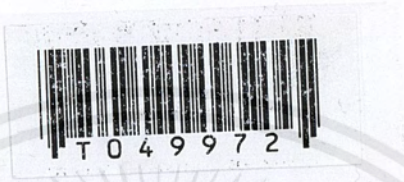
ลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

Embedded Linux on Gameboy Advance



นาย อภิชาติ ปู่มี

นาย อิทธิพล แย้มบุญยิ่ง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

Embedded Linux on Gameboy Advance

โดย

นาย อภิชาติ ปู่มี

นาย อิทธิพล แย้มบุญยิ่ง

อาจารย์ที่ปรึกษา

ผศ. อภิเนตร อูนากูล

อ. ดุสิต นิยะโต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

Embedded Linux on Gameboy Advance

ผู้จัดทำ

1. นาย อภิชาติ ปู่มี รหัสประจำตัว 42010434
2. นาย อิทธิพล แย้มบุญยิ่ง รหัสประจำตัว 42010446

_____ อาจารย์ที่ปรึกษา

(ผศ. อภิเนตร อูนากูล)

_____ อาจารย์ที่ปรึกษา

(อ. ดุสิต นิยะโต)

ลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

นายอภิชาติ ปู่มี 42010434
นายอิทธิพล แย้มบุญยิ่ง 42010446
ผศ.อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
อ.ดุสิต นิยะโต อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

ถ้าเรากล่าวถึงเครื่องเกมบอยแอดวานซ์แล้ว จากโครงสร้างที่มีความทนทาน และความสามารถที่มีประสิทธิภาพมากกว่าที่จะนำมาเพียงเล่นเกม ไม่ว่าจะเป็นซีพียูที่มีขนาด32บิต หน้าจอแอลซีดีสีที่มีขนาดใหญ่ และยังมีคีย์แพดไว้อินพุตถึง10ปุ่มด้วยกัน ทั้งนี้ยังเคลื่อนย้ายง่าย และทนทานเนื่องจากได้ถูกออกแบบมาให้เด็กเล่นเกมโดยเฉพาะ

ซึ่งในปัจจุบันโรงงานอุตสาหกรรมจะมีอุปกรณ์ต่างๆที่ใช้เทคโนโลยีแบบฝังตัวทำให้สามารถทำงานได้โดยอัตโนมัติ แต่การจัดการต่างๆกับระบบฝังตัวเหล่านี้ไม่ว่าจะเป็นการตั้งค่าหรือตรวจสอบความถูกต้องของการทำงานนั้นจำเป็นต้องมีอุปกรณ์ที่มีความสามารถในการจัดการ เช่นเครื่องคอมพิวเตอร์เป็นต้น เนื่องด้วยระบบฝังตัวโดยส่วนมากนั้นไม่มีอินพุตเอาต์พุตในการจัดการ แต่ในความเป็นจริงนั้นระบบฝังตัวจะไม่ได้มีเพียงระบบเดียว จึงเป็นเรื่องยากลำบากในการที่จะนำเครื่องคอมพิวเตอร์มาจัดการ เราจึงได้เล็งเห็นถึงความสามารถของเครื่องเกมบอยแอดวานซ์ที่มีประสิทธิภาพและราคาถูกมาจัดการ

โครงงานนี้จึงนำเครื่องเกมบอยแอดวานซ์มาสร้างแอพพลิเคชันในการจัดการระบบฝังตัวที่ต้องการการปรับแต่งหรือการดูค่าต่างๆ แต่ในการที่จะสร้างแอพพลิเคชันที่จะมาทำงานบนเครื่องเกมบอยแอดวานซ์นั้นจะต้องมีการเข้าถึงฮาร์ดแวร์ของเครื่องเกมบอยแอดวานซ์ ซึ่งเป็นเรื่องที่ยุ่งยาก เราจึงจะนำระบบปฏิบัติการมาทำหน้าที่นี้ โดยเราจะสามารถสร้างแอพพลิเคชันได้โดยไม่ต้องจัดการเกี่ยวกับฮาร์ดแวร์เลย

วิทยานิพนธ์ฉบับนี้จึงจัดทำขึ้นเพื่อทำให้ระบบปฏิบัติการลีนุกซ์นั้นสามารถทำงานบนเครื่องเกมบอยแอดวานซ์ได้ และได้เสนอแนวทางการแก้ไขปัญหาด้วยการนำเครื่องเกมบอยแอดวานซ์ที่มีความเหมาะสมกับการใช้งานภายในโรงงานมาทำเป็นเครื่องมือเคลื่อนที่ที่สามารถใช้ในการกำหนดค่าคุณสมบัติของอุปกรณ์ต่างๆ

Embedded Linux on Gameboy Advance

Apichat	Poomee	42010434
Ittipol	Yamboonying	42010446
Asst. Prof. Apinetr	Unakul	Advisor
Dusit	Niyato	Advisor

Abstract

Gameboy Advance has a durable architecture because it's designed for children and it has high efficiency because of 32 bits CPU, LCD monitor, and many inputs. So the Gameboy Advance has ability more than just gaming console, it is appropriate to be used in the factory.

Nowadays, the most of factory have many embedded devices/machines with can operate automatically but in order to manage or configure those devices/machines, a device like computer is needed. In reality, it's hard to use computer to manage or configure all of those devices/machines. So the gameboy advance come into play.

In order to turn the gameboy advance to be a configurable mobile device for each device/machine, an individual program/application is needed. In reality, it's still difficult because the developer need the well know in the hardware of gameboy advance and how to use them. To make this easier, an operating system comes into play

This thesis presents a way to develop an operating system, Linux, for gameboy advance. So the developer can develop program/application for gameboy advance without the well know in the hardware of gameboy advance, and presents the gameboy advance as a configurable mobile device.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่สามารถสำเร็จลุล่วงไปได้ด้วยดีหากปราศจากความช่วยเหลือและสนับสนุนจากหลายๆบุคคลด้วยกัน บุคคลแรกที่จะต้องกล่าวถึงก่อนเพราะเป็นผู้มีส่วนสำคัญที่ทำให้วิทยานิพนธ์ฉบับนี้เสร็จลงได้คือ ผู้ช่วยศาสตราจารย์ อภิเนตร อูนากูล ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ให้ความเอาใจใส่อย่างมาก และแนะนำช่วยเหลือเสมอมาและไม่เพียงแต่แนะนำในเรื่องการทำงานเท่านั้น ท่านยังได้สอนการใช้ชีวิตในสังคมและมุมมองต่างๆ บุคคลต่อไปที่จะต้องกล่าวถึงให้ได้ก็คือ อาจารย์ ดุสิต นิยะโต อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเป็นกันเอง ให้คำปรึกษาเรื่องต่างๆ ทั้งในเรื่องของวิทยานิพนธ์ และเรื่องการใช้ชีวิตต่างๆ ซึ่งถ้าไม่มีอาจารย์ทั้งสองท่าน วิทยานิพนธ์ฉบับนี้คงจะไม่สำเร็จได้ด้วยดี ทางผู้จัดทำจึงขอขอบคุณท่านทั้งสองด้วยความเคารพเป็นอย่างยิ่ง

บุคคลผู้ที่จะขาดไม่ได้สำหรับความสำเร็จของวิทยานิพนธ์ฉบับนี้ก็คือ บริษัท NetGadgets โดยเฉพาะพี่พรเทพ นฤหล้า และพี่ภาค ทองกุล ผู้ซึ่งสนับสนุนในเรื่องเครื่องมืออุปกรณ์ในการทำวิทยานิพนธ์เรื่องนี้ขึ้น ซึ่งยังรวมถึงคำแนะนำต่างๆที่เป็นเป็นความรู้ แนวทางการจัดทำวิทยานิพนธ์ฉบับเป็นอย่างมากโดยไม่เคยปฏิเสธคำขอความเชื่อเหลือ ขอขอบคุณเพื่อนๆในห้อง ESL และเพื่อนๆนอกห้องวิจัยที่ให้ความช่วยเหลือตลอดมาในทุกๆเรื่อง และได้อยู่เป็นกำลังใจเสมอมา

และขอขอบคุณพระคุณของ บิดา มารดา ที่เคารพรักของผู้จัดทำวิทยานิพนธ์ฉบับนี้ที่คอยสนับสนุนให้ข้าพเจ้ามีวันนี้ สนับสนุนผู้จัดทำในด้านการศึกษา และดูแลเอาใจใส่โดยไม่เคยที่จะมีข้อบกพร่องใดๆ ข้าพเจ้าขอระลึกถึงคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

นายอภิชาติ ปู่มี

นายอิทธิพล แย้มบุญยิ่ง

สารบัญ

หน้าที่

หน้าที่

สารบัญรูปภาพ

หน้าที่

หน้าที่

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในอนาคตอุปกรณ์เครื่องใช้ต่างๆ ไม่ว่าจะเป็นทีวี ตู้เย็น รถยนต์ ฯลฯ ที่ถูกใช้งานเป็นประจำในชีวิตประจำวันจะมีความฉลาดอีกทั้งยังมีความสามารถเพิ่มมากขึ้นเนื่องจากมีการนำเทคโนโลยีระบบฝังตัว (Embedded System) มาใช้ประโยชน์มากขึ้น

ระบบฝังตัวดังกล่าวมีหลักการดังนี้คือ นำหน่วยประมวลผลขนาดเล็ก (Microprocessor) หรือ หน่วยควบคุมขนาดเล็ก (Micro-controller) มาใช้เป็นส่วนควบคุมส่วนต่างๆของอุปกรณ์ไม่ว่าจะเป็นการป้อนข้อมูล (Input) หรือการแสดงผลลัพธ์ (Output) ต่างๆ โดยการควบคุมส่วนต่างๆ นั้นขึ้นอยู่กับโปรแกรมที่ทำงานบนหน่วยประมวลผลหรือหน่วยควบคุมนั้นๆ ทำให้อุปกรณ์ที่ใช้เทคโนโลยีระบบฝังตัวนั้นมีความฉลาดและความสามารถเพิ่มมากขึ้น ยกตัวอย่างเช่น เครื่องซักผ้าในปัจจุบัน ที่เป็นแบบทำงานได้ด้วยตัวเองจะสามารถควบคุมระบบการซักผ้าทั้งหมดและสามารถตั้งค่าตัวแปรตามต่างๆ เช่น ค่าอุณหภูมิของน้ำ ขนาดของผงซักฟอกที่จะใช้ ความเร็วในการหมุนขณะซักล้าง และค่าเวลาต่างๆ ซึ่งในบางระบบสามารถพูดคุยสื่อสารได้ด้วย ทั้งนี้ค่าต่างๆ ที่ตั้งโดยตัวเครื่องซักผ้าจะขึ้นอยู่กับการเลือกชนิดของผ้า (ลินิน, คอตตอน, ใยสังเคราะห์, ฯลฯ) ของผู้ใช้

ในปัจจุบัน ระบบฝังตัวได้เข้ามามีบทบาทในการทำงานภายในโรงงานอุตสาหกรรมด้วยโดยเครื่องมือหรือเครื่องจักรภายในโรงงานอุตสาหกรรมต่างๆ จะนำเทคโนโลยีระบบฝังตัวมาใช้อย่างกว้างขวาง ทำให้เครื่องมือหรือเครื่องจักรเหล่านั้นมีความฉลาดและมีความสามารถเพิ่มมากขึ้น อย่างไรก็ตาม อุปกรณ์ที่ใช้เทคโนโลยีระบบฝังตัวนั้นจะสามารถทำงานได้จะต้องมีการควบคุมหรือกำหนดค่าคุณสมบัติก่อนการทำงาน ดังตัวอย่าง เครื่องซักผ้าที่ทำงานได้ด้วยตัวเอง จำเป็นต้องมีการควบคุมหรือกำหนดค่าคุณสมบัติของ ชนิดผ้า จึงจะทำให้เครื่องซักผ้าทำงานได้ด้วยตัวเอง

ด้วยการที่ต้องมีการควบคุมหรือกำหนดค่าคุณสมบัติก่อนการทำงานนั้น อุปกรณ์ต่างๆ จะต้องมีฮาร์ดแวร์เฉพาะในการทำหน้าที่รับข้อมูลป้อนเข้าและการแสดงผลลัพธ์ ดังในตัวอย่างเครื่องซักผ้าที่ทำงานได้ด้วยตนเอง จะมีปุ่มให้เลือก ชนิดผ้า เป็นฮาร์ดแวร์เฉพาะในการรับข้อมูลเข้า และอาจจะมีลำโพงหรือจอภาพผลึกเหลว (LCD) ในการแสดงผลลัพธ์ ซึ่งถ้ามีความจำเป็นต้องใช้อุปกรณ์หรือเครื่องมือที่ใช้เทคโนโลยีระบบฝังตัวจำนวนมากจะทำให้เกิดความสิ้นเปลืองในส่วนของฮาร์ดแวร์ที่ใช้รับข้อมูลเข้าและแสดงผลลัพธ์ ซึ่งเป็นฮาร์ดแวร์ที่มีราคาในตัวเอง แต่มีการใช้งานเพียงน้อยครั้ง ไม่คุ้มค่ากับราคาของฮาร์ดแวร์ จึงเกิดแนวทางแก้ไขปัญหาความสิ้นเปลืองเพื่อลดต้นทุนต่างๆ ได้ โดยแนวคิดที่ใช้ในการแก้ปัญหามีอยู่ 2 แนวคิดด้วยกัน แนวคิดแรก คือการสร้างหน่วยควบคุมกลาง (Control center) ขึ้นมาแล้วเชื่อมต่ออุปกรณ์ต่างๆ เข้ากับหน่วยควบคุมกลาง เป็นการทำให้อุปกรณ์ต่างๆ มีการเชื่อมต่อ (On-line) จึงทำให้สามารถควบคุมหรือกำหนดค่าคุณสมบัติต่างๆ ได้ที่หน่วยควบคุมกลางเพียงที่เดียว สามารถลดการสิ้นเปลืองดังกล่าวได้ แนวคิด

ที่สอง คือ การสร้างเครื่องมือที่สามารถเคลื่อนที่ และสามารถควบคุมหรือกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆได้

วิทยานิพนธ์ฉบับนี้จะนำเสนอแนวคิดการแก้ปัญหาดังกล่าว ด้วยการสร้างเครื่องมือที่สามารถเคลื่อนที่ และสามารถควบคุมหรือกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ โดยการนำเครื่องเกมบอยแอดวานซ์ ที่มีทนทานสูงเพราะถูกออกแบบมาสำหรับเด็ก อีกทั้งยังมีความสามารถสูงคือ ใช้ หน่วยประมวลผล (CPU) ขนาด 32 บิต และมีสายเชื่อมต่อสามารถส่งข้อมูลและรับข้อมูลจากภายนอกได้ มาทำเป็นเครื่องมือเคลื่อนที่ที่ใช้ในการควบคุมหรือกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ ในการที่จะนำเครื่องเกมบอยแอดวานซ์มาทำงานดังกล่าวนั้นจำเป็นต้องมีโปรแกรมที่ทำงานในการควบคุมหรือกำหนดค่าคุณสมบัติต่างๆ ของอุปกรณ์ซึ่งจำเป็นที่จะต้องมีโปรแกรมสำหรับแต่ละอุปกรณ์ที่แตกต่างกัน ดังนั้นเพื่อความยืดหยุ่นในการใช้งานและเพื่อความไม่ยุ่งยากสำหรับการเขียนโปรแกรมจึงใช้ประโยชน์ของระบบปฏิบัติการในการจัดการในด้านฮาร์ดแวร์ต่างๆ ของเครื่องเกมบอยแอดวานซ์ โดยระบบปฏิบัติการที่มีความเหมาะสมกับเครื่องเกมบอยแอดวานซ์นั้นคือไมโครซีลีนุกซ์ หรือ ยูซีลีนุกซ์ (uClinux) และ อีคอส (eCos) เนื่องจากเป็นระบบปฏิบัติที่ไม่ต้องเสียค่าใช้จ่ายในการนำมาใช้อีกทั้งลักษณะเฉพาะตัวของ ไมโครซีลีนุกซ์ และ อีคอส นั้นมีความเหมาะสมกับเครื่องเกมบอยแอดวานซ์อย่างมาก ซึ่งจะกล่าวถึงต่อไป

จากที่ได้กล่าวมาทั้งหมด ผู้อ่านคงจะเห็นถึงความสิ้นเปลืองที่เกิดจากการติดตั้งฮาร์ดแวร์ที่มีการใช้งานเพียงน้อยครั้งในอุปกรณ์ที่มีความฉลาด และความสำคัญในการแก้ปัญหาดังกล่าวเพื่อลดต้นทุนต่างๆได้

1.2 วัตถุประสงค์ของงานวิจัย

1. ศึกษาความเป็นไปได้ในการนำระบบปฏิบัติการ ไมโครซีลีนุกซ์ มาทำงานบนเครื่องเกมบอยแอดวานซ์
2. ศึกษาวิธีการสร้างโปรแกรมบนเครื่องเกมบอยแอดวานซ์โดยศึกษาทั้งวิธีการสร้างโปรแกรมโดยตรงสู่เครื่องเกมบอยแอดวานซ์ และวิธีการสร้างโปรแกรมบนระบบปฏิบัติการ เพื่อนำไปใช้ในการควบคุมอุปกรณ์ต่างๆ
3. เพื่อนำเสนอแนวคิดและความเป็นไปได้ในการนำเครื่องเกมบอยแอดวานซ์ไปใช้ในการแก้ปัญหาความสิ้นเปลืองที่เกิดขึ้น

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะพยายามนำระบบปฏิบัติการ ไมโครซีลีนุกซ์ มาทำงานบนเครื่องเกมบอยแอดวานซ์โดยทำการถ่ายโอน (Load) ข้อมูลของ ไมโครซีลีนุกซ์ จากเครือข่ายอินเตอร์เน็ต (Internet) มาแก้ไขให้สามารถทำงานได้บนเครื่องเกมบอยแอดวานซ์

ทั้งนี้โครงการนี้ยังถือว่าเป็นโครงการทดลอง นำระบบปฏิบัติการ ไมโครซีลีนุกซ์และระบบปฏิบัติการอีคอส มาทำงานบนเครื่องเกมบอยแอดวานซ์ เพื่อศึกษาความเป็นไปได้ในการนำมาใช้งาน

1.4 วิธีการดำเนินงาน

งานวิจัยในโครงการนี้จะเริ่มด้วยการศึกษาลักษณะต่างๆของเครื่องเกมบอยแอดวานซ์โดยจะศึกษาในด้านฮาร์ดแวร์ ความสามารถ และข้อจำกัดต่างๆ ของเครื่องเกมบอยแอดวานซ์ จากนั้นศึกษาโครงสร้างต่างๆ ของระบบปฏิบัติการ ไมโครซีลีนุกซ์ แล้วตั้งสมมติฐานการนำมาทำงานจริงบนเครื่องเกมบอยแอดวานซ์โดยอาศัยข้อมูลที่ได้ศึกษามาก่อนหน้านี้ รวมไปถึงศึกษาความเสี่ยงต่างๆ ที่จะทำให้ระบบปฏิบัติการดังกล่าวไม่สามารถทำงานได้จริงบนเครื่องเกมบอยแอดวานซ์ จากนั้นทำการตั้งข้อกำหนดในการทำงานที่จะทำให้เครื่องเกมบอยแอดวานซ์กลายเป็นอุปกรณ์ที่สามารถใช้ในการควบคุมหรือกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆได้ โดยกำหนดข้อกำหนดแรก คือ การสร้างชุดพัฒนาโปรแกรมบนเครื่องเกมบอยแอดวานซ์แบบไม่พึ่งพาระบบปฏิบัติการ ข้อกำหนดที่สอง คือ การสร้างชุดพัฒนาโปรแกรมบนเครื่องเกมบอยแอดวานซ์แบบพึ่งพาระบบปฏิบัติการ ซึ่งภายในข้อกำหนดที่สองนั้นจะมีข้อกำหนดย่อยอีก 2 ข้อกำหนดแยกตามระบบปฏิบัติการ อันได้แก่ ระบบปฏิบัติการ อีคอส (eCos: embedded Configurable operating system) และ ไมโครซีลีนุกซ์ แล้วทำงานตามข้อกำหนดที่ได้กำหนดขึ้นมา ซึ่งรายละเอียด จะอยู่ภายในบทต่างๆ ภายในวิทยานิพนธ์ฉบับนี้

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ระบบ Embedded

ระบบ embedded คืออุปกรณ์อะไรก็ตามที่มีไมโครชิปฝังอยู่ ซึ่งเป็นระบบอิเล็กทรอนิกส์ที่ใช้สำหรับงานควบคุมรวมถึงการแสดงผลการทำงานต่างๆ โดยที่ระบบ embedded เหล่านี้ถูกใช้เป็นส่วนหนึ่งของระบบและอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรต่างๆ การที่ใช้คำว่า "ระบบแบบฝังตัว" เนื่องจากระบบเหล่านี้เป็นส่วนหนึ่งของระบบใหญ่ ในกรณีที่ผู้ใช้ตัวไปอาจไม่ทราบว่าอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรรวมถึงระบบใดที่ใช้งานเป็นประจำเหล่านั้นเป็นระบบแบบฝังตัว ในบางครั้งแม้แต่ผู้ที่มีความรู้ทางด้านเทคนิคก็ไม่สามารถระบุได้แน่ชัดว่าระบบใดมีระบบแบบฝังตัวอยู่ จนกว่าจะมีการทำงานและตรวจสอบกับระบบและอุปกรณ์ควบคุมนั้นระยะหนึ่งเลยทีเดียว

ระบบ embedded ไม่ใช่เครื่องคอมพิวเตอร์ แต่ก็มีระบบคอมพิวเตอร์อยู่ภายใน อาจจะเป็นเพียงชิปธรรมดาหรือไมโครโปรเซสเซอร์ที่ประกอบด้วยชิปที่มีวงจรซับซ้อน โดยจะมีหลักการทำงาน คือ มีสัญญาณข้อมูลเข้าจากอุปกรณ์ เซนเซอร์เข้าสู่ระบบ และมีสัญญาณผลลัพธ์ ของระบบไปควบคุมบังคับสวิตช์เครื่องควบคุมต่างๆ เช่นสวิตช์เครื่องจักร หรือ วาล์วควบคุมทิศทางการไหลของท่อต่างๆ

นอกจากนี้แบบและรุ่นของระบบ embedded ก็มีมากมายทั้งระบบที่เป็นแบบง่ายๆการทำงานไม่ซับซ้อน ตลอดจนแบบที่ซับซ้อน ซึ่งขึ้นอยู่กับประเภทและจำนวนไมโครโปรเซสเซอร์ รวมถึงงานโปรแกรมควบคุมระบบ

ไมโครโปรเซสเซอร์หรือชิปที่ได้รับการออกแบบจะมีทั้งแบบเพื่อให้สามารถเพิ่มเข้าไปในระบบ embedded ได้ แต่ที่พบเห็นโดยทั่วไปเป็นแบบที่ควบคุมคำสั่ง โปรแกรมนี้จะไม่สามารถแก้ไขได้อีก ถ้าต้องการแก้ไขก็จะต้องนำชิปตัวใหม่ที่บรรจุคำสั่งโปรแกรมที่แก้ไขแล้ว

ระบบ embedded สามารถนำไปประยุกต์ใช้งานได้หลายทาง อาทิ ใช้เป็นส่วนประกอบของคอมพิวเตอร์ เช่น เครื่องพิมพ์, มัลติมีเดีย, กราฟิก ในระบบสื่อสาร เช่น โมเด็ม, โทรสาร, โทรศัพท์ อุปกรณ์อำนวยความสะดวกภายในบ้าน เช่น เครื่องเล่นซีดี, วีซีอาร์, ไมโครเวฟ รวมทั้ง ใช้ในระบบควบคุม เช่น โรบอท, รถยนต์, ดาวเทียม เป็นต้น

2.1.1 บทบาทของระบบ Embedded

- ทำหน้าที่เป็นมอนิเตอร์ให้กับอุปกรณ์ต่างๆ โดยจะอ่านอินพุตจากเซนเซอร์และโปรเซสอินพุตเข้ามา จากนั้นทำการแสดงผลลัพธ์ที่ได้ออกทางอุปกรณ์แสดงผล
- ทำหน้าที่ควบคุมอุปกรณ์ต่างๆ โดยทำการสร้างและส่งคำสั่งควบคุมไปยังอุปกรณ์ควบคุม
- ทำการเปลี่ยนแปลงแก้ไขข้อมูลทั้งในส่วนของการเพิ่มและลดขนาดของข้อมูล

2.1.2 ลักษณะการใช้งานระบบ Embedded

การนำระบบ embedded ใช้งานต้องมีลักษณะดังนี้

- ASIC ต้องเร็ว มีการอินทิเกรตที่สูงแต่จะยุ่งยากในการออกแบบที่ซับซ้อน
- ไมโครโปรเซสเซอร์ ซึ่งสามารถควบคุมฟังก์ชั่นต่างๆในระบบได้ง่าย มีความยืดหยุ่นสูงและสามารถเพิ่มเติมแก้ไขได้ง่าย แต่อาจเกิดปัญหาหากไมโครโปรเซสเซอร์ช้า ซึ่งแก้ไขโดยใช้มัลติโปรเซสเซอร์
- ออกแบบทั้งด้านซอฟต์แวร์และฮาร์ดแวร์โดยเน้นไปที่ฮาร์ดแวร์เพราะมีมูลค่าสูงกว่าซอฟต์แวร์
- เป็นระบบชิปเดียว

2.1.3 คุณสมบัติของระบบ Embedded

- มีแอพพลิเคชั่นที่เฉพาะเจาะจง
- มีโครงสร้างเป็นแบบสแตติก
- ใช้ระบบ DSP
- เป็นระบบฮาร์ดแวร์เรียลไทม์
- เป็นระบบที่มีการตอบรับที่ดี (Reactive Systems)
- มีการกระจายระบบที่ดี (Distributed Systems)

2.1.4 ข้อดีของระบบ Embedded

- เหมาะสมในการเป็นเครื่องมือในการพัฒนาระบบ
- สนับสนุน มาตรฐาน API สามารถพัฒนาแอพพลิเคชั่นได้ง่ายและรวดเร็ว
- สนับสนุนอุปกรณ์ภายนอก เช่น โมเด็ม
- สามารถใช้งานระบบเน็ตเวิร์กได้
- ใช้ต้นทุนในการพัฒนาน้อย
- สามารถเพิ่มเติมความสามารถของระบบได้อย่างต่อเนื่อง

2.2 ลีนุกซ์

ในระบบ Embedded โดยทั่วไปแล้วจะเป็นระบบที่มีไมโครชิปหรือไมโครโปรเซสเซอร์ที่ไม่มีส่วนจัดการความหน่วยจำ (MMU – Memory Management Unit) เป็นส่วนกลางในการประมวลผล ดังนั้นระบบปฏิบัติการที่เลือกใช้นั่นคือระบบปฏิบัติการ ไมโครคอนโทรลเลอร์ ลีนุกซ์ (uClinux) ซึ่งเป็นระบบปฏิบัติการที่ใช้กับไมโครโปรเซสเซอร์ที่ไม่มีหน่วยจัดการความจำ และยังเป็นระบบปฏิบัติการที่นิยมในการพอร์ตลงในแพลตฟอร์มต่างๆ

แท้ที่จริงลีนุกซ์เป็นระบบปฏิบัติการที่ถูกพัฒนาให้ใช้งานบนเดสก์ทอแปละเซิร์ฟเวอร์โดยเฉพาะทำงานร่วมกับซีพียูบนสถาปัตยกรรมแบบ x86 แต่ด้วยข้อดีของลีนุกซ์ที่เปิดเผยโค้ดเคอร์เนล รวมทั้งความทนทานและความยืดหยุ่นในการนำไปใช้งาน จึงได้พัฒนาให้ทำงานร่วมกับแอพพลิเคชั่นแบบ embedded ซึ่งลักษณะของแอพพลิเคชั่นแบบ embedded บางตัวไม่มีส่วนจัดการหน่วยความจำ แต่เนื่องจากมาตรฐานของลีนุกซ์เคอร์เนลจะสนับสนุนแอพพลิเคชั่นที่ต้องการส่วนจัดการหน่วยความจำ ดังนั้นเพื่อให้ลีนุกซ์สามารถใช้งานร่วมกับแอพพลิเคชั่นแบบ embedded ที่ไม่มีส่วนจัดการหน่วยความจำจำเป็นต้องมีการจัดเตรียมวิธีการพอร์ตลีนุกซ์ไปกับอุปกรณ์เหล่านั้น

ลีนุกซ์ที่ใช้ทำงานร่วมกับแอพพลิเคชั่นแบบ embedded คือ ไมโครคอนโทรลเลอร์ ลีนุกซ์ หรือ uClinux (จะเรียกแทนว่า ไมโครซีลีนุกซ์ ในภายหลัง) โดยทำการเอาส่วนที่ต้องการส่วนจัดการหน่วยความจำออกจากลีนุกซ์แล้วแทนที่ด้วยโมเดลหน่วยความจำแบบแฟลช ซึ่งสถาปัตยกรรมใหม่ที่เกิดขึ้นยังคงความสามารถของเคอร์เนลไว้อยู่และเปิดเผยโค้ด

ไมโครซีลีนุกซ์ กำเนิดมาจากลีนุกซ์เคอร์เนลเวอร์ชั่น 2.0 ทำการพัฒนาโดย D. Jeff Dionne และ Kenneth Albanowski มีจุดมุ่งหมายเพื่อใช้งานสำหรับไมโครคอนโทรลเลอร์ที่ไม่มีส่วนจัดการส่วนความจำ โดยมุ่งเน้นไปที่ระบบ embedded เนื่องด้วยโปรเซสเซอร์ที่ไม่มีส่วนจัดการหน่วยความจำจะราคาถูกจึงเหมาะที่จะนำไปใช้กับระบบ embedded

ไมโครซีลีนุกซ์มีลักษณะการทำงานแบบหลายงานพร้อมกัน (multitasking) สามารถนำแอพพลิเคชั่นมาทำงานบนไมโครซีลีนุกซ์โดยไม่จำเป็นต้องเป็นแบบหลายงานพร้อมกันได้

เคอร์เนลของไมโครซีลีนุกซ์เป็นฉบับย่อของเคอร์เนลเวอร์ชั่น 2.0 ซึ่งยังคงข้อดีหลักของลีนุกซ์ไว้ ได้แก่ ความมีเสถียรภาพ (stability), ความสามารถในการติดต่อกับเน็ตเวิร์ก, สนับสนุนระบบไฟล์ (file systems) และมาตรฐาน API รวมทั้งขนาดของโค้ดจะมีขนาดเล็กกว่าลีนุกซ์

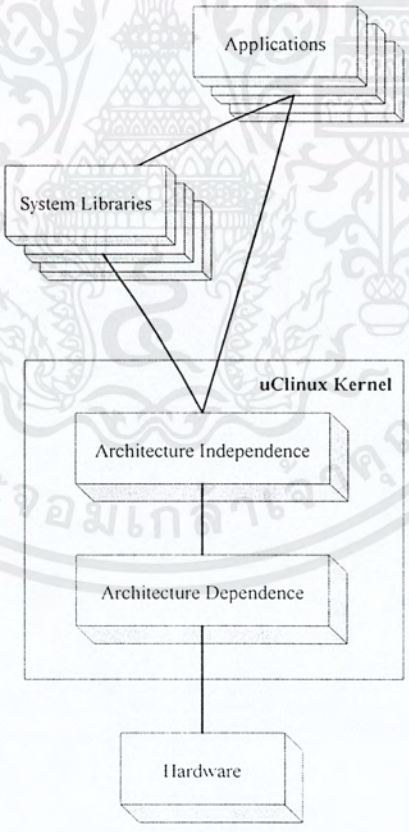
2.2.1 สถาปัตยกรรมของไมโครซีลีนุกซ์

ลักษณะที่ต้องมีการพิจารณาในการออกแบบลีนุกซ์เคอร์เนล ได้แก่ ความชัดเจน (clarity), ความเข้ากันได้ (compatibility), ทนทาน (robustness), ปลอดภัย (security), และเร็ว (speed) ซึ่งสามารถแยกพิจารณาทีละส่วนดังนี้

1. Clarity คือ ลีนุกซ์เคอร์เนลที่ออกแบบขึ้นมาต้องมีความชัดเจน สามารถทำความเข้าใจได้ง่ายทั้งส่วนเรื่องของความเร็วและความทนทาน โดยความชัดเจนในเรื่องความทนทานหมายความว่าสามารถพิสูจน์ความถูกต้องของข้อมูลได้ง่าย และหากเกิดความผิดพลาดขึ้นสามารถดีบักได้ง่าย ระบบเกิดความเสียหายได้ยาก แต่ในส่วนของความชัดเจนกับความเร็วจะตรงกันข้าม ถ้าในตัวคอมไพล์โค้ดความชัดเจนจะสำคัญกว่า แต่ในการออกแบบเคอร์เนลควรเน้นไปที่ความเร็ว เพราะผู้พัฒนาสามารถทำการอธิบายเกี่ยวกับความชัดเจนได้อยู่แล้ว
2. Compatibility ลีนุกซ์เคอร์เนลสามารถนำไปใช้งานร่วมกับยูนิกซ์ได้โดยสนับสนุนมาตรฐาน POSIX และยังใช้งานร่วมกับตัวอื่นๆ ได้แก่ สนับสนุนการรันไฟล์ .class ของจา

วา สามารถทำการเอ็กซ์ซิคิว DOS ผ่าน DOSEMU อีมูเลเตอร์ รวมทั้งสนับสนุนการระบบเน็ตเวิร์กด้วยโปรโตคอล TCP/IP

3. Portability หมายความว่าลีนุกซ์เคอร์เนลสามารถใช้งานร่วมกับฮาร์ดแวร์โดยสามารถรันลีนุกซ์เคอร์เนลบนฮาร์ดแวร์หลายแพลตฟอร์มได้ ได้แก่ อัลฟา, ARM, โมโตโรล่า, MIPS, พาวเวอร์พีซี, SPAEC และ SPARC-64 เป็นต้น การที่ลีนุกซ์สามารถทำงานได้เนื่องจากว่าโค้ดของลีนุกซ์เคอร์เนลไม่ขึ้นอยู่กับสถาปัตยกรรม ทำให้สามารถนำไปใช้งานได้หลากหลาย
4. Robustness และ Security ลีนุกซ์เคอร์เนลที่พัฒนาขึ้นต้องมีความทนทานและปลอดภัย ควรจะมีการป้องกันระบบจากระบบอื่น โดยเฉพาะทำการป้องกันโปรเซสที่ทำงานจากส่วนอื่นๆ สามารถทำการดีบักและแก้จ้อผิดพลาดได้ง่าย
5. Speed ในการออกแบบลีนุกซ์เคอร์เนลส่วนที่ต้องพิจารณาอันดับแรกซึ่งมีความสำคัญมาก ได้แก่ ความเร็ว ความทนทาน ความปลอดภัย และความเข้ากันได้ เพราะสามารถตรวจสอบประสิทธิภาพการทำงานของระบบได้จากความเร็วในการทำงาน การรันงานต่างๆ ซึ่งบางครั้งสามารถจับเวลาในการรันโปรเซสด้วยตัวเองได้



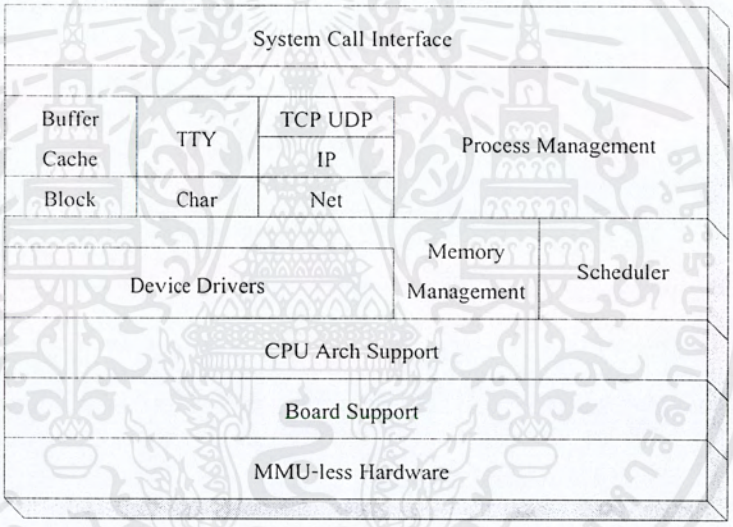
รูปที่ 2-1 ภาพรวมสถาปัตยกรรมเคอร์เนล

จากรูปที่ 2-1 แสดงภาพรวมภายในสถาปัตยกรรมของเคอร์เนลที่เหมือนกับระบบยูนิกซ์ โดยมีคุณสมบัติของเคอร์เนล ได้แก่

1. เคอร์เนลทำการแยกแอพพลิเคชั่นออกจากฮาร์ดแวร์
2. สถาปัตยกรรมของเคอร์เนลประกอบด้วย ฮาร์ดว์และพอร์ตเทเบิล

โดยส่วนที่แยกแอพพลิเคชั่นออกจากฮาร์ดแวร์คือส่วนของเคอร์เนลดังแสดงในรูป 2-1 โดยส่วนของเคอร์เนลนั้นแบ่งออกเป็นสองส่วนด้วยกัน ได้แก่

1. Architecture Independence คือส่วนที่ไม่ยึดติดกับฮาร์ดแวร์ใดๆ
2. Architecture Dependence คือส่วนที่ยึดติดกับฮาร์ดแวร์ เมื่อนำเคอร์เนลไปทำงานบนแพลตฟอร์ม หรือฮาร์ดแวร์ที่แตกต่างกัน จะต้องมีการปรับเปลี่ยนโค้ดในส่วนนี้ให้ทำงานเข้ากับแพลตฟอร์ม หรือฮาร์ดแวร์นั้นๆ โดยส่วนนี้จะถูกเรียกใช้งานจาก Architecture Dependence



รูปที่ 2-2 รายละเอียดสถาปัตยกรรมเคอร์เนลของไมโครซีลีนุกซ์

จาก รูปที่ 2-2 แสดงสถาปัตยกรรมของเคอร์เนล ซึ่งประกอบด้วยส่วนประกอบหลัก 5 ส่วนดังนี้

1. Process Management
2. Scheduler
3. Memory Management
4. File Systems
5. Networking
6. Device Drivers

2.2.1.1 Process Management

เคอร์เนลทำหน้าที่สร้างและทำลายโปรเซส และทำการแฮนเดิลโปรเซสที่ติดต่อกับภายนอก (อินพุตและเอาท์พุต) โดยสามารถติดต่อผ่านซิกเนล, ไปป์ และติดต่อภายในโปรเซสด้วยกันเอง รวมถึงมีตัวจัดลำดับการรันโปรเซสนั่นคือ Scheduler

2.2.1.2 Scheduler

การที่โปรเซสหนึ่งโปรเซสใดจะถูกนำไปทำงาน หรือจะหยุดการทำงานโดยเคอร์เนลจะจัดการดูแลลำดับการนำโปรเซสใดไปรัน หรือหยุดการทำงานของโปรเซสนั้นชั่วคราวและนำโปรเซสนั้นๆรอในการรันต่อไป

2.2.1.3 Memory Management

หน่วยความจำภายในคอมพิวเตอร์เป็นทรัพยากรหลักของระบบซึ่งสามารถใช้วัดประสิทธิภาพของระบบได้ เคอร์เนลทำการจัดสรรพื้นที่ว่างในหน่วยความจำให้กับโปรเซส โดยส่วนประกอบต่างๆของเคอร์เนลทำการติดต่อกับส่วนของการจัดการหน่วยความจำผ่านชุดของฟังก์ชัน call ได้แก่ malloc/free

2.2.1.4 File Systems

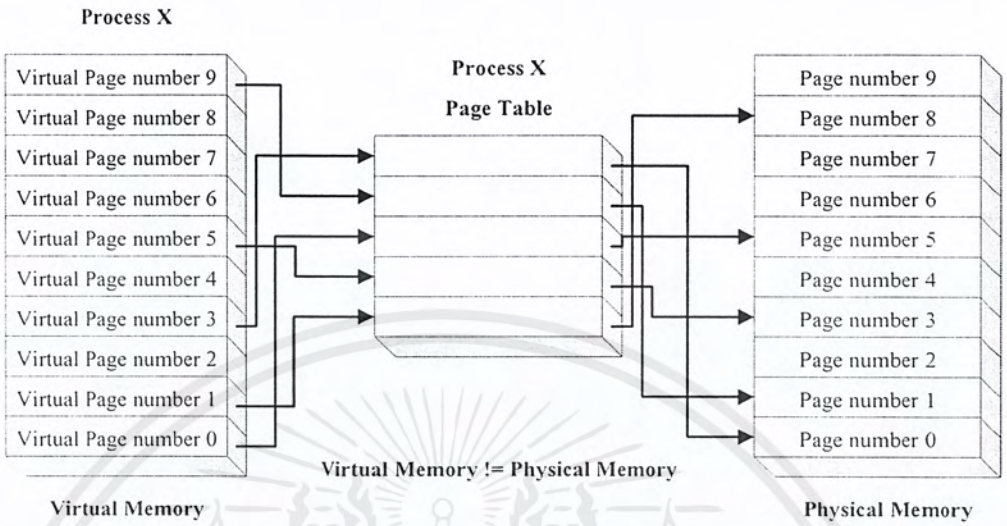
ระบบยูนิกซ์อยู่บนแนวคิดพื้นฐานของ filesystem โดยมองส่วนต่างๆภายในยูนิกซ์เป็นไฟล์ โดยส่วนนี้จะรวมทั้ง Character Device Drivers และ Block Device Drivers ด้วย เคอร์เนลทำการสร้างโครงสร้างของ filesystem บนฮาร์ดแวร์ โดยลีนุกซ์สนับสนุน multiple filesystem ซึ่งมีหนทางในการจัดการกับดาต้าบนสื่อที่ต่างกัน

2.2.1.5 Networking

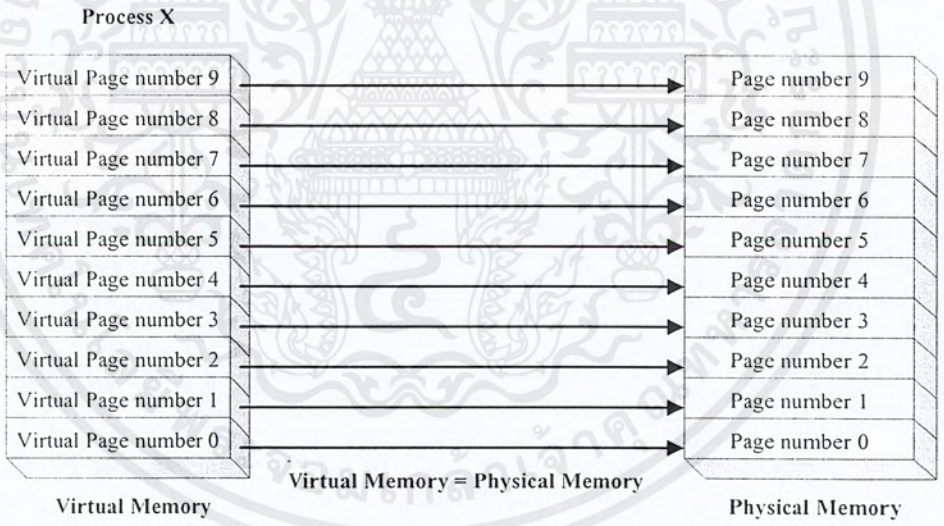
ในส่วนของเน็ตเวิร์กต้องมีการจัดการ โดยระบบปฏิบัติการ เนื่องจากว่าโอเปอเรชั่นของเน็ตเวิร์กส่วนใหญ่แล้วต้องไม่ถูกระบุในโปรเซส ซึ่งแพ็กเกจที่เข้ามาเป็นเหตุการณ์แบบอะซิงโครนัส ทำให้ต้องมีการจัดกลุ่ม กำหนดชื่อ และเลือกแพ็กเกจก่อนที่โปรเซสจะใช้งานแพ็กเกจเหล่านี้ ระบบทำการส่งข้อมูลของแพ็กเกจผ่านโปรแกรมและเน็ตเวิร์กอินเตอร์เฟส ซึ่งโปรแกรมต้องทำการกำหนดการคอยข้อมูลจากเน็ตเวิร์ก

2.2.1.6 Device Drivers

ส่วนใหญ่ระบบปฏิบัติการทั่วๆไปทำการแมปไปยังอุปกรณ์ภายนอก ยกเว้นโปรเซสเซอร์, หน่วยความจำ, เอนติตี้ 2-3 ตัว และอุปกรณ์ที่ใช้ในการควบคุมโอเปอเรชั่นจะปฏิบัติตามโค้ดที่ทำการระบุแต่ละดีไวซ์ ซึ่งโค้ดที่กล่าวถึงนี้คือ ดีไวซ์ไดรเวอร์ (device driver) เคอร์เนลต้องมีส่วนของดีไวซ์ไดรเวอร์สำหรับอุปกรณ์ที่เชื่อต่อกับระบบ เช่นไดรเวอร์ของคีย์บอร์ด, หน้าจอ เป็นต้น



รูปที่ 2-3 แสดงการเก็บเมมโมรีในลีนุกซ์



รูปที่ 2-4 แสดงการเก็บเมมโมรีในไมโครซีลีนุกซ์

จากรูป 2-3 เป็นการเก็บเมมโมรี่ในรูปแบบของเวอร์ชวลเมมโมรี ในระบบปฏิบัติการลีนุกซ์ทั่วไป ซึ่งจะเห็นว่าตำแหน่งของเวอร์ชวลเมมโมรีจะไปดูค่าตำแหน่งจริงในเพจเทเบิล ซึ่งในส่วนนี้ก็คือส่วนจัดการหน่วยความจำ หรือ MMU ซึ่งจากรูปจะเห็นว่า ตำแหน่งของเวอร์ชวลเมมโมรี่จะไม่เท่ากับตำแหน่งเมมโมรี ซึ่งจะไม่เหมือนกับรูปที่ 2-4 ซึ่งเป็นการเก็บเมมโมรีในระบบปฏิบัติการ ไมโครซีลีนุกซ์ ซึ่งไม่มีส่วนจัดการหน่วยความจำ ซึ่งจะไม่มีการสร้างเพจเทเบิล ซึ่งจะสามารถอ้างตำแหน่งของ

หน่วยความจำได้โดยตรง ซึ่งจากรูปจะเห็นว่าตำแหน่งของเวอร์ชวลเมมโมรีจะมีค่าเท่ากับตำแหน่งจริงในเมมโมรี

2.2.2 คุณลักษณะเด่นของไมโครซีลีนุกซ์

1. สนับสนุนมาตรฐาน API
2. uCkernel < 512 KB
3. uCkernel + tools < 900 KB
4. มีทีซีพี/ไอพีแสตก พร้อมใช้งานอินเตอร์เน็ตได้
5. สนับสนุนโปรโตคอลที่ใช้เชื่อต่อเน็ตเวิร์ก
6. สนับสนุน File System ประกอบด้วย NFS, ext2, ROMfs, JFFS (Journaling FLASH File System), MS-DOS และ FAT15/32
7. มีตัวจัดการงานแบบ preemptive
8. สนับสนุนมาตรฐาน POSIX 4 คือเพิ่มความสามารถในด้านเรียลไทม์เข้าไป
10. สามารถรันแอพพลิเคชั่นได้ทั้งแบบ Single และ Multithread
11. รันภายใต้โปรเซสเซอร์หลายตระกูล ได้แก่ ARM7TDMI, Motorola DragonBall (M68EZ328), M68328, M68EN322, ColdFire, QUICC (Quad Integrated Communications Controller), MC68EN302, Axis ETRAX, Intel i960, PRISMA, Atari 68k

2.2.3 ข้อจำกัดของไมโครซีลีนุกซ์

ไมโครซีลีนุกซ์ใช้กับโปนเซสเซอร์ที่ไม่มีส่วนจัดการหน่วยความจำ ทำให้มีข้อจำกัดดังนี้

1. ไมโครซีลีนุกซ์ไม่สามารถใช้งาน vfork() ได้ ซึ่งเป็นฟังก์ชันในการสร้างโปรเซสลูกขึ้นมารันที่มีองค์ประกอบเหมือนกันทุกอย่าง โดยทำการใช้งาน vfork() แทน ซึ่งไม่ได้หมายความว่าไม่สามารถทำงานหลายงานพร้อมกันได้ แต่หมายความว่าโปรเซสทำงานคอยจนกว่าโปรเซสลูกทำการ exec() หรือ exit() ซึ่งยังคงใช้งานแบบทำงานพร้อมกันได้อย่างเต็มที่
2. ไมโครซีลีนุกซ์ไม่สามารถขยายขนาดสแตกได้ และไม่มี brk() (การขยายขนาดของหน่วยความจำที่ร้องขอ) โดยใช้ mmap() แทนเพื่อจองหน่วยความจำ
3. ไม่มีการปกป้องหน่วยความจำ โปรแกรมต่างๆสามารถเกิดความเสียหายรวมทั้งเคอร์เนล ทั้งนี้เนื่องจากการที่ไม่มีส่วนจัดการหน่วยความจำนั่นเอง
4. สถาปัตยกรรมที่ใช้มีขนาดของโค้ดที่แตกต่างกัน ขึ้นอยู่กับรูปแบบการนำไปใช้งานว่าต้องการความสามารถอย่างไรบ้าง

2.2.4 ข้อดีของไมโครซีลีนุกซ์

1. ขนาดเล็ก
2. เปิดเผยโค้ด

3. เหมาะในการนำไปใช้กับโปรเซสเซอร์ที่ไม่มีส่วนจัดการหน่วยความจำ โดยเฉพาะระบบ embedded
4. เหมาะกับระบบที่ราคาถูก และใช้พลังงานต่ำ โดยเฉพาะระบบ embedded

2.3 แนวทางการพอร์ตไมโครซีลีนุกซ์สำหรับฮาร์ดแวร์ใหม่

เนื้อหาในส่วนนี้อธิบายเกี่ยวกับการจัดเตรียมระบบปฏิบัติการ ได้แก่ ไมโครซีลีนุกซ์ เพื่อสามารถใช้งานร่วมกับสถาปัตยกรรมใหม่ที่ออกแบบขึ้นได้ โดยเน้นไปที่การใช้งานร่วมกับระบบ embedded

2.3.1 Cross-Development Tools

ขั้นตอนแรกในการจัดเตรียมให้สามารถสนับสนุนสถาปัตยกรรมใหม่ที่สร้างขึ้นมาต้องจัดหาเครื่องมือในการสร้างโค้ดซึ่งเครื่องมือเหล่านี้ต้องประกอบด้วยส่วนประกอบพื้นฐานในการพัฒนา โดยทำการเลือก GNU (http://www.gnu.org) เป็นเครื่องมือในการสร้างโค้ด เนื่องจากมีความยืดหยุ่นเพียงพอในการที่จะพบความต้องการของระบบ ซึ่งในโครงการนี้เป็นโครงการที่จะนำไมโครซีลีนุกซ์พอร์ตลงโปรเซสเซอร์ ARM7TDMI จึงได้นำเครื่องมือเหล่านี้ที่ http://www.uclinux.org

ในการสร้างโค้ดส่วนของเคอร์เนลมีความแตกต่างจากโค้ดของยูสเซอร์ คือว่า โค้ดเคอร์เนลถูกเชื่อมโยงไปยังตำแหน่งที่เจาะจงเป็นพิเศษซึ่งอยู่ในหน่วยความจำจริงๆ ดังนั้น การอ้างอิงถึงดาต้าและโปรแกรมทั้งหมดต้องเป็นแบบ absolute หรือแบบ relative ทำให้คอมไพล์เลอร์สามารถจัดการกับโค้ดได้อย่างอิสระ ในส่วนของยูสเซอร์โปรแกรมจะถูกโหลดมาจากไฟล์ซิสเต็มซึ่งเกี่ยวข้องกับการกำหนดตำแหน่งที่หน่วยความจำโดยตรง และต้องสนับสนุนตำแหน่งใหม่ที่กำหนดขึ้นด้วย มิฉะนั้นจะไม่สามารถใช้งานยูสเซอร์โปรแกรมได้

ส่วนประกอบหลักที่ไมโครซีลีนุกซ์ต้องการใช้ในการพัฒนา ได้แก่ binutils (ประกอบด้วยแอสเซมเบลอร์และลิงค์เกอร์), GCC เป็นตัวคอมไพล์เลอร์, newlib เป็นไฟล์เฮดเดอร์, GDB เป็นตัวดีบักเกอร์ และยูทิลิตี้เพิ่มเติม เครื่องมือของ GNU ใช้ autoconf เพื่อทำการสร้างสคริปต์ของการคอนฟิก ซึ่งค่าตัวเลือกกลางจะระบุในไดเรกทอรี่ของการติดตั้ง, สถาปัตยกรรมหลัก และสถาปัตยกรรมของโฮสต์ (/usr/local/arm-elf (ขึ้นอยู่กับยูสเซอร์ระบุ), arm-elf และ i386-pc-linux) สามารถแสดงการกำหนดค่าต่างๆ ดังรูปที่ 2-5

```
#!/bin/sh
export CC="gcc"
/bin/sh ../egcs/configure <\ \>
-prefix="/usr/local/arm-elf: -build=i386-pc-linux <\ \>
-host=i386-pc-linux -target=arm-elf -enable-language=c,c++
```

รูปที่ 2-5 ส่วนของเชลล์สคริปต์ที่ใช้กำหนดค่าต่างๆสำหรับคอนฟิก GNU

2.3.1.1 Binutils

แพ็กเกจ binutils ประกอบด้วยตัวแอสเซมเบลอร์, ลิงค์เกอร์ และยูทิลิตี้พื้นฐานสำหรับการแฮนเดิลไฟล์ การคอนฟิกแพ็กเกจ binutils ต้องกำหนดรูปแบบและจุดมุ่งหมายของออบเจ็กต์ไฟล์ที่ต้องการด้วย เครื่องมือทั้งหมดที่อยู่ในแพ็กเกจ binutils ทำการสร้างขึ้นโดยใช้ไลบรารี่ BFD (Binary File Description) เพื่อทำการแลกเปลี่ยนข้อมูลกัน ส่วนไฟล์คอนฟิก config.bfd จะทำการระบุรูปแบบของไบนารี่ให้เป็นดีฟอลต์ เช่น elf little endian แสดงดังรูปที่ 2-6 และการ

```
arm-*-uclinux | armel-*-uclinux*)
targ_defvec=bfd_elf32_littlearm_vec
targ_selves="bfd_elf32_bigarm_vec armcoff_little_vec armcoff_big_vec"
;;
```

รูปที่ 2-6 การเพิ่มเติมส่วนของไฟล์ config.bfd เพื่อทำการระบุรูปแบบไบนารีเป้าหมาย

2.3.1.2 คอมไพเลอร์ C

ตัวคอมไพเลอร์ที่ใช้คือ EGCS (Experimental GNU Compiler System) ซึ่งเป็นคอมไพเลอร์ที่สามารถคอมไพล์ได้หลายภาษารวมทั้งจาวาด้วย

2.3.1.3 newlib

newlib นั้นจะบรรจุเฮดเดอร์ไฟล์ที่จำเป็นต้องใช้และอื่นๆ newlib ที่เลือกใช้คือเวอร์ชัน newlib-1.8.1

```
#/bimutils/configure --target=arm-elf --prefix=/usr/local/arm-elf
```

รูปที่ 2-7 แสดงการคอนฟิก binutils

```
#/egcs/configure --target=arm-elf --prefix=/usr/local/arm-elf --with-cpu=arm7tdmi
--with-newlib --with-header=../newlib-1.8.1/newlib/libc/include --enable-language=c,c++
--with-gnu-as --with-gnu-ld
```

รูป 2-8 ที่ แสดงการคอนฟิกคอมไพเลอร์

```
#/newlib-1.8.1/configure --target=arm-elf --prefix=/usr/local/arm-elf
```

รูปที่ 2-9 แสดงการคอนฟิก newlib

2.3.1.4 ดีบักเกอร์

สำหรับตัวดีบักเกอร์ที่ใช้ในการพัฒนาซอฟต์แวร์ ได้แก่ GNU ดีบักเกอร์ (GDB) ซึ่งตัวโปรเซสเซอร์บางตัวสนับสนุนวิธีการดีบักที่สามารถทำการอินเตอร์เฟสผ่านโมดูลของโปรเซสเซอร์ ซึ่งโมดูลเหล่านี้ได้แก่ ARM EmbeddedICE และ Motolora BDM ถ้าฮาร์ดแวร์ไม่ได้เตรียมการสนับสนุนในส่วนนี้ จำเป็นต้องทำการแพตช์ (patch) kGDB ที่เป็นโมดูลของเคอร์เนล เพื่อให้สามารถสนับสนุนตัวรีโมทดีบักเกอร์ ปัจจุบันเคอร์เนลเวอร์ชั่น 2.2 และ 2.3 ประกอบด้วยโมดูลนี้อยู่แล้ว

2.3.1.5 Object Format Utilities

เนื่องจากว่า เครื่องมือของ GNU ไม่สนับรูปแบบ แฟลทไบนารี่ (Flat Binary) ของยูสเซอร์โปรแกรม จึงจำเป็นต้องเพิ่มส่วนไบนารียูทิลิตี้ เพื่อใช้สร้างแฟลทไบนารี ได้แก่ Coff2flt และ elf2flt ซึ่งเป็นยูทิลิตี้ที่ใช้กับไมโครซีลีนุกซ์เพื่อทำการแปลงไบนารีไปเป็นรูปแบบที่สามารถใช้งานได้โดยตัวโหลดเคอร์เนล (kernel loader)

2.3.2 การปรับปรุงแก้ไขเคอร์เนล

ขณะที่ส่วนประกอบหลักของโค้ดลีนุกซ์ก็คือส่วนที่เป็นอิสระจากโปรเซสเซอร์และสถาปัตยกรรม แต่โค้ดระดับล่างสุดของโค้ดจะมีลักษณะเฉพาะตัวของแต่ละระบบ ลักษณะที่ต้องมีเหมือนกันในสถาปัตยกรรมต่างๆ ได้แก่ มีตัวอินเทอรัพท์แฮนเดิลเป็นของตัวเอง, การบำรุงรักษาแมมโมรีแมป, โครงสร้างของทาส์กและโปรเซสต่างๆ ซึ่งรูทีนเหล่านี้ถูกเก็บไว้ใน arch/{architecture} ถ้าต้องการให้สถาปัตยกรรมต่างๆสนับสนุนลีนุกซ์ รูทีนในระดับล่างที่ใช้สำหรับสถาปัตยกรรมต้องสามารถโมเดลรูปแบบของสถาปัตยกรรมที่เหมือนๆกันได้ เพื่อให้การเพิ่มรูทีนมนระดับล่างเป็นไปได้ง่าย จำเป็นต้องมีพอร์ตอนุกรมช่วยในการดีบักเมสเซจ

2.3.2.1 Initialization

มีอยู่ 2 ขั้นใน Initialization นั่นคือ bootloader เพื่อใช้ทำการรีเซ็ตและจัดเตรียมระบบสำหรับการเอ็กซิคิวต์ start_kernel โดย start_kernel อยู่ใน init/main.c และกำหนดค่าให้กับสิ่งแวดล้อมต่างๆ สำหรับการเซ็ตอัพระบบ โดยโค้ดส่วนนี้จะอยู่ที่ setup_arch ใน arch/{architecture}/kernel/setup.c ซึ่งเราต้องการระบบที่โหลดได้เร็ว โค้ดที่ใช้กำหนดค่าเริ่มต้นให้กับระบบคือ head.S ตัวพารามิเตอร์สำหรับการเซ็ตอัพจะประกอบไปด้วยตำแหน่งเริ่มต้นและสิ้นสุดของหน่วยความจำ ส่วนของคำสั่งที่เพิ่มเติมขึ้นมาส่งผ่านมาจากตัวโหลดเดอร์อีกที และส่วนของงานหลักสำหรับเอ็กซิคิวต์เคอร์เนลเมื่อทำการบูตเคอร์เนลขึ้นมา

2.3.2.2 Scheduling/Tasking

ลักษณะของการทำงานพร้อมกันภายใต้ไมโครซีลีนุกซ์มีลักษณะง่ายกว่า ทำงานภายใต้ลีนุกซ์เนื่องจากไม่มีเพจเทเบิล และการป้องกันการแฮนเดิล โดยการทำงานจริงที่เกิดขึ้นต้องแสดงคอนเท็กซ์ ที่

ถูกต้องทั้งส่วนเก็บค่าและคืนค่า ซึ่งคอนเท็กซ์จะเก็บค่ารีจีสเตอร์ทั้งหมดไว้และนำมาใช้เมื่อโปรเซสเซอร์ถูกอินเตอร์รัพท์

ตัวคอนเท็กซ์รีจีสเตอร์ถูกกำหนดไว้ในโครงสร้าง rt_regs จะอยู่ใน asm/ptrace.h ถ้าหากใช้รีจีสเตอร์ที่มีการรีเทิร์นค่าคืนกลับมา จำเป็นต้องมีการเก็บค่าเดิมไว้ (จากตัวอย่างจะเป็น ARM_ORIG_r0) เมื่อระบบมีการรีเทิร์นจาก system call ต้องใส่ค่ารีจีสเตอร์ที่รีเทิร์นกลับมาให้กับตัวรีจีสเตอร์ด้วย แต่ถ้าระบบถูกรีเทิร์นจากการอินเตอร์รัพท์ ตัวรีจีสเตอร์ก็ยังคงเห็บค่าเดิมไว้

2.3.2.3 Memory Accounting

บทบาทของหน่วยความจำเสมือนจะเกี่ยวข้องกับการใช้ประโยชน์จากทรัพยากรของระบบ โดยใช้รูทีน paging_init เพื่อจัดสรรพื้นที่ว่างในหน่วยความจำ ถ้าหากระบบถูกคอนฟิกสำหรับใช้เมมโมรีแมป รูทีนนี้จะถูกเพิ่มไปในแต่ละเซ็กชั่นของหน่วยความจำโดยไม่ขึ้นอยู่กับแมปที่ใช้

```
struct rt_regs { long uregs[18] };
#define  ARM_cpsr          uregs[16]
#define  ARM_cp            uregs[15]
#define  ARM_lr            uregs[14]
#define  ARM_sp            uregs[13]
#define  ARM_ip            uregs[12]
#define  ARM_fp            uregs[11]
#define  ARM_r10           uregs[10]
#define  ARM_r9            uregs[9]
#define  ARM_r8            uregs[8]
#define  ARM_r7            uregs[7]
#define  ARM_r6            uregs[6]
#define  ARM_r5            uregs[5]
#define  ARM_r4            uregs[4]
#define  ARM_r3            uregs[3]
#define  ARM_r2            uregs[2]
#define  ARM_r1            uregs[1]
#define  ARM_r0            uregs[0]
#define  ARM_ORIG_r0       uregs[17]      /*-1*/
```

รูปที่ 2-10 แสดงรีจีสเตอร์ ptrace เพื่อใช้สนับสนุนการเก็บค่าและคืนค่าของคอนเท็กซ์

2.3.2.4 อินเตอร์รัพท์

ในการจัดการเกี่ยวกับอินเตอร์รัพท์ ระบบต้องจัดเตรียม entry point เหล่านี้ได้แก่ enable_irq, disable_irq, get_irq_list, request_irq, free_irq, probe_irq_on, probe_irq_off และ init_irq ซึ่งเตอร์เนลและดีไวซ์ไดรเวอร์ทำการเรียกใช้งานรูทีนเหล่านี้สำหรับควบคุมส่วนหลักๆของระบบ

2.3.2.5 ไทม์เมอร์

ฟังก์ชันหลักๆของระบบต้องการส่วนที่จัดการเกี่ยวกับไทม์เมอร์แบบ periodic อินเตอร์รัพท์ โดยใช้รูทีน time_init เพื่อกำหนดค่าเริ่มต้นให้กับเวลาและอินเตอร์รัพท์ และทำการเรียกฟังก์ชัน do_timer ซึ่งจะให้ค่าของเวลาเป็น Hz เพื่อทำการรอการอินเตอรร์รัพท์เข้ามา

2.3.2.6 เคอร์เนลไลบรารีฟังก์ชัน (kernel Library Function)

ไมโครซีลีนุกส์ใช้ไลบรารีที่สนับสนุนฟังก์ชันของซี เพื่อให้ประสิทธิภาพที่ดีที่สุด

2.3.3 ดีไวซ์ไดรเวอร์ (Device Driver)

โปรเซสเซอร์จำเป็นต้องสนับสนุนดีไวซ์ไดร์เวอร์ เพราะถึงแม้ว่าเคอร์เนลจะทำงานได้ แต่จะไม่สามารถทำอะไรได้เลยถ้าหากไม่สนับสนุนดีไวซ์ต่างๆ ซึ่งอย่างน้อยโปรเซสเซอร์ต้องสนับสนุนดีไวซ์เหล่านี้ ได้แก่ คอนโซล, ซีเรียลดีไวซ์ และ บล็อกดีไวซ์ ที่บรรจุรูทไฟล์ซิสเต็มไว้ด้วย ไดรเวอร์เหล่านี้ไม่ถูกใช้ในยูสเซอร์โดยตรง แต่ต้องใช้งานผ่านเคอร์เนลโมดูล เช่นไดรเวอร์ tty หรือไฟล์ซิสเต็ม

2.3.3.1 คอนโซล (Console)

ขั้นแรกในการจัดเตรียมซีเรียลไดรเวอร์ ต้องเตรียมเอาท์พุตของการดีบักคอนโซลก่อน ซึ่งการดีบักคอนโซลสามารถรวมไว้กับช่วงที่ทำการบูตโดยเรียกผ่าน register_console ควรจะเรียกผ่านฟังก์ชัน console_init ที่อีนาเบิลอยู่และตั้งค่าของ bit rate ให้มีค่าคงที่ตายตัว

2.3.3.2 ซีเรียล

ซีเรียลดีไวซ์ถูกออกแบบมาเพื่อใช้แฮนเดิลอินพุตและเอาท์พุตของไดรเวอร์ tty เนื่องจากยูสเซอร์โปรเซสที่ต้องการติดต่อไปยังไดรเวอร์ tty จะไม่สามารถติดต่อโดยตรงไปยังซีเรียลดีไวซ์ได้ ซึ่งจริงๆแล้วซีเรียลดีไวซ์ถูกเรียกจากดีไวซ์ tty สามารถดูแบบจำลองของไดรเวอร์ได้ที่ serial.c

2.3.3.3 ไฟล์ซิสเต็ม บล็อกดีไวซ์ (Filesystem Block Device)

ทางเลือกที่ดีสำหรับเริ่มต้นการทำงานของระบบคือการรวมบล็อกเมมโมรีดีไวซ์ไดรเวอร์ (block memory device driver) (ใช้งานผ่าน blkmem.c) และ แรมดิสก์ไดรเวอร์ (ramdisk driver) เข้าด้วยกัน แรมดิสก์ได้ถูกออกแบบมาเพื่อทำการจองพื้นที่ว่างและอ่านข้อมูลของมันจากบล็อกดีไวซ์ ซึ่งข้อมูลสามารถอยู่ในรูปแบบเดิม หรือบีบอัดได้ โดยแรมดิสก์ไดรเวอร์ทำการอ่านบล็อกแรกเพื่อกำหนดรูปแบบของข้อมูลที่ใช้แสดง

2.3.4 รันไทม์ไลบรารี (Runtime Library)

รันไทม์ไลบรารีใช้เตรียมการอินเทอร์เฟซจากยูสเซอร์โปรแกรมไปยังรูทีนของเคอร์เนล คนส่วนใหญ่คิดว่าระบบเหล่านี้คือระบบลีนุกซ์ซึ่งหมายถึงการรันเคอร์เนล แต่ที่จริงระบบไม่ได้ถูกกำหนดโดยเคอร์เนล แต่กำหนดด้วยรันไทม์ไลบรารีของซี เช่นใช้เป็นรันไทม์ไลบรารีที่ทำการเคลื่อนย้ายการ call printk ไปไว้ในดาต้าบัฟเฟอร์และทำการเขียนเพื่อส่งเอาท์พุตไปยังไฟล์เดสคริปเตอร์ 1 (stdout) ซึ่งระบบการเรียกไลบรารีทำการปรับปรุงเพื่อเพิ่มความสามารถของระบบ โดยแพ็กเกจของไมโครซีลีนุกซ์ มีไลบรารีเพื่อใช้งานคำสั่งพื้นฐานของ ซี ซึ่งส่วนใหญ่ใช้ GNU glibc หรือ newlib

ดังนั้น เพื่อให้สามารถใช้งานร่วมกับสถาปัตยกรรมใหม่ที่สร้างขึ้นมา ต้องประกอบด้วยโค้ดโปรแกรมที่สร้างขึ้นมา, ข้อกำหนดต่างๆของ system call API, การปรับปรุงแก้ไขรูทีนของ I/O, และการเปลี่ยนแปลงสถาปัตยกรรมของแฟลชเมมโมรี ซึ่งแสดงรายละเอียดในแต่ละส่วนต่อไป

2.3.4.1 Program Entry Code

ฟังก์ชันหลักของยูสเซอร์โปรเริ่มด้วย main โดย entry จริงๆจะชี้ไปยัง entry ที่อยู่ในไลบรารีของซี โค้ดนี้เริ่มด้วยสแตติกดาต้าเช่น ค่า error (errno) และพอยเตอร์ที่เกี่ยวข้อง (environ) จากนั้นทำการส่งอาร์กิวเมนต์ argc และ argv ไปยัง main โดยอาร์กิวเมนต์และส่วนที่เกี่ยวข้องกับโปรแกรมถูกเกรียกผ่านรูทีน start_thread (อยู่ใน proc/processor.h) ในส่วน execve system call จะใช้ start_thread เพื่อเซ็ตอัพค่าผิดพลาดของคอนเท็กซ์ที่คืนค่ากลับมา ในส่วนของ program entry code เป็นส่วนประกอบการทำงานของ start_thread แสดงดังรูป 2-11

2.3.4.2 System Calls

ยูสเซอร์โปรแกรมทำการเข้าถึงทรัพยากรของระบบโดยผ่าน system calls เนื่องจากโปรเซสเหล่านี้เกี่ยวข้องกับการสวิตช์จากยูสเซอร์คอนเท็กซ์ไปยังซิสเต็มคอนเท็กซ์ โดยโค้ดของ sstem call มีการระบุโปรเซสเซอร์ที่เฉพาะเจาะจงลงไป โปรเซสของระบบที่เก็บยูสเซอร์คอนเท็กซ์จะบรรจุพารามิเตอร์ของการ call ไว้ด้วย ยูสเซอร์โปรเซสถูกพักไว้จนกว่าเคอร์เนลทำทรานเซ็กชั่นเสร็จและคืนค่าเธรดไปยังฝั่งผู้ใช้ ตัวอย่างของ system call อาร์กิวเมนต์ถูกเก็บไว้ในตัวแปร และเธรดของการเอ็กซิคิวต์ทำให้หยุดด้วย คำสั่งการอินเตอร์รัพท์ซอฟต์แวร์ (SWI) เคอร์เนลทำการเก็บค่าของคอนเท็กซ์และแสดงการ call ด้วยโค้ด เช่น 0x900004 เป็นการ call sys_write เป็นต้น เมื่อเคอร์เนลทำการ call เสร็จสิ้น จะคืนค่ายูสเซอร์คอนเท็กซ์ที่ใช้ในการเอ็กซิคิวต์ โดยรีเทิร์นค่า r0

2.3.4.3 Basic IO Routines

ไมโครซีลีนุกซ์สามารถใช้งานได้กับสถาปัตยกรรมที่มีทรัพยากรที่จำกัดและไม่สนับสนุนการแชร์ไลบรารี แต่ต้องใช้งานรูทีนของไลบรารีได้เป็นอย่างดี รวมทั้งใช้บัฟเฟอร์เป็นสองเท่าเพื่อเพิ่มประสิทธิภาพให้สูงขึ้น

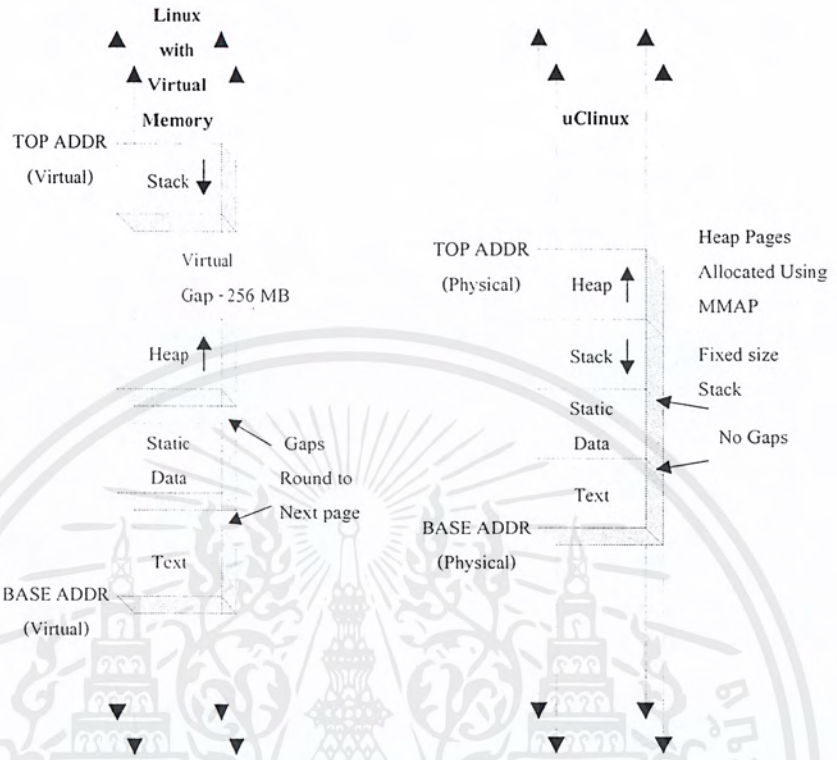
```
@#include <sysdep.h>
@r0 = argc
@r1 = argv
@r2 = envp
@s1 = data segment

        .data
        .align 2
        .global environ, errno
environ:
        .long 0                         @allocate static space for environment pointer
errno:
        .long 0                         @allocate static space for errno
        .text
        .align 2
        .global start, _start, __syscall_error
        .type start, %function
        .type __syscall_error, %function
start:
_start:
        ldr r3, =__data_start           @adjust the data segment base pointer
        sub s1, s2, s3
        ldr r3, .L3                     @load the address of environment
        str r2, [s1, r3]                @store it to the static data
        bl main                         @call the function main
        ldr r0, =0                      @normal exit return 0
.L3:
        .word environ                   @address of label 'environ'
__syscall_error:                        @handle errors during system calls
        ldr r3, .L4                     @load the address of errno
        rsb r2, r0, $0
        str r2, [s1, r3]                @errno=-result
        mvn r0, $0                      @return -1
.L4:
        .word errno                     @address of label 'errno'
```

รูปที่ 2-11 แสดง Program entry code ที่ทำการแฮนเดิล environ ซึ่งส่งมาจาก sys_execve kernel call

2.3.4.4 Heap Allocation

การกำหนดหน่วยความจำแบบไดนามิกจากยูสเซอร์โปรแกรมสามารถกำหนดผ่านการเรียก malloc เพื่อกำหนดพื้นที่ในการจัดเก็บโปรแกรม และเรียกใช้ sbrk เพื่อขยายขนาดข้อมูลของโปรแกรม และจากนั้นแบ่งหน่วยความจำที่ถูกกำหนดไว้เป็นส่วนๆ เนื่องจากหน่วยความจำเสมือนสามารถขยายขนาดของพื้นที่ว่างระหว่างส่วนบนของฮีพ (heap) และส่วนล่างของแสตกได้ แต่ในโมเดลของแฟลชจะไม่มีพื้นที่ว่างให้กับเพจเสมือน ผลที่ตามมาทำให้ heap storage ถูกกำหนดด้วยการ mmap แทน

การใช้ uC-libc ในการกำหนดพื้นที่ว่างทำให้ง่ายในการแฮนเดิลเพจ เนื่องจากมีไลบรารีมากมาย เช่น glibc ซึ่งจัดเตรียมเธรดสำหรับบัฟเฟอร์ แต่ถ้าใช้ sbrk ในการกำหนดก็สามารถใช้ mmap แทนได้เช่นกัน แสดงได้ดังรูป 2-12



รูปที่ 2-12 ไดอะแกรมของ วิชวลโปรแกรมและแฟลชโปรแกรม

2.3.4.5 Stack Sizing

จากรูป 2-12 จะเห็นได้ว่าแสตกจะอยู่ติดกับสแตติกดาต้าดังนั้นต้องมีขนาดเพียงพอที่จะไม่ทับสแตติกดาต้าและพื้นที่ของโค้ด ซึ่งพื้นที่ว่างในลีนุกซ์โปรแกรมอนุญาตให้พื้นที่ของ heap และสแตกขยายใหญ่ได้อย่างไม่จำกัด โดยระบบสามารถตรวจสอบเพจเพื่อป้องกันไม่ให้มีพื้นที่ว่างมากเกินไป

การใช้พื้นที่ว่างในไมโครซีลีนุกซ์ ระบบต้องทำการกำหนดหน่วยความจำจริงให้มากกว่าหน่วยความจำเสมือน เนื่องจากว่าไม่มีการป้องกันหน่วยความจำทำให้ตรวจสอบไม่ได้ว่าสแตกมีขนาดเกินไปหรือเปล่า ดังนั้นควรจะกำหนดขนาดของสแตก และใช้ GDB ในการตรวจสอบระหว่างการรันโปรแกรมเพื่อให้แน่ใจว่ามีพื้นที่เพียงพอในการใช้งาน

2.3.4.6 Flat Memory Support

ความแตกต่างที่เห็นได้ชัดระหว่างไมโครซีลีนุกซ์กับลีนุกซ์คือรูปแบบหน่วยความจำแบบแฟลช ซึ่งโปรแกรมทั้งหมดไม่สามารถแชร์หน่วยความจำในเวลาเดียวกันได้ ในลีนุกซ์ให้หน่วยความจำเสมือนเพื่อจัดเตรียมฟังก์ชันพื้นฐานของ system calls จะมี fork ซึ่งสามารถทำการแชร์เพจเดียวกันได้ โดย fork

ไม่ได้ถูกนำมาใช้งาน ในไมโครซีลีนุกซ์จะใช้ vfork แทน ซึ่ง vfork นั้นแตกต่างจาก fork ตรงที่โปรเซสลูกไม่สามารถแชร์ข้อมูลและสแตกที่ว่างได้เหมือนกับโปรเซสแม่

ในการนำลีนุกซ์ให้สามารถใช้งานร่วมกับสถาปัตยกรรมที่ไม่มีส่วนจัดการหน่วยความจำ ที่จริงมีความยืดหยุ่นในการนำลีนุกซ์มาใช้เป็นอย่างมาก โดยส่วยใหญ่ใช้กับสถาปัตยกรรมเน็ตเวิร์กแบบ embedded ต้องมีการเปลี่ยน API ของลีนุกซ์ให้มีขนาดเล็กลง และพัฒนาในส่วนเกี่ยวข้องต่างๆ ได้แก่เคอร์เนล, ไลบรารี และยูสเซอร์โปรแกรมต้องดัดแปลงให้ใช้ได้กับสถาปัตยกรรมแบบแฟลชเมมโมรีได้ รวมทั้งเครื่องมือที่ใช้ในการพัฒนาต้องมีความยืดหยุ่น และการใช้งานได้เป็นอย่างดีกับสถาปัตยกรรมที่ออกแบบมา

2.4 เกมบอยแอดวานซ์

โปรแกรมที่รันบนเกมบอยแอดวานซ์ (GBA) นั้นปกติแล้วจะถูกบรรจุในตลับ โดย ตลับนั้น โดยหลักแล้วจะเก็บรอม(ROM) และอาจจะมี Cart RAM (SRAM, Flash ROM, EEPROM จะถูกใช้ในการเซฟข้อมูลเกม) ในรอมนั้นจะมีโค้ดที่คอมไพล์แล้ว และข้อมูลต่างๆบรรจุอยู่ ซึ่งไม่เหมือนกับเครื่องพีซี, เวิร์กสเตชัน, เซิฟเวอร์ เพราะว่าตัวเกมบอยแอดวานซ์นั้นไม่ได้มีดิสก์หรือไดร์ฟอื่นๆ ดังนั้นจึงจำเป็นที่จะต้องเก็บทรัพยากรต่างๆ ไว้ในรอมตัวเดียว

โดยวิธีการหลักนั้นโปรแกรมระบุฮาร์ดแวร์สำหรับกราฟฟิก, เสียง และอินพุตเอาท์พุตอื่นๆผ่านทาง เมมโมรีแมปอินพุตเอาท์พุต โดย เมมโมรีแมปอินพุตเอาท์พุต หมายถึง ฟังก์ชันของฮาร์ดแวร์ภายในตัวอย่างเช่น ถ้าต้องการเขียนข้อมูลที่ตำแหน่ง 0x4000000 ด้วยค่า "0x0100" หมายถึงบอกฮาร์ดแวร์ว่า "enabled background 0 and graphic mode 0" โดยวิธีรองลงมาเป็นการผ่านทางไบออสที่ถูกฝังอยู่ในเกมบอยแอดวานซ์ โดยใช้ซอฟต์แวร์อินเตอร์รัพท์ในการที่จะเข้าสู่พรีโปรแกรมรูทีน ในซิสเต็มรอมโดยรูทีนเหล่านี้จะติดต่อฮาร์ดแวร์ผ่านทาง เมมโมรีแมปอินพุตเอาท์พุต

ในเมมโมรีบริเวณอื่นที่ต่อกับฮาร์ดแวร์โดยตรงนั้นมี Palette RAM (เป็นตารางที่เก็บค่าของสีที่ใช้ทั้งหมด) , VRAM (ทำหน้าที่เหมือน video RAM บน PC) และ OAM (เก็บค่าต่างๆสำหรับ hardware accelerated sprite)

โดยองค์ประกอบของเกมบอยแอดวานซ์มีดังนี้

1. ซีพียู ARM7TDMI เป็นซีพียูขนาด 32 บิต และเป็นสถาปัตยกรรมแบบ RISC
2. จอภาพ TFT LCD color non-backlit ความละเอียด 240x160 พิกเซล
3. แรม 32 KB + 256 KB
4. Video memory 96 KB
5. 4 + 2 sound channel
6. Serial communication
7. 10 buttons keypad

2.4.1 ตำแหน่งแอดเดรสต่างๆของเกมบอยแอดวานซ์

ในส่วนนี้จะแสดงถึงบริเวณของหน่วยความจำต่างๆของ GBA และการใช้งาน

2.4.1.1 System ROM

Start : 0x00000000

End : 0x00003FFF

Size : 16 KB

Data bus : 32 bit

เป็นบริเวณที่เก็บ BIOS ไว้ โดยจะสามารถประมวลผลได้อย่างเดียว แต่ไม่สามารถที่จะอ่านค่าได้ โดยการที่จะเข้าไปอ่านข้อมูลในช่วง 0x00000000 ถึง 0x1FFFFFFF จะไม่สามารถอ่านได้

2.4.1.2 External Work RAM

Start : 0x02000000

End : 0x0203FFFF

Size : 256 KB

Data bus : 16 bit

ในบริเวณนี้เป็นพื้นที่สำหรับโค้ดต่างๆและเป็นพื้นที่ที่ใหญ่ที่สุดของแรมที่มีบนเกมบอยแอดวานซ์

2.4.1.3 Internal Work RAM

Start : 0x03000000

End : 0x03007FFF

Size : 32 KB

Data bus : 32 bit

เป็นแรมของเกมบอยแอดวานซ์ที่เร็วที่สุด เพราะว่าเป็นแรมที่ฝังอยู่ในตัว ARM7TDMI และใช้ data bus 32 บิต โดยที่ ROM และ EWRAM นั้นใช้ data bus เพียง 16 บิต เพราะฉะนั้นแรมในตำแหน่งนี้จะมีประสิทธิภาพมากกว่า แต่ไม่สามารถที่จะวางข้อมูลได้มาก เนื่องจากมีความจุน้อย

2.4.1.4 I/O RAM

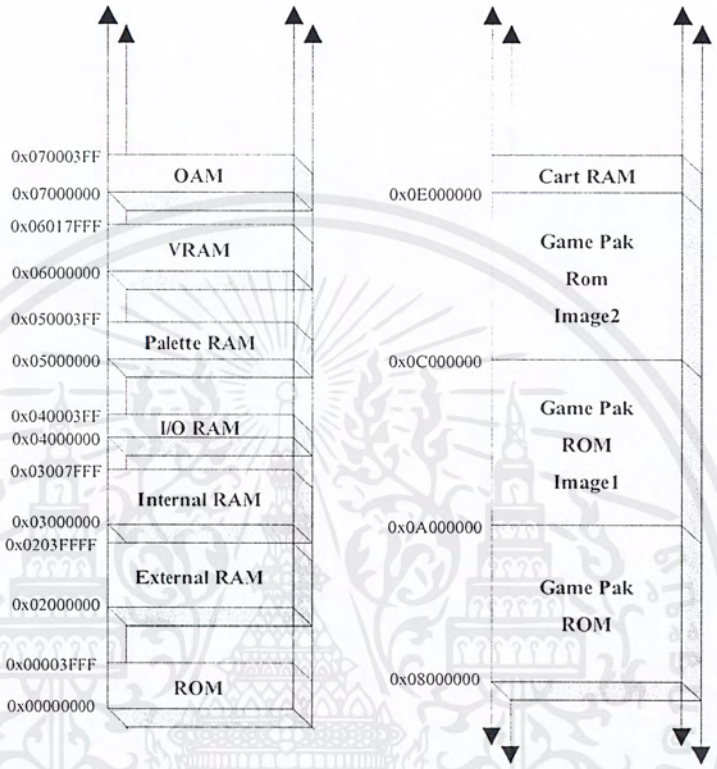
Start : 0x04000000

End : 0x040003FF

Size : 1 KB

Data bus : dual 16 bit

เป็นบริเวณที่เก็บ ASIC (Application Specific Integrated Circuit) registers ซึ่งจะเก็บค่าสถานะต่างๆของฮาร์ดแวร์ของเกมบอยแอดวานซ์และไว้ควบคุมฮาร์ดแวร์ต่างๆ เช่น graphics, sound, DMA และส่วนอื่นๆ



รูปที่ 2-13 แสดงตำแหน่งแอดเดรสของส่วนต่างๆในเกมบอยแอดวานซ์

2.4.1.5 Palette RAM

Start : 0x05000000

End : 0x050003FF

Size : 1 KB

Data bus : 16 bit

เป็นบริเวณที่กำหนดค่า 16-bit color สำหรับ paletted modes โดยจะมีพื้นที่อยู่ 2 ตำแหน่งสำหรับ paletted นั่นคือ สำหรับ backgrounds (0x05000000) และ สำหรับส่วนอื่นสำหรับ sprites (0x05000200)

2.4.1.6 VRAM

Start : 0x06000000

End : 0x06017FFF

Size : 96 KB

Data bus : 16 bit

Video RAM ใช้เก็บเฟรมบัฟเฟอร์ใน bitmapped mode และเก็บ tile data และ tile maps สำหรับ tiled-based text และ rotate/scale mode

2.4.1.7 OAM

Start : 0x07000000

End : 0x070003FF

Size : 1 KB

Data bus : 32 bit

Object Attribute Memory ใช้ควบคุม GBA's sprites

2.4.1.8 GAME PAK ROM

Start : 0x08000000

Size : The size of the cartridge (0 - 32 Megabytes)

Data bus :16 bit

เป็นตำแหน่งของตลับเกม

2.4.1.9 GAME PAK ROM IMAGE 1

Start : 0x0A000000

Size : The size of the cartridge (0 - 32 Megabytes)

Data bus :16 bit

Wait state : 1

เป็นตำแหน่งของตลับเกมเช่นกัน แต่จะสามารถดึงข่อมูลไปเอกซิคิวต์ได้เมื่อ เกิด wait จากตำแหน่งข้างบน

2.4.1.10 GAME PAK ROM IMAGE 2

Start : 0x0C000000

Size : The size of the cartridge (0 - 32 Megabytes)

Data bus :16 bit

Wait state : 2

เหมือนกับตำแหน่งข้างบน

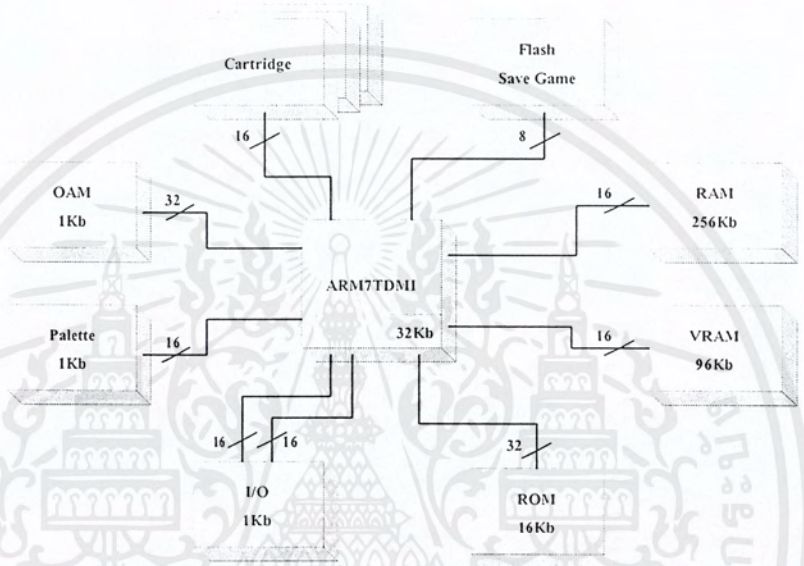
2.4.1.11 CART RAM

Start : 0x0E000000

Size : 0-64 KB

Data bus :8 bit

เป็นตำแหน่งในการบันทึกข้อมูลลงตลับ ซึ่งอาจจะเป็นแบบ SRAM หรือ Flash ROM ก็ได้



รูปที่ 2-14 แสดงขนาดบัสของส่วนต่างๆของเกมบอยแอดวานซ์

2.5 Xport

เป็นอุปกรณ์ที่ใช้เชื่อมต่อกับเครื่องเกมบอยแอดวานซ์ทางช่องเสียบตลับเกมของเครื่องเกมบอยแอดวานซ์ ซึ่ง Xport มีความสามารถในการเก็บข้อมูลที่สามารถทำงานได้ คล้ายตลับเกมแต่มีประโยชน์มากกว่าตลับเกมทั่วไปมาก

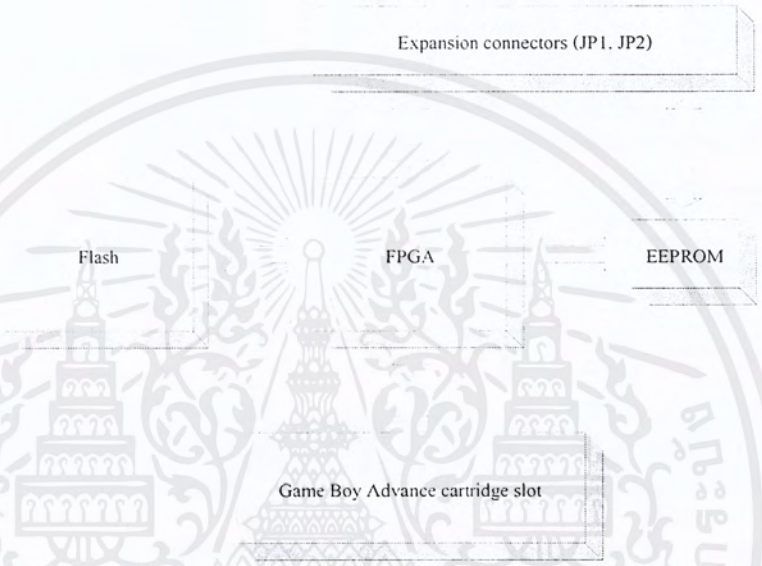
2.5.1 ประโยชน์และความสามารถของ Xport

1. สามารถควบคุมสัญญาณอินพุทและเอาท์พุทได้ 55 สัญญาณ
2. 4 เมกกะไบต์ (32 เมกกะบิต) เป็นหน่วยความจำแบบแฟลช
3. มี FPGA ที่สามารถโปรแกรมได้ 10,000 logic gates
4. มี Xport programmer ช่วยในการเชื่อมต่อ Xport กับเครื่องคอมพิวเตอร์
5. มีปุ่มรีเซตช่วยในการพัฒนาต่างๆ
6. ใช้พลังงานน้อย

7. รองรับระบบปฏิบัติการ eCos

2.5.2 หลักการทำงานของ Xport

ดังที่แสดงในรูปที่ 2-15 จะแสดงถึงแผนภาพอธิบายถึงการทำงานของ Xport ซึ่งมีส่วนประกอบหลักๆได้แก่ FPGA, หน่วยความจำแบบแฟลช และ EEPROM โดย FPGA จะใช้ SRAM ในการเก็บค่าคุณสมบัติต่างๆของ FPGA ขณะใช้งาน ส่วน EEPROM จะเก็บค่าข้อมูลคุณสมบัติต่างๆเมื่อมีการเปิดปิดเครื่องเกมบอยแอดวานซ์ ทำให้เมื่อมีการโปรแกรม EEPROM แล้วก็จะมีผลต่อ FPGA ไปด้วย



รูปที่ 2-15 บล็อกไดอะแกรมของตลับ Xport

2.5.3 ความสามารถของ Xport

ข้อแตกต่างระหว่างตัวเครื่องเกมบอยแอดวานซ์กับตลับเกมบอย นั้นคือที่ตัวเครื่องเกมบอยแอดวานซ์นั้นจะมีช่องเสียบตลับขนาด 16 บิต ส่วนตลับเกมบอยนั้นจะมีขนาด 8 บิตโดยที่ขาที่เชื่อมกันนั้นจะเป็นตำแหน่งในทางกายภาพเดียวกัน ดังนั้นการที่จะทำให้สัญญาณต่างๆสามารถทำงานได้ ที่เครื่องเกมบอยแอดวานซ์จะต้องทำการมัลติเพล็กแอดเดรส และสัญญาณข้อมูล (data signals) จึงทำให้ไม่สามารถเข้าไปจัดการข้อมูลภายในตลับได้โดยตรง แต่มีความสามารถในการถ่ายข้อมูลจากหน่วยความจำแบบแฟลชมายังตัวเครื่องเกมบอยแอดวานซ์ ดังนั้นระบบมัลติเพล็กแอดเดรสจึงมีความจำเป็นในการส่งข้อมูลเข้าสู่ตัวเครื่องเกมบอยแอดวานซ์

อย่างไรก็ตามการทำมัลติเพล็กนั้นจะใช้ FPGA's logic gates เพียงเล็กน้อยเท่านั้น ส่วนที่เหลือทั้งหมดสามารถโปรแกรมให้เป็นไปตามต้องการได้ ซึ่งโปรแกรมต่างๆที่ใช้ประโยชน์จาก FPGA นั้นแบ่งออกเป็น 2 กลุ่มด้วยกัน คือ โปรแกรมที่ไม่ต้องใช้ อินพุท/เอาท์พุท กับโปรแกรมที่ต้องการใช้ประโยชน์จาก อินพุท/เอาท์พุท

2.6 eCos

eCos : embedded Configurable operating system เป็นระบบปฏิบัติการที่มีการเปิดเผยข้อมูลคำสั่งต้นฉบับ (Source code) ซึ่งได้รับการสนับสนุนจาก GNU ทำให้นักพัฒนาสามารถเข้าใจระบบปฏิบัติการด้วยการศึกษาข้อมูลคำสั่ง (code) ที่มีการเปิดเผย และทำได้แม้แต่แก้ไขข้อมูลคำสั่งต่างๆ เพื่อให้ได้มาซึ่งระบบปฏิบัติการที่นักพัฒนาต้องการ อีกทั้งยังสามารถพัฒนาโปรแกรมภายใต้ระบบปฏิบัติการได้อย่างมีอิสรภาพอีกด้วย

สิ่งที่ทำให้ระบบปฏิบัติการ eCos นั้นแตกต่างจากระบบปฏิบัติการที่ใช้สำหรับระบบฝังตัวอื่นๆ คือระบบปฏิบัติการ eCos นั้นจะสามารถกำหนดค่าคุณสมบัติ (Configurable) จึงทำให้นักพัฒนาสามารถกำหนดความสามารถของระบบปฏิบัติการ eCos ได้เพื่อให้ระบบปฏิบัติการที่มีขนาดเล็กที่สุดเท่าที่จะสามารถทำงานได้ เนื่องจากโปรแกรมที่นำมาใช้งานในระบบฝังตัวนั้นจะเป็นโปรแกรมที่ใช้งานเฉพาะทาง ซึ่งบางโปรแกรมไม่มีความจำเป็นจะต้องใช้ประโยชน์ที่มีในระบบปฏิบัติการทั้งหมด ดังนั้นการที่ระบบปฏิบัติการสามารถกำหนดค่าคุณสมบัติได้นั้น จะเป็นประโยชน์ต่อนักพัฒนาที่ใช้ระบบฝังตัวซึ่งมีหน่วยความจำจำกัด

ระบบปฏิบัติการ eCos ต้องการหน่วยความจำเพียง 10 – 100 กิโลไบต์ซึ่งขนาดของหน่วยความจำที่ระบบปฏิบัติการต้องการจะเปลี่ยนแปลงไปตามแต่การกำหนดค่าคุณสมบัติ ในขณะที่ระบบปฏิบัติการลีนุกซ์ ในปัจจุบันนั้นจะต้องใช้ขนาดหน่วยความจำถึง 500 กิโลไบต์สำหรับเคอร์เนล (Kernel) อีกทั้งยังต้องการขนาดของแรม (RAM) อีก 1.5 เมกกะไบต์ ในการใช้ทำงานของโปรแกรมต่างๆซึ่งไม่เหมาะที่จะนำมาทำงานในระบบฝังตัว

ระบบปฏิบัติการ eCos ถูกออกแบบมาให้รองรับโปรแกรมที่ต้องการการทำงานแบบเรียลไทม์ (Real-time), มีอินเตอร์รัพท์ขนาดเล็กที่แฝงอยู่ภายใน, มีการจัดการ schedule อีกทั้งมีการออกแบบฟังก์ชันต่างๆที่จำเป็นในการสร้างโปรแกรมที่ทำงานบนอุปกรณ์ระบบฝังตัว ซึ่งฟังก์ชันต่างๆ นี้ประกอบไปด้วยฟังก์ชันที่จัดการ device drivers, file system, TCP/IP, memory management, etc.

2.6.1 ฟังก์ชันการทำงานของ eCos

ระบบปฏิบัติการ eCos มีฟังก์ชันหลักๆ ดังนี้

1. Hardware Abstraction Layer (HAL) เป็นส่วนที่จะมีความแตกต่างกันตามแต่อุปกรณ์ที่ระบบปฏิบัติการ eCos จะไปทำงาน หรือเป็นส่วนที่ขึ้นกับฮาร์ดแวร์
2. Kernel เป็นส่วนที่ทำหน้าที่จัดการส่วนต่างๆที่เกี่ยวข้องกับเคอร์เนล ไม่ว่าจะเป็น Interrupt, Exception, Scheduler, Thread, Memory management, etc.
3. ISO C and Math libraries เป็นส่วนเฮดเดอร์ต่างๆของการเขียนโปรแกรม เพื่อช่วยในการเขียนโปรแกรมต่างๆ
4. Device drivers มาตรฐาน ซึ่งช่วยในการสร้างหรือเขียนโปรแกรมเนื่องจากมี driver มาตรฐานมาให้เรียบร้อยแล้ว อันได้แก่ Serial, Ethernet, etc.

5. USB มีความสามารถรองรับอุปกรณ์ที่ใช้ USB
6. TCP/IP สแตก มีโปรโตคอลการเชื่อมต่อมาตรฐาน
7. GDB มีความสามารถรองรับวิธีการแก้ไขโปรแกรมมาตรฐานของ GNU

2.6.1.1 Hardware Abstraction Layer (HAL)

เป็นส่วนที่กำหนดไว้ตั้งแต่เริ่มแรกของ eCos โดยส่วนสำคัญของ HAL จะขึ้นอยู่กับฮาร์ดแวร์ของโครงสร้างหน่วยประมวลผล หรือ โครงสร้างของแพลตฟอร์มที่จะใช้งานระบบปฏิบัติการ eCos

HAL แบ่งออกเป็นส่วนย่อยๆได้ 3 ส่วนได้แก่

- architecture เป็นส่วนที่เกี่ยวกับโครงสร้างพื้นฐานเช่นซีพียู รวมไปถึงอินเตอร์รัพท์ต่างๆ
- platform เป็นส่วนที่เกี่ยวกับแพลตฟอร์ม เช่น registers I/O รวมไปถึง device ต่างๆ
- implementation เป็นส่วนที่ระหว่าง 2 ส่วน เช่น on-chip devices

HAL ทั้ง 3 ส่วนนี้การแบ่งแยกยังไม่แน่ชัดเนื่องจากมีความใกล้เคียงกันมาก

กฎเกณฑ์ทั่วไปของการสร้าง HAL

- ในส่วนของการ implement นั้นให้ใช้ภาษาซี หรือ แอสแซมบลี
- ในส่วนของ interface นั้นให้ใช้ มาโคร CPP ในการสร้าง
- HAL จะต้องเปิดช่องทางให้ใช้งานได้ง่าย ทำให้การเปลี่ยน platform สามารถทำได้ง่าย

2.6.1.2 เคอร์เนล

หัวใจหลักของเคอร์เนล คือ scheduler ซึ่งเป็นตัวกำหนดว่าเธรดต่างๆจะทำงานอย่างไรซึ่งรวมไปถึงอินเตอร์รัพท์ต่างๆ ที่ใช้สำหรับเธรดอีกด้วย

2.6.1.3 ISO C และ Math libraries

สำหรับระบบปฏิบัติการ eCos นั้นถูกออกแบบมาให้รองรับภาษาซีมาตรฐาน ISO 9899:1990 โดยระบบปฏิบัติการ eCos จะมีอยู่ 3 รูปแบบได้แก่ C library, Math library, library ที่ขึ้นกับสภาพแวดล้อม

2.6.1.4 Device drivers

ในการกำหนดค่าคุณสมบัติต่างๆของระบบปฏิบัติการนั้นสามารถกำหนด device drivers เข้าสู่ระบบได้เหมือนกับส่วนอื่นๆ และสามารถเพิ่มเติมส่วนที่ไม่เป็นมาตรฐานได้อีกด้วย

ระบบปฏิบัติการ eCos ถูกออกแบบมาให้สามารถกำหนดค่าคุณสมบัติ และทำการพัฒนาตัวระบบปฏิบัติการได้ทั้งบนสภาพแวดล้อมของลีนุกซ์ และ วินโดวส์ โดยสามารถทำงานกับลีนุกซ์ที่มีอยู่ใน

ปัจจุบัน ส่วนวินโดวส์นั้นสามารถใช้งานได้ทุกรุ่น แต่รุ่นที่แนะนำคือ Windows 2000 เนื่องจากรุ่นอื่นๆ นั้นยังไม่มีเสถียรภาพที่ดีพอ

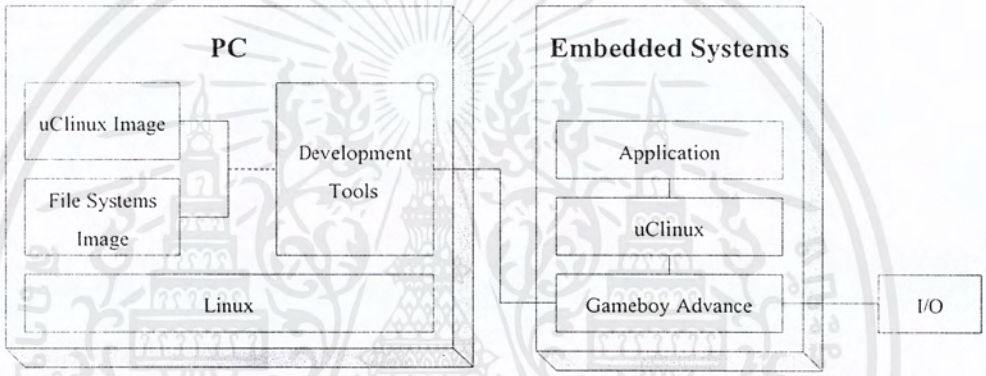
2.6.2 การใช้งาน eCos

เมื่อทำการ คอมไพล์ ระบบปฏิบัติการ eCos แล้วสิ่งที่จะได้จากการคอมไพล์ คือ library.a ซึ่งเป็นส่วนที่ระบบปฏิบัติการ eCos ต่างจากระบบปฏิบัติการอื่นๆ เนื่องจากระบบปฏิบัติการอื่นๆนั้นเมื่อทำการคอมไพล์จะได้ข้อมูลที่เป็น data กับ image เพื่อใช้ในการทำงานของระบบปฏิบัติการนั้นๆ แต่สำหรับระบบปฏิบัติการ eCos แล้วจะได้ library.a ซึ่งจะเป็นการรวมเอาความสามารถต่างๆของระบบปฏิบัติการที่ได้กำหนดค่าคุณสมบัติก่อนการคอมไพล์ ดังนั้นระบบปฏิบัติการ eCos นี้จะยังไม่สามารถทำงานได้ถ้ายังไม่มีโปรแกรมเรียกใช้

บทที่ 3
การวิเคราะห์และออกแบบ

3.1 ภาพรวมของโครงงาน

โครงงานลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์เป็นโครงงานที่นำไมโครลีนุกซ์สามารถทำงานบนระบบ embedded ซึ่งเป็นเครื่องเกมบอยแอดวานซ์เพื่อเพิ่มความสามารถให้กับเครื่องเกมบอยแอดวานซ์ และสามารถสร้างแอพพลิเคชันที่ทำงานบนเกมบอยแอดวานซ์โดยที่เป็นแอพพลิเคชันที่ทำงานบนระบบปฏิบัติการ



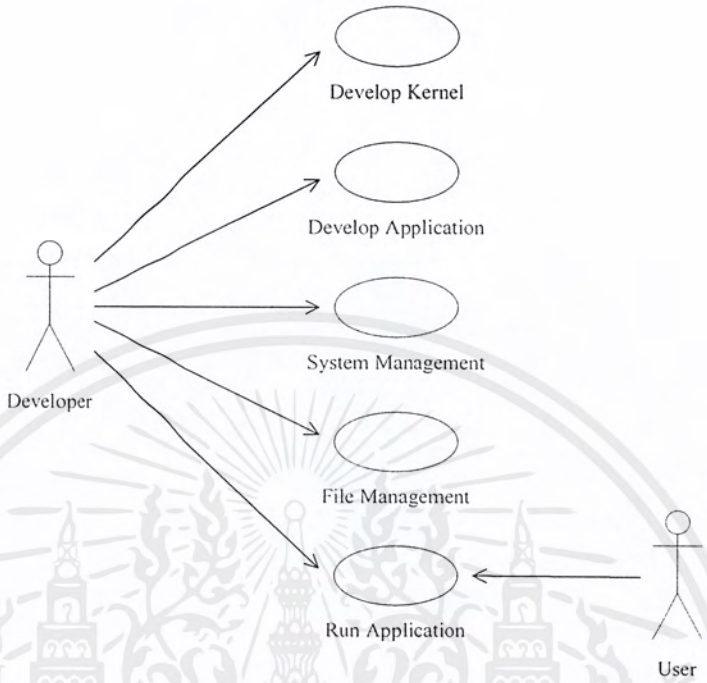
รูปที่ 3-1 ภาพรวมโครงสร้างของระบบ

จากภาพรวมของโครงงานที่แสดงในรูปที่ 3-1 มีส่วนประกอบที่ต้องใช้ในระบบอยู่หลายส่วน โดยเฉพาะในส่วนที่แรเงาจำเป็นส่วนที่จำเป็นต้องมีการวิเคราะห์เพื่อนำมาใช้กับระบบ ซึ่งสามารถแบ่งการวิเคราะห์ได้เป็น 2 ส่วนที่สำคัญ คือ

1. ไมโครซีลีนุกซ์
2. เกมบอยแอดวานซ์

3.2 Use Case Diagram

แสดงให้เห็นถึงฟังก์ชันการทำงานหลักที่สามารถใช้งานได้โดยผู้ใช้งาน ซึ่งผู้ใช้ส่วนใหญ่คือผู้พัฒนาระบบ (developer) และผู้ใช้ (user) ซึ่งมีทั้งหมด 5 ฟังก์ชันหลักดังรูปที่ 3-2

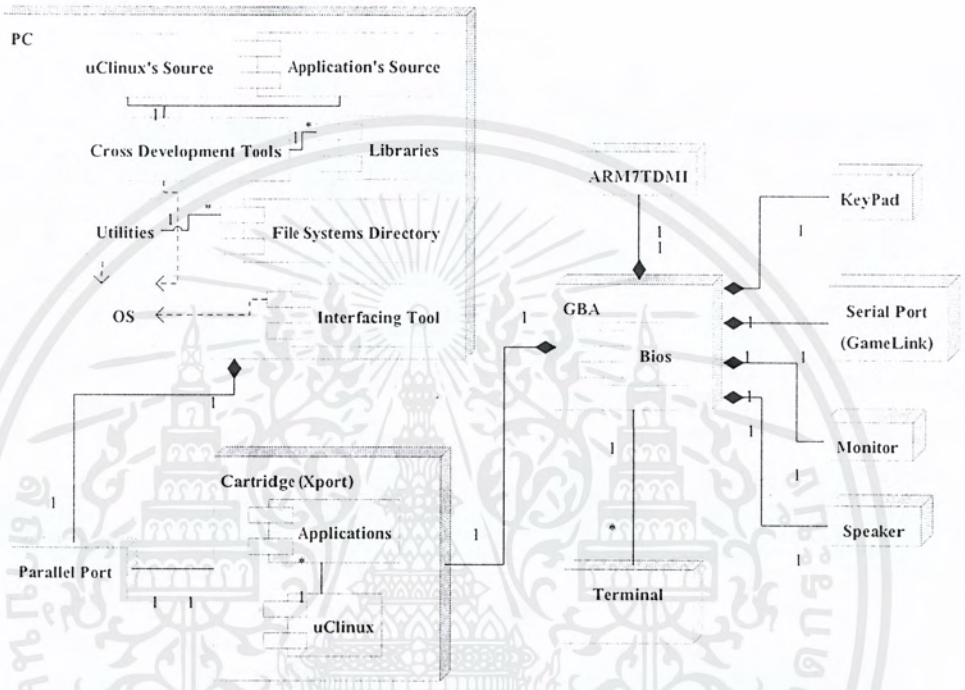


รูปที่ 3-2 แสดง Use Case Diagram ของระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

1. Develop Kernel คือ การปรับปรุงแก้ไขและคอนฟิกไมโครซีลีนุกซ์บนเครื่องพีซี เพื่อนำไปใช้งานบนเกมบอยแอดวานซ์ ซึ่งเป็นส่วนที่สามารถปรับแต่งตามแต่ที่นักพัฒนาต้องการไม่ว่าจะเป็นการเพิ่มคำสั่ง หรือลบคำสั่งที่ไม่ต้องการ
2. Develop Application คือ การพัฒนาแอพพลิเคชันที่จะนำมาทำงานอยู่บนระบบปฏิบัติการไมโครซีลีนุกซ์ที่ได้รับการปรับแต่ง โดยการพัฒนาแอพพลิเคชันนี้จะต้องอยู่บนข้อจำกัดของฮาร์ดแวร์ และความสามารถของเคอร์เนลที่ถูกปรับแต่งมา เพื่อทำงานอยู่บนเครื่องเกมบอยแอดวานซ์
3. System Management คือ การจัดการซิสเต็มของระบบ รวมทั้งการดูโปรเซสการทำงาน, การหยุดโปรเซสที่ทำงานอยู่, การจัดการสภาพแวดล้อมต่าง ๆลฯ
4. File Management คือ การจัดการไฟล์ที่อยู่ในระบบ ไม่ว่าจะเป็นการเพิ่มไฟล์, ลบไฟล์ ฯลฯ
5. Run Application คือ การที่สั่งให้แอพพลิเคชันต่างๆที่ได้พัฒนาขึ้นมาทำงาน รวมทั้งการอินพุต เอาต์พุตผ่านสู่ระบบ ซึ่งเป็นที่สามารถทำได้ เป็นทั้งนักพัฒนา และผู้ใช้ทั่วไป

3.3 โครงสร้างของระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

จากการศึกษาค้นคว้าและทำการวิเคราะห์เพื่อกาส่วนต่างๆที่จำเป็นที่ต้องนำมาใช้ในระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์ในรูปที่ 3-1 สามารถแสดงรายละเอียดโครงสร้างของระบบทั้งหมดได้ดังรูปที่ 3-4



รูปที่ 3-3 โครงสร้างของระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

ในรูปที่ 3-4 แสดงให้เห็นในถึงระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์ซึ่งได้แบ่งออกได้เป็น 3 ส่วน คือ ส่วนที่ใช้งานบนเครื่องพีซี, ส่วนที่ทำงานตลับเกมบอย และส่วนที่ทำงานบนเครื่องเกมบอยแอดวานซ์

3.3.1 ส่วนที่ทำงานบนเครื่องพีซี

ส่วนที่ใช้งานบนเครื่องพีซีเป็นส่วนที่ใช้ในการพัฒนาไมโครซีลีนุกซ์และแอพพลิเคชันที่จะนำมาใช้งาน โดยประกอบด้วยส่วนต่างๆเหล่านี้

1. PC คือเครื่องคอมพิวเตอร์ส่วนบุคคลที่นำมาใช้ในการพัฒนา
2. Parallel Port คือพอร์ตขนานบนพีซี สำหรับเชื่อมต่อกับตลับ Xport ในการอัพโหลดข้อมูลทั้งไมโครซีลีนุกซ์ และแอพพลิเคชันที่จะนำไปทำงานบนเครื่องเกมบอยแอดวานซ์

3. OS คือระบบปฏิบัติการที่นำมาพัฒนาจะมีทั้งระบบปฏิบัติการลีนุกซ์และระบบปฏิบัติการวินโดวส์ ซึ่งโปรแกรมที่ใช้ในการพัฒนาทุกตัวนั้นจะทำงานอยู่บนลีนุกซ์ แต่ในการอัพโหลดข้อมูลจากเครื่องพีซีไปยังตลับ Xport นั้น แอพพลิเคชันที่จะทำการอัพโหลดข้อมูลนั้นจะสามารถทำงานได้บนปฏิบัติการวินโดวส์เท่านั้น ทั้งนี้เนื่องจากเป็นแอพพลิเคชันทางผู้ผลิตตลับ Xport
4. uClinux's Source คือโค้ดทั้งหมดของไมโครซีลีนุกซ์ที่สามารถนำไปพัฒนาต่อ หรือคอมไพล์เพื่อนำไปทำงานบนเครื่องเกมบอยแอดวานซ์
5. Application's Sources คือโค้ดของแอพพลิเคชันที่จะนำไปทำงานกับไมโครซีลีนุกซ์บนเครื่องเกมบอยแอดวานซ์
6. Libraries คือไลบรารีพื้นฐานสำหรับพัฒนาแอพพลิเคชันบนไมโครซีลีนุกซ์ ได้แก่ uClibc
7. Cross Development Tools คือ ชุดโปรแกรมในการคอมไพล์ไมโครซีลีนุกซ์และแอพพลิเคชัน โดยชุดโปรแกรมนี้นำมาจาก Cross Compiler for ARM7TDMI (www.uclinux.org) ซึ่งประกอบไปด้วย
   (1) Binutils คือชุดโปรแกรมอรรถประโยชน์ (utilities) ในการจัดการกับไลบรารี่
   (2) GCC คือชุดของโปรแกรมคอมไพเลอร์ภาษา C, C++, Object C, Fortran และอื่นๆ
   (3) newlib คือชุดเฮดเดอร์ไฟล์ที่ใช้ร่วมกับ GCC
8. Utilities คือชุดโปรแกรมอรรถประโยชน์ ประกอบด้วย
   (1) elf2flt คือโปรแกรมที่ทำหน้าที่แปลงไบนารีไฟล์ที่ได้จากการคอมไพล์ในรูปแบบ ELF ไปเป็นแบบแฟลชเพื่อให้ใช้งานบนไมโครซีลีนุกซ์ได้
   (2) genromfs คือโปรแกรมสร้างอิมเมจของ File Systems จากไดเรกทอรี่เพื่อนำไปใช้เป็น Root File Systems ของไมโครซีลีนุกซ์
9. Files Systems Directory คือไดเรกทอรี่ที่ประกอบด้วยไฟล์และไดเรกทอรี่ในการสร้าง Root File Systems ด้วยโปรแกรม genromfs สำหรับใช้งานบนไมโครซีลีนุกซ์ ซึ่งแอพพลิเคชันที่พัฒนามาทำงานกับไมโครซีลีนุกซ์จะอยู่ภายในไดเรกทอรี่นี้ด้วย
10. Interfacing Tool คือ โปรแกรมในการอัพโหลดโปรแกรมและข้อมูลผ่านทางพอร์ตขนานกับตลับ Xport ซึ่งเรียกโปรแกรมนี้ว่า Xport Tool

3.3.2 ส่วนที่ทำงานบนตลับ Xport

ส่วนของตลับ Xport นั้นจะบรรจุระบบปฏิบัติการไมโครซีลีนุกซ์ และอาจจะรวมทั้งแอพพลิเคชันที่จะนำมาทำงานบนเกมบอยแอดวานซ์ ดังนี้

1. uClinux คือระบบปฏิบัติการไมโครซีลีนุกซ์ที่ได้รับการปรับแต่งให้ทำงานเข้ากับระบบของเกมบอยแอดวานซ์ โดยจะมีคำสั่งให้เรียกใช้งานตามที่ถูกปรับแต่งมา

2. Applications คือแอพพลิเคชันที่จะทำไปทำงานบนเครื่องเกมบอยแอดวานซ์ ซึ่งแอพพลิเคชันนั้นๆจะทำงานได้ ต้องมีความเข้ากับกับระบบปฏิบัติการไมโครซีลีนุกซ์ที่ได้ปรับแต่งมาแล้วด้วย

3.3.3 ส่วนที่ทำงานบนเครื่องเกมบอยแอดวานซ์

หลังจากที่ได้สร้างไมโครซีลีนุกซ์และแอพพลิเคชันที่จะทำงานบนเกมบอยแอดวานซ์แล้วจากส่วนที่ใช้งานบนเครื่องพีซีนั้น เมื่อนำมาทำงานบนเครื่งเกมบอยแอดวานซ์ จะประกอบไปด้วยส่วนต่างๆ ดังต่อไปนี้

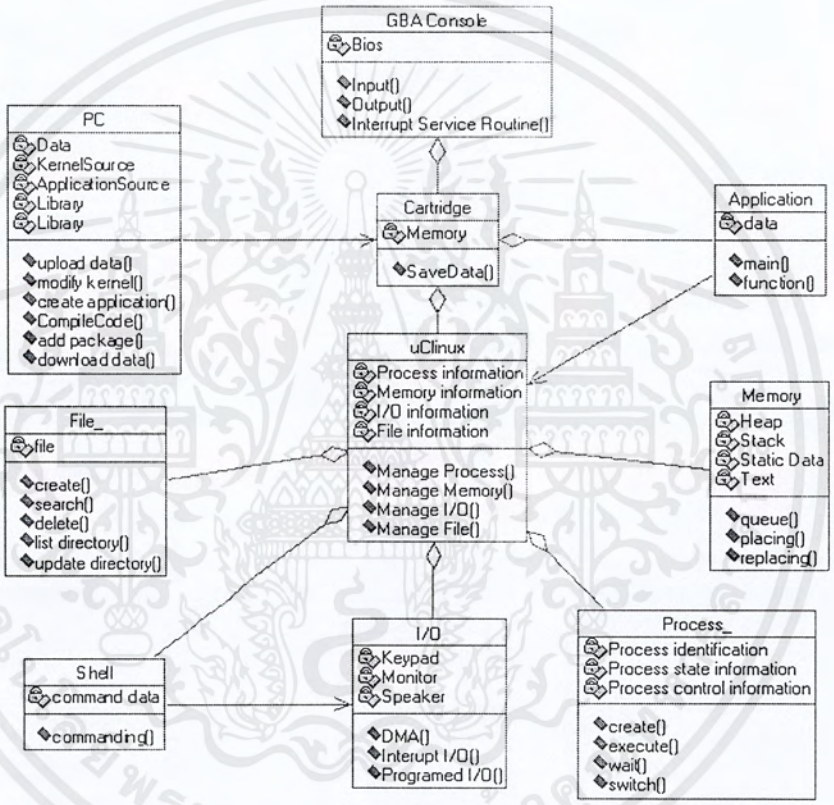
1. Bios คือส่วนที่จัดการอินเตอร์รัพท์เซอร์วิสรูทีน (SWI) และค่างต่างๆซึ่งเหมือนกัน Bios ในระบบคอมพิวเตอร์ และจะเป็นตัวพิจารณาว่าจะนำข้อมูลที่แอดเดรสของตลับ หรือของเมมโมรีมาทำงาน
2. ARM7TDMI คือซีพียูส่วนกลางของเกมบอยแอดวานซ์
3. KeyPad คือปุ่มกดของเครื่องเกมบอยแอดวานซ์ซึ่งมีอยู่ทั้งหมด 10 ปุ่มด้วยกัน
4. Serial Port คือพอร์ตที่ติดต่อกันระหว่างเครื่องเกมบอยแอดวานซ์ด้วยกันเอง และยังสามารถนำมาส่งและรับข้อมูลจากเครื่องพีซีได้โดยตรง
5. Monitor คือจอภาพของเกมบอยแอดวานซ์ซึ่งจะแสดงผลของเอาท์พุตต่างๆ
6. Speaker คือลำโพงที่ติดอยู่กับเครื่องเกมบอยแอดวานซ์ ที่จัดการแสดงเอาท์พุตที่เป็นเสียง
7. Terminal คือหน้าจอที่แสดงการทำงานของอุปกรณ์ต่างๆที่อาจจะนำเกมบอยแอดวานซ์ไปเป็นตัวเทอร์มินอลกับระบบอื่นๆได้
8. Cartridge คือส่วนของตลับเกมที่บรรจุข้อมูลที่จะนำมาทำงานบนเครื่องเกมบอยแอดวานซ์ ซึ่งในที่นี้ก็คือตลับ Xport นั่นเอง

3.4 Class Diagram

รูปที่ 3-4 แสดงคลาสไดอะแกรมของระบบทั้งหมด ซึ่งมีรายละเอียดดังนี้

1. PC เป็นคลาสที่เก็บโค้ดต่างๆ และเป็นที่ปรับแต่งแก้ไขตัวเคอร์เนลและแอพพลิเคชันที่จะทำงานบนไมโครซีลีนุกซ์ รวมทั้งไลบรารีต่างๆ โดยมีฟังก์การทำงานหลักๆ คือการคอมไพล์ , แก้ไข, เพิ่มเติม เคอร์เนล ซึ่งรวมถึงการอัพโหลดดาต้าสู่ตลับ X port ด้วย
2. Cartridge เป็นคลาสของตัวตลับ Xport ซึ่งจากไดอะแกรมจะเห็นว่ามีคลาส uClinux และคลาส Application รวมอยู่ในตลับนั่นเอง ซึ่งตลับนี้ก็จะเป็นส่วนหนึ่งของ Gameboy Console โดยมีแอททริบิวต์ Memory เป็นตัวเก็บข้อมูล
3. Application เป็นคลาสที่เก็บแอพพลิเคชันที่นักพัฒนาได้สร้างขึ้นมาให้ทำงานบนไมโครซีลีนุกซ์ โดยมีโอเปเรชัน เป็นฟังก์ชันให้เรียกใช้
4. uClinux คลาสนี้เป็นคลาสหลักที่เก็บระบบปฏิบัติการทั้งหมดซึ่งจะมีการทำงานหลักๆคือการจัดการโปรเซส, การจัดการไฟล์, การจัดการเมมโมรี, และอินพุตเอาท์พุต

5. Memory คลาสเป็นคลาสที่จัดการเกี่ยวกับเมมโมรี, ฮีป, สแตก, สแตติกดาต้า ซึ่งถูกรวมอยู่ในคลาส uClinux
6. File คลาสนี้เป็นคลาสที่จัดการเกี่ยวกับไฟล์ซิสเต็มต่างๆ ซึ่งถูกรวมอยู่ในคลาส uClinux
7. I/O คลาสนี้จัดการเกี่ยวกับอินพุตเอาท์พุตต่างๆทั้งจอมอนิเตอร์, คีย์แพด, ลำโพง โดยถูกรวมอยู่ในคลาส uClinux เช่นกัน
8. Shell คลาสนี้เป็น Interface ในการส่งคำสั่งสู่คลาส uClinux ซึ่งจะเรียกใช้บริการของคลาส I/O ในการอินพุต



รูปที่ 3-4 คลาสไดอะแกรมของระบบลีนุกซ์ฝังตัวบนเกมบอยแอดวานซ์

3.5 ไมโครซีลีนุกซ์ที่นำมาใช้กับโครงการ

เนื่องจากโครงงานนี้เป็นการนำไมโครซีลีนุกซ์พอร์ตลงสู่เครื่องเกมบอยแอดวานซ์ ซึ่งเป็นแพลตฟอร์มที่ใหม่ จึงจะต้องทำการเปลี่ยนแปลงโค้ดใหม่เกือบทั้งหมด ซึ่งแพลตฟอร์มที่ใกล้เคียงก็จะมีเช่นแพลตฟอร์มของบอร์ด ATMEL AT91EB01 ที่พอจะสามารถนำมาเทียบเคียงในการปรับแต่งได้ โดยศึกษาจากขั้นตอนและลำดับการทำงานของไมโครซีลีนุกซ์โดยอ้างอิงจากแพลตฟอร์มอื่นๆ และดูส่วนที่

จำเป็นต้องเปลี่ยนแปลงโดยดูความถูกต้องของความเข้ากันของโค้ดที่เราได้แก้ไขกับฮาร์ดแวร์ของเกมบอยแอดวานซ์

3.6 ไมโครซีลีนุกซ์กับเครื่องเกมบอยแอดวานซ์

ในการนำเอาไมโครซีลีนุกซ์มาพัฒนาแก้ไขให้ทำงานให้เข้ากับเครื่องเกมบอยแอดวานซ์นั้นต้องต้องมีการแก้ไขและเพิ่มเติมไฟล์ต่างๆดังที่จะแสดงต่อไป โดยวิเคราะห์จากไมโครซีลีนุกซ์ที่ทำงานบนแพลตฟอร์มที่ใกล้เคียง และแนวทางการพอร์ตไมโครซีลีนุกซ์สำหรับฮาร์ดแวร์ใหม่ ซึ่งในรูปที่ 3-5 เป็นการแสดงไฟล์หลักที่ได้ถูกแก้ไข และเพิ่มเติมโค้ด

```
Makefile
arch/armnommu/Makefile
arch/armnommu/Config
arch/armnommu/kernel/head-arm-gba.S
arch/armnommu/kernel/setup.c
arch/armnommu/kernel/time.c
arch/armnommu/lib/io-gba.c
arch/armnommu/lib/irqs-gba.c
include/asm-armnommu/arch-gba/a.out.h
include/asm-armnommu/arch-gba/dma.h
include/asm-armnommu/arch-gba/hardware.h
include/asm-armnommu/arch-gba/io.h
include/asm-armnommu/arch-gba/irq.h
include/asm-armnommu/arch-gba/irqs.h
include/asm-armnommu/arch-gba/mmu.h
include/asm-armnommu/arch-gba/processor.h
include/asm-armnommu/arch-gba/system.h
include/asm-armnommu/arch-gba/time.h
include/asm-armnommu/proc-armv/page.h
include/linux/string.h
vendors/Nintendo/GBA/Makefile
vendors/Nintendo/GBA/config.arch
```

รูปที่ 3-5 แสดงรายชื่อไฟล์ที่ได้แก้ไขเพิ่มเติม

```
ifeq ($(CONFIG_ARCH_GBA),y)
HEAD            := arch/armnommu/kernel/head-arm-gba.o
MACHINE         = gba
TEXTADDR        = 0x08000000
ARCHLDFLAGS     = -T $(TOPDIR)/arch/armnommu/vmlinux-armv.lds.in
endif
```

รูปที่ 3-6 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/Makefile

จากรูปที่ 3-6 แสดงโค้ดส่วนหนึ่งของไฟล์ arch/armnommu/Makefile จากรูปหมายความว่าถ้าได้มีการเลือก config platform เป็น เกมบอยแอดวานซ์ (CONFIG_ARCH_GBA) ให้ใช้เฮดเดอร์ไฟล์คือ arch/armnommu/kernel/head-arm-gba.o ให้ชื่อ MACHINE ว่า gba และออปชันในการคอมไพล์

ARCHLDFLAGS เป็นดังรูป และที่สำคัญ คือ TEXTADDR ที่เซ็ตค่าให้เท่ากับ 0x08000000 นั่นคือค่าตำแหน่งที่จะให้ IMAGE เริ่มทำงาน ซางก็คือตำแหน่งของตลับ Xport นั่นเอง

```
if [ "$CONFIG_ARCH_NEXUSPCI" = "y" -o "$CONFIG_ARCH_EBSA110" = "y" ]; then
  define_bool CONFIG_CPU_SA110 y
else
  if [ "$CONFIG_ARCH_TRIO" = "y" -o "$CONFIG_ARCH_GBA" = "y"  ]; then
            define_bool CONFIG_CPU_ARM7 y
  else
    if [ "$CONFIG_ARCH_AT91" = "y" ]; then
      define_bool CONFIG_ARCH_ATMEL y
      define_bool CONFIG_CPU_ARM7 y
      int 'Asyncronous Clock Frequency' CONFIG_ARM_CLK 32768000
      hex 'Base Address for DRAM' DRAM_BASE 0x02000000
      hex 'Amount of DRAM present' DRAM_SIZE 0x00100000
      hex 'Base Address for Flash Memory' FLASH_MEM_BASE 0x01000000
      hex 'Amount of FLASH present' FLASH_SIZE 0x00100000
      bool 'Use EBI to configure system addresses' CONFIG_EBI y
      bool 'System will boot from flash' CONFIG_WILL_BOOT_FROM_FLASH y
    else
      if [ "$CONFIG_ARCH_NETARM" = "y" ]; then
        define_bool CONFIG_CPU_ARM7 y
      else
            if [ "$CONFIG_ARCH_A5K" = "y" ]; then
        define_bool CONFIG_CPU_ARM6 y
            else
              choice 'ARM cpu type'                       \
                "ARM2           CONFIG_CPU_ARM2 \
                ARM3            CONFIG_CPU_ARM3 \
                ARM6/7                  CONFIG_CPU_ARM6 \
                StrongARM       CONFIG_CPU_SA110" StrongARM
            fi
      fi
    fi
  fi
fi

if [ "$CONFIG_ARCH_GBA" = "y" ]; then
   bool 'GBA TEXT console support' CONFIG_GBATXT
   define_int CONFIG_ARM_CLK 167800000
   define_hex DRAM_BASE 0x02000000
   define_hex DRAM_SIZE 0x00040000
   define_hex FLASH_MEM_BASE 0x08000000
   define_hex FLASH_SIZE 0x02000000
fi
```

รูปที่ 3-7 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/config.in

จากรูปที่ 3-7 แสดงโค้ดส่วนหนึ่งของไฟล์ arch/armnommu/config.in จากรูปหมายความว่าถ้ามีการเลือก config platform เป็น เกมบอยแอดวานซ์ (CONFIG_ARCH_GBA) ให้ใช้ค่าคอนฟิกของ ARM7TDMI (CONFIG_CPU_ARM7) และใช้คอนโซลเท็กซ์ซัพพอร์ตคือ CONFIG_GBATXT และกำหนดความถี่ของสัญญาณนาฬิกาให้มีค่าเท่ากับ 16.78 เมกกะเฮิร์ต ซึ่งเป็นความถี่ของซีพียู ARM7TDMI และกำหนดตำแหน่งของแรม (DRAM_BASE) ให้มีค่าเท่ากับ 0x02000000 ซึ่งก็คือตำแหน่งของเอ็กซ์เทอนอลแรม และขนาดของแรม (DRAM_SIZE) และตำแหน่งของแฟลชเมมโมรี (FLASH_MEM_BASE) ซึ่งก็คือตำแหน่งของตลับนั่นเอง และได้ระบุขนาดด้วย

```
#include <linux/config.h>
#include <asm/assembler.h>
#include <asm/hardware.h>
#define  ARM_MODE_USR  0x10
#define  ARM_MODE_IRQ  0x12
#define  ARM_MODE_SVC  0x13
#define  DRAM_LIMIT        (DRAM_BASE + DRAM_SIZE)
.extern _stext, _etext
.extern _sdata, _edata
.extern _sbss, _ebss
.global start
.global _start
.global _entry
.global reset
.text
.align
start:
_start:
        /* GBA has header info at start of ROM */
    .long 0,0,0,0,0,0,0,0
    .long 0,0,0,0,0,0,0,0
    .long 0,0,0,0,0,0,0,0
    .long 0,0,0,0,0,0,0,0
    .long 0,0,0,0,0,0,0
    .long 0,0,0,0,0,0,0,0
reset:
_entry:
        /* Disable interrupts */
        mov     r0, #(ARM_MODE_SVC | I_BIT | F_BIT )  /*supervisor mode and disable IRQ and FIQ */
        msr     cpsr, r0            /*move register to cpsr*/
        /*
         *      Copy data segment into RAM, and clear BSS.
         */
        ldr     r0, =_etext         /* Addr of data in FLASH load from memory to register*/
        ldr     r1, =_sdata         /* Addr of real RAM data */
        ldr     r2, =_edata         /* Addr of end of data */
_copydata:
        cmp     r1, r2              /* All copied? */
        ldrcc   r3, [r0], #4
        strcc   r3, [r1], #4        /* Copy next word */
        bcc     _copydata           /* Keep going till done */
        ldr     r2, =_ebss          /* Addr of end of bss */
        mov     r3, #0
_zerobss:
        cmp     r1, r2              /* All zeroed? */
        strcc   r3, [r1], #4        /* Zero next word */
        bcc     _zerobss            /* Keep going till done */
        ldr     r0, =arm_id
        ldr     r1,=0x41007300      /* Hard code ARM CPU id */
        str     r1, [r0]
        ldr     sp, =start_kernel_stack+4096   /* Set initial stack */
        mov     fp, #0                         /* Set initial frame */
        b       start_kernel                   /* Kernel C code entry */
```

รูปที่ 3-8 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/kernel/head-arm-gba.S

รูปที่ 3-8 แสดง Startup code คือโค้ดในระดับฮาร์ดแวร์ในการโหลดข้อมูลลงแรม แล้วจึงสตาร์ทเคอร์เนล โดยขั้นตอนแรกจะทำการ deisable interrupt และโหลดข้อมูลจากแฟลชสู่รีจีสเตอร์ เมื่อเสร็จแล้วจึงเรียกฟังก์ชัน start_kernel ให้ทำงาน

```
#if defined(CONFIG_ARCH_GBA)
unsigned long setup_timer (void)
{
        volatile unsigned short         *tcp;
        tcp = (volatile unsigned short *) GBA_TIMER0_CR;
        *tcp = (GBA_TCR_CLK1024 | GBA_TCR_ENB);
        tcp = (volatile unsigned short *) GBA_TIMER1_CR;
        *tcp = (GBA_TCR_CLK256 | GBA_TCR_CASCADE | GBA_TCR_ENB |GBA_TCR_IRQ);
#if 1
{
        *((unsigned short *) 0x04000004) = 0x0008;
        setup_arm_irq(IRQ_VBLANK, &irq_kernel_timer);
}
#endif
#if 0
{
        volatile unsigned short *td0, *td1;
        td0 = (volatile unsigned short *) GBA_TIMER0_DATA;
        td1 = (volatile unsigned short *) GBA_TIMER1_DATA;
        for (;;)
                printk("TIMER0='%04x  TIMER1='%04x\n", *td0, *td1);
}
#endif
}
#endif
```

รูปที่ 3-9 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/kernel/time.c

```
#ifndef __ASM_ARCH_A_OUT_H
#define __ASM_ARCH_A_OUT_H
#endif
```

รูปที่ 3-10 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/a.out.h

```
#ifndef __ASM_ARCH_DMA_H
#define __ASM_ARCH_DMA_H
#define MAX_DMA_ADDRESS         0x03000000
typedef enum {
        DMA_MODE_READ,
        DMA_MODE_WRITE
} dmamode_t;
#define MAX_DMA_CHANNELS        1
#endif
```

รูปที่ 3-11 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/dma.h

```
#ifndef __ASM_ARCH_HARDWARE_H
#define __ASM_ARCH_HARDWARE_H
#include <linux/config.h>
#ifndef __ASSEMBLER__
#define ARM_CLK  ((unsigned long)(CONFIG_ARM_CLK))
#else
#define ARM_CLK  CONFIG_ARM_CLK
#endif
#ifndef __ASSEMBLER__
#define MAPTOPHYS(a)     ((unsigned long)a)
#define KERNTOPHYS(a)    ((unsigned long)(&a))
#define GET_MEMORY_END(p) ((p->u1.s.page_size) * (p->u1.s.nr_pages))
#define PARAMS_BASE      0x7000
#define HARD_RESET_NOW()  { arch_hard_reset(); }
#endif
#endif
```

รูปที่ 3-12 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/hardware.h

```
#ifndef __ASM_ARM_ARCH_IO_H
#define __ASM_ARM_ARCH_IO_H
#undef ARCH_IO_DELAY
#define outb_t(v,p) (*(volatile unsigned char *)(p) = (v))
#define outl_t(v,p) (*(volatile unsigned long *)(p) = (v))
#define inb_t(p)    (*(volatile unsigned char *)(p))
#define inl_t(p)    (*(volatile unsigned long *)(p))
#define outw_t(v,p) (*(volatile unsigned short *)(p) = (v))
#define inw_t(p)    (*(volatile unsigned short *)(p))
extern __inline__ void __outb (unsigned int value, unsigned int port) { outb_t(value,port); }
extern __inline__ void __outw (unsigned int value, unsigned int port) { outw_t(value,port); }
extern __inline__ void __outl (unsigned int value, unsigned int port) { outl_t(value,port); }
#define DECLARE_DYN_IN(sz,fnsuffix,instr)
extern __inline__ unsigned sz __in##fnsuffix (unsigned int port) { return in##fnsuffix##_t(port); }
DECLARE_DYN_IN(char,b,"b")
DECLARE_DYN_IN(short,w,"")
DECLARE_DYN_IN(long,l,"")
#undef DECLARE_DYN_IN
#define __outbc(value,port) outb_t(value,port)
#define __outwc(value,port) outw_t(value,port)
#define __outlc(value,port) outl_t(value,port)
#endif
```

รูปที่ 3-13 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/io.h

```
#undef IRQ_interrupt4
#define   IRQ_interrupt4        timer_IRQ_interrupt
#define IRQ_INTERRUPT(n)  (void (*)(void))IRQ_interrupt##n
#define FAST_INTERRUPT(n)  (void (*)(void))fast_IRQ_interrupt
#define BAD_INTERRUPT(n)  (void (*)(void))bad_IRQ_interrupt
#define PROBE_INTERRUPT(n) (void (*)(void))probe_IRQ_interrupt
static __inline__ void mask_irq(unsigned int irq)
{
        *((unsigned short *) GBA_IE) &= ~(0x1 << irq);
}
static __inline__ void unmask_irq(unsigned int irq)
{
        *((unsigned short *) GBA_IE) |= (0x1 << irq);
}
static __inline__ unsigned long get_enabled_irqs(void)
{
        return((unsigned long) *((unsigned short *) GBA_IE));
}
static __inline__ void irq_init_irq(void)
{
        extern void vector_IRQ(void);
#if 0
        *((unsigned long *) GBA_VECIRQ) = vector_IRQ;
#endif
#if 0
        printk("%s(%d): setting time_IRQ...\n", __FILE__, __LINE__);
        *((unsigned long *) GBA_VECIRQ) = (unsigned long) timer_IRQ_interrupt;
#endif
        *((unsigned short *) GBA_IE) = 0;
        *((unsigned short *) GBA_IME) = 0x0001;
}
#endif
```

รูปที่ 3-14 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/irq.h

```
#ifndef __ASM_ARCH_IRQS_H
#define __ASM_ARCH_IRQS_H
#define   GBA_IE              0x04000200          /* Interrupt mask */
#define   GBA_IF              0x04000202          /* Interrup service */
#define   GBA_IME             0x04000208          /* Enable/disable */
#define   IRQ_VBLANK          0                   /* Vertical blank intr */
#define   IRQ_HBLANK          1                   /* Horizontal blank intr */
#define   IRQ_YTRIG           2                   /* Y Trigger intr */
#define   IRQ_TC0             3                   /* Timer 0 intr */
#define   IRQ_TC1             4                   /* Timer 1 intr */
#define   IRQ_TC2             5                   /* Timer 2 intr */
#define   IRQ_TC3             6                   /* Timer 3 intr */
#define   IRQ_COMMS           7                   /* Comms intr */
#define   IRQ_DMA0            8                   /* DMA 0 intr */
#define   IRQ_DMA1            9                   /* DMA 1 intr */
#define   IRQ_DMA2            10                  /* DMA 2 intr */
#define   IRQ_DMA3            11                  /* DMA 3 intr */
#define   IRQ_KEYPAD          12                  /* Keypad intr */
#define   IRQ_CART            13                  /* CART intr */
#define   GBA_IRQ_MASK        0x3fff              /* All intr mask */
#define IRQ_MACHSPEC     (0x10000000L)
#define IRQ_IDX(irq)    ((irq) & ~IRQ_MACHSPEC)
#define IRQ_FLG_LOCK     (0x0001)                /* handler is not replaceable  */
#define IRQ_FLG_REPLACE (0x0002)                 /* replace existing handler    */
#define IRQ_FLG_FAST     (0x0004)
#define IRQ_FLG_SLOW     (0x0008)
#define IRQ_FLG_STD     (0x8000)                 /* internally used             */
#endif
```

รูปที่ 3-15 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/irqs.h

จากรูปที่ 3-15 แสดงถึงการแก้ไขโค้ดที่เกี่ยวข้องกับการอินเทอร์รัพท์ต่างๆ ซึ่งค่าเหล่านี้สามารถอ้างอิงได้จากฮาร์ดแวร์ของเกมบอยแอดวานซ์ และฮาร์ดแวร์ของซีพียู ARM7TDMI

```
#ifndef __ASM_ARCH_MMU_H
#define __ASM_ARCH_MMU_H
#define __virt_to_phys__is_a_macro
#define __virt_to_phys(vpage) vpage
#define __phys_to_virt__is_a_macro
#define __phys_to_virt(ppage) ppage
#define __virt_to_bus__is_a_macro
#define __virt_to_bus(vpage) vpage
#define __bus_to_virt__is_a_macro
#define __bus_to_virt(ppage) ppage
#define __flush_entry_to_
#endif
```

รูปที่ 3-16 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/mmu.h

```
#ifndef __ASM_ARCH_PROCESSOR_H
#define __ASM_ARCH_PROCESSOR_H
#define EISA_bus 0
#define EISA_bus__is_a_macro
#define MCA_bus 0
#define MCA_bus__is_a_macro
#define TASK_SIZE (0x01a00000UL)
#define INIT_MMAP { \  &init_mm, 0, 0x02000000, PAGE_SHARED, VM_READ | VM_WRITE | VM_EXEC \ }
#endif
```

รูปที่ 3-17 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/processor.h

```
#ifndef __ASM_ARCH_SYSTEM_H
#define __ASM_ARCH_SYSTEM_H
#include <asm/hardware.h>
#include <asm/io.h>
extern  void arch_hard_reset (void);
#endif
```

รูปที่ 3-18 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/system.h

```
#ifndef __ASM_ARCH_TIME_H
#define __ASM_ARCH_TIME_H
#define   GBA_TIMER0_DATA       0x04000100          /* Timer 0 data reg */
#define   GBA_TIMER1_DATA       0x04000104          /* Timer 1 data reg */
#define   GBA_TIMER2_DATA       0x04000108          /* Timer 2 data reg */
#define   GBA_TIMER3_DATA       0x0400010c          /* Timer 3 data reg */
#define   GBA_TIMER0_CR         0x04000102          /* Timer 0 control reg */
#define   GBA_TIMER1_CR         0x04000106          /* Timer 1 control reg */
#define   GBA_TIMER2_CR         0x0400010a          /* Timer 2 control reg */
#define   GBA_TIMER3_CR         0x0400010e          /* Timer 3 control reg */
#define   GBA_TCR_CLK           0x0000              /* Use clock freq */
#define   GBA_TCR_CLK64         0x0001              /* Use clock/64 freq */
#define   GBA_TCR_CLK256        0x0002              /* Use clock/256 freq */
#define   GBA_TCR_CLK1024       0x0003              /* Use clock/1024 freq */
#define   GBA_TCR_CASCADE       0x0008              /* Cascade timer */
#define   GBA_TCR_IRQ           0x0040              /* Generate IRQ */
#define   GBA_TCR_ENB           0x0080              /* Enable timer */
#define   KERNEL_TIMER_IRQ_NUM 1
extern __inline__ unsigned long gettimeoffset (void)
{
        return(0);
}
extern __inline__ int reset_timer(void)
{
        return (1);
}
#define update_rtc() {}
#endif
```

รูปที่ 3-19 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/arch-gba/time.h

รูปที่ 3-19 เป็นการเพิ่มข้อมูลในส่วนของเวลา ซึ่งสามารถอ้างอิงได้จากรีจีสเตอร์ของเกมบอยแอดวานซ์

```
#ifdef CONFIG_ARCH_ATMEL
#define PAGE_OFFSET             0x02040000
#clif CONFIG_ARCH_GBA
#define   PAGE_OFFSET           DRAM_BASE
#else
#define PAGE_OFFSET             0x00000000
#endif
```

รูปที่ 3-20 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/asm-armnommu/proc-armv/page.h

รูปที่ 3-20 เป็นการเปลี่ยนออฟเซ็ตของเมมโมรี่ ซึ่งถ้ามีการคอนฟิกแพลตฟอร์มเกมบอนแอดวานซ์จะใช้ค่าเท่ากับ DRAM_BASE ซึ่งมีค่าเท่ากับ 0x02000000

```
#if !defined( CONFIG_ARCH_ATMEL ) && !defined(CONFIG_ARCH_GBA)
extern void * memset(void *,int,__kernel_size_t);
extern void * memcpy(void *,const void *,__kernel_size_t);
extern int memcmp(const void *,const void *,__kernel_size_t);
#endif
```

รูปที่ 3-21 แสดงโค้ดที่เพิ่มเติมในไฟล์ include/linux/string.h

รูปที่ 3-21 เป็นการกำหนดตัวอักษรของคอนโซล ซึ่งถ้ามีการกำหนดเป็นแพลตฟอร์มเกมบอยแอดวานซ์แล้ว จะให้อ้างอิงจาก GBA_TXT แทน

```
#include <asm/hardware.h>
#include <asm/io.h>
static __inline__ void do_outw(unsigned int reg, const void* buf, unsigned int count)
{
        register const unsigned short* wbuf = (const unsigned short*) buf;
        while(count--)
                outw(*wbuf++, reg);
}
static __inline__ void do_inw(unsigned int reg, void* buf, unsigned int count)
{
        register unsigned short* wbuf = (unsigned short*) buf;
        while(count--)
                *wbuf++ = inw(reg);
}
void outsw(unsigned to_reg, const void *from, int len_in_words)
{
        do_outw(to_reg, from, len_in_words);
}
void outswb(unsigned  to_reg, const void *from, int len_in_bytes)
{
        do_outw(to_reg, from, len_in_bytes/2);
}
void insw(unsigned from_port, void *to, int len_in_words)
{
        do_inw(from_port, to, len_in_words);
}
void  inswb(unsigned from_port, void *to, int len_in_bytes)
{
        do_inw(from_port, to, len_in_bytes/2);
}
void arch_hard_reset (void)
{
        for (;;)
                ;
}
```

รูปที่ 3-22 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/lib/io-gba.c

```
void do_IRQ(int irq, struct pt_regs * regs);
asmlinkage int probe_IRQ_interrupt(int irq, struct pt_regs * regs)
{       mask_irq(irq);
        irq_ack(0);
        return 0;
}
asmlinkage int bad_IRQ_interrupt(int irqn, struct pt_regs * regs)
{       printk("bad interrupt %d recieved!\n", irqn);
        irq_ack(0);
        return 0;
}
asmlinkage int IRQ_interrupt(int irq, struct pt_regs * regs)
{       register unsigned long flags;
        register unsigned long saved_count;
        saved_count = intr_count;
        intr_count = saved_count + 1;
        save_flags(flags);
        sti();
        do_IRQ(irq, regs);
        restore_flags(flags);
        intr_count = saved_count;
        irq_ack(0);
        return 0;
}
asmlinkage int timer_IRQ_interrupt(int irq, struct pt_regs * regs)
{       register unsigned long flags;
        register unsigned long saved_count;
#if 1
        printk("timer_IRQ_interrupt(irq=%d)\n", irq);
#endif

        saved_count = intr_count;
        intr_count = saved_count + 1;

        save_flags(flags);
        do_timer(regs);
        restore_flags(flags);
        intr_count = saved_count;
        irq_ack(0);
        return 0;
}
asmlinkage int fast_IRQ_interrupt(int irq, struct pt_regs * regs)
{       register unsigned long saved_count;
        saved_count = intr_count;
        intr_count = saved_count + 1;

        do_IRQ(irq, regs);
        cli();
        intr_count = saved_count;
        return 1;
}
```

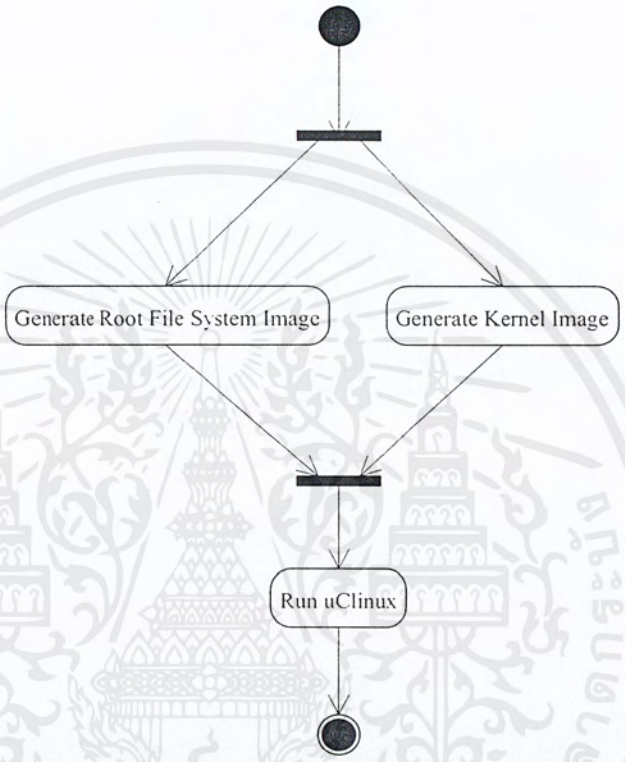
รูปที่ 3-23 แสดงโค้ดที่เพิ่มเติมในไฟล์ arch/armnommu/lib/irqs-gba.c

3.7 การใช้งานไมโครซีลีนุกซ์กับเกมบอยแอดวานซ์

การใช้งานไมโครซีลีนุกซ์ในการพัฒนาและนำไปทำงานบนเกมบอยแอดวานซ์ สามารถแบ่งออกได้เป็น 3 ส่วนหลัก ดังที่แสดงดังรูปที่ 3-24

1. Generate Root File System Image คือการสร้าง Root File Systems Image สำหรับใช้เป็น Root File System สำหรับไมโครซีลีนุกซ์ ซึ่งประกอบด้วยแอพพลิเคชัน โปรแกรมทั่วไป แฟ้มข้อมูล และแฟ้มที่จำเป็นในการทำงาน เช่น dev

2. Generate Kernel Image คือการสร้าง Kernel Image ของไมโครซีลีนุกซ์แบบไบนารี เพื่อนำไปทำงานบนเครื่องเกมบอยแอดวานซ์
3. Run uClinux คือการนำเอาไมโครซีลีนุกซ์และแอพพลิเคชันจาก Root File Systems Image ไปทำงานบนเครื่องเกมบอยแอดวานซ์



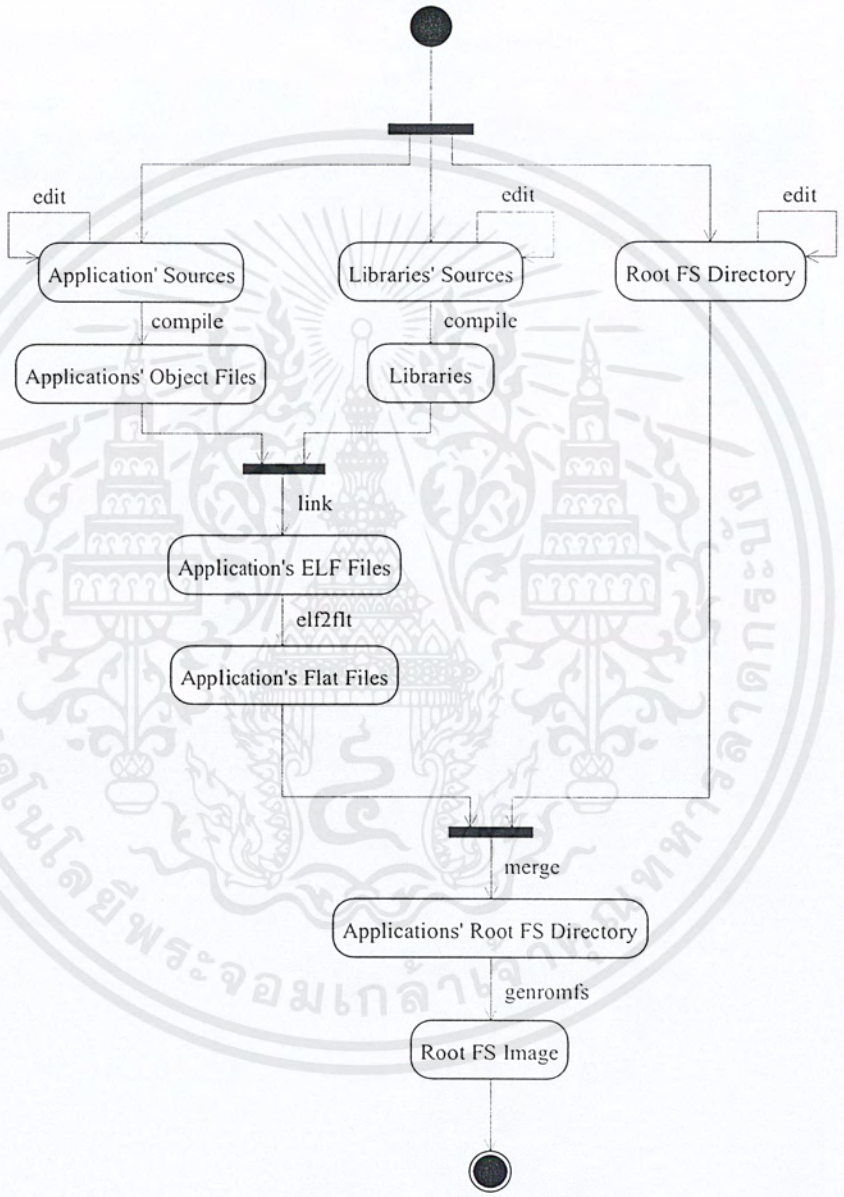
รูปที่ 3-24 การใช้งานไมโครซีลีนุกซ์กับเกมบอยแอดวานซ์

3.7.1 Generate Root File Systems Image

ในการสร้าง Root File Systems Image ซึ่งมีความสำคัญสำหรับผู้พัฒนาแอพพลิเคชันสำหรับเกมบอยแอดวานซ์มากกว่าการสร้าง Kernel Image ซึ่งได้มีการเตรียมไว้ให้แล้ว ไม่จำเป็นต้องสร้างใหม่ Root File Systems Image ประกอบไปด้วยแอพพลิเคชัน โปรแกรทั่วไป แฟ้มข้อมูล และแฟ้มที่จำเป็นในการทำงานนั้นสามารถแยกการพัฒนาได้เป็น 2 ส่วนดังรูปที่ 3-25 ตามเส้นทางด้านซ้ายและขวา

ตามเส้นทางด้านซ้ายจากโค้ดต้นแบบของแอพพลิเคชัน (Application's Sources) ที่ทำการสร้างและแก้ไขนำไปทำการคอมไพล์เป็นออปเจ็กของแอพพลิเคชันแบบ ELF และทำการลิงค์กับไลบรารี ซึ่งเราอาจจะทำการแก้ไขจากโค้ดต้นแบบของไลบรารีได้เนื่องจากไลบรารีที่ใช้งานยังอยู่ในการพัฒนา หลังจากที่ทำการลิงค์เรียบร้อยแล้ว จะได้แอพพลิเคชันที่สามารถทำงานได้แบบ ELF แล้วทำการแปลงจากแบบ ELF ไปเป็นแบบแฟลทให้สามารถทำงานได้บนเกมบอยแอดวานซ์ด้วยโปรแกรม elf2flt

ตามเส้นทางด้านขวาจากไดเรกทอรี่สำหรับทำเป็นรูตไดเรกทอรี่ของไมโครซีลีนุกซ์ซึ่งจำเป็นต้องจัดเตรียมไหล์ที่จำเป็นในการทำงานสำหรับแอพพลิเคชันและตัวไมโครซีลีนุกซ์เอง แล้วนำแอพพลิเคชันที่ได้จากเส้นทางด้านซ้ายมารวมด้วย ได้เป็นไดเรกทอรี่ที่พร้อมจะนำไปสร้างเป็นรูทไดเรกทอรี่ แล้วนำมาแปลงมาเป็นไบนารีที่พร้อมนำไปทำงานบนเกมบอยแอดวานซ์

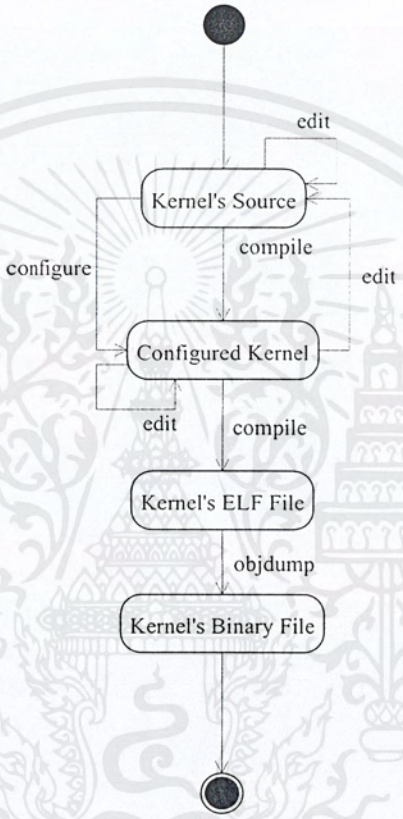


รูปที่ 3-25 Generate Root FS Image

3.7.2 Generate Kernel Image

ในการสร้าง Kernel Image นั้นไม่จำเป็นที่ต้องมีการสร้างทุกครั้ง แต่จะทำการสร้างเมื่อมีการแก้ไข หรือคอนฟิกไมโครซีลีนุกซ์ใหม่เท่านั้น ซึ่งขั้นตอนการสร้างแสดงดังรูปที่ 3-26

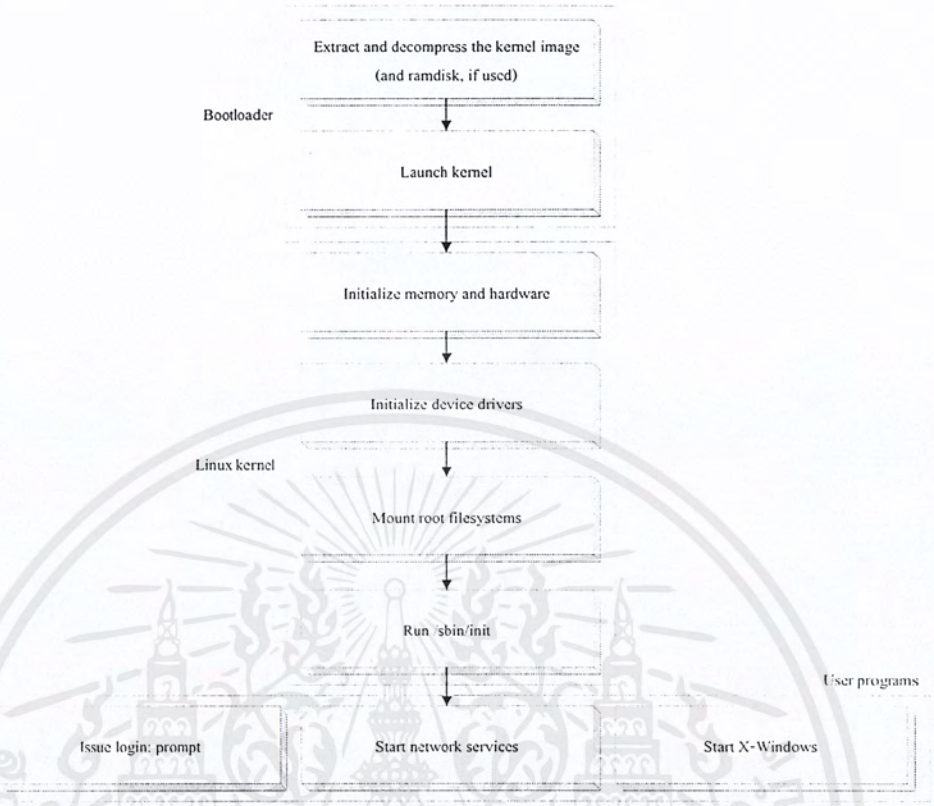
จากโค้ดต้นแบบของไมโครซีลีนุกซ์ที่ทำการแก้ไขแล้วทำการคอนฟิกให้พร้อมสำหรับคอมไพล์ ได้เป็นไฟล์แบบ ELF และการแปลงเป็นไบนารีด้วยโปรแกรม objdump



รูปที่ 3-26 Generate Kernel Image

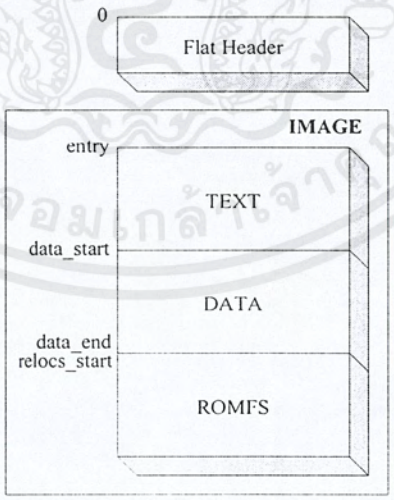
3.7.3 Run Kernel

เมื่อได้ Kernel Image และ Root File Systems Image แล้วนำมาทำงานบนเกมบอยแล้วนั้นมีขั้นตอนการทำงานดังรูปที่ 3-27

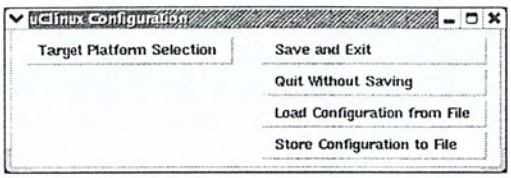


รูปที่ 3-27 Run uClinux

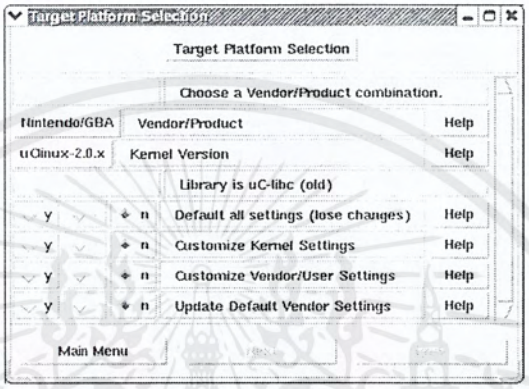
เมื่อทำการสั่งให้ไมโครซีลีนุกซ์ทำงาน ตัวไมโครซีลีนุกซ์จะทำการเมาท์กับ Root File System ซึ่งอยู่ในหน่วยความจำ เมื่อเมาท์เรียบร้อยแล้วไมโครซีลีนุกซ์จะทำการรันแอพพลิเคชัน



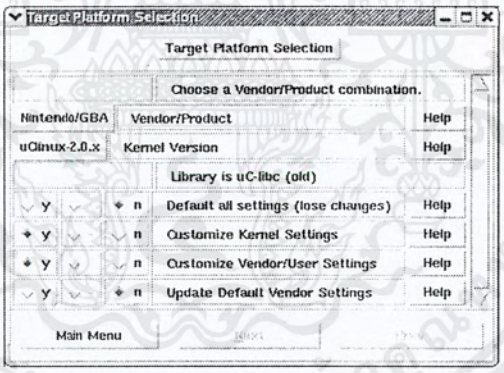
รูปที่ 3-28 แสดงส่วนประกอบของไมโครซีลีนุกซ์ที่คอมไพล์เรียบร้อยแล้ว



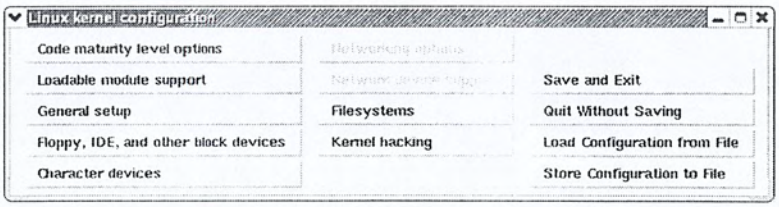
รูปที่ 3-29 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์โดยคำสั่ง make xconfig (1)



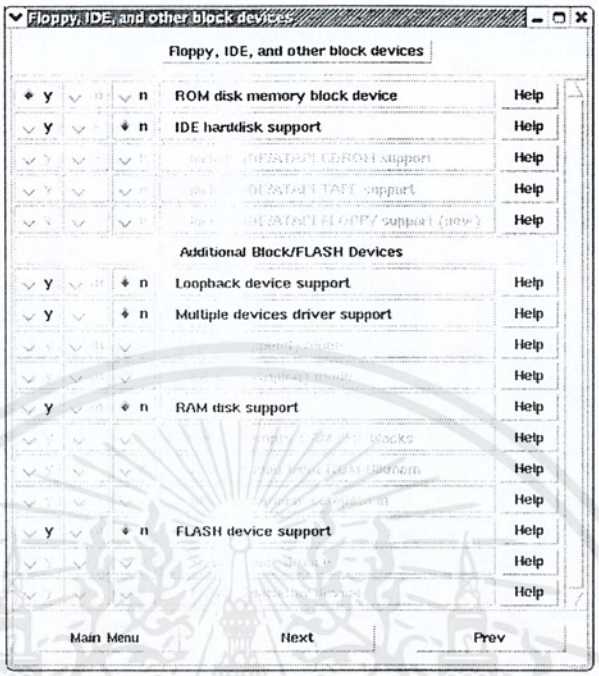
รูปที่ 3-30 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์โดยคำสั่ง make xconfig (2)



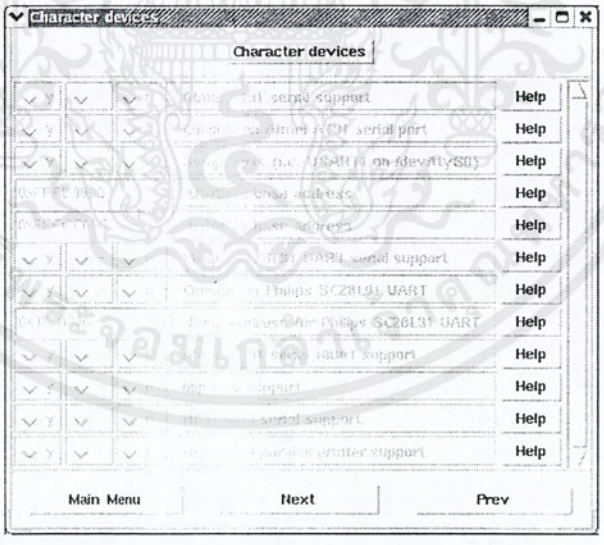
รูปที่ 3-31 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์โดยคำสั่ง make xconfig (3)



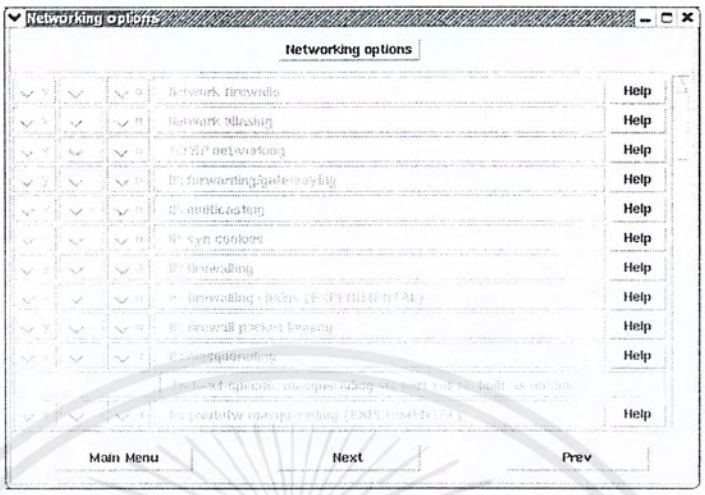
รูปที่ 3-32 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์โดยคำสั่ง make xconfig (4)



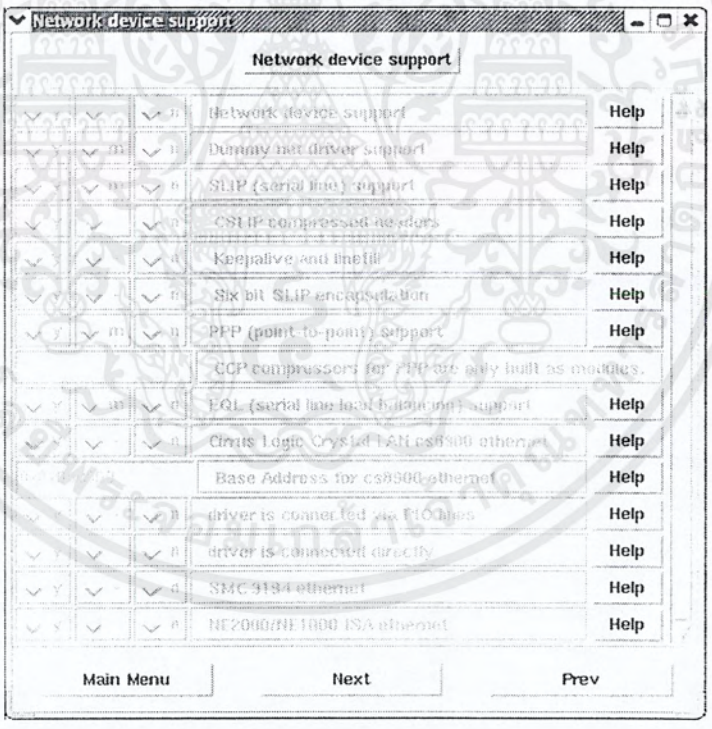
รูปที่ 3-33 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (5)



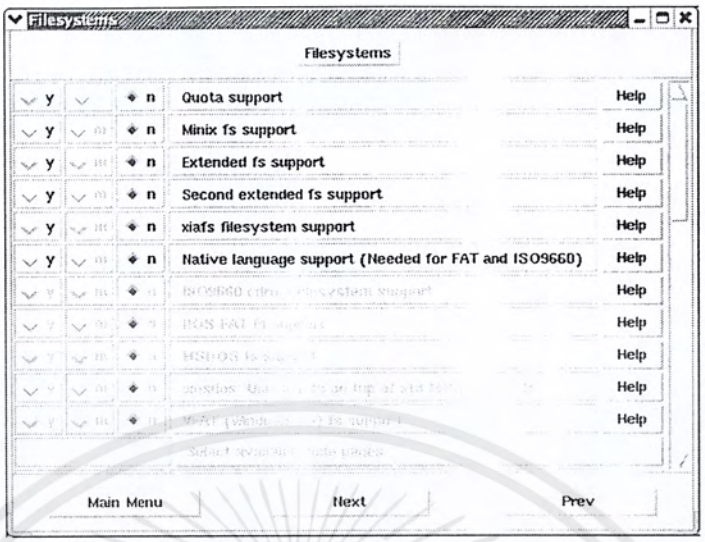
รูปที่ 3-34 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (6)



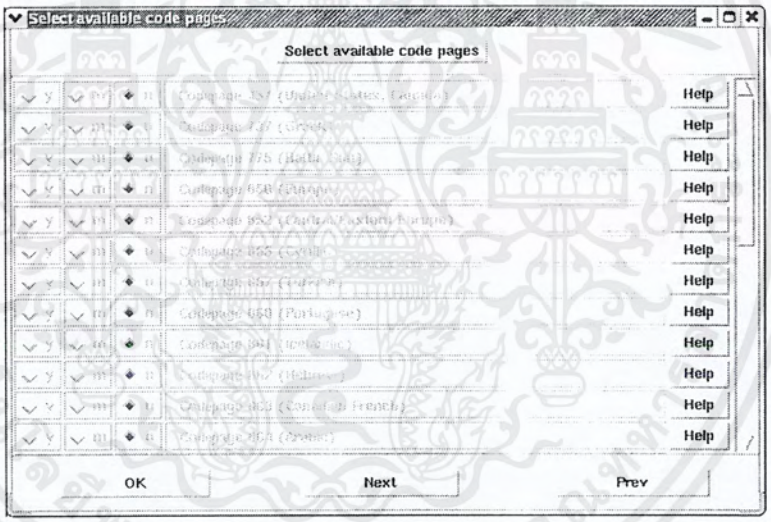
รูปที่ 3-35 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (7)



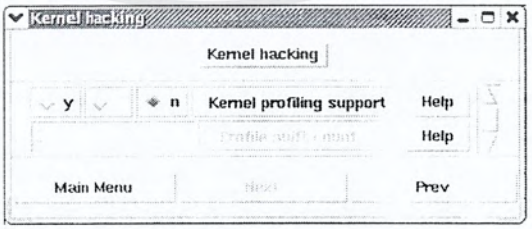
รูปที่ 3-36 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (8)



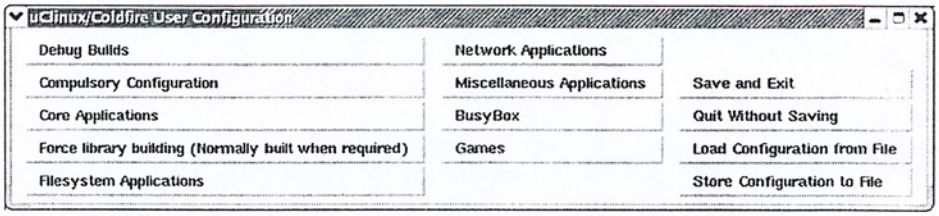
รูปที่ 3-37 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (9)



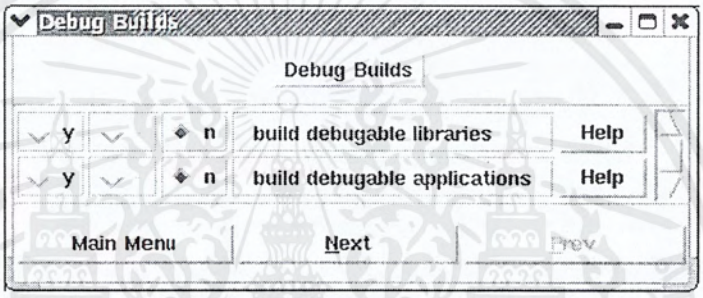
รูปที่ 3-38 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (10)



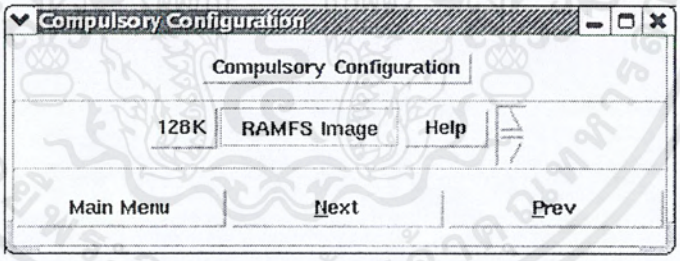
รูปที่ 3-39 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (11)



รูปที่ 3-40 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (12)

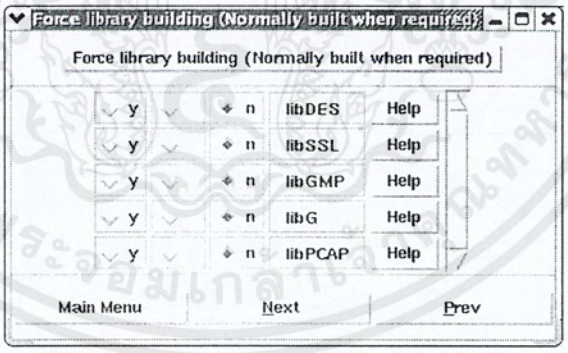


รูปที่ 3-41 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (13)

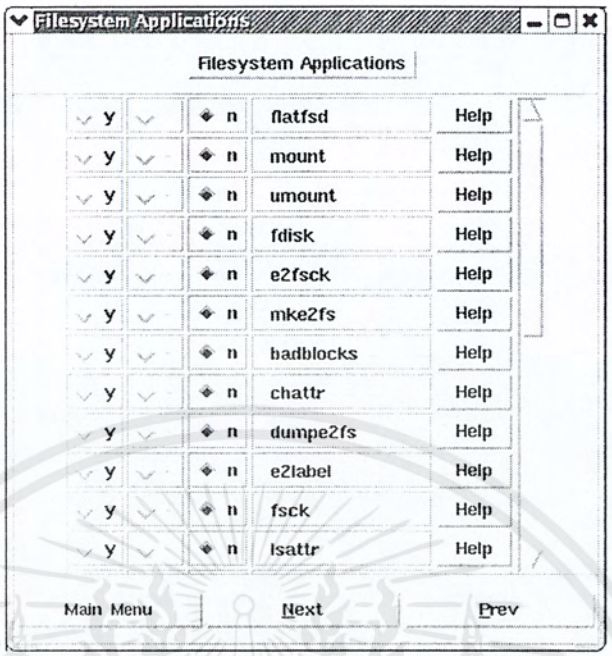


รูปที่ 3-42 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (14)

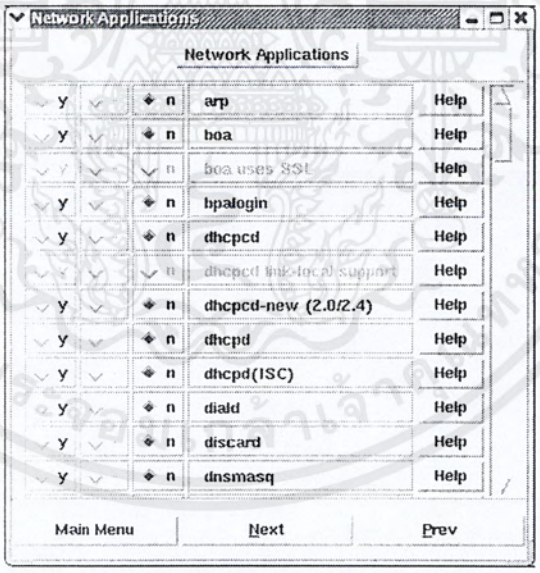
รูปที่ 3-43 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (15)



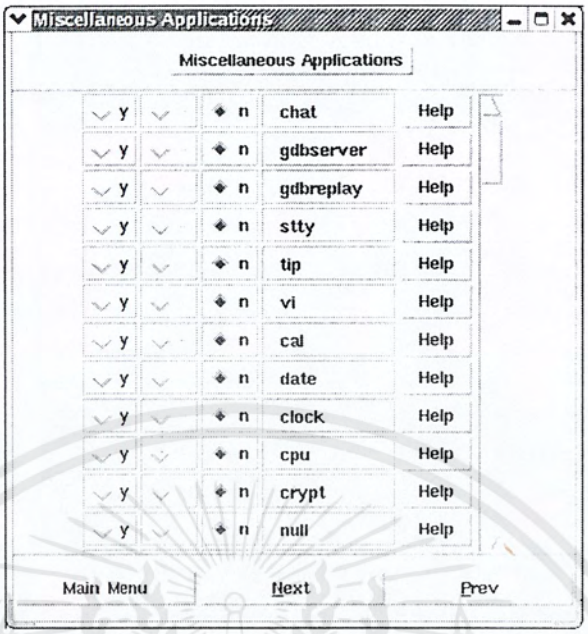
รูปที่ 3-44 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (16)



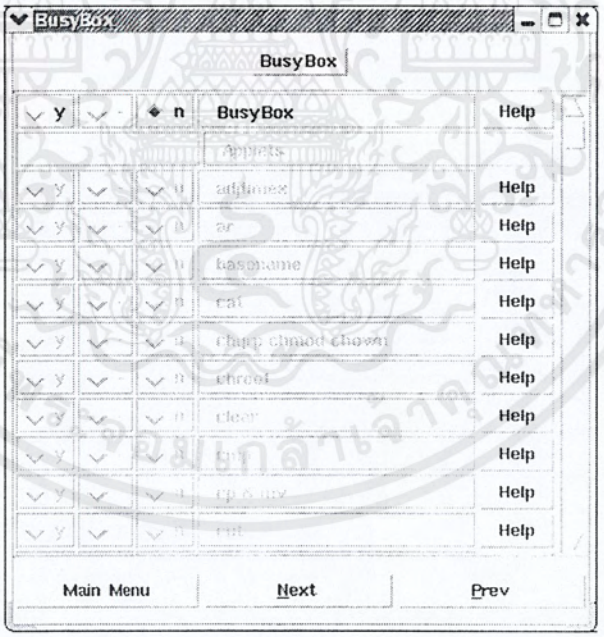
รูปที่ 3-45 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (17)



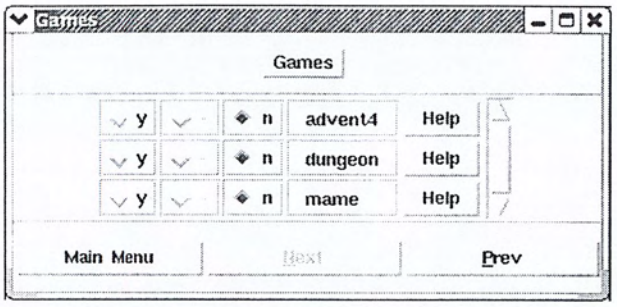
รูปที่ 3-46 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (18)



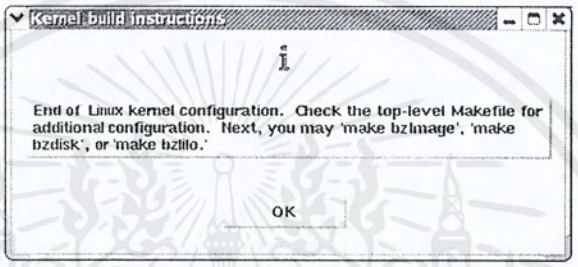
รูปที่ 3-47 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (19)



รูปที่ 3-48 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (20)

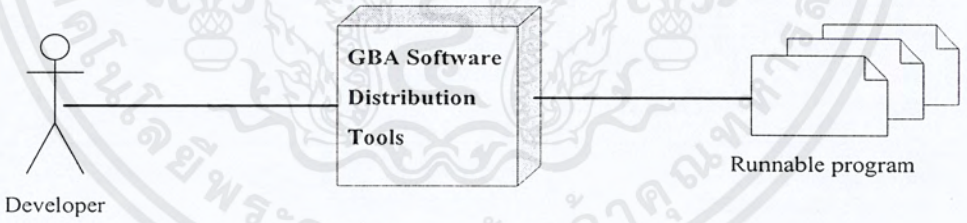


รูปที่ 3-49 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (21)



รูปที่ 3-50 ขั้นตอนการคอนฟิกไมโครซีลีนุกซ์ โดยคำสั่ง make xconfig (22)

3.8 การสร้างเครื่องมืออำนวยความสะดวกในการพัฒนาโปรแกรมบนเครื่องเกมบอยแอดวานซ์



รูปที่ 3-51 รูปแบบการใช้งานเครื่องมือพัฒนาแอพพลิเคชันบนเกมบอยแอดวานซ์

ในปัจจุบันการที่จะเขียนโปรแกรมให้สามารถทำงานได้บนเครื่องเกมบอยแอดวานซ์ ยังถือว่าทำได้ยาก เนื่องจากการสร้างโปรแกรมบนเครื่องเกมบอยแอดวานซ์จะต้องอาศัยเครื่องมือหลายๆอย่างด้วยกัน อีกทั้งยังจะต้องศึกษาวิธีการใช้เครื่องมือต่างๆด้วย ในโครงงานจึงจัดทำเครื่องมืออำนวยความสะดวกในการสร้างโปรแกรมบนเครื่องเกมบอยแอดวานซ์ขึ้น โดยสามารถแบ่งออกได้เป็น 2 ประเภทคือ เป็นเครื่องมือช่วยในการสร้างโปรแกรมบนเครื่องเกมบอยแอดวานซ์โดยตรงหรือไม่พึ่งระบบปฏิบัติการ และ

เครื่องมือช่วยในการสร้างโปรแกรมบนเครื่องเกมบอยแอดวานซ์แบบพึ่งระบบปฏิบัติการ โดยผลงานที่ออกมาจะอยู่ในรูปของซีดีรอม

3.8.1 เครื่องมือช่วยในการพัฒนาโปรแกรมบนเครื่องเกมบอยแอดวานซ์โดยตรง

เป็นการรวบรวมเครื่องมือที่มีการเผยแพร่อยู่ในระบบเครือข่าย และทำการศึกษาเครื่องมือต่างๆ เหล่านั้น แล้วสร้างเป็นเอกสารการใช้งานเครื่องมือต่างๆขึ้นมาเพื่อให้นักพัฒนาสามารถใช้งานเครื่องมือต่างๆได้อย่างรวดเร็ว โดยเครื่องมือและเอกสารต่างๆจะถูกจัดเก็บอยู่ในซีดีรอมแผ่นที่ 1

ภายใน CD-ROM แผ่นที่ 1 มีแพกเกจดังนี้

1. Compiler จะเป็นแพกเกจที่เก็บเครื่องมือคอมไพล์ เฉพาะของเครื่องเกมบอยแอดวานซ์
2. Emulator จะเป็นแพกเกจที่ใช้เก็บเครื่องมือทำงานเสมือนเครื่องเกมบอยแอดวานซ์
3. Misc Tools จะเป็นแพกเกจที่ใช้เก็บเครื่องมือในการแปลงข้อมูลต่างๆ ให้อยู่ในรูปแบบที่เป็นประโยชน์ต่อการสร้างโปรแกรม รวมไปถึงเครื่องมือในการส่งข้อมูลเข้าสู่ตลับ
4. MayNeedTool จะเป็นแพกเกจที่ใช้เก็บเครื่องมือซึ่งบางเครื่องมือข้างต้นมีความต้องการก่อนจึงจะสามารถใช้งานได้ เช่น Cygwin, DirectX, etc.
5. 1st Document.pdf เป็นเอกสารบอกถึงวิธีการใช้งานทุกๆ เครื่องมือภายใน CD-ROM

3.8.2 เครื่องมือช่วยในการพัฒนาโปรแกรมบนเครื่องเกมบอยแอดวานซ์แบบพึ่งพาระบบปฏิบัติการ

เป็นการรวบรวมเครื่องมือรวมไปถึง ข้อมูลคำสั่ง ของระบบปฏิบัติการ eCos กับระบบปฏิบัติการยูซีลีนุกซ์ที่จำเป็นต้องทำการดึงข้อมูลจากเครื่อข่ายมาไว้ในซีดีรอมแผ่นที่ 2 แล้วสร้างเอกสารในการใช้งานรวมไปถึงแนวทางการคอมไพล์ระบบปฏิบัติการทั้งสอง

ภายใน CD-ROM แผ่นที่ 2 มีโครงสร้างแพกเกจดังนี้

1. Compiler จะเป็นที่เก็บเครื่องมือครอส-คอมไพล์
2. eCos จะเป็นที่เก็บข้อมูลคำสั่งของระบบปฏิบัติการ eCos
3. uClinux จะเป็นที่เก็บข้อมูลคำสั่งของระบบปฏิบัติการ ยูซีลีนุกซ์
4. Xport จะเป็นสารบบที่เก็บโปรแกรมที่มากับอุปกรณ์ Xport
5. 2nd Document.pdf เป็นเอกสารบอกถึงวิธีการคอมไพล์ระบบปฏิบัติการทั้งสอง

บทที่ 4
การทดสอบและวิเคราะห์ผล

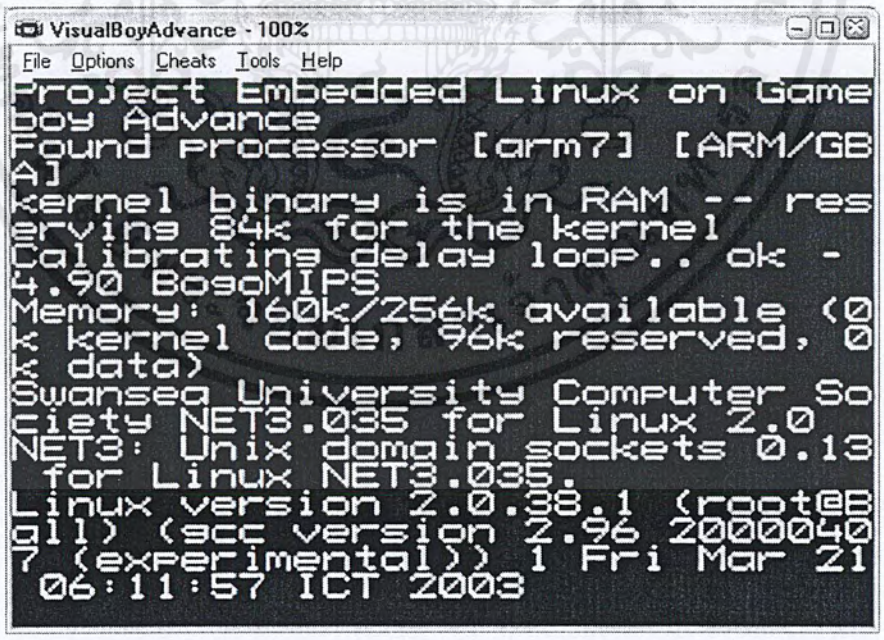
ในแต่ละขั้นตอนในการพัฒนาโครงงานนั้นมีการทดสอบและวิเคราะห์ผลว่าสิ่งที่ทำไปแล้วนั้นเป็นไปตามเป้าหมายของแต่ละขั้นหรือไม่ ซึ่งจำเป็นต้องให้เป็นไปตามเป้าหมายนั้น แต่ถ้าไม่สามารถทำให้เป็นไปได้ตามเป้าหมาย จะมีการพิจารณาในสิ่งที่เกิดขึ้นและหาทางแก้ไขในโครงสร้างของระบบ หรือขั้นตอนและเป้าหมายต่างๆ แต่ไม่ให้กระทบถึงเป้าหมายของโครงงาน เมื่อได้ทำตามขั้นตอนและเป้าหมายต่างๆเป็นที่เรียบร้อยนั้นจะมีการทดสอบและวิเคราะห์ผลอีกทีว่าระบบที่พัฒนาขึ้นมานั้นเป็นไปตามเป้าหมายของโครงงานหรือไม่

ในการทดสอบและวิเคราะห์ผลของโครงงานนี้ ทำตามขั้นตอนการใช้งานในหัวข้อการใช้งานไมโครซีลีนุกซ์กับเกมบอยแอดวานซ์ ดังนั้นสามารถแบ่งการทดสอบและวิเคราะห์ผลได้เป็น 2 ขั้นหลักๆ คือ

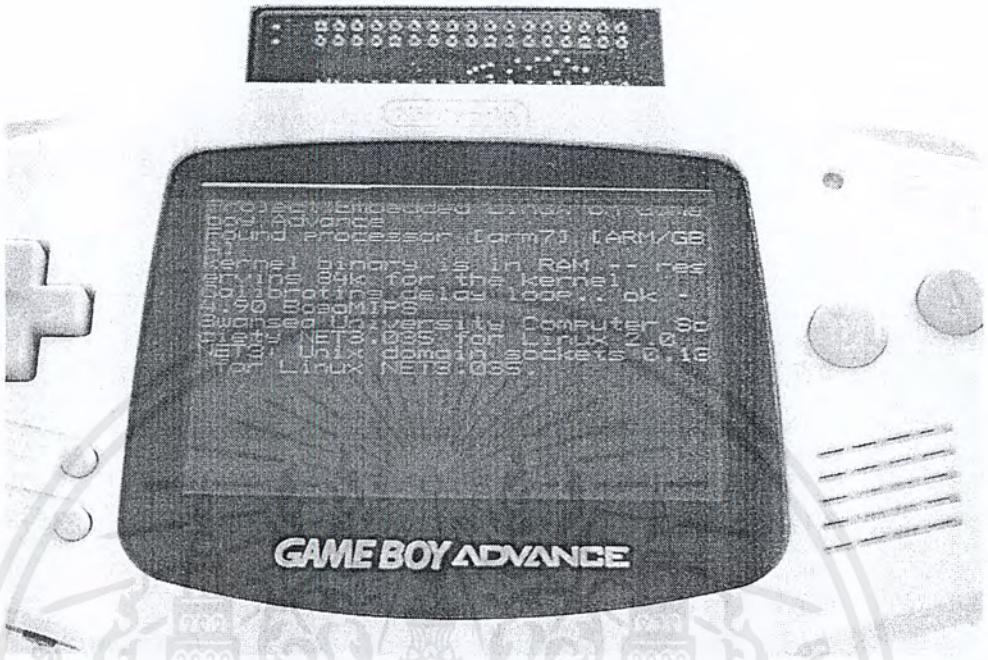
1. ไมโครซีลีนุกซ์กับการทำงานบนเครื่องเกมบอยแอดวานซ์
2. การใช้งานแอพพลิเคชันบนไมโครซีลีนุกซ์

4.1 ไมโครซีลีนุกซ์กับการทำงานบนเครื่องเกมบอยแอดวานซ์

การทดสอบในขั้นนี้ ได้เปรียบเทียบระหว่าง การทำงานบนอีมูเลเตอร์ กับ การทำงานบนระบบฮาร์ดแวร์จริง ซึ่งได้ข้อเปรียบเทียบดังนี้

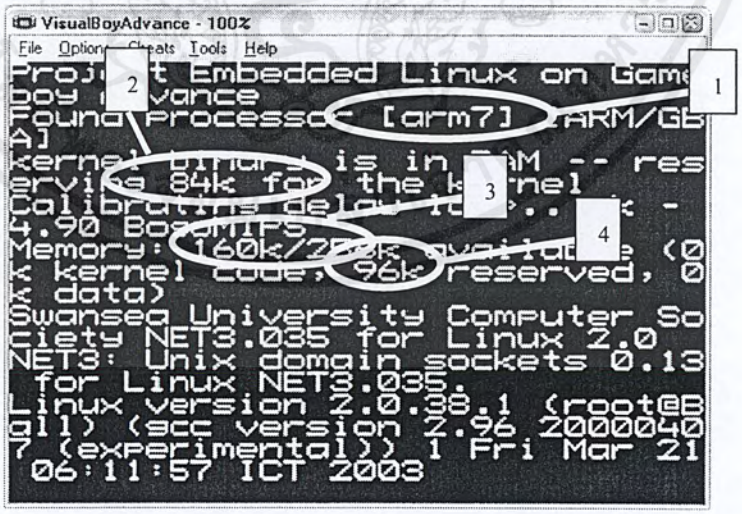


รูปที่ 4-1 การทำงานโดยจำลองการทำงานบนอีมูเลเตอร์



รูปที่ 4-2 การทำงานบนเครื่องเกมบอยแอดวานซ์จริง

เมื่อทำการทดสอบได้ผลลัพธ์ที่เหมือนกัน ซึ่งจริงๆแล้วเมื่อเราคอมไพล์ได้อิมเมจไฟล์ออกมา จะไม่สามารถนำมาลงกับเครื่องเกมบอยได้ จะต้องเพิ่มค่าในเฮดเดอร์ไฟล์จึงจะรันได้



รูปที่ 4-3 แสดงการจองเมมโมรี

จากรูปที่ 4-3 ที่ตำแหน่งที่ 1 จะเห็นว่าไมโครซีลีนุกซ์ได้พบซีพียู arm7tdmi แล้ว ที่ตำแหน่งที่ 2 ได้จองเมมโมรีขนาด 80 Kb ไว้สำหรับเตอร์เนล และ เมมโมรีทั้งหมดที่ว่างอยู่เท่ากับ 160 Kb ที่ตำแหน่ง 3 และมีการจองเมมโมรีมีค่า 96 Kb ที่ตำแหน่ง 4 แต่จะเห็นว่ายังไม่สามารถที่จะพบในส่วนของดาต้าและเตอร์เนล เนื่องจากยังไม่สามารถทำให้ไมโครซีลีนุกซ์นั้นเข้ากับเครื่องเกมบอยแอดวานซ์ได้

4.2 การใช้งานแอพพลิเคชันบนไมโครซีลีนุกซ์

เนื่องจากในส่วนนี้ยังไม่สามารถทำได้ เนื่องจากระบบปฏิบัติการไมโครซีลีนุกซ์ยังไม่สำเร็จตามเป้าหมาย จึงยังไม่สามารถทำการทดลองขั้นนี้ได้

4.3 การทดลองและการทดสอบโปรแกรมที่สร้างจากเครื่องมืออำนวยความสะดวก

ในการดำเนินการทดสอบนั้นจะแบ่งเป็นช่วงๆ ในช่วงเริ่มต้นของการดำเนินโครงการนั้นจะเป็นช่วงที่ทำการศึกษาส่วนประกอบต่างๆของเครื่องเกมบอยแอดวานซ์ แต่ในช่วงเริ่มต้นนั้นจะยังไม่มีเครื่องเกมบอยแอดวานซ์จริงเนื่องจากติดปัญหาทางด้านขนส่ง อย่างไรก็ตามการทดลองในช่วงแรกก็ได้เริ่มต้นขึ้นจากการที่ได้ทำการศึกษาส่วนประกอบต่างๆของเครื่องเกมบอยแอดวานซ์และเครื่องมือที่ใช้ในการสร้างโปรแกรมจึงเริ่มทดลองสร้างโปรแกรมที่สามารถทำงานได้บนอีมูเลเตอร์เป็นการทดลองที่ 1

ในการดำเนินการทดลองต่อมานั้นเมื่อได้เครื่องเกมบอยแอดวานซ์ และอุปกรณ์ที่ใช้ในการพัฒนา Xport การทดลองต่อมาจึงเริ่มขึ้น เพื่อทดสอบความสามารถของเครื่องเกมบอยแอดวานซ์และอุปกรณ์ Xport จึงทดลองนำข้อมูลที่สามารถทำงานได้จากการทดลองที่ 1 มาทำงานบนอุปกรณ์จริง เป็นการทดลองที่ 2

ในการดำเนินการทดลองในช่วงต่อมาจะเป็นช่วงที่ศึกษาเกี่ยวกับระบบปฏิบัติการที่มีความเหมาะสมกับระบบฝังตัว จากการศึกษาพบว่าระบบปฏิบัติการ 2 ระบบที่เหมาะที่จะนำมาทำงานบนเครื่องเกมบอยแอดวานซ์นั้นได้แก่ระบบปฏิบัติการ eCos และระบบปฏิบัติการไมโครซีลีนุกซ์ การทดลองต่อมาจึงเริ่มขึ้นเพื่อทำการทดลองสร้างระบบปฏิบัติการ eCos ในแบบต่างๆและพยายามทำการสร้างอิมเมจของระบบปฏิบัติการไมโครซีลีนุกซ์ จากนั้นสร้างโปรแกรมเพื่อทดสอบการทำงานของระบบปฏิบัติการ eCos ที่สร้างขึ้น เป็นการทดลองที่ 3

ในการดำเนินโครงการต่อมาเมื่อมีความชำนาญในการสร้างระบบปฏิบัติการ eCos ตามต้องการแล้วจึงทำการทดลองทำให้เครื่องเกมบอยแอดวานซ์เป็นเครื่องมือในการกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ ในขณะเดียวกันก็แก้ไขข้อผิดพลาดในอิมเมจของไมโครซีลีนุกซ์ จากนั้นทำการทดลองนำไปทำงานบนเครื่องเกมบอยแอดวานซ์ เป็นการทดลองที่ 4

4.3.1 เงื่อนไขการทดลองและการทดสอบโปรแกรมที่สร้างจากเครื่องมืออำนวยความสะดวก

4.3.1.1 การทดลองที่ 1

ในการทดลองที่ 1 นั้นจะต้องสร้างโปรแกรมจากเครื่องมือช่วยต่างๆโดยอาศัยความรู้ความเข้าใจในส่วนประกอบต่างๆของเครื่องเกมบอยแอดวานซ์ โดยโปรแกรมที่ได้จะนำไปทำงานบนอีมูเลเตอร์ซึ่งต้องการทดสอบเขียนโปรแกรมรับส่ง อินพุท/เอาท์พุท และต้องการเข้าใจถึงรีจีสเตอร์ภายในมากขึ้น

4.3.1.2 การทดลองที่ 2

ในการทดลองที่ 2 นั้นมีเงื่อนไขคล้ายการทดลองที่หนึ่ง แต่จะนำโปรแกรมที่ได้ไปทำงานบนเครื่องเกมบอยแอดวานซ์จริงเพื่อทดสอบ อินพุท/เอาท์พุท และรีจีสเตอร์ภายใน

4.3.1.3 การทดลองที่ 3

ในการทดลองนี้ต้องการทดลองสร้างระบบปฏิบัติการ eCos ในหลายๆแบบและทดสอบ โดยการสร้างโปรแกรมมาทำงานบนระบบปฏิบัติการ eCos แล้วนำไปใช้งานบนเครื่องเกมบอยแอดวานซ์จริง ทำการทดลองสร้างไมโครซีลีนุกซ์ และตรวจสอบข้อผิดพลาด

4.3.1.4 การทดลองที่ 4

ในการทดลองนี้จะเป็นการทดลองทำให้เครื่องเกมบอยแอดวานซ์กลายเป็นเครื่องมือที่ใช้ในการกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ โดยทำการติดต่อกับอุปกรณ์ Com86 ที่ทำหน้าที่ควบคุมการปิดเปิดไฟและพัดลม

4.3.2 ผลการทดลอง

4.3.2.1 ผลการทดลองที่ 1

โปรแกรม	การทดสอบ	ผลการทดสอบ
Virtual Keyboard (Text mode)	ทดสอบการรับอินพุทจากปุ่มต่างๆโดยเลือกตัวอักษรบน Virtual Keyboard	สามารถรับค่าจากปุ่มต่างๆ ได้
	ทดสอบการแสดงเอาท์พุท โดยเมื่อรับค่าจากปุ่มต่างๆแล้วให้แสดงผลการเลือกอักษรนั้นๆ	สามารถแสดงเอาท์พุททางอีมูเลเตอร์ได้
Picture motion (Graphic mode)	ทดสอบการรับอินพุทจากปุ่มต่างๆ	สามารถรับค่าจากปุ่มต่างๆได้
	ทดสอบการแสดงเอาท์พุทเป็นรูปภาพ	สามารถแสดงเอาท์พุทเป็นรูปภาพได้

ตารางที่ 4-1 แสดงผลการทดลองที่ 1

4.3.2.2 ผลการทดลองที่ 2

โปรแกรม	การทดสอบ	ผลการทดสอบ
Virtual Keyboard (Text mode)	ทดสอบการรับอินพุทจากปุ่มต่างๆ โดยเลือกตัวอักษรบน Virtual Keyboard	สามารถรับค่าจากปุ่มต่างๆ ได้ แต่มีความเร็วมากกว่าบนอีมูเลเตอร์
	ทดสอบการแสดงเอาท์พุท โดยเมื่อรับค่าจากปุ่มต่างๆ แล้วให้แสดงผลการเลือกอักษรนั้นๆ	สามารถแสดงเอาท์พุททางจอภาพได้
Picture motion (Graphic mode)	ทดสอบการรับอินพุทจากปุ่มต่างๆ	สามารถรับค่าจากปุ่มต่างๆได้ แต่มีความเร็วมากกว่าบนอีมูเลเตอร์
	ทดสอบการแสดงเอาท์พุทเป็นรูปภาพ	สามารถแสดงเอาท์พุทเป็นรูปภาพได้

ตารางที่ 4-2 แสดงผลการทดลองที่ 2

จากผลการทดลองพบว่าการทำงานของอีมูเลเตอร์กับเครื่องเกมบอยแอดวานซ์นั้นจะคล้ายกันมาก มีความแตกต่างกันเล็กน้อยจากโปรแกรมปกติที่ทำงานได้บนอีมูเลเตอร์เมื่อทำการเพิ่มค่าหน่วงเวลาก็จะสามารถทำงานได้อย่างปกติบนเครื่องเกมบอยแอดวานซ์

4.3.2.3 ผลการทดลองที่ 3

ในการทดลองนี้กำหนดแนวทางการกำหนดค่าคุณสมบัติของ eCos ไว้ดังนี้คือ เลือกค่าคุณสมบัติในการสร้าง eCos ให้ทำงานแตกต่างกัน โดยเลือกที่แม่แบบต่างๆกัน แบบแรกจะเป็นแม่แบบที่เป็นค่าเริ่มต้น และแบบที่สองจะเป็นแม่แบบที่เล็กที่สุด จากนั้นเรียกทำงานโปรแกรมที่ใช้ตรวจสอบการใช้หน่วยความจำขณะทำงาน

Templates	ความสามารถในการทำงานโปรแกรมทดสอบ	หน่วยความจำที่เหลือ
Default	สามารถทำงานโปรแกรมทดสอบได้	237 KB
Minimal	ไม่สามารถคอมไพล์ผ่านได้ เนื่องจากขาดแพกเกจในการ malloc ไป	-

ตารางที่ 4-3 แสดงผลการทดลองที่ 3

ในการทดลองนี้ถ้าไม่ใช้แม่แบบ สามารถกำหนดแพกเกจเองได้ จากการทดลองนี้ทำให้ทราบได้ว่าขนาดของหน่วยความจำที่ใช้ไปกับระบบปฏิบัติการกับโปรแกรมคือ 288-237 = 51 KB

4.3.2.4 ผลการทดลองที่ 4

ในการทดลองนี้จะทดลองทำให้เครื่องเกมบอยแอดวานซ์เป็นเครื่องมือที่ใช้ในการกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ โดยจะสร้างโปรแกรมขึ้นมาเพื่อติดต่อกับบอร์ด Com86 ที่ทำหน้าที่ควบคุมการเปิดปิดของไฟและพัดลม

วิธีการทดสอบ	ผลการทดสอบ
ทำการเชื่อมต่อกับ Com86 เพื่อดูค่าปัจจุบัน	สามารถดูค่าปัจจุบันใน Com86 ได้
ทำการกำหนดค่าต่างๆ ให้กับ Com86	สามารถกำหนดค่าต่างๆ ให้กับ Com86 ได้

ตารางที่ 4-4 แสดงผลการทดลองที่ 4

จากการทดลองทำให้ทราบว่าเครื่องเกมบอยแอดวานซ์สามารถนำมาใช้เป็นเครื่องมือกำหนดค่าคุณสมบัติของอุปกรณ์อื่นๆ ได้

บทที่ 5
บทวิจารณ์และสรุปผล

5.1 ผลที่ได้รับจากโครงงาน

1. ไมโครซีลีนุกซ์ ที่ถูกพัฒนาขึ้นมาให้เข้ากับเครื่องเกมบอยแอดวานซ์ที่มีความเข้ากันได้เกือบจะสมบูรณ์ ดังนี้

- สามารถบูตเคอร์เนล โดยมีเอาต์พุตออกหน้าจอ
- สามารถจองเมมโมรีสำหรับเคอร์เนลได้แล้ว

2. วิธีการพอร์ตลีนุกซ์ให้สามารถทำงานบนระบบ embedded ได้

5.2 ปัญหาที่พบ

โครงงานนี้มีส่วนที่จำเป็นต้องทำการศึกษาค้นคว้าอยู่มาก แต่เนื่องจากโครงงานในลักษณะนี้ยังมีผู้พัฒนาน้อย และผู้พัฒนายังมีทักษะด้านนี้น้อย ส่งผลสะท้อนถึงปัญหาที่พบในโครงงานมีมาก ซึ่งสามารถแบ่งออกมา 3 ส่วนใหญ่ๆ

- ปัญหากับระบบ embedded
- ปัญหากับไมโครซีลีนุกซ์
- ปัญหาเกี่ยวกับฮาร์ดแวร์

5.2.1 ปัญหากับระบบ embedded

- เนื่องจากหน่วยความจำที่ใช้กับเครื่องเกมบอยแอดวานซ์ มีหน่วยความจำเพียง 256 Kb และไม่สามารถที่จะเพิ่มขนาดได้ ซึ่งถ้าต้องการ ความจำที่เพิ่มขึ้น อาจไม่เพียงพอได้
- เครื่องมือในการติดต่อผ่านเทอมินอล ต้องสร้างขึ้นมาเอง
- ผู้พัฒนาขาดทักษะด้านความรู้ เมื่อเกิดปัญหาจึงใช้เวลาเรียนรู้เป็นเวลานาน

5.2.2 ปัญหากับไมโครซีลีนุกซ์

- เนื่องจากยังไม่มีไมโครซีลีนุกซ์ที่ทำงานบนเครื่องเกมบอยแอดวานซ์จึงต้องเสียเวลาเรียนรู้ค่อนข้างเป็นเวลาที่นาน ทั้งฮาร์ดแวร์ และซอฟต์แวร์

5.2.3 ปัญหาเกี่ยวกับฮาร์ดแวร์

- เนื่องจากเกมบอยแอดวานซ์ ไม่มีการเปิดเผยข้อมูล จึงยากแก่การศึกษาและค้นคว้าความรู้ด้านฮาร์ดแวร์

5.2.4 ปัญหาเกี่ยวกับอีมูเลเตอร์

เนื่องจากในระยะแรกนั้นเรายังไม่ได้เครื่องเกมบอยแอดวานซ์ และตัวตลับ Xport ในการทำงานจริง เพราะฉะนั้นเราจึงได้ทำการจำลองการทำงานบนเครื่องเกมบอยแอดวานซ์ ทั้งนี้เมื่อได้รับเครื่องเกมบอยแอดวานซ์และตัวตลับแล้วนั้น เมื่อเรานำมาทำงาน เราได้พบว่าอีมูเลเตอร์นั้นไม่มีความสามารถเทียบเท่ากับเครื่องเกมบอยจริง ทั้งนี้เนื่องมาจากการทำงานที่โค้ดเดียวกัน แต่ผลการทำงานนั้นไม่เหมือนกัน เช่น เกมบอยจริงนั้นสามารถทำงานได้สถานะที่ไกลกว่าสถานะที่อีมูเลเตอน์สามารถทำงานได้

5.3 สรุปผล

ในการพัฒนาต่อไปนั้น เราสามารถพัฒนาให้ไมโครซีลีนุกซ์ทำงานให้เข้ากับสภาพแวดล้อมได้ แต่เนื่องจากปัญหาเรื่องความจุของเมมโมรี่นั้นที่มีน้อยเกินไป อาจจะเป็นปัญหาได้ เมื่อทำงานได้สำเร็จแล้ว อาจจะไม่มีเมมโมรี่เหลือพอที่จะสร้างแอพพลิเคชันได้ แต่ถึงอย่างไรก็ตามเราก็ควนจะหาทางในการพัฒนาต่อไป ตามแนวทางของผู้จัดทำ

ซึ่งขณะนี้ ไมโครซีลีนุกซ์ ยังไม่สามารถสตาร์ทเชลล์ได้ เนื่องจากยังไม่สามารถจองเมมโมรีสำหรับ kernel และ data

ในด้านการนำไปประยุกต์ใช้นั้น จากการทดลองทั้งหมดโดยเฉพาะการทดลองสุดท้ายทำให้ทราบว่าเครื่องเกมบอยแอดวานซ์สามารถนำมาใช้เป็นเครื่องมือในการกำหนดค่าคุณสมบัติอุปกรณ์อื่นๆ ได้ อย่างมีประสิทธิภาพและสามารถช่วยลดต้นทุนในการสร้างอุปกรณ์ระบบฝังตัวได้อีกด้วย

แต่ในท้ายที่สุดแล้ว โครงงานนี้สามารถเป็นแนวทางในการที่จะพัฒนาต่อไปจนสำเร็จ เนื่องจากเหลืออีกไม่มาก โดยสำหรับ ผลงานที่ได้จากโครงงานนี้ สามารถทำให้ไมโครซีลีนุกซ์ติดต่อกับระดับฮาร์ดแวร์ของเกมบอยแอดวานซ์ไม่ว่าจะเป็น การเอาท์พุตตัวอักขระออกหน้าจอ การติดต่อกับอินเทอร์รัพได้ สามารถติดต่อพอร์ตอินพุตเอาท์พุตได้การมองเห็นตำแหน่งเมมโมรีต่างๆ ไม่ว่าจะเป็น External RAM หรือ Internal RAM หรือตำแหน่งของตลับก็ตาม และสามารถสตาร์ทโค้ดในไฟล์ head-arm-gba.S ที่เป็นไฟล์แอสเซมบลี ผสมกับซี ซึ่งเข้าถึงฮาร์ดแวร์โดยตรง

แต่ในทั้งนี้ ทางผู้ดำเนินโครงงานได้สร้าง แอพพลิเคชันที่ทำให้เครื่องเกมบอยแอดวานซ์เป็นเสมือน เทอมินอล โดยสามารถเป็นอินพุตเอาต์พุตกับดีไวซ์ต่างๆได้ โดยแอพพลิเคชันที่ทำมานำเสนอเป็นการทำงานติดต่อกันระหว่าง บอร์ดคอม86 กับเครื่องเกมบอยแอดวานซ์ โดยไม่มีคอมพิวเตอร์มาเกี่ยวข้อง

แต่ในความเป็นจริงที่นักพัฒนายังขาดความรู้ด้านนี้ จึงทำให้ผลงานออกมาไม่ได้อย่างที่หวังไว้ แต่อย่างไรก็ตาม ทางผู้ดำเนินงาน คาดหวังเป็นอย่างยิ่งว่า ความรู้ต่างๆที่ได้ทำการค้นคว้าและวิจัยออกมานั้นจะเป็นประโยชน์สำหรับผู้ที่สนใจในด้านนี้ไม่มากก็น้อย และสามารถเป็นแนวทางในการพัฒนาระบบลีนุกซ์ฝังตัวต่อไป ถึงแม้ต้องใช้เวลานานในการศึกษาและค้นคว้าเกี่ยวกับหลักการและแนวทางที่ใช้ในการพัฒนา แต่ผลที่ได้รับกลับมาก็คุ้มค่ากับเวลาที่เสียไป

ภาคผนวก ก. การสร้างคอมไพเลอร์สำหรับไมโครซีลีนุกซ์

เนื่องด้วยระบบปฏิบัติการที่ทางผู้จัดทำได้เลือกใช้คือ ระบบปฏิบัติการไมโครซีลีนุกซ์ ซึ่งระบบปฏิบัติการไมโครซีลีนุกซ์นั้นในการคอมไพล์ไม่สามารถที่จะใช้ตัวคอมไพลเลอร์ทั่วไปได้ ซึ่งได้ภาคผนวกในส่วนนี้จะเสนอการสร้างคอมไพเลอร์สำหรับระบบปฏิบัติการไมโครซีลีนุกซ์

โดยในส่วนคอมไพเลอร์จะประกอบด้วยส่วนต่างๆดังนี้

1. EGCS (Experimental GNU Compiler System)
2. Binutils
3. newlib-1.8.1
4. elf2flt

ซึ่งส่วนประกอบต่างๆนั้นสามารถดาวน์โหลดได้จาก www.uclinux.org

โดยการจัดทำมีขั้นตอนดังนี้

1. แตกไฟล์ egcs โดยใช้คำสั่ง tar -xzf egcs.tar.gz
2. แตกไฟล์ binutils โดยใช้คำสั่ง tar -xzf uclinux-binutils.ta.gz
3. แตกไฟล์ newlib โดยใช้คำสั่ง tar -xzf newlib-1.8.1.tar.gz
4. สร้างไดเรกทอรีสำหรับ binutils โดยใช้คำสั่ง mkdir cross-binutils
5. คอนฟิกไฟล์ binutils และติดตั้ง binutils โดยเข้าไปยังไดเรกทอรีที่สร้างขึ้นแล้วใช้คำสั่งดังนี้

   (1) ../binutils/configure --target=arm-elf --prefix=/usr/local/arm-elf --with-include-gettext

   (2) make all

   (3) make install
6. สร้างไดเรกทอรีสำหรับ egcs โดยใช้คำสั่ง mkdir cross-gcc
7. คอนฟิกไฟล์ gcc และติดตั้ง gcc โดยเข้าไปยังไดเรกทอรีที่สร้างขึ้นแล้วใช้คำสั่งดังต่อไปนี้

   (1) ../egcs/configure --target=arm-elf --prefix=/usr/local/arm-elf --with-newlib --with-headers=../newlib-1.8.1/newlib/libc/include --enabled-languages=c,c++ --with-cpu=arm7tdmi --with-include-gettext --with-gnu-ld --with-gnu-as

   (2) make all-gcc

   (3) make install-gcc
8. สร้างไดเรกทอรีสำหรับ newlib โดยใช้คำสั่ง mkdir cross-newlib
9. คอนฟิกไฟล์ newlib และติดตั้ง โดยเข้าไปยังไดเรกทอรีที่สร้างขึ้นแล้วใช้คำสั่งต่อไปนี้

   (1) ../newlib-1.8.1/configure --target=arm-elf --prefix=/usr/local/arm-elf --with-include-gettext

   (2) make all

   (3) make install

10. เข้าไปยังไดเรกทอรี gcc และทำการติดตั้งอีกครั้งหนึ่งโดยใช้คำสั่งดังต่อไปนี้

(1) make all

(2) make install

11. แตกไฟล์ elf2flt โดยใช้คำสั่งดังต่อไปนี้ tar -xzf elf2flt.tar.gz

12. ทำการคอนฟิกและติดตั้งโดนใช้คำสั่งต่อไปนี้

(1) ./configure (และทำการคอนฟิกค่าไปยังที่ที่ได้สร้าง lib ไว้ก่อนหน้านี้)

(2) make all

(3) make install

เมื่อเสร็จขั้นตอนเราจะได้ตัวคอมไพล์เลอร์สำหรับไมโครซีลีนุกซ์เรียบร้อย โดยเข้าไปคอนฟิกค่าใน makefile ของไมโครซีลีนุกซ์โดยให้ตั้งค่าตัวคอมไพล์เลอร์ที่เราได้สร้างมาเป็นตัวคอมไพล์ โดยแทรกคำสั่ง CROSS-COMPILE=/usr/local/arm-elf/bin/arm-elf-

ภาคผนวก ข. แนวทางการสร้างระบบปฏิบัติการ eCos

แนวทางการสร้างระบบปฏิบัติการ eCos และการกำหนดค่าคุณสมบัติต่างๆ

1. สำหรับระบบปฏิบัติการลีนุกซ์

1.1 ใช้ระบบ CVS (Concurrent Versions System) ในการดึงข้อมูลจากเครือข่ายเพื่อให้ได้ข้อมูลที่ทันสมัย หรือ ใช้ข้อมูลคำสั่งจากแผ่น CD-ROM แผ่นที่2

- ใช้ CVS ในการดึงข้อมูลคำสั่งจากเครือข่ายโดยใช้คำสั่ง ที่ command prompt

  $ cvs –d:pserver:anonymous@cvs.gbaxport.sourceforge.net:/cvsroot/gbaxport login

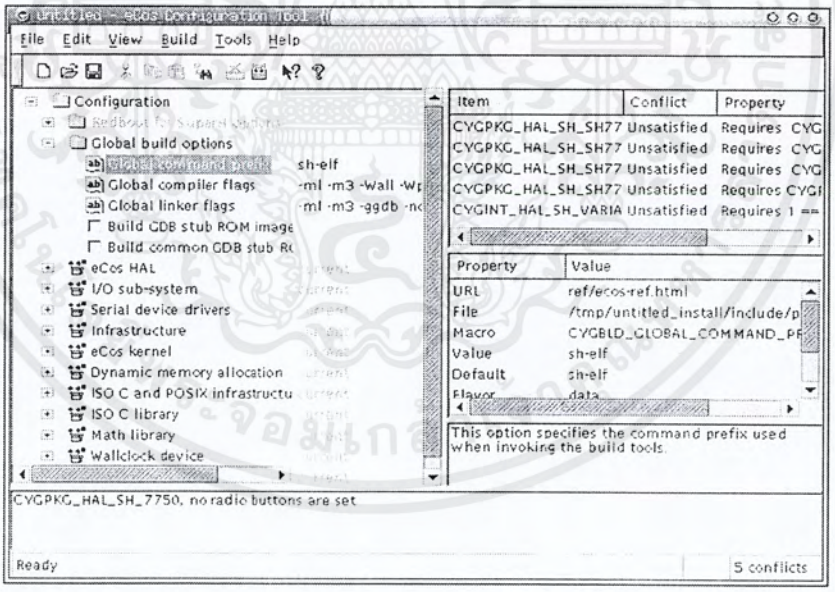
  เมื่อมีการขอรหัสผ่าน (Password) สำหรับ anonymous ให้กดปุ่ม Enter จากนั้นใส่คำสั่ง

  $ cvs –z3 –d:pserver:anonymous@cvs.gbaxport.sourceforge.net:/cvsroot/gbaxport co ecos

  จากนั้นรอจนดึงข้อมูลเสร็จโดยข้อมูลมีขนาด 90 เมกกะไบต์
- ใช้ข้อมูลจาก CD-ROM แผ่นที่2 โดยทำการคัดลอกข้อมูลทั้งหมดไปยังสารบบที่ต้องการ

1.2 ทำการกำหนดค่าคุณสมบัติต่างๆ ของระบบปฏิบัติการ eCos โดยใช้โปรแกรม Configtool หรือกำหนดค่าคุณสมบัติโดยใช้คำสั่ง ecosconfig ที่ command line

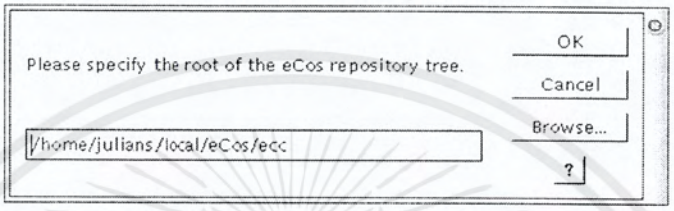
- Configtool เป็นโปรแกรมช่วยในการกำหนดค่าคุณสมบัติรวมไปถึงช่วยในการสร้างระบบปฏิบัติการด้วย สามารถเรียกใช้งานได้บนระบบ X windows หรือเรียกโปรแกรมด้วยคำสั่งที่ command line



รูปที่ ข-1 แสดงโปรแกรม Configtool ในระบบปฏิบัติการลีนุกซ์

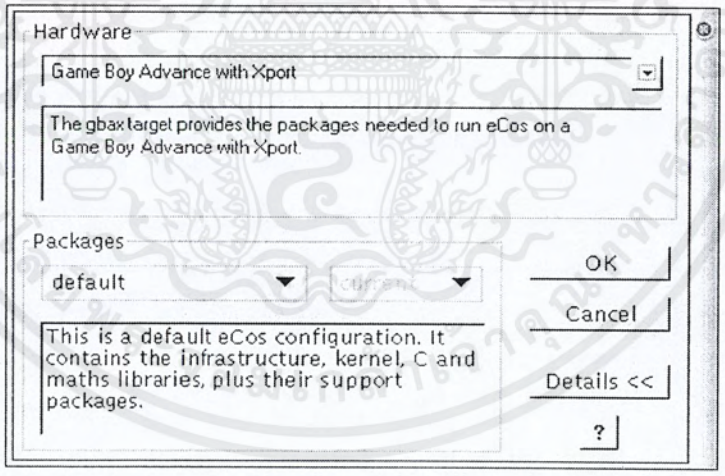
การใช้งาน Configtool นั้นมีขั้นตอนดังนี้

1.2.1 กำหนด Repository หรือ แหล่งเก็บข้อมูลคำสั่ง (Code) ทั้งหมดโดยกำหนดไปยังสารบบที่มีข้อมูลคำสั่งทั้งหมดของ eCos ซึ่งการกำหนดแหล่งเก็บข้อมูลคำสั่งสามารถเปลี่ยนแปลงหรือแก้ไขได้ที่ Build -> Repository... จากตัวอย่าง สารบบที่มีข้อมูลคำสั่งทั้งหมดของ eCos คือ "/home/julians/local/eCos/ecc" (หมายเหตุ: กำหนดไปยังสารบบที่มี สารบบย่อยชื่อ 'src' อยู่) ดังรูป ข-2



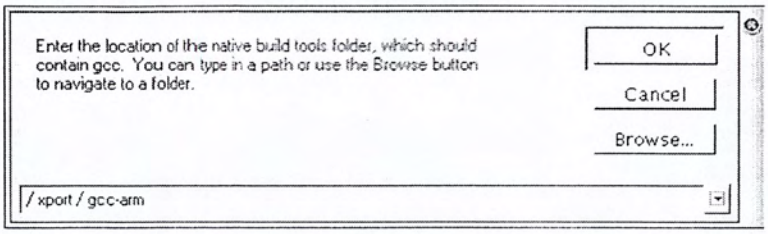
รูปที่ ข-2 แสดงการกำหนด Respository

1.2.2 กำหนดค่าคุณสมบัติต่างๆ โดยการเลือกแม่แบบแล้วทำการแก้ไขให้ได้คุณสมบัติตามต้องการ โดยการเลือกแม่แบบ ซึ่งสามารถทำได้ที่ Build -> Templates จากนั้นทำการเลือกแม่แบบที่ต้องการ ดังรูป ข-3



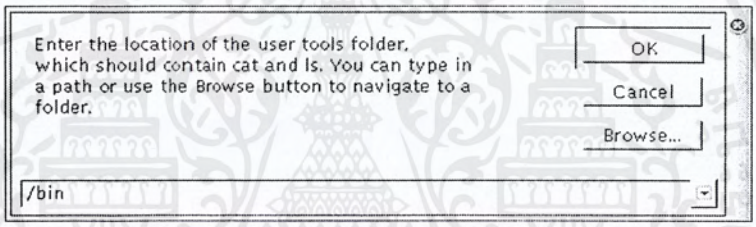
รูปที่ ข-3 แสดงการกำหนดคุณสมบัติต่างๆ

1.2.3 กำหนดค่าเครื่องมือในการคอมไพล์ เป็นการตั้งค่าสารบบไปยังเครื่องมือที่ใช้ในการสร้างระบบปฏิบัติการ สามารถตั้งค่าได้ที่ Tools -> Paths -> Build Tools... (หมายเหตุ: กำหนดไปยังสารบบที่เก็บคำสั่ง gcc, ld, etc.) ดังรูป ข-4



รูปที่ ข-4 แสดงการกำหนดค่าเครื่องมือในการคอมไพล์

1.2.4 กำหนดค่าเครื่องมือของผู้ใช้งานเพื่อใช้เป็นตัวช่วยในการ คอมไพล์ เช่นการแสดงผลการคอมไพล์ ต่างๆ สามารถตั้งค่าได้ที่ Tools -> Paths -> User Tools... (หมายเหตุ: กำหนดไปยังสารบบที่เก็บคำสั่ง cat, ls, etc.) ดังรูป ข-5



รูปที่ ข-5 แสดงการกำหนดไดเรกทอรีที่เก็บคำสั่งต่างๆ

เมื่อทำการตั้งค่าต่างๆเสร็จเรียบร้อยเครื่องมือจึงพร้อมใช้งาน เมื่อทำการปรับค่าคุณสมบัติต่างๆ ดังในรูปที่ ข-1 เสร็จเรียบร้อย จึงทำการสร้างระบบปฏิบัติการ eCos โดยเลือกคำสั่ง Build -> Library หรือ กดปุ่ม F7 จากนั้นรอจนเครื่องมือทำการสร้างเสร็จเรียบร้อย

- ecosconfig เป็นคำสั่งที่ให้ผลการทำงานเหมือนกับ Configtool แต่มีเสถียรภาพมากกว่า และแน่นอนว่ามีการใช้งานยากกว่าเนื่องจากเป็น command line

คำสั่ง ecosconfig มีโครงสร้างดังนี้

$ ecosconfig <qualifiers> <command>

ตัวอย่างคำสั่งการสร้าง eCos สำหรับเครื่องเกมบอยแอดวานซ์มีดังนี้

$ ecosconfig new gbax

$ ecosconfig tree

$ make

2. สำหรับระบบปฏิบัติการวินโดวส์

2.1 ทำการลง cygwin ซึ่งจะทำให้ได้สภาพแวดล้อมของ UNIX บน Windows

2.2 ใช้ระบบ CVS (Concurrent Versions System) ในการดึงข้อมูลจากเครือข่ายเพื่อให้ได้ข้อมูลที่ทันสมัย หรือ ใช้ข้อมูลคำสั่งจากแผ่น CD-ROM แผ่นที่2

- ใช้ CVS ในการดึงข้อมูลคำสั่งจากเครือข่ายโดยใช้คำสั่ง ที่ command prompt

  S cvs –d:pserver:anonymous@cvs.gbaxport.sourceforge.net:/cvsroot/gbaxport login

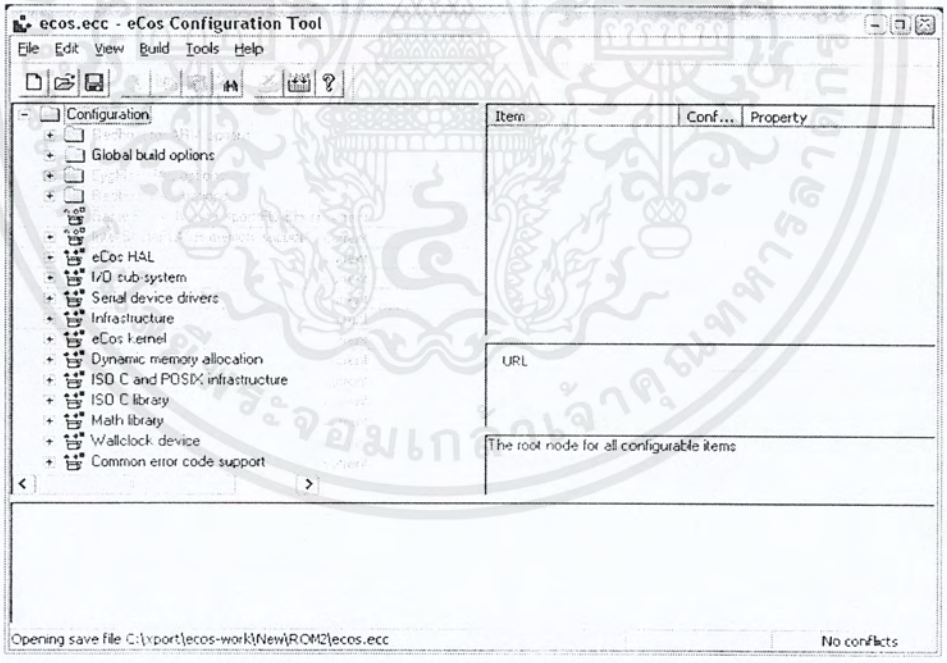
  เมื่อมีการขอรหัสผ่าน (Password) สำหรับ anonymous ให้กดปุ่ม Enter จากนั้นใส่คำสั่ง

  S cvs –z3 –d:pserver:anonymous@cvs.gbaxport.sourceforge.net:/cvsroot/gbaxport co ecos

  จากนั้นรอจนดึงข้อมูลเสร็จโดยข้อมูลมีขนาด 90 เมกกะไบต์
- ใช้ข้อมูลจาก CD-ROM แผ่นที่2 โดยทำการคัดลอกข้อมูลทั้งหมดไปยังสารบบที่ต้องการ

2.3 ทำการกำหนดค่าคุณสมบัติต่างๆของระบบปฏิบัติการ eCos โดยใช้โปรแกรม Configtool หรือ กำหนดค่าคุณสมบัติโดยใช้คำสั่ง ecosconfig ที่ command line บน cygwin ดังรูป ข-6

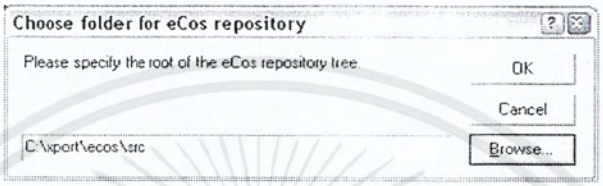
- Configtool เป็นโปรแกรมช่วยในการกำหนดค่าคุณสมบัติรวมไปถึงช่วยในการสร้างระบบปฏิบัติการด้วย



รูปที่ ข-6 แสดงโปรแกรม Configtool ในระบบปฏิบัติการวินโดวส์

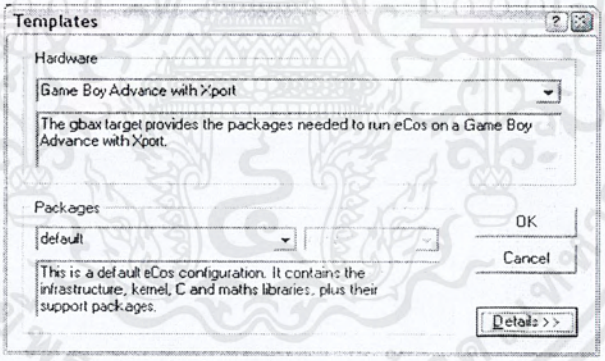
การใช้งาน Configtool นั้นมีขั้นตอนดังนี้

2.3.1 กำหนด Repository หรือ แหล่งเก็บข้อมูลคำสั่ง (Code) ทั้งหมดโดยกำหนดไปยังสารบบที่มีข้อมูลคำสั่งทั้งหมดของ eCos ซึ่งการกำหนดแหล่งเก็บข้อมูลคำสั่งสามารถเปลี่ยนแปลงหรือแก้ไขได้ที่ Build -> Repository... จากตัวอย่าง สารบบที่มีข้อมูลคำสั่งทั้งหมดของ eCos คือ "/home/julians/local/eCos/ecc" (หมายเหตุ: กำหนดไปยังสารบบที่มี สารบบย่อยชื่อ 'src' อยู่) ดังรูป ข-7



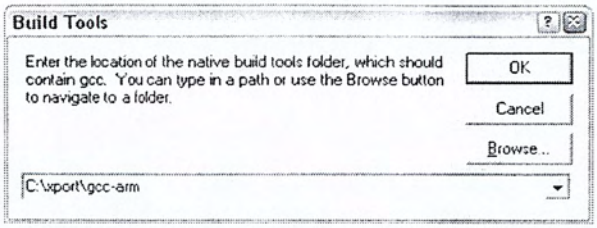
รูปที่ ข-7 แสดงการกำหนด Repository

2.3.2 กำหนดค่าคุณสมบัติต่างๆ โดยการเลือกแม่แบบแล้วทำการแก้ไขให้ได้คุณสมบัติตามต้องการ โดยการเลือกแม่แบบ ซึ่งสามารถทำได้ที่ Build -> Templates จากนั้นทำการเลือกแม่แบบที่ต้องการ ดังรูป ข-8



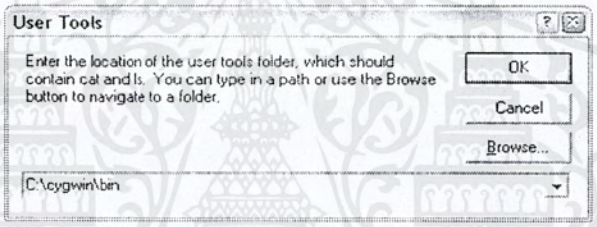
รูปที่ ข-8 แสดงการกำหนดคุณสมบัติต่างๆ

2.3.3 กำหนดค่าเครื่องมือในการคอมไพล์เป็นการตั้งค่าสารบบไปยังเครื่องมือที่ใช้ในการสร้างระบบปฏิบัติการ สามารถตั้งค่าได้ที่ Tools -> Paths -> Build Tools... (หมายเหตุ: กำหนดไปยังสารบบที่เก็บคำสั่ง gcc, ld, etc.) ดังรูป ข-9



รูปที่ ข-9 แสดงการกำหนดเครื่องมือในการคอมไพล์

2.3.4 กำหนดค่าเครื่องมือของผู้ใช้งานเพื่อใช้เป็นตัวช่วยในการคอมไพล์ เช่นการแสดงผลการคอมไพล์ ต่างๆ สามารถตั้งค่าได้ที่ Tools -> Paths -> User Tools... (หมายเหตุ: กำหนดไปยังสารบบที่เก็บคำสั่ง cat, ls, etc.) ดังรูป ข-10



รูปที่ ข-10 แสดงการกำหนดไดเรกทอรีที่เก็บคำสั่งต่างๆ

เมื่อทำการตั้งค่าต่างๆเสร็จเรียบร้อยเครื่องมือจึงพร้อมใช้งาน เมื่อทำการปรับค่าคุณสมบัติต่างๆ ดังในรูปที่ ข-1 เสร็จเรียบร้อยจึงทำการสร้างระบบปฏิบัติการ eCos โดยเลือกคำสั่ง Build -> Library หรือ กดปุ่ม F7 จากนั้นรอจนเครื่องมือทำการสร้างเสร็จเรียบร้อย

- ecosconfig เป็นคำสั่งที่สั่งงานบน cygwin ให้ผลการทำงานเหมือนกับ Configtool แต่มีเสถียรภาพมากกว่าและแน่นอนว่ามีการใช้งานยากกว่าเนื่องจากเป็น command line

  คำสั่ง ecosconfig มีโครงสร้างดังนี้

  $ ecosconfig <qualifiers> <command>

  ตัวอย่างคำสั่งการสร้าง eCos สำหรับเครื่องเกมบอยแอดวานซ์มีดังนี้

  $ ecosconfig new gbax

  $ ecosconfig tree

  $ make

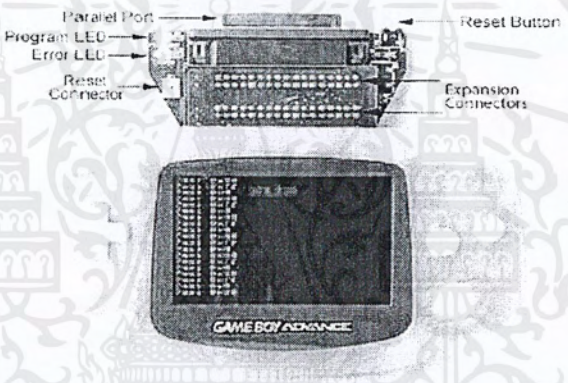
เมื่อทำการสร้างเสร็จสิ่งที่จะได้คือ library.a ซึ่งเป็นส่วนที่ระบบปฏิบัติการ eCos นี้จะต่างจากระบบปฏิบัติการอื่นๆ เนื่องจากระบบปฏิบัติการอื่นๆ นั้นเมื่อทำการ คอมไพล์ จะได้ข้อมูลที่เป็น data กับ image ซึ่งใช้ในการทำงานระบบปฏิบัติการนั้นๆ แต่สำหรับระบบปฏิบัติการ eCos แล้วจะได้ library.a ซึ่งจะเป็นการรวมเอาความสามารถต่างๆ ที่ได้กำหนดค่าคุณสมบัติก่อนการ คอมไพล์ ดังนั้น ระบบปฏิบัติการ eCos นี้จะยังไม่สามารถทำงานได้ถ้ายังไม่มีโปรแกรมเรียกใช้

ภาคผนวก ค. แนวทางการใช้งาน Xport

1. แนวทางการใช้งาน Xport

ความต้องการก่อนการใช้งาน Xport

- PC ที่ใช้ระบบปฏิบัติการ Windows กับ พอร์ตขนาน
- โปรแกรมที่มากับ Xport
- อุปกรณ์ Xport กับ Xport programmer
- เครื่องเกมบอยแอดวานซ์
- ทำการลงโปรแกรมที่มากับ Xport ทั้งนี้อยู่ใน CD-ROM แผ่นที่2 โดยเข้าไปยังสารบบ Compiler -> Xport จากนั้นทำการเรียกคำสั่ง setup.exe แล้วทำตามคำแนะนำจนเสร็จ
- เสียบ Xport programmer เข้ากับตัว Xport ทาง JP2 จากนั้นเสียบทั้งหมดเข้ากับช่องเสียบตลับของเครื่องเกมบอยแอดวานซ์ จะได้ดังรูป ค-1



รูปที่ ค-1 แสดงรายละเอียดเครื่องเกมบอยแอดวานซ์ และตลับ Xport

- ทำการต่อ Xport เข้ากับเครื่องคอมพิวเตอร์โดยใช้สาย DB-25 โดยปลายข้างหนึ่งเสียบที่พอร์ตขนานของเครื่องคอมพิวเตอร์ ส่วนอีกข้างเสียบกับ Xport ที่ช่องพอร์ตขนาน
- ทำการสร้างโปรแกรมแล้วทำการคอมไพล์ซึ่งจะกล่าวถึงต่อไปเมื่อทำการคอมไพล์เสร็จแล้วจะได้ข้อมูลที่สามารถทำงานได้อันได้แก่ Binary (*.bin), S-record (*.srec), Bitstream (*.bit)
- ทำการส่งข้อมูลที่ได้ทำการคอมไพล์เรียบร้อยแล้วไปยัง Xport โดยใช้โปรแกรม xppro.exe
- ถ้าไม่มีข้อผิดพลาดใดๆเมื่อปิดแล้วเปิดใหม่เครื่องเกมบอยแอดวานซ์จะทำงานตามโปรแกรมที่ได้สร้างไว้

2. วิธีการใช้โปรแกรม xppro.exe

- เรียกโปรแกรม command line ขึ้นมาโดยอาจจะใช้ cmd หรือ cygwin ก็ได้แล้วทำการตั้งค่าเส้นทาง (Path) ไปยังสารบบที่ได้ทำการลงโปรแกรมที่มากับ Xport ไว้ (หมายเหตุ: ที่สารบบนั้นจะมีโปรแกรม xppro.exe อยู่)
- ใช้คำสั่ง xppro.exe โดยคำสั่งนี้มีโครงสร้างดังนี้

>xppro (properties) (operations) (properties) ...

Operations มีคำสั่งทั้งหมดดังนี้

- –pf <Program file> เป็นการสั่งโปรแกรมข้อมูลไปยังหน่วยความจำแบบแฟลช ซึ่งการโปรแกรมแฟลชนั้นจะมีการตรวจสอบความถูกต้องด้วย
- –vf <Program file> เป็นการยืนยันความถูกต้องของข้อมูลโปรแกรม
- –pvf <Program file> เป็นการสั่งโปรแกรมข้อมูลลงหน่วยความจำแบบแฟลช พร้อมทั้งยืนยันความถูกต้องของข้อมูล
- –rf <Dump file> <length> เป็นการอ่านข้อมูลจากหน่วยความจำตามจำนวนความยาวที่กำหนดแล้วคัดลอกไปยัง Dump file
- –pe <Bitstream file> เป็นการโปรแกรม EEPROM ด้วย Bitstream file
- –ve <Bitstream file> เป็นการยืนยันความถูกต้องของ Bitstream file
- –pve <Bitstream file> เป็นการโปรแกรม EEPROM พร้อมทั้งตรวจเช็คความถูกต้องของ Bitstream file
- –re <Dump file> เป็นการอ่านและคัดลอกข้อมูลผ่านใน EEPROM ไปยัง Dump file ที่กำหนด
- –pl <Bitstream file> เป็นการโปรแกรม FPGA ด้วย Bitstream file แต่ไม่มีการโปรแกรมที่ EEPROM ดังนั้นเมื่อปิดแล้วเปิดเครื่องแล้วข้อมูลจะหายหมด
- –reset จะมีผลเหมือนปิดแล้วเปิดเครื่อง
- –execute จะมีผลเหมือนปิดแล้วเปิดเครื่อง
- –pause <milliseconds> จะหยุดการทำงานระหว่างคำสั่งเป็น milliseconds
- –loop <iterations> จะวนทำคำสั่งตามจำนวนครั้งที่ใส่ใน iterations
- –time จะแสดงผลเวลาที่ใช้เมื่อทำคำสั่งเสร็จ
- –version จะแสดงลำดับรุ่นของโปรแกรม xppro.exe
-

ตัวอย่าง

>xppro –ve redgreen10.bit –rf out.bin 0x200000 –pf redgreen.bin

จะทำการตรวจเช็คความถูกต้องของ redgreen10.bit แล้วทำการคัดลอกข้อมูลภายในแฟลชขนาด 2 เมกกะเวิร์ด หรือ 4 เมกกะไบต์ ไปยัง out.bin จากนั้นทำการโปรแกรม flash ด้วย redgreen.bin

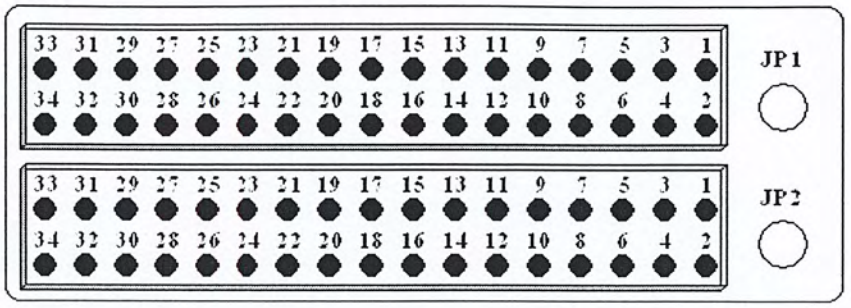
Properties มีคำสั่งทั้งหมดดังนี้

- -resetauto จะเป็นตัวกำหนดว่ามีการใส่การรีเซทอัตโนมัติ
- –portaddr <Address> (ใช้ใน Windows 9x กับ Me เท่านั้น) ใช้กำหนดตำแหน่งของพอร์ตขนานของเครื่องคอมพิวเตอร์ โดยปกติแล้วจะมีค่า 0x378
- –portnum <Value> (ใช้ใน Windows NT, 2000, XP) ใช้ในการกำหนดหมายเลขของพอร์ตขนาน โดยปกติแล้วจะใช้หมายเลข 0
- –debug <Debug level> ใช้กำหนดระดับขั้นในการแก้ไข ซึ่งปัจจุบันมีอยู่ 3 ระดับ ได้แก่ ระดับ 0 จะแสดงเพียงข้อความเมื่อเกิดเหตุการณ์ที่ล่อแหลมเท่านั้น, ระดับ 1 ซึ่งเป็นค่าพื้นฐานจะแสดงข้อความที่สำคัญๆ สำหรับผู้ใช้ทั่วไป, ระดับ 2 จะแสดงข้อมูลเป็นจำนวนมากซึ่งอาจมีประโยชน์ในการแก้ไข
- –delay <Delay value> เป็นการหน่วงเวลาก่อนที่จะอ่านข้อมูลหรือเขียนข้อมูลไปยัง พอร์ตขนาน ซึ่งระบบปฏิบัติการบางระบบต้องการ อย่างไรก็ตามการตั้งค่าหน่วงเวลานั้นไม่ควรตั้งค่าเกิน 500
- –model <Value> เป็นการกำหนดหมายเลขอุปกรณ์ Xport ซึ่งปกติมีค่า 1032
- –readdelay <Delay value> เหมือนกับคำสั่ง –delay แต่ใช้เฉพาะการอ่านจากพอร์ต
- –writedelay <Delay value> เหมือนกับคำสั่ง –delay แต่ใช้เฉพาะการเขียนสู่พอร์ต

3. อินพุท/เอาท์พุท ที่สามารถโปรแกรมได้

การที่เครื่องเกมบอยแอดวานซ์จะทำการดึงข้อมูลจากตลับเกมนั้นจำเป็นต้องมีการมัลติเพล็ก เนื่องจากจำนวนขาที่เชื่อมต่อระหว่างตลับเกมกับเครื่องเกมบอยแอดวานซ์นั้นไม่เท่ากันดังที่ได้กล่าวมาแล้วอย่างไรก็ตามการทำมัลติเพล็กนั้นจะใช้ FPGA's logic gates เพียงเล็กน้อยเท่านั้น ส่วนที่เหลือทั้งหมดสามารถโปรแกรมได้

ในส่วนของโปรแกรมที่ต้องการใช้ประโยชน์จาก อินพุท/เอาท์พุท นั้นมีส่วนของ PIO ที่สามารถโปรแกรมได้ผ่านทางขาเชื่อมต่อ JP1 กับ JP2 ซึ่งอาจใช้ภาษา VHDL ในการเขียนโปรแกรมอินพุท/เอาท์พุท โดยอนาคต Xport จะทำการพัฒนาอุปกรณ์ต่างๆ เพื่อใช้งานผ่านทางขาเชื่อมต่อ JP1 กับ JP2 เช่น USB, กล้องดิจิตอล, PCMCIA, Compact flash, Bluetooth, etc.



รูปที่ ค-2 แสดงรายละเอียดของสล็อตของ Xport

JP1

Pin	Signal	Description
1	Vcc	3.3 V
2	GND	0 V
3	P0	ใช้สำหรับโปรแกรม
4	P1	ใช้สำหรับโปรแกรม
5	P2	ใช้สำหรับโปรแกรม
6	P3	ใช้สำหรับโปรแกรม
7	P4	ใช้สำหรับโปรแกรม
8	P5	ใช้สำหรับโปรแกรม
9	P6	ใช้สำหรับโปรแกรม
10	P7	ใช้สำหรับโปรแกรม
11	P8	ใช้สำหรับโปรแกรม
12	P9	ใช้สำหรับโปรแกรม
13	P10	ใช้สำหรับโปรแกรม
14	P11	ใช้สำหรับโปรแกรม
15	P12	ใช้สำหรับโปรแกรม
16	P13	ใช้สำหรับโปรแกรม
17	P14	ใช้สำหรับโปรแกรม
18	P15	ใช้สำหรับโปรแกรม
19	P16	ใช้สำหรับโปรแกรม
20	P17	ใช้สำหรับโปรแกรม
21	P18	ใช้สำหรับโปรแกรม (LED สีแดง)

JP2

Pin	Signal	Description
1	Vcc	3.3 V
2	GND	0 V
3	P25	ใช้สำหรับโปรแกรม
4	P26	ใช้สำหรับโปรแกรม
5	P27	ใช้สำหรับโปรแกรม
6	P28	ใช้สำหรับโปรแกรม
7	P29	ใช้สำหรับโปรแกรม
8	P30	ใช้สำหรับโปรแกรม
9	P31	ใช้สำหรับโปรแกรม
10	P32	ใช้สำหรับโปรแกรม
11	P33	ใช้สำหรับโปรแกรม
12	P34	ใช้สำหรับโปรแกรม
13	P35	ใช้สำหรับโปรแกรม
14	P36	ใช้สำหรับโปรแกรม
15	P37	ใช้สำหรับโปรแกรม
16	P38	ใช้สำหรับโปรแกรม
17	P39	ใช้สำหรับโปรแกรม
18	P40	ใช้สำหรับโปรแกรม
19	P41	ใช้สำหรับโปรแกรม
20	P42	ใช้สำหรับโปรแกรม
21	P43	ใช้สำหรับโปรแกรม

22	P19	ใช้สำหรับโปรแกรม (LED สีเขียว)
23	P20	ใช้สำหรับโปรแกรม
24	P21	ใช้สำหรับโปรแกรม
25	P22	ใช้สำหรับโปรแกรม
26	P23	ใช้สำหรับโปรแกรม
27	P24	ใช้สำหรับโปรแกรม
28	CINIT	เป็นสัญญาณ init
29	CMODE	ใช้กำหนด Mode ของ FPGA
30	CPROGRAM	เป็นสัญญาณการโปรแกรม
31	CCLK	เป็นสัญญาณนาฬิกา
32	CCE	ทำให้ EEPROM ใช้งานได้
33	PDATA	ข้อมูลค่าคุณสมบัติของFPGA
34	SHIELD GND	ไม่มีการเชื่อมต่อ

22	P44	ใช้สำหรับโปรแกรม
23	P45	ใช้สำหรับโปรแกรม
24	P46	ใช้สำหรับโปรแกรม
25	P47	ใช้สำหรับโปรแกรม
26	P48	ใช้สำหรับโปรแกรม
27	P49	ใช้สำหรับโปรแกรม
28	P50	ใช้สำหรับโปรแกรม
29	P51	ใช้สำหรับโปรแกรม
30	P52	ใช้สำหรับโปรแกรม
31	P53	ใช้สำหรับโปรแกรม
32	P54	ใช้สำหรับโปรแกรม
33	P55	ใช้สำหรับโปรแกรม
34	SHIELD GND	ไม่มีการเชื่อมต่อ

ตารางที่ ค-1 แสดงหน้าที่ของขาเชื่อมต่อของ JP1 และ JP2

ภาคผนวก ง. การคอมไพล์โปรแกรมที่ทำงานบนเครื่องเกมบอยแอดวานซ์

1. แนวทางการคอมไพล์โปรแกรมที่ทำงานบนเครื่องเกมบอยแอดวานซ์

1.1 แนวทางการคอมไพล์โปรแกรมที่ทำงานบนเครื่องโดยตรง

1.1.1 ทำการลงโปรแกรมเครื่องมือที่ใช้สำหรับคอมไพล์ หรือ คอมไพเลอร์ที่ใช้เฉพาะสำหรับเครื่องเกมบอยแอดวานซ์ โดยมีขั้นตอนดังนี้ (ควรใช้ระบบปฏิบัติการ linux หรือใช้ cygwin)

- หาแหล่งข้อมูลคำสั่ง ซึ่งอาจจะใช้วิธีดึงข้อมูลจากอินเตอร์เน็ต หรือ จากแผ่น CD-ROM ที่มีข้อมูลคำสั่งที่เกี่ยวข้องเช่น binutils, gcc-arm, newlib, GDB และอื่นๆ
- ทำสร้างสารบบเพื่อเก็บข้อมูลคำสั่งเหล่านั้น และทำการกำหนดค่าคุณสมบัติต่างๆของเครื่องมือต่างๆด้วยคำสั่ง

  $configure [option] [target]
- ทำการสร้างตัวคอมไพเลอร์และไลบรารีขณะทำงาน
- ทำการติดตั้งตัวคอมไพเลอร์เข้าสู่เครื่องคอมพิวเตอร์ ด้วยคำสั่ง

  $make install

1.1.2 ทำการเขียนโปรแกรมตามต้องการตามความรู้เกี่ยวกับฮาร์ดแวร์ของเครื่องเกมบอยแอดวานซ์

1.1.3 ทำการคอมไพล์ด้วยโปรแกรมคอมไพเลอร์ เช่น gcc หรืออาจจะสร้าง Makefile ขึ้นมาช่วยในการคอมไพล์ก็ได้

1.1.4 ถ้าไม่มีความผิดพลาดใดๆเกิดขึ้นโดยปกติแล้วจะได้ Binary (*.bin) สามารถส่งข้อมูลไปทำงานบนเครื่องเกมบอยแอดวานซ์ หรือทดลองทำงานบนอีมูเลเตอร์ได้

1.2 แนวทางการสร้างโปรแกรมที่ทำงานบน eCos

1.2.1 ทำการลงโปรแกรมที่มากับ Xport หรือจาก CD-ROM

1.2.2 เขียนโปรแกรมตามความต้องการต่างๆ โดยใช้ไลบรารีจาก eCos ที่มีให้

1.2.3 ทำการคอมไพล์โปรแกรมด้วยตัวคอมไพล์เฉพาะสำหรับอุปกรณ์โดยใช้คอมไพเลอร์ที่มีมากับอุปกรณ์ Xport หรือจาก CD-ROM โดยมี flag ดังนี้

```
CFLAGS = -IC:(Path to cross-compile include) –I(Path to eCos include) -c -g -fno-exceptions
         -ffunction-sections -fdata-sections -Wall -save-temps -fverbose-asm –nostdlib
LDFLAGS = -nostdlib -nostartfiles -Ttarget.ld –L(Path to eCos library) -LC:(Path to gcc
          command) -Wl,--gc-sections -Wl,--static
```

ดังตัวอย่างต่อไปนี้

```
CFLAGS = -IC:/xport/gcc-arm/include -I../install/include -c -g -fno-exceptions -ffunction
         -sections -fdata-sections -Wall -save-temps -fverbose-asm –nostdlib
LDFLAGS = -nostdlib -nostartfiles -Ttarget.ld -L../install/lib -LC:/xport/gcc-arm -Wl,
          --gc-sections -Wl,--static
```

จากนั้นสั่ง คอมไพล์ ดังนี้

```
gcc $(CFLAGS) $(TARGET).c   <-- จะได้ $(TARGET).o
gcc -g -o $(TARGET).elf $(TARGET).o $(LDFLAGS)   <-- จะได้ $(TARGET).elf
objcopy -O srec $(TARGET).elf $(TARGET).srec   <-- จะได้ $(TARGET).srec
```

จากนี้จะได้ข้อมูล srec ซึ่งจะเป็นข้อมูลที่สามารถทำงานได้จริงบนอุปกรณ์ อีกทั้งมีความสามารถต่างๆ ของระบบปฏิบัติการที่ได้ทำการกำหนดค่าคุณสมบัติไว้

บรรณานุกรม

[1] Scott Maxell, *LINUX Core Kernel* The Coriolis Group, 1999

[2] John Lombardo, *Embedded Linux* New Riders,2001

[3] Xport, Charmed Labs Inc. (http://www.charmedlabs.com)

[4] Gameboy Advance Development Organization. (http://www.gbadev.org)

[5] Sergej Kravcenko, Codewaves. (http://codewaves.com/gba)

[6] Jeff, Dev'rs. (http://www.devrs.com/gba)

[7] Dovoto, The Pern Project. (http://www.thepernproject.com)

[8] Guyfawkes, Kojote, Costis, GBAEmu. (http://www.gbaemu.com)

[9] Devkit Advance. (http://devkitadv.sourceforge.net/index.html)

[10] CFXweb. (http://www.cfxweb.net)

[11] VisualBoy Advance Home Page. (http://vboy.emuhq.com)

[12] Bottled Light, Inc. (http://www.bottledlight.com)

[13] BATGBA. (http://batgba.zophar.net)

[14] Tom Happ, CowBite. (http://cowbite.emuunlim.com)

[15] Sean Reid. (http://www.seanreid.ca)

[16] Red hat, Inc. (http://www.redhat.com/embedded/technologies/ecos)

[17] eCos site. (http://sources.redhat.com/ecos)

[18] GNU, GCC Home Page. (http://gcc.gnu.org)

[19] LARTware, ARM Linux Cross Compiler. (http://www.lart.tudelft.nl/lartware/compile-tools)

[20] Embedded Linux Microcontroller Project, uCliux. (http://www.uclinux.org)

[21] uClinux Directory. (http://uclinux.home.at)

[22] C library for embedded systems. (http://www.uclibc.org)

[23] The embedded Linux portal. (http://www.linuxdevices.com)

[24] Embedded Linux Developer Forum. (http://www.ucdot.org)

[25] Linux-Community. (http://www.linux-community.de)

[26] ARM Ltd Home Page. (http://www.arm.com/armtech/cpus?OpenDocument)

[28] eLinux Home Page. (http://www.elinux.com)