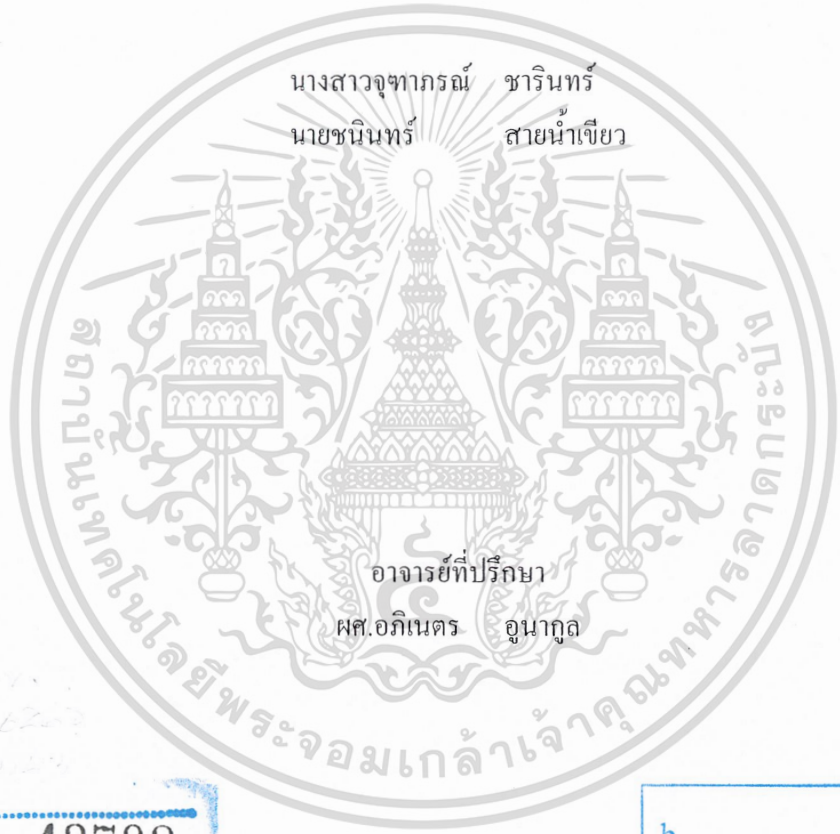
ระบบลีนุกซ์แบบเรียลไทม์

Real-Time Linux Systems

นางสาวจุฑาภรณ์ ชารินทร์
นายชนินทร์ สายน้ำเขียว



อาจารย์ที่ปรึกษา
ผศ.อภิเนตร อูนากูล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบลีนุกซ์แบบเรียลไทม์

Real-Time Linux Systems

ผู้จัดทำ

1. นางสาวจุฑาภรณ์ ชารินทร์ รหัสประจำตัว 41013525
2. นายชนินทร์ สายน้ำเขียว รหัสประจำตัว 41013526

_____ อาจารย์ที่ปรึกษา

(ผศ.อภิเนตร อูนากูล)

ระบบลีนุกซ์แบบเรียลไทม์

นางสาวจุฑาภรณ์ ชารินทร์ 41013525
นายชนินทร์ สายน้ำเขียว 41013526
ผศ.อภิเนตร อูนากุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในปัจจุบันระบบ embedded ถูกนำมาใช้งานอย่างกว้างขวาง โดยเฉพาะถูกนำไปใช้กับระบบเรียลไทม์ ยกตัวอย่างเช่น ระบบควบคุม ระบบสื่อผสม ฯลฯ ในการพัฒนางานแบบเรียลไทม์นั้นมีระดับของความซับซ้อนอยู่หลายระดับ และจำเป็นต้องคำนึงถึงเงื่อนไขของเวลาในการให้บริการที่แตกต่างกัน ซึ่งขึ้นอยู่การประยุกต์ใช้งาน

การพัฒนางานแบบเรียลไทม์ที่มีความซับซ้อนค่อนข้างมากจำเป็นต้องนำระบบปฏิบัติการมาใช้ในการพัฒนา เพื่อให้การพัฒนาเป็นไปได้ง่ายและมีความน่าเชื่อถือ ระบบปฏิบัติการที่นิยมนำมาใช้ในการพัฒนาระบบ embedded กับระบบเรียลไทม์ คือ ลีนุกซ์

โครงงานนี้นำเสนอการศึกษาค้นคว้าและพัฒนาระบบเรียลไทม์ที่มีลีนุกซ์เป็นระบบปฏิบัติการเพื่อใช้ในระบบ embedded

Real-Time Linux Systems

Jutaporn Charin
Chanin Sainumkheaw
Apinetr Unakul (Advisor)

Abstract

Currently the embedded systems is used widely especially in the Real-Time systems for example control system, multimedia etc.To develope the real-time task has many complex levels and necessary to consider time constraint for service that depend on the application

To develope the complex real-time task must use the operating system. The operating system will make development procedures easier and convincingly.The famous operating system for developing embedded systems and Real Time systems is Linux.

This project presents the research and development the Real Time systems that use Linux as the operating system for using in the embedded systems.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ อภิเนตร อูนากูล อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ขอขอบพระคุณห้องปฏิบัติการ ESL ที่อำนวยความสะดวกเกี่ยวกับสถานที่ทำงาน เป็นที่พักผ่อนเมื่ออ่อนล้าจากการทำงาน และยังเป็นสถานที่พบปะกับเพื่อนๆ เพื่อให้มีกำลังในการทำงานต่อไปด้วย ขอบคุณอย่างยิ่ง

ขอขอบคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ถ่ายทอดวิชาความรู้ที่เป็นประโยชน์แก่ตัวข้าพเจ้าในอนาคตอันใกล้ และสืบต่อไป ข้าพเจ้าขอให้คำมั่นว่าจะนำความรู้ที่ได้มาใช้อย่างเต็มที่ เพื่อเป็นประโยชน์ต่อสังคมและประเทศชาติต่อไป

ขอบคุณเพื่อนๆ พี่ๆ น้องๆ ร่วมสถาบัน ที่เป็นแรงผลักดันในการทำงานอย่างดียิ่ง คอยถามข่าวคราว ว่าเมื่อไหร่จะทำเสร็จ ซึ่งเป็นข้อความที่สร้างกำลังให้ข้าพเจ้าเป็นอย่างมาก และคอยถามตัวข้าพเจ้าเองตลอดเวลาว่า เมื่อไหร่หล่ะจะทำเสร็จ ขอบคุณจริงๆ

ขอบคุณตัวข้าพเจ้าทั้งสอง ที่ไม่ท้อแท้ไปเสียก่อนที่จะพบกับความสำเร็จ ขอบคุณที่ยังมีแรงและกำลังในการทำงาน ขอบคุณที่มีชีวิตมาถึงทุกวันนี้ และโอกาสดีๆที่ได้รับมา ขอขอบคุณจากใจจริง

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูข้าพเจ้ามาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

จุฑาภรณ์ ชารินทร์
ชนินทร์ สายน้ำเขียว

สารบัญ

บทคัดย่อภาษาอังกฤษ I
บทคัดย่อภาษาไทย II
กิตติกรรมประกาศ III
สารบัญ IV
สารบัญภาพ VII
สารบัญตาราง IX
บทที่1 บทนำ 1
1.1 ความสำคัญและที่มา 1
1.2 วัตถุประสงค์ของโครงงาน 2
1.3 ขอบเขตของโครงงาน 2
1.4 เป้าหมายของโครงงาน 2
1.5 วิธีการดำเนินงาน 3
บทที่ 2 ทฤษฎีและหลักการที่เกี่ยวข้อง 5
2.1 ระบบ embedded 5
2.1.1 บทบาทของระบบ embedded 5
2.1.2 ลักษณะการใช้งานระบบ embedded 6
2.1.3 คุณสมบัติของระบบ embedded 6
2.1.4 ข้อดีของระบบ embedded 6
2.2 ระบบเรียลไทม์ 6
2.2.1 รูปแบบของเรียลไทม์ 7
2.2.2 ลักษณะของเรียลไทม์ 7
2.2.3 การประยุกต์ใช้งานระบบเรียลไทม์ 8
2.3 ระบบปฏิบัติการแบบเรียลไทม์ (RTOS) 8
2.3.1 คุณลักษณะของ RTOS 8
2.3.2 คุณสมบัติของ RTOS 9
2.3.3 ข้อแตกต่างของ RTOS กับ OS ทั่วๆ ไป 9
2.4 ลีนุกซ์ 10
2.4.1 สถาปัตยกรรมของลีนุกซ์ 11
2.4.2 ลักษณะเด่นของลีนุกซ์ 14
2.4.3 ข้อดีของลีนุกซ์ 14
2.5 เรียลไทม์ลีนุกซ์ 15
2.5.1 TimeSys/RT 15

2.5.2 RTLinux 17
2.5.3 RTAI 18
2.6 uClinux 22
2.6.1 คุณลักษณะเด่นของ uClinux 23
2.6.2 ข้อจำกัดของ uClinux 23
2.6.3 ข้อดีของ uClinux 24
2.7 แนวทางการพอร์ต uClinux สำหรับฮาร์ดแวร์ใหม่ 24
2.7.1 Cross-Development Tools 24
2.7.2 การปรับปรุงแก้ไขเคอร์เนล 26
2.7.3 ดีไวซ์ไดรเวอร์ (Device Driver) 28
2.7.4 รันไทม์ไลบรารี่ (Runtime Library) 28
2.8 บอร์ด AT91EB01 32
2.8.1 ส่วนประกอบของบอร์ด AT91EB01 32
2.8.2 การ์ดขยายหน่วยความจำ AT91MEC01 33
2.9 ไมโครคอนโทรลเลอร์ AT91M40400 34
2.9.1 คุณสมบัติของ AT91M40400 34
2.9.2 สถาปัตยกรรมของ AT91M40400 35
2.9.3 ข้อดีของ AT91M40400 36
2.10 ไมโครโปรเซสเซอร์ ARM7TDMI 36
บทที่ 3 การวิเคราะห์และออกแบบ 39
3.1 ภาพรวมของโครงงาน 39
3.2 ลีนุกซ์กับระบบเรียลไทม์ 39
3.2.1 ระบบเรียลไทม์ 39
3.2.2 ลีนุกซ์กับการรองรับระบบเรียลไทม์ 41
3.2.3 ส่วนขยายเพิ่มเติมทางด้านเรียลไทม์สำหรับลีนุกซ์ 41
3.3 ลีนุกซ์กับระบบ embedded 42
3.3.1 บอร์ด AT91EB01 43
3.3.2 เครื่องมือในการพัฒนา 43
3.4 uClinux ที่นำมาใช้ในโครงงาน 44
3.4.1 uClinux กับบอร์ด At91EB01 44
3.4.2 uClinux กับ RTAI 45
3.5 Use Case 45
3.6 โครงสร้างของระบบลีนุกซ์แบบเรียลไทม์ 47
3.6.1 ส่วนที่ใช้งานบนเครื่องพีซี 47

3.6.2 ส่วนที่ทำงานบนบอร์ด AT91EB01 48
3.7 การใช้งาน uClinux กับบอร์ด AT91EB01 49
3.7.1 Generate Root File Systems Image 50
3.7.2 Generate Kernel Image 52
3.7.3 Run uClinux 52
บทที่ 4 การทดสอบและวิเคราะห์ผล 54
4.1 uClinux กับการทำงานบนบอร์ด AT91EB01 54
4.2 uClinux กับการให้บริการพื้นฐาน 56
4.2.1 การทดสอบกับไลบรารี่ 57
4.2.2 การทดสอบกับ shell 58
4.2.3 การทดสอบกับแอพพลิเคชั่นพื้นฐาน 64
4.3 uClinux กับงานแบบเรียลไทม์ 67
บทที่ 5 บทวิจารณ์และสรุปผล 69
5.1 ผลที่ได้รับจากโครงงาน 69
5.2 ปัญหาที่พบ 69
5.2.1 ปัญหากับระบบ embedded 70
5.2.2 ปัญหากับ uClinux 70
5.2.3 ปัญหากับระบบเรียลไทม์ 71
5.3 แนวทางการพัฒนาต่อ 71
5.3.1 ระบบ embedded 71
5.3.2 uClinux 71
5.3.3 ระบบเรียลไทม์ 72
5.4 สรุปผลโครงงาน 72
ภาคผนวก ก ฮาร์ดแวร์ที่ใช้ 74
ก.1 AT91EB01's Schematics 74
ก.2 AT91MEC01's Schematics 83
บรรณานุกรม 87

สารบัญภาพ

รูปที่ 2-1 สถาปัตยกรรมเคอร์เนล 12
รูปที่ 2-2 รายละเอียดเพิ่มเติมของสถาปัตยกรรมเคอร์เนล 13
รูปที่ 2-3 สถาปัตยกรรมของ Timesys/RK 16
รูปที่ 2-4 สถาปัตยกรรมของ RTLinux 17
รูปที่ 2-5 แสดงสถาปัตยกรรมของ RTAI 18
รูปที่ 2-6 โครงสร้างการทำงานของ Interrupt Dispatcher 22
รูปที่ 2-7 ส่วนของเชลล์สคริปต์ที่ใช้กำหนดค่าต่างๆสำหรับคอนฟิก GNU 24
รูปที่ 2-8 การเพิ่มเติมส่วนของไฟล์ config.bfd เพื่อทำการระบุรูปแบบไบนารี่เป้าหมาย 25
รูปที่ 2-9 แสดงส่วนที่คัดลอกมาจาก uclinux-arm.h ซึ่งระบุดีฟอลต์ของออปชั่นต่างๆ 25
รูปที่ 2-10 การเพิ่มแฟลกเมนต์ (fragment) t-arm-uclinuxl สำหรับไฟล์ makefile เพื่อสร้าง crtbrgin.o และ crtend.o 25
รูปที่ 2-11 แสดงรีจิสเตอร์ ptrace เพื่อใช้สนับสนุนการเก็บค่าและคืนค่าของคอนเท็กซ์ 27
รูปที่ 2-12 แสดง Program entry code ที่ทำการแฮนเดิล environ ซึ่งส่งมาจาก sys_execve kernel call30
รูปที่ 2-13 ไดอะแกรมของ วิชวลโปรแกรมและแฟลซโปรแกรม 31
รูปที่ 2-14 โครงสร้างของสถาปัตยกรรม AT91EB01 33
รูปที่ 2-15 ส่วนประกอบของ AT91MEC01 34
รูปที่ 2-16 สถาปัตยกรรมของ AT91M40400 35
รูปที่ 2-17 โครงสร้างของ ARM7TDMI 37
รูปที่ 2-18 โครงสร้างของคอร์ภายใน ARM7TDM 38
รูปที่ 3-1 ระบบลีนุกซ์แบบเรียลไทม์ 39
รูปที่ 3.2 รูปแบบของระบบเรียลไทม์กับการพัฒนา 40
รูปที่ 3-3 แสดง Use Case Diagram ของระบบลีนุกซ์แบบเรียลไทม์ 46
รูปที่ 3-4 โครงสร้างของระบบลีนุกซ์แบบเรียลไทม์บนบอร์ด AT91EB01 47
รูปที่ 3-5 การใช้งาน uClinuxกับบอร์ด AT91EB01 49
รูปที่ 3-6 Generate Root FS Image 51
รูปที่ 3-7 Generate Kernel Image 52
รูปที่ 3-8 Run uClinux 53
รูปที่ 4-1 เอาท์พุทแสดงการทำงานของ uClinux 56
รูปที่ 4-2 การรอรับคำสั่งและการใส่คำสั่งให้กับ shell 60
รูปที่ 4-3 คำสั่งภายในของ shell 61
รูปที่ 4-4 ผลการสั่งงานบางคำสั่งของ shell 62
รูปที่ 4-5โปรแกรม atmel-init 65

รูปที่ 4-6 โปรแกรม ping....66
รูปที่ 4-7 โปรแกรม shutdown....67
รูปที่ ก-1 ส่วนประกอบต่างๆภายในบอร์ด T91M40400....75
รูปที่ ก-2 PCB เลเอาต์....76
รูปที่ ก-3 แสดงการอินเตอร์เฟสกับบัสภายนอก....77
รูปที่ ก-4 JTAG อินเตอร์เฟส, รีเซ็ต, อินเตอร์รัพท์ และ LEDs....78
รูปที่ ก-5 แสดงส่วนขยาย I/O....79
รูปที่ ก-6 แสดงพาวเวอร์, คริสตัล ออสซิเลเตอร์ และ คล็อกดิสทริบิวชั่น....80
รูปที่ ก-7 แสดง AT91M40400 ในแพ็กเกจ TQFP....81
รูปที่ ก-8 แสดงซีเรียลอินเตอร์เฟส....82
รูปที่ ก-9 แสดง PCB เลเอาต์....84
รูปที่ ก-10 แสดง EBI คอนเน็กเตอร์ และแฟลช....85
รูปที่ ก-11 ส่วนของ SRAM....86

สารบัญตาราง

ตารางที่ 3-1 การเปรียบเทียบลักษณะเด่นของ Timesys/RK, RTAI และ RTLInux.....42
ตารางที่ 4-1 คำสั่งของ shell ที่ทำการทดสอบ.....62

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันได้พัฒนาอุปกรณ์ที่มีระบบคอมพิวเตอร์ควบคุมอยู่ภายใน ซึ่งเรียกว่าระบบ embedded อย่างแพร่หลาย ซึ่งระบบ embedded ได้รับการพัฒนาหลังจากที่ได้คิดค้นซีพียูขนาดจิ๋ว ที่เรียกว่า โมโครโปรเซสเซอร์ มาแล้ว โดยเป็นอุปกรณ์หลักที่อยู่ภายในเครื่องมือและอุปกรณ์อิเล็กทรอนิกส์เป็นจำนวนมาก ตั้งแต่เครื่องใช้ภายในบ้าน เช่น เครื่องซักผ้า เครื่องปรับอากาศ เครื่องครัว เครื่องใช้จำพวกเครื่องเสียง หรือแม้แต่เครื่องเล่นวีดีโอเทป เครื่องเล่นซีดี เป็นต้น ไมโครโปรเซสเซอร์ยังแทรกเข้าไปอยู่ในเครื่องมือที่ใช้ในธุรกิจต่าง ๆ เช่น เครื่องเก็บเงินสด เครื่องถ่ายเอกสาร โทรสาร ใช้ในโรงงานอุตสาหกรรม ได้แก่ เครื่องควบคุมกรรมวิธีการผลิต เครื่องมือวัด เครื่องจักรที่ทำงานอัตโนมัติต่าง ๆ

ในยุคแรก ในการพัฒนาเครื่องมือเครื่องใช้ที่มีระบบคอมพิวเตอร์ควบคุมอยู่ภายในหรือการพัฒนาระบบ embedded ได้รับการออกแบบในขอบเขตจำกัด กล่าวคือมีการทำงานเฉพาะภายในเครื่องนั้น เช่น เครื่องปรับอากาศก็ปรับควบคุมการทำงานภายในเครื่อง แต่ในระยะไม่กี่ปีมานี้ พัฒนาการทางด้านอุปกรณ์ได้ก้าวหน้ามากขึ้น มีการทำงานเป็นระบบ มีการรับส่งข้อมูลระหว่างตัวอุปกรณ์ต่างๆ อย่างอัตโนมัติ การทำงานซับซ้อนขึ้น เช่น ในระบบของโรงงานอุตสาหกรรมการผลิตต่างๆ มีระบบซอฟต์แวร์ที่ใช้ในการควบคุมที่สลับซับซ้อนขึ้น และที่สำคัญคือ เป็นการรวมผลิตภัณฑ์จากหลายแหล่งเข้าเป็นระบบ ทำให้ความซ่อนเร้นภายในของระบบดูจะเป็นเรื่องยุ่งยากมากยิ่งขึ้น และสามารถนำระบบ embedded มาใช้งานได้หลากหลายมากขึ้น โดยเฉพาะในระบบควบคุมเช่น หุ่นยนต์, รถยนต์ และดาวเทียมเป็นต้น

เหตุผลที่เรานำระบบ embedded มาใช้งานอย่างกว้างขวางเนื่องจากว่า ใช้ต้นทุนในการพัฒนาต่ำ สามารถพัฒนาได้อย่างรวดเร็วและหลากหลายโดยจะเน้นในส่วนของฮาร์ดแวร์มากกว่าซอฟต์แวร์เพราะมีมูลค่ามากกว่า

ส่วนใหญ่ได้นำแนวคิดและเทคนิคของระบบเรียลไทม์มาใช้ในระบบ embedded ซึ่งในระบบนี้จะมีลักษณะเด่นสองประการ โดยประการแรกคือเป็นระบบที่มีขนาดเล็กและไม่ซับซ้อน ทำงานในลักษณะเฉพาะด้านไม่มีการเปลี่ยนแปลงมากนักและยังรวมการทำงานไว้ในที่เดียวกัน ส่วนอีกประการหนึ่งคือในส่วนของฮาร์ดแวร์ซึ่งจะมีมูลค่าสูงกว่าในส่วนของซอร์ฟแวร์โดยมีการจำกัดความสามารถทางด้านฮาร์ดแวร์บางอย่างคือต้องมีขนาดเล็กและสิ้นเปลืองพลังงานน้อย

ด้วยข้อจำกัดทางด้านฮาร์ดแวร์ทำให้การพัฒนาแอพพลิเคชั่นเพื่อใช้ในระบบ embedded ต้องทำการจัดสรรทรัพยากรของระบบด้วยตัวเอง ทำให้ลักษณะแอพพลิเคชั่นส่วนใหญ่จึงเป็นของระบบเรียลไทม์ ดังนั้นในการพัฒนาระบบเรียลไทม์เพื่อใช้ในระบบ embedded จำเป็นต้องพัฒนาระบบปฏิบัติการแบบเรียลไทม์ขึ้นมาเอง เพื่อให้สามารถพัฒนาระบบเรียลไทม์ที่มีความซับซ้อนให้ง่ายขึ้น

ระบบเรียลไทม์ที่ทำการพัฒนาขึ้นมานั้น จะมีขอบเขตและขนาดการใช้งานที่แตกต่างกัน ตั้งแต่นาฬิกาข้อมือ เตาอบไมโครเวฟ จนถึง ระบบอัตโนมัติในอุตสาหกรรม โรงงานไฟฟ้าพลังนิวเคลียร์ รวมถึงระบบทีวีและวีดีโอ, เครื่องเล่นเพลง , ระบบควบคุมเครื่องบิน หรือแม้กระทั่ง ระบบควบคุมจรวด ซึ่งต้องการให้สามารถทำงานตามเวลาที่เรากำหนดได้

ระบบปฏิบัติการเป็นส่วนประกอบที่สำคัญในระบบทั่วๆไป ซึ่งในระบบเรียลไทม์จำเป็นต้องมีระบบปฏิบัติการแบบเรียลไทม์เพื่อสามารถรันงานแบบเรียลไทม์ได้ การที่มีระบบปฏิบัติการแบบเรียลไทม์ทำให้เราคาดเดาเวลาที่ใช้ในการทำงานให้เสร็จได้ และยังมีความน่าเชื่อถือ รวมทั้งสามารถจัดลำดับของงานเพื่อทำการรันได้ แต่เนื่องจาระบบปฏิบัติการแบบเรียลไทม์ในปัจจุบันส่วนใหญ่เป็นเชิงการค้า ซึ่งราคาค่อนข้างแพง และต้องเสียค่าใช้จ่ายในการบำรุงรักษาเป็นอย่างมาก ทำให้ไม่เหมาะที่จะนำมาใช้ในระบบงานขนาดเล็ก จึงได้นำระบบปฏิบัติการที่มีอยู่เดิมและมีราคาถูกมาพัฒนาและเพิ่มความสามารถในด้านเรียลไทม์ เพื่อให้สามารถทำงานแบบเรียลไทม์ได้

โดยระบบปฏิบัติการที่นิยมนำมาใช้คือ ลีนุกซ์ ซึ่งทำการเพิ่มความสามารถในด้านเรียลไทม์ให้กับลีนุกซ์ ด้วยเหตุผลคือเปิดเผยโค้ดสามารถทำการปรับปรุงและแก้ไขได้ง่ายและไม่เสียค่าใช้จ่ายในการนำมาใช้งาน ทำให้ผู้พัฒนาตรวจสอบความถูกต้องของระบบได้อย่างรวดเร็ว รวมทั้งเหมาะนำมาใช้ในระบบ embedded

ดังนั้นในการพัฒนาระบบเรียลไทม์เพื่อนำใช้งานในระบบ embedded เราจำเป็นที่ต้องนำระบบปฏิบัติการมาใช้ในการพัฒนาเพื่อให้การพัฒนาเป็นไปได้ง่ายและมีความรวดเร็ว ซึ่งลีนุกซ์เป็นระบบปฏิบัติการที่นิยมนำมาใช้งานในระบบ embedded โดยเพิ่มความสามารถแบบเรียลไทม์ให้กับลีนุกซ์

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อให้เข้าใจถึงระบบ embedded และระบบเรียลไทม์
2. ศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบเรียลไทม์
3. ศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบ embedded
4. พัฒนาให้ลีนุกซ์สามารถทำงานบนระบบ embedded ที่เลือกใช้ได้
5. พัฒนาให้ลีนุกซ์ที่ทำงานบนระบบ embedded สามารถทำงานแบบเรียลไทม์ได้

1.3 ขอบเขตของโครงงาน

โครงงานนี้ต้องการนำเสนอการพัฒนาระบบเรียลไทม์เพื่อสามารถใช้งานในระบบ embedded ได้ โดยได้นำระบบปฏิบัติการช่วยในการพัฒนาระบบเรียลไทม์ให้ง่ายและมีความรวดเร็วโดยใช้ลีนุกซ์เป็นระบบปฏิบัติการซึ่งต้องสามารถทำงานบนระบบ embedded ได้รวมทั้งเพิ่มความสามารถทางด้านเรียลไทม์ให้กับลีนุกซ์ เพื่อให้ทำงานแบบเรียลไทม์ได้อย่างถูกต้อง

1.4 เป้าหมายของโครงงาน

1. บทสรุปจากการศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบเรียลไทม์ เพื่อนำมาใช้ในการตัดสินใจเลือกส่วนขยายเพิ่มทางด้านเรียลไทม์ของลีนุกซ์ที่จะนำมาใช้ในการพัฒนา

2. บทสรุปจากการศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบ embedded เพื่อนำมาใช้ในการตัดสินใจเลือกบอร์ดแบบ embedded ที่ตรงตามความต้องการจะนำมาใช้ในการพัฒนา
3. ลีนุกซ์ที่ได้พัฒนาให้สามารถทำงานบนบอร์ดแบบ embedded ที่เลือก มีคุณสมบัติดังนี้
   - สามารถให้บริการพื้นฐานเดิมที่มีอยู่บนระบบปฏิบัติการลีนุกซ์ที่ทำงานบนเครื่องคอมพิวเตอร์ทั่วไป
   - ใช้หน่วยความจำในการทำงานที่เหมาะสม
   - การพัฒนาแอพพลิเคชั่นและเคอร์เนลทำบนเครื่องคอมพิวเตอร์ทั่วไป
   - อินพุทและเอาท์พุทผ่านทางเทอร์มินัล
4. ลีนุกซ์ ที่ทำงานบนระบบ embedded และได้พัฒนาให้สามารถทำงานแบบเรียลไทม์ มีคุณสมบัติดังนี้
   - งานแบบเรียลไทม์สามารถทำงานได้อย่างถูกต้อง

1.5 วิธีการดำเนินงาน

การดำเนินงานของโครงการนี้แบ่งออกเป็นขั้นตอนต่างๆดังนี้

1. ศึกษาค้นคว้าและทำความเข้าใจเกี่ยวกับระบบ embedded และระบบเรียลไทม์

   เป็นการศึกษาว่าระบบ embedded คืออะไร มีความเหมาะสมที่จะนำมาใช้พัฒนางานแบบเรียลไทม์ได้อย่างไรบ้าง รวมถึงศึกษาว่า ถ้าต้องการพัฒนาระบบเรียลไทม์เพื่อให้สามารถใช้งานบนระบบ embedded ได้จำเป็นต้องมีวิธีการและทรัพยากรอะไรบ้างเพื่อให้สามารถพัฒนาระบบเป็นไปได้ง่ายและรวดเร็ว
2. ศึกษาค้นคว้าและทำความเข้าใจเกี่ยวกับลีนุกซ์และระบบเรียลไทม์

   ทำการศึกษาและค้นคว้าเกี่ยวกับการพัฒนาลีนุกซ์ให้เป็นแบบเรียลไทม์ที่มีอยู่ในปัจจุบัน และเหตุผลของการนำลีนุกซ์มาใช้ในการพัฒนางานแบบเรียลไทม์ว่าลีนุกซ์มีข้อดีและข้อจำกัดในการพัฒนาอะไรบ้าง รวมถึงศึกษาถึงแอพพลิเคชั่นที่เป็นแบบเรียลไทม์เพื่อใช้เป็นแนวทางในการพัฒนาต่อไป
3. วิเคราะห์หาบทสรุปของระบบลีนุกซ์แบบเรียลไทม์

   เป็นขั้นตอนในการเลือกระบบลีนุกซ์แบบเรียลไทม์ที่มีอยู่ในปัจจุบันเพื่อนำมาใช้ในการพัฒนา โดยเลือกระบบลีนุกซ์แบบเรียลไทม์ที่มีความเหมาะสมที่นำมาพัฒนาในโครงงานซึ่งต้องคำนึงถึงความต้องการของระบบ ความเป็นไปได้ของการพัฒนาและ สามารถนำมาใช้งานได้อย่างมีประสิทธิภาพ ซึ่งขั้นตอนนี้อาจใช้เวลาในการหาข้อสรุป เพราะปัจจุบันมีการพัฒนาระบบลีนุกซ์แบบเรียลไทม์อยู่หลายผลิตภัณฑ์ จำเป็นที่ต้องวิเคราะห์ผลิตภัณฑ์แต่ละตัวเพื่อเลือกผลิตภัณฑ์ที่เหมาะสมที่สุดในการพัฒนาโครงงานนี้
4. วิเคราะห์หาบทสรุปของระบบ embedded ที่นำมาใช้ในงานแบบเรียลไทม์

   เป็นขั้นตอนในการเลือกระบบ embedded มาใช้ในการพัฒนา โดยคำนึงถึงความสามารถของระบบในการพัฒนางานแบบเรียลไทม์ และต้องมีประสิทธิภาพสูงด้วยข้อจำกัดทรัพยากรของระบบ รวมทั้งสามารถพัฒนาระบบเรียลไทม์ได้อย่างง่ายและรวดเร็ว

5. ออกแบบระบบเรียลไทม์ที่มีลีนุกซ์เป็นระบบปฏิบัติการเพื่อนำมาใช้งานในระบบ embedded

เมื่อเราได้ส่วนของระบบ embedded และระบบลีนุกซ์แบบเรียลไทม์ที่จะนำมาใช้ในการพัฒนาแล้ว ก็ทำการออกแบบระบบตามความต้องการคือ พัฒนาลีนุกซ์แบบเรียลไทม์เพื่อให้สามารถทำงานบนระบบ embedded ได้

6. จัดเตรียมทรัพยากรที่ต้องการในการพัฒนาระบบ

เป็นการเตรียมเครื่องมือที่ต้องการในการพัฒนา ทั้งส่วนของ อิดิเตอร์, อินเตอร์เฟส , ดีบักเกอร์ รวมทั้งลีนุกซ์เพื่อใช้เป็นระบบปฏิบัติการพื้นฐานในการพัฒนาบนพีซี และซ็อสโค้ดที่จำเป็นในการพัฒนา

7. ทดลองใช้งานลีนุกซ์ที่ทำการพัฒนาไว้แล้วซึ่งเป็นลีนุกซ์ที่ทำงานบนระบบ embedded

ในการเลือกใช้ลีนุกซ์เพื่อให้สามารถทำงานบนระบบ embedded จำเป็นที่ต้องมีแพทเทิร์นของระบบ embedded ที่มีลีนุกซ์ทำงานอยู่แล้ว เพื่อเป็นรูปแบบที่นำมาใช้ในการพัฒนากับระบบ embedded ที่ได้เลือกไว้ ทำให้สามารถพัฒนาระบบได้ง่ายขึ้น แต่จะมีความยุ่งยากในส่วนที่ต้องศึกษาระบบ embedded เพิ่มขึ้นอีกเพื่อให้เข้าในถึงแนวทางและวิธีการพัฒนาลีนุกซ์เพื่อให้สามารถใช้งานบนระบบ embedded

8. ทำการพอร์ตลีนุกซ์ให้สามารถใช้งานบนระบบ embedded ได้

เป็นขั้นตอนพอร์ตลีนุกซ์ที่เลือกไว้ให้สามารถทำงานบนระบบ embedded ที่เลือกได้ ตามรูปแบบและแนวทางในการพัฒนาที่ได้ศึกษาไว้แล้ว โดยทำการพัฒนาบนเครื่องพีซี แล้วค่อยพอร์ตลงบนระบบ embedded อีกที

9. ทดสอบการให้บริการพื้นฐานของลีนุกซ์ที่ทำงานบนระบบ embedded

เมื่อทำการพอร์ตลีนุกซ์ลงบนระบบ embedded เรียบร้อยแล้ว จำเป็นที่จะต้องทดสอบความสามารถของลีนุกซ์ซึ่งต้องสามารถให้บริการพื้นฐานของลีนุกซ์ได้ เช่นมาตรฐาน API ,เชลล์ สคริปต์, คำสั่งพื้นฐานต่าง ๆ เป็นต้น ซึ่งขั้นตอนนี้เป็นการวัดความสามารถของลีนุกซ์ที่ได้พัฒนาบนระบบ embedded ว่า มีประสิทธิภาพมากน้อยแค่ไหน เหมาะที่จะนำมาใช้งานต่อไปหรือไม่

10. พัฒนาแอพพลิเคชั่นสำหรับลีนุกซ์ที่ทำงานบนระบบ embedded

เป็นขั้นตอนในการสร้างแอพพลิเคชั่นขึ้นมาเพื่อทดสอบการใช้งานลีนุกซ์ที่ได้พัฒนาขึ้นมา และทดสอบว่าสามารถทำงานได้ตามที่ต้องการหรือไม่ และแอพพลิเคชั่นควรเป็นมีคุณลักษณะและคุณสมบัติอย่างไรที่สามารถทำงานได้อย่างมีประสิทธิภาพบนลีนุกซ์ที่ได้พัฒนาขึ้น รวมทั้งวิธีในการพัฒนาแอพพลิเคชั่นแบบเรียลไทม์เพื่อให้สามารถทำงานบนระบบ embedded ได้

11. สรุปและประเมินความสามารถของลีนุกซ์ที่ทำงานบนระบบ embedded

เป็นการสรุปความสามารถของลีนุกซ์ที่พัฒนาขึ้นมาให้สามารถทำงานบนระบบ embedded ได้ และทำการเปรียบเทียบกับลีนุกซ์ที่ทำงานบนพีซี ว่ามีความเหมือนและแตกต่างในด้านใดบ้าง รวมทั้งคุณลักษณะของลีนุกซ์ว่ามีประสิทธิภาพมากน้อยแค่ไหนเมื่อนำไปใช้งานในระบบ embedded

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ระบบ embedded

ระบบ embedded คืออุปกรณ์อะไรก็ตามที่มีไมโครชิปฝังอยู่ ซึ่งเป็นระบบอิเล็กทรอนิกส์ที่ใช้สำหรับงานควบคุมรวมถึงการแสดงผลการทำงานต่าง ๆ โดยที่ระบบ embedded เหล่านี้ถูกใช้เป็นส่วนหนึ่งของระบบและอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรต่าง ๆ การที่ใช้คำว่า "ระบบแบบฝังตัว" เนื่องจากระบบเหล่านี้เป็นส่วนหนึ่งของระบบใหญ่ ในหลายกรณีที่ผู้ใช้ทั่วไปอาจไม่ทราบว่าอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรรวมถึงระบบใดที่ใช้งานเป็นประจำเหล่านั้นเป็นระบบแบบฝังตัว ในบางครั้งแม้แต่ผู้ที่มีความรู้ทางด้านเทคนิคก็ไม่สามารถระบุได้แน่ชัดว่าใดมีระบบแบบฝังตัวอยู่ จนกว่าจะมีการทำงานและตรวจสอบกับระบบและอุปกรณ์ควบคุมนั้นระยะหนึ่งเลยที่เดียว

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

นอกจากนี้แบบและรุ่นของระบบ embedded ก็มีมากมายมีทั้งระบบที่เป็นแบบง่ายๆการทำงานไม่ซับซ้อน ตลอดจนแบบที่ซับซ้อน ซึ่งขึ้นอยู่กับประเภทและจำนวนไมโครโปรเซสเซอร์ รวมถึงงานโปรแกรมควบคุมในระบบ

ไมโครโปรเซสเซอร์หรือชิปที่ได้รับการออกแบบจะมีทั้งแบบที่ออกแบบเพื่อให้สามารถเพิ่มโปรแกรมเข้าไปในระบบ embedded ได้ แต่ที่พบเห็นโดยทั่วไปเป็นแบบที่ควบคุมด้วยคำสั่งโปรแกรมนี้จะไม่สามารถแก้ไขได้อีก ถ้าต้องการแก้ไขก็จะต้องเอาชิปดังกล่าวออกและเปลี่ยนชิปตัวใหม่ที่บรรจุคำสั่งโปรแกรมที่แก้ไขแล้ว

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

2.1.1 บทบาทของระบบ embedded

- ทำหน้าที่เป็นมอนิเตอร์ให้กับอุปกรณ์ต่าง ๆ โดยจะอ่านอินพุต จากเซนเซอร์และโปรเซสอินพุตที่เข้ามา จากนั้นทำการแสดงผลลัพท์ที่ได้ออกทางอุปกรณ์แสดงผล
- ทำหน้าที่ควบคุมอุปกรณ์ต่างๆโดยทำการสร้างและส่งคำสั่งควบคุมไปยังอุปกรณ์ควบคุม
- ทำการเปลี่ยนแปลงแก้ไขข้อมูลทั้งในส่วนของการเพิ่มและลดขนาดของข้อมูล

2.1.2 ลักษณะการใช้งานระบบ embedded

การนำระบบ embedded มาใช้งานต้องมีลักษณะดังนี้

- ASIC ต้องเร็วมีการอินติเกรตที่สูงแต่จะยุ่งยากในการออกแบบที่ซับซ้อน
- ไมโครโปรเซสเซอร์ ซึ่งสามารถควบคุมฟังก์ชั่นต่างๆในระบบได้ง่าย มีความยืดหยุ่นสูงและสามารถเพิ่มเติมแก้ไขได้ง่าย แต่อาจเกิดปัญหาหากโมโครโปรเซสเซอร์ช้า ซึ่งแก้ไขโดยใช้มัลติโปรเซสเซอร์
- ออกแบบทั้งด้านซอฟต์แวร์และฮาร์ดแวร์โดนเน้นไปที่ฮาร์ดแวร์เพราะมีมูลค่าสูงกว่าซอฟต์แวร์
- เป็นระบบชิปเดียว

2.1.3 คุณสมบัติของระบบ embedded

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

2.1.4 ข้อดีของระบบ embedded

- เหมาะสมในการเป็นเครื่องมือในการพัฒนาระบบ
- สนับสนุน มาตรฐาน API สามารถพัฒนาแอพพลิเคชั่นได้ง่ายและรวดเร็ว
- สนับสนุนอุปกรณ์ภายนอกเช่นโมเด็ม, RF
- สามารถใช้งานระบบเน็ตเวิร์กได้
- ใช้ต้นทุนในการพัฒนาน้อย
- สามารถเพิ่มเติมความสามารถของระบบได้อย่างต่อเนื่อง

2.2 ระบบเรียลไทม์

ระบบเรียลไทม์ คือระบบที่ทำงานตามเงื่อนไขของเวลา โดยเวลาส่วนใหญ่วัดจากประสิทธิภาพการตอบรับของระบบ ซึ่งระบบต้องตอบรับในเวลาที่ถูกต้องตามเหตุการณ์ที่เข้ามาจากภายนอก การออกแบบระบบเรียลไทม์ ต้องคาดเดา (Predictable) เวลาที่ต้องการของระบบได้และต้องเชื่อถือได้ (Reliability) ว่าประสิทธิภาพของระบบที่ได้ต้องถูกต้อง (Correct) และ ตามเวลาที่เรากำหนดไว้ (Timely) สิ่งที่เราต้องการในการตรวจสอบความถูกต้องตามเงื่อนไขทางเวลาคือ ต้องพบ เดดไลน์ (Deadline)

2.2.1 รูปแบบของเรียลไทม์

2.2.1.1 ฮาร์ดเรียลไทม์ (Hard Real-Time)

หมายความว่า ระบบต้องพบเดดไลน์ หากข้อมูลเกิดความผิดพลาดขึ้นจะทำให้ระบบเกิดความเสียหาย ซึ่งระบบต้องสามารถคาดเดาได้ และมีการตอบรับที่เร็ว (Low Latency) เราไม่สามารถวัดประสิทธิภาพด้วยกรณีเฉลี่ย (average case performance) แทนการวัดประสิทธิภาพด้วยกรณีที่แย่ที่สุดได้ (worst-case performance)

ระบบเรียลไทม์ที่เป็นแบบฮาร์ดเรียลไทม์ ต้องสามารถรับประกันได้ว่าต้องพบเดดไลน์ และมีการจัดลำดับของงานทั้งหมดเป็นแบบสแตติกคือต้องรู้ลำดับความสำคัญของแต่ละงาน ตัวอย่างของฮาร์ดเรียลไทม์ เช่น การยิงจรวด หากยิงไม่ตรงตามเวลาที่กำหนด ทำให้จรวดยิงไม่โดนเป้าหมาย หรือการควบคุมรถไฟ ซึ่งต้องทำการควบคุมความเร็วและรักษาระดับการลดความเร็วเพื่อคำนวณเวลาในการหยุด เพราะถ้าหากเงื่อนไขของเวลาผิดพลาดทำให้รถไฟเกิดความเสียหายอย่างมากอาจถึงขั้นตกรางหรือประสานงากันก็ได้

2.2.1.2 ซอฟต์เรียลไทม์ (Soft Real-Time)

หมายความว่าระบบปกติแล้วจะมีเดดไลน์ ซึ่งจะยังคงทำตามลำดับของงานจนเสร็จ แม้ว่าจะผ่านเดดไลน์มาแล้วก็ตาม สามารถวัดประสิทธิภาพด้วยกรณีเฉลี่ยได้ตัวซอฟต์เรียลไทม์ ได้แก่ระบบ มัลติมีเดีย เช่นการเล่นเพลง เอ็มพี3 ถึงแม้จะเกิดการกระตุกเราก็สามารถยอมรับได้

2.2.1.3 เฟิร์มเรียลไทม์ (Firm Real-Time)

จะรวมทั้งซอฟต์เรียลไทม์ และฮาร์ดเรียลไทม์ ตัวอย่างของ เฟิร์มเรียลไทม์ จากตัวอย่างโรบอทจะเป็นฮาร์ดเรียลไทม์ ถ้าหากโรบอทมาถึงช้าเนื่องจากโอเปอเรชั่นทำงานไม่ถูกต้องและจะเป็นซอฟต์เรียลไทม์ถ้าหากโรบอทมาช้าเนื่องจากการที่ throughput หายไป

ระบบคอมพิวเตอร์โดยทั่วไป ก็สามารถเป็นระบบเรียลไทม์ได้ โดยอาจจะเรียกว่าระบบ ซอฟต์เรียลไทม์ หรือระบบที่ถ้าเกิดความล้มเหลวในการทำงานขึ้น ความเสียหายจะไม่รุนแรงมากนัก ส่วนระบบที่ถ้าเกิดความล้มเหลวแล้ว ความเสียหายจัดอยู่ในขั้นรุนแรง ก็จะเรียกว่า ระบบฮาร์ดเรียลไทม์ ส่วนระบบที่เป็น ฮาร์ดเดดไลน์ แต่ว่า บางครั้งก็สามารถละเลยเดดไลน์นั้นได้ ก็จะเรียกว่า เฟิร์มเรียลไทม์

2.2.2 ลักษณะของเรียลไทม์

1. Predictably คือสามารถคาดเดาการตอบรับได้อย่างรวดเร็ว
2. Schedulability หมายความว่างานในระบบต้องพบ เดดไลน์ คือสามารถทำงานตามเวลาที่กำหนดได้
3. Stability คือความทนทานต่อความเสียหายที่จะเกิดขึ้น นั่นคือระบบสามารถทำงานจนเสร็จ แม้ว่าจะเกิดเหตุการณ์ที่มากระทบก็ตาม

2.2.3 การประยุกต์ใช้งานระบบเรียลไทม์

ปัจจุบันมีการพัฒนาระบบเรียลไทม์เพื่อใช้ในงานต่างๆ มากมาย ได้แก่ อุปกรณ์สื่อสาร เช่น อินเตอร์เน็ต, เราเตอร์ ดาวเทียม ฯลฯ อุปกรณ์อินเตอร์เน็ต เช่นดิจิตอลคาเมล่า, ออดิโอและวีดิโอ เป็นต้น เครื่องใช้สำหรับโทรศัพท์ อาทิเช่น เสียงผ่านเน็ต (Voice Over IP), โทรสารผ่านเน็ต (Fax Over IP) และระบบตอบรับโทรศัพท์อัตโนมัติ ในเครื่องมือเครื่องใช้ทางการแพทย์เช่น เครื่อง เอ็กซเรย์, เตียงผู้ป่วย, เครื่องมือวัดต่างๆ เป็นต้น อีกทั้งยังนำมาใช้กับ เครื่องพิมพ์, เครื่องมือวัดทางอิเล็กทรอนิกส์, เอทีเอ็ม โดยเฉพาะงานที่ใช้เวลามาเกี่ยวข้องทั้งในระบบควบคุมเช่น ขีปนาวุธ, ดาวเทียม, หุ่นยนต์ ฯล ตัวอย่างของระบบเรียลไทม์ อย่างเช่น การควบคุมโรบอทให้หยิบสิ่งของจากสายพานที่มีการเคลื่อนที่โดยโรบอทมีหน้าต่างเล็กๆ เพื่อทำการหยิบวัตถุ ถ้าหากว่าโรบอททำงานช้าจะหยิบวัตถุไม่ได้ทำให้การทำงานเกิดความผิดพลาด ถึงแม้ว่าโรบอทจะไปยังตำแหน่งที่ถูกต้องก็ตาม ในทางตรงกันข้ามถ้าหากโรบอทมาถึงก่อนที่วัตถุจะมาถึงโรบอทจะทำการบล็อกมัน

2.3 ระบบปฏิบัติการแบบเรียลไทม์(RTOS)

ในการพัฒนาระบบเรียลไทม์จำเป็นต้องใช้ RTOS เพื่อให้การพัฒนาระบบเรียลไทม์เป็นไปได้ง่าย และสามารถทำงานตามคุณลักษณะของระบบเรียลไทม์ได้

2.3.1 คุณลักษณะของ RTOS

1. Determinism

OS ที่มี Deterministic คือว่า จะทำโอเปอเรชั่นตามเวลาที่กำหนด โดยจะขึ้นอยู่กับ ความเร็วในการตอบรับการอินเตอร์รัพท์ และความสามารถของระบบที่ทำตามการร้องขอทั้งหมดที่เข้ามาภายในเวลาที่ต้องการ โดยพิจารณาที่ค่าหน่วงเวลาสูงสุดจากอุปกรณ์ที่มีลำดับความสำคัญสูงกว่าถูกอินเตอร์รัพท์ ถ้าหากเราไม่ใช้ระบบเรียลไทม์ ค่าหน่วงเวลาจะเกิดที่ประมาณ 10-100 มิลลิวินาที แต่ถ้าเป็นระบบเรียลไทม์หน่วงเวลาที่เกิดประมาณ 2-3 ไมโครวินาที

2. Responsiveness

ลักษณะ Determinism จะเกี่ยวกับว่า OS เกิดการหน่วงเวลานานขนาดไหนก่อนที่จะถูก ตอบรับ (Acknowledge) โดยอินเตอร์รัพท์ ส่วน Responsiveness จะเกี่ยวข้องว่า OS ใช้เวลานานขนาดไหนในการให้บริการกับอินเตอร์รัพท์ ซึ่งในระบบเรียลไทม์ต้องพบเวลาที่ต้องการ

3. User Control

ในระบบที่ไม่ใช่ระบบเรียลไทม์ ผู้ใช้ไม่ต้องควบคุมลำดับหน้าที่ของ OS และสามารถเพิ่มเติมอะไรเข้าไปก็ได้ เช่น การแบ่งกลุ่มผู้ใช้ตามลำดับความสำคัญ ส่วนในระบบเรียลไทม์ ผู้ใช้สามารถแบ่งแยกได้อย่างชัดเจนระหว่างงานแบบฮาร์ดเรียลไทม์และ ซอฟต์เรียลไทม์รวมทั้งต้องควบคุมความสัมพันธ์ของลำดับความสำคัญภายในแต่ละคลาสด้วย

4. Reliability

เป็นลักษณะที่ใช้มากในระบบเรียลไทม์ ในระบบที่ไม่ใช่เรียลไทม์หากเกิดความผิดพลาดในช่วงเวลาใดเวลาหนึ่ง จะทำการแก้ปัญหาโดยการรีบูตระบบใหม่ แต่ในระบบเรียลไทม์ข้อมูลที่ผิดพลาดอาจเป็นผลเสียต่อระบบอย่างมาก ซึ่งระบบเรียลไทม์ ต้องทำให้มั่นใจได้ว่าสามารถทำงานได้อย่างถูกต้อง และพบเดดไลน์

5. Fail-soft operation

เรียกอีกอย่างหนึ่งว่า Stability เป็นการปกป้องข้อมูลหากระบบเกิดความเสียหายขึ้นมา หากเป็นระบบที่ไม่ใช่ระบบเรียลไทม์ เช่น ในยูนิกซ์เมื่อพบข้อมูลที่เสียหายในเคอร์เนลที่เป็นผลทำให้เมสเซจผิดพลาด จะทำการดัมพ์ (dump) ข้อมูลของหน่วยความจำ ไปยังดิสก์ ในระบบเรียลไทม์ จะพยายามตรวจสอบและแก้ไข แล้วทำการรันงานต่อไป

2.3.2 คุณสมบัติของ RTOS

1. ขนาดเล็ก
2. คอนเท็กซ์สวิตช์เร็ว
3. สามารถตอบรับการอินเตอร์รัพท์จากภายนอกได้อย่างรวดเร็ว
4. ทำงานแบบหลายงานพร้อมกัน (multitasking) ด้วย IPC (Inter Process Communication) เช่น semaphore, signal และ event ฯล
5. ใช้ Preemptive scheduling เป็นพื้นฐานในการจัดลำดับความสำคัญของงาน
6. มีเวลาว่างน้อยในช่วงการอินเตอร์รัพท์หรือช่วงดิสเอเบิล
7. มีการตั้งเวลาเข้าและเวลาออก (Time out)
8. สามารถป้องกัน priority inversion โดยใช้ Real-Time Scheduling Algorithm ได้แก่ Rate-Monotonic Algorithm (RMA), Deadline-Monotonic Algorithm (DMA) เป็นต้น

หัวใจของระบบแบบเรียลไทม์คือต้องมีช่วงของแต่ละงานที่สั้นสามารถพบเดดไลน์ของงานได้

2.3.3 ข้อแตกต่างของ RTOS กับ OS ทั่วๆ ไป

โดยทั่วไประบบปฏิบัติการต้องทำงานโดยทำการตรวจสอบความถูกต้องทางลอจิกอยู่แล้ว แต่ RTOS เป็นระบบปฏิบัติการที่เพิ่มการตรวจสอบความถูกต้องโดยใช้เงื่อนไขของเวลามาเกี่ยวข้องด้วย

RTOS ต้องมีหน้าที่เมือนกับ OS ทั่วๆไป ได้แก่

1. จัดการอินเตอร์เฟสระหว่างอุปกรณ์ฮาร์ดแวร์
2. จัดลำดับและการขัดจังหวะงาน (Scheduling และ Preempting Task)
3. จัดการหน่วยความจำ
4. จัดเตรียมบริการ ต่างๆเช่น คีย์บอร์ด , หน้าจอ , อุปกรณ์ไดรเวอร์ต่างๆ เป็นต้น

คุณลักษณะเด่นของ RTOS ที่แตกต่างจาก OS ได้แก่

1. Scalability สามารถเปลี่ยนแปลงขนาดได้

2. Scheduling Policies สามารถกำหนดรูปแบบการจัดลำดับของงาน อาทิเช่น Rate-Monotonic Scheduling (RMS), Deadline-Monotonic Scheduling (DMS) เป็นต้น
3. สนันสนุน ระบบ embedded

2.4 ลีนุกซ์

ลีนุกซ์ถือกำเนิดขึ้นในฟินแลนด์ ปี คศ. 1980 โดยไลนัส โทรวัลด์ส (Linus Trovalds) นักศึกษาภาควิชาวิทยาการคอมพิวเตอร์ในมหาวิทยาลัยเฮลซิงกิ

ไลนัส เห็นว่าระบบมินิกซ์ (Minix) ที่เป็นระบบยูนิกซ์บนพีซีในขณะนั้น ซึ่งทำการพัฒนาโดย ศ.แอนดรูว์ เอส ทาเนนบวาม (Andrew S. Tanenbaum) ยังมีความสามารถไม่เพียงพอแก่ความต้องการ จึงได้เริ่มต้นทำการพัฒนาระบบยูนิกซ์ของตนเองขึ้นมาโดยจุดประสงค์อีกประการคือต้องการทำความเข้าใจในวิชาระบบปฏิบัติการคอมพิวเตอร์ด้วยเมื่อเขาเริ่มพัฒนาลีนุกซ์ไปช่วงหนึ่งแล้ว เขาก็ได้ทำการชักชวนให้นักพัฒนาโปรแกรมอื่นๆมาช่วยทำการพัฒนาลีนุกซ์ ซึ่งความร่วมมือส่วนใหญ่ก็จะเป็นความร่วมมือผ่านทางอินเทอร์เน็ต

ไลนัสจะเป็นคนรวบรวมโปรแกรมที่ผู้พัฒนาต่างๆได้ร่วมกันทำการพัฒนาขึ้นมาและแจกจ่ายให้ทดลองใช้เพื่อทดสอบหาข้อบกพร่อง ที่น่าสนใจก็คืองานต่างๆเหล่านี้ผู้คนทั้งหมดต่างก็ทำงานโดยไม่คิดค่าตอบแทน และทำงานผ่านอินเทอร์เน็ตทั้งหมด

ปัจจุบันเวอร์ชันล่าสุดของระบบลีนุกซ์ที่ได้ประกาศออกมาคือเวอร์ชัน 2.4.3 ข้อสังเกตในเรื่องเลขรหัสเวอร์ชันนี้ก็คือ ถ้ารหัสเวอร์ชันหลังทศนิยมตัวแรกเป็นเลขคู่เช่น 1.0.x,1.2.x เวอร์ชันเหล่านี้จะถือว่าเป็นเวอร์ชันที่เสถียรแล้วและมีความมั่นคงในระดับหนึ่ง แต่ถ้าเป็นเลขคี่เช่น 1.1.x, 1.3.x จะถือว่าเป็นเวอร์ชันทดสอบ ซึ่งในเวอร์ชันเหล่านี้จะมีการเพิ่มเติมความสามารถใหม่ๆลงไป และยังต้องทำการทดสอบหาข้อผิดพลาดต่างๆอยู่

ลีนุกซ์เป็นระบบปฏิบัติการเสมือนยูนิกซ์ที่ได้รับการพัฒนาขึ้นมาโดยโปรแกรมเมอร์ทั่วโลกผ่านเครือข่ายอินเทอร์เน็ตเป็นระยะเวลายาวนานจนเป็นซอฟแวร์ระบบปฏิบัติการ ไม่เสียค่าใช้จ่ายนำมาใช้งานที่เชื่อถือได้ สามารถดาวน์โหลดได้ฟรีจากอินเตอร์เน็ต ลีนุกซ์เป็นเพียงเคอร์เนลตัวหนึ่ง สามารถนำโค้ดมาคอมไพล์แล้วติดตั้งพร้อมกับซอฟต์แวร์ตัวอื่นๆ ได้

โดยความหมายทางเทคนิคแล้วลีนุกซ์ เป็นเพียงเคอร์เนลของระบบปฏิบัติการ ซึ่งจะทำหน้าที่ในด้านของการจัดสรรและบริหารโปรเซส การจัดการไฟล์และอุปกรณ์ I/O ต่างๆ แต่ผู้ใช้ทั่วๆไปจะรู้จักลีนุกซ์ผ่านทางแอพพลิเคชั่นและระบบอินเตอร์เฟสที่เขาเหล่านั้นเห็น เช่น Shell หรือ X วินโดวส์

ถ้าคุณรันลีนุกซ์บนเครื่อง 386 หรือ 486 ของคุณ มันจะเปลี่ยนพีซีของคุณให้กลายเป็นยูนิกซ์เวิร์กสเตชันที่มีความสามารถสูง เคยมีผู้เทียบประสิทธิภาพระหว่างลีนุกซ์บนเครื่องเพนเทียม และเครื่องเวิร์กสเตชั่นของซันในระดับกลาง และได้ผลออกมาว่าให้ประสิทธิภาพที่ใกล้เคียงกัน

และนอกจากแพลตฟอร์มอินเทลแล้ว ปัจจุบันลีนุกซ์ยังได้ทำการพัฒนาระบบเพื่อให้สามารถใช้งานได้บนแพลตฟอร์มอื่นๆด้วย เช่น อัลฟา (Alpha), พาวเวอร์พีซี (PowerPC), มิปส์ (MIPS), อาร์ม

(ARM) เป็นต้น เมื่อคุณสร้างแอพพลิเคชั่นขึ้นมาบนแพลตฟอร์มใดแพลตฟอร์มหนึ่งแล้ว คุณก็สามารถย้ายแอพพลิเคชั่นของคุณไปวิ่งบนแพลตฟอร์มอื่นได้ไม่ยาก

ลีนุกซ์มีทีมพัฒนาโปรแกรมที่ต่อเนื่อง ไม่จำกัดจำนวนของอาสาสมัครผู้ร่วมงาน และส่วนใหญ่จะติดต่อกันผ่านทางอินเทอร์เน็ต เพราะที่อยู่อาศัยจริงๆของแต่ละคนอาจจะอยู่ไกลคนละซีกโลกก็ได้ และมีแผนงานการพัฒนาในระยะยาว ทำให้เรามั่นใจได้ว่า ลีนุกซ์เป็นระบบปฏิบัติการที่มีอนาคต และจะยังคงพัฒนาต่อไปได้ตราบนานเท่านาน

2.4.1 สถาปัตยกรรมของลีนุกซ์

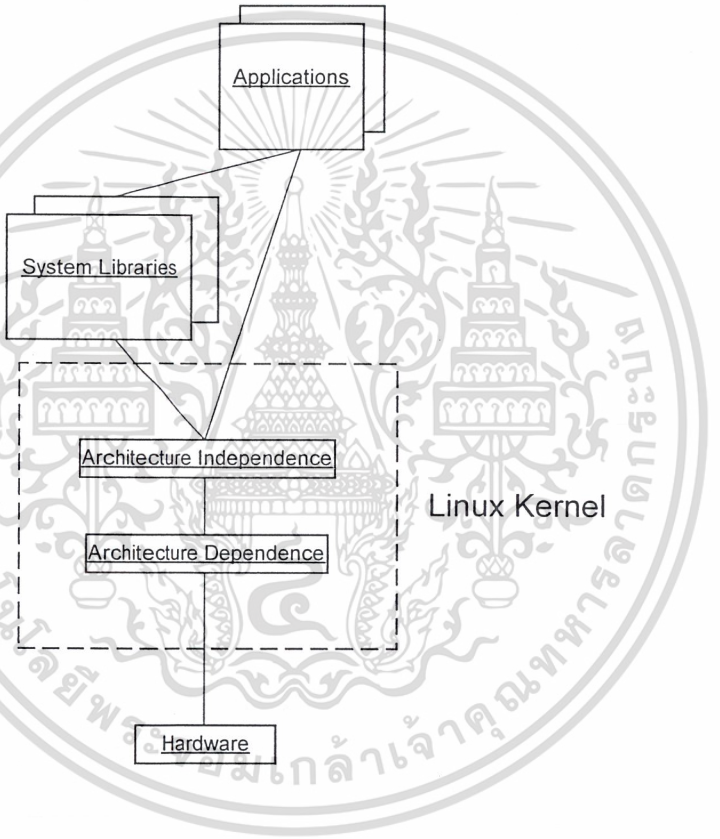
ลักษณะที่ต้องพิจารณาในการออกแบบลีนุกซ์เคอร์เนล ได้แก่ ความชัดเจน(clarity), ความเข้ากันได้ (compatibility), สะดวกในการโยกย้าย (portability), ทนทาน (robustness), ปลอดภัย (security), และเร็ว (speed) ซึ่งสามารถแยกพิจารณาทีละส่วนดังนี้

1. Clarity คือ ลีนุกซ์เคอร์เนลที่ออกแบบขึ้นมาต้องมีความชัดเจนสามารถทำความเข้าใจได้ง่าย ทั้งส่วนเรื่องของความเร็วและความทนทาน โดยความชัดเจนในเรื่องความทนทานหมายความว่า สามารถพิสูจน์ความถูกต้องของข้อมูลได้ง่าย และหากเกิดความผิดพลาดขึ้นสามารถทำการดีบักได้ง่าย ระบบเกิดความเสียหายได้ยาก แต่ในส่วนของความชัดเจนกับความเร็วจะตรงข้ามกัน ถ้าในส่วนตัวคอมไพล์โค้ด ความชัดเจนจะสำคัญกว่า แต่ในการออกแบบเคอร์เนล ควรเน้นไปที่ความเร็ว เพราะผู้พัฒนาสามารถทำการอธิบายเกี่ยวกับความชัดเจนได้อยู่แล้ว
2. Compatibility ลีนุกซ์เคอร์เนลสามารถนำไปใช้งานร่วมกับยูนิกซ์ได้โดยสนับสนุนมาตรฐาน POSIX และยังใช้งานร่วมกับตัวอื่นๆ ได้แก่ สนับสนุนการรันไฟล์ . class ของจาวา สามารถทำการเอ็กซ์คิว DOS ผ่าน DOSEMU อีมูเลเตอร์ ทำการแชร์ไฟล์และเครื่องพิมพ์ได้เหมือนกันวินโดว์โดยใช้ SAMBA สามารถใช้งานไดร์ฟจากระบบปฏิบัติการอื่นๆได้ รวมทั้งสนับสนุนระบบเน็ตเวิร์กด้วยโปรโตคอล TCP/IP
3. Portability หมายความว่าลีนุกซ์เคอร์เนลสามารถใช้งานร่วมกับ ฮาร์ดแวร์ โดยสามารถรันลีนุกซ์เคอร์เนลบนฮาร์ดแวร์หลายแพลตฟอร์มได้ เริ่มแรกทำการพัฒนาลีนุกซ์บนสถาปัตยกรรม X86 แต่ปัจจุบันทำการพัฒนาลีนุกซ์เคอร์เนลที่สามารถใช้งานบนสถาปัตยกรรมหลายแพลตฟอร์มได้แก่ อัลฟา, ARM, โมโตโรล่า, MIPS, พาวเตอร์พีซี, SPARC, และ SPARC-64 เป็นต้น ยิ่งไปกว่านั้นสามารถรันลีนุกซ์บน Amigas, Mac, เวิร์กสเตชั่นจากซันและ SGI, NeXT และอื่นๆอีกมากมาย การที่ลีนุกซ์สามารถทำงานบนหลายแพลตฟอร์มได้เนื่องจากว่า โค้ดของลีนุกซ์เคอร์เนลไม่ขึ้นอยู่กับสถาปัตยกรรม ทำให้สามารถนำไปใช้งานได้หลากหลาย
4. Robustness และ Security ลีนุกซ์เคอร์เนลที่พัฒนาขึ้นต้องมีความทนทานและปลอดภัย ควรจะมีการป้องกันระบบจากระบบอื่น โดยเฉพาะทำการป้องกันโปรเซสที่ทำงานจากส่วนอื่นๆ สามารถทำการดีบัก และแก้ไขข้อผิดพลาดได้ง่าย

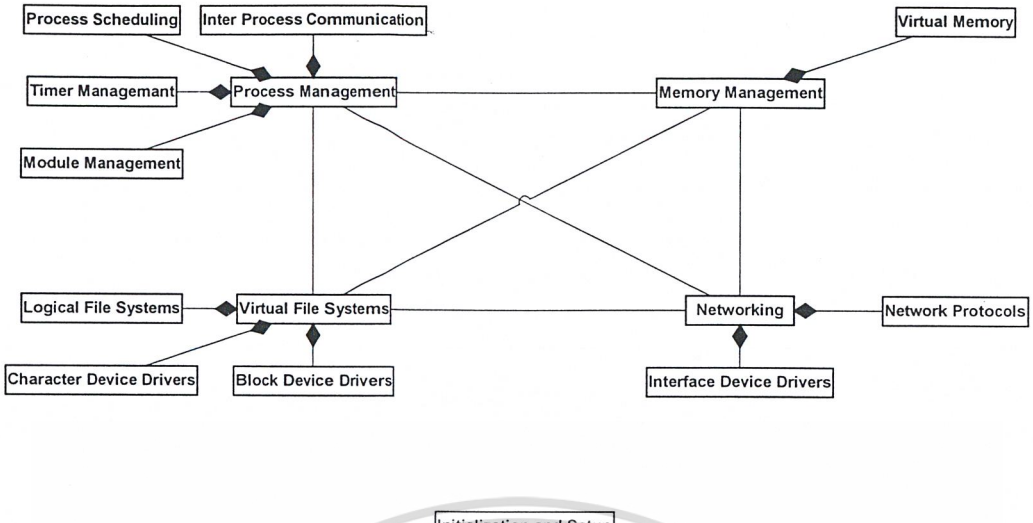
5. Speed ในการออกแบบลีนุกซ์เคอร์เนลส่วนที่ต้องพิจารณาอันดับแรกซึ่งมีความสำคัญมากได้แก่ ความเร็ว ตามมาด้วย ความทนทาน ความปลอดภัย และความเข้ากันได้ เพราะสามารถตรวจสอบประสิทธิภาพการทำงานของระบบได้จากความเร็วในการทำงาน การรันงานต่างๆ ซึ่งบางครั้ง สามารถจับเวลาในการรันโปรเซส ด้วยตัวเองได้

จากรูปที่ 2-1 แสดงภาพรวมภายในสถาปัตยกรรมของเคอร์เนลที่เหมือนกับยูนิกซ์ โดยมีคุณสมบัติของเคอร์เนลได้แก่

1. เคอร์เนลทำการแยกแอพพลิเคชั่นออกจากฮาร์ดแวร์
2. สถาปัตยกรรมของเคอร์เนลประกอบด้วย ฮาร์ดแวร์และพอร์ตเทเบิล



รูปที่ 2-1 ภาพรวมสถาปัตยกรรมเคอร์เนล



รูปที่ 2-2 รายละเอียดเพิ่มเติมของสถาปัตยกรรมเคอร์เนล

จาก รูปที่ 2-2 แสดง สถาปัตยกรรมของเคอร์เนลซึ่งประกอบด้วยส่วนประกอบหลัก 5 ส่วนดังนี้

1. Process Management
2. Memory Management
3. Filesystems
4. Networking
5. Device Control

2.4.1.1 Process Management

เคอร์เนลทำหน้าที่สร้างและทำลายโปรเซส และทำการแฮนเดิลโปรเซสที่ติดต่อกับภายนอก (อินพุตและเอาต์พุต) โดยสามารถติดต่อผ่านซิกเนล, ไปป์ และติดต่อภายในโปรเซสด้วยกันเอง รวมถึงมีตัวจัดลำดับการรันโปรเซสนั่นคือ scheduler

2.4.1.2 Memory Management

หน่วยความจำภายในคอมพิวเตอร์เป็นทรัพยากรหลักของระบบ ซึ่งสามารถใช้วัดประสิทธิภาพของระบบได้ เคอร์เนลทำการจัดสรรพื้นที่ว่างในหน่วยความจำให้กับโปรเซส โดยส่วนประกอบต่างๆ ของเคอร์เนลทำการติดต่อกับส่วนของการจัดการหน่วยความจำผ่านชุดของฟังก์ชั่น call ได้แก่ malloc/free

2.4.1.3 Filesystems

ระบบยูนิกซ์อยู่บนแนวคิดพื้นฐานของ filesystem โดยมองส่วนต่างๆ ภายในยูนิกซ์เป็นไฟล์ เคอร์เนลทำการสร้างโครงสร้างของ filesystem บนฮาร์ดแวร์ โดยลีนุกซ์สนับสนุน multiple filesystem ซึ่งมีหนทางในการจัดการกับดาต้าบนสื่อที่แตกต่างกัน

2.4.1.4 Networking

ในส่วนของเน็ตเวิร์กต้องมีการจัดการโดยระบบปฏิบัติการ เนื่องจากว่าโอเปอเรชั่นของเน็ตเวิร์กส่วนใหญ่แล้วต้องไม่ถูกระบุในโปรเซส ซึ่งแพ็กเกจที่เข้ามาเป็นเหตุการณ์แบบอะซิงโครนัส ทำให้ต้องมีการจัดกลุ่ม กำหนดชื่อ และ เลือกแพ็กเกจก่อนที่โปรเซสจะใช้งานแพ็กเกจเหล่านี้ ระบบทำการส่งข้อมูลของแพ็กเกจผ่านโปรแกรมและเน็ตเวิร์กอินเตอร์เฟส ซึ่งโปรแกรมต้องทำการกำหนดการคอยข้อมูลจากเน็ตเวิร์ก ได้อย่างถูกต้อง

2.4.1.5 Device Control

ส่วนใหญ่ระบบปฏิบัติการทั่วๆ ไปทำการแมปไปยังอุปกรณ์ภายนอก ยกเว้นโปรเซสเซอร์, หน่วยความจำ, เอนติตี้ 2-3 ตัว และอุปกรณ์ที่ใช้ในการควบคุมโอเปอเรชั่นจะปฏิบัติตามโค้ดที่ทำระบุดีไวซ์ ซึ่งโค้ดที่กล่าวถึงนี้คือ ดีไวซ์ไดรเวอร์ (device driver) เคอร์เนลต้องมีส่วนของดีไวซ์ไดรเวอร์สำหรับอุปกรณ์ที่เชื่อมต่อกับระบบ เช่น ไดรเวอร์ของฮาร์ดดิสก์, คีย์บอร์ด, การ์ดจอ เป็นต้น

2.4.2 ลักษณะเด่นของลีนุกซ์

1. เป็นระบบปฏิบัติการแบบ 32 บิต ที่เป็นยูนิกซ์โคลน สำหรับเครื่องพีซี และแจกจ่ายให้ใช้ฟรี
2. สนับสนุนการใช้งานแบบหลากงาน หลายผู้ใช้ (MultiUser-MultiTasking)
3. มีการปกป้องหน่วยความจำ
4. หน่วยความจำเป็นแบบเสมือน
5. มีระบบ X วินโดวส์ ซึ่งเป็นระบบการติดต่อผู้ใช้แบบกราฟิก ที่ไม่ขึ้นกับโอเอสหรือฮาร์ดแวร์ใดๆ (มักใช้กันมากในระบบยูนิกซ์)
6. มาตรฐานการสื่อสาร TCP/IP ที่ใช้เป็นมาตรฐานการสื่อสารในอินเทอร์เน็ตมาให้ในตัว
7. ลีนุกซ์มีความเข้ากันได้ (compatible) กับ มาตรฐาน POSIX ซึ่งเป็นมาตรฐานอินเทอร์เฟสที่ระบบยูนิกซ์ส่วนใหญ่จะต้องมีและมีรูปแบบบางส่วนที่คล้ายกับระบบปฏิบัติการยูนิกซ์จากค่าย Berkeley และ System V
8. เร็วและมีเสถียรภาพสูง

2.4.3 ข้อดีของลีนุกซ์

ลีนุกซ์เป็นระบบปฏิบัติการที่เหมาะในการนำมาใช้งานในระบบ embedded ด้วยเหตุผลดังนี้คือ

1. เนื่องจากเป็นระบบปฏิบัติการที่ฟรี คุณสามารถจะขอจากผู้ที่มีลีนุกซ์ หรือจะดาวน์โหลดจากอินเตอร์เน็ต หรือบีบีเอสได้โดยไม่ผิดกฎหมาย เหมาะในการใช้กับระบบ embedded เนื่องจากสามารถควบคุมขนาดของระบบได้ ยังสามารถทำการดีบักและตรวจสอบความผิดพลาดที่อาจเกิดขึ้นได้

2. เนื่องจากมีผู้นิยมใช้มาก ทำให้มีผู้นำลีนุกซ์ไปแก้ไขให้สามารถใช้งานได้บนตัวประมวลผลกลางหลากหลายตั้งแต่อินเทล,โ มโตโรล่า,ดิจิ ตอลอัลฟา,พ าวเวอร์พีซี,ไ ปจนถึง สปาร์ค ของซัน นอกจากนี้ยังมีผู้พัฒนาโปรแกรมประยุกต์ออกมากันมากมาย
3. มีประสิทธิภาพและมีคุณภาพสูง ลีนุกซ์เป็นระบบปฏิบัติการ 32 บิตเต็มรูปแบบ ซึ่ง สามารถจะดึงเอาพลังของเครื่องคอมพิวเตอร์ออกมาได้อย่างเต็มกำลัง ลีนุกซ์ถูกพัฒนา จากผู้พัฒนานับร้อยทั่วโลก แต่ ไลนัส จะเป็นคนวางทิศทางในการพัฒนาด้วยตัวเอง
4. มีคุณลักษณะของระบบ UNIX เต็มรูปแบบ และเป็นระบบหลากผู้ใช้ หลายงานอย่าง แท้จริง ลีนุกซ์มีระบบอินเตอร์เฟสแบบกราฟฟิกที่เรียกกันว่า X Windows ซึ่งเป็น มาตรฐานของระบบยูนิกซ์ทั่วๆ ไป และสามารถใช้ window manager ได้หลายชนิด ตามความต้องการ นอกจากนี้ยังสนับสนุนโปรโตคอลแบบ TCP/IP ,SLIP, PPP, UUCP และอื่นๆ
5. คุณสามารถหาข้อมูลเพิ่มเติมได้ง่าย มีเอกสารหลากหลาย และผู้คนมากมายคอยสนับสนุนคุณผ่านอินเตอร์เน็ต หรือคุณอาจจะหาการสนับสนุนจากบริษัทที่ปรึกษา หรือจากบริษัทผู้จัดจำหน่ายระบบลีนุกซ์ก็ได้

มีเหตุผลหลายประการที่สามารถอธิบายได้ว่าทำไมผู้คนถึงชอบลีนุกซ์ แต่โดยส่วนตัวแล้ว น่าจะเป็นเพราะการพัฒนาอย่างรวดเร็วของลีนุกซ์ เนื่องจากคุณสามารถเห็นการเปลี่ยน แปลงตัวเคอร์เนล และการพัฒนาโปรแกรมประยุกต์ใหม่ๆ ออกมาอย่างรวดเร็ว ซึ่งไม่เคยพบเห็นในระบบที่แจกจ่ายฟรีแบบนี้ที่ไหนมาก่อนเลย

2.5 เรียลไทม์ลีนุกซ์

จุดมุ่งหมายที่ ไลนัส ทำการพัฒนาลีนุกซ์ขึ้นมาก็เพื่อนำไปใช้งานโดยไม่เสียค่าใช้จ่าย อยู่บนพื้นฐานของระบบยูนิกซ์ ที่ไม่ใช่ยูนิกซ์แบบเรียลไทม์ และได้สืบทอดคุณลักษณะของยูนิกซ์ทั้งหมด

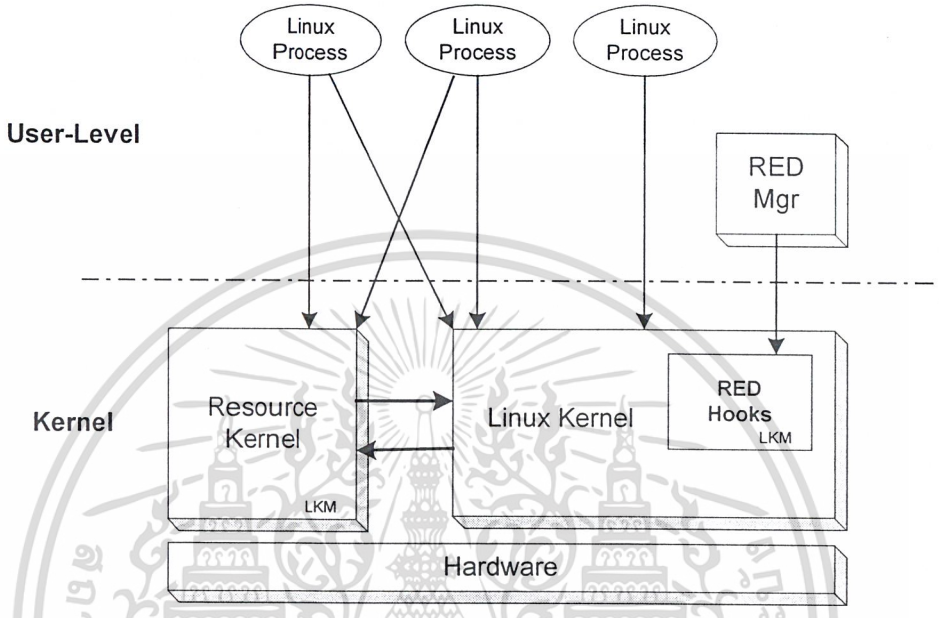
แท้ที่จริงแล้ว เคอร์เนลของลีนุกซ์ไม่ได้ถูกออกแบบมาเพื่อใช้ในงานแบบเรียลไทม์ แต่ในปัจจุบันมีผลงานหลายผลงานที่ทำการพัฒนาลีนุกซ์ให้เป็นแบบเรียลไทม์ เนื่องจากข้อดีต่างๆที่ได้กล่าวมาแล้วในข้างต้น จากการศึกษาเรียลไทม์ลีนุกซ์ที่มีอยู่ในปัจจุบัน มีผลงานที่น่าสนใจในการนำมาใช้ในการพัฒนากับโครงงานดังนี้

1. TimeSys/RT
2. RTLinux
3. RTAI

2.5.1 TimeSys/RT

TimeSys Linux/RT แตกต่างจากระบบเรียลไทม์ลีนุกซ์ทั่วๆไป คือ ทำการเปลี่ยนแปลงแก้ไขเคอร์เนลเพื่อให้สามารถแฮนเดิลแอพพลิเคชั่นแบบเรียลไทม์ได้ หมายความว่าถ้าโปรเซสแบบเรียลไทม์บางโปรเซสเกิดความเสียหายขึ้น โปรเซสอื่นๆ และเคอร์เนลยังสามารถทำงานต่อไปได้ ซึ่งถ้าเป็นระบบเรียลไทม์ลีนุกซ์ตัวอื่น ถ้าโปรเซสแบบเรียลไทม์พังเสียหายก็มีผลให้ระบบเกิดความเสียหายด้วย

TimeSys Linux /RT ใช้หลักการพื้นฐานของ Linux/RK (Linux resource kernel) พัฒนาที่มหาวิทยาลัยคาร์เนเกียเมลลอน (Carnegie Mellon University) โดย Linux/RK เป็นโมดูลที่อยู่ใน TimeSys/RT ซึ่งถูกดึงมาใช้งานเมื่อจำเป็น โค้ดของ TimeSys/RT ถูกนำไปไว้ในลีนุกซ์เคอร์เนลทุกครั้งที่ต้องการใช้งานแบบเรียลไทม์



รูปที่ 2-3 สถาปัตยกรรมของ Timesys/RK

จากรูปที่ 2-3 แสดงสถาปัตยกรรมของ TimeSys Linux /RT ซึ่งประกอบด้วยส่วนประกอบพื้นฐาน 3 ส่วนได้แก่ ส่วนที่เป็นของลีนุกซ์เดิม , Resource Kernel (RK), RED Linux โดยมีรายละเอียดดังนี้

Resource Kernel ทำหน้าที่โหลดโมดูลไปยังเคอร์เนล (Linux Loadable Kernel Module : LKM) เพื่อใช้แทนฟังก์ชั่นแบบเรียลไทม์ของลีนุกซ์ ตัวอย่างเช่น RK สนับสนุนการจัดลำดับงานแบบ fixed-priority ซึ่งมีได้สูงสุด 256 priority และแบบ priority inheritance ซึ่งใช้เทคนิค Rate Monotonic Analysis

RED Linux ผลงานของ ยูชุง วาง (Yu-Chung Wang) และ ไกเจ ลิน (Kwei-Jay Lin) ณ มหาวิทยาลัยแคลิฟอร์เนีย ประกอบด้วยระบบเรียลไทม์และระบบ embedded ใช้ลีนุกซ์เวอร์ชั่น 2.2.14 ซึ่งสนับสนุนสำหรับเหตุการณ์ logging ของระบบและผู้ใช้ โดยจะมี " hooks" เฉพาะ

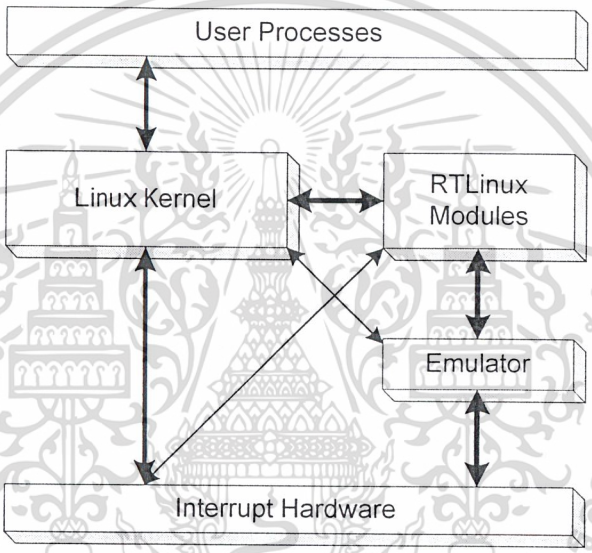
ลักษณะเด่นของ TimeSys Linux/RT

1. เหมาะกับระบบ embedded, diskless
2. ใช้กับสถาปัตยกรรม x86, PowerPC, ARM/StrongARM และ MIPs
3. มีเครื่องมือพัฒนาอย่างครบถ้วนได้แก่ TimeTrace, TimeWarp, TimeWiz และ TimeBench
4. สนับสนุนมาตรฐาน POSIX
5. 256 fixed-priority
6. สนับสนุน priority inheritance เพื่อหลีกเลี่ยงปัญหา priority inversion
7. High-resolution Timer

8. รับประกัน QoS ทั้งแอพพลิเคชั่นแบบเรียลไทม์และไม่ใช่เรียลไทม์
9. สนับสนุนโปรเซสแบบ periodic

2.5.2 RTLinux

RTLinux เป็นฮาร์ดเรียลไทม์ทำการวิจัยที่ NMT (New Mexico Tech) พัฒนามาเพื่อใช้ในห้องปฏิบัติการ เพื่อควบคุมเครื่องมือต่าง ๆ เช่น data acquisition หรือตัวเซนเซอร์ รวมทั้งในระบบ embedded เช่น หุ่นยนต์, เทเลสโคป เป็นต้น โดย RTLinux สามารถให้บริการในรูปแบบฮาร์ดเรียลไทม์ได้แก่ คาดเดาเวลาได้,การตอบรับที่เร็ว,มี การจัดลำดับงานที่ง่าย รวมทั้งสามารถให้บริการตามมาตรฐาน POSIX ได้แก่ GUI, TCP/IP, NFS, คอมไพเลอร์ และ เว็บเซิร์ฟเวอร์ เป็นต้น



รูปที่ 2-4 สถาปัตยกรรมของ RTLinux

แอพพลิเคชั่นที่ใช้ใน RTLinux แบ่งเป็น 2 ส่วนได้แต่ หนึ่งในส่วนของฮาร์ดเรียลไทม์โดยทำการเอ็กซิคิวต์เหมือนกับงานในแบบเรียลไทม์ โดยทำการก๊อปปี้ข้อมูลจากอุปกรณ์ไปยังตัวอินเตอร์เฟสอุปกรณ์อินพุทเอาต์พุทที่มีลักษณะพิเศษที่เรียกว่า *Real-Time fifo* และสองเป็นส่วนของโปรแกรมหลักที่จะทำการเอ็กซิคิวต์โปรเซสธรรมดาทั่วๆไปโดยทำการอ่านข้อมูลจากส่วนอื่น ๆ นอกเหนือจาก *Real-Time fifo* แล้วทำการแสดงผลจากนั้นทำการเก็บข้อมูลไว้ในไฟล์ ส่วนประกอบต่าง ๆของเรียลไทม์ถูกเขียนในโมดูลของเคอร์เนลแสดงดังรูปที่ 2-4 ซึ่งลีนุกซ์จะอนุญาตให้ทำการคอมไพล์และโหลด โมดูลของเคอร์เนล นอกเหนือจากการรีบูตระบบ มีตัวอีมูเลเตอร์ใช้สำหรับตรวจสอบการอินเตอร์รัพท์ที่เกิดขึ้นจากฮาร์ดแวร์ โดยลีนุกซ์จะไม่สามารถดิสเอเบิลอินเตอร์รัพท์ที่เกิดขึ้นได้ และทำการรันโปรเซสของลีนุกซ์เป็นงานที่มีความสำคัญที่ต่ำ

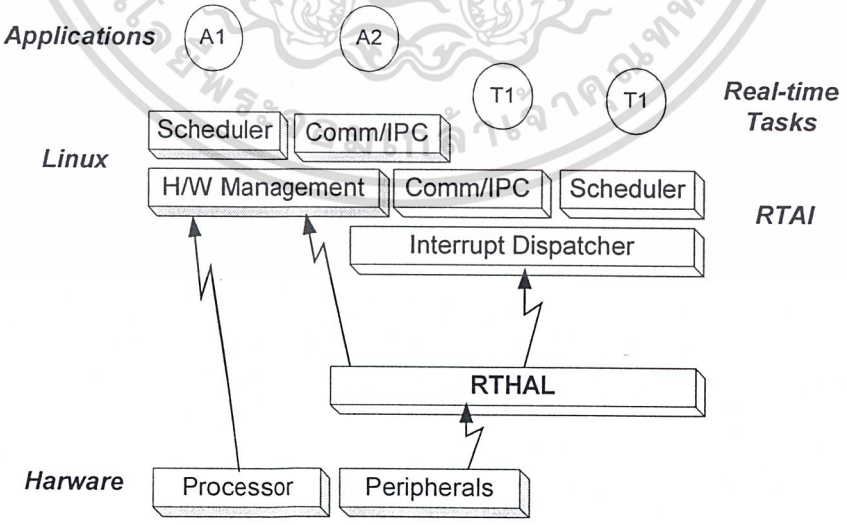
ข้อจำกัดของ RTLinux คือการให้บริการต่าง ๆที่ถูกจัดเตรียมโดย RTLinux core ไม่สามารถ เรียกใช้ API มาตรฐานหรือ ดีไวซ์ไดรเวอร์ของลีนุกซ์ได้ เหมือนกับว่า โปรเซสของลีนุกซ์ ไม่สามารถเรียกใช้

บริการจาก RTLinux ได้ ส่วนข้อดีของ RTLinux คือว่าจะเหมือนเป็นตัวดีบัก เมื่อมีบักเกิดขึ้นเราสามารถหาและทำการ fixed ได้ง่าย

2.5.3 RTAI

RTAI เป็นผลงานของ DIAPM (Dipartimento di Ingegneria AerOSpaziale - Politecnico di Milano) พัฒนามาจาก RTLinux โดยมีลักษณะดังนี้

1. สนับสนุนมาตรฐาน POSIX 1003.1c API ที่ประกอบไปด้วย pthreads, เงื่อนไขของตัวแปร และ mutexes ที่สนับสนุน priority inheritance
2. RTOS IPCs ประกอบด้วย Semaphores, named FIFOS, mailboxes, mutexes, RPCs, shared memory
3. PERL Bindings สำหรับสร้างงานเรียลไทม์แบบ soft real-time tasks จาก PERL scripts
4. สนับสนุน SMP (symmetric multi-processing)
5. The Real-Time Linux Common API ที่อนุญาตให้ผู้พัฒนา single API สำหรับแต่ละโค้ด RTLinux or RTAI ที่ใช้ในการพัฒนา
6. ใช้กับงานลักษณะ ฮาร์ดเรียลไทม์ ยังคงมีลักษณะเด่นและการให้บริการของลีนุกซ์ทั้งหมด
7. สนับสนุน UP และ SMP ซึ่งสามารถกำหนดได้ทั้ง 2 งาน และ IRQs เพื่อระบุซีพียูได้แก่ x486 และเพนเทียม
8. ทำ one-short และ periodic schedulers ในเวลาพร้อมกันได้
9. มีการแชร์หน่วยความจำทั้งภายในและภายนอกลีนุกซ์
10. สนับสนุน FPU (Floating-Point Operations)



รูปที่ 2-5 แสดงสถาปัตยกรรมของ RTAI

จาก รูปที่ 2-5 แสดงสถาปัตยกรรมของ RTAI ซึ่งใช้ RTHAL (Real-Time Hardware Abstract Layer) ในการติดต่อกับฮาร์ดแวร์ ซึ่งปกติเมื่อไม่เกิดการอินาเบิลอินเตอร์รัพท์จาก RTAI ตัว RTHAL จะ

ส่งอินเตอร์รัพท์จากฮาร์ดแวร์ไปยังลีนุกซ์ แต่ถ้า RTAI ทำการอินาเบิลอินเตอร์รัพท์จากฮาร์ดแวร์ ตัว RTHAL จะเป็นตัวตัดสินใจว่าจะส่งอินเตอร์รัพท์ให้กับ ลีนุกซ์ หรือ RTAI ซึ่งถ้า ส่งอินเตอร์รัพท์ให้กับ RTAI จะมี Interrupt Dispatcher ทำหน้าที่ในการจัดการเกี่ยวกับการให้บริการและทรัพยากรต่างๆที่ งานแบบเรียลไทม์ต้องการ

ในส่วนโปรเซสของลีนุกซ์ เมื่อ RTHAL ส่งอินเตอร์รัพท์จากฮาร์ดแวร์ให้กับลีนุกซ์ ตัวลีนุกซ์ทำหน้าที่จัดการทั้งหมด เรานำหลักการ RTHAL มาใช้ทำให้ไม่ไปยุ่งเกี่ยวกับฮาร์ดแวร์มากนัก

RTAI มีสถาปัตยกรรมเหมือน RTLinux ซึ่งอนุญาตให้งานที่เป็นแบบเรียลไทม์และอินเตอร์รัพท์แฮนเดิลมีการติดต่อสื่อสารกันโดยผ่านอุปกรณ์อินเตอร์เฟสหรือหน่วยความจำที่แชร์กันอยู่

ข้อดีของ RTAI และ RTLinux คือจะทำการโหลดโมดูลของเคอร์เนลเมื่อให้บริการงานเรียลไทม์ แต่ระหว่าง RTAI และ RTLinux ยังมีข้อแตกต่าง กันนั่นคือ RTLinux ทำการเปลี่ยนแปลง ซอร์สไฟล์ ของเคอร์เนลโดยตรง โดยทำการเพิ่มเติมโค้ด ซึ่งเป็นผลทำให้เพิ่มโค้ดในการบำรุงรักษาขึ้น การที่เราทำการเปลี่ยนแปลงแก้ไขไฟล์เคอร์เนล เป็นผลให้เกิดความยุ่งยากในการหาความผิดพลาดที่เกิดขึ้นมา ส่วน RTAI จะพยายามจำกัดการเปลี่ยนแปลงเคอร์เนลโดยเพิ่ม HAL ซึ่งประกอบด้วยโครงสร้างของพอยเตอร์สำหรับ อินเตอร์รัพท์เวกเตอร์ (Interrupt vectors) และ อินเตอร์รัพท์ อินาเบิล/ดิสเอเบิล ฟังก์ชั่น (interrupt enable/disable functions) ซึ่ง HAL จะทำการเปลี่ยนแปลงโค้ดไม่เกิน 20 บรรทัด และทำการเขียนโค้ดเพิ่มอีกประมาณ 50 บรรทัด ทำให้ไม่ไปยุ่งกับ เคอร์เนลมากนัก

ข้อดีของการใช้เทคนิค HAL ได้แก่

- จำกัดการเปลี่ยนแปลงแก้ไขที่เคอร์เนล ซึ่งเราทำการแก้ไขเพียงไม่กี่บรรทัด
- ได้ลักษณะเรียลไทม์ที่เพิ่มเติมขึ้นมาสามารถเอาออกได้ง่ายโดยนำรูทีนของลีนุกซ์มาแทนอินเตอร์รัพท์ฟังก์ชั่นพอยเตอร์ (interrupt function pointer) สามารถทำการดีบักได้ง่าย

ข้อเสีย คือ ประสิทธิภาพของระบบจะสูญหายไปหาก RTAI ที่ทำการเพิ่มขึ้นมากเกิดจากการรีไดเรก (redirection) ผ่านพอยเตอร์ แต่ถ้าเราทำการพิจารณาทั้งจุดอ่อนจุดแข็ง เทคนิคนี้จะก่อประสิทธิผล (Efficient) รวมทั้งมีความยืดหยุ่น (Flexible) สามารถรับประกัน การจัดลำดับของงานและเวลาที่ใช้ในการตอบรับได้

2.5.3.1 RTHAL

RTHAL นำมาใช้สำหรับให้ RTAI สามารถตรวจสอบและรีไดเร็ก (Redirect) การอินเตอร์รัพท์ได้ โดย RTHAL ทำหน้าที่ขีดขวางการอินเตอร์รัพท์จากฮาร์ดแวร์ทั้งหมดและกำหนดเส้นทางของการอินเตอร์รัพท์เหล่านั้นว่าควรส่งให้ลีนุกซ์หรือ งานแบบเรียลไทม์ โดยขึ้นอยู่กับความต้องการของ RTAI Scheduler

อินเตอร์รัพท์ที่ต้องการสำหรับงานแบบเรียลไทม์จะถูกส่งไปยังงานนั้นโดยตรง ซึ่ง RTHAL ทำการจัดเตรียมเฟรมเวิร์ก (Framework) เพื่อให้ RTAI สามารถทำการขัดจังหวะลีนุกซ์ได้อย่างเต็มที่

ส่วนประกอบหลักของ RTHAL คือ IDT (Interrupt Descriptor Table) ซึ่งประกอบด้วยกลุ่มของพอยเตอร์ที่กำหนดไว้สำหรับโปรเซสที่ถูกอินเตอร์รัพท์ IDT ช่วยให้ใช้งานและดิสเอเบิล การให้บริการของ RTHAL ได้ทั้งหมด

หน้าที่หลักของ RTHAL

1. รวบรวมพอยเตอร์ทั้งหมดที่ต้องการสำหรับข้อมูลภายในและฟังก์ชั่นต่างๆที่อยู่ในโครงสร้างเดียวกัน โดย RTHAL ช่วยในการแทรป (Trap) ฟังก์ชั่นของเคอร์เนลทั้งหมดที่มีความสำคัญสำหรับเรียลไทม์แอพพลิเคชั่น โดยสามารถใช้ RTAI ในการสวิตช์ไปยังฮาร์ดเรียลไทม์เมื่อใดก็ได้ตามต้องการ
2. เป็นตัวแทนของฟังก์ชั่นที่อยู่บนส่วนอื่นและกลุ่มของพอยเตอร์ของ RTHAL จะทำการชี้ไปยังฟังก์ชั่นเหล่านั้น
3. เป็นตัวแทนของฟังก์ชั่นที่กำลัง call อยู่โดยทำการ call ไปยังพอยเตอร์ของ RTHAL ที่อยู่ในฟังก์ชั่น ของเคอร์เนลทั้งหมด เพื่อใช้งานฟังก์ชั่นนั้นๆ

ลีนุกซ์ไม่โดนกระทบกระเทือนเมื่อใช้ RTHAL แต่ทำให้ประสิทธิภาพลดลงเล็กน้อยเมื่อทำการ call cli และ sti รวมทั้งแฟลกที่เกี่ยวข้องกับฟังก์ชั่นเหล่านี้ โดยขึ้นอยู่กับว่าทำการ call ฟังก์ชั่นและมาโครตัวไหนของลีนุกซ์ อีกทั้งลีนุกซ์ใช้พอยเตอร์ในส่วนของโครงสร้างที่สูงขึ้นไปเป็นไปได้ที่ RTAI ทำการเปลี่ยนฟังก์ชั่นที่ลีนุกซ์ใช้

ส่วนประกอบภายใน RTHAL

1. พอยเตอร์ที่ชี้ไปยัง IDT
2. ฟังก์ชั่นเพื่อทำการอินาเบิล/ดิสเอเบิลอินเตอร์รัพท์ที่เกิดขึ้น (cli, sti & flags)
3. ฟังก์ชั่น เพื่อทำการ มาส์กและอันมาส์ก อินเตอร์รัพท์คอนโทรลเลอร์ (8259)
4. ฟังก์ชั่นเพื่อจัดการกับ SMP (APIC)
5. Data descriptor สำหรับการอินเตอร์รัพท์ (status, handler, nested level,...)

2.5.3.2 โครงสร้างของ RTHAL

```
struct rt_hal
{
        struct desc_struct *idt_table;
        void (*disint)(void);
        void (*enint)(void);
        unsigned int (*getflags)(void);
        void (*setflags)(unsigned int flags);
        void (*mask_and_ack_8259A)(unsigned int irq);
        void (*unmask_8259A_irq)(unsigned int irq);
        void (*ack_APIC_irq)(void);
        void (*mask_IO_APIC_irq)(unsigned int irq);
        void (*unmask_IO_APIC_irq)(unsigned int irq);
        unsigned long *io_apic_irqs;
        void * irq_controller_lock;
        void *irq_desc;
        int *irq_vector;
        void *irq_2_pin;
```

};

เมื่อใช้หลักการของ HAL จะทำการเปลี่ยนแปลงโค้ดประมาณ 70 บรรทัด ในไฟล์ต่างๆ ดังนี้

/usr/src/linux/arch/i386/kernel/entry.S

/usr/src/linux/arch/i386/kernel/irq.c

/usr/src/linux/arch/i386/kernel/i386_ksyms.c

/usr/src/linux/arch/i386/kernel/smp.c

/usr/src/linux/arch/i386/kernel/time.c

/usr/src/linux/arch/i386/kernel/io_apic.c

สถาปัตยกรรมซอฟต์แวร์ของ RTAI ทำมาจาก

1. 1 I/F เพื่อจัดการเกี่ยวกับฮาร์ดแวร์ของลีนุกซ์ซึ่งใช้ HAL จะเป็นโครงสร้างข้อมูลพื้นฐาน ส่วนประกอบพื้นฐานได้แก่ dispatcher, scheduler, fifo's

1 I/F (กลุ่มของฟังก์ชั่น) ใช้ในงานของผู้ใช้เพื่อกำหนดค่าเริ่มต้นและเริ่มใช้งานส่วนประกอบพื้นฐาน

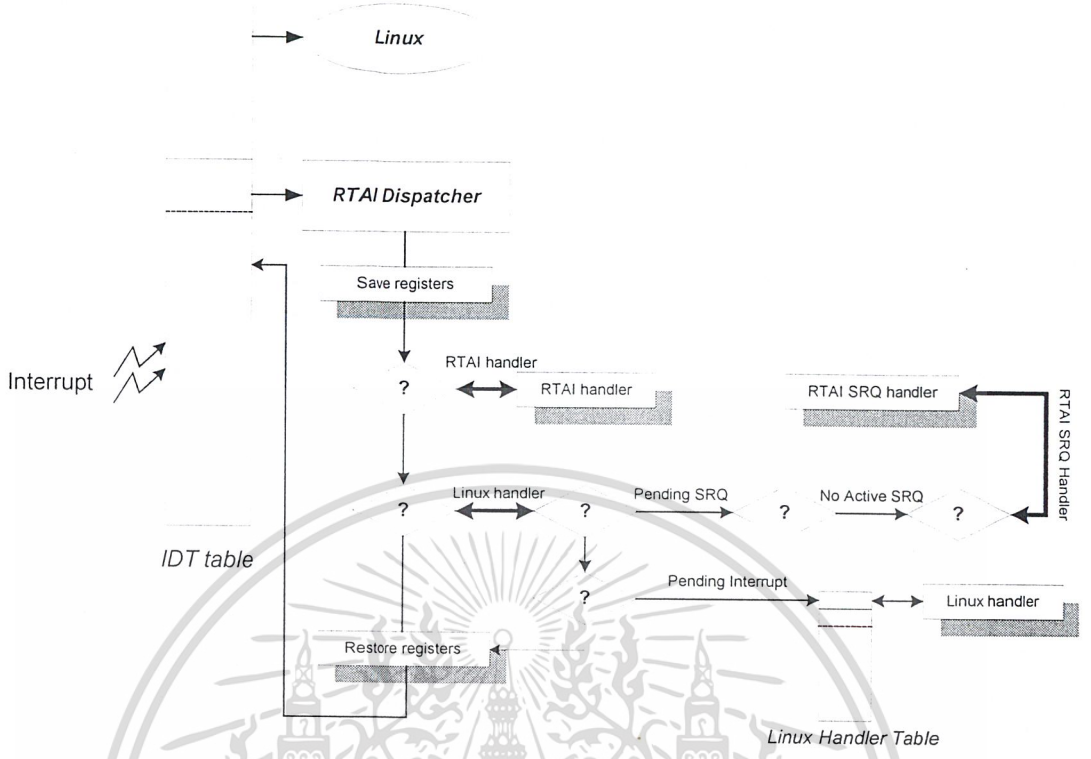
I/F เป็นโครงสร้างที่เกี่ยวข้องกับฟังก์ชั่นและดาต้าที่สำคัญนั่นคือ RTHAL ซึ่งทำการเปลี่ยนแปลงแก้ไขเคอร์เนลประมาณ 2-3 ไฟล์ เพื่อสามารถใช้งาน RTHAL ได้ ซึ่ง HAL ถูกใช้งานโดย RTAI เพื่อทำการควบคุมฮาร์ดแวร์ และสำหรับลีนุกซ์เพื่อให้สามารถเข้าถึงฟังก์ชั่นที่ถูก RTAI ทำการแก้ไขได้ ข้อดีของ RTHAL

2. ง่ายในการแก้ไขเปลี่ยนแปลงเพื่อใช้กับลีนุกซ์เคอร์เนลเวอร์ชั่นอื่น ทำการพอร์ต RTAI จากลีนุกซ์เวอร์ชั่นหนึ่งไปยังอีกเวอร์ชั่นได้ง่าย สามารถใช้ร่วมกับ OS อื่นๆแทน RTAI ได้

2.5.3.3 Interrupt Dispatcher

เป็นตัวแฮนเดิลที่ถูก call ผ่าน IDT เมื่อ ISA, SMP และอินเตอร์รัพท์อื่นๆ เกิดขึ้น ซึ่งถูกกระตุ้นการทำงานได้จาก RTAI หรือทั้ง RTAI และลีนุกซ์โดยเริ่มจาก RTAI แล้วตามด้วยลีนุกซ์ ตำแหน่งของตัวลีนุกซ์แฮนเดิลถูกจัดเก็บก่อนที่มีการเปลี่ยน IDT ใน HAL ตัว dispatcher ทำการแฮนเดิลอินเตอร์รัพท์คอนโทรลเลอร์และจัดการกับการอินเตอร์รัพท์และเซอร์วิสที่ร้องขอเข้ามา

หลักการทำงานของ Interrupt Dispatcher



รูปที่ 2-6โครงสร้างการทำงานของ Interrupt Dispatcher

2.6 uClinux

แท้ที่จริงลีนุกซ์เป็นระบบปฏิบัติการที่ถูกพัฒนาให้ใช้งานบนเดสก์ทอปและเซิร์ฟเวอร์โดยเฉพาะ ทำงานร่วมกับซีพียูบนสถาปัตยกรรม x86 แต่ด้วยข้อดีของลีนุกซ์ที่เปิดเผยโค้ดเคอร์เนล รวมทั้งความทนทานและความยืดหยุ่นในการนำไปใช้งาน จึงได้พัฒนาให้ทำงานร่วมกับแอพพลิเคชั่นแบบ embedded ซึ่งลักษณะของแอพพลิเคชั่นแบบ embedded บางตัวไม่มีส่วนจัดการหน่วยความจำ หรือ MMU แต่เนื่องจากมาตรฐานของลีนุกซ์เคอร์เนลจะสนับสนุนแอพพลิเคชั่นที่ต้องการ MMU ดังนั้นเพื่อให้ลีนุกซ์สามารถใช้งานร่วมกับแอพพลิเคชั่นแบบ embedded ที่ไม่มี MMU จำเป็นต้องมีการจัดเตรียมวิธีการพอร์ตลีนุกซ์ไปกับอุปกรณ์เหล่านั้น

แต่ก่อนระบบปฏิบัติการที่รองรับระบบไมโครคอนโทรลเลอร์เหล่านี้จะเป็นพวก RTOS ซึ่งจัดสร้างจากงานพื้นฐานของเคอร์เนลแล้วเพิ่มเติมส่วนต่างๆเข้าไป เช่น TCP/IP Stack หรือ สภาพแวดล้อมแบบ POSIX เป็นต้น โดยการทำเป็นโมดูล ลักษณะของ RTOS เหล่านี้นั้นได้มีการสืบทอดมาสู่ลีนุกซ์และได้นำไปทดสอบการทำงานในหลายๆ แพลตฟอร์ม

ลีนุกซ์ที่ใช้ทำงานร่วมกับแอพพลิเคชั่นแบบ embedded คือ ไมโครคอนโทรลเลอร์ ลีนุกซ์ หรือ uClinux โดยทำการเอาส่วนที่ต้องการ MMU ออกจากลีนุกซ์แล้วแทนที่ด้วยโมเดลหน่วยความจำแบบแฟลช ซึ่งสถาปัตยกรรมใหม่ที่เกิดขึ้นยังคงความสามารถของเคอร์เนลไว้อยู่และเปิดเผยโค้ด

uClinux กำเนิดมาจากลีนุกซ์เคอร์เนลเวอร์ชั่น 2.0 ทำการพัฒนาโดย D. Jeff Dionne และ Kenneth Albanowski มีจุดมุ่งหมายเพื่อใช้งานสำหรับโมโครคอนโทรลเลอร์ที่ไม่มีส่วนการจัดการหน่วย

ความจำ (MMU: Memory Management Units) โดยมุ่งเน้นไปที่ระบบ embedded เนื่องด้วย โปรเซสเซอร์ที่ไม่มี MMU จะราคาถูก จึงเหมาะที่จะนำไปใช้กับระบบ embedded

uClinux มีลักษณะการทำงานแบบหลายงานพร้อมกัน (multitasking) สามารถนำแอพพลิเคชั่นมารันบน uClinux โดยไม่จำเป็นต้องเป็นแบบหลายงานพร้อมกันได้

เคอร์เนลของ uClinux เป็นฉบับย่อของเคอร์เนลเวอร์ชั่น 2.0 ซึ่งยังคงข้อดีหลักของลีนุกซ์ไว้ ได้แก่ ความมีเสถียรภาพ (stability), ความสามารถในการติดต่อกับเน็ตเวิร์ก, สนับสนุน File Systems และมาตรฐาน API รวมทั้งขนาดของโค้ดจะมีขนาดเล็กกว่าลีนุกซ์

2.6.1 คุณลักษณะเด่นของ uClinux

1. สนับสนุนมาตรฐาน API
2. uCkernel < 512 KB
3. uCkernel + tools < 900 KB
4. จัดเตรียม TCP/IP stack นั่นคือ uClinuxคือ ระบบปฏิบัติการแบบ embedded ที่ใช้งานอินเทอร์เน็ตได้
5. สนับสนุนโปรโตคอลที่ใช้เชื่อมต่อเน็ตเวิร์ก
6. สนับสนุน File System ประกอบด้วย NFS, ext2, ROMfs และ JFFS (Journaling FLASH File System), MS-DOS, และ FAT16/32
7. มีตัวจัดการงานแบบ preemptive
8. การทำงานแบบหลายงานพร้อมกัน
9. สนับสนุนมาตรฐาน POSIX 4 คือเพิ่มความสามารถในด้านเรียลไทม์เข้าไป
10. สามารถรันแอพพลิเคชั่นได้ทั้งแบบ Single และ Multithread
11. รันภายใต้โปรเซสเซอร์หลายตระกูล ได้แก่ Motorola DragonBall (M68EZ328), M68328, M68EN322, ColdFire, QUICC (Quad Integrated Communications Controller), ARM7TDMI, MC68EN302, Axis ETRAX, Intel i960, PRISMA, Atari 68k

2.6.2 ข้อจำกัดของuClinux

uClinux ใช้กับโปรเซสเซอร์ที่ไม่มี MMU ทำให้มีข้อจำกัดดังนี้

1. uClinux ไม่สามารถใช้งาน fork() ได้ โดยทำการใช้งาน vfork() แทน ซึ่งไม่ได้หมายความว่าไม่สามารถทำงานหลายงานพร้อมกันได้ แต่หมายความว่า โปรเซสแม่ทำการคอยจนกว่าโปรเซสลูก ทำการ exec() หรือ exit() ซึ่งยังคงใช้งานแบบทำงานพร้อมกันได้อย่างเต็มที่
2. uClinux ไม่สามารถขยายขนาดสแตกได้ และไม่มี brk() (การขยายขนาดของหน่วยความจำที่ทำการร้องขอ) โดยใช้ mmap() แทนเพื่อจองหน่วยความจำ
3. ไม่มีการปกป้องหน่วยความจำ โปรแกรมต่างๆสามารถเกิดความเสียหายรวมทั้งเคอร์เนล
4. สถาปัตยกรรมที่ใช้มีขนาดของโค้ดที่แตกต่างกัน ขึ้นอยู่กับรูปแบบการนำไปใช้งาน

2.6.3 ข้อดีของ uClinux

1. ขนาดเล็ก
2. เปิดเผยโค้ด
3. เหมาะในการนำไปใช้กับโปรเซสเซอร์ที่ไม่มี MMU โดยเฉพาะระบบ embedded
4. เหมาะกับระบบที่ราคาถูก และใช้พลังงานต่ำ โดยเฉพาะระบบ embedded

2.7 แนวทางการพอร์ต uClinux สำหรับฮาร์ดแวร์ใหม่

เนื้อหาในส่วนนี้อธิบายเกี่ยวกับการจัดเตรียมระบบปฏิบัติการได้แก่ uClinux เพื่อสามารถใช้งานร่วมกับสถาปัตยกรรมใหม่ที่ออกแบบขึ้นได้ โดยเน้นไปที่การใช้งานร่วมกับระบบ embedded

2.7.1 Cross-Development Tools

ขั้นตอนแรกในการจัดเตรียมให้สามารถสนับสนุนสถาปัตกรรมใหม่ที่สร้างขึ้นมา ต้องจัดหาเครื่องมือในการสร้างโค้ดซึ่งเครื่องมือเหล่านี้ต้องประกอบด้วยส่วนประกอบพื้นฐานในการพัฒนา โดยทำการเลือก GNU (http://www.gnu.org) เป็นเครื่องมือในการสร้างโค้ด เนื่องจากมีความยืดหยุ่นเพียงพอในการที่จะพบความต้องการของระบบ

ในการสร้างโค้ดส่วนของเคอร์เนลมีความแตกต่างจากโค้ดของยูสเซอร์ คือว่า โค้ดเคอร์เนลถูกเชื่อมโยงไปยังตำแหน่งที่เจาะจงเป็นพิเศษซึ่งอยู่ในหน่วยความจำจริงๆ ดังนั้นการอ้างอิงถึงดาต้าและโปรแกรมทั้งหมดต้องเป็นแบบ absolute หรือแบบ relative ทำให้คอมไพเลอร์สามารถจัดการกับโค้ดได้อย่างอิสระ ในส่วนของยูสเซอร์โปรแกรมจะถูกโหลดมาจากไฟล์ซิสเต็มซึ่งเกี่ยวข้องกับการกำหนดตำแหน่งที่หน่วยความจำโดยตรง และต้องสนับสนุนตำแหน่งใหม่ที่กำหนดขึ้นด้วย มิฉะนั้นจะไม่สามารถใช้งานยูสเซอร์โปรแกรมได้

ส่วนประกอบหลักที่ uClinux ต้องการใช้ในการพัฒนาได้แก่ binutils (ประกอบด้วยแอสเซมเบลอร์และลิงค์เกอร์), GCC เป็นการคอมไพล์เลอร์, GDB เป็นตัวดีบักเกอร์ และยูทิลิตี้เพิ่มเติม เครื่องมือของ GNU ใช้ autoconf เพื่อทำการสร้างสคริปต์ของการคอนฟิก ซึ่งค่าตัวเลือกกลางจะระบุในไดเรกทอรี่ของการติดตั้ง, สถาปัตยกรรมหลัก และสถาปัตยกรรมของโฮสต์ (/usr/local/armtools_glibc, arm-uclinux และ i386-pc-linux ตามลำดับ) สามารถแสดงการกำหนดค่าต่างๆ ดังรูปที่ 2-7

```
#!/bin/sh
export CC="gcc"
/bin/sh ../arm_uclinux_gcc/configure <\\>
-prefix ="/usr/local/armtools_glibc: -build=i386-pc-linux <\\>
-host=i386-pc-linux -target=arm-uclinux -enable-languages=
```

รูปที่ 2-7 ส่วนของเชลล์สคริปต์ที่ใช้กำหนดค่าต่างๆสำหรับคอนฟิก GNU

2.7.1.1 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 2-8

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

รูปที่ 2-8 การเพิ่มเติมส่วนของไฟล์ config.bfd เพื่อทำการระบุรูปแบบไบนารี่เป้าหมาย

2.7.1.2 คอมไพเลอร์ C

ตัวคอมไพเลอร์ที่ใช้คือ GCC (GNU Compiler Collection) ต้องทำการคอนฟิกตัวเลือกสำหรับกำหนดเป็นดีฟอลต์และวิธีการรวมไฟล์เพื่อให้สามารถทำงานได้ โดย GNU จะมีเอกสารที่ให้ข้อมูลเกี่ยวกับการจัดเตรียมวิธีการคอนฟิก GCC เพื่อสามารถใช้งานร่วมกับชุดคำสั่งของโปรเซสเซอร์ใหม่ได้ สำหรับการสวิตช์คอมไพเลอร์สามารถกำหนดไว้ในไพล์ <target.h> โดยใช้มาโคร TARGET_DEFAULT สำหรับกำหนดระบบที่เฉพาะเจาะจงลงไป แสดงได้ดังรูปที่ 2-9 และรูปที่ 2-10

```
#undef TARGET_DEFAULT
#define TARGET_DEFAULT (ARM_FLAG_APCS_32 | ARM_FLAG_NO_GOT)
```

รูปที่ 2-9 แสดงส่วนที่คัดลอกมาจาก uclinux-arm.h ซึ่งระบุดีฟอลต์ของออปชั่นต่างๆ

```
MULTILIB_OPTIONS = mno-got
MULTILIB_DIRNAMES = pic
EXTRA_MULTILIB_PARTS = crtbegin.o crtend.o
```

รูปที่ 2-10 การเพิ่มแฟลกเมนต์ (fragment) t-arm-uclinuxl สำหรับไฟล์ makefile เพื่อสร้าง crtbrgin.o และ crtend.o

2.7.1.3 ดีบักเกอร์

สำหรับตัวดีบักเกอร์ที่ใช้ในการพัฒนาซอฟต์แวร์ ได้แก่ GNU ดีบักเกอร์ (GDB) ซึ่งตัวโปรเซสเซอร์บางตัวสนับสนุนวิธีการดีบักที่สามารถทำการอินเตอร์เฟสผ่านโมดูลของโปรเซสเซอร์ ซึ่งโมดูลเหล่านี้ได้แก่ โมโตโรล่า BDM และ อาร์ม EmdeddedICE ถ้าฮาร์ดแวร์ไม่ได้เตรียมการสนับสนุนในส่วนนี้ จำเป็นต้องทำการแพตช์ (patch) kGDB ที่เป็นโมดูลของเคอร์เนล เพื่อให้สามารถสนับสนุนตัวรีโมทดีบักเกอร์ ปัจจุบันเคอร์เนลเวอร์ชั่น 2.2 และ 2.3 ประกอบด้วยโมดูลนี้อยู่แล้ว

2.7.1.4 Object Format Utilities

เนื่องจากว่า เครื่องมือของ GNU ไม่สนับสนุนรูปแบบ แฟลทไบนารี่ (Flat Binary) ของยูสเซอร์โปรแกรม จึงจำเป็นต้องเพิ่มส่วนไบนารี่ยูทิลิตี้ เพื่อใช้สร้างแฟลทไบนารี่ได้แก่ Coff2flt และ elf2flt ซึ่งเป็นยูทิลิตี้ที่ใช้ กับ uClinux เพื่อทำการแปลงไบนารี่ไปเป็นรูปแบบที่สามารถใช้งานได้โดยตัวโหลดเคอร์เนล (kernel loader) นั่นคือ fs/binfmt_flat.c

2.7.2 การปรับปรุงแก้ไขเคอร์เนล

ขณะที่ส่วนประกอบหลักของโค้ดลีนุกซ์/uClinux ก็คือ ส่วนที่เป็นอิสระจากโปรเซสเซอร์และสถาปัตยกรรม แต่โค้ดระดับล่างสุดของโค้ดจะมีลักษณะเฉพาะตัวของแต่ละระบบ ลักษณะที่ต้องมีเหมือนกันในสถาปัตยกรรมต่างๆได้แก่ มีตัวอินเตอร์รัพท์แฮนเดิลเป็นของตัวเอง, การบำรุงรักษาเมมโมรี่แมป, โครงสร้างของทาส์กและโปรเซสต่างๆ ซึ่งรูทีนเหล่านี้ถูกเก็บไว้ใน arch/{architecture} ถ้าต้องการให้สถาปัตยกรรมต่างๆ สนับสนุนลีนุกซ์ รูทีนในระดับล่างที่ใช้สำหรับสถาปัตยกรรมใหม่ต้องสามารถโมเดลรูปแบบของสถาปัตยกรรมที่เหมือนๆ กันได้ เพื่อให้การเพิ่มรูทีนในระดับล่างเป็นไปได้ง่ายจำเป็นต้องมีพอร์ตอนุกรมช่วยในการดีบักเมสเซจ

ในระบบของ uClinux ส่วนใหญ่ถูกพอร์ต ให้มีอุปกรณ์คอนโซลแบบอนุกรม (Serial Console Device) อยู่แล้ว แต่ต้องรีจิสเตอร์ไปยังคอนโซล เมื่อตอนที่ไดรเวอร์ถูกโหลดเข้ามาจะมีการจัดเตรียมช่องทางสำหรับ printk สำหรับการดีบักเมสเซจ

2.7.2.1 Initialization

มีอยู่ 2 ขั้นใน Initialization นั้นคือ 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 ตัวพารามิเตอร์สำหรับการเซ็ตอัพจะประกอบไปด้วยตำแหน่งเริ่มต้นและสิ้นสุด ของหน่วยความจำ ส่วนของคำสั่งที่เพิ่มเติมขึ้นมาซึ่งส่งผ่านมาจากตัวโหลดเดอร์อีกที และส่วนของงานหลักสำหรับการเอ็กซิคิวต์เคอร์เนลเมื่อทำการบูตเคอร์เนลขึ้นมา

2.7.2.2 Scheduling/Tasking

ลักษณะของการทำงานพร้อมกันภายใต้ uClinux มีลักษณะง่ายกว่า ทำงานภายใต้ลีนุกซ์ เนื่องจากไม่มีเพจเทเบิล และการป้องกันการแฮนเดิล โดยการทำงานจริงที่เกิดขึ้นต้องแสดงคอนเท็กซ์ ที่ถูกต้องทั้งส่วน เก็บค่าและคืนค่า ซึ่งคอนเท็กซ์จะเก็บค่ารีจิสเตอร์ทั้งหมดไว้และนำมาใช้เมื่อโปรเซสถูกอินเตอร์รัพท์

ตัวคอนเท็กซ์รีจิสเตอร์ถูกกำหนดไว้ในโครงสร้าง rt_regs จะอยู่ใน asm/ptrace.h ถ้าหากใช้รีจิสเตอร์ที่มีการรีเทิร์นค่าคืนกลับมา จำเป็นต้องมีการเก็บค่าเดิมไว้ (จากตัวอย่างจะเป็น ARM_ORIG_r0) เมื่อระบบมีการรีเทิร์นจาก system call ต้องใส่ค่ารีจิสเตอร์ที่รีเทิร์นกลับมาให้กับตัวรีจิสเตอร์ด้วย แต่ถ้าระบบถูกรีเทิร์นจากการอินเตอร์รัพท์ ตัวรีจิสเตอร์ก็ยังคงเก็บค่าเดิมไว้

```
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 2-11 แสดงรีจิสเตอร์ ptrace เพื่อใช้สนับสนุนการเก็บค่าและคืนค่าของคอนเท็กซ์

2.7.2.3 Memory Accounting

บทบาทของหน่วยความจำเสมือนจะเกี่ยวข้องกับการใช้ประโยชน์จากทรัพยากรของระบบ โดยใช้รูทีน paging_init เพื่อจัดสรรพื้นที่ว่างในหน่วยความจำ ถ้าหากระบบถูกคอนฟิกสำหรับใช้เมมโมรี่แมป รูทีนนี้จะถูกเพิ่มไปในแต่ละเซ็กชั่นของหน่วยความจำโดยไม่ขึ้นอยู่กับแมปที่ใช้

2.7.2.4 อินเตอร์รัพท์

ในการจัดการเกี่ยวกับอินเตอร์รัพท์ระบบต้องจัดเตรียม entry point เหล่านี้ได้แก่ enable_irq, disable_irq, get_irq_list, request_irq, free_irq, probe_irq_on, probe_irq_off และ init_IRQ ซึ่งเคอร์เนลและดีไวซ์ไดรเวอร์ทำการเรียกใช้งานรูทีนเหล่านี้สำหรับควบคุมส่วนหลักๆ ของระบบ

2.7.2.5 ไทม์เมอร์

ฟังก์ชั่นหลักๆ ของระบบต้องการส่วนจัดการเกี่ยวกับไทม์เมอร์แบบ periodic อินเตอร์รัพท์ โดยใช้รูทีน time_init เพื่อกำหนดค่าเริ่มต้นให้กับเวลาและอินเตอร์รัพท์ และทำการเรียกฟังก์ชั่น do_timer ซึ่งจะให้ค่าของเวลาเป็น Hz เพื่อทำการรอการอินเตอร์รัพท์ที่เข้ามา

2.7.2.6 เคอร์เนลไลบรารี่ฟังก์ชั่น (Kernel Library Functions)

ลีนุกซ์ใช้ไลบรารี่ที่สนับสนุนฟังก์ชั่นของซี เพื่อให้ได้ประสิทธิภาพที่ดีที่สุด

2.7.3 ดีไวซ์ไดรเวอร์ (Device Driver)

โปรเซสเซอร์จำเป็นต้องสนับสนุนดีไวซ์ไดรเวอร์ เพราะถึงแม้ว่าเคอร์เนลจะทำงานได้ แต่จะไม่สามารถทำอะไรได้เลยถ้าหากไม่สนับสนุนดีไวซ์ต่างๆ ซึ่งอย่างน้อยโปรเซสเซอร์ต้องสนับสนุนดีไวซ์เหล่านี้ ได้แก่ คอนโซล, ซีเรียลดีไวซ์และ บล็อกดีไวซ์ ที่บรรจุรูตไฟล์ซิสเต็มไว้ด้วย ไดรเวอร์เหล่านี้ไม่ถูกใช้ในยูสเซอร์โปรเซส โดยตรง แต่ต้องใช้งานผ่านเคอร์เนลโมดูล เช่นไดรเวอร์ tty หรือไฟล์ซิสเต็ม

2.7.3.1 คอนโซล (Console)

ขั้นแรกในการจัดเตรียมซีเรียลไดรเวอร์ ต้องเตรียมเอาต์พุตของการดีบักคอนโซลก่อน ซึ่งการดีบักคอนโซลสามารถรวมไว้กับช่วงที่ทำการบูตโดยเรียกผ่าน register_console ควรจะเรียกผ่านฟังก์ชั่น console_init ที่อินาเบิลอยู่และตั้งค่าของ bit rate ให้มีค่าคงที่ตายตัว

2.7.3.2 ซีเรียล

ซีเรียลดีไวซ์ ถูกออกแบบมาเพื่อใช้แฮนเดิลอินพุตและเอาต์พุตของไดรเวอร์ tty เนื่องจากยูสเซอร์โปรเซสที่ต้องการติดต่อไปยัง ไดรเวอร์ tty จะไม่สามารถติดต่อโดยตรงไปยังซีเรียลดีไวซ์ได้ ซึ่งจริงๆแล้วซีเรียลดีไวซ์ ถูกเรียกจาก ดีไวซ์ tty สามารถดูแบบจำลองของไดรเวอร์ที่ serial.c

2.7.3.3 ไฟล์ซิสเต็ม บล็อกดีไวซ์ (Filesystem Block Device)

ทางเลือกที่ดีสำหรับเริ่มต้นการทำงานของระบบคือการรวม บล็อกเมมโมรี่ดีไวซ์ไดรเวอร์ (block memory device driver) (ใช้งานผ่าน blkmem.c) และ แรมดิสก์ ไดรเวอร์ (ramdisk driver) เข้าด้วยกัน แรมดิสก์ ไดรเวอร์ถูกออกแบบมาเพื่อทำการจองพื้นที่ว่างและอ่านข้อมูลของมันจากบล็อกดีไวซ์ ซึ่งข้อมูลสามารถอยู่ในรูปแบบเดิมหรือถูกบีบอัดได้ โดยแรมดิสก์ ไดรเวอร์ ทำการอ่านบล็อกแรกเพื่อกำหนดรูปแบบของข้อมูลที่ใช้แสดง

2.7.4 รันไทม์ไลบรารี่ (Runtime Library)

รันไทม์ไลบรารี่ใช้เตรียมการอินเทอร์เฟซจากยูสเซอร์โปรแกรมไปยังรูทีนของเคอร์เนล คนส่วนใหญ่คิดว่าระบบเหล่านี้คือระบบลีนุกซ์ซึ่งหมายถึงการรันเคอร์เนล แต่ที่จริงระบบไม่ได้ถูกกำหนดโดยเคอร์เนลแต่กำหนดด้วยรันไทม์ไลบรารี่ของซี เช่นใช้เป็นรันไทม์ไลบรารี่ที่ทำการเคลื่อนย้ายการ call

printk ไปไว้ในดาต้าบัฟเฟอร์ และทำการเขียนเพื่อส่งเอาต์พุตไปยัง ไฟล์เดสคริปเตอร์ 1 (stdout) ซึ่งระบบการเรียกไลบรารี่ทำการปรับปรุงเพื่อเพิ่มความสามารถของระบบ โดยแพ็กเกจของ uClinux มีไลบรารี่เพื่อใช้งานคำสั่งพื้นฐานของ ซี ซึ่งส่วนใหญ่ใช้ GNU glibc หรือ newlib

ดังนั้น เพื่อให้สามารถใช้งานร่วมกับสถาปัตยกรรมใหม่ที่สร้างขึ้นมาได้ ต้องประกอบด้วยโค้ดโปรแกรมที่สร้างขึ้นมา, ข้อกำหนดต่างๆ ของ system call API, การปรับปรุงแก้ไขรูทีนของ I/O, และการเปลี่ยนแปลงสถาปัตยกรรมของแฟลชเมมโมรี่ ซึ่งแสดงรายละเอียดในแต่ละส่วนต่อไป

2.7.4.1 Program Entry Code

ฟังก์ชั่นหลักของยูสเซอร์โปรแกรมเริ่มด้วย main โดย entry จริงๆ จะชี้ไปยัง entry ที่อยู่ในไลบรารี่ของซี โค้ดนี้เริ่มด้วยสแตติกดาต้าเช่น ค่า error (errno) และพอยเตอร์ที่เกี่ยวข้อง (environ) จากนั้นทำการส่งอาร์กิวเมนต์ argc และ argv ไปยัง main โดยอาร์กิวเมนต์และส่วนที่เกี่ยวข้องกับโปรแกรมถูกเรียกผ่านรูทีน start_thread (อยู่ใน proc/processor.h) ในส่วน execve system call จะใช้ start_thread เพื่อเซ็ตอัพ ค่าผิดพลาดของคอนเท็กซ์ที่คืนค่ากลับมา ในส่วนของ program entry code เป็นส่วนประกอบการทำงานของ start_thread แสดงดังในรูปที่ 2-12

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

        .data
        .align 2
        .global environ,errno
environ: .long 0                    @allocate statice space for
environ 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,r3

        ldr   r3,.l3                    @load the address of eviron
        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
        mov pc,lr


.L4:    .word errno       @address of label 'errno'
```

รูปที่ 2-12 แสดง Program entry code ที่ทำการแฮนเดิล environ ซึ่งส่งมาจาก sys_execve kernel call

2.7.4.2 System Calls

ยูสเซอร์โปรแกรมทำการเข้าถึงทรัพยากรของระบบโดยผ่าน system calls เนื่องจากโปรเซสเหล่านี้เกี่ยวข้องกับการสวิตช์จาก ยูสเซอร์คอนเท็กซ์ ไปยัง ซิสเต็มคอนเท็กซ์ โดยโค้ดของ system call มีการระบุโปรเซสเซอร์ที่เฉพาะเจาะจงลงไป โปรเซสของระบบที่เก็บยูสเซอร์คอนเท็กซ์ จะบรรจุพารามิเตอร์ของการ call ไว้ด้วย ยูสเซอร์โปรเซสถูกพักไว้จนกว่าเคอร์เนลทำ ทรานเซ็กชั่นเสร็จและคืนค่าเธรดไปยังฝั่งผู้ใช้ ตัวอย่า งของ system call อาร์กิวเมนต์ถูกเก็บไว้ในตัวแปร และเธรดของการเอ็กซิคิวต์ทำให้หยุดด้วย คำสั่งการอินเตอร์รัพท์ของซอฟต์แวร์ (SWI) เคอร์เนลทำการเก็บค่าของคอนเท็กซ์และแสดงการ call ด้วยโค้ด เช่น 0x900004 เป็นส่วนการ call sys_write เป็นต้น เมื่อเคอร์เนลทำการ call เสร็จสิ้น จะคืนค่ายูสเซอร์คอนเท็กซ์ที่ใช้ในการเอ็กซิคิวต์ โดยรีเทิร์นค่า r0

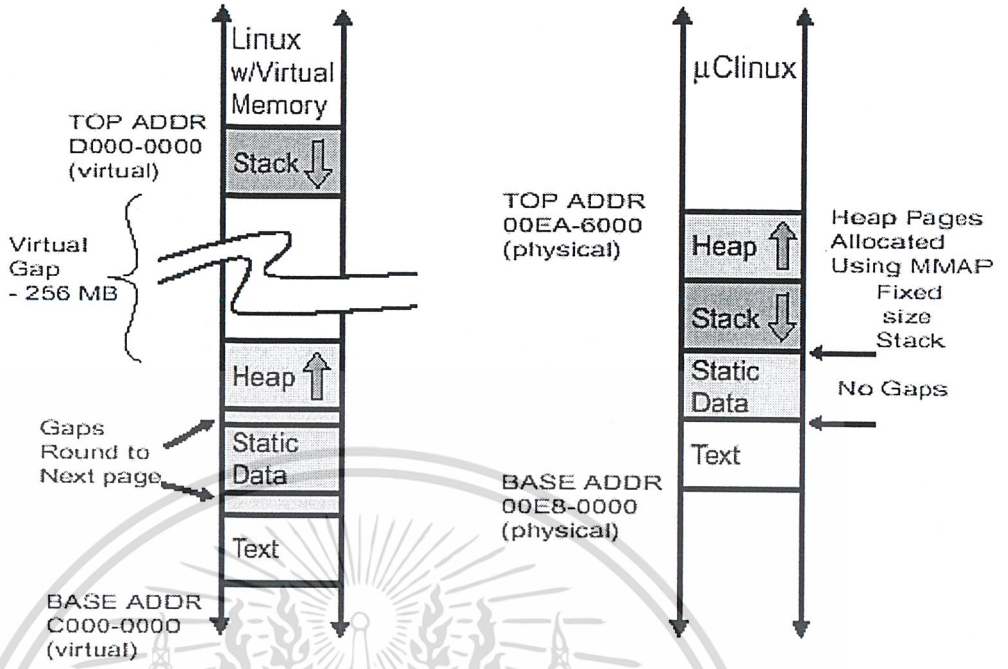
2.7.4.3 Basic IO Routines

uClinux สามารถใช้งานได้กับสถาปัตยกรรมที่มีทรัพากรที่จำกัดและไม่สนับสนุนการแชร์ไลบรารี่ แต่ต้องใช้งานรูทีนของไลบรารี่ ได้เป็นอย่างดี รวมทั้งใช้บัฟเฟอร์เป็นสองเท่าเพื่อเพิ่มประสิทธิภาพให้สูงขึ้น

2.7.4.4 Heap Allocation

การกำหนดหน่วยความจำแบบไดนามิกจากยูสเซอร์โปรแกรมสามารถกำหนดผ่านการเรียก malloc เพื่อกำหนดพื้นที่ในการจัดเก็บโปรแกรม และเรียกใช้ sbrk เพื่อขยายขนาดข้อมูลของโปรแกรม และจากนั้นแบ่งหน่วยความจำที่ถูกกำหนดไว้เป็นส่วนๆ เนื่องจากหน่วยความจำเสมือนสามารถขยายขนาดของพื้นที่ว่างระหว่างส่วนบนของฮีพ (heap) และส่วนล่างของสแตกได้ แต่ในโมเดลของแฟลซ จะไม่มีพื้นที่ว่างให้กับเพจเสมือน ผลที่ตามมาทำให้ heap storage ถูกกำหนดด้วยการ call mmap แทน

การใช้ uC-libc ในการกำหนดพื้นที่ว่างทำให้ง่ายในการแฮนเดิลเพจ เนื่องจากมีไลบรารี่มากมาย เช่น glibc ซึ่งจัดเตรียมเธรดสำหรับบัฟเฟอร์ แต่ถ้าใช้ sbrk ในการกำหนดก็สามารถใช้ mmap แทนได้เช่นกัน แสดงได้ดังรูปที่ 2-13



รูปที่ 2-13 ไดอะแกรมของ วิชวลโปรแกรมและแฟลซโปรแกรม

2.7.4.5 Stack Sizing

พิจารณารูป จะเห็นว่าสแตกจะอยู่ติดกับสแตติกดาต้าดังนั้นต้องมีขนาดเพียงพอที่จะไม่ไปทับสแตติกดาต้าและพื้นที่ของโค้ด ซึ่งพื้นที่ว่างในลีนุกซ์โปรแกรมอนุญาตให้พื้นที่ของ heap และ สแตกขยายใหญ่ได้อย่างไม่จำกัดโดยระบบสามารถตรวจสอบเพจเพื่อป้องกันไม่ให้มีพื้นที่ว่างมากเกินไป

การใช้พื้นที่ว่างใน uClinux ระบบต้องทำการกำหนดหน่วยความจำจริงให้มากกว่าหน่วยความจำเสมือน เนื่องจากว่าไม่มีการป้องกันหน่วยความจำทำให้ตรวจสอบไม่ได้ว่าสแตกมีขนาดเกินไปหรือเปล่า ดังนั้นควรจะกำหนดขนาดของสแตก และใช้ GDB ในการตรวจสอบระหว่างการรันโปรแกรมเพื่อให้แน่ใจว่ามีพื้นที่เพียงพอในการใช้งาน

2.7.4.6 Flat Memory Support

ความแตกต่างที่เห็นได้ชัดระหว่าง uClinux กับลีนุกซ์คือ รูปแบบหน่วยความจำแบบแฟลซซึ่งโปรแกรมทั้งหมดไม่สามารถแชร์หน่วยความจำในเวลาเดียวกันได้ ในลีนุกซ์ใช้หน่วยความจำเสมือนเพื่อจัดเตรียมฟังก์ชั่นพื้นฐานของ system calls จะมี fork ซึ่งสามารถทำการแชร์เพจเดียวกันได้ โดย fork ไม่ได้ถูกนำมาใช้งานใน uClinux จะใช้ vfork แทน ซึ่ง vfork แตกต่างจาก fork ตรงที่โปรเซสลูกไม่สามารถแชร์ข้อมูลและสแตกที่ว่างได้เหมือนกันโปรเซสแม่

ในการนำลีนุกซ์ให้สามารถใช้งานร่วมกับสถาปัตยกรรมที่ไม่มี mmu ที่จริงมีความยืดหยุ่นในการนำลีนุกซ์มาใช้เป็นอย่างมากโดยส่วนใหญ่ใช้กับสถาปัตยกรรมเน็ตเวิร์กแบบ embedded ต้องมีการเปลี่ยน API ของลีนุกซ์ให้มีขนาดเล็กลง และพัฒนาในส่วนเกี่ยวข้องต่างๆ ได้แก่เคอร์เนล, ไลบรารี่และยูสเซอร์

โปรแกรมต้องดัดแปลงให้ใช้ได้กับสถาปัตยกรรมแบบแฟลซเมมโมรี่ ได้ ร่วมทั้งเครื่องมือที่ใช้ในการพัฒนาต้องมีความยืดหยุ่น และใช้งานได้เป็นอย่างดีกับสถาปัตยกรรมที่ออกแบบมา

2.8 บอร์ด AT91EB01

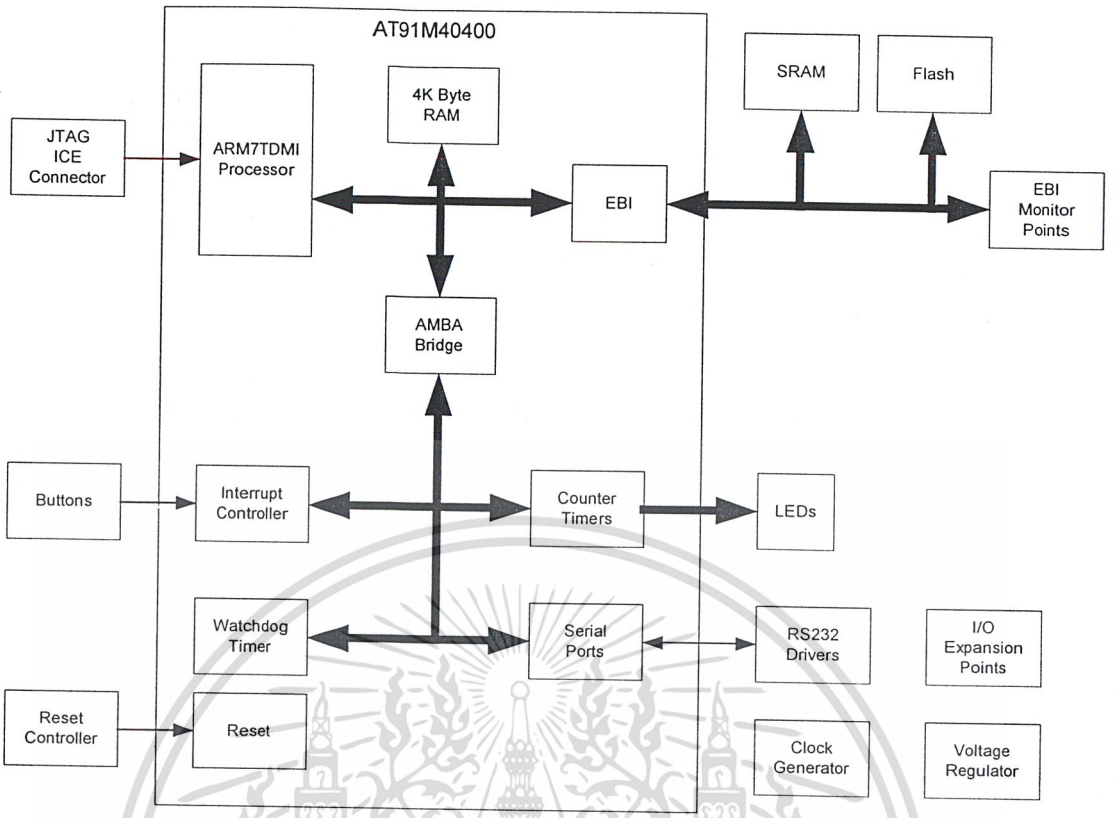
บอร์ด AT91EB01 เป็นบอร์ดที่อยู่ในชุดพัฒนา AT91EB01 Evaluation Kit ของบิรษัท ATMEL ซึ่งเป็นแพลฟอร์มที่มีราคาถูก มีความสามารถในการพัฒนาและประเมินผลโค้ดแบบเรียลไทม์ โดยการพัฒนาบนเครื่องคอมพิวเตอร์ ซึ่งใช้ไมโครคอนโทรลเลอร์ตระกูล AT91M40400

ในการนำบอร์ด AT91EB01 มาเพื่อใช้งานกับลีนุกซ์นั้นจำเป็นต้องมีการเพิ่มหน่วยความจำจากเดิมที่มีมาให้อีกอย่างน้อย 1 เมกาไบต์ ซึ่งทางบริษัท ATMEL ได้เตรียมการ์ดขยายหน่วยความจำมาอีกในชุดพัฒนา AT91MEC01

2.8.1 ส่วนประกอบของบอร์ด AT91EB01

โครงสร้างของบอร์ด AT91EB01 แสดงได้ดังรูปที่ 2-14 มีส่วนประกอบดังนี้

- ไมโครคอนโทรลเลอร์ AT91M40400 โดยมีโปรเซสเซอร์ภายในเป็น ARM7TDMI
- ตัวสร้างสัญญาณนาฬิกา (Clock Generator) ซึ่งสามารถเลือกความเร็วได้
- SRAM 16 บิต ขนาด 512 กิโลไบต์ (สามารถเพิ่มได้ถึง 2 Mb)
- Flash 16 บิต ขนาด 128 ไบต์ (ใช้สำหรับซอฟต์แวร์ประมาณ 64K)
- ช่องเชื่อมต่อ(Connector) แบบขยาย (EBI)
- ช่องเชื่อมต่อสำหรับ I/O
- ซีเรียลพอร์ต 2 พอร์ต
- ปุ่มรีเซ็ต
- ปุ่มใช้งานประกอบด้วย FIQ, TIOB0, IRQ0
- LEDs 3 ตัว ได้แก่ TIOA0, TIOA1, TIOB0
- ช่องเชื่อมต่อ JTAG 20 ขา



รูปที่ 2-14 โครงสร้างของสถาปัตยกรรม AT91EB01

2.8.2 การ์ดขยายหน่วยความจำ AT91MEC01

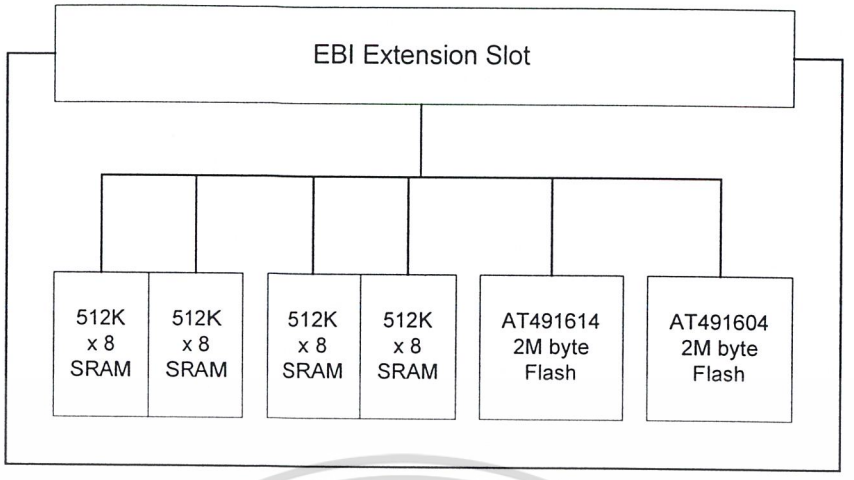
AT91MEC01 เป็นการ์ดขยายหน่วยความจำ สามารถเชื่อมต่อได้ทั้ง บอร์ด AT91EB01 และ AT91EB63 ซึ่ง AT91MEC01 สามารถเพิ่มขนาดของหน่วยความจำได้ถึง 2 เมกาไบต์ ในส่วนของ SRAM และ 4 เมกาไบต์ในส่วนของแฟลซ บนบัสภายนอก ของไมโครคอนโทรลเลอร์ AT91

AT91MEC01 ใช้ช่องเชื่อมต่อกับบอร์ด AT91 และมีดิสก์เกตที่บรรจุชุดพัฒนาซอฟต์แวร์และเอกสารประกอบต่างๆ โดยถ้าผู้ใช้ต้องการใช้การ์ดและซอฟต์แวร์เพียงจัดเตรียมเครื่องคอมพิวเตอร์และระบบการดีบัก

ความต้องการของระบบในการใช้งาน AT91MEC01 ต้องมีเครื่องคอมพิวเตอร์ที่สามารถทำการรันตัวดีบักเกอร์ที่อยู่ในชุดพัฒนาซอฟต์แวร์อาร์เวอร์ชั่น 2.5 ได้

จากรูปที่ 2-15 ส่วนประกอบของ AT91MEC01 AT91MEC01 แบ่งหน่วยความจำเป็น 4 ส่วน ประกอบด้วย

- AT49BV1604-90TC มีแฟลซขนาด 2 เมกาไบต์ โดยออกแบบหน่วยความจำ 2 ตัวที่มีความสามารถในการอ่านและเขียน
- AT49BV1614-90TC มีแฟลซขนาด 2 เมกาไบต์ โดยออกแบบหน่วยความจำ 2 ตัวที่มีความสามารถในการอ่านและเขียน และมี 1 พินสำหรับสัญญาณ Ready/Busy
- SRAM ขนาด 512 กิโลไบต์ จำนวน 4 ตัว รวมเป็น 2 เมกาไบต์ เวลาในการเข้าถึง 15 ns ทำให้ไม่ต้องใช้ wait state



รูปที่ 2-15 ส่วนประกอบของ AT91MEC01

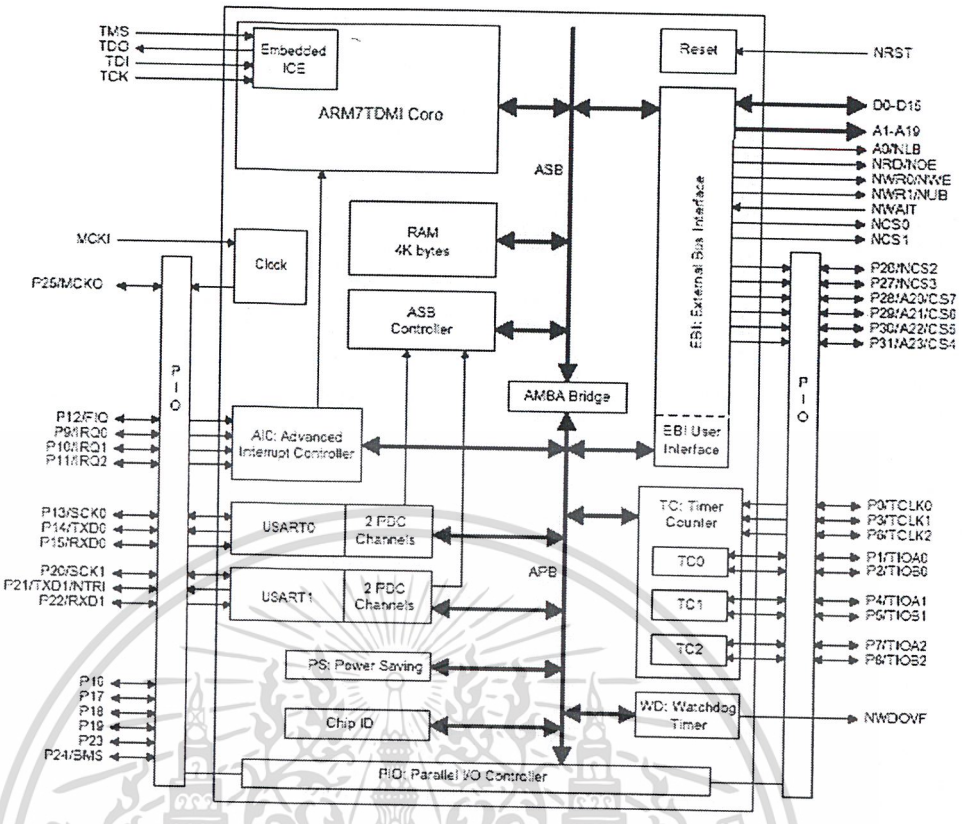
2.9 ไมโครคอนโทรลเลอร์ AT91M40400

ไมโครคอนโทรลเลอร์ AT91M40400 เป็นไมโครคอนโทรลเลอร์ที่ใช้บนบอร์ด AT91EB01 ซึ่งสามารถหาข้อมูลเพิ่มเติมได้จาก [8]

2.9.1 คุณสมบัติของ AT91M40400

AT91M40400 เป็นไมโครคอลโทรลเลอร์ ตระกูล Atmel AT91 16/32 บิต อยู่บนพื้นฐานโปรเซสเซอร์ ARM7TDMI ซึ่งเป็นโปรเซสเซอร์ที่มีประสิทธิภาพสูงด้วยสถาปัตยกรรม RISC 32 บิต ทั้งยังมีชุดคำสั่ง (Instruction Set) ขนาด 16 บิต และสิ้นเปลืองพลังงานน้อยมาก สามารถควบคุมแอพพลิเคชั่นแบบเรียลไทม์

AT91M40400 สามารถทำการติดต่อโดยตรงกับหน่วยความจำด้วยแฟลช โดยผ่านบัสภายนอก(External Bus)



รูปที่ 2-16 สถาปัตยกรรมของ AT91M40400

2.9.2 สถาปัตยกรรมของ AT91M40400

สถาปัตยกรรมของ AT91M40400 ดังรูปที่ 2-16 ประกอบด้วย 2 บัสหลักๆ ได้แก่ ASB (the Advanced System Bus) และ APB (the Advanced Peripheral Bus (APB) โดย ASB ถูกออกแบบสำหรับประสิทธิภาพที่สูงที่สุด ทำการเชื่อมต่อกับโปรเซสเซอร์, หน่วยความจำบนชิปขนาด 32 บิตและหน่วยความจำภายนอกรวมทั้งอุปกรณ์ต่างโดยผ่าน EBI (Extenal Bus Interface) ส่วน APB ถูกออกแบบสำหรับเข้าถึงอุปกรณ์ต่างๆบนชิปและใช้พลังงานน้อย ระหว่าง APB และ ASB ทำการเชื่อมต่อถึงกันโดยใช้ AMBA Bridge

PDC (Peripheral Data Controller) ทำการโอนย้ายข้อมูลระหว่าง USARTs ที่อยู่บนชิปกับหน่วยความจำทั้งภายในและภายนอกชิป ที่สำคัญ PDC จะทำการลดโอเวอร์ฮีด ที่โปรเซสเซอร์ทำการอินเตอร์รัพท์แฮนดิ่งและลดจำนวนของไซเคิลสัญญาณนาฬิกา ที่ต้องการสำหรับการโอนย้ายข้อมูล ซึ่งสามารถทำการโอนย้ายขนาด 64 กิโลไบต์ได้อย่างต่อเนื่อง ทำให้เพิ่มประสิทธิภาพของไมโครคอนโทรลเลอร์และลดการสิ้นเปลืองพลังงานลงได้

อุปกรณ์รอบข้างภายในของ AT91M40400 ถูกออกให้ใช้ชุดคำสั่งในการทำงานน้อยในการทำงานโดยแต่หละตัวจะมีช่วงแอดเดรสขนาด 16 กิโลไบต์ในขอบเขต 3 เมกาไบต์บนของช่วงแอดเดรส 4 จิกะไบต์ เพื่อให้ประสิทธิภาพในการดำเนินงานกับบิต ยังได้มีการแยกรีจิสเตอร์ที่มีการใช้บ่อยเป็น 3

ตำแหน่งและให้ตำแหน่งแรกเป็นการเซตบิต ตำแหน่งที่สองเป็นการเคลียร์บิต และตำแหน่งที่สามเป็นการอ่านค่าของรีจิสเตอร์นั้น

สัญญาณภายนอกทั้งหมดของอุปกรณ์รอบข้างจะถูกควบคุมโดย PIO (Parallel I/O Controller) ซึ่งสามารถโปรแกรมให้ PIO เพิ่มตัวกรองอินพุตได้

โปรเซสเซอร์ ARM7TDMI ที่อยู่ภายในทำงานในโหมดลิตเติ้ลเอนเดี้ยน (little-endian) รายละเอียดของโปรเซสเซอร์นี้อยู่ในหัวข้อถัดไป

2.9.3 ข้อดีของ AT91M40400

- มีความยืดหยุ่น
- ลดต้นทุนในการพัฒนา
- เหมาะสำหรับควบคุมแอพพลิเคชั่นในระบบ embedded

2.10 ไมโครโปรเซสเซอร์ ARM7TDMI

ARM7TDMI เป็นไมโครโปรเซสเซอร์ขนาด 32 บิต ตระกูล ARM (Advanced RISC Machines) มีประสิทธภาพสูง สิ้นเปลืองพลังงานน้อยและราคาถูก

สถาปัตยกรรมARMใช้หลักการพื้นฐานของ RISC (Reduced Instruction Set Computer) โดยมีชุดคำสั่งและวิธีการดีโค้ด (Decode) ที่ง่ายกว่า CISC (Complex Instruction Set Computers) ทำให้คำสั่งเหล่านี้มี Throughput ที่สูง และเหมือนกับว่ามีการตอบรับการอินเตอร์รัพท์ในลักษณะของเรียลไทม์จากชิป ที่มีขนาดเล็กและทำให้คุ้มค่าในการลงทุน

นำหลักการไปป์ไลน์ (Pipeline) มาใช้กับทุกส่วนของโปรเซสและกับระบบหน่วยความจำซึ่งสามารถทำการโอเปอเรชั่น (Operation) ได้อย่างต่อเนื่อง อย่างเช่น เมื่อมีคำสั่งหนึ่งถูกเอ็กซิคิวต์ โปรเซสเซอร์จะเริ่มทำการดีโค้ดและคำสั่งต่อไปจะถูกดึงมาจากหน่วยความจำ

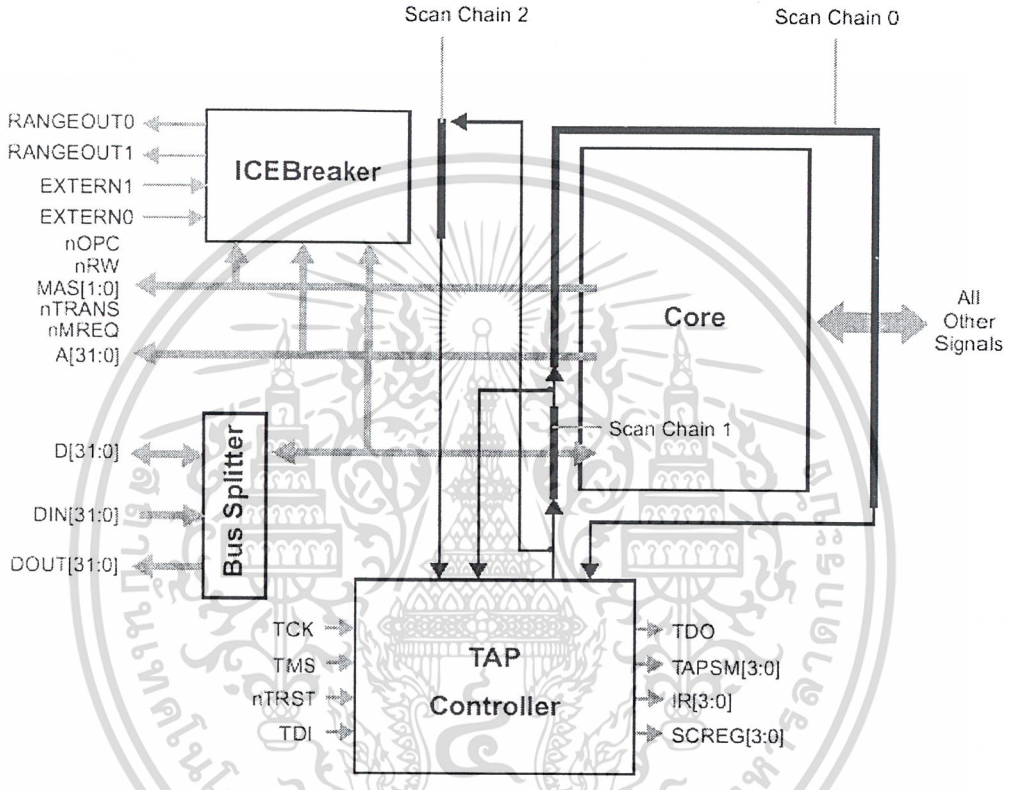
การเชื่อต่อกับหน่วยความจำใน ARM ถูกออกแบบให้ใช้งานได้อย่างมีประสิทธิภาพกับระบบหน่วยความที่ราคาไม่สูง สัญญานควบคุมสำคัญที่ถูกควบคุมโดยไปป์ไลน์นั้นใช้ลอกจิกมาตรฐานแบบพลังงานต่ำและรูปแบบของสัญญาณที่สามารถใช้ได้กับโหมดการเข้าถึงแบบเร็วของหน่วยความจำมาตรฐานในอุตสาหกรรม

สถาปัตยกรรมของ ARM7TDMI จากรูปที่ 2-17 และ มีรายละเอียดดังนี้ซึ่งสามารถหาข้อมูลเพิ่มเติมได้จาก [7]

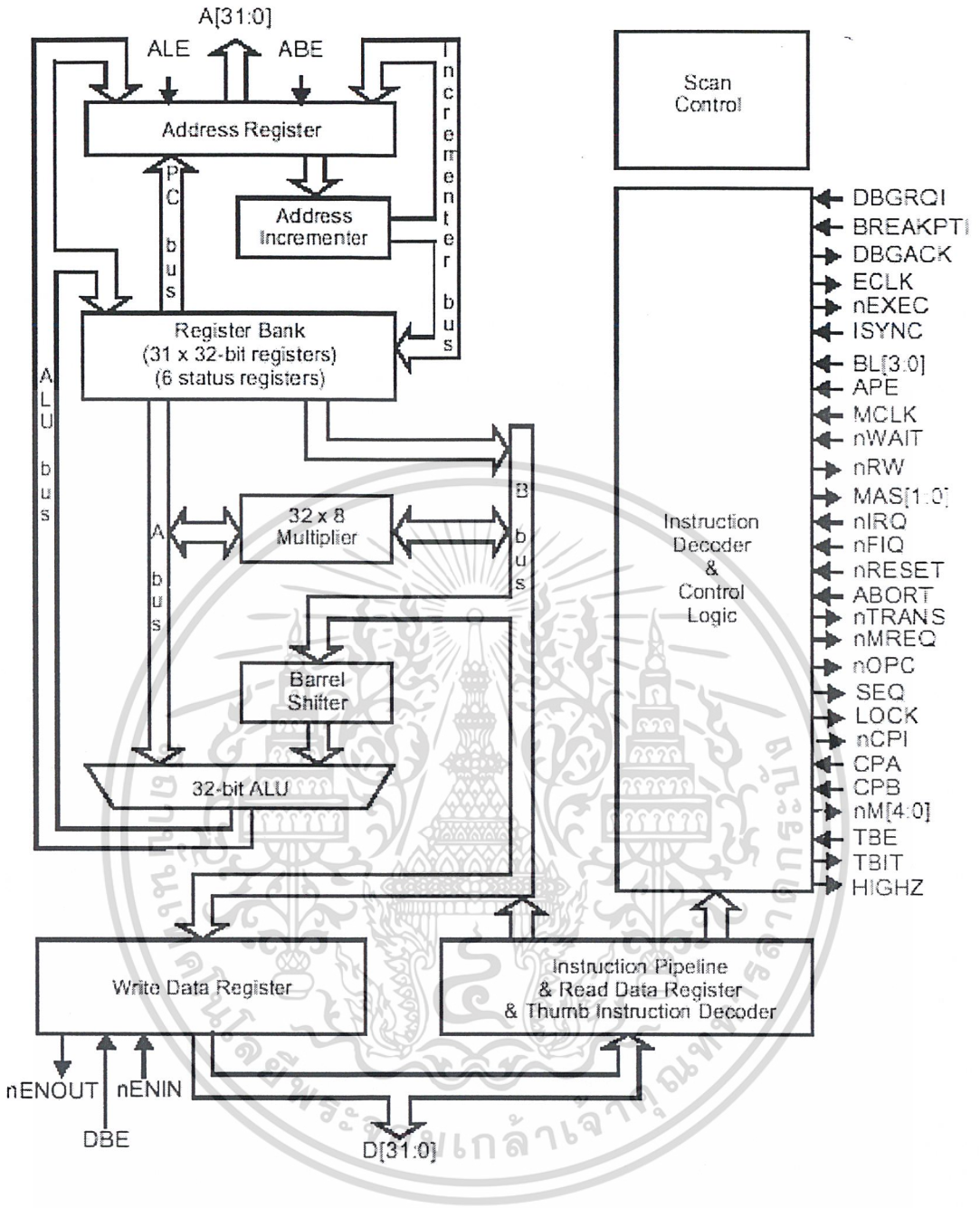
อินเตอร์เฟสการดีบักยึดตามมาตรฐาน IEEE 1149.1-1990 (Standard Test Access Port and Boundary-Scan Architecture) ผ่านทาง TAP Controller

- ดีบักแบบบนชิปโดย ICEBreaker
- ไปป์ไลน์ 3 สถานะ
- โปรเซสเซอร์ RISC 32 บิต เป็นสถาปัตยกรรมแบบ Von Neumann load/store ซึ่งมีบัสสำหรับแอดเดรสและดาต้าสำหรับชุดคำสั่งและข้อมูลเพียงบัสเดียว

- ซีพียูมีชุดคำสั่ง 2 แบบได้แก่ ชุดคำสั่งARM ขนาด 32 บิต และชุดคำสั่ง Thumb ขนาด 16 บิต โดยคำสั่งเหล่านี้สามารถทำโอเปอเรชั่นกับดาต้าขนาด 8, 16 และ 32 บิต
- ซีพียูมีโหมดการทำการ 7 โหมดซึ่งแต่ละโหมดจะมีรีจิสเตอร์เพื่อความรวดเร็วในการแฮนเดิลเอ็กซ์เซปชั่น (exception)
- โปรเซสเซอร์มีรีจิสเตอร์ขนาด 32 บิตจำนวน 37 ตัว รวมถึง 6 รีจิสเตอร์สถานะ



รูปที่ 2-17 โครงสร้างของ ARM7TDMI

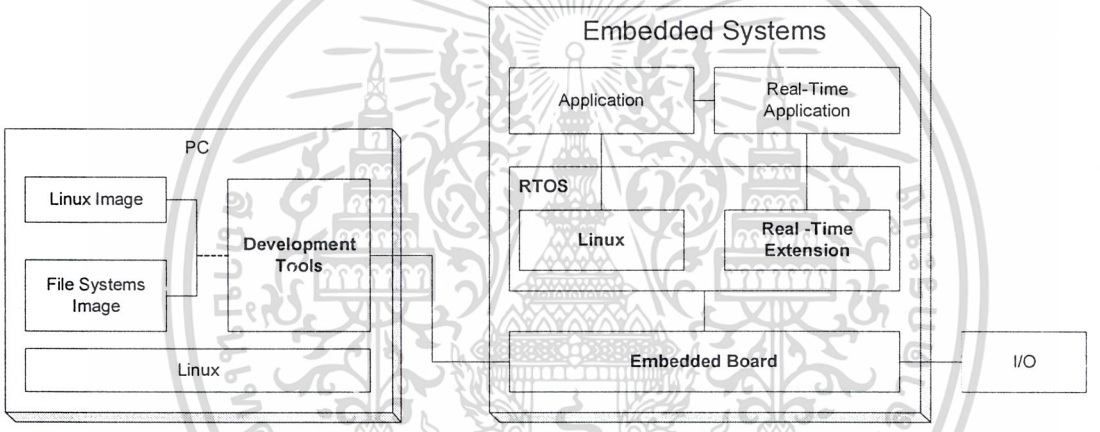


รูปที่ 2-18 โครงสร้างของคอร์ภายใน ARM7TDM

บทที่ 3
การวิเคราะห์และการออกแบบ

3.1 ภาพรวมของโครงงาน

โครงงานระบบลีนุกซ์แบบเรียลไทม์ที่ได้พัฒนาระบบปฏิบัติการลีนุกซ์ให้สามารถทำงานบนระบบ embedded และเพิ่มความสามารถแบบเรียลไทม์ให้กับระบบปฏิบัติการลีนุกซ์ ในระบบลีนุกซ์แบบเรียลไทม์สามารถแบ่งออกได้เป็น 2 ส่วนคือ ส่วนที่ใช้ในการพัฒนาเคอร์เนลและแอพพลิเคชั่นบนเครื่องพีซีเพื่อนำมาใช้งานบนระบบ embedded และ ส่วนที่ทำงานบนระบบ embedded ที่ได้รับการพัฒนาจากส่วนแรก แสดงดังในรูปที่ 3-1



รูปที่ 3-1 ระบบลีนุกซ์แบบเรียลไทม์

จากภาพรวมของโครงการที่แสดงในรูปที่ 3-1 มีส่วนประกอบที่ต้องใช้ในระบบอยู่หลายส่วน โดยเฉพาะในส่วนที่แสดงเป็นแรเงาเป็นส่วนที่จำเป็นต้องมีการวิเคราะห์เพื่อนำมาใช้กับระบบ ซึ่งสามารถแบ่งการวิเคราะห์ได้เป็น 2 ส่วนที่สำคัญ คือ

1. ลีนุกซ์กับระบบเรียลไทม์
2. ระบบ embedded

3.2 ลีนุกซ์กับระบบเรียลไทม์

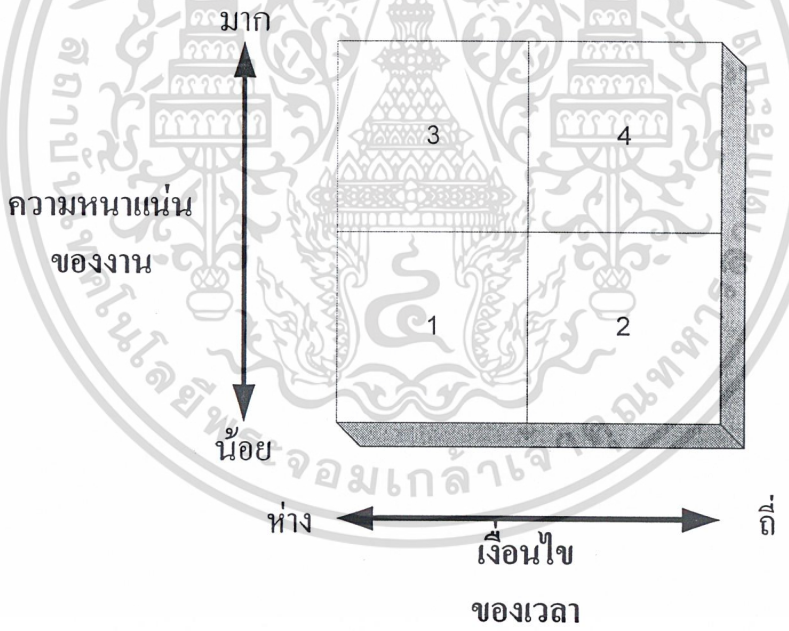
ลีนุกซ์กับระบบเรียลไทม์เป็นการวิเคราะห์เพื่อศึกษาความสามารถของลีนุกซ์ในการรองรับระบบเรียลไทม์ และศึกษาค้นคว้าเพื่อหาส่วนขยายเพิ่มเติมให้ลีนุกซ์สามารถรองรับระบบเรียลไทม์ได้สมบูรณ์มากขึ้น โดยในส่วนแรกกล่าวถึงระบบเรียลไทม์ว่าสามารถแยกแยะออกมาเป็นรูปแบบได้อย่างไร ส่วนที่สองกล่าวถึงลีนุกซ์สามารถเข้าไปรองรับการทำงานรูปแบบใดได้บ้าง ส่วนสุดท้ายกล่าวถึงส่วนขยายเพิ่มเติมและบทสรุปในการนำมาใช้งาน

3.2.1 ระบบเรียลไทม์

ระบบเรียลไทม์ คือระบบที่ทำงานตามเงื่อนไขของเวลา โดยเวลาส่วนใหญ่เราจะวัดจากประสิทธิภาพการตอบรับของระบบ ซึ่งระบบต้องตอบรับในเวลาที่ถูกต้องตามเหตุการณ์ที่เข้ามาจากภายนอก การออกแบบระบบเรียลไทม์ต้องคาดเดา (Predictable) เวลาที่ต้องการของระบบได้และต้องเชื่อถือได้ (Reliability) ว่าประสิทธิภาพของระบบที่ได้ต้องถูกต้อง (Correct) และ ตามเวลาที่เรากำหนดไว้ (Timely) สิ่งที่เราต้องการในการตรวจสอบความถูกต้องตามเงื่อนไขทางเวลาคือ ต้องพบ เดดไลน์ (Deadline)

ระบบคอมพิวเตอร์โดยทั่วไป ก็สามารถเป็นระบบเรียลไทม์ได้ โดยอาจจะเรียกว่า ระบบแบบซอฟต์เรียลไทม์ (Soft Real-time) คือระบบที่ถ้าเกิดความล้มเหลวในการทำงานขึ้น ความเสียหายจะไม่รุนแรงมากนัก ส่วนระบบที่ถ้าเกิดความล้มเหลวแล้วความเสียหายจัดอยู่ในขั้นรุนแรงเรียกว่า ระบบแบบฮาร์ดเรียลไทม์ (Hard Real-time) ส่วนระบบที่เป็นฮาร์ดเดดไลน์ แต่บางครั้งก็สามารถละเลยเดดไลน์นั้นได้ ก็จะเรียกว่า ระบบแบบเฟิร์มเรียลไทม์ (Firm Real-time)

รูปแบบของระบบเรียลไทม์กับการพัฒนาที่สามารถแยกแยะออกมาได้ โดยขึ้นอยู่กับความสัมพันธ์ของความหนาแน่นของงานและเงื่อนไขของเวลา ซึ่งสามารถสามารแสดงได้ดังรูปที่ 3-2



รูปที่ 3-2 รูปแบบของระบบเรียลไทม์กับการพัฒนา

จากรูปที่ 3-2 ในส่วนของพื้นที่ถูกแบ่งออกเป็น 4 ส่วน ซึ่งแต่ละส่วนระบุถึงรูปแบบของการพัฒนาระบบเรียลไทม์ดังนี้

1. แบบที่มีความหนาแน่นของงานน้อยและเงื่อนไขเวลาห่าง สามารถพัฒนาได้ง่าย
2. แบบที่มีความหนาแน่นของงานน้อยและเงื่อนไขเวลาถี่ ในการพัฒนาจำเป็นที่ต้องมีการคำนึงถึงการใช้เวลา

3. แบบที่มีความหนาแน่นของงานมากและเงื่อนไขเวลาห่าง จำเป็นที่ต้องใช้ระบบปฏิบัติการมาช่วยเพื่อให้การพัฒนาง่ายขึ้น
4. แบบที่มีความหนาแน่นของงานมากและเงื่อนไขเวลาถี่ ระบบปฏิบัติการที่นำมาใช้จำเป็นที่ต้องเพิ่มความสามารถในการจัดลำดับและระดับความสำคัญงาน เพื่อให้ยืนยันและพิสูจน์ว่าสามารถทำงานได้ตามเวลา

3.2.2 ลีนุกซ์กับการรองรับระบบเรียลไทม์

จากรูปที่ 3-2 บอกถึงรูปแบบของการพัฒนาระบบเรียลไทม์ เมื่อนำมาเปรียบเทียบกับลีนุกซ์สามารถแบ่งระบุได้ถึงรูปแบบที่ลีนุกซ์รองรับได้นั้นก็คือ รูปแบบที่ 1–3 ซึ่งในรูปแบบที่ 1–2 นั้นลีนุกซ์สามารถรองรับการพัฒนาได้ แต่ในรูปแบบที่ 3 กับในงานแบบเรียลไทม์บางงานนั้นจำเป็นที่จะต้องมีการจัดรูปแบบการจัดลำดับงานและลำดับความสำคัญของงานให้เหมาะสมเพื่อให้สามารถทำงานได้ โดยเฉพาะในระบบที่มีทรัพยากรจำกัด ในรูปแบบที่ 4 นั้นงานแบบเรียลไทม์บางงานที่มีทรัพยากรมากพอลีนุกซ์ก็สามารถรองรับการพัฒนาได้ แต่ก็ไม่สามารถยืนยันและพิสูจน์ได้ว่าสามารถทำงานได้ตามเวลา ดังที่กล่าวมาแล้วว่าในงานรูปแบบนี้ระบบปฏิบัติการที่จะนำมาใช้จำเป็นต้องมีการเพิ่มความสามารถในการจัดลำดับและระดับความสำคัญของงาน ดังนั้นจึงจำเป็นต้องมีการเพิ่มความสามารถให้กับลีนุกซ์

ในปัจจุบันได้มีการจัดสร้างส่วนขยายความสามารถทางด้านเรียลไทม์ให้กับลีนุกซ์ อยู่หลายตัว ซึ่งในแต่ละตัวจะความสามารถและหลักการในการทำงานแตกต่างกันไป ในโครงการนี้ได้ทำเลือกส่วนขยายเพิ่มเติมมาใช้ในการพัฒนาให้กับลีนุกซ์ที่ทำงานบนระบบ embedded โดยมีความต้องการพื้นฐานดังต่อไปนี้

1. เปิดเผยโค้ด
2. ทำการพอร์ตง่าย
3. แก้ไขโค้ดของเคอร์เนลน้อย

3.2.3 ส่วนขยายเพิ่มเติมทางด้านเรียลไทม์สำหรับลีนุกซ์

จุดมุ่งหมายที่ไลนัสทำการพัฒนาลีนุกซ์ขึ้นมาก็เพื่อนำไปใช้งานโดยไม่เสียค่าใช้จ่าย อยู่บนพื้นฐานของระบบยูนิกซ์ ที่ไม่ใช่ยูนิกซ์แบบเรียลไทม์ และได้สืบทอดคุณลักษณะของยูนิกซ์ทั้งหมด แท้ที่จริงแล้วเคอร์เนลของลีนุกซ์ไม่ได้ถูกออกแบบมาเพื่อใช้ในงานแบบเรียลไทม์ แต่ในปัจจุบันมีผลงานหลายผลงานที่ทำการพัฒนาลีนุกซ์ให้เป็นแบบเรียลไทม์ จากการศึกษาได้พบว่ามีตัวที่ควรนำเอาตัดสินใจเลือกอยู่ 3 ตัว คือ Timeysy/RK,RTLinux และ RTAI สามารถแสดงตารางเปรียบเทียบได้ดังตารางที่ 3-1

ตารางที่ 3-1 การเปรียบเทียบลักษณะเด่นของ Timesys/RK, RTAI และ RTLinux

RTOS	คุณลักษณะเด่น	ความต้องการลักษณะเรียลไทม์
Timesys/RK	- สนับสนุนระบบ embedded - รับประกัน QoS - ให้บริการในส่วนของลีนุกซ์ได้ทั้งหมด - 256 priority level	- ไม่เปิดเผยโค้ด
RTLinux	- เมื่อมีบักเกิดขึ้น สามารถทำการ Fixed ได้ง่าย	- เปิดเผยโค้ด - ทำการพอร์ตได้ง่าย
RTAI	- ประสิทธภาพสูงมาก - ใช้หลักการ RTHAL ทำให้ไม่ไปยุ่งเกี่ยวกับ kernel มากนัก - Fixed priority	- เปิดเผยโค้ด - ทำการพอร์ตได้ง่าย - ทำการแก้ไขโค้ดของเคอร์เนลน้อย

เมื่อเราได้ทำการศึกษาระบบลีนุกซ์แบบเรียลไทม์ที่มีอยู่แล้ว จะเห็นว่าแต่ละระบบมีคุณลักษณะที่แตกต่างกันไปขึ้นอยู่กับจุดมุ่งหมายในการสร้างและการนำไปใช้งาน ระบบลีนุกซ์แบบเรียลไทม์ที่เรากำลังพัฒนาขึ้นมาจะนำหลักการของ RTAI มาใช้ ซึ่งได้พิจารณาแล้วเห็นว่าเหมาะสมในการนำมาใช้งานเป็นอย่างยิ่งและตรงตามความต้องการระบบเรียลไทม์เพื่อใช้พัฒนา

3.3 ลีนุกซ์กับระบบ embedded

ในส่วนนี้เป็นการวิเคราะห์เพื่อจัดหาระบบ embedded ที่สามารถทำงานกับลีนุกซ์มาใช้ในการพัฒนาโครงงาน ซึ่งระบบ embedded ที่ต้องการนั้นก็คือ บอร์ดแบบ embedded ที่ต้องมีคุณสมบัติเบื้องต้นดังต่อไปนี้

1. สามารถรันระบบปฏิบัติการลีนุกซ์
2. ต้นทุนเมื่อนำมาจัดสร้างน้อย
3. ขนาดของหน่วยความจำอย่างน้อย 2 เมกาไบต์
4. มีพอร์ตอนุกรม
5. ต้นทุนในการพัฒนาต่ำ
6. มีเครื่องมือสำหรับสนับสนุนการพัฒนาที่สามารถนำมาใช้ในการพัฒนากับลีนุกซ์

จากการที่ได้นำเอาคุณสมบัติเบื้องต้นที่ต้องการไปหาบอร์ดแบบ embedded ที่จะนำมาใช้ในโครงงงานนั้นได้พบว่ามีบอร์ดที่มีคุณสมบัติใกล้เคียงกับคุณสมบัติเบื้องต้นที่ต้องการ คือ บอร์ด AT91EB01

3.3.1 บอร์ด AT91EB01

บอร์ด AT91EB01 เป็นบอร์ดที่นำใช้งานในโครงงานนี้ ถูกพัฒนาโดยบริษัท ATMEL ใช้ไมโครคอนโทรลเลอร์ AT91M40400 ที่มี ARM7TDMI เป็นโปรเซสเซอร์ภายใน บอร์ดนี้มีคุณสมบัติเบื้องต้นดังนี้

- ต้นทุนในการสร้างน้อย
- หน่วยความจำขนาด 512 K สามารถขยายหน่วยความจำได้ถึง 2M
- มีพอร์ตอนุกรม
- ต้นทุนในการพัฒนาต่ำ
- มีเครื่องมือสำหรับสนับสนุนการพัฒนาที่สามารถนำมาใช้ในการพัฒนากับลีนุกซ์

จากคุณสมบัติเบื้องต้นยังขาดคุณสมบัติที่จำเป็นมากนั้นก็คือ สามารถรันระบบปฏิบัติการลีนุกซ์ได้ แต่เนื่องจากโปรเซสเซอร์ภายในนั้นได้มีการพัฒนาลีนุกซ์ไว้แล้วและมีไมโครคอนโทรลเลอร์รุ่นที่ใกล้เคียงกันได้ทำการพัฒนาส่วนแพลตฟอร์ม ที่จำเป็นไว้แล้ว ทำให้การนำมาบอร์ด AT91EB01 มีนำมาพัฒนาให้ลีนุกซ์สามารถทำงานได้

ลีนุกซ์ที่ได้รับการพัฒนาให้ทำงานกับไมโครคอนโทรลเลอร์รุ่นใกล้เคียงกันนี้คือ AT75C310 ของบริษัท ATMEL ลีนุกซ์ที่ทำงานกับไมโครคอนโทรลเลอร์นี้เป็นลีนุกซ์ที่ได้รับการพัฒนาต่อเพื่อให้สามารถทำงานได้กับไมโครคอนโทรลเลอร์ที่ไม่มีหน่วยจัดการหน่วยความจำ โดยเรียกลีนุกซ์ที่ได้รับการพัฒนาต่อนี้ว่า uClinux (อ่านว่า ยู-ซี-ลี-นุกซ์)

3.3.2 เครื่องมือในการพัฒนา

เครื่องมือในการพัฒนาสำหรับลีนุกซ์เป็นเครื่องมือที่ในชุด GNU ของ Free Software Foundation คือ binutils และ gcc ซึ่งทั้งคู่รองรับโปรเซสเซอร์ ARM7TDMI ทำให้สามารถคอมไพล์ให้ uClinux สามารถทำงานบนบอร์ดได้ เนื่องจากการพัฒนาเป็นแบบ Cross Development ทำให้ต้องมีวิธีการและเครื่องมือในการนำเอา uClinux ไปทำงานบนบอร์ดซึ่งในปัจจุบันมีวิธีที่นิยมอยู่ 2 วิธี คือ

1. Burning คือการนำเอา uClinux ที่ได้รับการคอมไพล์แล้วไปใส่ไว้หน่วยความจำประเภทคงอยู่ (Nonvolatile Memory) เช่น Flash, EEPROM เป็นต้น แล้วค่อยเริ่มทำงาน ซึ่งวิธีนี้จำเป็นต้องอาศัยโปรแกรมประเภท Bootloader ด้วย
2. Downloading คือการนำเอา uClinux ที่ได้รับการคอมไพล์แล้วโหลดผ่านทางสายสื่อสารไปใส่ไว้ในแรมแล้วสั่งให้เริ่มต้นทำงาน ณ ตำแหน่ง uClinux โหลดลงไป

วิธีแรกนั้นไม่เหมาะที่จะนำมาใช้ได้กับบอร์ดนี้ เนื่องจากโปรแกรมที่ใช้ทำ Burning นั้นทำงานบนระบบปฏิบัติการ Windows 9x ในการนำมาใช้งานจำเป็นต้องมีการสลับการใช้งานระบบปฏิบัติการ เพราะการพัฒนาอยู่บนระบบปฏิบัติการลีนุกซ์ซึ่งจะทำให้เสียเวลาในการพัฒนาอย่างมาก ดังนั้นวิธีที่สองจึงเป็นวิธีที่เหมาะมากกว่าสำหรับบอร์ด AT91EB01 ซึ่งก็ยังมีอยู่อีก 2 วิธี คือ ผ่านทางพอร์ตอนุกรม และผ่านทางช่องสื่อสาร JTAG การดาวน์โหลดผ่านทางพอร์ตอนุกรมใช้เฟิร์มแวร์เรียกว่า Angel ที่มีมาให้กับบอร์ดแต่การใช้งานบนลีนุกซ์นั้นยังไม่มีโปรแกรมที่รองรับโปรโตคอลในการสื่อสาร และผ่านทางช่องสื่อ

สารจำเป็นต้องทำมีโปรแกรมในการสื่อสารแบบ JTAG และฮาร์ดแวร์ที่ใช้ในการแปลงสัญญาณจากพอร์ตขนานไปสู่ JATG ซึ่งเรียกว่า Wigler

จากการศึกษาและได้ทำการวิเคราะห์แล้ว การ download ผ่านทาง JTAG นั้นไม่จำเป็นที่ต้องมีการพัฒนาเนื่องจากมีโปรแกรมที่ทำการสื่อสารแบบ JTAG อย่างง่ายบนลีนุกซ์ให้แล้ว และ Wigler ที่นำมาใช้นั้นสามารถหาได้ทั่วไปหรือสามารถทำขึ้นมาได้ไม่ยาก เพียงแค่ทำให้การแก้ไขโปรแกรมให้เข้ากันได้กับ Wigler นั้นๆ

Wigler ที่นำมาใช้ในโปรแกรมนี้คือ FlashLink adapter ส่วนรายละเอียดในการเชื่อมต่อสามารถดูได้จากคู่มือการพัฒนาระบบลีนุกซ์แบบเรียลไทม์ และข้อมูลในสร้าง Wigler ขึ้นมาเองนั้นดูได้จากคู่มือการพัฒนาระบบลีนุกซ์แบบเรียลไทม์

แม้ว่าจะสามารถทำให้ uClinux ทำงานบนบอร์ดได้แล้ว แต่ในการใช้พัฒนาแอพพลิเคชั่นนั้นจำเป็นที่ต้องมีไลบรารี่พื้นฐาน เพื่อให้การพัฒนาเป็นไปได้ง่าย ซึ่งในลีนุกซ์ทั่วไปนั้นจะนิยมใช้ไลบรารี่ที่ชื่อ glibc แต่ glibc นั้นมีขนาดเมื่อนำมารวมกับแอพพลิชั่นแล้วไม่เหมาะสมในการใช้งานในระบบ embedded ดังนั้น uClinux จึงได้ทำการพัฒนาไลบรารี่ที่ชื่อ uClibc ขึ้นมาสำหรับใช้งานกับ uClinux ด้วย

3.4 uClinux ที่นำมาใช้ในโครงงาน

uClinux ที่นำมาพัฒนากับบอร์ด AT91EB01 นั้นจำเป็นที่ต้องมีการแก้ไขเพิ่มเติมเพื่อให้สามารถทำงานบนบอร์ดได้นั้นจำเป็นที่ต้องศึกษาถึงโครงสร้างของลีนุกซ์และค้นหาส่วนที่ต้องทำการแก้ไขเพิ่มเติม โดยดูจากลำดับการทำงานของลีนุกซ์และอ้างอิงจาก uClinux ที่ทำงานบนไมโครคอนโทรลเลอร์ที่ใกล้เคียง เมื่อได้ทำการแก้ไขและเพิ่มเติมให้ uClinux สามารถทำงานบนบอร์ด AT91EB01 แล้วจึงทำการพัฒนาให้ RTAI สามารถทำงานกับ uClinux ได้

3.4.1 uClinux กับบอร์ด AT91EB01

ในการนำเอา uClinux มาพัฒนาต่อให้สามารถทำงานบนบอร์ด AT91EB01 นั้นต้องมีการแก้ไขและเพิ่มเติมไฟล์ต่างๆดังที่จะแสดงต่อไป โดยวิเคราะห์จาก uClinux ที่ทำงานบนไมโครคอนโทรลเลอร์รุ่นใกล้เคียงและจากแนวทางการพอร์ต uClinux สำหรับฮาร์ดแวร์ใหม่

```
Makefile
arch/armnommu/Makefile
arch/armnommu/Config
arch/armnommu/kernel/arch-atmel.S
arch/armnommu/kernel/setup.c
arch/armnommu/lib/io-atmel.c
drivers/block/Makfile
drivers/block/blkmem.c
drivers/block/serial-atmel.c
drivers/block/serial-atmel.h
drivers/char/tty_io.c
include/asm-armnommnu/arch-atmel/a.out.h
include/asm-armnommnu/arch-atmel/dma.h
include/asm-armnommnu/arch-atmel/hardware.h
include/asm-armnommnu/arch-atmel/io.h
include/asm-armnommnu/arch-atmel/irq.h
include/asm-armnommnu/arch-atmel/irqs.h
```

```
include/asm-armnommnu/arch-atmel/mmu.h
include/asm-armnommnu/arch-atmel/processor.h
include/asm-armnommnu/arch-atmel/shmparam.h
include/asm-armnommnu/arch-atmel/system.h
include/asm-armnommnu/arch-atmel/time.h
include/linux/flat.h
kernel/ksym.c
```

รายละเอียดในการแก้ไขเพิ่มเติม และขั้นตอนการพัฒนาสามารถดูได้จากคู่มือการพัฒนาระบบลีนุกซ์แบบเรียลไทม์ โดยการแก้ไขเพิ่มเติมส่วนมากเป็นส่วนที่ยึดติดกับฮาร์ดแวร์ (Hardware Dependent) ซึ่งเป็นข้อดีของลีนุกซ์ในการนำไปพัฒนาบนฮาร์ดแวร์หรือแพลตฟอร์มใหม่

3.4.2 uClinux กับ RTAI

เนื่องจาก RTAI เดิมที่ทำงานกับลีนุกซ์นั้นทำงานบนสถาปัตยกรรม x86 และมีบางส่วนที่ยึดติดกับสถาปัตยกรรมนี้อย่างมาก ดังนั้นการทำให้ RTAI ทำงานกับ uClinux ที่ได้รับการแก้ไขเพิ่มเติมแล้วนั้น จำเป็นที่ต้องศึกษาถึงหลักการและโครงสร้างของ RTAI ให้เข้าใจก่อนทำการแก้ไข

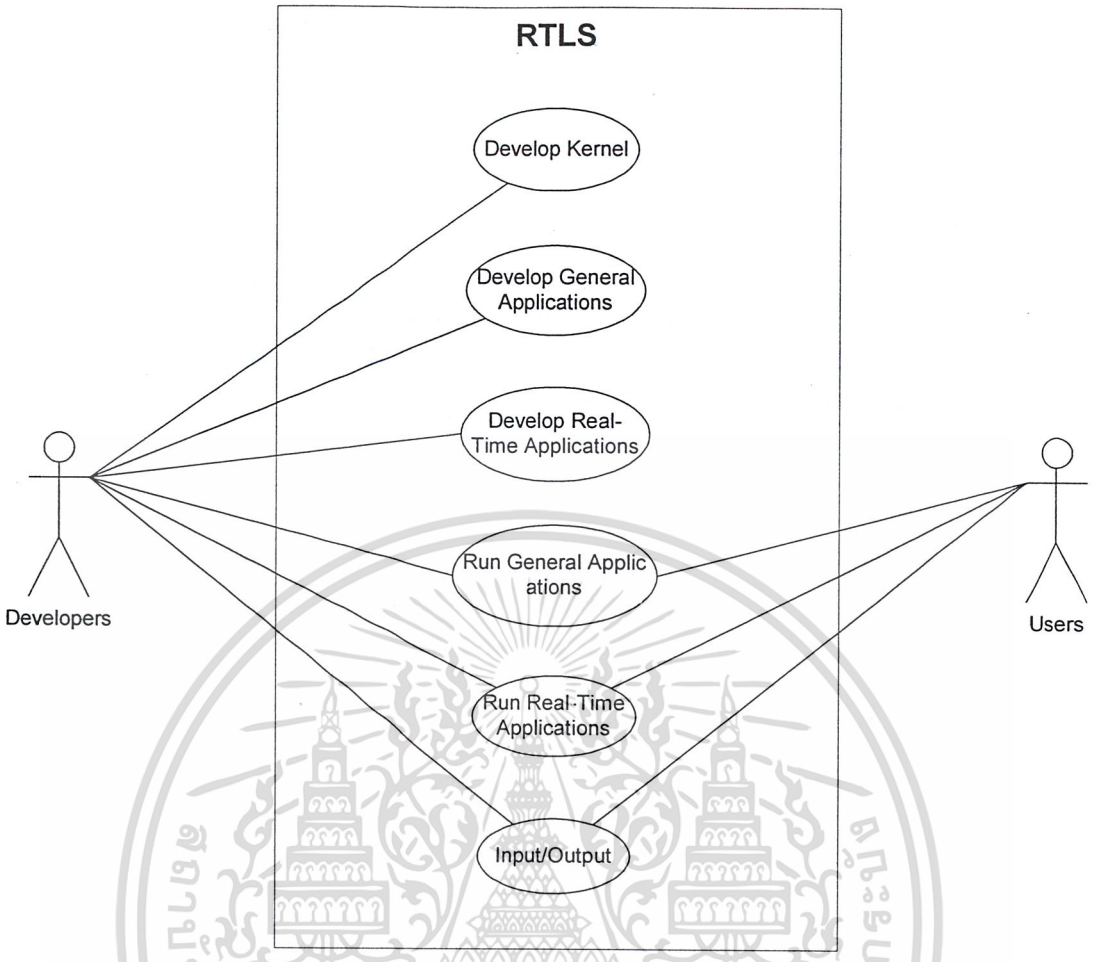
ในส่วนของการขยายความสามารถทางด้านเรียลไทม์ของ RTAIให้กับ uClinux ที่ได้รับการพัฒนาให้ทำงานบนบอร์ด AT91EB01 นั้นยังไม่สามารถทำงานได้ เนื่องส่วนสำคัญของ RTAI นั้นก็คือ Interrupt Dispatcher ยังพัฒนาไม่เรียบร้อย เนื่องจากต้องเขียนขึ้นมาใหม่ทั้งหมดจากหลักการของ RTAI แต่ในส่วนของ RTHAL ที่ได้ทำการเข้าไปแก้เคอร์เนล นั้นได้ทำการแก้ไขเรียบร้อย แต่อาจจะยังขาดบางส่วนที่จำเป็นต้องใช้ใน Interrupt Dispatcher ตัวใหม่ซึ่งไม่เหมือนกับใน x86

ถึงแม้ว่าในส่วนของการพัฒนาเพื่อขยายขีดความสามารถทางด้านเรียลไทม์ให้กับ uClinux ไม่เรียบร้อย แต่ก็ไม่ได้หมายความว่าระบบที่ได้พัฒนาขึ้นจะไม่สามารถให้บริการสำหรับงานแบบเรียลไทม์ได้ แต่จะมีงานแบบเรียลไทม์บางงานที่ไม่สามารถให้บริการได้ดังที่อธิบายในหัวข้อระบบเรียลไทม์

โดยการที่จะให้บริการสำหรับงานแบบเรียลไทม์ในลีนุกซ์นั้นจำเป็นต้องมีการเปลี่ยนการจัดลำดับงานเป็นแบบ FIFO หรือ Round Robbin และให้เพิ่มระดับความสำคัญให้อยู่ในระดับเรียลไทม์ ซึ่งความสามารถนี้เป็นไปตามมาตรฐาน POSIX 1003.1b ซึ่งเป็นมาตรฐานการให้บริการทางด้านเรียลไทม์ของระบบยูนิกซ์

3.5 Use Case

แสดงให้เห็นถึงฟังก์ชั่นการทำงานหลักที่สามารถใช้งานได้โดยผู้ใช้งาน ซึ่งผู้ใช้ส่วนใหญ่คือผู้พัฒนาระบบ (developer) มี 5 ฟังก์ชั่นหลักดังรูปที่ 3-3

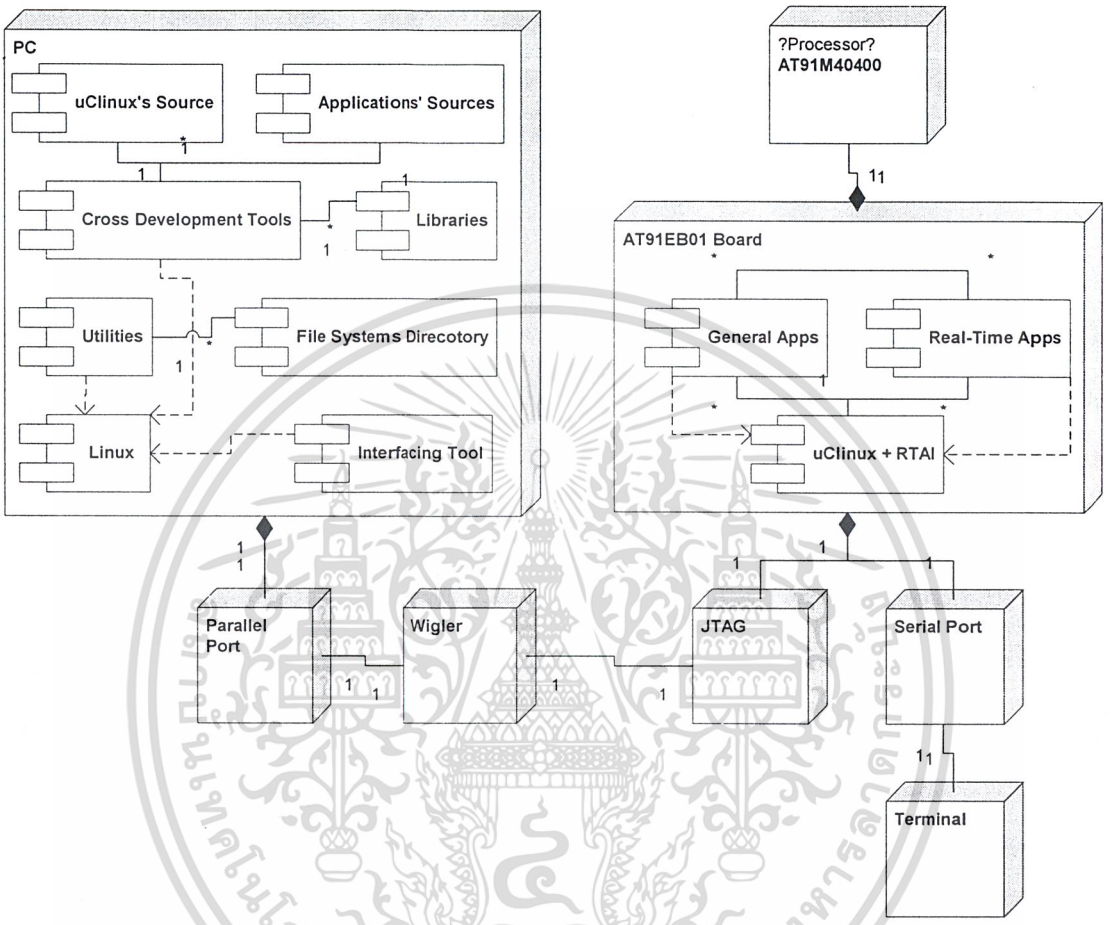


รูปที่ 3-3 แสดง Use Case Diagram ของระบบลีนุกซ์แบบเรียลไทม์

1. Develop Kernel คือ การปรับปรุงแก้ไขและคอนฟิก uClinux บนเครื่องพีซี เพื่อนำไปทำงานบนบอร์ด AT91EB01
2. Develop General Applications คือ การพัฒนาแอพพลิเคชั่นแบบทั่วๆไปที่ไม่ได้ทำงานแบบเรียลไทม์บนเครื่องพีซี เพื่อนำไปทำงานบนบอร์ด AT91EB01
3. Develop Real-Time Applications คือ การพัฒนาแอพพลิเคชั่นแบบเรียลไทม์บนเคื่องพีซี เพื่อนำไปทำงานบนบอร์ด AT91EB01
4. Run General Applications คือ การนำเอาแอพพลิเคชั่นแบบทั่วๆไปที่ทำการพัฒนาขึ้นมาไปทำงานบนบอร์ด AT91EB01
5. Run Real-Time Applications คือ การนำเอาแอพพลิเคชั่นแบบเรียลไทม์ที่ทำการพัฒนาขึ้นมาซึ่งไม่จำเป็นต้องใช้ความสามารถทางด้านเรียลไทม์ของ RTAI ไปทำงานบนบอร์ด AT91EB01
6. Input/Ouput คือ การแสดงผลลัพธ์และควบคุมการทำงานของ uClinux และแอพพลิเคชั่นที่นำมาทำงานบนบอร์ด

3.6 โครงสร้างของระบบลีนุกซ์แบบเรียลไทม์

จากการศึกษาค้นคว้าและทำการวิเคราะห์เพื่อหาส่วนต่างๆที่จำเป็นที่ต้องนำมาใช้ในระบบลีนุกซ์แบบเรียลไทม์ในรูปที่ 3-1 สามารถแสดงได้ดังรูปที่ 3-4



รูปที่ 3-4 โครงสร้างของระบบลีนุกซ์แบบเรียลไทม์บนบอร์ด AT91EB01

ในรูปที่ 3-4 แสดงให้เห็นในระบบลีนุกซ์แบบเรียลไทม์ซึ่งแบ่งออกได้เป็น 2 ส่วน คือ ส่วนที่ใช้งานบนเครื่องพีซี และส่วนที่ทำงานบนบอร์ด AT91EB01 ซึ่งได้รับได้รับการพัฒนาจากส่วนแรก

3.6.1 ส่วนที่ใช้งานบนเครื่องพีซี

ส่วนที่ใช้งานบนเครื่องพีซี เป็นส่วนที่ใช้ในการพัฒนา uClinux และแอพพลิเคชั่นทั้งแบบทั่วไปและแบบเรียลไทม์ แล้วนำไปทำงานในส่วนต่อไป ประกอบด้วยส่วนต่างๆ ดังที่แสดงต่อไป โดยเริ่มที่ส่วนที่เกี่ยวกับฮาร์ดแวร์ก่อน

1. PC คือ เครื่องคอมพิวเตอร์ส่วนบุคคลที่นำมาใช้ในการพัฒนา
2. Parallel Port คือ พอร์ตขนานบนพีซีสำหรับเชื่อมต่อกับวิกเกอร์ (Wigler)
3. Wiggler เป็นวิกเกอร์ที่ใช้แปลงสัญญาณระหว่าง Parallel Port บนพีซีกับ JTAG บนบอร์ด AT91EB01

4. Linux คือ ระบบปฏิบัติการลีนุกต์ที่ทำงานบนเครื่องพีซีซึ่งโปรแกรมที่ใช้ในการพัฒนาทุกตัวจะทำงานอยู่บนลีนุกซ์
5. uClinux's Source คือ โค้ดทั้งหมดของ uClinux ที่สามารถนำไปพัฒนาต่อ หรือคอมไพล์เพื่อนำไปทำงานบนบอร์ด AT91EB01
6. Applications' Sources คือ โค้ดของแอพพลิเคชั่นที่จะนำไปทำงานกับ uClinux บนบอร์ด AT91EB01 ทั้งแบบทั่วไปและแบบเรียลไทม์
7. Libraries คือ ไลบรารี่พื้นฐานสำหรับพัฒนาแอพพลิเคชั่นบน uClinux ได้แก่ uClibc
8. Corss Development Tools คือ ชุดโปรแกรมในการคอมไพล์ uClinux และแอพพลิเคชั่น โดยชุดโปรแกรมนี้นำมาจากโปรแกรมในชุด GNU ของ Free Software Foundation (http://www.gnu.org) ประกอบไปด้วย
   ก) Binutils คือ ชุดของโปรแกรมอรรถประโยชน์ (utilities) ในการจัดการกับไบนารี่
   ข) GCC คือ ชุดของโปรแกรมคอมไพเลอร์ภาษา C, C++, Object C, Fortan และอื่นๆ

1. Utilities คือ ชุดโปรแกรมอรรถประโยชน์ ประกอบด้วย
   ก) elf2flt คือ โปรแกรมที่ทำหน้าที่แปลงไบนารี่ไฟล์ที่ได้จาการคอมไพล์ในรูปแบบ ELF ไปเป็นแบบแฟลทเพื่อให้ใช้งานบน uClinux ได้
   ข) genromfs คือโปรแกรมสร้าง Image ของ File Systems จากไดเรกทอรี่เพื่อนำไปใช้เป็น Root File Systems ของ uClinux
2. File Systems Directory คือ ไดเรกทรอรี่ที่ประกอบด้วยไฟล์และไดเรกทรอรี่ในการสร้าง Root File Systems ด้วยโปรแกรม genromfs สำหรับใช้งานบน uClinux ซึ่งแอพพลิชั่นที่พัฒนามาทำงานกับ uClinux จะอยู่ภายในไดเรกทอรี่นี้ด้วย
3. Interfacing Tool โปรแกรมที่ใช้ทำการดาวน์โหลดและอัพโหลดโปรแกรมและข้อมูลผ่านทางพอร์ตขนานกับบอร์ด AT91EB01 ผ่านทางการเชื่อมต่อแบบ JTAG ซึ่งเรียกโปรแกรมนี้ว่า armtool

3.6.2 ส่วนที่ทำงานบนบอร์ด AT91EB01

หลังที่ได้จัดสร้าง uClinux และแอพพลิเคชั่นที่จะนำมาทำงานบนบอร์ด AT91EB01 แล้วจากส่วนที่ใช้งานบนเครื่องพีซีนั้น เมื่อนำมาทำงานบนบอร์ด AT91EB01 จะประกอบไปด้วยส่วนต่างๆดังต่อไปนี้

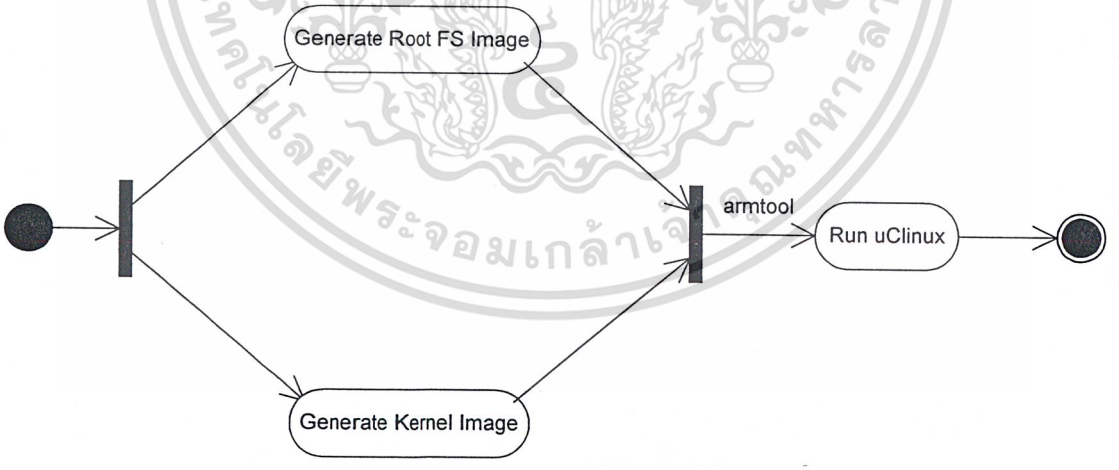
1. AT91EB01 Board คือ ระบบ embedded ที่เลือกมาใช้งานในโครงงาน
2. AT91M40400 คือ ไมโครคอนโทรลเลอร์สำหรับบอร์ด AT91EB01 ที่มีโปรเซสเซอร์ ARM7TDMI อยู่ภายใน
3. JTAG คือ ช่องเชื่อมต่อเพื่อใช้ทำการควบคุมการทำงานของไมโครคอนโทรลเลอร์ที่ใช้ ซึ่งในโครงงานนี้นำมาใช้เพื่อนการดาวน์โหลดและอัพโหลดโปรแกรมและข้อมูล
4. Serial Port คือ พอร์ตอนุกรมสำหรับเชื่อมต่อกับอินพุตเอาท์พุตที่เป็นเทอร์มินอล

5. Terminal คือ อุปกรณ์อินพุตเอาท์พุตแบบอักขระ
6. uClinux+RTAI คือ ระบบปฏิบัติการ uClinux ที่ได้เพิ่มความสามารถทางด้านเรียลไทม์แบบ RTAI เข้าไปที่ได้รับการจัดสร้างจากส่วนที่ทำงานบนเครื่องพีซี
7. General App. คือ แอพพลิเคชั่นทั่วๆไปที่ทำงานบน uClinux ไม่ได้ทำงานแบบเรียลไทม์ที่ได้รับการจัดสร้างจากส่วนที่ทำงานบนเครื่องพีซี
8. Real-Time App. คือ แอพพลิเคชั่นแบบเรียลไทม์ที่ทำงานบน uClinux อาจจะไม่จำเป็นต้องอาศัยความสามารถทางด้านเรียลไทม์แบบ RTAI

3.7 การใช้งาน uClinux กับบอร์ด AT91EB01

การใช้งาน uClinux ในการพัฒนาและนำไปทำงานบนบอร์ด AT91EB01 สามารถแบ่งออกได้เป็น 3 ส่วนใหญ่ ดังที่แสดงในรูปที่ 3-5

1. Generate Root FS. Image คือ การสร้าง Root File Systems Image สำหรับใช้เป็น Root File System สำหรับ uClinux ซึ่งประกอบไปด้วยแอพพลิเคชั่น โปรแกรมทั่วไป แฟ้มข้อมูล และแฟ้มที่จำเป็นในการทำงาน เช่น dev
2. Generate Kernel Image คือ การสร้าง Kernel Image ของ uClinux แบบไบนารี่ เพื่อนำไปทำงานบนบอร์ด AT91EB01
3. Run uClinux คือ การนำเอา uClinux และแอพพลิเคชั่นจาก Root File Systems Image ไปทำงานบนบอร์ด AT91EB01



รูปที่ 3-5 การใช้งาน uClinuxกับบอร์ด AT91EB01

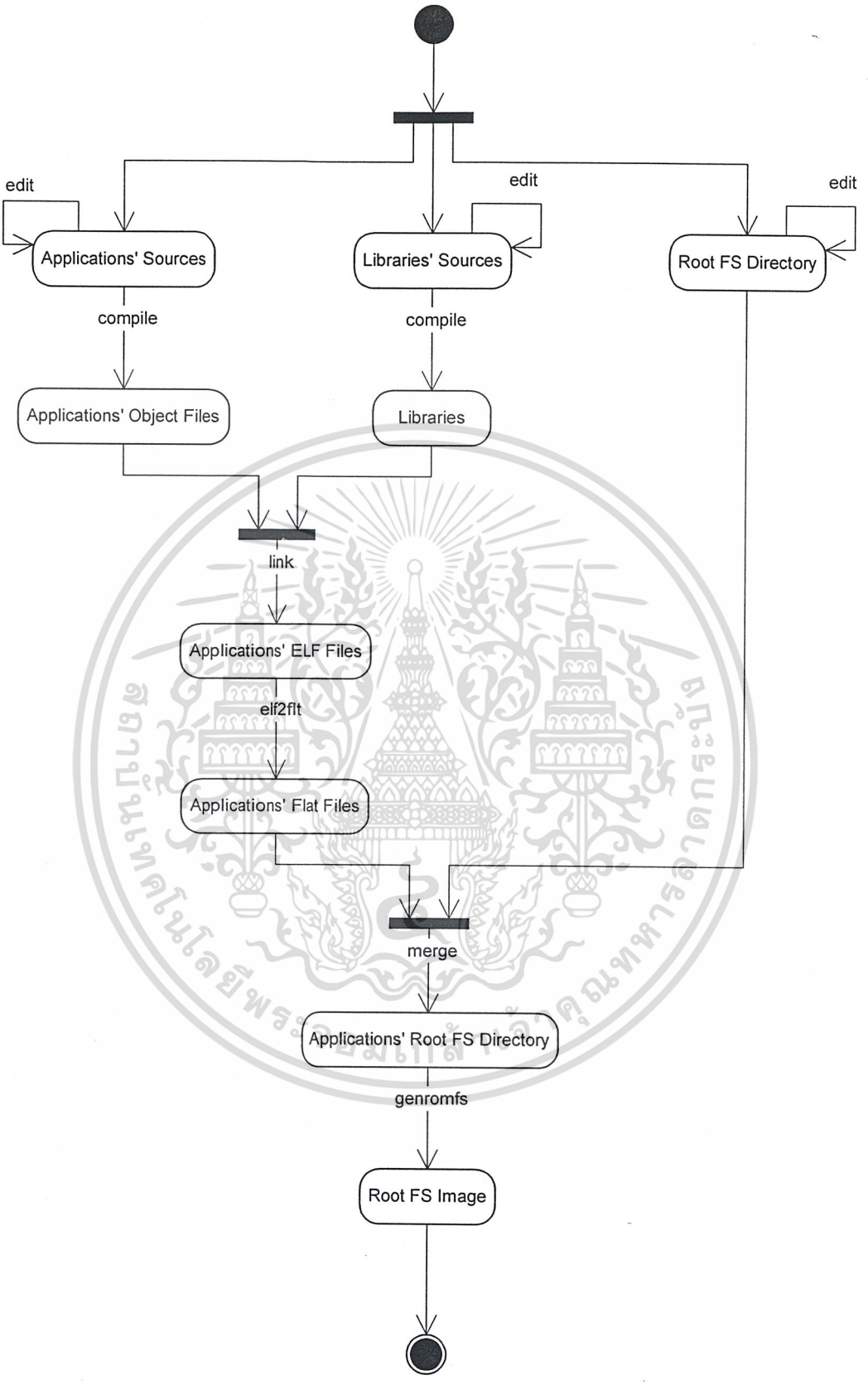
ก่อนการ Run uClinux นั้นจำเป็นที่ต้องทำการนำเอา Kernel Image และ Root File Systems Image ไปใส่ในหน่วยความจำบนบอร์ด AT91EB01 ด้วยโปรแกรม armtool

3.7.1 Generate Root File Systems Image

ในการสร้าง Root File Systems Image ซึ่งมีความสำคัญสำหรับผู้พัฒนาแอพพลิชั่นสำหรับบอร์ด AT91EB01 มากกว่าการสร้าง Kernel Image ซึ่งได้มีการเตรียมไว้ให้แล้วไม่จำเป็นต้องสร้างใหม่ Root File Systems Image ประกอบไปด้วยแอพพลิเคชั่น โปรแกรมทั่วไป แฟ้มข้อมูล และแฟ้มที่จำเป็นในการทำงานนั้นสามารถแยกการพัฒนาได้เป็น 2 ส่วนดังรูปที่ 3-6 ตามเส้นทางด้านซ้ายและขวา

ตามเส้นทางด้านซ้ายจากโค้ดต้นแบบของแอพพลิเคชั่น (Applications' Sources) ที่ทำการสร้างและแก้ไขนำไปทำการคอมไพล์ (compile) เป็นออบเจ็กต์ไฟล์ของแอพพลิเคชั่น (Applications' Object Files) แบบ ELF และทำการลิงค์ กับไลบรารี่ ซึ่งเราอาจจะทำการแก้ไขจากโค้ดต้นแบบของไลบรารี่ (Libraries' Sources) ได้เนื่องจากไลบรารี่ที่ใช้งานยังอยู่ในการพัฒนา หลังจากที่ทำการลิงค์เรียบร้อย แล้วจะได้เป็นแอพพลิชั่นที่สามารถทำงานได้แบบ ELF (Applications' ELF Files) แล้วทำการแปลงจากแบบ ELF ไปเป็นแบบแฟลท (Applications' Flat Files) ให้สามารถทำงานได้บน uClinux ด้วยโปรแกรม elf2flt

ตามเส้นทางด้านขวาจากไดเรกทอรี่สำหรับทำเป็นรูตไดเรกทอรี่ของ uClinux (Root FS Directory) ซึ่งจำเป็นต้องจัดเตรียมไฟล์ที่จำเป็นในการทำงานสำหรับแอพพลิชั่นและตัว uClinux เอง แล้วนำเอาแอพพลิชั่นที่ได้จากเส้นทางด้านซ้ายมารวมด้วยได้เป็นไดเรกทอรี่ที่พร้อมจะนำไปสร้างเป็นรูทไดเรกทอรี่ (Applications' Root FS Directory) แล้วนำมาแปลงเป็นไบนารี่ (Root FS Image) ที่พร้อมนำไปทำงานบนบอร์ด AT91EB01 ด้วยโปรแกรม genromfs

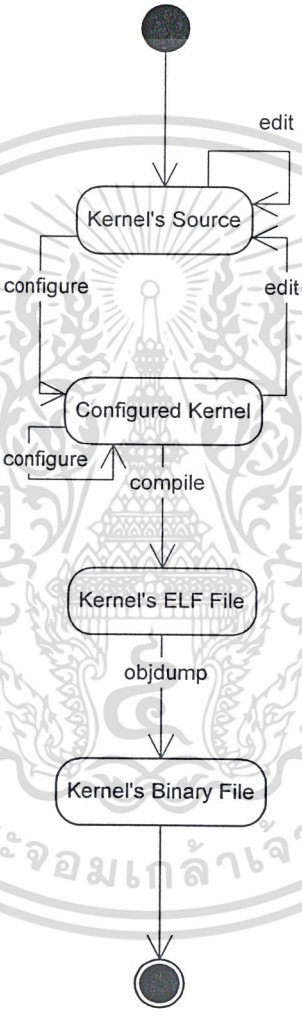


รูปที่ 3-6 Generate Root FS Image

3.7.2 Generate Kernel Image

ในการสร้าง Kernel Image นั้นไม่จำเป็นที่ต้องมีการสร้างทุกครั้ง แต่จะทำการสร้างเมื่อมีการแก้ไขหรือคอนฟิก uClinux ใหม่เท่านั้น ซึ่งขั้นตอนการสร้างแสดงดังรูปที่ 3-7

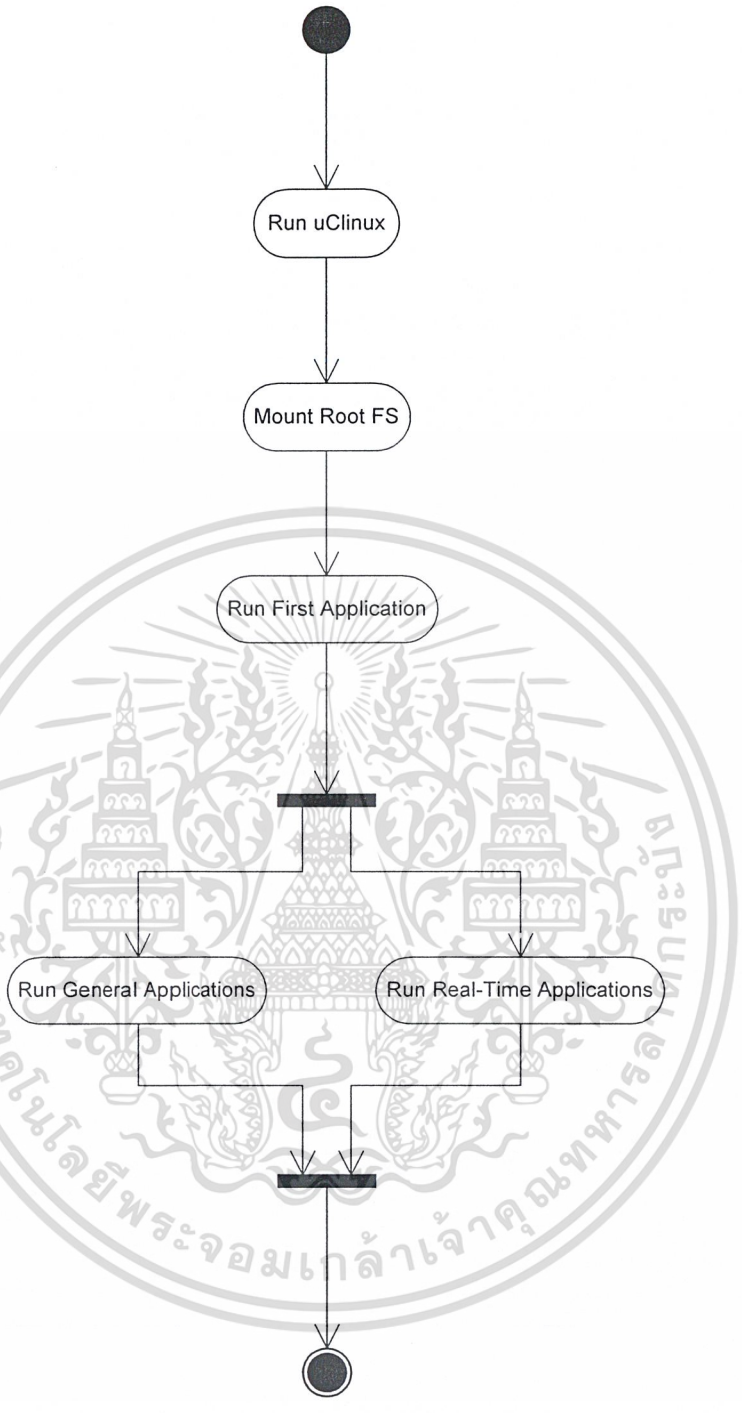
จากโค้ดต้นแบบของ uClinux (Kernel's Source) ที่ทำการแก้ไข แล้วทำการคอนฟิก (configure) ให้พร้อมสำหรับคอมไพล์ (Configured Kernel) ได้เป็นไฟล์แบบ ELF (Kernel's ELF File) และการแปลงเป็นไบนารี่ (Kernel's Binary File) ด้วยโปรแกรม objdump



รูปที่ 3-7 Generate Kernel Image

3.7.3 Run uClinux

เมื่อได้ Kernel Image และ Root File Systems Image แล้วนำมาทำงานบนบอร์ด AT91EB01 นั้นมีขั้นตอนการทำงานดังรูปที่ 3-8



รูปที่ 3-8 Run uClinux

เมื่อทำการสั่งให้ uClinux ทำงาน (Run uClinux) ด้วยโปรแกรม armtool แล้วตัว uClinux จะทำการเมาท์กับ Root File System Image ซึ่งอยู่ในหน่วยความจำ (Mount Root FS) เมื่อเมาท์เรียบร้อย uClinux จะทำการรันแอพพลิเคชั่นแรก (Run First Application) จาก Root File Systems ซึ่งต้องทำตั้งค่าไว้ก่อนที่ uClinux จะทำงานด้วยโปรแกรม armtool หลังจากที่รันแอพพลิเคชั่นแล้วอาจจะทำการรันแอพพลิเคชั่นอื่นเพิ่มเติมทั้งแบบทั่วไป (Run General Applications) และแบบเรียลไทม์ (Run Real-Time Application

บทที่ 4
การทดสอบและวิเคราะห์ผล

ในแต่ละขั้นตอนในการพัฒนาโครงงานนั้นมีการทดสอบและวิเคราะห์ผลว่าสิ่งที่ได้ทำไปแล้วนั้นเป็นไปตามเป้าหมายของแต่ละขั้นหรือไม่ ซึ่งจำเป็นต้องให้เป็นไปตามเป้าหมายนั้น แต่ถ้าไม่สามารถทำให้เป็นไปได้ตามเป้าหมายได้ จะมีการพิจารณาในสิ่งที่เกิดขึ้นและหาทางแก้ไขในโครงสร้างของระบบหรือขั้นตอนและเป้าหมายต่างๆ แต่ไม่ให้กระทบถึงเป้าหมายของโครงงาน เมื่อได้ทำตามขั้นตอนต่างๆ เป็นที่เรียบร้อยนั้นจะมีการทดสอบและวิเคราะห์ผลอีกทีว่าระบบที่พัฒนาขึ้นมานั้นเป็นไปตามเป้าหมายของโครงงานหรือไม่

ในการทดสอบและวิเคราะห์ผลของโครงงานนี้ทำตามขั้นตอนการใช้ในงานในหัวข้อการใช้งาน uClinux กับบอร์ด AT91EB01 ในบทที่ 3 โดยยึดถือเป้าหมายเป็นหลัก ในเป้าหมายบางส่วนไม่จำเป็นต้องจัดขั้นตอนทดสอบอย่างชัดเจน เนื่องจากเป็นส่วนหนึ่งในขั้นตอนการใช้งานและทดสอบแล้ว ดังนั้นสามารถแบ่งการทดสอบและวิเคราะห์ผลได้เป็น 3 ขั้นหลักๆ คือ

1. uClinux กับการทำงานบนบอร์ด AT91EB01
2. uClinux กับการให้บริการพื้นฐาน
3. uClinux กับงานแบบเรียลไทม์

4.1 uClinux กับการทำงานบนบอร์ด AT91EB01

การทดสอบในขั้นนี้เป็นการทดสอบเพื่อแสดงว่า uClinux ที่ได้ทำการพัฒนานั้นสามารถทำงานบนบอร์ด AT91EB01 ได้และใช้หน่วยความจำที่เหมาะสมเมื่อเทียบกับขนาดหน่วยความจำทั้งหมด โดยมีขั้นตอนในการทดสอบดังนี้

1. คอนฟิกเคอร์เนลโดยเลือกทุกตัวที่สามารถใช้ได้ เพื่อดูขนาดหน่วยความจำที่ใช้ในการทำงานสูงสุด โดยมีตัวเลือกดังต่อไปนี้

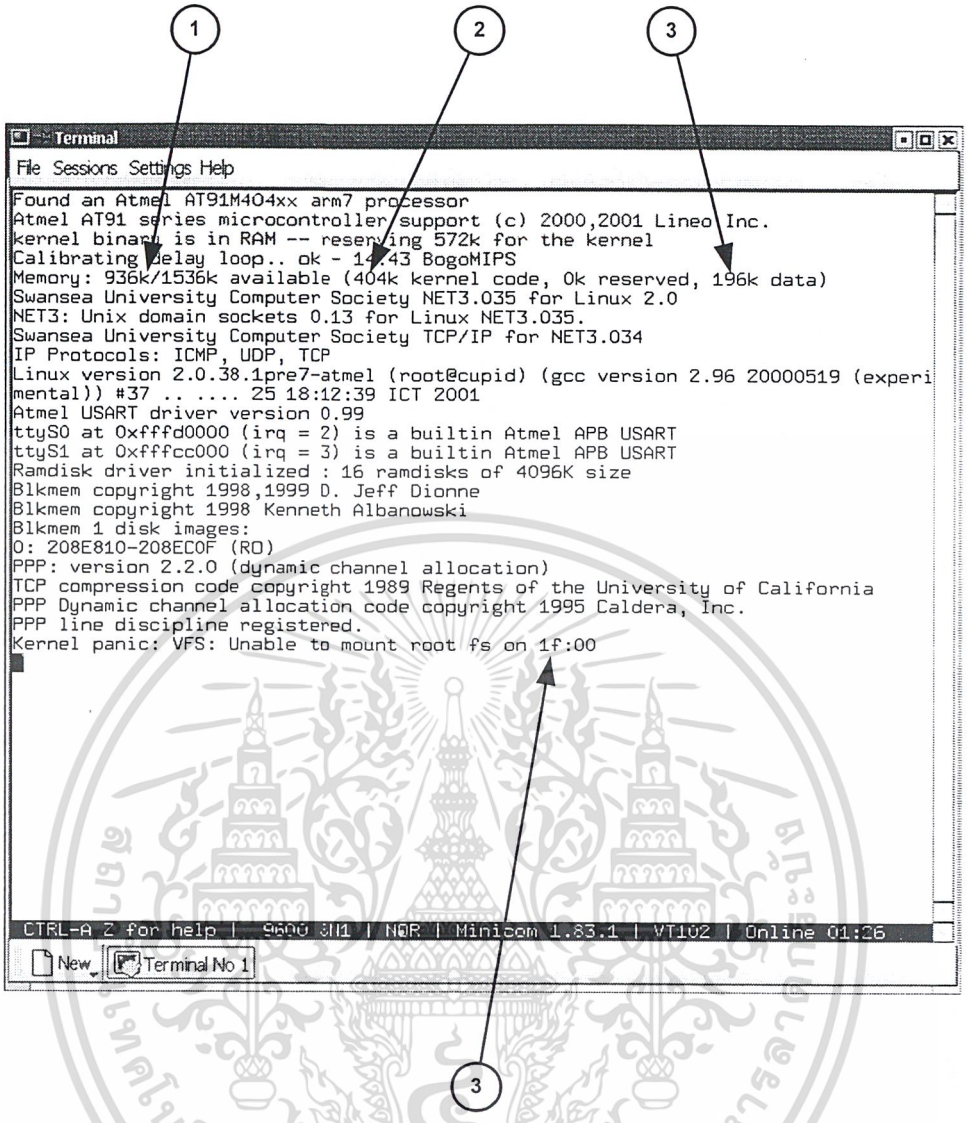
```
- Code maturity level options
   - [*] Prompt for development and/or incomplete
     code/drivers
- Loadable module support
   - [*] Enable loadable module support
- General setup
   - (Atmel-AT91) Arm system type
   - (327680000) Asyncronous Clock Frequency
   - (00000200) Base Address for boot parameters
   - (02000000) Base Address for DRAM
   - (00180000) Amount of DRAM present
   - (01002000) Base Address for Flash memory
   - (00008000) Amount of Flash present
   - [*] System will boot from flash
   - [*] Compile kernel with frame pointer (for useful
     debugging)
   - [*] Use new compilation options (for GCC 2.8)
   - [*] Debug kernel errors
```

```
    - [*] Networking support
    - [*] System V IPC
    - [*] Reduce memory footprint
  - Floppy, IDE, and other block devices
    - <*> ROM disk memory block device
    - <*> RAM disk support
    - <*>    Release empty RAM disk block
  - Character Devices
    - [*] Atmel AT91 serial support
    - [*] Console on Atmel AT91 serial support
    - (FFFD0000) USART 0 base address
    - (FFFCC000) USART 1 base address
  - Networking options
    - [*] TCP/IP networking
  - Network device support
    - [*] Network device support
    - <*> PPP (point-to-point) support
  - File systems
    - [*] /proc filesystem support
    - <*> ROM filesystem support
```

2. นำเคอร์เนลที่ได้ไปทำงานบนบอร์ด AT91EB01 แล้วดูเอาท์พุตที่แสดงทางเทอร์มินัล ถ้ามีการแสดงถึงว่า uClinux ได้ทำการเมาท์ Root File Systems ที่เป็น ROM File Systems (majort #1F) แสดงว่าการทำงานของ uClinux เป็นไปอย่างถูกต้อง และเอาท์พุตที่แสดงทางเทอร์มินัลนั้นยังบอกถึงขนาดหน่วยความจำที่ใช้ไปและที่เหลืออยู่ ซึ่งสามารถนำมาเปรียบเทียบว่าการใช้งานหน่วยความจำนั้นเหมาะสมหรือไม่

เมื่อทำการทดสอบตามขั้นตอนแล้วได้เอาท์พุตทางเทอร์มินัลดังรูปที่ 4-1 วงกลมหมายเลข 2 และ 3 นั้นแสดงถึงหน่วยความจำที่เคอร์เนลใช้ในการทำงาน ในหมายเลข 2 คือขนาดหน่วยความจำขนาด 404 กิโลไบต์สำหรับโค้ด และหมายเลข 3 คือขนาดหน่วยความจำขนาด 196 กิโลไบต์สำหรับข้อมูล รวมเป็นหน่วยความจำขนาด 600 กิโลไบต์สำหรับเคอร์เนล ส่วนวงกลมหมายเลข 1 แสดงหน่วยความจำที่เหลืออยู่และหน่วยความจำทั้งหมดที่ใช้ได้ 936 กิโลไบต์และ 1536 กิโลไบต์ ตามลำดับ เมื่อเปรียบเทียบกับขนาดหน่วยความจำของเคอร์เนลกับหน่วยความจำที่เหลืออยู่แล้วยังมีขนาดน้อยกว่ามากและขนาดที่เหลืออยู่ยังมีขนาดใหญ่มากสำหรับแอพพลิเคชั่นทั่วๆไป ถึงแม้ว่าหน่วยความจำทั้งหมดจะลดลงเหลือแค่ 1 เมกาไบต์ก็ตาม ขนาดของหน่วยความจำที่เหลือ คือ 424 กิโลไบต์นั้นก็ยังเพียงพอสำหรับแอพพลิเคชั่นทั่วๆไป

ในวงกลมหมายเลข 4 แสดงหมายเลขเมเจอร์ (Major Number) สำหรับอุปกรณ์ที่ทำการเมาท์เป็น Root File System นั้นคือ หมายเลขเมเจอร์ $1F_h$ (31) สำหรับอุปกรณ์ประเภทบล็อก (block) สำหรับหน่วยความจำแบบอ่านอย่างเดียว (ROM) ซึ่งเป็นอุปกรณ์ที่จำลองหน่วยความจำแบบอ่านอย่างเดียวเป็นอุปกรณ์ที่สามารถเข้าถึงได้สำหรับทำเป็น File System แบบ ROM File System แต่ในบรรทัดเดียวกันกับหมายเลขเมเจอร์นั้นมีการบอกว่าไม่สามารถเมาท์ได้นั้นไม่ได้แสดงว่าเคอร์เนลไม่สามารถทำงานได้ แต่เป็นการบอกความผิดพลาดทั่วไปของเคอร์เนล เพราะการคอนฟิกแบบ [*] System will boot from flash นั้นจำเป็นที่ต้องมีการนำเอา ROM File System ไปใส่ไว้ในโค้ดของเคอร์เนลด้วยแต่ในที่นี่ไม่ได้ใส่ไว้มีเพียงหน่วยความจำว่างๆเท่านั้นทำให้ไม่สามารถเมาท์ได้



รูปที่ 4-1 เอาท์พุทแสดงการทำงานของ uClinux

4.2 uClinux กับการให้บริการพื้นฐาน

การทดสอบในขั้นนี้เป็นการทดสอบว่า uClinux ที่สามารถทำงานบนบอร์ด AT91EB01 ได้แล้วนั้นสามารถให้บริการพื้นฐานของลีนุกซ์เดิมได้ อาจจะทำได้ด้วยการรเขียนโปรแกรมขึ้นมาเพื่อทำการทดสอบแต่ละกลุ่มการให้บริการเฉพาะซึ่งเป็นวิธีการทดสอบที่ทำให้ทราบผลแน่นอน ถึงแม้ว่าจะให้บริการได้อย่างสมบูรณ์แบบแล้ว แต่ก็จำเป็นที่ต้องนำโปรแกรมหรือแอพพลิชั่นต่างๆที่มีอยู่แล้วมาทดสอบด้วย ตามที่ได้บอกไว้ในข้อดีในการนำเอาลีนุกซ์ว่า มีโปรแกรมให้นำใช้งานในปัจจุบันอยู่มากช่วยลดเวลาในการพัฒนาได้ เนื่องด้วยระยะเวลาในการพัฒนาของโครงงานมีเวลาที่จำกัด การที่จะไปพัฒนาโปรแกรมขึ้นมาเฉพาะนั้นต้องใช้เวลามาก ดังนั้นในการทดสอบการให้บริการพื้นฐานนี้ จะเป็นการรนำโปรแกรมในรูปแบบต่างๆที่มีอยู่แล้วมาทำงานบนบอร์ด AT91EB01 โดยแบ่งเป็นการทดสอบเป็น 3 ขั้นตอนย่อย คือ

1. การทดสอบกับไลบรารี่
2. การทดสอบกับ shell

3. การทดสอบกับแอพพลิเคชั่น

4.2.1 การทดสอบกับไลบรารี่

เนื่องจากไลบรารี่ที่เตรียมมาให้เป็นไลบรารี่พื้นฐานที่ใช้ในการเขียนโปรแกรมกับ uClinux ซึ่งจะเป็นตัวกลางในการเรียกบริการขั้นพื้นฐานของ uClinux อีกที ถ้าฟังก์ชั่นที่มีให้ในไลบรารี่สามารถทำงานได้อย่างถูกต้องแล้วแสดงว่า uClinux สามารถให้บริการขั้นพื้นฐานได้

การทดสอบนั้นจะใช้โปรแกรมง่ายๆในการทดสอบตามกลุ่มฟังก์ชั่นต่างๆ ที่มีมาให้กับไลบรารี่ดังที่จะแสดงต่อไป ซึ่งสามารถทดสอบได้โดยนำโปรแกรมไปใส่ไว้ใน ROM File System Directory และตั้งให้เคอร์เนลรันโปรแกรมนั้นๆหลังจากที่เมาท์ file system ได้สำเร็จ แล้วดูผลลัพธ์ที่ได้ทางเทอร์มินัล (รายละเอียดในแต่ละขั้นตอนสามารถดูได้จากคู่มือการติดตั้งและใช้งานระบบลีนุกซ์แบบเรียลไทม์)

1. arg_test ทดสอบการดึงค่า arguments และ environment ที่ผ่านให้โปรแกรม
2. assert ทดสอบการทำงานของฟังก์ชั่น assert ที่แสดงเมสเสจถ้าเงื่อนไขเป็นเท็จ
3. ctype ทดสอบการทำงานของฟังก์ชั่นในกลุ่มการจัดการกับอักขระ ได้แก่ isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper
4. test_grp ทดสอบการทำงานของฟังก์ชั่น setgrent, getgrent และ endgrent ที่จัดการดึงค่าเกี่ยวกับ group จากไฟล์ /etc/group
5. test_pwd ทดสอบการทำงานของฟังก์ชั่น setpwent, getpwent และ endpwent ที่จัดการดึงค่าเกี่ยวกับ user จากไฟล์ /etc/passwd
6. setjmp_test ทดสอบการทำงานของฟังก์ชั่น setjmp และ longjmp ที่ใช้ในการ goto ที่ไม่เป็นแบบ local
7. signal ทดสอบการรับ signal
8. mallocbug ทดสอบบักของฟังก์ชั่น malloc ในการจัดการกับการขอใช้หน่วยความจำ
9. teststrtol ทดสอบการทำงานของฟังก์ชั่น strtol ที่แปลง char * (สตริง)ในฐานต่างๆตั้งแต่ฐาน 2-36 ให้เป็น long int
10. string, testcopy ทดสอบการทำงานของฟังก์ชั่นในกลุ่มการจัดการกับสตริง

หลังจากที่ได้ทดสอบแต่ละโปรแกรมแล้วได้ผลว่า โปรแกรมในข้อ 1-3 และ 6-9 สามารถทำงานได้อย่างถูกต้อง ส่วนโปรแกรมในข้อ 4-5 นั้นเมื่อทำการทดสอบไม่สามารถแสดงให้เห็นผลที่ต้องการ เนื่องจากไม่ได้เตรียมไฟล์ /etc/group และ /etc/passwd มาให้ และโปรแกรมในข้อ 10 สามารถทำงานได้แต่จำเป็นต้องมีการเปรียบเทียบผลลัพธ์ที่ได้กับไลบรารี่มาตรฐานตัวอื่น เช่น glibc เป็นต้น

1. arg_test จากการทดสอบสามารถแสดงค่าของค่า arguments และ environment ออกมาได้อย่างถูกต้อง แสดงว่าเคอร์เนลสามารถส่งผ่านค่า arguments และ environment มาให้โปรเซสที่สร้างขึ้นมาใหม่ได้

2. assert เป็นการทดสอบไลบรารี่มากกว่าเคอร์เนล เนื่องจากเป็นฟังก์ชั่นที่มีความเกี่ยวข้องกับเคอร์เนลน้อยซึ่งจะเป็นการส่ง signal แต่เรื่องของ signal จะมีการกล่าวในการทดสอบเฉพาะ
3. ctype เป็นการทดสอบฟังก์ชั่นการทำงานของไลบรารี่ทั้งหมด
4. test_grp เป็นการทดสอบฟังก์ชั่นการทำงานของไลบรารี่ทั้งหมด
5. test_pwd เป็นการทดสอบฟังก์ชั่นการทำงานของไลบรารี่ทั้งหมด
6. setjmp_test เป็นการทดสอบไลบรารี่มากกว่าเคอร์เนล
7. signal สามารถส่งและรับ signal ได้อย่างถูกต้องตามที่ได้ตั้งไว้
8. mallocbug เป็นการทดสอบระหว่างไลบรารี่ในการจัดการหน่วยความจำที่ขอใช้และเคอร์เนลในการให้พื้นที่หน่วยความจำ และในการทดสอบยังมีการรายงานความผิดพลาดที่ถูกต้องของเคอร์เนลเมื่อไม่สามารถให้พื้นที่หน่วยความจำได้
9. teststrtol เป็นการทดสอบฟังก์ชั่นการทำงานของไลบรารี่ทั้งหมด
10. string, testcopy เป็นการทดสอบฟังก์ชั่นการทำงานของไลบรารี่ทั้งหมด

ถึงแม้ว่าโปรแกรมส่วนใหญ่จะมีลักษณะและรูปแบบการเรียกใช้บริการเคอร์เนลที่ไม่ซับซ้อนมากนัก แต่จะมีบางบริการที่จะถูกเรียกใช้อยู่แล้วในทุกโปรแกรม คือ การจัดการเกี่ยวกับไฟล์ โดยเฉพาะไฟล์ที่เป็นเอาท์พุต เพราะทุกโปรแกรมต้องมีการแสดงผล

การทดสอบในขั้นนี้อาจจะไม่ซับซ้อนมากนักในการทดสอบบริการเคอร์เนล แต่ก็ไม่ได้หมายความไม่มีความจำเป็น เพราะโปรแกรมที่ใช้ในขั้นตอนการทดสอบต่อไปนั้นจำเป็นต้องใช้ฟังก์ชั่นของไลบรารี่อย่างมาก ส่งผลให้ขั้นตอนนี้เป็นขั้นตอนในการทดสอบไลบรารี่เพื่อเตรียมใช้งานต่อไปมากกว่าการทดสอบบริการเคอร์เนล

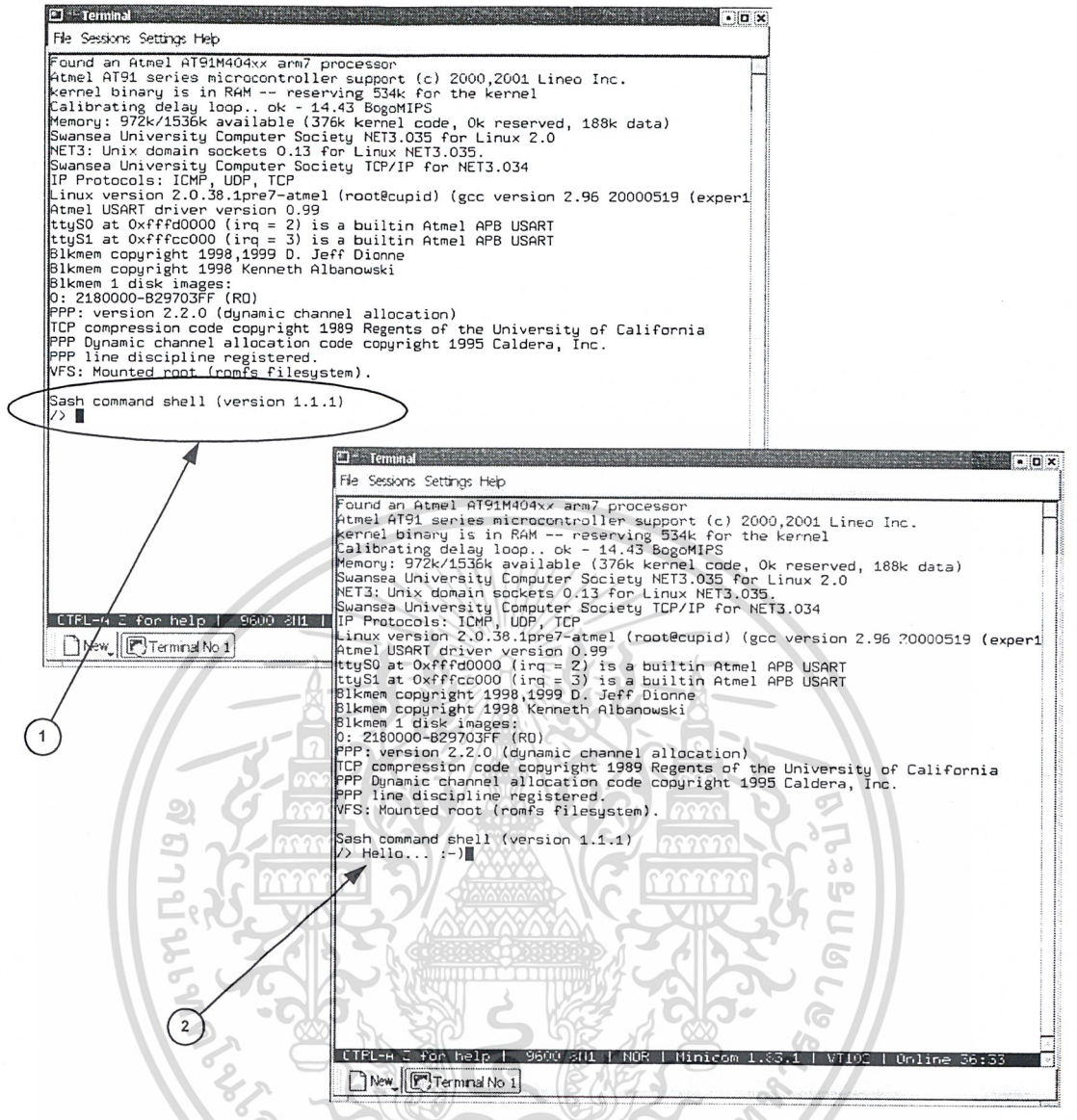
4.2.2 การทดสอบกับ Shell

หลังจากการทดสอบกับไลบรารี่ผ่านเรียบร้อยแล้ว ขั้นตอนต่อมาเป็นการทดสอบโปรแกรมที่จำเป็นอย่างยิ่งในการใช้ระบบปฏิบัติการยูนิกซ์ทั้งหลายรวมทั้งลีนุกซ์และ uClinux ด้วย นั้นก็คือโปรแกรม Shell โดยมีลักษณะการทดสอบเช่นเดียวกับขั้นตอนก่อนหน้าโดยดูที่ผลลัพธ์ทางเทอร์มินัล แต่เนื่องจากมีโปรแกรมเดียวและเป็นโปรแกรมที่สำคัญ จึงมีขั้นตอนในการทดสอบอยู่ 2 ขั้น คือ

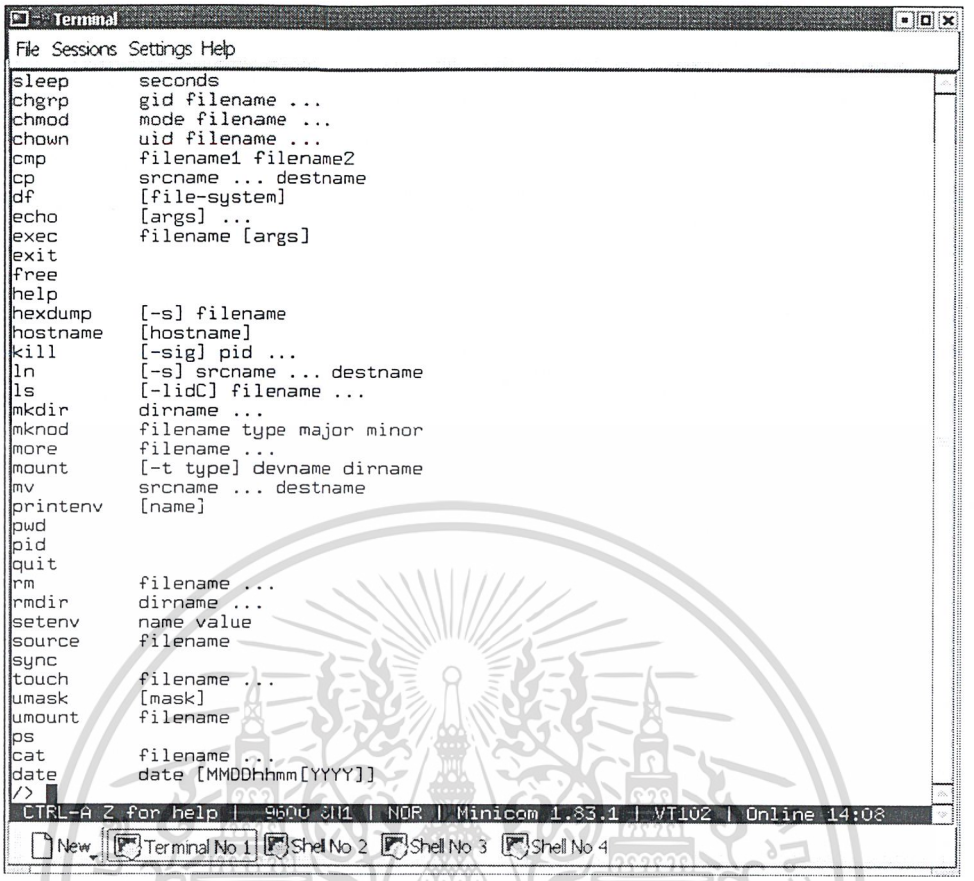
1. การทดสอบว่า Shell สามารถรับคำสั่งจากผู้ใช้ผ่านทางเทอร์มินัลได้
2. การทดสอบคำสั่งพื้นฐานที่มีมาให้กับ Shell ดังที่จะแสดงต่อไปทั้งหมด
   - cd เปลี่ยนไดเรกทอรี่ปัจจุบัน
   - sleep หยุดการทำงานในหน่วยวินาที
   - chgrp แก้ไข group ของไฟล์
   - chown แก้ไขเจ้าของ (user) ของไฟล์
   - cmp เปรียบเทียบไฟล์สองไฟล์
   - cp ทำสำเนาไฟล์

- df รายงานขนาดที่ใช้ไปและเหลืออยู่ของ file system
- echo แสดงกลุ่มอักขระ
- exec รันโปรแกรมอื่นขึ้นมาแทน
- exit ออกจาก shell
- free รายงานขนาดที่ใช้ไปและเหลืออยู่ของหน่วยความจำ
- hexdump แสดงข้อมูลในไฟล์แบบฐานสิบหก
- hostname แสดงชื่อเครื่อง
- kill ทำลายหรือส่ง signal ไปให้โปรเซส
- ln ลิงค์ไฟล์
- ls แสดงไฟล์และหรือรายละเอียดในไดเรกทอรี่ปัจจุบัน
- mkdir สร้างไดเรกทอรี่
- mknod สร้างไฟล์สำหรับอุปกรณ์ (device file)
- more แสดงข้อมูลในไฟล์แบบอักขระทีละหนึ่งหน้าจอ
- mount เมาท์ file system ไปยังไดเรกทอรี่ที่กำหนด
- umount เลิกเมาท์ file system ที่ได้ทำการเมาท์ไว้
- mv ย้ายหรือเปลี่ยนชื่อไฟล์
- printenv แสดง environment ปัจจุบัน
- pwd แสดงไดเรกทอรี่ปัจจุบัน
- pid แสดง id ของโปรเซสปัจจุบัน
- quit ออกจาก shell
- rm ลบไฟล์
- rmdir ลบไดเรกทอรี่
- setenv ตั้งค่า environment
- source รันโปรแกรมตามที่อธิบายในไฟล์
- sync flush บัฟเฟอร์ของ file systems
- touch เปลี่ยน timestamp ของไฟล์มาที่วันเวลาปัจจุบัน
- umask กำหนด mask สำหรับ permission เวลาจัดการกับไฟล์
- ps แสดงโปรเซสที่ทำงานอยู่
- cat แสดงข้อมูลในไฟล์แบบอักขระ
- date แสดงและเปลี่ยนวันเวลาของระบบ

หลังจากที่ได้ทดสอบขั้นที่ 1 แล้วได้ผลปรากฏดังรูปที่ 4-2 ในรูปที่ชี้โดยวงกลมหมายเลข 1 แสดงให้เห็นว่า shell สามารถทำงานจนถึงขั้นรอรับคำสั่งได้แล้ว และในรูปที่ชี้โดยวงกลมหมายเลข 2 แสดงให้เห็นว่าสามารถป้อนคำสั่งเข้าไปให้ shell ได้



รูปที่ 4-2 การรอรับคำสั่งและการใส่คำสั่งให้กับ shell

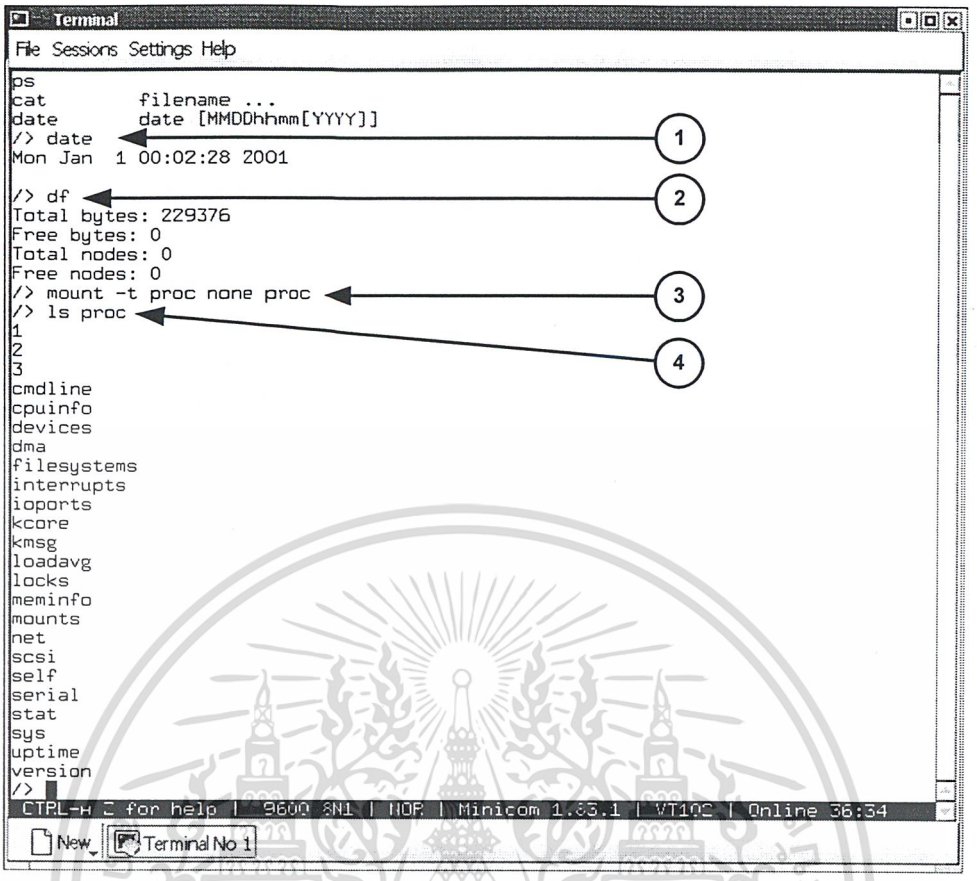


รูปที่ 4-3 คำสั่งภายในของ shell

เมื่อ shell พร้อมที่จะรอรับคำสั่งแล้ว ต่อไปก็ทดลองรันสั่งคำสั่งของ shell ทุกคำสั่งซึ่งสามารถเรียกดูได้จาก shell โดยใช้คำสั่ง help แสดงในรูปที่ 4-3 และในรูปที่ 4-4 แสดงการสั่งงาน shell บางคำสั่ง ดังนี้ วงกลมหมายเลข 1 คำสั่ง date เพื่อขอดูวันเวลาปัจจุบัน วงกลมหมายเลข 2 คำสั่ง df วงกลมหมายเลข 3 คำสั่ง mount เพื่อทำการเมาท์ proc file system ไปที่ /proc และวงกลมหมายเลข 4 คำสั่ง ls เพื่อแสดงไฟล์และไดเรกทอรี่ใน /proc

จากตารางที่ 4-1 ที่ประกอบด้วย 4 สดมภ์ ดังต่อไปนี้

1. คำสั่ง คือ คำสั่งภายใน shell ที่สามารถเรียกใช้ได้
2. เรียกใช้บริการของเคอร์เนล คือลักษณะการเรียกใช้บริการเคอร์เนลจากคำสั่งนั้นมีความเกี่ยวข้องมากหรือไม่ ซึ่งในจะเป็นตัวบอกว่าคำสั่งนั้นต้องสามารถทำงานได้อย่างถูกต้อง แต่ก็จะมีบางคำสั่งที่น่าจะมีการเรียกใช้บริการของเคอร์เนลแต่ไม่ได้เรียกใช้ เช่น free เป็นต้นเพราะเป็นอ่านข้อมูลจาก proc file systems (/proc) แทน
3. สามารถทดสอบได้ คือคำสั่งนั้นสามารถที่จะทำการทดสอบได้หรือไม่ เพราะบางคำสั่งต้องมีการเขียนลงไปใน file system ด้วยซึ่ง root file system ที่ใช้อยู่อ่านได้อย่างด้วย จำเป็นต้องมีการเมาท์ file systems อื่นอีกด้วย เช่น RAM file system เป็นต้น
4. ทดสอบผ่าน คือ คำสั่งนั้นเมื่อทำการทดสอบแล้วทำงานได้ถูกต้องหรือไม่



รูปที่ 4-4 ผลการสั่งงานบางคำสั่งของ shell

ตารางที่ 4-1 คำสั่งของ shell ที่ทำการทดสอบ

คำสั่ง	เรียกใช้บริการของเคอร์เนล	สามารถทดสอบได้	ทดสอบผ่าน
1. cd	X	X	X
2. sleep	X	X	X
3. chgrp	-	-	-
4. chown	-	-	-
5. cmp	-	X	X
6. cp	-	-	-
7. df	X	X	X
8. echo	-	X	X
9. exec	X	X	X
10. exit	-	X	X
11. free	-	X	X

คำสั่ง	เรียกใช้บริการของเคอร์เนล	สามารถทดสอบได้	ทดสอบผ่าน
12. kill	X	X	X
13. ln	X	-	-
14. ls	X	X	X
15. mkdir	X	-	-
16. mknod	X	-	-
17. more	-	X	X
18. mount	X	X	X
19. umount	X	X	X
20. mv	X	-	-
21. printenv	-	X	X
22. pwd	X	X	X
23. pid	X	X	X
24. quit	-	X	X
25. rm	X	-	-
26. rmdir	X	-	-
27. setenv	X	X	X
28. source	-	X	X
29. sync	-	X	X
30. touch	X	-	-
31. umask	-	-	-
32. ps	-	X	-
33. cat	-	X	X
34. date	X	X	X

การทดสอบในขั้นนี้เป็นส่วนที่สำคัญมากเนื่องจากบริการของเคอร์เนลที่แต่ละคำสั่งของ shell เรียกใช้นั้นเป็นบริการพื้นฐานของเคอร์เนลที่ต้องทำงานได้อย่างถูกต้อง มีบางคำสั่งที่ไม่สามารถทดสอบได้ด้วยเหตุผลที่บอกจำเป็นต้องมีการพัฒนาเพิ่มเติมอีก และมีคำสั่งหนึ่งที่เมื่อทดสอบแล้วไม่ผ่านนั่นก็คือคำสั่ง ps ดังนั้นจึงจำเป็นต้องเข้าไปดูในโค้ดแล้วพบว่ามีการคำนวณแบบทศนิยม และเมื่อทำการแก้ไนเป็นจำนวนเต็มแล้วคำสั่งก็สามารถทำงานได้ เมื่อแก้ไขแล้วทำงานได้จึงต้องมีการทดสอบเพิ่มว่าเป็นปัญหาที่การคำนวณแบบทศนิยมหรือไม่

จากปัญหาที่อาจจะเกิดจากการคำนวณแบบทศนิยมจึงได้ทดลองสร้างโปรแกรมขึ้นมาทดสอบโดยมีการคำนวณแบบเดียวกัน พบว่าไม่มีปัญหาอะไร ทำให้สามารถสรุปได้ว่าปัญญาที่เกิดขึ้นไม่ได้มาจากการคำนวณแบบทศนิยม แต่อาจจะเกิดจากการที่คอมไพเลอร์ต้องมีการรวมไลบรารี่แล้วส่งผลให้การจำลองการคำนวณแบบทศนิยมมีปัญหา เพราะเนื่องจากการโปรเซสเซอร์ที่ใช้ไม่มีตัวจัดการเกี่ยวกับเลขทศนิยม

4.2.3 การทดสอบกับแอพพลิชั่นพื้นฐาน

ถึงแม้ว่าการทดสอบโปรแกรม Shell แล้วจะสามารถทำงานได้ แต่ก็ไม่ได้หมายความว่า uClinux จะให้บริการขั้นพื้นฐานได้ทั้งหมด จึงจำเป็นต้องนำโปรแกรมอื่นที่จำเป็นต้องและนิยมใช้งานบนลีนุกซ์มาทำการทดสอบด้วย ซึ่งในที่นี้ได้นำโปรแกรมดังต่อไปนี้มาทำการทดสอบ

1. โปรแกรม atmel-init
2. โปรแกรม ping
3. โปรแกรม shutdown

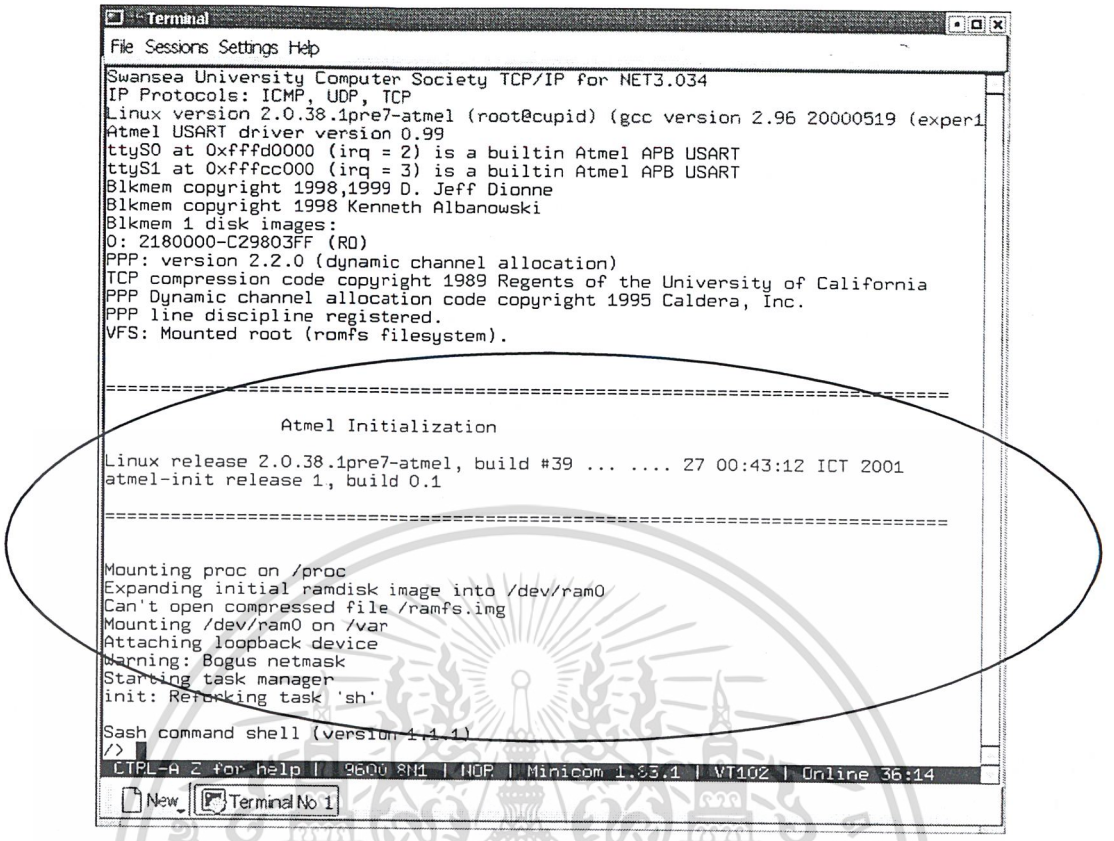
ในการทดสอบนี้ต้องทำการคอนฟิกเคอร์เนลให้รองรับเครือข่ายด้วย (Network support) ถ้าใช้เคอร์ตั้งแต่การทดสอบแรกนั้นก็ไม่จำเป็นต้องทำการคอนฟิกและคอมไพล์ใหม่ แต่เคอร์เนลนี้อาจจะมีขนาดใหญ่เกินไปทำให้ดาวน์โหลดช้า ดังนั้นควรทำการคอนฟิกให้รองรับเครือข่ายและค่าอื่นที่จำเป็นเท่านั้น เพื่อความรวดเร็วในการดาวน์โหลด

4.2.3.1 โปรแกรม atmel-init

โปรแกรม atmel-init เป็นโปรแกรมที่ทำหน้าที่เตรียมค่าต่างๆที่จำเป็นต้องใช้ในโปรแกรม ได้แก่

- การเมาท์ file systems เช่น proc FS, RAM FS เป็นต้น
- การเตรียม interface สำหรับอุปกรณ์เครือข่าย เช่น loopback, Ethernet เป็นต้น
- การเตรียม service สำหรับแอพพลิเคชั่นแบบเซิร์ฟเวอร์ เช่น telnetd, httpd, ftpd เป็นต้น
- การรันโปรแกรมทั่วไป เช่น shell เป็นต้น

เมื่อทำการทดสอบโดยให้ atmel-init รันเป็นแอพพลิเคชั่นแรก (First Application) นั้นได้ผลลัพท์ดังรูปที่ 4-5 ในวงกลมแสดงให้เห็นผลลัพท์ของโปรแกรมว่าสามารถทำการเมาท์ เตรียม inteface และรัน shell ได้อย่างถูกต้อง



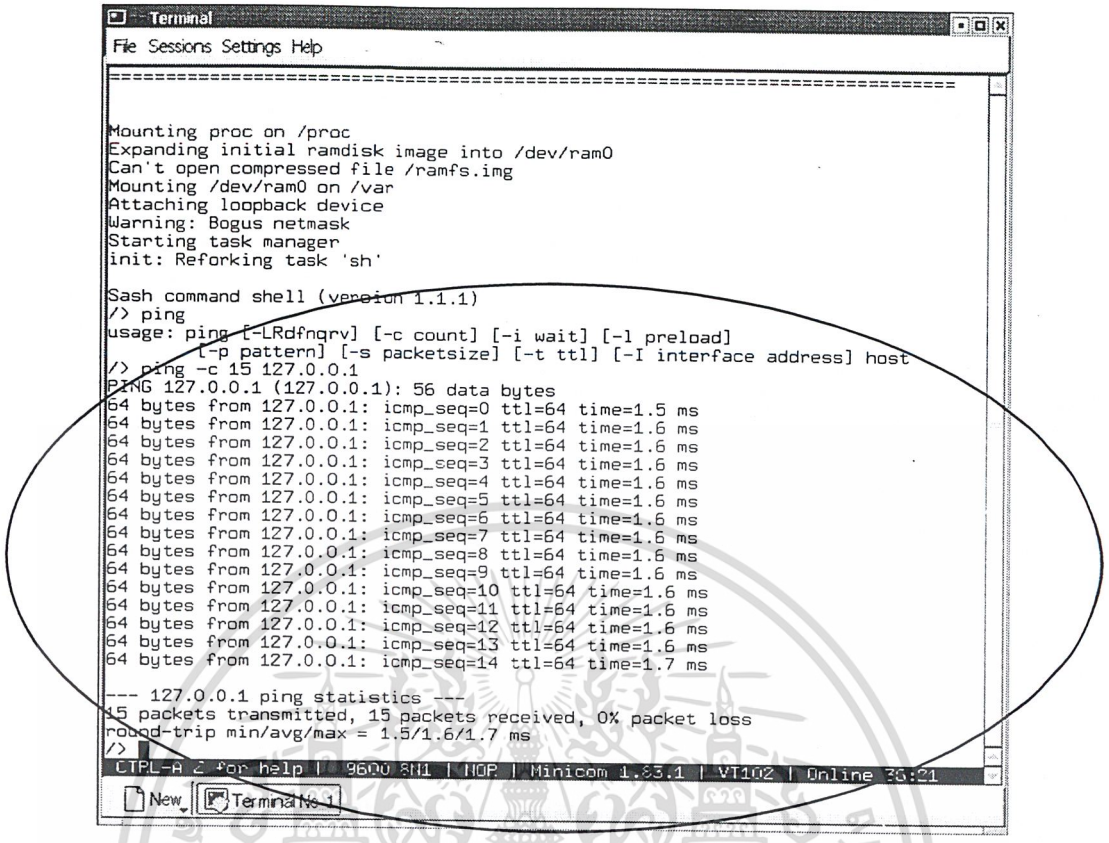
รูปที่ 4-5 โปรแกรม atmel-init

ในการทดสอบนี้ที่ใช้ atmel-init ซึ่งเป็นโปรแกรมที่มีการทำงานหลายประเภทนั้น แต่ละประเภทก็มีการเรียกใช้บริการของเคอร์เนลต่างๆกันไป ช่วยให้เป็นการทดสอบการใช้บริการของเคอร์เนลได้อย่างดี

4.2.3.2 โปรแกรม ping

โปรแกรม ping เป็นโปรแกรมที่ใช้การทดสอบการตอบรับทางเครือข่ายของอุปกรณ์เครือข่ายปลายทางในชุดโปรโตคอล TCP/IP ซึ่งเป็นโปรโตคอลพื้นฐานสำหรับอินเทอร์เน็ต โดยที่โปรแกรม ping จะทำการทดสอบอุปกรณ์เครือข่าย loopback ที่ได้จัดเตรียม interface ไว้แล้วโดยโปรแกรม atmel-init

ผลที่ได้จากการทดสอบโดยการ ping ไปที่ไอพี 127.0.0.1 ซึ่งเป็นไอพีของ loopback จำนวน 15 แพ็กเกจ และได้รับการตอบรับกลับมาจำนวน 15 แพ็กเกจเช่นกัน ดังผลลัพท์ในรูปที่ 4-6 ที่แสดงด้วยวงกลมอีกที



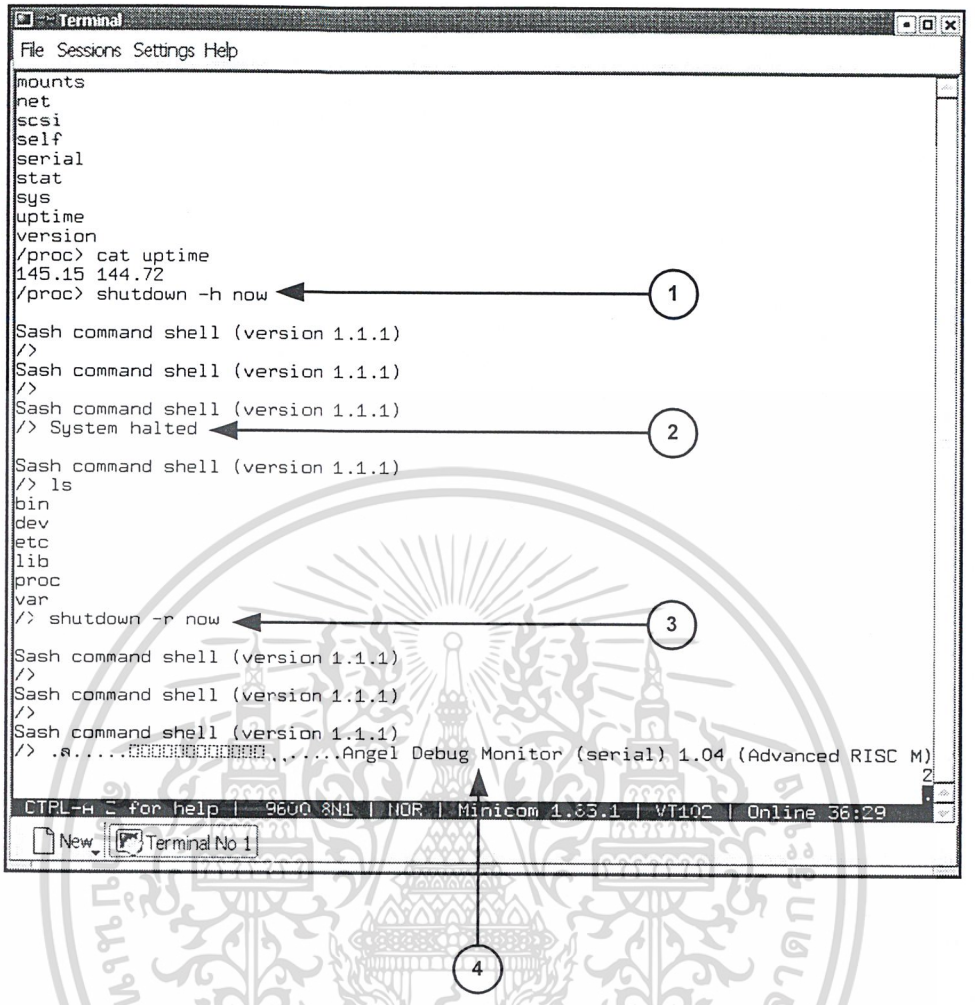
รูปที่ 4-6โปรแกรม ping

ถึงแม้ว่าการทดสอบนี้จะผ่าน แต่ก็เป็นเพียงการทดสอบบริการพื้นฐานทางด้านเครือข่ายระดับล่างเท่านั้น จำเป็นที่ต้องมีการทดสอบเพิ่มเติมอีกในระดับสูงขึ้นและกับอุปกรณ์เครือข่ายหลายๆประเภท ถึงแม้ว่าจะไม่ได้มีการทดสอบ แต่เนื่องจากส่วนนี้มีส่วนที่เกี่ยวข้องกับฮาร์ดแวร์น้อย ทำให้มีความเป็นไปได้สูงว่าบริการพื้นฐานทางด้านเครือข่ายจะทำงานได้อย่างสมบูรณ์ตามที่มีในลีนุกซ์เดิม

4.2.3.3 โปรแกรม shutdown

โปรแกรม shutdown เป็น โปรแกรมที่ไว้สำหรับยกเลิกการทำงาน (halt) หรือสั่งให้ระบบเริ่มทำงานใหม่ (reset) ถึงแม้ว่าโปรแกรมนี้จะมีการทำงานที่ไม่ซับซ้อนและมีการเรียกใช้บริการของเคอร์เนลน้อยมาก แต่โปรแกรมนี้ก็มีความจำเป็นในการใช้งาน

การทดสอบโปรแกรมนี้จะแบ่งออกเป็น 2 แบบ คือ halt หรือ reset ซึ่งผลลัพธ์จากการทดสอบแสดงไว้ในรูปที่ 4-7



รูปที่ 4-7 โปรแกรม shutdown

จากรูปที่ 4-7 วงกลมหมายเลข 1 คือคำสั่งในการ halt ระบบ และในวงกลมหมาย 2 แสดงว่าระบบได้ทำการ halt แล้ว แต่เมื่อลองป้อนคำสั่งเข้าไปให้ shell ปรากฏว่ายังสามารถทำงานงาน แสดงบริการนี้ของเคอร์เนลทำงานได้ไม่ถูกต้อง จำเป็นต้องมีการแก้ไขต่อไป วงกลมหมายเลข 3 คือคำสั่งในการ reset ระบบ และในวงกลมหมายเลข 4 เป็นผลลัพธ์ที่ได้จากโปรแกรมของตัวบอร์ด AT91EB01 เองเมื่อเริ่มทำงานใหม่ แสดงให้เห็นว่าบริการนี้ทำงานได้ถูกต้อง

4.3 uClinux กับงานแบบเรียลไทม์

ในส่วนของการพัฒนาเพิ่มความสามารถทางด้านเรียลไทม์ให้กับ uClinux โดยใช้ระบบลีนุกซ์แบบเรียลไทม์ที่เลือกไว้นั่นคือ RTAI นั้น ยังไม่สามารถเพิ่มความสามารถในส่วนนี้ได้ ซึ่งจำเป็นที่ต้องพัฒนาต่อไป แต่ได้ทำการทดสอบความสามารถของ uClinux ทางด้านเรียลไทม์ ว่าสามารถทำงานแบบเรียลไทม์ได้หรือไม่ โดยนำโปรแกรมทดสอบงานแบบเรียลไทม์ มารันบน uClinux ที่ทำงานบนบอร์ AT91EB01 ซึ่งรายละเอียดและโครงสร้างการทำงานของโปรแกรมที่นำมาทดสอบ สามารถดูได้จาก คู่มือการพัฒนาระบบลีนุกซ์แบบเรียลไทม์

การทดสอบความสามารถทางด้านเรียลไทม์ของ uClinux โดยนำโปรแกรมมารัน โดยโปรแกรมที่นำมาทดสอบมีการจัดลำดับงานโดยทำการรันงาน 2 งานในเวลาพร้อมกัน กำหนดความสำคัญของงานเพื่อให้งานที่มีลำดับความสำคัญมากกว่าสามารถทำการ preemptive งานที่มีลำดับความสำคัญต่ำกว่าได้

ผลที่ได้จากการทดลองคือ uClinux สามารถรันงานแบบเรียลไทม์ได้อย่างถูกต้อง โดยสนับสนุนลักษณะเด่นทางด้านเรียลไทม์ดังนี้

- สนับสนุนมาตรฐาน POSIX 4
- fixed priority
- สามารถทำงานหลายงานพร้อมกัน
- สนับสนุนการจัดลำดับงานแบบ preemptive

ถึงแม้ว่า uClinux สามารถทำงานแบบเรียลไทม์ได้ ก็ยังคงต้องพัฒนาระบบลีนุกซ์แบบเรียลไทม์ให้สามารถใช้งานกับ uClinux ต่อไปเพื่อสามารถทำงานแบบเรียลไทม์ได้หลากหลายและมีความถูกต้องมากขึ้น

บทที่ 5
บทวิจารณ์และสรุปผล

5.1 ผลที่ได้รับจากโครงงาน

1. ระบบ embedded คือบอร์ด AT91EB01 ที่สามารถรันระบบปฏิบัติการลีนุกซ์ซึ่งคือ uClinux ได้
2. uClinux ที่พัฒนาขึ้นสามารถทำงานบน บอร์ด AT91EB01 ที่ได้เพิ่มหน่วยความจำด้วยการ์ด AT91MEC01 มีคุณลักษณะเด่น ดังนี้
   - รันภายใต้โปรเซสเซอร์ บนสถาปัตยกรรมตระกูล ARM นั่นคือ ARM7TDMI
   - เปิดเผยโค้ด
   - ต้องการหน่วยความจำอย่าง 1 เมกาไบต์
   - ขนาดเคอร์เนลน้อยกว่า 600 กิโลไบต์
   - สนับสนุนการทำงานแบบหลายงานพร้อมกัน
   - สนับสนุน ROM Filesystems
   - Root Filesystem เก็บอยู่ภายในแรม
   - ขนาดหน่วยความจำที่สามารถใช้ในการทำงานขึ้นอยู่กับขนาดของ uClinux และ Root Filesystem ประมาณ 800kB – 1MB โดยทำการเพิ่มขนาดของหน่วยความจำโดยใช้ ใช้การ์ดขยายหน่วยความจำ AT91MEC01
   - อินพุตและเอาท์พุตผ่านทางพอร์ตอนุกรมไปยังเทอร์มินอล
   - ให้บริการ API มาตรฐานที่จำเป็นในการพัฒนางานทั่วไป โดยการให้บริการนี้ควรใช้ร่วมกับไลบรารี่ uClibc ด้วย
3. uClinux ที่สามารถทำงานบน บอร์ด AT91EB01 สามารถให้บริการพื้นฐานหลักๆของลีนุกซ์ที่ได้ทดสอบแล้ว ดังนี้
   - การจัดการเกี่ยวกับไฟล์และไดเรกทอรี่ รวมไปถึงการแสดงผลผ่านทางเทอร์มินัล
   - การจัดการเกี่ยวกับหน่วยความจำ
   - การจัดการเกี่ยวกับโปรเซส
   - การเมาท์ filesystems แบบ ROM และ proc
   - บริการทางด้านเครือข่ายระดับล่าง รวมไปถึงอุปกรณ์เครือข่าย ประเภท loopback
4. บทสรุปการประยุกต์ใช้งานลีนุกซ์กับระบบ embedded
5. บทสรุปการประยุกต์ใช้งานลีนุกซ์กับระบบเรียลไทม์
6. วิธีการพอร์ตลีนุกซ์ให้สามารถทำงานบนระบบ embedded ได้

5.2 ปัญหาที่พบ

โครงงานนี้มีส่วนที่จำเป็นต้องทำการศึกษาค้นคว้าอยู่มาก แต่เนื่องจากโครงงานในลักษณะนี้ยังมีผู้ที่พัฒนาน้อยและผู้พัฒนายังมีทักษะทางด้านนี้น้อย ส่งผลสะท้อนถึงปัญหาที่พบในโครงงานมีมากซึ่งสามารถแบ่งออกมา 3 ส่วนใหญ่ๆ คือ

- ปัญหากับระบบ embedded
- ปัญหากับ uClinux
- ปัญหากับระบบเรียลไทม์

5.2.1 ปัญหากับระบบ embedded

- เนื่องจากหน่วยความจำที่ใช้กับบอร์ด AT91EB01 เป็นแบบ SRAM มีขนาดหน่วยความจำได้สูงสุด 2M ซึ่งถ้าต้องการหน่วยความจำที่มีขนาดมากขึ้นจำเป็นที่ต้องเสียค่าใช้จ่ายในการพัฒนาที่เพิ่มขึ้นด้วย
- เครื่องมือการเชื่อมต่อผ่าน JTAG จำเป็นต้องสร้างขึ้นมาเอง
- ผู้พัฒนามีทักษะในด้านสถาปัตยกรรมของบอร์ด embedded ที่ใช้ในการพัฒนาน้อยเกินไป ส่งผลให้ต้องใช้เวลาในการเรียนรู้ค่อนข้างมาก
- การขาดทักษะของผู้พัฒนา เมื่อเกิดปัญหาขึ้นต้องใช้เวลาในการเรียนรู้และแก้ไขปัญหาค่อนข้างมาก

5.2.2 ปัญหากับ uClinux

- เนื่องจากยังไม่มี uClinux ที่ทำงานบนบอร์ด AT91EB01 ทำให้ต้องมีการศึกษาและเรียนรู้ในรายละเอียดของเคอร์เนลค่อนข้างมาก และศึกษาในส่วนที่จำเป็นสำหรับเพิ่มเติมความสามารถทางด้านเรียลไทม์ให้กับ uClinux ซึ่งต้องใช้ระยะเวลาในการศึกษาเรียนรู้อย่างมาก
- เนื่องจากข้อมูลเกี่ยวกับวิธีการพอร์ตและโครงสร้างของ uClinux และตัวลีนุกซ์เองยังมีน้อย ทำให้ต้องเสียเวลาในการพัฒนา uClinux ให้ทำงานบนบอร์ด AT91EB01 อย่างมาก
- ลักษณะในการพัฒนาแบบ cross-development ที่นำมาใช้กับระบบ embedded ที่มีสถาปัตกรรมไม่เหมือนกับสถาปัตยกรรมบนเครื่องคอมพิวเตอร์ที่ใช้พัฒนา ส่งผลให้ต้องมีการจัดเตรียมเครื่องมือที่จะนำใช้งาน ซึ่งในบ้างครั้งก็จะมีปัญหาที่เกิดขึ้นจากเครื่องมือนั้น เช่น ลักษณะโค้ดที่สร้างมาไม่ตรง หรือ ไม่สามารถแปลงรูปแบบไปเป็นแบบ flat ได้ เป็นต้น นั้นทำให้ต้องใช้เวลาเข้ามาศึกษา ทำการแก้ไขและคอนฟิกเครื่องมือเหล่านั้นด้วย
- วิธีการในการพัฒนาแอพพลิเคชั่นของระบบนี้ กับการพัฒนาจริงๆนั้น ยังมีความไม่เหมาะสมเนื่องจากต้องเสียเวลาในการเตรียมแอพพลิชั่นเพื่อไปทำการทดสอบบนบอร์ดค่อนข้างมาก

- ระยะเวลาในการดาวน์โหลดใช้เวลานานเกินไป โดยเฉพาะถ้า image มีขนาดใหญ่ ไม่เหมาะสมกับการพัฒนาที่ต้องมีการแก้ไขและทดสอบบ่อยโดยเฉพาะเคอร์เนล รวมถึงแอพพลิเคชั่นที่มีขนาดใหญ่ อย่างเช่น daemon นั้นยิ่งทำให้ต้องใช้เวลานานมากขึ้นไปอีก
- การทดสอบด้วยแอพพลิชั่นนั้นถึงแม้จะช่วยประหยัดเวลา แต่ก็ยังต้องเสียเวลามาศึกษาการใช้งานและเตรียมสิ่งที่จำเป็นต้องมีเพื่อให้แอพพลิเคชั่นสามารถทำงาน

5.2.3 ปัญหากับระบบเรียลไทม์

- RTAI ที่นำมาใช้ในการพัฒนาเป็นส่วนขยายเพิ่มเติมความสามารถในด้านเรียลไทม์นั้น ถึงแม้จะมีหลักการและลักษณะที่เอื้ออำนวยต่อการพัฒนาต่อ แต่เนื่องจากโค้ดในส่วนของ Interrupt Dispatcher นั้นยึดติดกับสถาปัตยกรรมของ x86 อย่างมาก ทำให้การพัฒนาต่อในส่วนนี้เป็นไปได้ยากพอสมควร จำเป็นต้องมีการสร้างขึ้นมาใหม่ และด้วยโครงสร้างของลีนุกซ์ ที่ใช้ในสถาปัตยกรรมของ ARM นั้นมีความแตกต่างกับ x86 ทำให้ต้องมีการแก้ไขเพื่อให้สามารถใช้งานกับสถาปัตยกรรมของ ARM

5.3 แนวทางการพัฒนาต่อ

5.3.1 ระบบ embedded

- ถ้าต้องการหน่วยความจำที่มากขึ้น แต่ระบบที่ได้ควรมีราคาที่ต่ำนั้นควรเปลี่ยนไปใช้โปรเซสเซอร์ที่มีการใช้ DRAM แทน และมีการพัฒนาลีนุกซ์ไว้แล้ว
- พัฒนา armtool ให้มีความสามารถในการดีบักกับ ARM7TDMI

5.3.2 uClinux

- ในขณะนี้ได้มีการพัฒนา uClinux จากเคอร์เนล 2.4 และได้จำกัดข้อจำกัดบางส่วนของ uClinux เดิมออก ดังนั้นการพัฒนาและใช้งาน uClinux ต่อไปจึงควรทำอยู่บน uClinux สำหรับเคอร์เนล 2.4
- ควรมีการทดสอบในส่วนของบริการทั่วไปของ uClinux ให้ครบมากกว่านี้ และโดยเฉพาะในส่วนของบริการเครือข่าย
- ในการทดสอบบริการของลีนุกซ์นั้นควรสร้างโปรแกรมขึ้นมาทดสอบโดยเฉพาะก่อน เพื่อให้แน่ใจว่า uClinux ทำงานได้ถูกต้องจริง เมื่อเวลานำแอพพลิเคชั่นมาทดสอบจะได้แน่ใจได้ว่าปัญหาที่เกิดขึ้นมาจากแอพพลิชั่น เนื่องจากเป็นการยากที่ทำความเข้าใจถึงการเรียกใช้บริการว่ามีอะไรบ้างและทำอย่างไร
- รูปแบบในการพัฒนาแอพพลิเคชั่นควรมีการปรับปรุงให้สะดวกและเหมาะสมในการพัฒนายิ่งขึ้น โดยอาจจะใช้ระบบของ Filsystem แบบ NFS ผ่านทางพอร์ตอนุกรม ซึ่งจะช่วยให้ไม่ต้องเสียเวลาในการดาวน์โหลด Filesystem ลงไป โดย NFS จะอาศัยไดเรกทรอรี่บนเครื่องพีซีเป็น Root Filesystem เมื่อพัฒนาแอพพลิเคชั่นเรียบร้อยแล้วก็แค่นำไปใส่ในไดเรกทรอรี่

ก็สามารถเรียกใช้งานโดย uClinux ที่ทำงานอยู่บอร์ดได้ทันที โดยไม่ต้องเริ่มการทำงานของระบบใหม่

- จำเป็นต้องมีการพัฒนาและทดสอบการใช้งานดีบักเกอร์รวมถึงแบบระยะไกล (remote debugging) เนื่องจากเป็นโปรแกรมที่จำเป็นอย่างมากในการพัฒนาแอพพลิเคชั่น

5.3.3 ระบบเรียลไทม์

- เนื่องจากในโครงงานยังพัฒนาในส่วนของ Interrupt Dispatcher ไม่เรียบร้อย จึงจำเป็นที่ต้องมีการพัฒนาให้เรียบร้อยก่อน จากนั้นค่อยไปแก้ไขโมดูลของ RTAI ในส่วนอื่นๆซึ่งมีส่วนที่ยึดติดกับสถาปัตยกรรม x86 น้อยกว่าเพื่อให้สามารถทำงานในส่วนของเรียลไทม์ได้อย่างสมบูรณ์ยิ่งขึ้น
- จัดสร้างโปรแกรมขึ้นมาทดสอบความสามารถทางด้านเรียลไทม์ในหลายรูปแบบ

5.4 สรุปผลโครงงาน

การพัฒนาระบบลีนุกซ์แบบเรียลไทม์ขึ้นมานั้น เริ่มจากทำการศึกษาและทำเข้าใจเกี่ยวกับระบบ embedded และระบบเรียลไทม์ เมื่อมีความเข้าใจในระบบทั้งสองแล้วจึงมาพิจารณาถึงโครงสร้างของระบบ เพื่อกำหนดเป้าหมาย และแนวทางการพัฒนา

โดยเริ่มจากการศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบเรียลไทม์ เพื่อหาบทสรุปของการนำเอาลีนุกซ์รองรับการทำงานแบบเรียลไทม์ในระบบ embedded และทำการศึกษาและค้นคว้าการประยุกต์ใช้งานลีนุกซ์กับระบบ embedded เพื่อการจัดหาบอร์ดแบบ embedded โดยมีเงื่อนไขที่สำคัญคือ ลีนุกซ์ ต้องสามารถทำงานบนบอร์ดนั้นๆได้

จากการศึกษาและค้นคว้าทำให้ได้ข้อสรุปของส่วนประกอบหลักที่จะนำใช้และพัฒนากับระบบลีนุกซ์แบบเรียลไทม์ ได้แก่ บอร์ด AT91EB01 ระบบปฏิบัติการ uClinux และส่วนขยายความสามารถทางด้านเรียลไทม์ RTAI เมื่อเราได้ส่วนประกอบหลักมาแล้วจำเป็นที่จัดหาเครื่องมือและส่วนประกอบที่จะนำมาใช้พัฒนา เช่น คอมไพเลอร์ โปรแกรมที่ใช้ในการติดต่อกับบอร์ด เป็นต้น

เมื่อได้ส่วนประกอบหลักมาครบแล้วก็เริ่มทำการพัฒนาในส่วนของ uClinux เพื่อให้สามารถทำงานบนบอร์ด AT91EB01 ได้ในขณะที่พัฒนาก็ทำการศึกษาและทดลองใช้งาน RTAI เพื่อนำหลักมาใช้ในการพัฒนาต่อไป หลังที่ได้พัฒนาส่วนของ uClinux เป็นที่เรียบร้อย จำเป็นที่จะต้องจัดหาโปรแกรมมาใช้งานบน uClinux เพื่อมาทำการทดสอบการให้บริการพื้นฐานต่างๆ เมื่อ uClinux มีความเสถียรภาพจึงทำการพัฒนาในส่วนขยายความสามารถทางด้านเรียลไทม์ให้กับ uClinux โดยใช้การพัฒนาบน RTAI แต่ในส่วนของการพัฒนานี้จำเป็นที่ต้องมีการพัฒนาต่อไปให้สามารถทำงานได้ เพื่อให้ระบบลีนุกซ์แบบเรียลไทม์ที่ทำงานบนระบบ embedded สามารถให้บริการกับงานทางด้านเรียลไทม์ได้คลอบคลุมมากยิ่งขึ้น

ผลที่ได้จากโครงงาน คือสามารถที่ทำการพอร์ตลีนุกซ์ให้สามารถทำงานบนระบบ embedded คือบอร์ดAT91EB01โดยมีคุณลักษณะเด่นดังที่ได้กล่าวไว้แล้ว แต่ยังไม่สามารถเพิ่มความสามารถทางด้าน

เรียลไทม์ให้กับ uClinux ด้วยข้อจำกัดบางอย่างของระบบลีนุกซ์แบบเรียลไทม์นั่นคือ RTAI ที่ทำการเลือกมาใช้ในการพัฒนา จำเป็น ที่ต้องใช้เวลาพอสมควรในการศึกษาและทำความเข้าใจระบบ รวมถึงวิธีการพอร์ต ให้สามารถใช้งานได้กับระบบ embedded จึงไม่สามารถพัฒนาในส่วนนี้ได้สำเร็จ แต่อย่างไรก็ตาม uClinux ที่พัฒนาให้ทำงานบนบอร์ด AT91EB01 ยังมีประสิทธิภาพในการให้บริการขั้นพื้นฐานของลีนุกซ์ สามารถนำไปพัฒนาต่อได้เป็นอย่างดี รวมทั้งปัจจุบัน มีการแก้ไขปรับปรุง uClinux .ให้มีความเสถียรมากขึ้น ทำให้สามารถทำการพอร์ตได้ง่ายขึ้น

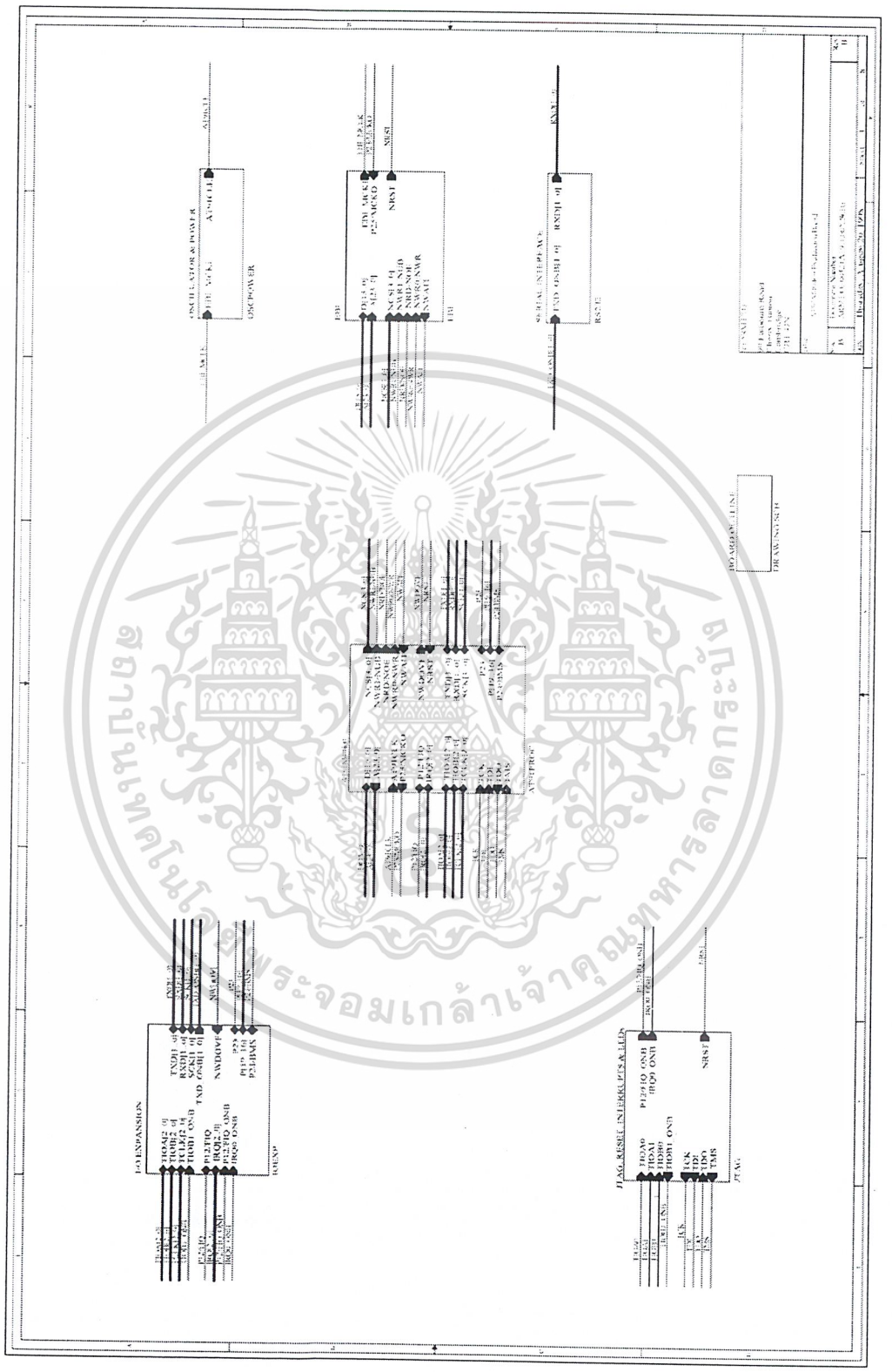
เมื่อผู้ใช้งานหรือผู้พัฒนาพิจารณาถึงผลงานที่ได้จากโครงงานนี้ อาจมองว่าผลงานที่ได้พัฒนาขึ้นนั้นยังน้อยอยู่ เพราะยังไม่สามารถทำการพัฒนาระบบในส่วนของการเพิ่มความสามารถทางด้านเรียลไทม์ให้กับ uClinux ได้สำเร็จ ตามเป้าหมายที่ได้วางไว้ เนื่องจากว่า การขาดความรู้ความชำนาญในด้านระบบลีนุกซ์แบบเรียลไทม์และวิธีการพอร์ตของผู้พัฒนาเอง ทำให้ในช่วงแรกของการพัฒนาต้องใช้เวลาในการศึกษาและทำความเข้าใจในเรื่องเหล่านี้เป็นอย่างมาก รวมถึงความยุ่งยากในการพอร์ต uClinux ให้สามารถใช้งานบนบอร์ดได้ จำเป็นต้องเข้าใจโครงสร้างและรายละเอียดส่วนต่าง ๆ ทั้งของ uClinux เองและของบอร์ดด้วย ทำให้ระยะเวลาที่ใช้ในการพอร์ตเพิ่มขึ้นเป็น 2 เท่า รวมถึงความเสี่ยงในการพัฒนามีสูง ต้องศึกษาและ ทำความเข้าใจในส่วนต่างๆ ให้ดีเสียก่อน เนื่องจากว่า หากบางโมดูลไม่สามารถทำงานได้อาจกระทบกับทั้งระบบได้ และในขั้นตอนการพัฒนาบางขั้นได้มีงานหรือขั้นตอนที่ไม่ได้คาดไว้ก่อนเพิ่มเติมเข้ามามาก ทำให้เราต้องเสียเวลาไปในส่วนที่ไม่ใช่การพัฒนาตัว uClinux จริงๆ

หากพิจารณาถึงระยะเวลาที่ใช้ในการพัฒนาทั้งส่วนของการศึกษาและทำความเข้าใจกับส่วนต่างๆที่จำเป็นในการพัฒนารวมถึงระยะเวลาที่ใช้ในการพอร์ตลีนุกซ์ เพื่อให้สามารถทำงานบนระบบ embedded ได้ ถึงแม้ว่ายังไม่สามารถพัฒนาในส่วนของเรียลไทม์ได้ในตอนนี้ ก็ยังถือว่า โครงงานนี้ได้ให้ประโยชน์ต่อผู้พัฒนาในการเรียนรู้และเข้าใจหลักการต่างๆ ของการพัฒนาระบบเป็นอย่างมาก หลักการและแนวทางบางอย่างที่ผู้พัฒนานำมาใช้ในการพัฒนาระบบ เกิดจากการค้นคว้าและหาข้อสรุปสำหรับหลักการและแนวทางนั้นๆ เพื่อเลือกวิธีการที่เหมาะสมในการพัฒนา รวมทั้งสามารถที่จะนำไปพัฒนาต่อไปได้เป็นอย่างดี

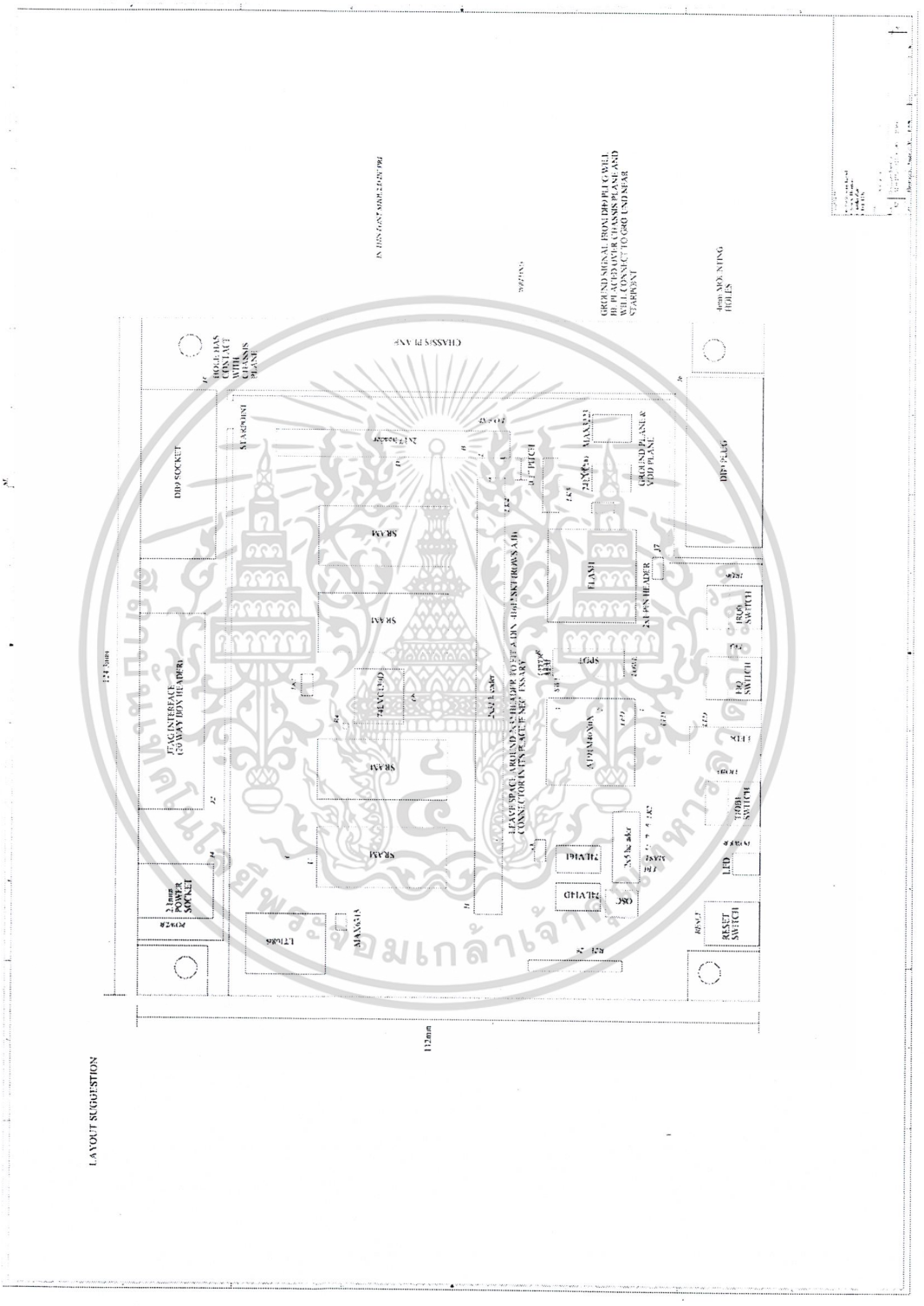
สุดท้ายผู้พัฒนามีความคาดหวังเป็นอย่างยิ่งว่า ความรู้ต่างๆ ที่ได้ทำการค้นคว้าและวิจัยออกมานั้นจะเป็นประโยชน์สำหรับผู้ที่สนใจในด้านนี้ไม่มากก็น้อย และสามารถเป็นแนวทางในการพัฒนาระบบลีนุกซ์แบบเรียลไทม์ต่อไป ถึงแม้ต้องใช้เวลานานในการศึกษาและค้นคว้าเกี่ยวกับหลักการและแนวทางที่ใช้ในการพัฒนา แต่ผลที่ได้รับกลับมาก็คุ้มค่ากับเวลาที่เสียไป

ภาคผนวก ก
ฮาร์ดแวร์ที่ใช้

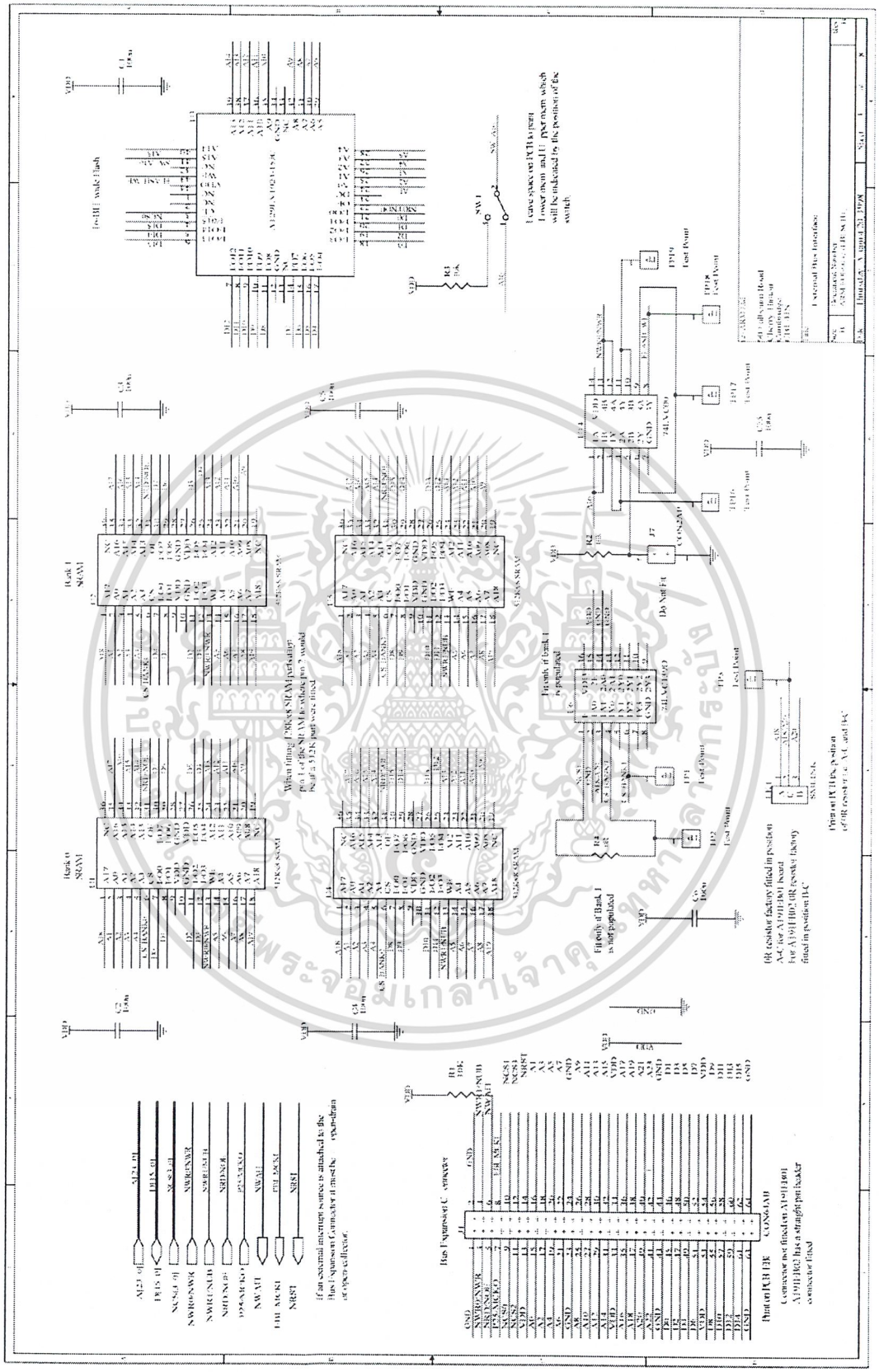
ก.1 AT91EB01's Schematics



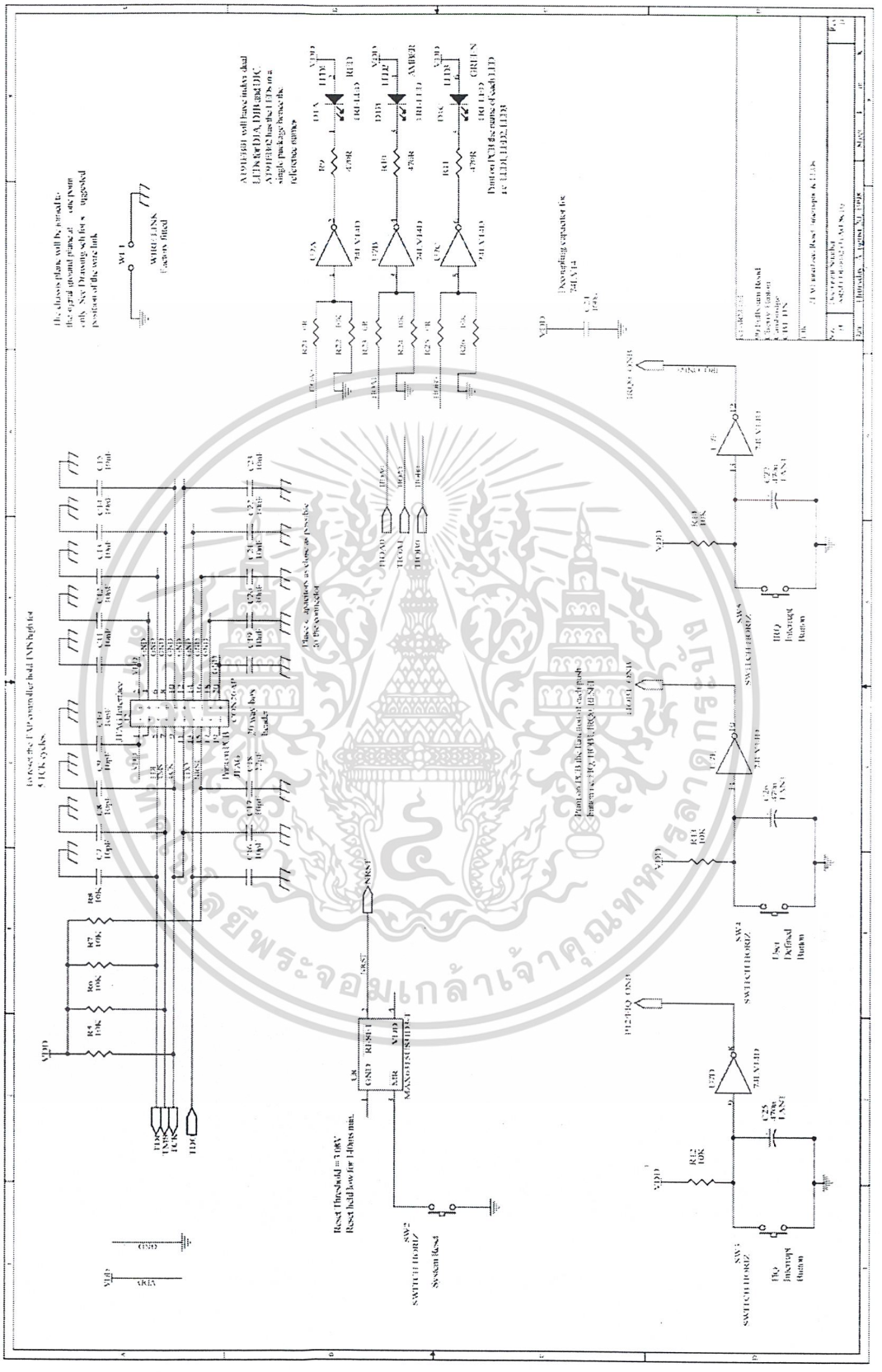
รูปที่ ก-1 ส่วนประกอบต่างๆภายในบอร์ด T91M40400



รูปที่ ก-2 PCB เลเอาต์

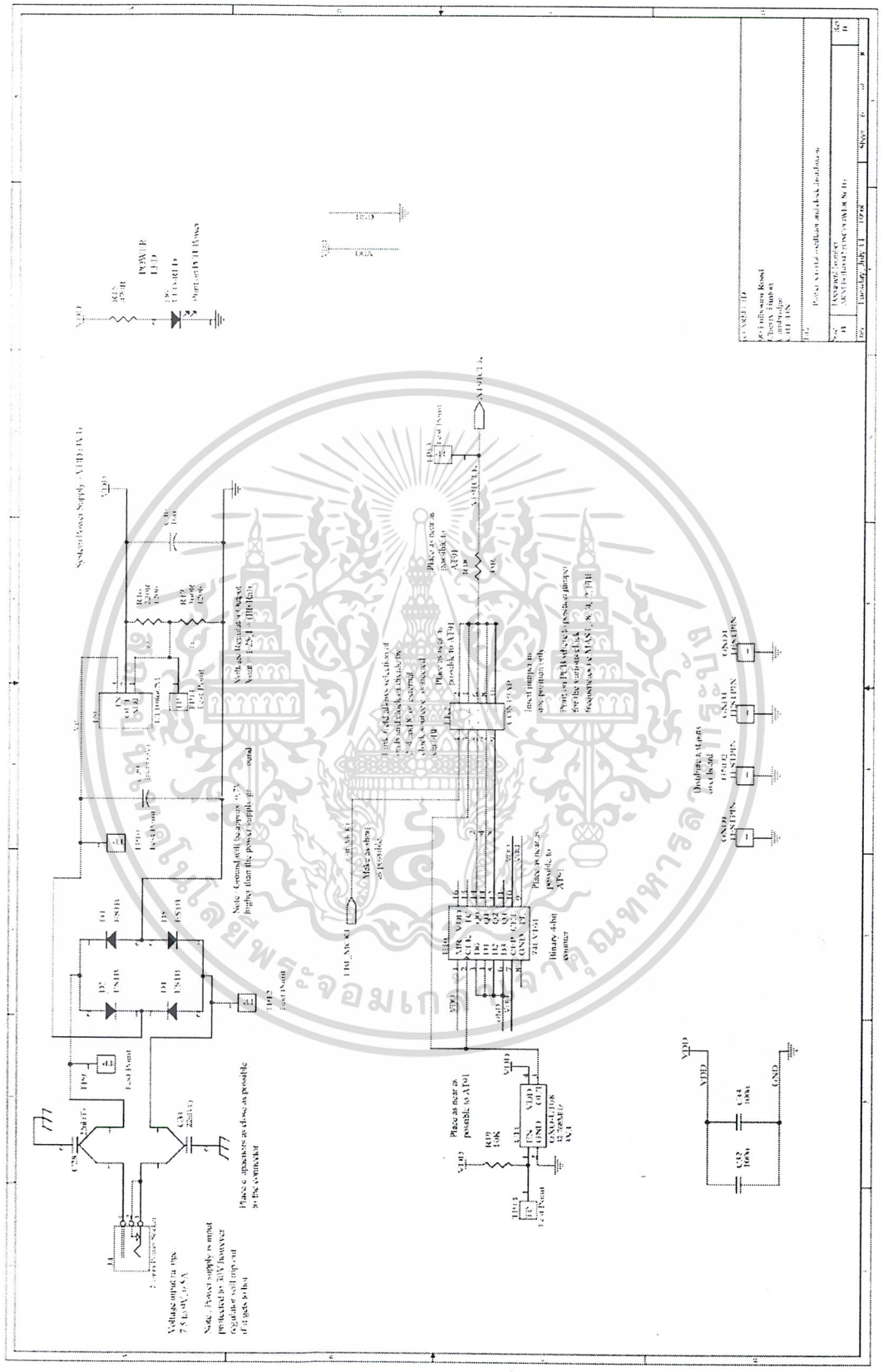


รูปที่ ก-3 แสดงการอินเตอร์เฟสกับบัสภายนอก

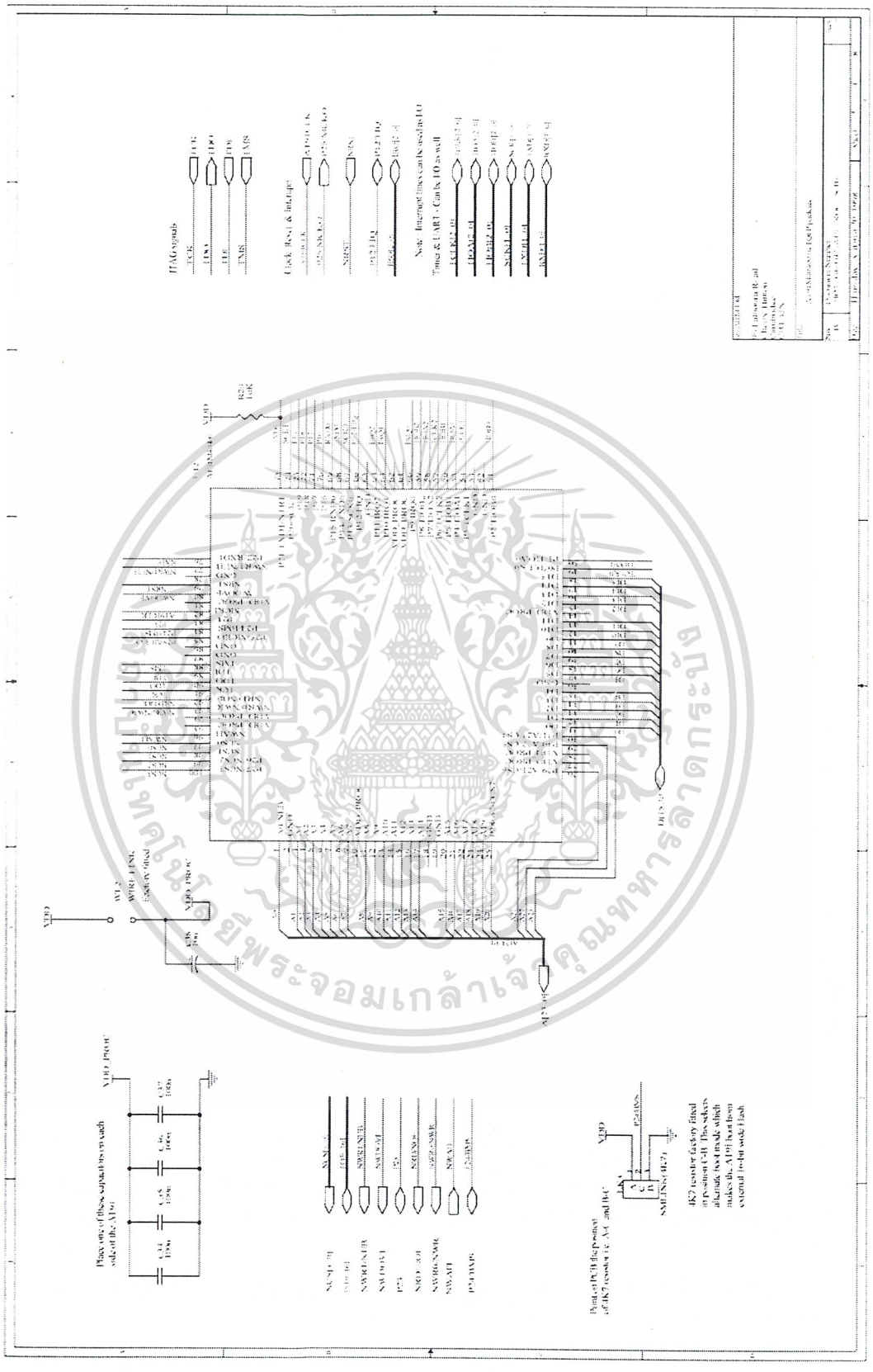


รูปที่ ก-4 JTAG อินเตอร์เฟส, รีเซ็ต, อินเตอร์รัพท์ และ LEDs

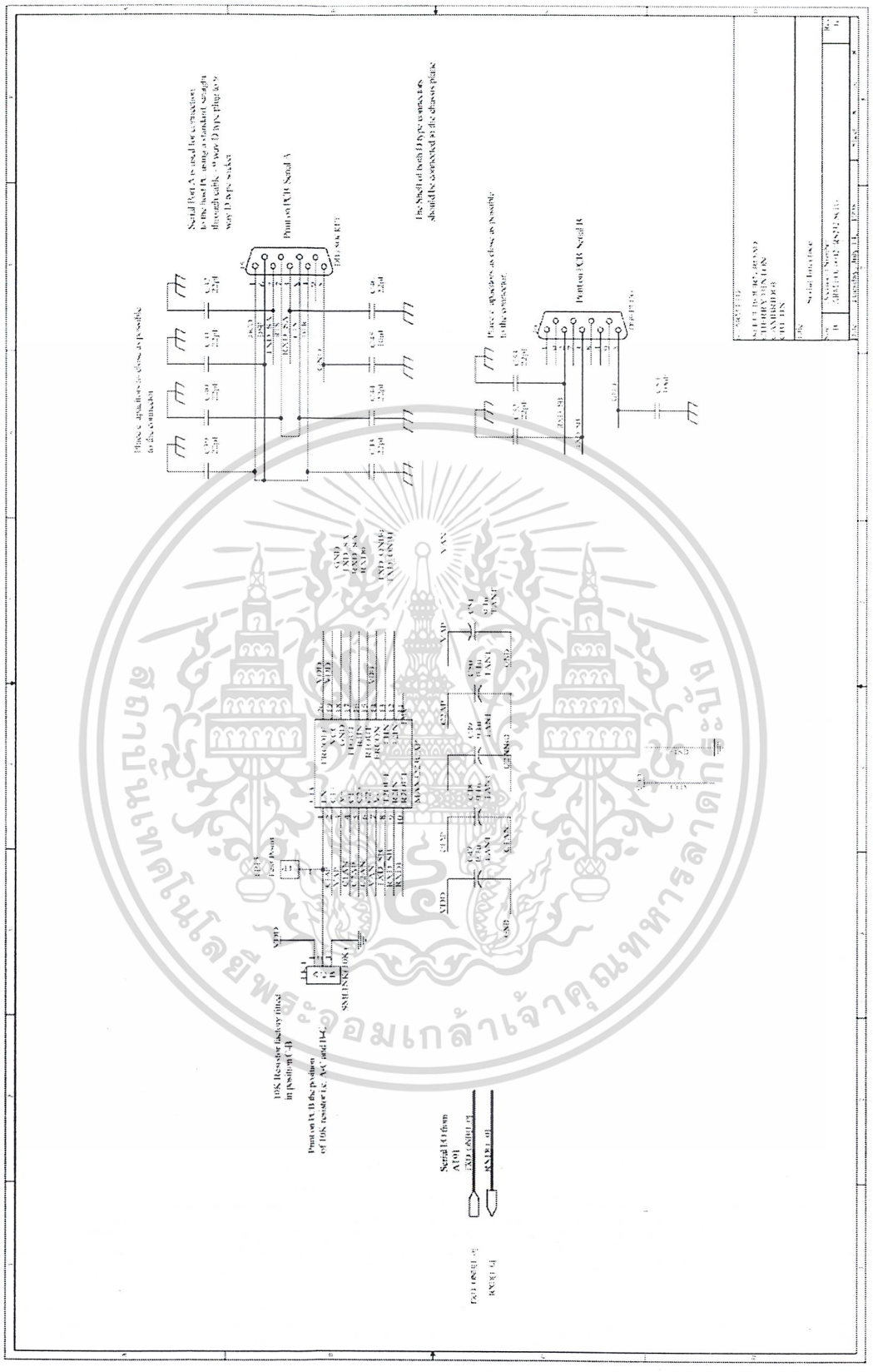
รูปที่ ก-5 แสดงส่วนขยาย I/O



รูปที่ ก-6 แสดงพาวเวอร์, คริสตัล ออสซิเลเตอร์ และ คล็อกดิสทริบิวชั่น

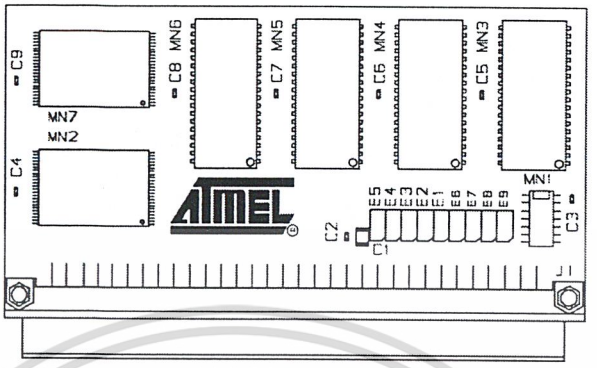


รูปที่ ก-7 แสดง AT91M40400 ในแพ็กเกจ TQFP

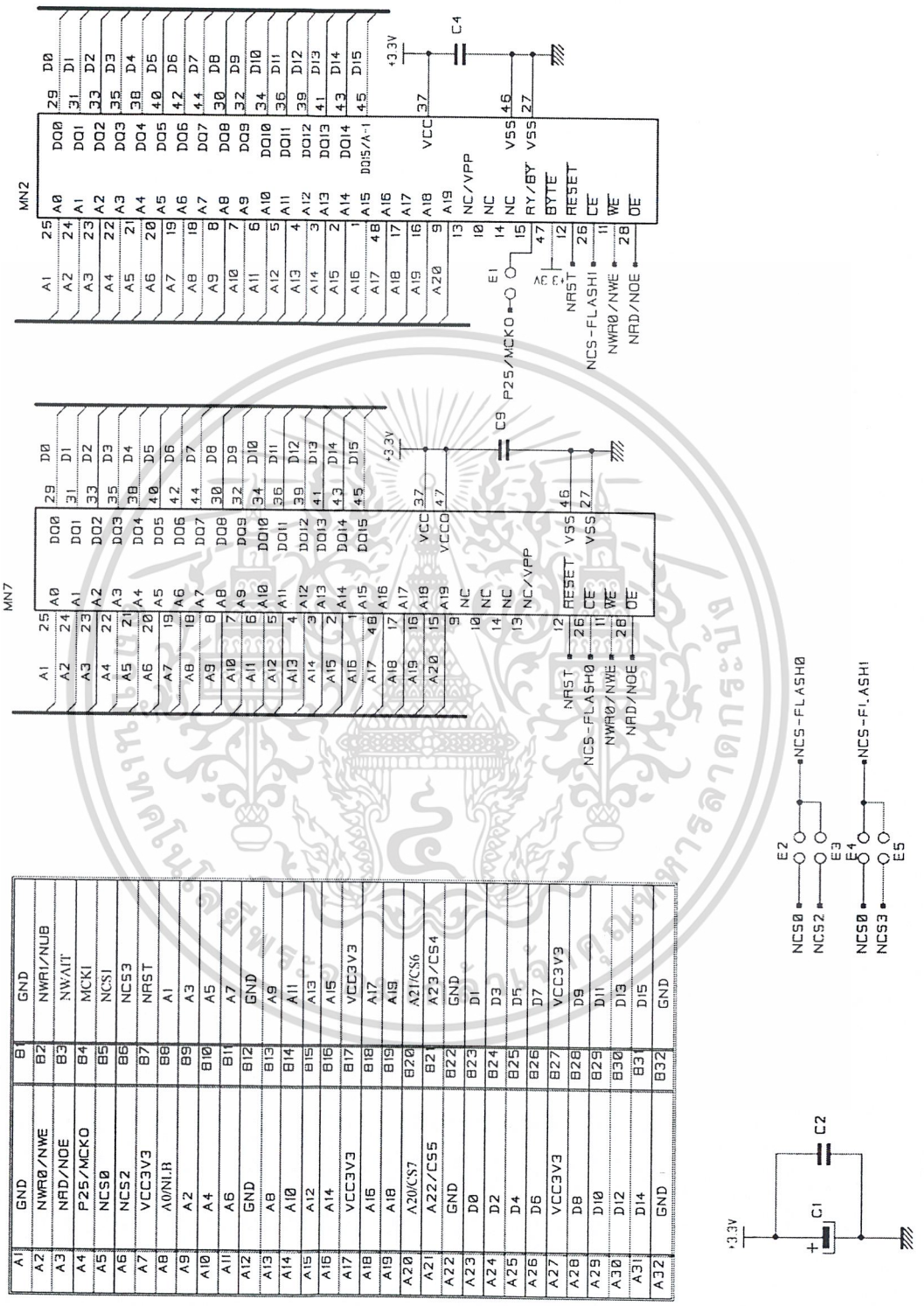


รูปที่ ก-88 แสดงซีเรียลอินเตอร์เฟส

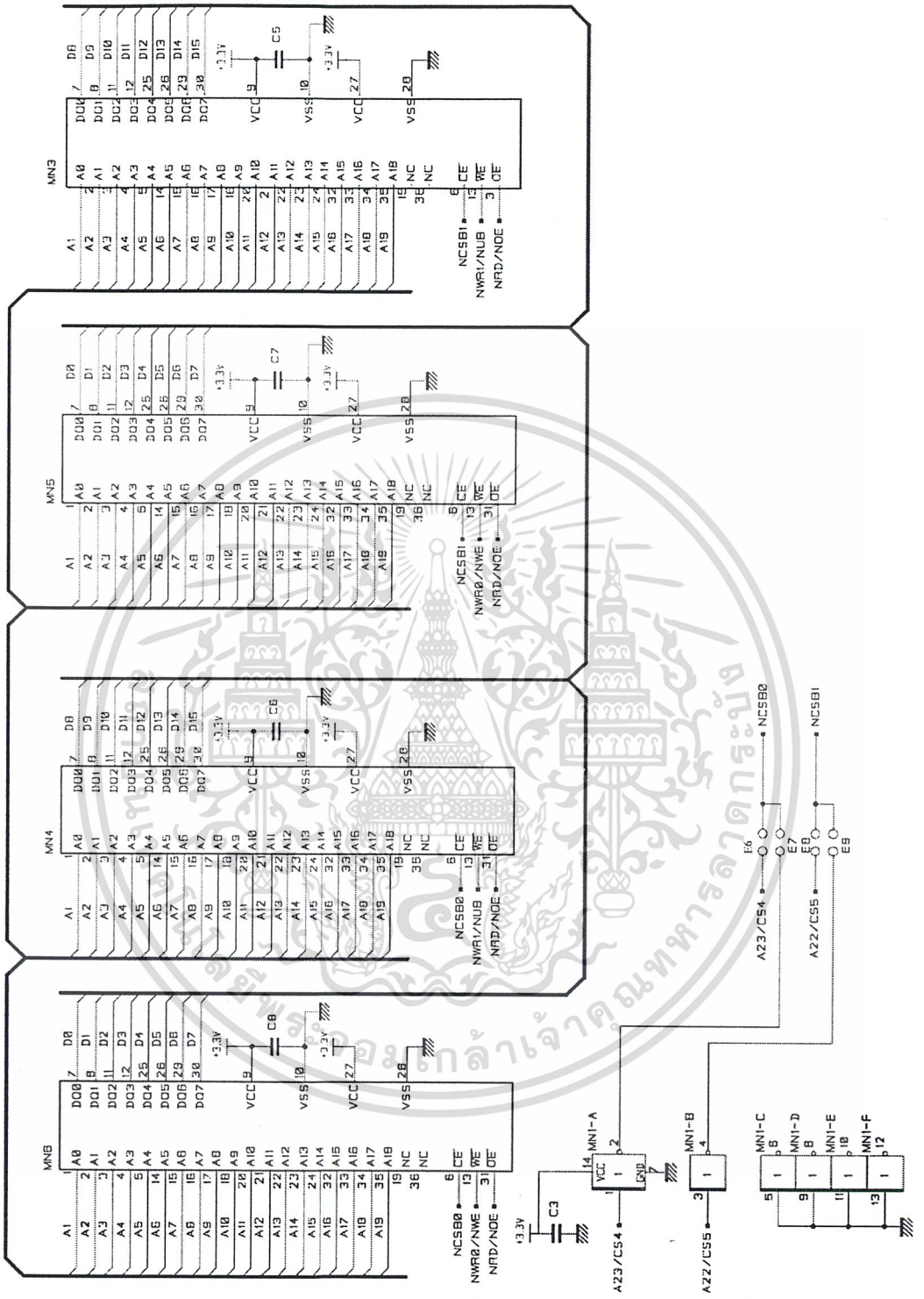
ก.2 AT91MEC01's Schematics



รูปที่ ก-9 แสดง PCB เลเอาต์



รูปที่ ก-10 แสดง EBI คอนเน็กเตอร์ และแฟลช



รูปที่ ก-11 ส่วนของ SRAM

บรรณานุกรม

[1] Scott Maxwell : *"LINUX Core Kernel Commentary"*,Arizona : Coriolis, 1999.

[2] Douglass, Bruce Powel : *"Real Time UML : developong Efficient Objects for Embedded System "*, Massachusetts : Addision Wesley, 1998

[3] Steve Heath: "*Embedded Systems Design*", Newnes, 1997

[4] C.M. Krishna and Kang G. shin : "*Real-time systems*", The McGraw-Hill, 1997

[5] Alessandro Rubini : "*Linux Device Drivers*", O'Reilly & Asscociates Inc:California, 1998

[6] Daniel P. Bovet & Marco Cesati : "*Understanding the Linux Kernel*", O'Reilly & Asscociates Inc:California, 2001

[7] Atmel ES2 : "*Atmel Corporation ARM7TDMI™ (Thumb®)* Datasheet January 1999", Advanced RISC Machines Limited (ARM), France, 1996

[8] Atmel ES2 : "*AT91 ARM® Thumb™ Microcontrollers AT91M40400*", Atmel, 1999