะเบียนอุปกรณ์ไว้กับเคอร์เนลพร้อมทั้งกำหนดข้อมูลต่าง ๆ ให้แก่อุปกรณ์นั้น และจะทำงานเมื่อมีคำสั่งให้อ่าน / เขียนข้อมูลโดยทำผ่านฟังก์ชัน request_fn ซึ่งจะส่งผ่านข้อมูลระหว่างอุปกรณ์ส่งข้อมูลของเน็ตเวิร์กและซ็อกเก็ตบัฟเฟอร์เพื่อทำการประมวลผลมาตรฐานของโพรโตคอลชั้นต่าง ๆ อธิบายโดยให้ไดรเวอร์ใช้โพรโตคอลด้านฮาร์ดแวร์แบบอีเทอร์เน็ต (Ethernet) และไอพี (IP) ควบคุมการส่งแพ็กเกตข้อมูล

โดยกำหนดตัวอย่างการทำงานของไดรเวอร์ขึ้น ให้โมดูลของไดรเวอร์ spnd (Special Network Driver) รันเป็นอิสระจากอุปกรณ์ฮาร์ดแวร์ และจำลองเส้นทางการส่งผ่านข้อมูลระหว่างแอดเดรสต้นทางและปลายทางเสมือนว่าอยู่คนละเครื่อง โดยการแก้ไขแอดเดรสของเครื่องต้นทางและปลายทาง ซึ่งแพ็กเกตที่ส่งออกจากอุปกรณ์ที่ชื่อ nd0 จะไปปรากฏที่อุปกรณ์ชื่อ nd1 เสมือนส่งผ่านเน็ตเวิร์ก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารนี้ทุกครั้งที่มีการนำไปใช้

5.7.1 การส่งแพ็กเกตข้อมูล

เมื่อเคอร์เนลต้องการส่งแพ็กเกตข้อมูล จะทำการเก็บแพ็กเกตข้อมูลลงในซ็อกเก็ต (struct sk_buffer) โดยข้อมูลที่อยู่ในซ็อกเก็ตบัฟเฟอร์จะเป็นเฟรม (Frame) ข้อมูลตามมาตรฐานของการส่งข้อมูลในระดับต่ำ (Physical Layer) ดังนั้น ไฟล์ดีไวซ์จะมองซ็อกเก็ตเป็นเพียงแพ็ตเกตข้อมูล และลีนุกซ์จะใช้ลิสต์ของโครงสร้างข้อมูล sk_buffer เดียวกันสำหรับการรับและส่งข้อมูลภายใน Network Subsystem

ตัวอย่างโค้ดต่อไปนี้ เป็นฟังก์ชันที่ใช้สำหรับควบคุมฮาร์ดแวร์เพื่อส่งข้อมูลจริง ๆ โดยจะแยกออกมาต่างหากเนื่องจากฟังก์ชันขึ้นกับอุปกรณ์ที่ใช้งานด้วย (Device specific) ซึ่งจะต้องตรวจสอบจุดที่จำเป็นก่อนแล้วค่อยเริ่มส่งข้อมูล

```
int spnd_tx(struct sk_buffer *skb, struct device *dev)
{
        int len, retval = 0;
        char *data;

        if (dev -> tbusy) {
                retval = -EBUSY;
        goto tx_done;
        }

        if (skb = = NULL) {
                PDEBUG("tint for &p\n", dev);
                dev_tint(dev);
                return 0;          /* We are ready to transmit */
        }
        dev -> tbusy = 1;          /* Transmission is busy */

        /* Minimum len */
        len = ETH_ZLEN < skb -> len ? skb -> len : ETH_ZLEN;
        data = skb -> data;
        dev -> trans_start = jiffies;       /* Save the timestamp */

        /* actual deliver of data id device-specific, and not shown here */
        spnd_hw_tx(data, len, dev);

        /* Release it */
        tx_done: dev_kfree_skb(skb, FREE_WRITE);
        return retval;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ถ้าการส่งข้อมูลทำได้สำเร็จ จะต้องทำการเคลียร์ค่าฟิลด์ dev -> tbusy เพื่อบอกว่าว่างแล้ว

5.7.2 การรับแพ็กเกตข้อมูล

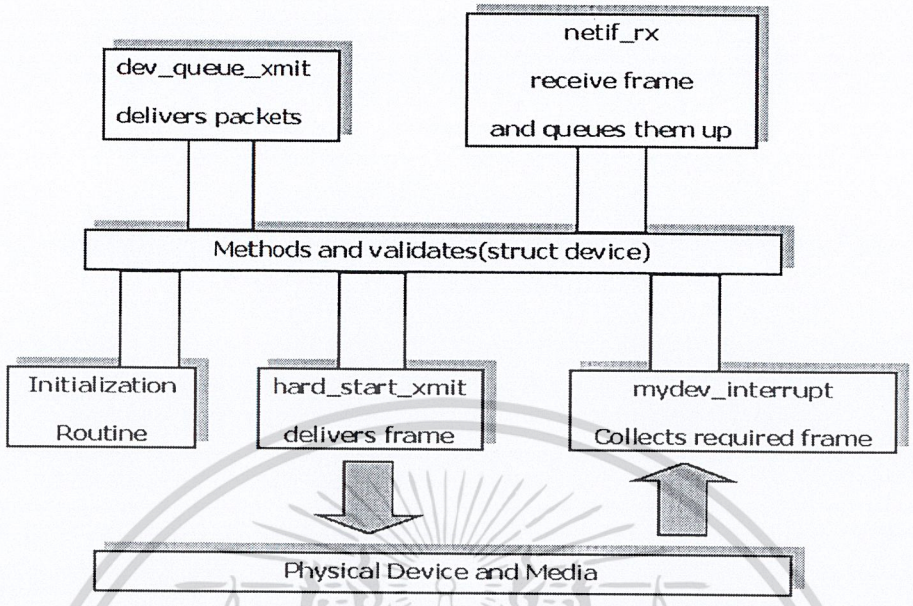
ต้องมีการจองพื้นที่ของหน่วยความจำ (sk_buffer) เพื่อทำการเก็บข้อมูลที่รับมาจากอุปกรณ์เน็ตเวิร์ก ซึ่งวิธีที่มีประสิทธิภาพมากที่สุดในการทำงาน คือ การอินเทอร์รัพต์ (Interrupt)

ตัวอย่างฟังก์ชัน spnd_rx ต่อไปนี้ เป็นการรับอาร์กิวเมนต์เพื่อเป็นพอยน์เตอร์ชี้ไปยังข้อมูลของอุปกรณ์และความยาวของข้อมูล แล้วจะทำการส่งข้อมูลไปยังโพรโตคอลอีกที โดยจะทำฟังก์ชันนี้หลังจากรับข้อมูลจากอุปกรณ์เน็ตเวิร์กมาเก็บในบัฟเฟอร์แล้ว

```
void spnd_rx(struct device *dev, int len, unsigned char *buf)
{
        struct sk_buffer *skb;
        struct spnd_priv = (struct spnd_priv *) dev -> priv;

        /* The packet has been retrieved from the transmission medium, Build
        an skb so the upper layers can handle it */

        skb = dev_alloc_skb(len + 2);
        if (!skb) {
                printk("spnd rx: low on mem\n");
                return;
        }
        memcpy(skb_put(skb, len), buf, len);

        /* Write metadata nad then pass to the receive level */
        skb -> dev = dev;
        skb -> protocol = eth_type_trans(skb, dev);
        skb -> ip_summed = CHECKSUM_UNNECESSARY;
        privp -> starts.rx_packets++;
        netif_rx(skb);
        return;
}
```

เมื่อฟังก์ชันทำการจองพื้นที่บนหน่วยความจำสำหรับโครงสร้างข้อมูล skb เรียบร้อยแล้ว ฟังก์ชันจะทำการคัดลอกข้อมูลจากบัฟเฟอร์ไปยัง skb โดยใช้ฟังก์ชัน memcpy สำหรับฟังก์ชัน skb_put จะทำการอัพเดตพอยน์เตอร์ที่ชี้จุดสิ้นสุดของข้อมูลภายใน skb และความยาวข้อมูล (len) หลังจากที่คัดลอกข้อมูลเสร็จแล้ว และขั้นตอนสุดท้าย ไดรเวอร์จะการอัพเดตจำนวนแพ็กเกตที่รับได้ในโครงสร้างข้อมูล stats ซึ่งได้เก็บสถิติเกี่ยวกับเน็ตเวิร์กด้านต่าง ๆ ไว้ จากนั้นจะส่งบัฟเฟอร์ skb ไปยังโพรโตคอลชั้นถัดไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น และห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



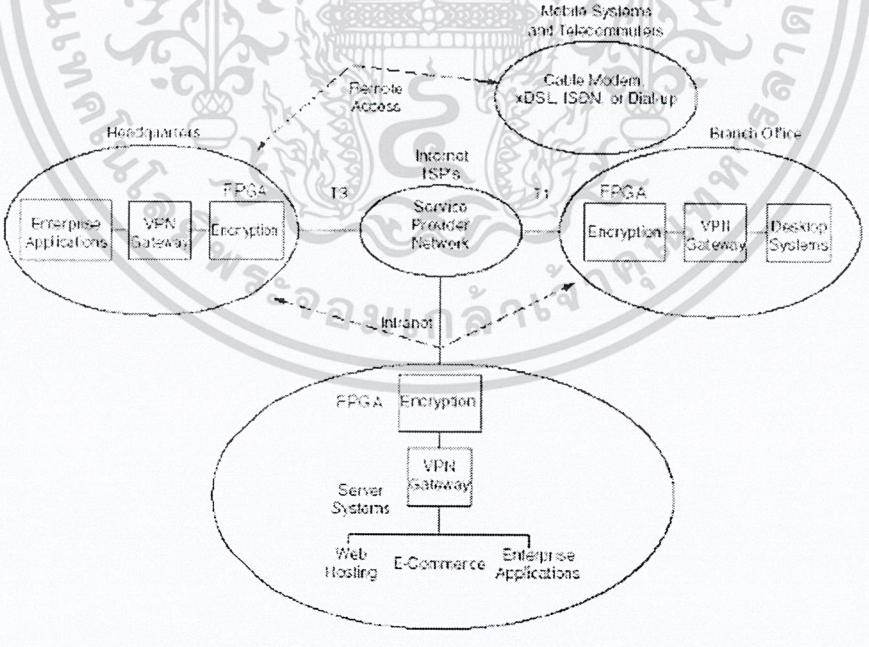
รูปที่ 5-3: โครงสร้างในการรับและส่งแพ็กเกตข้อมูลของลีนุกซ์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6
วีพีเอ็น

การสื่อสารในปัจจุบัน มีการใช้ประโยชน์ของเทคโนโลยีเพื่ออำนวยความสะดวกในด้านต่างๆ โดยเฉพาะอย่างยิ่งในด้านการรับ-ส่งข้อมูลผ่านทางอินเทอร์เน็ต (Internet) ซึ่งช่วยให้ข้อมูลที่ถูกส่งนั้นถึงมือผู้รับอย่างสะดวกและรวดเร็ว แต่ทราบได้อย่างไรว่า ข้อมูลที่ส่งออกไปนั้นถึงมือผู้รับอย่างปลอดภัยโดยไม่ถูกดักระหว่างทาง หรือ ข้อมูลที่รับมานั้นเป็นข้อมูลที่ถูกต้องและได้มาจากบุคคลที่ส่งให้เราจริงๆ

และเนื่องจากอินเทอร์เน็ตซึ่งเป็นเครือข่ายสื่อสารแบบ แวน (WAN) ที่มีค่าใช้จ่ายต่ำกว่าและมีพื้นที่ครอบคลุมทั่วโลกมากกว่าการใช้ลิสไลน์ (Leased Line) เฟรมรีเลย์ (Frame Relay) หรือ เอทีเอ็ม (ATM) แต่ไม่สามารถจะรับประกันในด้านความปลอดภัย ด้านแบนด์วิดธ์ (Bandwidth) หรือ Quality of Service (QoS) โดยเฉพาะอย่างยิ่งในระบบไพรเวตเน็ตเวิร์ก (Private Network) อีกทั้งระบบอินเทอร์เน็ตยังสนับสนุนเฉพาะโพรโตคอลทีซีพี/ไอพี (TCP/IP Protocol) เพียงอย่างเดียว ในขณะที่เครือข่ายส่วนใหญ่มักประกอบด้วยโพรโตคอลต่างๆมากมาย ผู้ให้บริการอินเทอร์เน็ต (ISO) ที่สนับสนุนหลากหลายโพรโตคอลของไพรเวตเน็ตเวิร์กในราคาที่ไม่แพงและมีบริการครอบคลุมเช่นเดียวกับอินเทอร์เน็ตให้บริการในชื่อของเวอร์ชวลไพรเวตเน็ตเวิร์ก (Virtual Private Network) หรือเอ็กซ์ทราเน็ต (Extranet)



รูปที่ 6-1: การนำเครือข่ายวีพีเอ็น (VPN) มาใช้ในระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.1 ความหมายของวีพีเอ็น (VPN: Virtual Private Network)

วีพีเอ็นเป็นการสร้างการเชื่อมต่อแบบเสมือนขึ้นระหว่างเครือข่ายตั้งแต่ 2 เครือข่ายขึ้นไป ให้เหมือนกับติดต่อกันโดยใช้ช่องทางส่วนตัวส่งข้อมูลถึงกัน โดยใช้การเข้ารหัสข้อมูล (Encryption) เพื่อให้ข้อมูลที่นำมาใช้ในการติดต่อถึงกันมีความปลอดภัยมากขึ้น ซึ่งจะนำเอาโครงข่ายสาธารณะ เช่น อินเทอร์เน็ตมาเป็นตัวกลางในการติดต่อสื่อสารระหว่างกัน

วีพีเอ็นถือเป็นทางออกที่ช่วยให้องค์กรของท่านสามารถเชื่อมต่อสำนักงานในแต่ละที่ด้วยค่าใช้จ่ายที่ประหยัด แต่มีความปลอดภัยสูง นอกจากนี้ ยังช่วยให้ผู้ใช้งานภายนอกสำนักงานสามารถติดต่อเข้ามาที่สำนักงานเพื่อรับส่งข้อมูลด้วยความปลอดภัยและสะดวก เสมือนกับนั่งทำงานอยู่ใน LAN ของสำนักงานนั้น ๆ

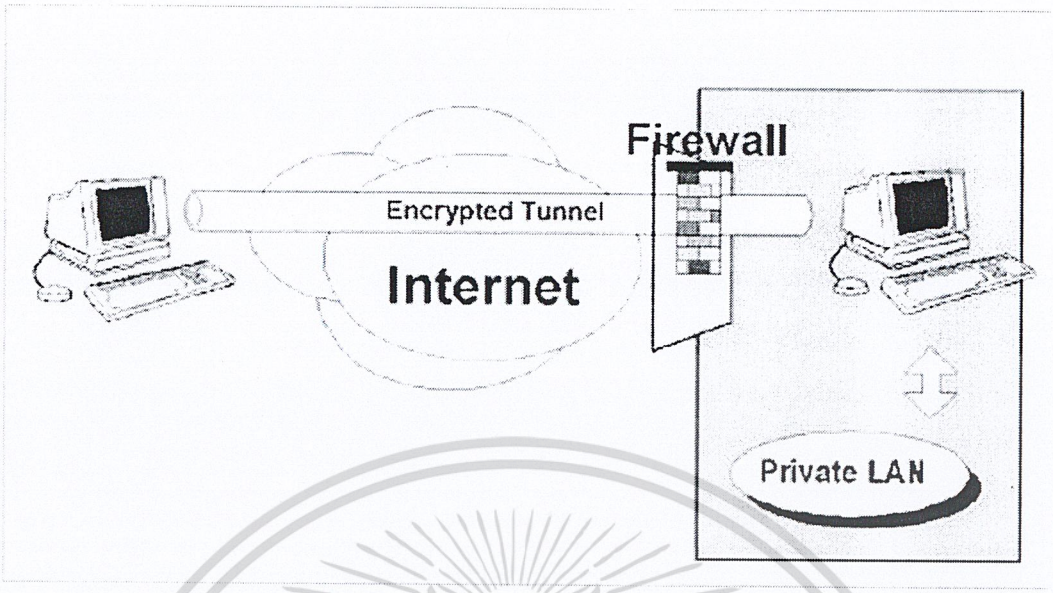
6.2 คุณสมบัติของวีพีเอ็น

- ไว้ใจได้ : ระบบวีพีเอ็นจะต้องให้ความรู้สึกมั่นใจว่า ข้อมูลที่เป็นความลับและเป็นส่วนตัวของแต่ละผู้ใช้นั้น จะไม่ถูกดักหรือเปิดใช้งานโดยผู้อื่นที่ไม่ได้รับอนุญาต
- รัดกุม : ระบบวีพีเอ็นจะต้องมีระบบการตรวจสอบผู้ใช้งานอย่างละเอียดรัดกุม ผู้ที่จะมีสิทธิ์ใช้งานในระบบต้องเป็นสมาชิกที่ผ่านการตรวจสอบแล้วเท่านั้น นอกจากนี้ วีพีเอ็นจะต้องสร้างความมั่นใจให้กับระบบข้อมูลว่า ข้อมูลและแหล่งข้อมูลที่ส่งผ่านเข้ามาปะปนกับข้อมูลภายในนั้นถูกต้องมาจากผู้ที่เชื่อถือได้
- มั่นคง : ข้อมูลที่ผู้ส่ง ส่งออกมานั้น จะต้องเหมือนกับข้อมูลที่ผู้รับได้รับ หมายความว่า ข้อมูลที่ส่งออกระบบเน็ตเวิร์ค จะต้องไม่มีการสูญหาย บิดเบือน หรือ ติดค้าง ข้อมูลจะต้องมีระบบการตรวจสอบเพื่อยืนยันการรับ-ส่งที่ถูกต้อง
- รองรับได้หลายโพรโตคอล : ในระบบเน็ตเวิร์คแต่ละที่นั้น มีการใช้โพรโตคอลแตกต่างกันออกไป แต่การที่จะให้สามารถรับ-ส่งข้อมูลข้ามเน็ตเวิร์คได้นั้น ระบบจะต้องรองรับโพรโตคอลที่แตกต่างกันได้
- ไม่สิ้นเปลืองแบนด์วิดธ์ มากกว่าการรับ-ส่งข้อมูลแบบธรรมดามากนัก
- ปรับเข้าสู่ระบบที่มีอยู่แล้วได้ง่าย
- ง่ายต่อการขยายระบบวีพีเอ็นในอนาคต

6.3 การทำงานของวีพีเอ็น

เมื่อเครื่องคอมพิวเตอร์ที่อยู่ต่างเน็ตเวิร์คกัน แต่ต้องการส่งข้อมูลถึงกัน เครื่องคอมพิวเตอร์ 2 เครื่องนี้จะมีการสร้างทันแนล (Tunneling) เพื่อเชื่อมระหว่างกัน โดยทันแนล (Tunnel) นี้จะเปรียบเสมือนท่อต่อเชื่อมระหว่างปลายทาง 2 ข้าง ซึ่งท่อนี้จะปิดกั้นไม่ให้บุคคลที่อยู่ภายนอกท่อสามารถดักหรือเก็บข้อมูลไปได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 6-2: การทำงานของวีพีเอ็น

6.4 การสร้างทันแนลในวีพีเอ็น

การสร้างทันแนล คือ การ Encapsulate ข้อมูลแล้วจัดรูปแบบให้เป็นแพ็กเกจของข้อมูลปกติ เพื่อส่งออกไปภายนอกเครื่อง ซึ่งรูปแบบของแพ็กเกจนั้น ขึ้นอยู่กับโพรโตคอลที่จะใช้ในการสื่อสาร ดังนั้น เมื่อนำเอาการสร้างทันแนลมาใช้ในการทำวีพีเอ็น เราจึงทำการเพิ่มความปลอดภัยโดยการเข้ารหัสข้อมูล โดยเอาข้อมูลที่จะส่งในรูปไอพีแพ็กเกจ (IP Packet) ปกติมาเข้ารหัส (เนื่องจากโพรโตคอลไอพีนี้ เป็นโพรโตคอลที่ใช้ในระบบอินเทอร์เน็ต) จากนั้น ก็จะ encapsulate ให้เป็นรูปแบบโพรโตคอลสำหรับใช้ในระบบวีพีเอ็น ซึ่งวีพีเอ็นโพรโตคอลที่นิยมใช้ทั่วไป ได้แก่

- Point to Point Tunneling Protocol (PPTP)
- Layer2 Tunneling Protocol (L2TP)
- Layer2 Forwarding (L2F)
- IP Security (IPSec)
- SOCKETS Protocol (SOCKS v5)

PPTP จะสร้างทันแนลโดยห่อ PPP แพ็กเกจไว้ในไอพี ซึ่งเป็นโพรโตคอลที่ทำงานในเลเยอร์ 3 ในขณะที่ L2F ทำงานในเลเยอร์ 2 โดยใช้พวกเฟรมรีเลย์หรือเอทีเอ็มใช้ในการทำทันแนล

PPTP สามารถใช้เป็นแบบ Client-initiated (ซึ่งทรานส์พาเรนต์สำหรับไอเอสพี) หรือใช้เป็นแบบ Client-transparent ก็ได้ ทั้งสองแบบนี้ในปัจจุบันมีใช้เฉพาะในวินโดวส์เอ็นที (NT) เท่านั้น ซึ่งต้องการใช้ทั้งเอ็นพีไคลเอนต์และเอ็นพีเซิร์ฟเวอร์ ในทางตรงข้าม L2F ต้องการการสนับสนุนในแอ็กเซสเซิร์ฟเวอร์และในเราเตอร์ ดังนั้น ในการใช้งานไอเอสพี ต้องสนับสนุน L2F ด้วยระบบการป้องกันของ L2F มีการจัดเตรียมบางอย่างที่ PPTP ไม่มี อย่างเช่น การพิสูจน์ตนหรือ ออเทนทิเคชัน (Authentication) ของปลายทั้งสองข้างของทันแนล (แอ็กเซสเซิร์ฟเวอร์และทันแนลเซิร์ฟเวอร์) เป็นต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ข้อได้เปรียบหลักของ PPTP คือ สนับสนุนทั้งไคลเอนต์และทันแนลเซิร์ฟเวอร์ สำหรับ PPTP มีมาพร้อมกับวินโดวส์เอ็นพี 4.0 และในส่วนของวินโดวส์ 95 กำลังอยู่ในช่วงการพัฒนาข้อได้เปรียบอื่นๆ คือ PPTP สนับสนุนการทำโฟลว์คอนโทรล (Flow Control) ทำให้ทั้งไคลเอนต์และเซิร์ฟเวอร์ปลอดภัยจากเหตุการณ์ข้อมูลโอเวอร์โฟลว์ (Over Flow) ช่วยเพิ่มประสิทธิภาพโดยลดการดรอปแพ็กเกจลง รวมไปถึงการส่งซ้ำด้วย อย่างไรก็ตาม PPTP ก็ยังคงต้องการไอพี (แม้ว่าจะสามารถสร้างทันแนลให้แก่ IPX และ NetBEUI ได้เช่นเดียวกับ PPP) และปัจจุบันใน PPTP ยังไม่มีความสามารถออเทนทิเคต (Authenticate) ของปลายทั้งสองด้านของทันแนล จึงจำเป็นต้องใช้ความสามารถของ PPP ในการตรวจสอบ

การรวมข้อดีของทั้งสองโพรโตคอลเข้าด้วยกันในชื่อ "Layer Two Tunneling Protocol (L2TP)" ซึ่งในด้านการจัดการความปลอดภัยใช้ Secure IP หรือ IPSEC เป็นพื้นฐานในการเข้ารหัสในการเชื่อมต่อระหว่างปลายทั้งสองของ L2TP (มาตรฐานการเข้ารหัสยังไม่มีใน PPTP หรือใน L2F) นอกจากนั้น L2TP ยังสนับสนุนการทำทันแนลหลายๆอันพร้อมกันบนไคลเอนต์เพียงตัวเดียว ซึ่งคุณสมบัตินี้ยิ่งทวีความสำคัญมากขึ้นในอนาคต เมื่อทันแนลสามารถสนับสนุนการจองแบนด์วิดธ์และ QoS

อีกทางเลือกหนึ่งของ PPTP และ L2F คือ SOCKS ซึ่งเป็นโพรโตคอลใช้ในการตรวจสอบข้อมูลที่ผ่านไฟร์วอลล์ที่อนุญาตให้สายของข้อมูลไหลผ่านไฟร์วอลล์โดยขึ้นอยู่กับการตรวจสอบผู้ใช้งานแทนการตรวจสอบไอพีแพ็กเกจ

ข้อได้เปรียบของ SOCKS เมื่อเทียบกับวิธีอื่นคือ SOCKS สนับสนุนทั้งยูนิกส์และเอ็นพี นอกจากนั้น ยังมีปลั๊กอิน (Plug in) ที่สนับสนุนการเข้ารหัสและการจัดการหลายรูปแบบ อีกทั้งมีสถาปัตยกรรมความปลอดภัยแบบทางเดียว ซึ่งอนุญาตให้ทำทันแนลไปยังเครือข่ายอื่นได้ในขณะที่ลดการถูกโจมตีกลับจากเครือข่ายนั้น การทำงานของ SOCKS นั้นทำงานในระดับทีซีพี ทำให้ง่ายต่อการสร้างแอปพลิเคชั่นเพื่อเชื่อมต่อแบบทันแนลผ่านทางพอร์ตของทีซีพีที่กำหนด โดยไฟร์วอลล์ใดๆก็ตาม สามารถกำหนดให้ส่งผ่านทราฟฟิก (Traffic) ของ SOCKS แบบทรานส์พาเรนต์ได้

6.5 รูปแบบบริการของวีพีเอ็น

แบ่งออกเป็น 3 รูปแบบ

1. **แอ็กเซสวีพีเอ็น (Access VPN)**: เป็นรูปแบบในการเข้าถึงเครือข่ายวีพีเอ็น จากอุปกรณ์เคลื่อนที่ต่างๆ ซึ่งสามารถเข้าถึงเครือข่ายได้ใน 2 ลักษณะ
   - เป็นการเข้าถึงจากไคลเอนต์ใดๆก็ได้ โดยอาศัยผู้ให้บริการอินเทอร์เน็ตเป็นตัวกลางในการติดต่อ ซึ่งจะมีการเข้ารหัสในการส่งสัญญาณจากเครื่องไคลเอนต์ไปยังไอเอสพี
   - เป็นการเข้าถึงจากเครื่องแอ็กเซสเซิร์ฟเวอร์ (Network Access Server - NAS) โดยเริ่มต้นจากผู้ใช้หมุนโมเด็มติดต่อมายังไอเอสพี และจากนั้นจะมีการเข้ารหัสข้อมูลและส่งต่อไปยังปลายทาง
2. **อินทราเน็ตวีพีเอ็น (Intranet VPN)**: เป็นรูปแบบในการเข้าถึงเครือข่ายวีพีเอ็น ที่เฉพาะภายในองค์กรเท่านั้น อาทิ การเชื่อมต่อเครือข่ายระหว่างสำนักงานใหญ่ในกรุงเทพฯและสาขาย่อยในต่างจังหวัด เสมือนกับการทดแทนการเข้าวงจรลิสไลน์ระหว่างกรุงเทพฯกับต่างจังหวัด โดยที่แต่ละสาขาสามารถ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ต่อเชื่อมเข้ากับผู้ให้บริการอินเทอร์เน็ตในท้องถิ่นของตน เพื่อเชื่อมเข้าโครงข่ายวีพีเอ็น ขององค์กรอีกทีหนึ่ง

3. **เอ็กซ์ทราเน็ตวีพีเอ็น (Extranet VPN)**: เป็นรูปแบบในการเข้าถึงเครือข่ายที่คล้ายกับ Intranetวีพีเอ็น แต่มีการขยายวงกว้างออกไปยังกลุ่มลูกค้า ผู้สนับสนุน (Supplier) และหุ้นส่วน (Partner) เพื่อให้ใช้เครือข่ายได้ จุดสำคัญอย่างหนึ่งในการเลือกติดตั้งวีพีเอ็น คือ การเลือกผู้ให้บริการอินเทอร์เน็ตที่วางระบบรักษาความปลอดภัยเป็นอย่างดี มีส่วนอย่างมากในการส่งข้อมูลบนวีพีเอ็น ให้ปลอดภัยมากยิ่งขึ้น เพราะถ้าไอเอสพีมีระบบรักษาความปลอดภัยที่รัดกุม ย่อมจะช่วยให้ข้อมูลที่ส่งมามีความปลอดภัยมากขึ้น

6.6 อุปกรณ์ทียูเอ็นและทีเอพี (TUN/TAP device)

อุปกรณ์ทียูเอ็นและทีเอพีเป็นอุปกรณ์ชนิดหนึ่งที่นำมาใช้ในการทำวีพีเอ็น โดยมีหน้าที่ในการจัดการกับการรับและการส่งแพ็กเกตจากโปรแกรมของผู้ใช้ ซึ่งมันจะถูกมองเป็นอุปกรณ์ที่ทำการเชื่อมต่อแบบ Point-to-Point หรือแบบอีเทอร์เน็ตแบบง่าย ๆ แต่ตัวรับและตัวส่งแพ็กเกตได้เปลี่ยนแปลงไป เป็นการรับและส่งแพ็กเกตมาจากตัวโปรแกรมแทนการรับและส่งมาจากอุปกรณ์จริง

6.6.1 ความหมายของทียูเอ็น

ทียูเอ็นคืออุปกรณ์เน็ตเวิร์กเสมือนที่มีการเชื่อมต่อแบบ Point-to-Point (Virtual Point-to-Point network device) โดยไดรเวอร์ของทียูเอ็นถูกออกแบบให้สนับสนุนเคอร์เนลระดับต่ำเพื่อการทำ IP tunneling ซึ่งมันจะจัดการกับโปรแกรมของผู้ใช้อยู่ 2 ส่วนคือ

- /dev/tunX เป็นคาแร็กเตอร์ดีไวซ์
- tunX เป็นจุดเชื่อมต่อเสมือนแบบ Point-to-Point (Virtual Point-to-Point interface)

โปรแกรมของผู้ใช้จะทำการเขียนเฟรมไอพีลงไปใน /dev/tunX และเคอร์เนลจะทำการรับเฟรมนี้มาจากจุดเชื่อมต่อของ tunX อีกที และในขณะเดียวกัน เมื่อเคอร์เนลทำการเขียนเฟรมส่งไปยังจุดเชื่อมต่อ tunX และตัวโปรแกรมก็จะทำการอ่านเฟรมจาก /dev/tunX เช่นเดียวกัน

6.6.2 ความหมายของทีเอพี

ทีเอพีคืออุปกรณ์เน็ตเวิร์กเสมือนที่มีการเชื่อมต่อแบบอีเทอร์เน็ต (Virtual Ethernet network device) โดยไดรเวอร์ของทียเอพีถูกออกแบบให้สนับสนุนเคอร์เนลระดับต่ำเพื่อการทำ Ethernet tunneling ซึ่งมันจะจัดการกับโปรแกรมของผู้ใช้อยู่ 2 ส่วนคือ

- /dev/tapX เป็นคาแร็กเตอร์ดีไวซ์
- tapX เป็นจุดเชื่อมต่อเสมือนแบบอีเทอร์เน็ต (Virtual Ethernet interface)

โปรแกรมของผู้ใช้จะทำการเขียนเฟรมอีเทอร์เน็ต ลงไปใน /dev/tapX และเคอร์เนลจะทำการรับเฟรมนี้มาจากจุดเชื่อมต่อของ tapX อีกที และในขณะเดียวกัน เมื่อเคอร์เนลทำการเขียนเฟรมส่งไปยังจุดเชื่อมต่อ tapX และตัวโปรแกรมก็จะทำการอ่านเฟรมจาก /dev/tapX เช่นเดียวกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.6.3 การทำงานของอุปกรณ์เน็ตเวิร์กแบบเสมือน (Virtual Network Device)

สมมติว่า ทำการคอนฟิก IPX ให้แก่อุปกรณ์ที่ชื่อ tap0 แล้ว เมื่อเคอร์เนลต้องการที่จะส่งแพ็กเกต IPX ให้แก่อุปกรณ์ tap0 แพ็กเกตจะถูกส่งผ่านไปยังแอพพลิเคชันก่อน เพื่อทำการเข้ารหัสและย่อ (Compress) แพ็กเกต แล้วจึงจะทำการส่งต่อไปให้อีกฝั่งหนึ่งด้วย TCP หรือ UDP และเมื่อแอพพลิเคชันของอีกฝั่งหนึ่งทำการรับแพ็กเกตนั้นมาแล้ว ต้องมาทำการขยาย (Decompress) แพ็กเกตและถอดรหัสออกมาก่อน แล้วถึงจะนำแพ็กเกตนั้นส่งไปให้อุปกรณ์ทีเอพีอีกที ซึ่งเคอร์เนลจะกระทำกับแพ็กเกตนี้เหมือนว่ารับมาจากอุปกรณ์จริงแบบปกติ

6.6.4 ไดรเวอร์ของทียูเอ็นและทีเอพี

ไดรเวอร์ของทีพูเอ็นและทีเอพีใช้สำหรับการทำ Tunneling โดยเฉพาะ ซึ่งเป็นสิ่งจำเป็นสำหรับการทำ Vtun (Virtual Tunneling) อย่างมาก โดยไดรเวอร์ของทีพูเอ็นและทีเอพีต่างกันตรงที่เฟรมที่นำมาใช้งาน ไดรเวอร์ของทียูเอ็นจะใช้เฟรมไอพีในการทำงาน ส่วนไดรเวอร์ทีเอพีใช้เฟรมที่เป็นอีเทอร์เน็ต

6.6.5 การตั้งค่าคอนฟิกกูเรชัน

1) การสร้างโหนดของอุปกรณ์

```
mknod /dev/net/tun c 10 200
```

2) การเซตให้โมดูลของไดรเวอร์ถูกโหลดแบบอัตโนมัติ

ตัวเคอร์เนลจะต้องสนับสนุนการโหลดโมดูลไดรเวอร์แบบอัตโนมัติ ซึ่งมีขั้นตอนการเซตดังนี้

- ทำการเพิ่มคำสั่งลงไปในไฟล์ /etc/modules.conf

```
alias char-major-10-200 tun
```

- ทำการรันโมดูล

```
depmod –a
```

ซึ่งไดรเวอร์จะถูกโหลดขึ้นมาทันทีที่มีแอพพลิเคชันใดก็ตามมาแอ็กเซส /dev/net/tun

6.6.6 การเขียนโปรแกรมส่วนติดต่อ

1) การกำหนดอุปกรณ์เน็ตเวิร์ก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
int tun_alloc(char *dev)
{
```

```
struct ifreq ifr;
int fd, err;

if (fd = open("/dev/net/tun", O_RDWR) ) < 0)
        return tun_alloc_old(dev);

memset(&ifr, 0, sizeof(ifr) );
/* Flags: IFF_TUN – TUN device (no Ethernet headers)
 *        IFF_TAP        - TAP device
 *
 *        IFF_NO_PI      - Do not provide packet information
 */
ifr.ifr_flags = IFF_TUN;
if (*dev)
        strncpy(ifr.ifr_name, dev, IFNAMSIZ);

if (err = ioctl(fd, TUNSETIFF, (void *) &ifr) ) < 0) {
        close(fd);
        return err;
}
strcpy(dev, ifr.ifr_name);
return fd;
}
```

2) รูปแบบของเฟรม

ถ้ายังไม่มีการเซตแฟลก IFF_NO_PI รูปแบบของเฟรมแต่ละเฟรมจะเป็นดังนี้

```
Flags [2 bytes]
Proto [2 bytes]
Raw protocol(IP, Ipv6, etc) frame.
```

6.7 การนำเอาวีพีเอ็นมาใช้ในชีวิตจริง

เนื่องจากระบบเครือข่ายเน็ตเวิร์คส่วนใหญ่ที่เชื่อมต่ออยู่กับอินเทอร์เน็ต จะมีไฟร์วอลล์อยู่แล้ว ดังนั้น การกล่าวถึงการนำเอาวีพีเอ็นมาใช้นั้น จึงเป็นในรูปแบบของไฟร์วอลล์ (Firewall-Based VPN) ซึ่งเป็นที่นิยมใช้แพร่หลายมากที่สุด

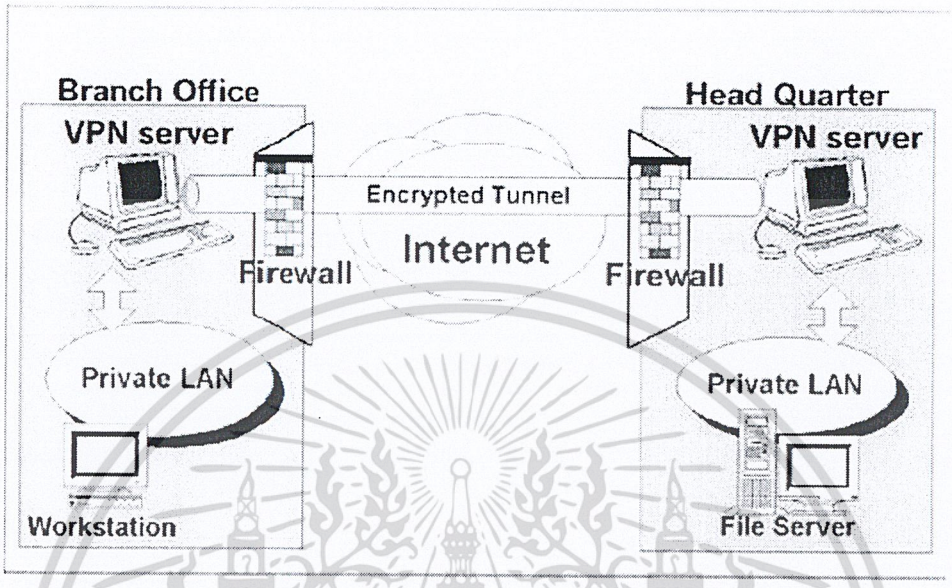
การพัฒนาเอาเทคโนโลยีวีพีเอ็นไปใช้ในองค์กรเพื่อช่วยอำนวยความสะดวกและปลอดภัย สามารถจำแนกได้เป็น 2 รูปแบบ

1. Site-to-Site

เมื่อองค์กรใด ต้องการติดต่อกับระบบเน็ตเวิร์คของสาขาขององค์กรตนข้ามเครือข่ายอินเตอร์เน็ต และใช้วีพีเอ็นในการควบคุมความปลอดภัยข้ามเน็ตเวิร์ค สิ่งที่ทุกองค์กรจะต้องมีในการทำระบบวีพีเอ็น คือ วีพีเอ็นเซิร์ฟเวอร์ (VPN Server) เพื่อตรวจสอบและทำการเข้าและถอด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รหัสแพ็กเก็ตต่างๆที่ส่งเข้าออกระหว่างภายในและภายนอกเน็ตเวิร์ค ถ้าหากไม่มีวีพีเอ็นเซิร์ฟเวอร์ในด้านใดด้านหนึ่ง การติดต่อจะเป็นไปไม่ได้เลย



รูปที่ 6-3: การนำวีพีเอ็นมาใช้ในรูปแบบ Site-to-Site

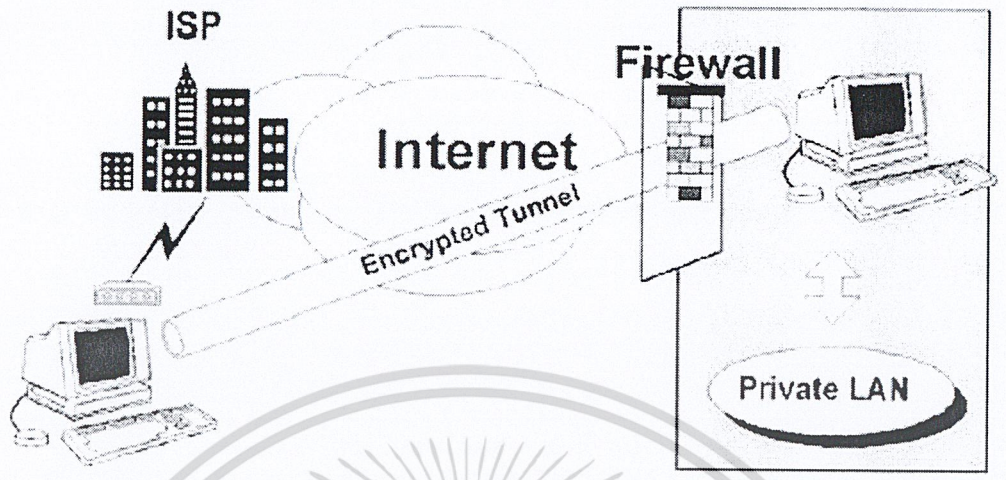
2. PC-to-Site

เนื่องจากทุกวันนี้มีความต้องการที่จะให้ผู้ที่ออกไปปฏิบัติงานนอกสำนักงาน สามารถติดต่อและส่งผ่านข้อมูลกับสำนักงานใหญ่ได้อย่างสะดวก รวดเร็วและประหยัด เมื่อสำนักงานใหญ่ต่อเข้าระบบอินเทอร์เน็ตที่มีเครือข่ายเชื่อมต่อกันทั่วโลก และเปิดให้ผู้ใช้งานจากภายนอกติดต่อผ่านไอเอสพีได้ การเชื่อมต่อที่เกิดขึ้น จะถูกควบคุมโดยระบบวีพีเอ็น

ระบบวีพีเอ็นในรูปแบบนี้ ทำได้ 2 ลักษณะ คือ

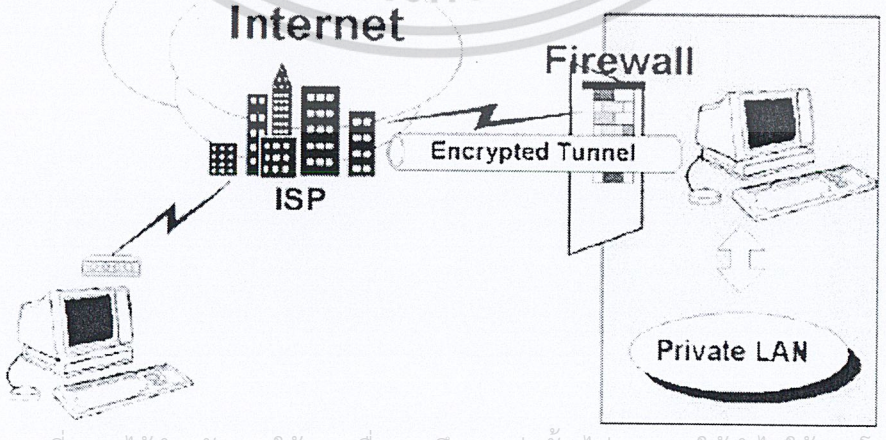
- Client-initiated: ผู้ใช้ที่ต้องการติดต่อกับเน็ตเวิร์คภายในของบริษัทจะโทรเข้าหาไอเอสพี หลังจากผ่านการตรวจสอบชื่อและรหัสผ่านแล้ว ไอเอสพีจะทำการตรวจสอบเพื่อดูว่าผู้ที่โทรเข้ามานั้นเป็นสมาชิกของตนหรือไม่ ถ้าใช่ ไอเอสพีจะอนุญาตให้ผู้นั้นสามารถผ่านเข้าสู่ระบบอินเทอร์เน็ต หลังจากนั้น ผู้ใช้งานจะสร้างทันแนลจากเครื่องเวิร์กสเตชัน (Workstation) ของตนไปสู่เครื่องวีพีเอ็นเซิร์ฟเวอร์เพื่อส่งข้อมูลกัน ดังนั้น ถ้าเครื่องเวิร์กสเตชันต้องการดึงข้อมูลจากไฟล์ภายในระบบเน็ตเวิร์คของบริษัท ก็จะสามารถทำได้โดยผ่านทางทันแนลนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 6-4: การนำวีพีเอ็นมาใช้ในรูปแบบ Client-initiated

- Network Access Server: บริษัทที่ต้องการให้ผู้ใช้งานสามารถโทรเข้าไอเอสพีและติดต่อมาที่ระบบเน็ตเวิร์คของบริษัท จะต้องทำการติดต่อกับไอเอสพีที่บริษัทตนเชื่อมสายเข้ามา บริษัททำการเดินสายเช่ามายังไอเอสพีและบอกให้ไอเอสพีทราบว่าทางบริษัทต้องการให้พนักงานของตนติดต่อผ่านระบบวีพีเอ็นมายังบริษัทของตนได้ เมื่อพนักงานทำการล็อกเข้า (log in) สู่ระบบอินเทอร์เน็ตที่ไอเอสพีแล้ว ทางไอเอสพีจะทำการสร้างทันแนลจากไอเอสพีถึงที่บริษัทของพนักงานคนนั้นและให้พนักงานทำการส่งข้อมูลไปยังบริษัทของตนเองผ่านทางทันแนลที่มีความปลอดภัย



รูปที่ 6-5: การนำวีพีเอ็นมาใช้ในรูปแบบ Network Access Network

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ทั้ง 2 ลักษณะนี้มีข้อเสียแตกต่างกันไป ในแบบ Client-initiated มีความยืดหยุ่นในการต่อเข้าไอเอสพีใดๆก็ได้ เนื่องจากทันแนลสร้างขึ้นที่ตัวผู้ใช้เองโดยตรงโดยไม่ต้องพึ่งไอเอสพี ในขณะที่แบบ Network access server ต้องใช้ไอเอสพีหรือกลุ่มของไอเอสพีเดียวกับที่บริษัทต่อระบบเน็ตเวิร์ค ทำให้มีข้อจำกัดในการใช้งานจากที่ต่างๆ แต่แบบ Network access server นี้ ผู้ใช้งานภายนอกสามารถใช้งานได้ทันที ไม่ว่าจะเป็นเครื่องประเภทไหน ในขณะที่แบบ Client-initiated ก่อนการใช้งาน ผู้ใช้ต้องติดตั้งโปรแกรมซึ่งเป็นวีพีเอ็นไคลเอนต์ลงที่เครื่องก่อน จึงจะติดต่อเข้าไปยัง Server ของตนได้

6.8 ประโยชน์ที่ได้รับจากวีพีเอ็น

จะช่วยให้องค์กรประหยัดค่าใช้จ่าย เพราะสามารถเข้าถึงเครือข่ายวีพีเอ็นของตนได้ไม่ว่าจะอยู่ที่ไหน โดยการเชื่อมต่อเข้ากับผู้ให้บริการท้องถิ่นนั้นๆ ทำให้ช่วยลดค่าใช้จ่ายในการติดต่อสื่อสารและลดค่าใช้จ่ายในส่วนของการดูแลรักษาระบบอีกด้วย นอกจากนี้ ระบบเครือข่ายวีพีเอ็นยังสามารถให้ความคล่องตัวในการเปลี่ยนแปลง เช่น การขยายเครือข่ายในอนาคต ซึ่งเป็นการทำให้องค์กรมีประสิทธิภาพการทำงานสูงสุด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 7
การออกแบบ

โครงงานนี้ เป็นการสร้างฮาร์ดแวร์ (Hardware) ต้นแบบที่มีการนำเอาชิป (Chip) มาใช้ในการเข้าและถอดรหัส

7.1 แนวคิดในการออกแบบ

การออกแบบชุดต้นแบบการเข้ารหัสด้วยฮาร์ดแวร์ จะเริ่มด้วยการมองภาพรวมของการทำงานทั้งหมดของระบบ หลังจากนั้นจะทำการศึกษาทฤษฎีต่าง ๆ ที่เกี่ยวข้องจนเข้าใจความต้องการทั้งหมดของระบบแล้ว เราจึงเริ่มทำการออกแบบระบบรวมทั้งหมด โดยในการออกแบบนั้น จะทำการแบ่งระบบออกเป็นส่วนย่อย ๆ โดยในแต่ละส่วนจะถูกออกแบบให้มีส่วนติดต่อที่เหมือนกับส่วนติดต่อมาตรฐานให้มากที่สุด เพื่อความสามารถในการใช้งานร่วมกันได้กับระบบที่มีอยู่แล้วในปัจจุบัน และความสามารถในการนำเอาส่วนต่าง ๆ กลับมาใช้ได้อีก รวมถึงเพื่อลดภาระของงานที่จะใช้ในการแกะปัญหากล่องดำ

7.2 ภาพรวมในการออกแบบ

ในการออกแบบ เราจะยึดเอาการติดต่อระหว่างส่วนต่าง ๆ เป็นหลัก ซึ่งทำให้เราแบ่งส่วนต่าง ๆ ออกมาเป็น 3 ส่วนใหญ่ ๆ คือ ส่วนของอุปกรณ์ฝั่งฮาร์ดแวร์ ซอฟต์แวร์ดีไวซ์ไดรเวอร์ (Software Device Driver) และซอฟต์แวร์วีพีเอ็น (Software VPN) โดยฮาร์ดแวร์และซอฟต์แวร์ทั้ง 2 ส่วน จะทำงานร่วมกันโดยผ่านบริการของระบบปฏิบัติการ (OS Services)

ส่วนของฮาร์ดแวร์จะทำการติดต่อกับไอเอสเอบัส (ISA Bus) โดยมีการ์ดไอเอสเอ (ISA Card) เป็นส่วนหน้าในการเชื่อมต่อของฝั่งฮาร์ดแวร์ โดยการทำงานส่วนใหญ่ในฝั่งนี้จะขึ้นอยู่ที่บอร์ดเอฟพีจีเอ (FPGA Board) เป็นหลัก ซึ่งการเข้ารหัสของข้อมูลจะถูกกระทำในส่วนของฝั่งฮาร์ดแวร์นี้ รวมถึงการควบคุมการติดต่อกับไอเอสเอบัสด้วย ในรุ่นแรกนี้ เราจะยังไม่ได้ใช้ความสามารถทั้งหมดของบัส เนื่องด้วยข้อจำกัดของเวลา เนื่องจากการพัฒนาในส่วนของฮาร์ดแวร์จะต้องใช้เวลานานกว่าการพัฒนาในส่วนของซอฟต์แวร์

สำหรับส่วนของการเข้ารหัสที่อยู่ในฝั่งฮาร์ดแวร์นั้น จะใช้การเข้ารหัสแบบ XOR เนื่องด้วยข้อจำกัดของบอร์ดเอฟพีจีเอที่มีขนาดเล็ก แต่ผู้พัฒนาเองก็ได้ทำในส่วนของการเข้ารหัสที่ใช้วิธีการเข้ารหัสแบบ DES เอาไว้แล้วเช่นกัน

ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์ จะทำงานโดยฝังตัวเข้ากับบริการของระบบปฏิบัติการ โดยจะอาศัยบริการหลัก (กลาง) ของระบบปฏิบัติการทำงาน และสร้างบริการของระบบปฏิบัติการขึ้นมาใหม่ชุดหนึ่ง เพื่อใช้ในการเข้าถึงอุปกรณ์ฝั่งฮาร์ดแวร์ รวมถึงเพื่อให้บริการแก่โปรแกรมใช้งานทั่วไปให้สามารถเรียกใช้งานได้ด้วย ซึ่งในชุดทดลองนี้ก็คือโปรแกรมวีพีเอ็นนั่นเอง สำหรับระบบปฏิบัติการที่เลือกมาใช้

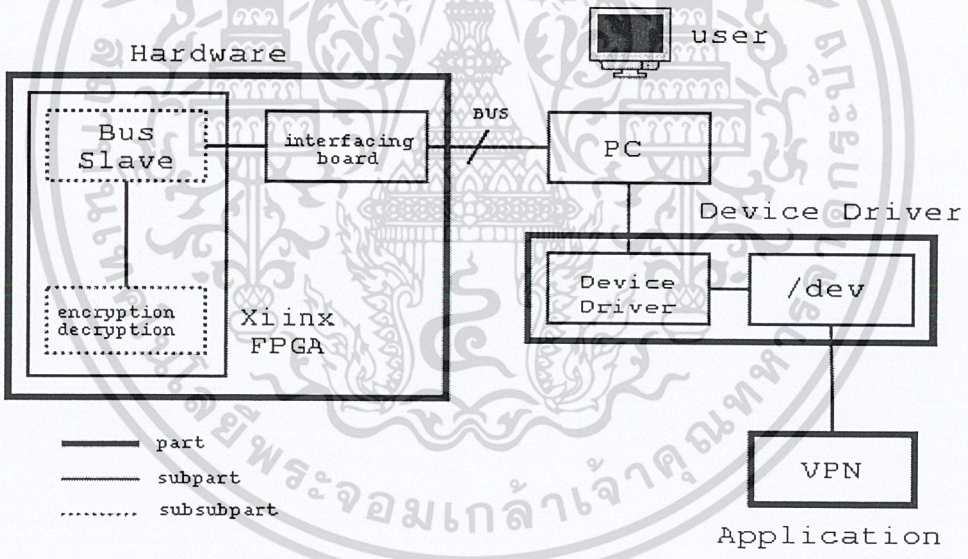
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะเป็นระบบปฏิบัติการ Linux ที่ Kernel รุ่น 2.4.0 มาตรฐาน สาเหตุที่เลือกนำระบบปฏิบัติการตัวนี้มาใช้เพราะเล็งเห็นถึงความง่ายในการพัฒนาและรวมถึงชุดพัฒนาที่สามารถหามาใช้ได้ง่าย

อีกส่วนหนึ่งของซอฟต์แวร์ คือ โปรแกรมวีพีเอ็น ที่เรานำมาดัดแปลงเพื่อให้มาเรียกใช้งานตัวดีไวซ์ไดรเวอร์อีกที ในส่วนของโปรแกรมวีพีเอ็นนี้ เราได้เลือกใช้โปรแกรมวีพีเอ็นที่มีอยู่แล้ว ซึ่งเป็นซอฟต์แวร์เสรี เพื่อความง่ายและรวดเร็วในการพัฒนา อันเกิดจากความไม่ซ้ำซ้อนในการทำงาน สำหรับสาเหตุที่เราทำการแบ่งส่วนของโปรแกรมวีพีเอ็นออกมาจากตัวซอฟต์แวร์วีพีเอ็นอีกทีหนึ่ง เนื่องจากเวลาที่ทำการเรียกใช้งานส่วนของโปรแกรมวีพีเอ็นกับส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์นั้น ยังต้องติดต่อถึงกันและกันโดยผ่านส่วนติดต่อมาตรฐานของระบบปฏิบัติการอีกที รวมถึงในด้านสภาวะแวดล้อมของการใช้งาน โปรแกรมดีไวซ์ไดรเวอร์จะรันในเคอร์เนลโหมด ส่วนโปรแกรมวีพีเอ็นจะรันในยูสเซอร์โหมด

7.3 การทำงานร่วมกันในแต่ละส่วน

ส่วนต่างๆของชุดต้นแบบนี้จะติดต่อซึ่งกันและกันตามรูปด้านล่าง ในระบบของชุดต้นแบบจะไม่มีการติดต่อถึงกันโดยตรงระหว่างส่วนต่าง ๆ นอกเหนือจากที่แสดงไว้ในรูปนี้



รูปที่ 7-1: การทำงานร่วมกันระหว่างแต่ละส่วนของระบบ

จากรูป จะเห็นส่วนของระบบถูกแย่งออกเป็น 3 ส่วน ตามที่ได้กล่าวเอาไว้ก่อนหน้านี้แล้ว

7.4 การออกแบบในแต่ละส่วนย่อย

7.4.1 การออกแบบส่วนของอุปกรณ์ฮาร์ดแวร์

เนื่องจากการทำงานโดยใช้ซอฟต์แวร์นั้น ยังมีขีดจำกัดในเรื่องความเร็วในการส่งข้อมูล และความยืดหยุ่นในการนำไปประยุกต์ใช้งานในสภาพการทำงานปกติ ดังนั้นโครงงานนี้จึงได้นำเอาการเข้ารหัสด้วยฮาร์ดแวร์เข้ามาช่วยแก้ปัญหาในส่วนนี้ และต้นแบบในโครงงานนี้ยัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ แสวงหาผลประโยชน์ทางการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สามารถนำไปพัฒนาต่อให้ใช้งานได้เป็นอิสระโดยไม่ต้องพึ่งพาเครื่องคอมพิวเตอร์ส่วนบุคคลในการทำงานได้ด้วย

ในส่วนของอุปกรณ์ฮาร์ดแวร์ จะประกอบไปด้วยอุปกรณ์ 2 ชิ้น คือ ส่วนของการ์ดไอเอสเอ และส่วนของบอร์ดเอฟพีจีเอ

7.4.1.1 ส่วนของการ์ดไอเอสเอ

ส่วนของการ์ดไอเอสเอนี้ จะทำหน้าที่ป้องกันการชนกันของสัญญาณไฟฟ้าจากไอเอสเอบัสและบอร์ดเอฟพีจีเอ ซึ่งในการทำงานของส่วนนี้จะไม่มีอะไรมาก โดยจะทำหน้าที่รับสัญญาณจากส่วนควบคุมการติดต่อกับไอเอสเอบัสในบอร์ดเอฟพีจีเอ ซึ่งเป็นสัญญาณการสั่งให้บอร์ดทำงาน โดยทำการส่งสัญญาณออกไปยังไอเอสเอบัส ในกรณีที่เป็นสัญญาณที่ต้องใช้ช่องทางร่วมกับอุปกรณ์ตัวอื่นในไอเอสเอบัส และการกำหนดทิศทางการรับส่งข้อมูล ในกรณีที่ข้อมูลเป็นแบบ 2 ทิศทาง เช่น ข้อมูลในคาต้าบัส (Data Bus)

7.4.1.2 ส่วนของบอร์ดเอฟพีจีเอ

ส่วนของบอร์ดเอฟพีจีเอนี้ จะติดต่อโดยตรงกับการ์ดไอเอสเอ และทำการรับส่งข้อมูลทั้งหมดผ่านทางการ์ดไอเอสเอ การติดต่อจะใช้การเดินสายลอยธรรมดา โดยมีที่เชื่อมจุดศักย์ไฟฟ้าศูนย์ของบอร์ดเอฟพีจีเอเข้ากับจุดศักย์ไฟฟ้าศูนย์ของคอมพิวเตอร์โดยผ่านทางเครื่องพีซี โดยบอร์ดเอฟพีจีเอที่เลือกนำมาใช้จะเป็นรุ่น XC4010E ซึ่งเป็นรุ่นที่ใช้กันในการเรียนของภาควิชา

ในบอร์ดเอฟพีจีเอนี้ จะถูกโปรแกรมให้ทำหน้าที่เป็นตัวควบคุมการติดต่อกับไอเอสเอบัส และทำหน้าที่ในส่วนของการเข้ารหัส ซึ่งในโปรแกรมที่เราเขียนมานั้น ได้แบ่งออกเป็น 2 ส่วน คือ ส่วนของชุดโปรแกรมการเข้ารหัสและโปรแกรมที่เชื่อมต่อระหว่างการ์ดไอเอสเอและชุดโปรแกรมการเข้ารหัส

ซึ่งในโอกาสหน้า ถ้าเรามีบอร์ดเอฟพีจีเอที่มีขนาดใหญ่กว่านี้ เราอาจจะถอดเอาชุดโปรแกรมการเข้ารหัสตัวปัจจุบันออกและแทนด้วยเข้ารหัสชุดใหม่ โดยไม่จำเป็นต้องแก้ส่วนที่ติดต่อกับไอเอสเอบัส รวมถึงยังสามารถแก้ส่วนที่ติดต่อกับไอเอสเอบัสเพื่อเปลี่ยนเป็นบัสชนิดอื่นได้โดยไม่กระทบกับส่วนของการเข้ารหัส

7.4.1.3 ส่วนของโปรแกรมที่โหลดลงบอร์ดเอฟพีจีเอ

ในส่วนของโปรแกรมที่ทำการโหลดลงบนบอร์ดเอฟพีจีเอนี้ ถูกเขียนด้วยภาษาวีเอชดีแอล ซึ่งจากโค้ดของโปรแกรมที่เขียนขึ้นนั้น จะมีส่วนที่ติดต่อกับด้านการ์ดไอเอสเอ ดังแสดงในโค้ดที่ตัดมาด้านล่างนี้

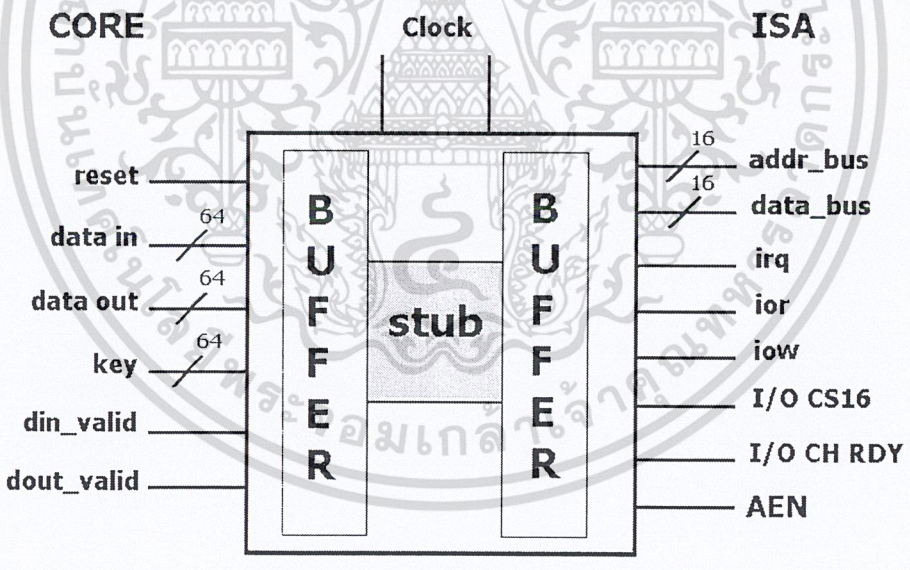
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
port( sys_clk     :in std_logic;
      sys_reset   :in std_logic;
```

```
        isa_aen     :in std_logic;

        cs_out      :out std_logic;
        dir_out     :out std_logic;

        addr_bus    :in std_logic_vector(11 downto 0);
        data_bus    :inout std_logic_vector(15 downto 0);
        irq         :out std_logic :='1';
        ior         :in std_logic;
        iow         :in std_logic;
        iocs16      :out std_logic;
        iochrdy     :out std_logic
);
```

พอร์ต cs_out และ dir_out ใช้เพื่อควบคุมการทำงาน และควบคุมทิศทางของสัญญาณภายในการ์ดไอเอสเอ ส่วนสัญญาณที่เหลือจะเป็นสัญญาณของไอเอสเอบัสที่ถูกควบคุมด้วยการ์ดไอเอสเอ

ส่วนโปรแกรมที่ใช้ในการเชื่อมต่อระหว่างการ์ดไอเอสเอและชุดโปรแกรมเข้ารหัส คือ โปรแกรมสตับ โดยโปรแกรมสตับจะมีส่วนเชื่อมต่อและโครงสร้าง ตามที่แสดงให้เห็นในรูปด้านล่างนี้



รูปที่ 7-2:ส่วนเชื่อมต่อและโครงสร้างของสตับ

โดยด้านซ้ายมือเป็นส่วนของชุดโปรแกรมเข้ารหัส และส่วนด้านขวามือคือ ไอเอสเอบัส ซึ่งเราสามารถปรับเปลี่ยนชุดโปรแกรมการเข้ารหัสได้โดยที่ไม่จำเป็นต้องแก้ไขตัวโปรแกรมสตับ ถ้าหากโปรแกรมการเข้ารหัสนั้นมีความกว้างของข้อมูลและกุญแจเท่ากันกับชุดโปรแกรมเข้ารหัสของเรา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทำงานของอุปกรณ์ฮาร์ดแวร์ของเรา จะเป็นลักษณะพอร์ตเบส (port-based) ดีไวซ์ โดยแอดเดรส (Address) ที่เราใช้จะอยู่ในช่องของ 0x0110 ถึง 0x012F โดยแบ่งออกเป็น 3 ช่วง คือ

- แอดเดรสช่วงแรก (0x0110 ถึง 0x0117) ใช้สำหรับรับข้อมูลที่จะทำการเข้ารหัสจากไอเอสเอบัส
- แอดเดรสช่วงที่ 2 (0x0118 ถึง 0x011F) สำหรับใช้รับข้อมูลที่เป็นคีย์ในการเข้ารหัส
- และช่วงสุดท้าย (0x0120 ถึง 0x0130) จะเป็นข้อมูลที่ผ่านการเข้ารหัสแล้ว

โดยการรับและส่งข้อมูล จะใช้การรับส่งข้อมูลทีละ 2 ไบต์ (ไบต์ละ 8 บิต) และการรับ-ส่งจะใช้แอดเดรสทีละ 2 ช่วง เพื่อให้สามารถนำไปดัดแปลงจากการทำงานด้วยพอร์ตเป็นเมโมรีแมป (Memory -Mapped) แทนได้ง่ายยิ่งขึ้น

7.4.2 ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์

ส่วนของซอฟต์แวร์ดีไวซ์ไดรเวอร์นั้นถูกโปรแกรมออกมาเพื่อทำหน้าที่ 2 อย่าง คือ การควบคุมข้อกำหนดในการใช้งานอุปกรณ์ และสร้างบริการสำหรับการใช้งานอุปกรณ์จากโปรแกรมผู้ใช้งานทั่วไป

แต่ซอฟต์แวร์ดีไวซ์ไดรเวอร์ยังแบ่งส่วนติดต่อได้เป็น 2 ฝั่ง คือ ส่วนติดต่อกับอุปกรณ์และส่วนติดต่อผู้ใช้งาน ในส่วนติดต่อกับอุปกรณ์นั้น เราจะใช้บริการในระดับล่างของระบบปฏิบัติการเพื่อจองทรัพยากร และเพื่อรับส่งข้อมูลกับอุปกรณ์ฮาร์ดแวร์ เช่น การจองพอร์ตแอดเดรสของอุปกรณ์ และการส่งข้อมูลออกทางพอร์ต เป็นต้น สำหรับส่วนติดต่อผู้ใช้งานนั้น จะใช้บริการในระดับบนเพื่อเพิ่มบริการของตัวเองเข้ากับบริการที่มีอยู่ของระบบปฏิบัติการ การทำงานหลัก ๆ ของไดรเวอร์ คือ การควบคุมการรับส่งข้อมูลระหว่างส่วนติดต่อทั้ง 2 ฝั่ง

สำหรับส่วนติดต่อกับผู้ใช้งานนั้น เราจะเลือกใช้การให้บริการโดยมองอุปกรณ์เสมือนเป็นแฟ้มข้อมูลธรรมดา (Device File) ซึ่งในการใช้งานนั้น โปรแกรมฝั่งผู้ใช้งานสามารถเปิดดีไวซ์ไฟล์ได้โดยผ่านทางการเรียกใช้ฟังก์ชัน (Function) open() และสามารถรับส่งข้อมูลเพื่อทำการเข้ารหัสได้โดยผ่านทางการเรียกใช้ฟังก์ชัน read() และ write() รวมถึงสามารถควบคุมการใช้งานอื่น ๆ ได้โดยผ่านทางการเรียกใช้ฟังก์ชัน ioctl() ซึ่งในตัวติดต่อกับผู้ใช้งานนี้นั้น จะให้บริการเฉพาะฟังก์ชันก์ที่จำเป็นสำหรับการใช้งานในเบื้องต้นนี้เท่านั้น โดยในซอฟต์แวร์ดีไวซ์ไดรเวอร์ของเรานั้นจะรองรับฟังก์ชันตามการประกาศโครงสร้างต่อไปนี้

```
static struct file_operations mind_fops = {
        read:    mind_read,
        write:   mind_write,
        ioctl:   mind_ioctl,
        open:    mind_open,
        release: mind_release,
};
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การควบคุมการติดต่อสื่อสารนั้น เราจะต้องสามารถรับประกันความถูกต้องของข้อมูลได้ด้วย ซึ่งจะต้องรับประกันได้ว่า ถ้าโปรแกรมสามารถ write() หรือ read() ได้ โดยไม่ได้รับการแจ้งเตือน (return) ความผิดพลาด ดังนั้นข้อมูลที่ได้ทำการรับส่งหรือทำการประมวลผลกับดีไวซ์ไดรเวอร์นั้น ก็จะต้องเป็นข้อมูลที่มีความถูกต้องด้วย ซึ่งสิ่งสำคัญสำหรับการนี้คือ การป้องกันการเรียกโปรแกรมซ้อนทับ โดยเฉพาะอย่างยิ่ง การเขียนทับที่ตำแหน่งเดียวกัน ซึ่งในประเด็นนี้เราจึงแก้ปัญหาโดยการใช้ซีมาฟอร์ (Semaphore) มาช่วยในการป้องกันการรันฟังก์ชันซ้อนทับกัน โดยเราได้ใช้ซีมาฟอร์ทั้งหมด 3 ตัว ดังตัวอย่างโค้ด (Code) ด้านล่าง

```
static struct semaphore mind_read_sem;      // mutex read()
static struct semaphore mind_write_sem;     // mutex write()
static struct semaphore mind_ioctl_sem;     // mutex key and port
```

โดยเราได้ทำการเริ่มต้นเซต (Set) ค่าทั้งหมดของซีมาฟอร์ในตอนเริ่มต้นของการทำงานของดีไวซ์ไดรเวอร์ นอกจากนั้น ยังมีการรับประกันในขณะที่มีการเปลี่ยนแปลงข้อมูลที่มีความสำคัญเป็นพิเศษ เช่น ข้อมูลที่ใช้ในการควบคุมและการนับของโปรแกรมดีไวซ์ไดรเวอร์โดยการใช้สปินล็อค (spinlock)

ดีไวซ์ไดรเวอร์จะถูกออกแบบมาโดยเน้นที่ความสามารถในการปรับแต่งได้ โดยซอสโค้ด (Source Code) ของโปรแกรมดีไวซ์ไดรเวอร์จะประกอบขึ้นมาจาก 2 ไฟล์ คือ ไฟล์ชื่อ mind.c และ mind.h โดยทั้ง 2 ไฟล์นี้ สามารถสร้างดีไวซ์ไดรเวอร์ได้ 2 รูปแบบการทำงาน คือ แบบแรกจะเป็นดีไวซ์ไดรเวอร์ที่จะทำการเรียกใช้อุปกรณ์เข้ารหัส และแบบที่สองจะเป็นดีไวซ์ไดรเวอร์ที่จะทำการเรียกใช้อีมูเลเตอร์ (Emulator) ที่รวมอยู่ในดีไวซ์ไดรเวอร์เอง

สาเหตุที่ทำดีไวซ์ไดรเวอร์ออกมา 2 ชุด เนื่องจากว่าในการทำงานของโปรแกรมวีพีเอ็น เราต้องมีเกตเวย์ (Gateway) 2 ฝั่ง แต่เนื่องจากเรามีอุปกรณ์ฮาร์ดแวร์ชุดเดียว เราจึงได้ทำดีไวซ์ไดรเวอร์ออกมา 2 ชุด อีกทั้งการที่ฝั่งหนึ่งเป็นฮาร์ดแวร์จึงสามารถทำงานร่วมกับโปรแกรมที่เป็นซอฟต์แวร์อีมูเลเตอร์ได้ ซึ่งสามารถแสดงให้เห็นถึงความเป็นมอดูลาร์ (Modular) ของระบบของเรา

นอกจากความสามารถในการปรับแต่งการทำงานแล้ว เรายังสามารถกำหนดระดับของการแสดงข้อความดีบั๊ก (Debug) ได้ โดยการส่งผ่านระดับของการดีบั๊กให้กับตัวคอมไพเลอร์ (Compiler) หรือสามารถกำหนดระดับของการดีบั๊กในซอสโค้ดเองได้ ความสามารถในการปรับแต่งทั้งหมดนี้จะอาศัยคอมไพเลอร์ เพื่อทำ preprocessing คือ เพื่อเลือกเอาชุดของคำสั่งที่ถูกต้องมาคอมไพล์ (Compile) เป็นดีไวซ์ไดรเวอร์ ซึ่งการคอมไพล์ดีไวซ์ไดรเวอร์จะสามารถทำได้โดยใช้โปรแกรม make ซึ่งในตำแหน่งที่เก็บซอสโค้ดของดีไวซ์ไดรเวอร์จะมีไฟล์สคริปต์ (Script

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

File) ชื่อ "Makefile" มาให้ด้วย โดยกฎที่สำคัญที่สุดของการเมกไฟล์ (Makefile) คือ เป็นตัวกำหนดชนิดของดีไวซ์ไดรเวอร์ ดังนี้

```
mind.o: mind.c mind.h ldd.h
	gcc -Wall -c -D__KERNEL__ -DMODULE -D__NO_VERSION__ -O2 mind.c \
		-o mind.o
no_mind.o: mind.c mind.h ldd.h
	gcc -Wall -c -DNO_MIND -D__KERNEL__ -DMODULE -D__NO_VERSION__ -
	O2 \ mind.c -o no_mind.o
```

จะเห็นได้ว่าสิ่งที่ต่างกันคือ ในกระบวนการสร้าง mind.o จะมีการประกาศมาโคร (Macro) ชื่อ NO_MIND ไว้ด้วย

7.4.3 ส่วนของโปรแกรมวีพีเอ็น

ส่วนของโปรแกรมวีพีเอ็นนี้ เราได้ทำการดัดแปลงส่วนของการเข้ารหัสของโปรแกรม รวมถึงการเพิ่มส่วนของการควบคุมข้อกำหนดในการใช้งานอุปกรณ์เข้าไปในตัวโปรแกรมด้วย ในการดัดแปลงนั้น เราได้เปลี่ยนส่วนเข้ารหัสของตัวเดิม โดยเราได้ลบตัวเข้ารหัสเดิมออก แล้วให้มาเรียกการเข้ารหัสโดยใช้ฮาร์ดแวร์เข้ารหัสผ่านทางซอฟต์แวร์ดีไวซ์ไดรเวอร์แทน โดยเราจะทำการเปิดดีไวซ์ไฟล์ในตอนต้นโปรแกรมเพื่อว่า เมื่อเราทำการแตกโพรเซส (fork) เพื่อรอบรับการงานซ้อนกันหลาย ๆ เซสชัน (Session) แล้ว เราจะได้ไม่จำเป็นต้องเปิดไฟล์อีก

เราเริ่มต้นเพื่อเตรียมการเข้ารหัสได้ง่าย ๆ ด้วยโค้ดที่นำมาแสดงนี้

```
if ((fdesc = open("/dev/misc/isagcrypt/mind", O_RDWR)) < 0) {
	syslog(LOG_ERR, "Can't open /dev/misc/isagcrypt/mind");
	return fdesc; }
else {
	syslog(LOG_INFO, "Opened /dev/misc/isagcrypt/mind");
	close(fdesc_r);
}
```

ในการแก้ไขโปรแกรมของเราในครั้งนี้ จะไม่กระทบกับการทำงานส่วนเดิมอื่น ๆ ของตัวโปรแกรมวีพีเอ็น ซึ่งทำให้เราสามารถทำการถอดฟังก์ชันการเข้ารหัสเข้าออกได้โดยไม่ต้องทำการคอมไพล์ตัวโปรแกรมวีพีเอ็นใหม่ และในการเปลี่ยนมาใช้อุปกรณ์เข้ารหัสของเรานั้น เราจะต้องป้องกันความถูกต้องของข้อมูลด้วย เพราะระบบการเข้ารหัสของเรายังเป็นระบบเดี่ยวแบบวิ่งทางเดียว ซึ่งทำให้ไม่สามารถทำการเข้ารหัสแบบขนานพร้อม ๆ กันทีละหลายชุดได้ แต่ในการทำงานจริงแล้ว เราจะต้องทำการประมวลผลให้แต่ละเซสชันทำงานไปพร้อม ๆ กัน ในกรณีนี้เราได้นำเอาการใช้ SysV Semaphore มาช่วย โดยทำการร้องขอซีมาฟอร์จากระบบปฏิบัติการเพื่อใช้ร่วมกันระหว่างโพรเซส (Process)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โปรแกรมวีพีเอ็นที่เราใช้นี้ ยังมีข้อจำกัดในการทำงานหลายอย่าง เช่น ยังไม่มีระบบการแลกเปลี่ยนกุญแจในการเข้ารหัส และการเชื่อมต่อยังเป็นแบบสแตติกไคลเอ็นต์เซิร์ฟเวอร์ (Static

Client Server) คือ เกตเวย์ฝั่งหนึ่งจะต้องเปิดรอการเชื่อมต่อ และอีกฝั่งหนึ่งจะต้องร้องขอการเชื่อมต่อเข้าไป ซึ่งถือว่าเป็นความไม่ยืดหยุ่นในการนำไปใช้งาน ตัวอย่างการคอนฟิกกูเรชัน (Configuration) ไฟล์ของโปรแกรมวีพีเอ็น

```
# TUN example. Session 'cobra'.
cobra {
  pass  Ma&^TU;          # Password
  type  tun;             # IP tunnel
  proto udp;             # UDP protocol
  comp  lzo:9;           # LZO compression level 9
  encr  yes;             # Encryption
  keepalive yes;         # Keep connection alive

  up {
        # Connection is Up
        # 10.3.0.1 - local, 10.3.0.2 - remote
        ifconfig "%% 10.3.0.1 pointopoint 10.3.0.2 mtu 1450";
  };
}
```

7.5 แนวคิดในการออกแบบชุดทดสอบ

ในการทำงานครั้งนี้ เราได้ออกแบบให้มีลักษณะเป็นมอดูลาร์ให้มากที่สุด ดังนั้นแต่ละส่วนจะมีการทำงานเป็นอิสระ ซึ่งจะช่วยเป็นอย่างมากในเรื่องของการนำเอาส่วนต่าง ๆ กลับมาใช้ได้อีกในงานต่อไป และในการทำงานครั้งนี้ เราได้ออกแบบชุดทดลองออกมาเป็น 2 ชุดเพื่อทำการทดลองส่วนของอุปกรณ์ฮาร์ดแวร์ และส่วนของโปรแกรมดีไวซ์ไดรเวอร์ ในชุดทดลองสำหรับแต่ละส่วนนี้จะทดลองการทำงานของส่วนนั้น ๆ ได้อย่างสมบูรณ์โดยไม่ต้องอาศัยส่วนอื่น ๆ ของระบบช่วยในการทดลอง

7.6 ชุดทดสอบส่วนอุปกรณ์ฮาร์ดแวร์

ในชุดทดลองฮาร์ดแวร์นั้น เราจะใช้การทดลองด้วยอินพุตเวกเตอร์ (Input Vector) โดยแบ่งออกเป็น 2 ชุดย่อย ชุดแรกเป็นการทำลองโดยใช้ค่าตายตัว โดยอิงเลือกค่าต่อไปนี้เป็นค่าหลัก 0x0000000000000000, 0x0123456789ABCDEF, 0xFEDCBA9876543210 และ 0xFFFFFFFFFFFFFFFF และทำการจับคู่กันทั้งหมดในแซมเปิลสเปซ (Sample Space) แล้วนำมาเทียบกับค่าที่คาดหมายว่าจะได้รับ ส่วนชุดทดลองหลังนั้น เราจะทำการสุ่มเลือกค่าระหว่าง 0x0000000000000000 ถึง 0xFFFFFFFFFFFFFFFF มาจำนวนหนึ่งและทำการทดลองการเข้ารหัสและเทียบกับโปรแกรมเข้ารหัสที่เขียนไว้ในชุดทดลอง โดยโปรแกรมทดสอบทั้ง 2 ชุดนั้น จะเขียนด้วยภาษาแอสเซมบลี และทำการทดสอบบนระบบปฏิบัติการไมโครซอฟท์ดอส (Microsoft DOS)

ในการทำงานในส่วนของอุปกรณ์ฮาร์ดแวร์นี้ยังมีการทดสอบเฉพาะกิจด้วยอุปกรณ์วัดค่าลอจิก (Logic) บางประเภทเช่น ลอจิกโพรบ (Logic Probe) และ ดิจิตอลออสซิลโลสโคป (Digital Oscilloscope) เพื่อทำการตรวจสอบการทำงานจริงในส่วนของตัวฮาร์ดแวร์ และ รวมถึงการป้อนค่าไปที่บอร์ดเอฟพีจีเอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เพื่อทำสอบการทำงานเฉพาะบางอย่าง แต่ในการทดลองส่วนนี้เป็นการทดลองเฉพาะกิจ ซึ่งจะไม่ขอกล่าวรายละเอียดในที่นี้

7.7 ชุดทดสอบส่วนของระบบดีไวซ์ไดรเวอร์

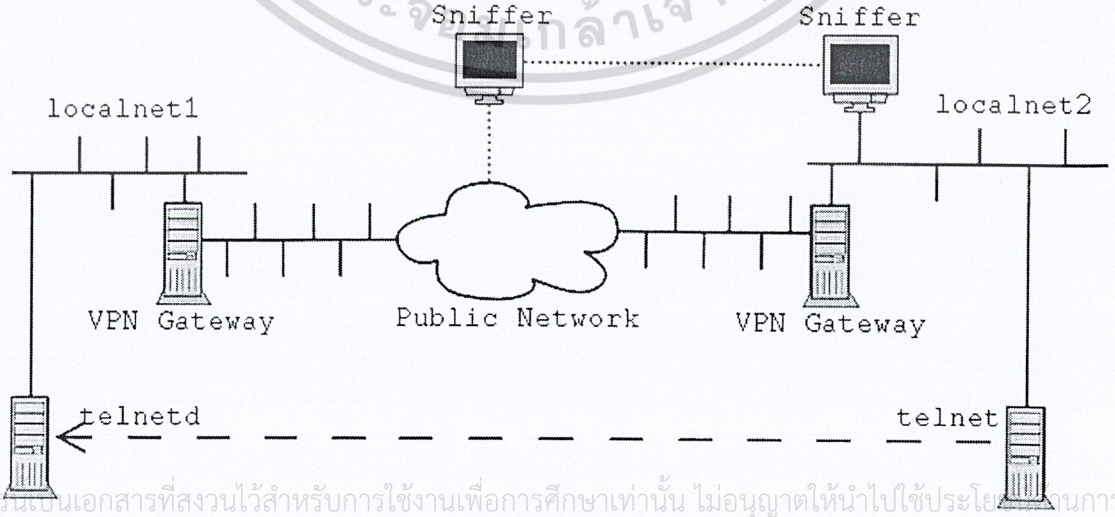
จากการออกแบบ ตัวดีไวซ์ไดรเวอร์จะเน้นที่ความยืดหยุ่นในการนำไปทำงาน โดยโปรแกรมดีไวซ์ไดรเวอร์ถูกออกแบบให้สามารถทำงานได้โดยไม่ต้องใช้อุปกรณ์ฮาร์แวร์ด้วย เป็นผลให้ *ถ้าเราสามารถมั่นใจได้ว่า ดีไวซ์ไดรเวอร์แบบที่ไม่อาศัยฮาร์ดแวร์สามารถทำงานได้ถูกต้องแล้ว เราก็สามารถนำเอาดีไวซ์ไดรเวอร์ชุดนี้ไปทดลองกับดีไวซ์ไดรเวอร์ในเวอร์ชันที่ทำงานกับอุปกรณ์ฮาร์ดแวร์ได้*

ในการพัฒนา เราได้ทำดีไวซ์ไดรเวอร์แบบที่ไม่ต้องใช้ฮาร์ดแวร์ขึ้นมาก่อน และทำการทดสอบดีไวซ์ไดรเวอร์ชุดนี้ด้วยอินพุตเวกเตอร์เดียวกับที่ใช้ในการทดสอบส่วนอุปกรณ์ฮาร์ดแวร์ แต่เขียนขึ้นมาใหม่ด้วยภาษาซีเพื่อความง่ายในการใช้งานและทำความเข้าใจ จนเมื่อเรามั่นใจดีไวซ์ไดรเวอร์ชุดนี้แล้ว เราจะสามารถนำดีไวซ์ไดรเวอร์ชุดนี้เป็นฐานในการทดสอบดีไวซ์ไดรเวอร์ในรูปแบบที่ใช้อุปกรณ์ฮาร์ดแวร์ได้ต่อไป

หลังจากที่เราทำดีไวซ์ไดรเวอร์เวอร์ชัน (Version) แรกเสร็จเรียบร้อยแล้ว ก็จะเริ่มทำงานปรับแก้ดีไวซ์ไดรเวอร์ตัวเดิมให้มาติดต่อกับอุปกรณ์ฮาร์ดแวร์เพื่อทำการเข้ารหัส แทนการใช้การเข้ารหัสภายในตัวดีไวซ์ไดรเวอร์ในแบบเดิม และทำการทดสอบดีไวซ์ไดรเวอร์ชุดนี้ โดยในการทดสอบดีไวซ์ไดรเวอร์ชุดนี้ จะเท่ากับการทดสอบการทำงานร่วมกันระหว่างอุปกรณ์ฮาร์ดแวร์เข้ารหัสร่วมกับดีไวซ์ไดรเวอร์ โดยใช้ดีไวซ์ไดรเวอร์ในเวอร์ชั่นแรกที่ไม่ต้องการฮาร์ดแวร์มาเป็นตัวเปรียบเทียบ

7.8 ชุดทดสอบโปรแกรมวีพีเอ็น

ในการทดสอบโปรแกรมนี้ จะใช้การทดสอบจากการใช้งานจริง โดยออกแบบสภาพแวดล้อมให้เหมือนในการใช้งานจริง ดังแสดงในรูปด้านล่าง



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 7-3: รูปแบบการเชื่อมต่อเพื่อทดสอบโปรแกรมวีพีเอ็น

โดยวีพีเอ็นเกตเวย์ฝั่งหนึ่งจะใช้โปรแกรมวีพีเอ็นร่วมกับดีไวซ์ไดรเวอร์ที่ไม่จำเป็นต้องใช้ฮาร์ดแวร์ ส่วนวีพีเอ็นเกตเวย์อีกฝั่งหนึ่งจะเป็นการทำงานร่วมกันระหว่างโปรแกรมวีพีเอ็นร่วมกับดีไวซ์ไดรเวอร์และอุปกรณ์ฮาร์ดแวร์เข้ารหัส

ในช่วงของการทดสอบนั้น อาจจะต้องมีการทดสอบผลกระทบของการปรับแก้โปรแกรมวีพีเอ็นด้วยการปรับแต่งการตั้งค่าในไฟล์คอนฟิกูเรชันของโปรแกรมวีพีเอ็น ในการทดสอบวีพีเอ็นนี้ ถือเป็นการทดสอบใหญ่ครั้งสุดท้าย โดยเป็นการทดสอบการรวมส่วนต่าง ๆ ของระบบ และเป็นการทดสอบการทำงานร่วมกัน รวมถึงเป็นการทดสอบการใช้งานจริง โดยหลังจากที่ระบบสามารถรันได้สมบูรณ์โดยไม่มีความผิดพลาดภายในตัวระบบเองแล้ว เราก็จะทดสอบครั้งสุดท้าย โดยใช้การสนิฟ (Sniff) ข้อมูลที่ติดต่อผ่านการเทลเน็ต (Telnet) ซึ่งเป็นข้อมูลแบบข้อความล้วน ๆ ที่สามารถอ่านเข้าใจได้บางส่วน ถ้าเราเปิดการใช้งานดีไวซ์ไดรเวอร์ทั้ง 2 ฝั่งและ 2 เวอร์ชั่น จากทั้ง 2 โปรแกรมวีพีเอ็นในเกตเวย์ทั้ง 2 ตัวแล้ว เราจะสามารถทำการปกปิดข้อมูลได้ ซึ่งถือว่า ระบบสามารถทำงานได้สมบูรณ์ในระดับของชุดต้นแบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 8

การทดลองและผลการทดลอง

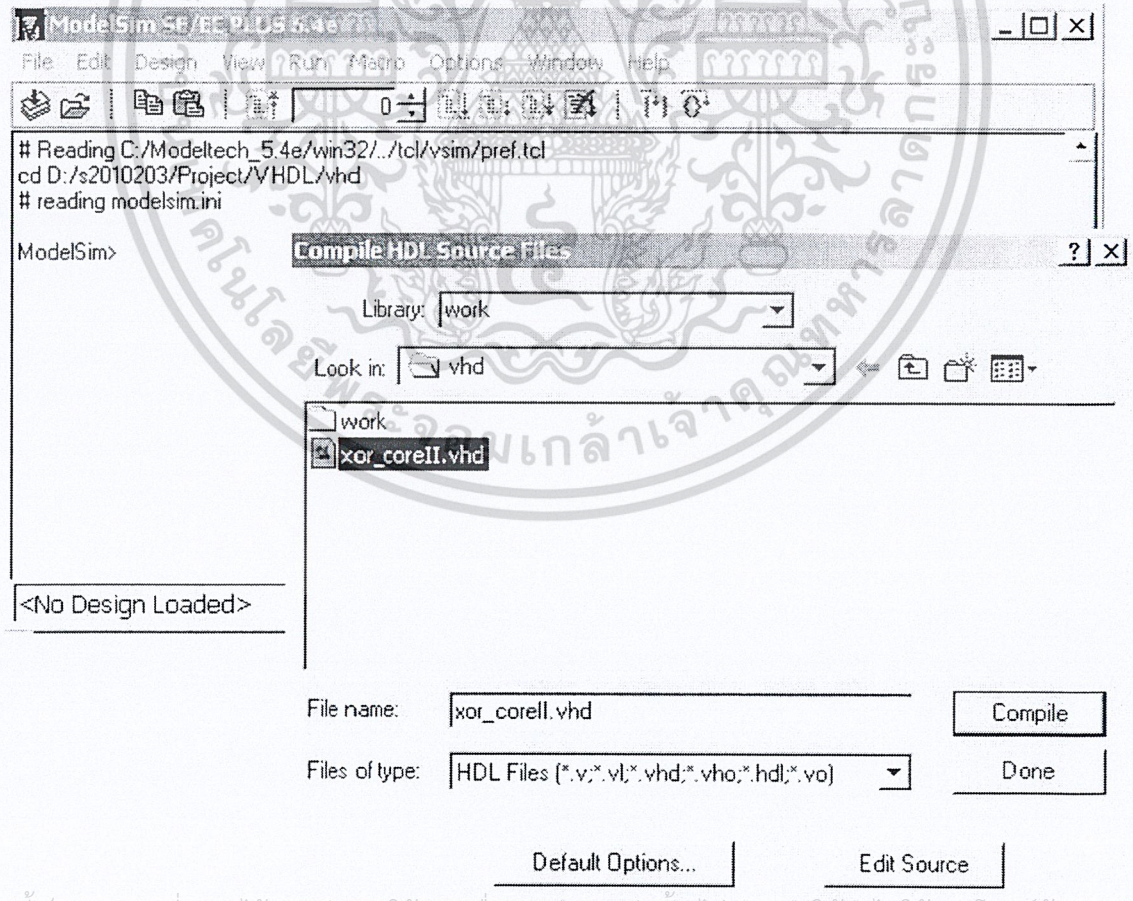
โครงงานชิ้นนี้ เป็นการพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส ซึ่งหลังจากที่ได้มีการออกแบบและพัฒนาโปรแกรมและตัวต้นแบบแล้ว จำต้องมีการทดสอบการทำงานของโปรแกรมและการทำงานของตัวต้นแบบ

8.1 การตรวจสอบความถูกต้องของโปรแกรม

นำโปรแกรมที่เขียนเสร็จแล้ว มาทำการตรวจสอบความสอบความถูกต้อง โดยการใช้โปรแกรม ModelSim, LeonardoSpectrum, Design Manager

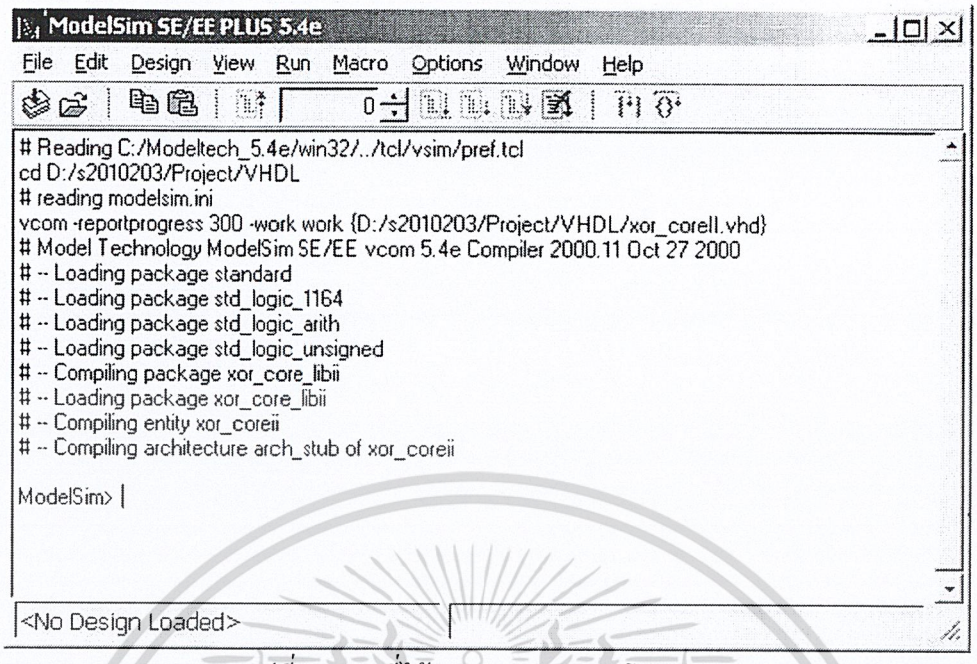
- โปรแกรม ModelSim

ใช้ตรวจสอบความถูกต้องของการเขียนภาษาวีเอชดีแอลและจำลองการทำงานของโปรแกรมเบื้องต้นได้



รูปที่ 8-1: การเลือกตัวโปรแกรมมาตรวจสอบความถูกต้อง

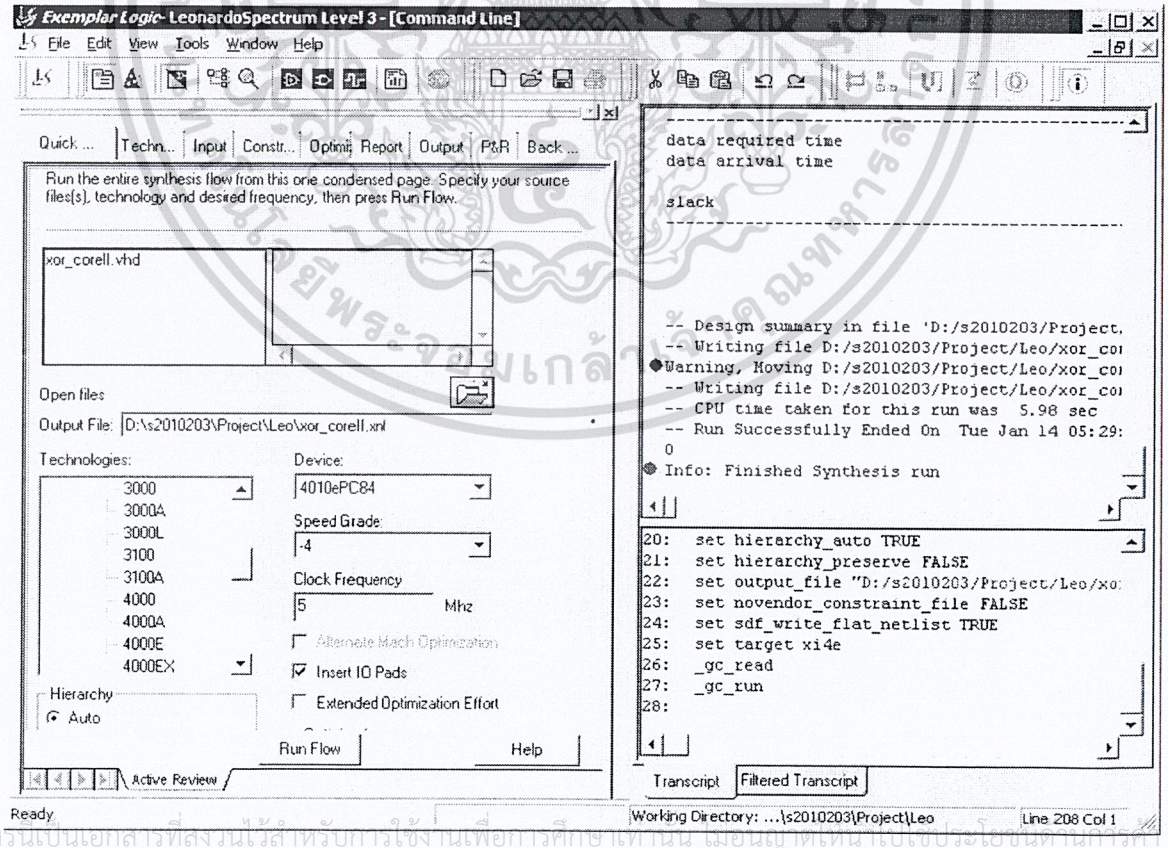
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8-2: ผลที่ได้จากการตรวจสอบโปรแกรม

- โปรแกรม LeonardoSpectrum

  ใช้ในการสังเคราะห์โปรแกรมภาษาวีเอชดีแอลให้ออกมาเป็นผังวงจร (Schematic)

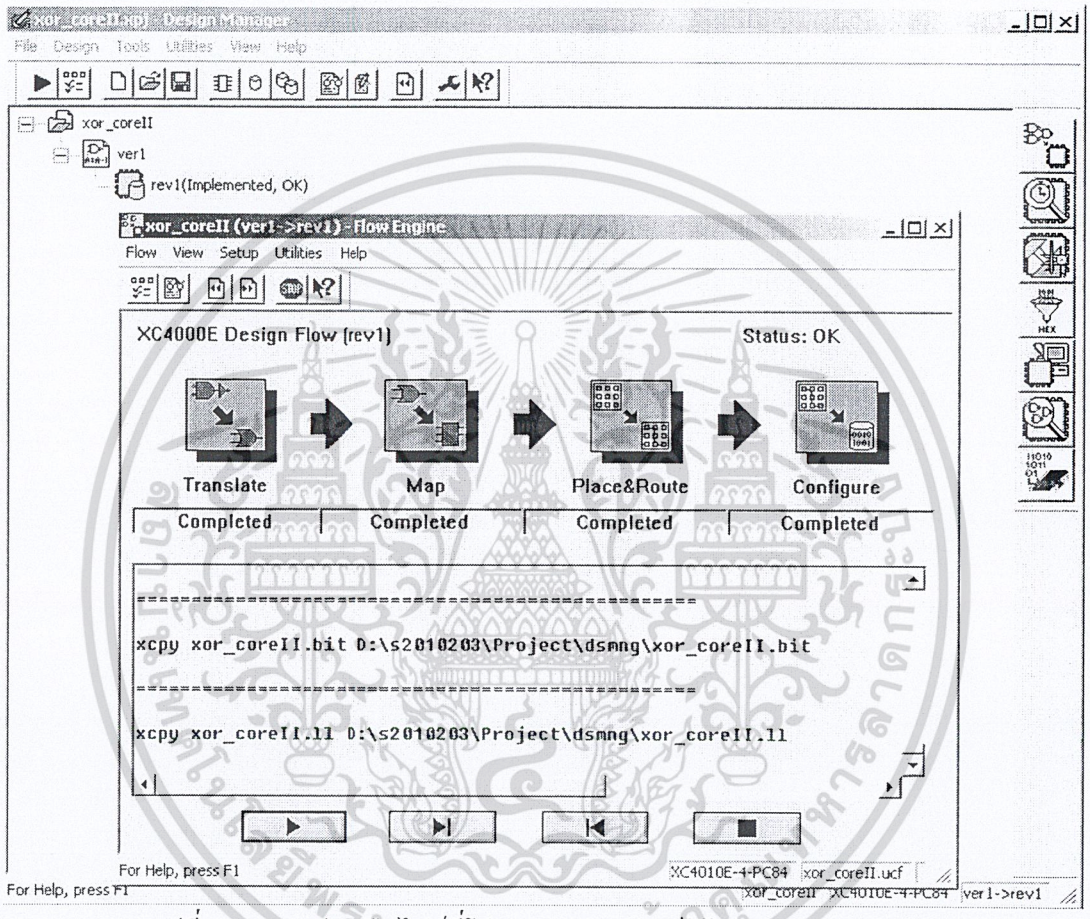


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 8-3: การสังเคราะห์โปรแกรมภาษาวีเอชดีแอลและผลของการสังเคราะห์โปรแกรม

- โปรแกรม Design Manager

ทำการแปลง Netlist file ที่ได้จากการสังเคราะห์ไปเป็นไฟล์รูปแบบต่าง ๆ ที่สามารถโปรแกรมลงอุปกรณ์เอฟพีจีเอได้



รูปที่ 8-4: การแปลงเป็นไฟล์ที่โปรแกรมลงอุปกรณ์ได้และผลของการแปลง

8.2 การโหลดโปรแกรมลงบอร์ด

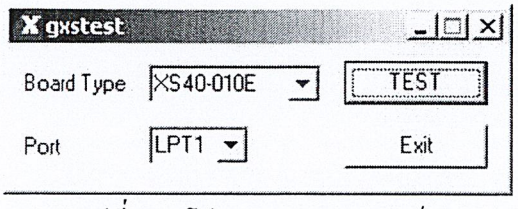
การโหลดโปรแกรมลงบนบอร์ดใช้โปรแกรม XSTOOL โดยไฟล์ที่ทำหน้าที่ในการแม็พขาสัญญาณ (.UCF) นั้น (ใช้ในโปรแกรม Design Manager) ได้นำข้อมูลมาจากคู่มือการใช้ของบอร์ดเอฟพีจีเอ XS40 เวอร์ชัน 1.4 มาประกอบการเขียน

- โปรแกรม XSTOOL

GXSTEST

เป็นโปรแกรมในการทดสอบบอร์ดดูว่า ใช้งานได้หรือไม่ โดยเลือกชนิดของบอร์ดเอฟพีจีเอและพอร์ตที่ใช้งานให้ถูกต้อง ซึ่งถ้าในขณะที่ทดสอบบอร์ดอยู่ ไฟของ Seven-Segment บนบอร์ดวิ่งวนไปเรื่อย ๆ นั้นหมายถึง บอร์ดสามารถใช้งานได้

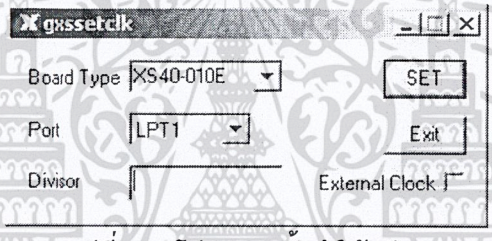
เอกสารนี้เป็นลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย ห้ามมิให้ทำซ้ำ ดัดแปลง หรือนำไปใช้ในการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น โดยไม่ได้รับอนุญาตจากเจ้าของลิขสิทธิ์ ยกเว้นการใช้งานเพื่อการศึกษาเท่านั้น ต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



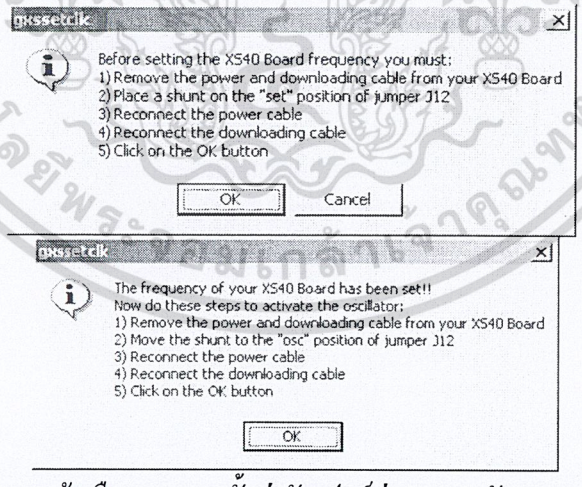
รูปที่ 8-5: โปรแกรมทดสอบบอร์ด

GXSSETCLK

เป็นโปรแกรมในการเซตคล็อก (Clock) ให้แก่ตัวบอร์ด เพื่อให้บอร์ดทำงานเร็วขึ้น หรือ ช้าลง โดยต้องเลือกชนิดของบอร์ดเอฟพีจีเอที่ใช้ พอร์ตที่ใช้ และทำการใส่ค่าของคล็อกที่ช่อง Divisor ให้ถูกต้อง เมื่อทำกดที่ปุ่ม SET แล้ว ตัวโปรแกรมจะแจ้งเตือนถึงการตั้งค่าจัมเปอร์ () ใหม่เพื่อการเซตคล็อกและเมื่อเซตคล็อกเสร็จแล้ว โปรแกรมก็จะแจ้งเตือนให้ตั้งค่าจัมเปอร์กลับอีกที ดังรูปที่



รูปที่ 8-6: โปรแกรมตั้งค่าให้คล็อก



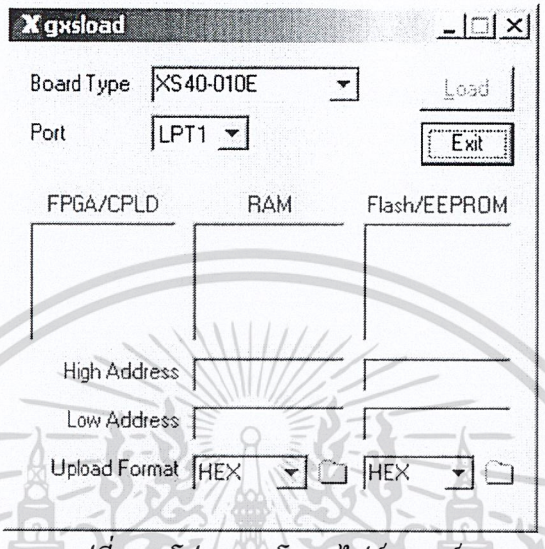
รูปที่ 8-7: การแจ้งเตือนของการตั้งค่าจัมเปอร์ก่อนและหลังจากการเซตคล็อก

GXSLOAD

เป็นโปรแกรมในการโหลดตัวโปรแกรมการเข้าและถอดรหัสกับโปรแกรมการทำงานของสตับลงไปในบอร์ด โดยต้องเลือกบอร์ดเอฟพีจีเอที่ใช้ พอร์ตที่ใช้ และทำการโหลดไฟล์ที่เป็น .BIT ลงไปที่ช่อง FPGA/CPLD ให้ถูกต้อง เมื่อกดปุ่ม Load แล้ว ตัวโปรแกรมจะทำการโหลดไฟล์ลงไปในบอร์ด ซึ่งเมื่อ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหลดไฟล์ลงไปในบอร์ดเรียบร้อยแล้ว ไฟตรงส่วนของ Seven-Segment จะเปลี่ยนไป โดยอาจจะเปลี่ยนไปตามโปรแกรมที่มีการเซต Seven-Segment ไว้ หรือไฟติดที่ตำแหน่ง E ของ Seven-Segment ถ้าโปรแกรมไม่มีการเซตค่าไว้



รูปที่ 8-8: โปรแกรมโหลดไฟล์ลงบอร์ด

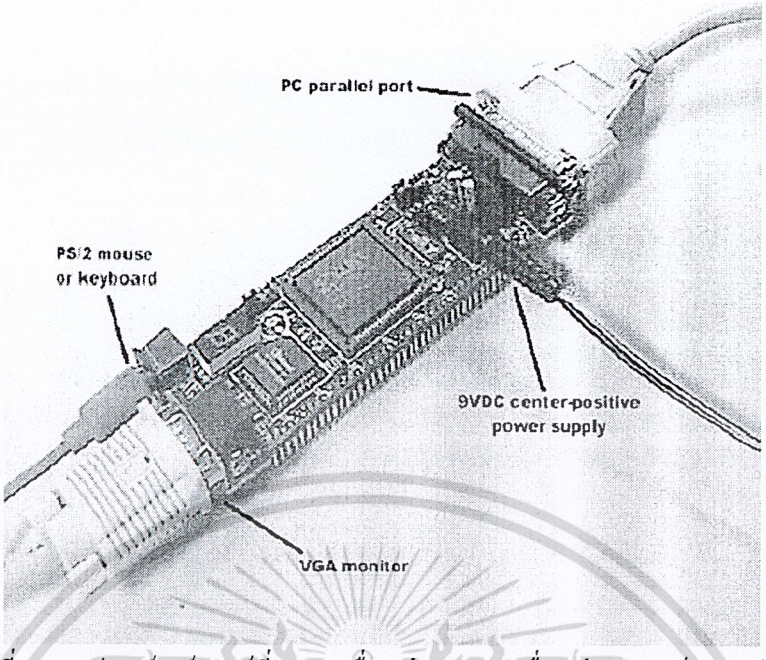
GXSPORT

เป็นโปรแกรมในการทดสอบการทำงานของบอร์ดในกรณีที่ต้องใช้พอร์ต (port) ในการทดสอบโปรแกรม (ในส่วนของโปรแกรมการเข้าและถอดรหัสนี้ ไม่มีส่วนเกี่ยวข้องกับการใช้งานพอร์ต)

ซึ่งในส่วนการโหลดโปรแกรมลงบอร์ดนั้น ยังจำเป็นต้องใช้อุปกรณ์ฮาร์ดแวร์ตัวอื่นมาประกอบการทำงานด้วย คือ

- สาย Parallel และ
- หม้อแปลง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8-9: อุปกรณ์ฮาร์ดแวร์ที่ทำการเชื่อมต่อและการเชื่อมต่อกับบอร์ดเอฟพีจีเอ

การทำงานของต้นแบบการใช้งานชิปเข้าและถอดรหัส จะเป็นการทำงานร่วมกันระหว่างบอร์ดเอฟพีจีเอที่มีการโหลดตัวโปรแกรมการเข้าและถอดรหัสไว้กับการ์ดไอเอสเอที่มีการต่อวงจรอย่างง่าย ๆ ไว้ จึงจำเป็นต้องมีการเชื่อมต่อขาสัญญาณของบอร์ดเอฟพีจีเอกับขาสัญญาณของการ์ดไอเอสเอเข้าด้วยกัน เนื่องจากเป็นการทำงานที่ต้องทำประสานกันของหลายส่วน ดังนั้น จึงต้องมีการตรวจสอบการทำงานของโปรแกรมการเข้าและถอดรหัส บอร์ดเอฟพีจีเอและการ์ดไอเอสเอ

8.3 การตรวจสอบการทำงานของโปรแกรม บอร์ดเอฟพีจีเอ และการ์ดไอเอสเอ

การตรวจสอบการทำงานนี้ ได้เขียนโปรแกรมขึ้นมาเพื่อเป็นตัวตรวจสอบ ซึ่งเป็นการเขียนด้วยภาษาแอสแซมบลี โดยตัวโปรแกรมทดสอบการทพำงานมีหน้าที่ส่งข้อมูลออกไปยังโปรแกรมการเข้ารหัสแล้วรับข้อมูลที่ได้จากการเข้ารหัสเข้ามาเก็บไว้ หลังจากนั้นจึงส่งออกไปถอดรหัส แล้วนำค่าที่ได้จากการถอดรหัสมาเทียบกับข้อมูลเริ่มต้นเพื่อตรวจสอบความถูกต้อง

โปรแกรมทดสอบการทำงานนั้น เป็นไฟล์ที่ชื่อว่า “testI.asm” โดยนำไฟล์นี้ไปทำการคอมไพล์ด้วยโปรแกรม TASM ซึ่งจะได้ออกมาเป็นไฟล์ “testI.exe”

```
C:\> tasm /zi testI.asm
C:\> tlink /v testI
C:\> testI
```

แต่เพื่อความรวดเร็วในการรันไฟล์แอสแซมบลี จึงได้ทำการเขียนไฟล์ .BAT ขึ้น โดยเก็บในส่วนของการคอมไพล์และส่วนของการรันโปรแกรม EXE เอาไว้ โดยตัวโปรแกรมทดสอบ จะทำการแสดงค่าของข้อมูลออกทางหน้าจอ เพื่อดูค่าที่ได้จากการทำงานของการเข้าและถอดรหัส

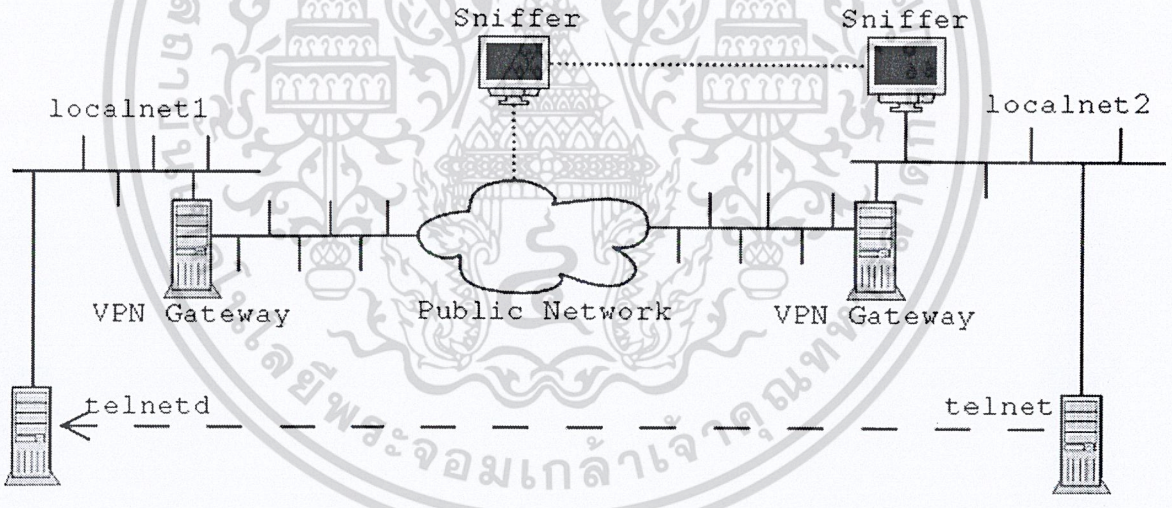
เอกสารนี้เป็นลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย ห้ามมิให้ทำซ้ำ ดัดแปลง หรือเผยแพร่ในรูปแบบใดๆ ไม่ว่ากรณีใดๆ เว้นแต่ได้รับอนุญาตจากจุฬาลงกรณ์มหาวิทยาลัย ผู้ใช้งานสามารถใช้งานเอกสารทุกครั้งที่มีการนำไปใช้

หากผลการทำงานที่ได้จากการตรวจสอบไม่ถูกต้อง จะเกิดจากเหตุผิดพลาดอยู่ 3 ประการ คือ

1. โปรแกรมการเข้าและถอดรหัส และ/หรือ
2. อุปกรณ์ฮาร์ดแวร์ คือ บอร์ดเอฟพีจีเอและการ์ดไอเอสเอ และ/หรือ
3. ตัวโปรแกรมการทดสอบเอง

อีกส่วนหนึ่งที่ไม่ใช่ส่วนของอุปกรณ์ฮาร์ดแวร์ ได้ทำการลงโปรแกรมไดรเวอร์ที่เขียนขึ้นลงในเคอร์เนล (ระบบปฏิบัติการลีนุกซ์) และทำการ Installation ส่วนของ VPN ซึ่งได้ทำเป็นแพ็กเกจ (Package) ไว้ลงไปในเครื่อง จากนั้น ก็ทำการทดสอบการทำงานของตัวต้นแบบทั้งระบบ

8.4 การทดสอบการทำงานของตัวต้นแบบ

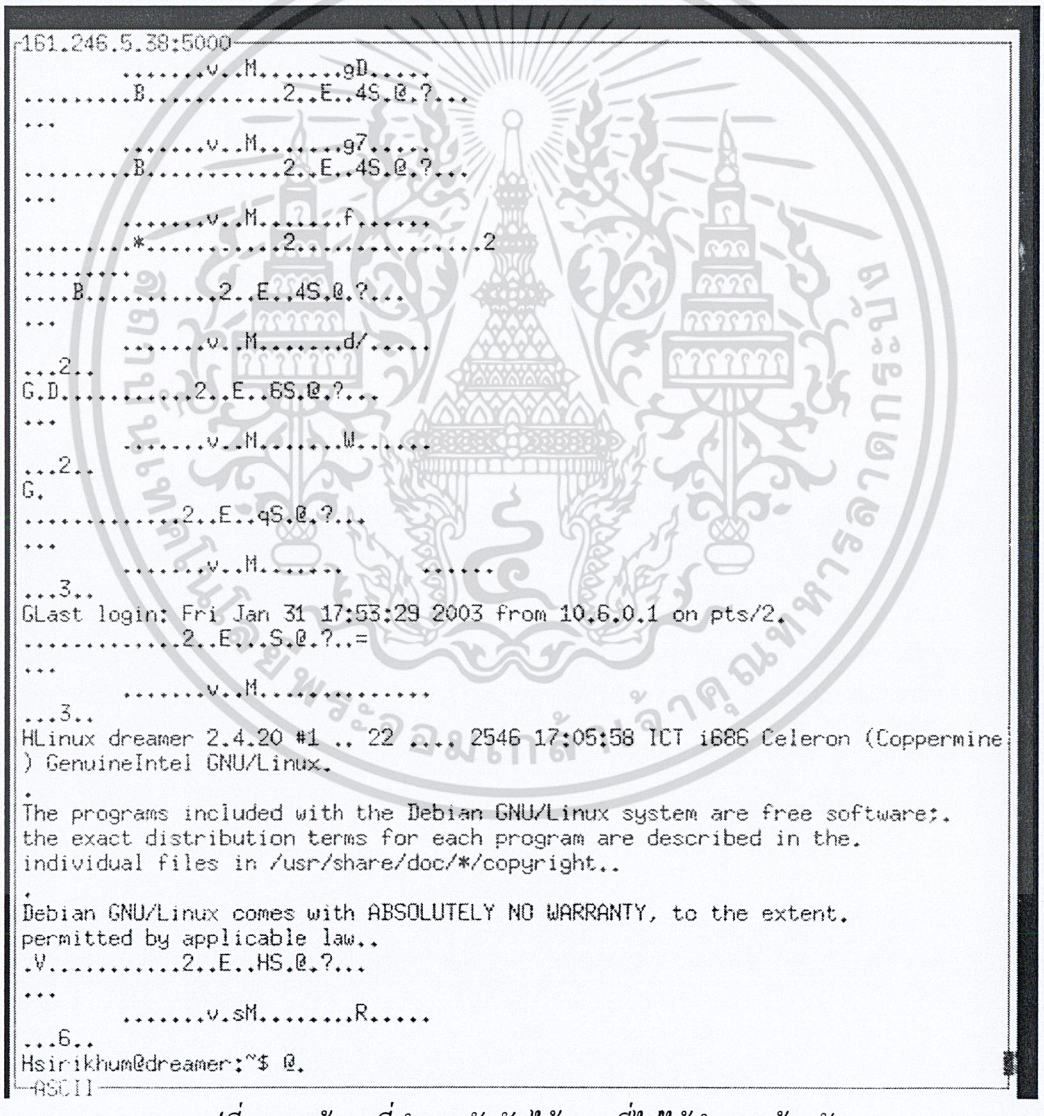


รูปที่ 8-10: การตรวจสอบการทำงานของตัวต้นแบบ

การทดสอบการทำงาน จะเป็นการรับส่งข้อมูลระหว่าง 2 เครือข่าย คือ localnet1 และ localnet2 โดยเครื่องทางฝั่ง localnet2 จะทำการ telnet ไปยังเครื่องที่อยู่ทางฝั่ง localnet1 โดยอีเทอร์เน็ตแพ็กเกตข้อมูลจะถูกส่งออกทาง VPN Gateway ของ localnet2 ไปยัง VPN Gateway ของ localnet1 ซึ่งอุปกรณ์ฮาร์ดแวร์ต้นแบบจะถูกฝังอยู่ใน VPN Gateway ฝั่ง localnet2 ส่วน VPN Gateway ทาง localnet1 จะเป็นการทำงานแบบไม่ใช้การ์ดไอเอสเอ แต่มีความสามารถทำงานได้เหมือนดังฝั่งที่ใช้การ์ด

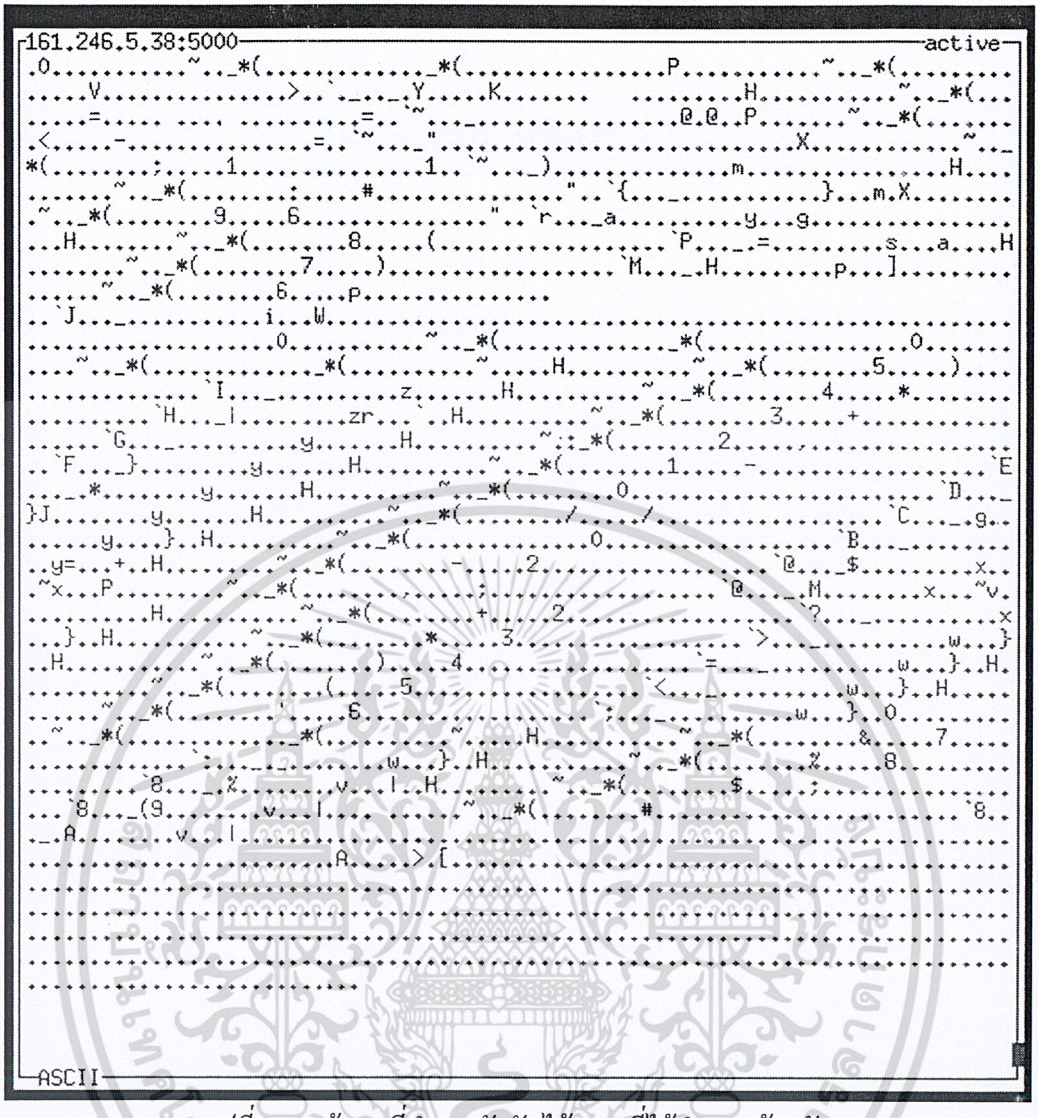
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในการส่งแพ็กเกตข้อมูลออกจากเครือข่ายผ่านทาง Public Network นั้น แพ็กเกตข้อมูลจะถูกทำการจัดรูปแบบให้เป็นอีเทอร์เน็ตเฟรม และตัวต้นแบบที่ VPN Gateway ของ localnet2 จะทำการเข้ารหัสทั้งอีเทอร์เน็ตเฟรมแล้วจึงจะส่งออกนอกเครือข่ายโดยใช้โพรโตคอลทีซีพี/ไอพี เมื่อแพ็กเกตที่ถูกเข้ารหัสนั้นถูกส่งมายัง localnet1 ตัว VPN Gateway ของทางฝั่ง localnet1 จะทำการถอดรหัสแพ็กเกตที่ส่งมา ซึ่งจะได้ออกมาเป็นอีเทอร์เน็ตเฟรมตามเดิมและทำการถอดเฟรมอีเทอร์เน็ตออก ซึ่งจะได้ออกมาเป็นแพ็กเกตข้อมูลธรรมดา หลังจากนั้นจะทำการส่งแพ็กเกตข้อมูลนั้นไปให้ยังเครื่องปลายทาง ซึ่งถ้าหากมีผู้ดักจับข้อมูลในระหว่างที่กำลังเดินทางอยู่ใน Public Network ไป ก็จะไม่สามารถตีความหมายของข้อมูลที่ส่งไปได้ โดยเราได้ทำทดลองทำการดักจับข้อมูลในระหว่างทางแล้ว ผลที่ได้จากการดักจับเป็นไปตามที่คาดการณ์ไว้ คือ ไม่สามารถทำการตีความหมายได้



รูปที่ 8-11: ข้อมูลที่ทำการดักจับได้ แบบที่ไม่ได้ทำการเข้ารหัส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8-12: ข้อมูลที่ทำการดักจับได้ แบบที่ได้ทำการเข้ารหัส

ดังนั้น เป็นการพิสูจน์ได้ว่า อุปกรณ์ฮาร์ดแวร์ต้นแบบการใช้งานชิปเข้าและถอดรหัสสามารถทำงานได้จริง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 9
บทวิจารณ์และสรุปผล

โครงงานชิ้นนี้ เป็นการพัฒนาต้นแบบประยุกต์ชิปเข้าและถอดรหัส ซึ่งเป็นการนำอุปกรณ์ฮาร์ดแวร์มาทำการเข้าและถอดรหัส โดยสามารถทำงานข้ามเครือข่ายได้ด้วยระบบวีพีเอ็น (VPN) และทำการเขียนดีไวซ์ไดรเวอร์ขึ้น เพื่อช่วยให้ตัวฮาร์ดแวร์ทำงานได้อย่างมีประสิทธิภาพมากขึ้น

9.1 การวิจารณ์โครงงาน

เนื่องจากโครงงานนี้ เป็นต้นแบบในการใช้งานชิปเข้าและถอดรหัสข้อมูล จึงเป็นการทำงานที่ใหม่หมด ซึ่งต้องทำการศึกษาทั้งหมด

9.1.1 ส่วนที่เกี่ยวข้องกับฮาร์ดแวร์

ในส่วนของการพัฒนาในด้านฮาร์ดแวร์นั้น ต้องใช้ความรู้และความเข้าใจอยู่หลายส่วน ทั้งเรื่องของตัวบอร์ดเอฟพีจีเอ การ์ดไอเอสเอ และส่วนของโปรแกรมที่ใช้โหลดลงบอร์ด ซึ่งการพัฒนาในด้านฮาร์ดแวร์นั้น มีความสลับซับซ้อนและยุ่งยาก

- บอร์ดเอฟพีจีเอที่ทำหน้าที่เข้ารหัส

ในด้านของการเขียนโปรแกรมภาษาวีเอชดีแอล (VHDL) ที่ทำหน้าที่ในการเข้าและถอดรหัสข้อมูล ได้เกิดปัญหาที่เหนือการแก้ไขได้อยู่ คือ ภาษาวีแอชดีแอลที่ยังมีข้อจำกัดบางอย่างซ่อนอยู่ เช่น ข้อจำกัดในเรื่องของไอโอ (I/O) เรื่องคล็อก (Clock) เรื่องขนาดของ H-Function ที่มีจำนวนน้อย และเรื่องอื่น ๆ และอัลกอริทึม DES ที่กำหนดไว้ตอนแรกนั้น ตัวโปรแกรมมีขนาดใหญ่เกินที่จะโหลดลงบอร์ดเอฟพีจีเอได้ จึงต้องทำการเปลี่ยนอัลกอริทึมการเข้าและถอดรหัส

- การ์ดไอเอสเอ

ในส่วนของการ์ดไอเอสเอที่ทำหน้าที่เชื่อมการทำงานระหว่างเครื่องคอมพิวเตอร์กับตัวบอร์ดเอฟพีจีเอ ยังคงมีปัญหาในเรื่องการทำงานของไอซีบางตัว ที่ทำงานได้ไม่ทันกับการทำงานของบอร์ดเอฟพีจีเอ และปัญหาในด้านการดีบัก (Debug)

9.1.2 ส่วนที่เกี่ยวข้องกับซอฟต์แวร์

ในส่วนของการพัฒนาซอฟต์แวร์ คือ ส่วนที่ทำการเพิ่มเข้ามาเพื่อทำให้การทำงานของอุปกรณ์ต้นแบบทำงานได้อย่างมีประสิทธิภาพและสะดวกสบายมากขึ้น

- ดีไวซ์ไดรเวอร์

ปัญหาที่เกิดขึ้นในการเขียนดีไวซ์ไดรเวอร์ คือ ความเข้าใจในด้านการทำงานของไดรเวอร์และการออกแบบโปรแกรมของไดรเวอร์ เนื่องจากการเขียนดีไวซ์ไดรเวอร์นั้น สลับซับซ้อนและยากที่จะเข้าใจได้ง่าย

- วีพีเอ็น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เป็นปัญหาในด้านการรับและส่งข้อมูล

9.2 สรุปผลของโครงงาน

โครงงานนี้ เป็นต้นแบบประยุกต์การใช้งานชิปเข้าและถอดรหัส และจากที่ได้ทำการออกแบบและการทดลองแล้ว ผลที่ได้ออกมา คือ ต้นแบบสามารถทำงานได้จริง ทั้งการเข้าและถอดรหัสข้อมูลและการรับ-ส่งข้อมูลระหว่างเครือข่ายที่เมื่อมีการดักจับข้อมูลระหว่างที่ส่งแล้ว ผู้ดักจับไม่สามารถทำการตีความหมายข้อมูลนั้น ๆ ได้

9.3 แนวทางการพัฒนาต่อไปในอนาคต

- เปลี่ยนแปลงในด้านของอัลกอริทึมที่นำมาใช้ในการเข้าและถอดรหัสจาก XOR ธรรมดาให้เป็นอัลกอริทึมอื่นที่มีความสลับซับซ้อนมาขึ้นกว่า เพื่อเพิ่มประสิทธิภาพความปลอดภัยให้แก่ข้อมูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บรรณานุกรม

[1] คู่มือเทียบเบอร์ไอซี TTL, SE-Education

[2] สันติ ศรีลาศักดิ์ และ เกศมณี เที่ยงธรรม, เปิดโลกลีนุกซ์ดีไวซ์ไดรเวอร์ kernel series, SE-Education

[3] เอกสารประกอบการทดลอง วิชา 01073007 การออกแบบฮาร์ดแวร์คอมพิวเตอร์

[4] ธีรา ทานตวณิช, Virtual Private Network: รู้จักให้ดีขึ้นเพื่อการทำงานที่ปลอดภัยกว่า

[5] ดร. วิรินทร์ เมฆประดิษฐสิน, คม-ชัด-ลึกกับเครือข่าย VPN ตอนจบ

เว็บไซต์อ้างอิง

[1] http://www.openssl.org/

[2] http://vtun.sourceforge.net/

[3] http://www.xml.com/ldd/chapter/book/

[4] http://www.uni-tuebingen.de/zdv/zriinfo/linux/books/khg/chapter2_3.html

[5] http://invalid.ed.ntnu.no/~kamben/hwb/co_ISA_Tech.html

[6] http://fossil.wpi.edu/docs/howto_module_dev_driver.html

[7] http://xilinx.com

[8] http://www.google.co.th

[9] http://csrc.nist.gov/fips

[10] http://www.uni-tuebingen.de/zdv/zriinfo/linux/books/khg/chapter2_3.html

[11] http://www.vardaan.com/writing_device_driver_made_easy.htm#wiadd

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้