แพลตฟอร์มลินุกซ์ฝังตัวสำหรับหุ่นยนต์

EMBEDDED LINUX ROBOT PLATFORM

พิเชฐ มาภู่

อรรถสิทธิ์ อารยางกูร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง แพลตฟอร์มลินุกซ์ฝังตัวสำหรับหุ่นยนต์

EMBEDDED LINUX ROBOT PLATFORM

ผู้จัดทำ

1. นาย อรรถสิทธิ์ อารยางกูร รหัสนักศึกษา 47010958

2. นาย พิเชฐ มาภู่ รหัสนักศึกษา 47010514

_____อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์อภิเนตร อูนากูล)

แพลตฟอร์มลินุกซ์ฝังตัวสำหรับหุ่นยนต์

นายพิเชฐ มาภู่ 47010514
นายอรรถสิทธิ์ อารยางกูร 47010958
ผศ.อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

โครงงานนี้เป็นโครงงานที่พัฒนาแพลตฟอร์ม (platform) ที่ใช้ลินุกซ์ฝังตัว ซึ่งสามารถนำไปพัฒนาไปเป็นหุ่นยนต์ โดยที่แพลตฟอร์มนี้จะประกอบไปด้วยซอร์ฟแวร์เฟรมเวิร์ค แพลตฟอร์ม และฮาร์ดแวร์แพลตฟอร์ม โดยที่ผู้ใช้จะสามารถจะจัดการฮาร์ดแวร์ได้อย่างง่ายดาย อาทิ เช่น การรับ อินพุต การส่ง เอาท์พุต ของ จีพีไอโอ (GPIO), อินเตอร์รัพท์, การสร้างสัญญานพีดับเบิลยูเอ็ม (PWM), การสั่งการมอเตอร์ และเซอร์โวมอเตอร์, การรับค่าจากเซ็นเซอร์, การแสดงผลแอลซีดี (LCD) และรับอินพุตจากผู้ใช้งานผ่านทางหน้าจอสัมผัส (touch screen), มีลักษณะการพัฒนาส่วนประกอบต่างๆที่มีรูปแบบของเสียบแล้วเล่นได้เลย (plug and play) ผ่านพอร์ตยูเอสบี (usb) การติดต่อกับอุปกรณ์บันทึกโดยใช้ยูเอสบี หรือเอสดีการ์ด (SD-card), และการรับภาพจากกล้องเว็ปแคม (webcam) เป็นต้น โดยที่ใช้ซีพียูพีเอ็กซ์เอเอ็กซ์สเกล 270 (PXA xscale 270) เป็นหน่วยประมวลผลการทำงานซึ่งมีประสิทธิภาพสูงในการทำงาน และแพลตฟอร์มนี้ใช้ลินุกซ์เป็นระบบปฏิบัติการ และซอร์ฟแวร์อื่นๆที่ฟรีซึ่งทำหน้าในการจัดการทรัพยากรต่างๆ โครงงานนี้ยังครอบคลุมถึงการพัฒนาหุ่นยนต์ตัวอย่างที่มีการใช้ความสามารถต่างๆของแพลตฟอร์มนี้ได้อย่างหลากหลาย รวมไปถึงการทำเอกสาร และคู่มือการพัฒนาเพื่อรวบรวมองค์ความรู้ที่จะเป็นประโยชน์ต่อผู้ที่สนใจต่อไป

EMBEDDED LINUX ROBOT PLATFORM

Mr. pichet maphoo 47010514

Mr. Autthasith Arrayangkool 47010958

Asst. Prof. Apinetr Unakul Advisor

Academic Year 2007

ABSTRACT

This project is the project that develops embedded linux platform which can be developed and become to robot. This platform have software framework and users will can manage the hardware easily such as taking input and send output of GPIO , interrupt , building PWM signals , commanding motors and servo motors , taking value from sensor , showing information by graphic LCD , taking input from user by touch screen , development of component that have the format of plug and play by USB port , the connection with mass storage device by use USB or MMC and taking picture from webcam , etc. By using Intel xscale PXA270 is CPU which have high effective in the work . This platform use Linux that is freeware operating system which manage resources. This project cover example robot which is development that has using various ability of platform , including doing document and application node for collect the knowledge which will be advantage to developer and persons that they take an interest this project.

กิตติกรรมประกาศ

รายงานเล่มนี้ไม่อาจสำเร็จได้ด้วยดี หากขาดการช่วยเหลือ สนับสนุนและให้คำปรึกษาจาก ผศ. อภิเนตร อูนากูล ซึ่งเป็นผู้ควบคุมดูแลในการทำรายงานในครั้งนี้ ข้าพเจ้ารู้สึกทราบซึ้งในความอนุเคราะห์ของอาจารย์ และขอกราบขอบพระคุณท่านเป็นอย่างสูง

ขอกราบขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกๆท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า รวมไปถึงบุคลากรทุกท่านที่คอยอำนวยความสะดวกและประสานงานในเรื่องต่างๆ

ขอขอบคุณพี่ๆ ที่ทีซ่า (TESA : Thai Embedded System Association) ที่อนุเคราะห์คำแนะนำต่างๆ ในครั้งนี้เป็นอย่างสูง

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ที่คอยให้กำลังใจ และให้คำแนะนำที่ดีเสมอมา

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจ และให้การสนับสนุนช่วยเหลือในทุกๆเรื่อง

คุณค่าและประโยชน์อันพึงมาจากรายงานเล่มนี้ ข้าพเจ้าขอมอบให้แก่ผู้มีพระคุณทุกท่าน

นาย พิเชฐ มาภู่

นาย อรรถสิทธิ์ อารยางกูร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญภาพ

รูปที่ หน้า

สารบัญภาพ (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

เป็นที่ยอมรับกันว่าอุตสาหกรรมในปัจจุบันจะต้องมีเทคโนโลยีหุ่นยนต์เข้ามามีบทบาทให้การกำเนินการสร้างสรรค์ผลิตภัณฑ์ อาทิเช่น อุตสาหกรรมรถยนต์, อุตสาหกรรมอีเลคทรอนิกส์ เป็นต้น ซึ่งหุ่นยนต์เหล่านี้จะต้องมีประสิทธิภาพสูงในการควบคุม และจัดการงานด้านต่างๆ

ในการพัฒนาหุ่นยนต์ที่มีความซับซ้อน และให้มีประสิทธิภาพสูงนั้น มีความยากลำบาก เนื่องจากแพลตฟอร์มที่มีความสามารถสูงนั้นมีราคาสูง และจะต้องมีความรู้อย่างลึกซึ้งในการนำไปพัฒนาหุ่นยนต์ด้วยแพลตฟอร์มนั้นๆ โครงงานนี้จึงเข้ามาช่วยเพิ่มความสะดวก รวดเร็วในการพัฒนา และมีโมดูลต่างๆที่เกี่ยวข้องกับการพัฒนาหุ่นยนต์ เพื่อให้ผู้พัฒนาสามารถพัฒนาหุ่นยนต์ โดยใช้แพลตฟอร์มของโครงงานนี้ได้อย่างง่ายดาย

1.2 วัตถุประสงค์ของโครงงาน

- เพื่อสร้างแพลตฟอร์มที่นำไปใช้ในการพัฒนาหุ่นยนต์ได้
- สามารถพัฒนาเฟรมเวิร์คที่นำไปใช้ทำหุ่นยนต์ได้ง่าย มีประสิทธิภาพ สะดวก และลดเวลาในการพัฒนาหุ่นยนต์
- สามารถจัดทำเอกสารและคู่มือการใช้งานเพื่อรวบรวมองค์ความรู้ที่ได้จากการพัฒนาโครงงาน เพื่อเป็นประโยชน์แก่ผู้พัฒนาหุ่นยนต์ และผู้ที่สนใจต่อไป

1.3 ประโยชน์ที่คาดว่าจะได้รับ

- แพลตฟอร์มที่ได้จากโครงงานสามารถนำไปพัฒนาเป็นหุ่นยนต์ได้จริง
- มีความสะดวก รวดเร็ว และได้หุ่นยนต์ที่มีประสิทธิภาพมากขึ้นโดยใช้แพลตฟอร์มนี้
- ให้ผู้พัฒนา และผู้ที่สนใจ มีความรู้ความเข้าใจในการพัฒนาหุ่นยนต์มากยิ่งขึ้น

1.4 ขอบเขตของโครงงาน

รูปที่ 1.1 ขอบเขตของโครงงาน

โครงงานนี้จะศึกษาเกี่ยวกับรายละเอียดของการพัฒนาแพลตฟอร์ม และการนำความรู้ที่ได้ไปพัฒนาจริง ซึ่งแบ่งเป็น การพัฒนาด้านซอฟแวร์ และการพัฒนาด้านฮาร์ดแวร์

การพัฒนาด้านซอฟแวร์ อันได้แก่ กาพัฒนาแบบคลอสคอมไฟล์, การคอมไฟล์เคอร์เนล, การทำรูทไฟล์ซิสเต็ม, การพัฒนาดีไวซ์ไดร์เวอร์, การทำซอร์ฟแวร์โครงสร้างซอร์ฟแวร์ ไลบรารี่ และการพัฒนาแอปพลิเคชั่นโปรแกรม เป็นต้น

การพัฒนาด้านฮาร์ดแวร์ อันได้แก่ การออกแบบแผนผังวงจร, การออกแบบพีซีบี , การจัดหาและติดตั้งอุปกรณ์ , กาพัฒนายูเอสบีดีไวซ์, การติดต่ออุปกรณ์เซ็นเซอร์, การติดต่ออุปกรณ์ควบคุม, การติดต่อกับอุปกรณ์อินพุท เอาท์พุท และการทดสอบและแก้ไข เป็นต้น

โครงงานนี้ยังครอบคลุมถึงการพัฒนาหุ่นยนต์ตัวอย่างซึ่งใช้ทดสอบแพลตฟอร์ม และเรียกใช้งานส่วนต่างๆของของแพลตฟอร์ม นี้ได้อย่างหลากหลาย รวมไปถึงการทำเอกสารและคู่มือการใช้งานเพื่อรวบรวมองค์ความรู้ที่จะเป็นประโยชน์ต่อผู้ที่สนใจต่อไป

1.5 วิธีการดำเนินงาน

แบ่งเป็น การดำเนินงานด้านซอฟแวร์ และการพัฒนาด้านฮาร์ดแวร์

การดำเนินงานด้านฮาร์ดแวร์ มีขั้นตอนคือ

- การศึกษาตัวอย่าง , ความหมาย และลักษณะโดยรวมของโครงงาน
- วิเคราะห์หาฮาร์ดแวร์ที่จำเป็นในการพัฒนาหุ่นยนต์ในแพลตฟอร์ม

- ศึกษาการทำงานของฮาร์ดแวร์ส่วนต่างๆ แล้วออกแบบแผนผังวงจร
- ออกแบบพีซีบีและจัดทำ
- จัดหาอุปกรณ์อิเลคทรอนิกส์ต่างๆ แล้วทำการติดตั้งอุปกรณ์ลงในพีซีบี
- รวมเข้ากับการทำงานของซอร์ฟแวร์
- สร้างหุ่นยนต์ตัวอย่างเพื่อนำมาทดสอบแพลตฟอร์ม
- ทำการทดสอบและค้นหาข้อผิดพลาด
- จัดทำเอกสาร

การดำเนินงานด้านซอฟแวร์ มีขั้นตอนคือ

- การศึกษาตัวอย่าง , ความหมาย และลักษณะโดยรวมของโครงงาน
- ศึกษา และทำการคอมไฟล์เคอร์เนล, การทำรูทไฟล์ซิสเต็ม
- ทำการติดตั้งบอร์ด โดยประกอบด้วยการ ลง ยูบูท, ลินุกซ์เคอร์เนล, รูทไฟล์ซิสเต็ม
- ศึกษาการพัฒนาดีไวซ์ไดร์เวอร์
- ศึกษาฮาร์ดแวร์ของแพลตฟอร์มที่ใช้ในการเขียนดีไวซ์ไดร์เวอร์
- ออกแบบและพัฒนาดีไวซ์ไดร์เวอร์ที่ใช้ในการติดต่อฮาร์ดแวร์และพัฒนาแอปพลิเคชั่นที่มีในแพลตฟอร์ม
- ออกแบบ และพัฒนาซอร์ฟแวร์โครงสร้าง ไลบรารี่ และส่วนติดต่อกับดีไวซ์ไดร์เวอร์
- พัฒนาแอปพลิเคชั่นหุ่นยนต์ตัวอย่างที่ใช้แฟลตฟอร์มนี้
- รวมการทำงานเข้ากับฮาร์ดแวร์
- ทำการทดสอบและค้นหาข้อผิดพลาด
- จัดทำเอกสาร

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 การศึกษาตัวอย่าง , ความหมาย และลักษณะโดยรวมของโครงงาน

ในส่วนนี้เป็นการกล่าวถึงภาพรวมของโครงงานที่ได้ศึกษามา เพื่อให้เข้าใจถึงพื้นฐานสำคัญของการออกแบบและพัฒนาแพลตฟอร์มที่ใช้ในการสร้างหุ่นยนต์ ซึ่งตัวอย่างที่ได้เลือกใช้เป็นกรณีศึกษา คือ หุ่นยนต์เกราโช่ (Groucho) ซึ่งมีส่วนประกอบที่หลากหลาย และน่าสนใจจึงเหมาะที่จะนำมาศึกษา

2.1.1 ศึกษาตัวอย่างแพลตฟอร์ม

โดยส่วนประกอบที่สำคัญของหุ่นยนต์ตัวนี้ได้แก่

- บอร์ดไอเบส 890ซี เพ็นเทียมเอ็ม (iBase 890c Pentium M Motherboard) ซึ่งเป็นคอนโทรลเลอร์บอร์ดที่สามารถรันระบบปฏิบัติการได้รวมทั้งมีซีพียูเพ็นเทียมเอ็ม 755 (CPU Pentium M 755 2.0 GHz) และเอสดีแรม (SDRAM) 1 กิกะไบต์ ที่ผู้พัฒนาได้เลือกใช้บอร์ดนี้เพราะมีความเร็วซีพียูสูงช่วยในการประมวลผลออดิโอและวีดิโอได้ดี
- มี โน๊ตบุ๊คฮาร์ดไดร์ฟ(notebook hard drive) ขนาด 40 กิกะไบต์ ที่ใช้พลังงานต่ำ
- มีไมโครโฟน เพื่อทำการจดจำเสียง (voice recognition)
- มีลำโพง
- มี เอลซีดีทีเอฟทีมอนิเตอร์ขนาดเล็ก (small LCD TFT monitor) ใช้ในการตรวจสอบข้อผิดพลาดและ แสดงผลต่างๆ
- มีเครื่องตรวจจับระยะทาง (Devantech SRF08 sonar range finder) จำนวน 12 ตัวใช้ในการตรวจจับระยะห่างโดยติดต่อผ่านการสื่อสารแบบไอสแควซี (I2C)
- มีเครื่องตรวจจับอุณหภูมิ (Devantech TPA81 thermopile array) ใช้ในการตรวจจับอุณหภูมิ
- มีอินฟราเรด (Sharp IR Ranger) ในการตรวจจับวัตถุที่อยู่ด้านหน้าของหุ่น
- มี 2 ลีฟสวิทช์ (leaf switch) เพื่อตรวจจับวัตถุที่อยู่ต่ำๆ

- มีชุดอินเทอร์เฟส (Phidget Interface Kit 8/8/8) ที่มียูเอสบีดีไวซ์ที่จะใช้ในการอ่านค่าจากอินฟราเรดและลีฟสวิทช์
- มีไวร์เลสแลนการ์ด (Linksys 802.11g WiFi PCI Card : Wireless lan)
- ใช้ตัวถัง (Zagros Robotics) เส้นผ่าศูนย์กลาง 16 นิ้วมีมอเตอร์และเอ็นโค้ดเดอร์ติดตั้งภายใน (build-in encoder) ซึ่งมอเตอร์นี้มีกำลังสูงแต่ความเร็วต่ำเพื่อป้องกันอุบัติเหตุกับอุปกรณ์ภายในบ้านและเด็กเล็กๆ
- ใช้แบตเตอรี่ (sealed lead-acid : SLA) ขนาด 12 โวลต์, 12 แอมป์ชั่วโมง สำหรับมอเตอร์ และใช้แบตเตอรี่ 12 โวลต์ , 7 แอมป์ชั่วโมง จำนวน 2 ชุดเพื่อเป็นแหล่งจ่ายให้กับวงจรอิเล็กทรอนิกส์

2.1.2 ความหมายของหุ่นยนต์

"หุ่นยนต์คืออะไร" มีคำตอบมากมายสำหรับคำถามนี้ แต่สำหรับโครงงานนี้หุ่นยนต์หมายถึงโมบายออโตโนมัสเมชชีน (Mobile Autonomous Machine) ซึ่ง

- โมบาย (Mobile) หมายถึง สิ่งที่สามารถเคลื่อนที่ไปในสิ่งแวดล้อมต่างๆได้
- ออโตโนมัส (Autonomous) หมายถึง สิ่งที่ทำงานของมันโดยปราศจากการควบคุมจากมนุษย์โดยตรง
- โมบายออโตโนมัสเมชชีน (Mobile Autonomous Machine) จึงหมายถึง เครื่องจักรที่สามารถเคลื่อนที่และทำงานต่างๆในสภาพแวดล้อมนั้นๆได้โดยปราศจากการควบคุมของมนุษย์

โดยสภาวะแวดล้อมนี้จะเป็นตัวส่งผลถึงความซับซ้อนของตัวแมคคานิก เช่น ถ้าต้องการหุ่นยนต์ที่ทำงานในป่าโครงสร้างทางแมคคานิกจำเป็นต้องมีความซับซ้อนและแข็งแรงกว่าหุ่นยนต์ที่ทำงานบนพื้นบ้านเพื่อให้สามารถทำงานได้ เป็นต้น

2.1.3 ชนิดของหุ่นยนต์

โครงสร้างและชนิดของหุ่นยนต์ควรพิจารณาจากเป้าหมายและหน้าที่ของหุ่นยนต์ว่าจะสร้างขึ้นเพื่ออะไร ในที่นี้ได้แบ่งหุ่นยนต์ออกเป็น 3 ประเภท คือ

- หุ่นยนต์ทดลอง (Experimental robots) เป็นหุ่นยนต์ที่สร้างขึ้นมาเพื่อทดลองหรือทดสอบเพื่อพัฒนาต่อให้ดียิ่งขึ้น

- หุ่นยนต์ที่มีจุดประสงค์เฉพาะ (Special-purpose robots) เป็นหุ่นยนต์ที่สร้างขึ้นมาเพื่อทำหน้าที่อย่างใดอย่างหนึ่ง หุ่นยนต์ประเภทนี้จะมีจุดประสงค์เพียงอย่างเดียว เช่น หุ่นยนต์ซูโม่ เป็นต้น
- หุ่นยนต์เอนกประสงค์ (General-purpose robots) เป็นหุ่นยนต์ที่ทำงานในสภาวะแวดล้อมต่างๆ ได้ โดยไม่ได้มีงานหรือหน้าที่ตายตัว เช่น อาซีโม่ (asimo) เป็นต้น

2.1.4 โมบายออโตโนมัสโรบอท (Mobile Autonomous Robot)

โมบายออโตโนมัสโรบอท หมายถึง หุ่นยนต์ที่สามารถเคลื่อนที่ได้เองโดยไม่ต้องอาศัยการควบคุมของมนุษย์ ซึ่งการที่จะทำให้หุ่นยนต์มีความสามารถเช่นนี้ได้ต้องอาศัยส่วนที่สำคัญอยู่หลายส่วนด้วยกัน ซึ่งสามารถแบ่งความซับซ้อนของหุ่นยนต์เป็นระดับการประมวลผล (Processing Levels) โดยทั่วไปได้ดังนี้

การควบคุมโดยรวม (Global Control) เป็น โมดูลในระดับซุปเปอร์ไวเซอร์ (supervisor level module) มีหน้าที่จัดตารางเวลา (scheduling) และขั้นตอนของพฤติกรรมต่างๆ ซึ่งจะเป็นตัวกำหนดความซับซ้อนของหุ่นยนต์

การวางแผนโดยรวม (Global Planning) เป็นการวางแผนหรือตัดสินใจในระดับบนหรือระดับสัญลักษณ์ (symbolic level) เพื่อให้ไปถึงเป้าหมาย (goal) ที่ต้องการ เช่นตัดสินใจจะเคลื่อนที่จากตึกนี้ไปที่ถนน

การนำทาง (Navigation) เป็นการวางแผนหรือตัดสินใจในระดับล่าง เช่นตัดสินใจจะเคลื่อนที่ไปข้างหน้า หรือเลี้ยวซ้ายขวาตรงไหน

การทำเรียลเวอล์โมเดลลิ่ง (Real-World Modeling) แทนที่สิ่งต่างๆ ในโลกแห่งความเป็นจริงด้วยภาษาที่เป็นสัญลักษณ์ ซึ่งการแทนข้อมูลเหล่านี้สอดคล้องกับข้อมูลที่ได้รับจากเซ็นเซอร์และเอ็กชั่นที่หุ่นยนต์กระทำ

การรวมเซ็นเซอร์ (Sensor Integration) เป็นการรวมรวมข้อมูลที่ได้จากเซ็นเซอร์ต่างๆ ให้เป็นระบบของเซ็นเซอร์เพื่อจะนำไปประมวลผลหรือแทนค่าในแผนที่สัญลักษณ์ (symbolic map)

การแปลค่าจากเซ็นเซอร์ (Sensor Interpretation) เป็นการวัดค่าต่างๆ ของสิ่งแวดล้อมแล้วปรับแต่งให้เหมาะสมกับการประมวลผล

การควบคุมหุ่นยนต์ (Robot Control) เป็นการควบคุมการทำงานทางกายภาพของหุ่นยนต์ เช่น ควบคุมมอเตอร์ต่างๆ

โดยในแต่ละระดับจะประกอบด้วยหลายๆโมดูล ซึ่งจะมีความยืดหยุ่นตามแอปพลิเคชั่นของหุ่นยนต์นั้นๆ เช่น หุ่นยนต์ดูดฝุ่นอาจจะไม่ต้องมีความสามารถของการทำ การวางแผนก็ได้ แต่ถ้าเป็น พาหนะนำทางอัตโนมัติ (automatic guide vehicle : AGV) เรื่องการทำแผนที่ และการวางแผนถือเป็นเรื่องสำคัญ เป็นต้น

2.1.5 การออกแบบ

ส่วนนี้จะเป็นตัวอย่างการออกแบบหุ่นยนต์เกราโซ่ โดยผู้พัฒนาได้ทำการออกแบบโดยเริ่มจากการกำหนดสโคป กว้างๆของหุ่นยนต์ก่อน เช่น

- ประเภทของหุ่นยนต์ : หุ่นยนต์ทดลอง
- ความคงทนแข็งแรง : ปานกลาง
- ความคล่องตัว : สูง
- ความไวต่อการเปลี่ยนแปลง : สูง
- ความซับซ้อนทางกายภาพ : ต่ำ

เป็นต้น
จากนั้นจึงออกแบบภาพรวมของระบบและรายละเอียดของส่วนต่างๆ ถ้ามีชุดคิท หรือ บางส่วนที่มีอยู่แล้วก็สามารถนำมาดัดแปลงมาใช้งานได้ เพื่อประหยัดเวลาในการออกแบบและพัฒนา รวมถึงการจัดหาอุปกรณ์ต่างๆที่จำเป็นต้องใช้ ก่อนที่จะทำการสร้าง
ตัวอย่างการออกแบบส่วนต่างๆมีดังนี้

2.1.6 โครงสร้างทางแมคคานิก

ใช้ฐานเป็นวงกลมเส้นผ่านศูนย์กลาง 16 นิ้ว จำนวน 4 ชั้น แต่ละชั้นเชื่อมต่อกันด้วยท่อพีวีซีขนาด 3/8 นิ้วจำนวน 4 อัน แต่ละชั้นมีรูทรงกลาง 3 นิ้ว เพื่อใช้เป็นทางผ่านของสายไฟ ด้านล่างติดมอเตอร์จำนวน 2 ตัวในระนาบเดียวกัน แล้วติดล้อกับมอเตอร์ 2 ข้างเพื่อใช้ในการขับเคลื่อน และอีกระนาบเป็นล้อธรรมดา 2 ข้างเพื่อการทรงตัว เนื่องจากหุ่นยนต์ตัวอย่างนี้ไม่เน้นความสามารถทางกายภาพ โครงสร้างทางแมคคานิก จึงไม่ซับซ้อนมากนัก

2.1.7 คอนโทรลเลอร์บอร์ด

ส่วนมากหุ่นยนต์ขนาดเล็กๆ จะใช้การควบคุมจากไมโครคอนโทรลเลอร์ ซึ่งต่างจากระบบที่ซับซ้อนของหุ่นยนต์ตัวอย่างจึงต้องใช้ระบบปฏิบัติการเข้ามาช่วย

หุ่นยนต์เกราโซ่ ได้เลือกใช้ บอร์ดของไอเบส (iBase 890c Mini-ITX motherboard) และใช้ระบบปฏิบัติการเป็น เจ็นทูลินุกซ์เคอร์เนลเวอร์ชั่น 2.6 (Gentoo Linux with 2.6 kernel) ที่เลือกใช้เจ็นทูลินุกซ์ เพราะ ผู้พัฒนาต้องการระบบที่สามารถอัพเกรด ส่วนต่าง ๆ ได้ง่าย และสามารถใช้จาว่า 1.4 , ซี , ซีพลัสพลัส ในการรันซอฟแวร์ที่ใช้ควบคุมได้ด้วย นอกจากนี้ลินุกซ์ยังมีประโยชน์อีกมากมายเช่น

- ลินุกซ์ เป็นระบบปฏิบัติการแบบมัลติทาส์ค (multitasking operating system)
- ลินุกซ์ มีไลบรารี่เกี่ยวกับการติดต่อสื่อสารมากมาย
- ลินุกซ์ มีระบบเน็ทเวิร์คภายใน
- ลินุกซ์ มีประสิทธิภาพและง่ายต่อการทำงานประเภทวีดีโอและออดิโอ
- ลินุกซ์ มีดีไวซ์ไดร์เวอร์และส่วนสนับสนุนการใช้งานฮาร์ดแวร์ที่หลากหลาย
- ลินุกซ์ เป็นฟรีแวร์ (Freeware) สามารถนำมาใช้งานได้โดยไม่ต้องมีลิขสิทธิ์
- ลินุกซ์ มีขนาดเล็กเหมาะกับระบบที่มีทรัพยากรที่จำกัด

2.1.8 เซ็นเซอร์

ถ้ามอเตอร์เปรียบเสมือนกล้ามเนื้อเซ็นเซอร์ก็เปรียบเสมือนหู ตา หรือ อวัยวะที่ใช้ในการสัมผัสสิ่งต่างๆ รอบตัวเซ็นเซอร์แบ่งเป็น 2 ชนิดคือ

- เซ็นเซอร์ภายใน(Internal sensors) เป็นเซ็นเซอร์ที่ใช้ในการตรวจจับสิ่งต่างๆ ที่อยู่ภายในตัวหุ่น เช่น เซ็นเซอร์วัดความเร่ง (Accelerometer) เป็นต้น
- เซ็นเซอร์ภายนอก (External sensors) เป็นเซ็นเซอร์ที่ใช้ตรวจจับสิ่งต่างๆ ที่อยู่ภายนอกตัวหุ่น เช่น อินฟราเรด, เข็มทิศดิจิตอล เป็นต้น

โดยหุ่นยนต์เกราโซ่ได้ใช้เซ็นเซอร์ที่ใช้ในการตรวจจับระยะของวัตถุได้แก่ โซน่า, อินฟราเรด, สวิทช์, เซ็นเซอร์ที่ใช้ในการตรวจจับเสียง ได้แก่ ไมโครโฟน, เซ็นเซอร์ตรวจจับภาพ ได้แก่ เว็ปแคม, เซ็นเซอร์ตรวจจับความร้อน ได้แก่ เครื่องเปลี่ยนความร้อนเป็นไฟฟ้า (thermopile array)

2.1.9 สตอเรจ

เกราโซ่ ใช้ 40 กิกะไบต์แลปท็อปฮาร์ดไดร์ฟ (Laptop hard drive) ซึ่งในการใช้งานจริงอาจใช้คอมแพคแฟลช (compact flash) แทนได้เพื่อใช้ในการเก็บลินุกซ์และการติดตั้งคอมโพเน็นท์ต่างๆ รวมถึงการอัพเกรดระบบด้วย นอกจากนี้อาจใช้ รีมูฟเอเบิลฮาร์ดไดร์ฟ (removable hard drive) ซึ่งเชื่อมต่อผ่านยูเอสบีได้อีกด้วย

2.1.10 มอเตอร์

มอเตอร์เป็นชิ้นส่วนที่ทำให้เกิดการหมุน การเคลื่อนที่ และเกิดสิ่งรบกวน (ในที่นี้หมายถึงเสียงรบกวนและ สัญญาณไฟฟ้ารบกวน) ได้ในเวลาเดียวกัน นอกจากนี้มอเตอร์ยังมีลักษณะอื่นๆ ได้แก่

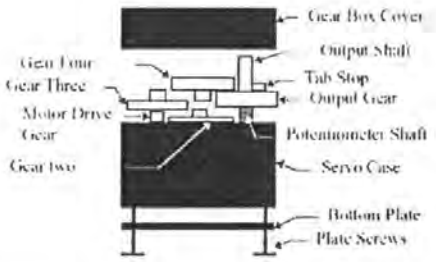
- ถ้ามอเตอร์ต่อกับล้อที่มีเส้นผ่านศูนย์กลางใหญ่ขึ้นจะทำให้แรงที่ขอบของล้อนั้นน้อยลง
- โวลต์เทจปรกติ (Nominal Voltage) เป็นโวลต์เทจ ที่ควรจ่ายให้กับ motor ตามปกติ แต่สามารถเพิ่ม โวลต์เทจ เข้าไปได้ แต่ข้อเสียคืออาจทำให้มอเตอร์ชำรุด
- รอบต่อนาที (RPM : Revolutions per minute) ใช้เป็นหน่วยวัดความเร็วของมอเตอร์มีทั้งที่วัดโดยไม่มี โหลด(load) และวัดในสภาวะที่มีโหลด
- ทอร์ค (Torque) เป็นแรงบิดของมอเตอร์ วัดจากรัศมีของล้อและแรงที่ผิว เช่น ถ้ามอเตอร์มีทอร์ค = 1 ฟุตปอนด์ (foot-pound) และมีรัศมีของล้อ 1 ฟุต แล้ว แสดงว่ามอเตอร์นี้มีแรงที่ขอบล้อเท่ากับ 1 ปอนด์
- ถ้ามีการทดเฟืองจะทำให้มอเตอร์มีรอบต่อนาทีต่ำลง แต่มีทอร์คเพิ่มขึ้น
- ควรมีตัวยึดหรือกรอบที่ใช้ใส่มอเตอร์ที่พอดีเพื่อความถูกต้องในการเคลื่อนที่

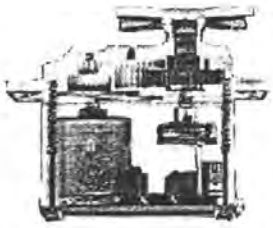
โดยมอเตอร์นั้นมีหลายชนิดแต่ในที่นี้จะอธิบาย 2 ชนิดที่นิยมนำมาใช้สร้างหุ่นยนต์ ซึ่งได้แก่

- **มอเตอร์กระแสตรง (DC motor)** เป็นมอเตอร์ที่ใช้ไฟฟ้ากระแสตรง (DC) เพื่อทำงาน ถ้าจ่ายโวลต์เทจน้อยมอเตอร์ก็จะหมุนช้า ถ้าความเร็วมากขึ้นมอเตอร์ก็จะหมุนเร็วขึ้น แต่ถ้าจ่ายเกินโวล์ตเทจปรกติอาจทำให้เกิดความร้อน การเกิดประกายไฟ (spark) หรือ อาจทำให้มอเตอร์พังได้ เราสามารถควบคุมความเร็วของมอเตอร์ได้โดยใช้พีคับเบิลยูเอ็มเป็นการควบคุมความเร็ว

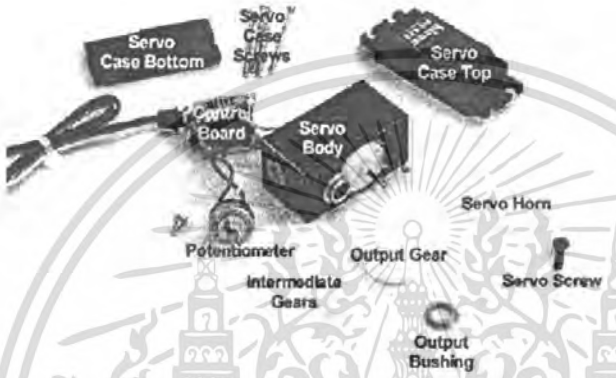
โดยใช้โวลต์เทจ ที่มีลักษณะเป็นพัลส์ (pulse) ซึ่งถ้าพัลส์มีดิวตี้ไซเคิล (duty cycle) มาก จะทำให้ความเร็วของมอเตอร์มาก

- เซอร์โวมอเตอร์ คือ มอเตอร์ไฟฟ้าแกระแสตรงที่ถูกประกอบรวมกับ ชุดเกียร์และส่วนควบคุมต่างๆไว้ในโมดูลเดียวกัน โดยมอเตอร์ชนิดนี้จะมีสายต่อใช้งานเพียง 3 เส้น คือ ไฟ (VCC), กราวด์ (GND) และ สายสัญญาณควบคุม(control line) ซึ่งสามารถควบคุมให้มอเตอร์หมุนซ้ายหรือ ขวาได้ จากสายสัญญาณเพียงเส้นเดียวโดยสัญญาณที่ใช้ควบคุมนี้จะเป็นสัญญาณพีดับเบิลยูเอ็ม แบบระดับทีทีเอล (TTL level) ระดับแรงดันที่จ่ายให้มอเตอร์นี้จะอยู่ในช่วงประมาณ 4-6 โวลต์ ขึ้นอยู่กับคุณสมบัติของมอเตอร์แต่ละตัว ข้อดีของมอเตอร์ชนิดนี้ก็คือ จะมีขนาดเล็กน้ำหนักเบา, ให้แรงบิดสูง, กินพลังงานน้อย และสามารถควบคุมด้วยสัญญาณลอจิกที่เป็นทีทีเอล ได้โดยตรงไม่จำเป็นต้องต่อวงจรขับอื่นๆ เพราะมอเตอร์ชนิดนี้จะมีวงจรควบคุมบรรจุไว้ภายในอยู่แล้ว ซึ่งมอเตอร์ชนิดนี้สามารถควบคุมให้หมุนไปในตำแหน่งหรือทิศทางที่ต้องการได้ โดยอาศัยสัญญาณความกว้างพัลส์ ที่ป้อนให้มอเตอร์ แต่เซอร์โวมอเตอร์นี้จะหมุนได้แค่เพียง 180 องศา หรือครึ่งรอบเท่านั้น หรือบางรุ่นอาจหมุนได้ถึง 210 องศา แต่จะไม่สามารถหมุนเป็นวงรอบได้เนื่องจากโครงสร้างภายในจะประกอบด้วย ตัวต้านทานชนิดปรับค่าได้ ที่ทำหน้าที่ตรวจสอบตำแหน่งการหมุนของของมอเตอร์ และตัวต้านทานนี้จะถูกยึดติดกับแกนหมุนของมอเตอร์ และตัวต้านทานนี้ไม่สามารถหมุนเป็นวงรอบได้ ดังนั้น เซอร์โวมอเตอร์จึงถูกออกแบบให้หมุนได้เพียงแค่ประมาณ 180 องศาเท่านั้น



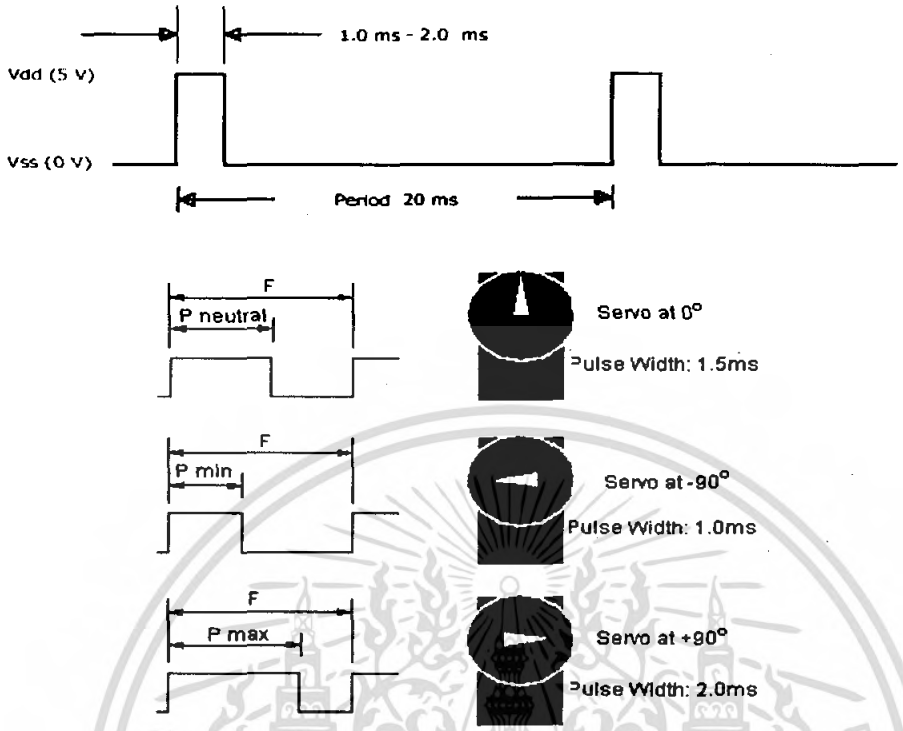


ส่วนประกอบต่างๆ ของ Servo Motor



รูปที่ 2.1 ส่วนประกอบของเซอร์โว

การควบคุมการทำงานของเซอร์โวมอเตอร์ ทำได้โดยการป้อนสัญญาณความกว้างพัลส์ ให้กับมอเตอร์ซึ่งตำแหน่งและทิศทางการหมุนของมอเตอร์จะขึ้นอยู่กับขนาดความกว้างของพัลส์



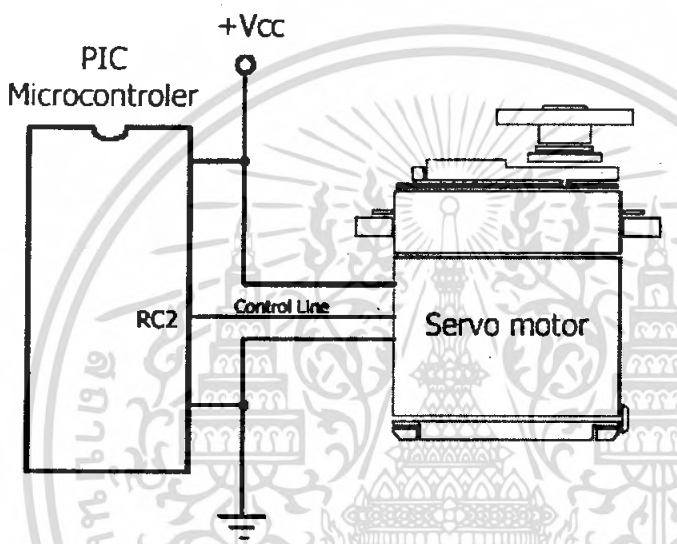
รูปที่ 2.2 สัญญาณความกว้างพัลส์ของเซอร์โวมอเตอร์

- สัญญาณความกว้างพัลส์ขนาด 1.5 มิลลิวินาที จะควบคุมให้เซอร์โวหมุนไปอยู่ที่ตำแหน่งมุม 0 องศาหรือจุด กึ่งกลางของมอเตอร์
- สัญญาณความกว้างพัลส์ขนาด 1 มิลลิวินาที จะควบคุมให้เซอร์โวหมุนไปอยู่ที่ตำแหน่งมุม –90 องศาหรือในทิศทางทวนเข็มนาฬิกา
- สัญญาณความกว้างพัลส์ขนาด 2 มิลลิวินาที จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม +90 องศา หรือในทิศทางเข็มนาฬิกา

ส่วนการที่จะควบคุมให้มอเตอร์หมุนเป็นมุมอื่นๆ นั้นก็สามารถทำได้โดยการป้อนสัญญาณพัลส์เป็นระดับความกว้างต่างๆ โดยอ้างอิงจากจุด ทั้ง 3จุดที่กล่าวมานี้ ตัวอย่างเช่น ถ้าต้องการให้มอเตอร์หมุนไปที่มุม –45 องศา เราก็จะต้องป้อนสัญญาณพัลส์ที่มีความกว้าง 1.25 มิลลิวินาที เป็นต้น และสัญญาณพัลส์นี้จะต้องจ่ายให้มอเตอร์ทุกๆ 20 มิลลิวินาที/คาบ เพื่อรักษาสภาพตำแหน่งของมอเตอร์ไว้

โดยหลักการก็คือ จะอาศัยการเปรียบเทียบช่วงเวลาของความกว้างพัลส์ที่จ่ายให้กับมอเตอร์ทางขาสัญญาณควบคุมกับค่าเวลาของวงจรอาซี (RC) ภายในบอร์ดควบคุมในตัวของมอเตอร์ ซึ่งค่าเวลาของวงจรอาซี นี้จะมีการเปลี่ยนแปลงตามการหมุนของมอเตอร์ เนื่องจากตัวต้านทานปรับค่าได้จะถูกยึด

ติดอยู่กับแกนมอเตอร์ ซึ่งการหมุนของมอเตอร์จะทำให้ค่าความต้านทานของตัวต้านทานเปลี่ยนแปลงไป เป็นผลทำให้ค่าเวลาของวงจรอาซี เปลี่ยนแปลงไปด้วย โดยในขณะที่เราป้อนสัญญาณความกว้างพัลส์ให้กับมอเตอร์ทางขาสัญญาณควบคุม สัญญาณนี้จะถูกนำไปเปรียบเทียบกับค่าเวลาของงจรอาซี หากค่าทั้ง 2 ไม่เท่ากันมอเตอร์ก็จะหมุนทำให้ค่าเวลาของวงจรอาซีเปลี่ยนแปลงจนกระทั่งค่าเวลาความกว้างพัลส์ของวงจรอาซีเปลี่ยนแปลงเท่ากับสัญญาณพัลส์ทางขาควบคุม มอเตอร์จึงจะหยุดหมุน



รูปที่ 2.3 เซอร์โวมอเตอร์และการเชื่อมต่อกับไมโครคอนโทรลเลอร์

ตารางที่ 2.1 เซอร์โวมอเตอร์รุ่นต่างๆ

Model	STD	BBM	Size (L x W x H) Mm/in	Weight		4.8V			6V			Modify 360 °
				g	oz	SPEED (sec / 60°)	Torque		SPEED (sec / 60°)	Torque		
							Kg-cm	Oz-in		Kg-cm	Oz-in	
Micro	✓		28 x 14 x 29.8 1.1 x 0.55 x 1.17	18	0.63	0.16	1.8	25	0.13	2.30	32	✓
S03N	✓		39.5 x 20.0 x 35.6 1.56 x 0.79 x 1.40	41	1.44	0.23	2.40	47	0.18	4	56	✓
S03NXF	✓		39.5 x 20.0 x 35.6 1.56 x 0.79 x 1.40	41	1.45	0.15	2.20	31	0.12	2.45	34	✓
S03T	✓		39.5 x 20.0 x 39.6 1.56 x 0.79 x 1.56	46	1.62	0.33	7.20	100	0.27	8	111	✓
S03TXF	✓		39.5 x 20.0 x 39.6 1.56 x 0.79 x 1.56	46	1.62	0.21	5	69	0.17	6.20	86	✓
S04		✓	54.4 x 26.5 x 51.5 2.14 x 1.04 x 2.03	114	4	0.25	10	138.88	0.20	13	180.5	
S666	✓		63.0 x 32.0 x 61.6 2.48 x 1.26 x 2.43	142.4	5.02	0.28	13	181	0.22	15	208	✓
PICO	✓		22.8 x 9.5 x 15.5 0.90 x 0.37 x 0.61	5.40	0.19	0.12	0.70	10	0.09	0.84	12	✓
MICRO/ 2BBM		✓	28 x 14 x 29.8 1.1 x 0.55 x 1.17	18	0.63	0.16	1.80	25	0.13	2.30	32	✓
HS-322	✓		40 x 20 x 36.5 1.57 x 0.78 x 1.43	43	1.51	0.19	3	41.66	0.15	3.5	48.6	✓
S3003	✓		41 x 20 x 36 1.6 x 0.8 x 1.4	37.2	1.3	0.23	3.2	44	0.19	4.1	56.8	✓
S03T/ 2BB/J		✓	39.5 x 20.0 x 39.6 1.56 x 0.79 x 1.56	46.0	1.62	0.33	7.2	100	0.27	8	111	✓
S03T/2B BMG/J		✓	40.6 x20.0 x 42.8 1.60 x 0.79 x 1.70	73	2.57	0.33	7.4	103	0.27	8.6	119	✓
S666/ NMG/J		✓	63.0 x 32.0 x 61.6 2.48 x 1.26 x 2.43	180	6.35	0.24	24	333	0.2	28.8	400	✓

2.1.11 การออกแบบทางซอร์ฟแวร์

ในการใช้ลินุกซ์ เป็นระบบปฏิบัติการนั้นต้องทำการสร้างดีไวซ์ไดร์เวอร์ซึ่งเป็นตัวสั่งการฮาร์ดแวร์ต่างๆ ไว้ด้วย ซึ่งในหุ่นยนต์เกราโซ่ต้องทำการ สร้างดีไวซ์ไดร์เวอร์ต่างๆ ในระดับเคอร์เนล ดังนี้

- **ยูเอสบีไดร์เวอร์** เพื่อใช้ในการติดต่อกับเซ็นเซอร์ต่างๆ
- **จีพีไอโอไดร์เวอร์** ใช้ในการสั่งการมอเตอร์
- **เอฉซีดีกราฟฟิคไดร์เวอร์** ใช้ในการสร้างสัญญาณ HSYNC , VSYNC และ อื่นๆ เพื่อให้เอลซีดีทำงานได้อย่างถูกต้อง เป็นต้น

นอกจากนี้ยังต้องพัฒนาซอร์ฟแวร์ในระดับแอปพลิชั่นเพื่อช่วยในการติดต่อกับผู้ใช้ซึ่งอาจเป็น

- **คิวที (QT)**

และแม้ว่ากล้องหรือเว็ปแคมจะเป็นเซ็นเซอร์ชนิดหนึ่งแต่มันก็มีข้อดีต่างจากเซ็นเซอร์ต่างๆ มาก เช่น

ทำให้เกิดการมองเห็น (Vision) ในตัวหุ่นยนต์ , สามารถรับภาพมากมายเพื่อทำการประมวลผล , สามารถใช้ได้ในระยะที่ไกลกว่ารวมถึงละเอียดกว่าเซ็นเซอร์ชนิดอื่น จึงเป็นเหตุผลที่ควรจะมีกระบวนการที่จะนำรูปภาพที่ได้มาใช้ประโยชน์ เช่น

- **การประมวลผลภาพ (Image processing)** หมายถึง การทำกระบวนการประมวลผลต่างๆ ที่เกี่ยวกับรูปภาพ โดยต้องทำกระบวนการพรีโปรเซสซิ่ง (preprocessing) มากมายรวมถึงการจดจำวัตถุ (object recognition) โดยใช้นิวรอนเน็ทเวิร์ค (Artificial Neural Networks : ANNs) ซึ่งถ้ามีจำนวนจุดสีหรือพิกเซล (pixel) มากๆ อาจทำให้การเทรน (train) นั้นช้าและทำได้ยากขึ้น

โดยการจดจำวัตถุนี้ทำได้โดยการวิเคราะห์จากเส้นทึบในแนวตั้งและแนวนอน และความสัมพันธ์ระหว่างเส้นเหล่านั้น ซึ่งอาจทำมุมที่แตกต่างกันไป ถ้าเราสามารถเก็บข้อมูลเหล่านี้ได้ก็จะสามารถจดจำวัตถุได้ และยังเป็นไปได้ที่จะแยกแยะวัตถุโดยใช้ความยาวสัมพัทธ์และตำแหน่งของเส้นต่างๆ

กระบวนการเหล่านี้อาจแบ่งเป็น 3 ส่วนหลักๆ คือ

การทำปรับสี (Color mapping) โดยการเปลี่ยนสีของภาพเพื่อให้เหมาะสมกับการประมวลผลซึ่งต้องทำ 3 กระบวนการ ดังนี้

- การทำเกรสเกล(Grayscale conversion) เป็นการเปลี่ยนแปลงสีของภาพให้เป็นแบบแยกแยะระดับความเข้มสีของแต่ละพิกเซล
- การทำฮิสโตรแกรม (Histogram equalization) ทำการประมวลผลเกี่ยวกับค่าของพิกเซล ที่จะแบ่งความแตกต่างออกเป็นหลายๆระดับ
- การทำบล็อบบิ้ง (Blobbing) เป็นการรีมูฟพิกเซลทั้งหมด ยกเว้นพิกเซลที่จำเป็นต่อการทำการจดจำวัตถุ

การทำเทรสโฮลด์ (Thresholding) เป็นการลดจำนวนสีในภาพ โดยไม่ต้องทำการสำเนารูปภาพ และทำการตรวจสอบทีละพิกเซลว่าถ้าค่าของพิกเซลนั้น มากกว่าค่า เทรสโฮลด์ให้พิกเซลนั้นมีค่าสูงสุด และถ้าน้อยกว่าค่าเทรสโฮลด์ก็จะให้ค่านั้นเป็นค่าต่ำสุดซึ่งวิธีนี้สามารถทำให้พิกเซลทั้งหมดหรือแต่ละคอมโพเน็นท์ถูกแยกออกจากกัน

การทำคอนโวลูชั่น (Convolution) เป็นกระบวนการที่ต้องประมวลผลทีละหลายๆ พิกเซลโดยดูจากค่าพิกเซลและค่าของพิกเซลรอบๆพิกเซลนั้นและต้องทำการสำเนารูปภาพเพื่อนำมาปรับปรุงโดยวิเคราะห์เป็นเมตริกของพิกเซล

และนอกจากการทำการประมวลผลภาพแล้วถ้าต้องการส่วนฮาร์ดแวร์ที่ทำงานได้รวดเร็วและต้องการไทม์มิ่ง (timing) ที่มีความถูกต้องแม่นยำแล้วก็ควรจะใช้เอฟพีจีเอ (FPGA : field programmable gate array) ซึ่งต้องใช้ซอฟแวร์เพื่อออกแบบฮาร์ดแวร์ในที่นี้นิยมใช้ภาษาวีเอชดีเอล (VHDL)

2.1.12 ทฤษฎีทางไฟฟ้าและนิยามที่เกี่ยวข้อง

- กราวด์ (Vss) เป็นจุดที่มีโวลต์เทจ = 0 หรือน้อยที่สุด เมื่อเทียบกับระบบ
- อิพุทโวลต์เทจ (Vin) เป็นจุดที่มีโวลต์เทจสูงสุดในวงจร บางครั้งอาจเรียก Vdd หรือ Vcc
- กระแส (Current : I) เป็นกระแสหรือการไหลของอิเล็กตรอนถ้าโวลต์เทจของ 2 ที่มีค่าต่างกันจะทำให้เกิดการไหลของกระแสขึ้น ซึ่งเราสามารถวัดกระแสได้โดยการนำแอมป์มิเตอร์ (ampmeter) มาต่ออนุกรม
- ความต้านทาน (Resistance : R) เป็นความต้านทานทางไฟฟ้า ซึ่งทำหน้าที่ป้องกันไม่ให้กระแสไหลผ่านได้ง่ายวัสดุที่มีค่า R น้อยๆ เช่น ทองแดง , เหล็ก และโลหะอื่นๆ เป็นต้น เรียกว่า ตัวนำไฟฟ้า (conductor) สิ่งที่มีความต้านทานสูงๆ เช่น ยางลบ , พลาสติก เป็นต้น เรียกว่า ฉนวนไฟฟ้า (insulator)
- กฎของโอห์ม (Ohm's law) เป็นความสัมพันธ์ระหว่าง โวลต์เทจ (V) , กระแส (I) และ ความต้านทาน (R) คือ V = IR
- การกระชากของกระแส (Power spikes & transients) ที่เกิดเมื่อวงจรต้องการกำลังสูงๆ เช่น วงจรที่ต้องใช้มอเตอร์เป็นต้น ทำให้แหล่งจ่ายไม่สามารถจ่ายไฟฟ้าได้ทันทำให้เกิดการครอปของกระแสหรือโวลต์เทจ ในขณะหนึ่งซึ่งส่งผลให้วงจรหรือไมโครคอนโทรลเลอร์ทำงานผิดพลาดหรือไม่สามารถทำงานได้ วิธีแก้ไข คือ การแยกวงจรและแบตเตอร์รี่ที่เป็นของมอเตอร์ ออกจากวงจรและแบตเตอร์รี่ของวงจรอิเล็กทรอนิกส์ อื่นๆ

2.1.13 อุปกรณ์อิเลคทรอนิกส์

- ตัวต้านทาน (Resistors) เป็นตัวต้านทานทางไฟฟ้ารวมทั้งจำกัดกระแสไม่ให้ไหลผ่านมากเกินไปเพื่อป้องกันความเสียหายของอุปกรณ์ สามารถนำมาทำตัวลดโวลต์เทจ (voltage divider) เพื่อให้ได้โวลต์เทจที่เหมาะสมที่จะนำไปใช้กับวงจรได้ตัวต้านทานมี 2 แบบคือ ความต้านทานคงที่ (standard resister) และ ความต้านทานปรับค่าได้ (variable resister)
- ตัวเก็บประจุ (Capacitors) เป็นตัวเก็บประจุหรือโวลต์เทจไว้ เพื่อช่วยลดการลดหรือเพิ่มโวลต์เทจกระทันหัน (voltage spike) แบ่งออกเป็น 2 ชนิด คือ แบบไม่มีขั้ว (standard capacitor) และแบบมีขั้ว (polarized capacitor)
- โวลต์เทจเรกูเรเตอร์(Voltage regulators) เป็นตัวกรองโวลต์เทจให้ออกมาพอดีที่จะนำไปใช้กับวงจรนั้นๆ
- แผนผังวงจร เป็นไดอะแกรมของวงจรที่ประกอบด้วยสัญลักษณ์และส่วนเชื่อมต่ออุปกรณ์ต่างๆ เป็นไดอะแกรมที่ช่วยอธิบายวงจรที่ได้ออกแบบไว้ก่อนที่จะทำการออกแบบพีซีบี (การออกแบบทั้งแผนผังวงจรและพีซีบีสามารถทำได้โดยใช้โปรแกรมช่วย เช่น โพเทล 99 (Potel 99) เป็นต้น)
- พีซีบี (Printed Circuit Boards : PCB) เป็นบอร์ดที่มีลายวงจรและรูสำหรับลงอุปกรณ์ส่วนนี้จะเป็นส่วนที่จะนำไปใช้ในหุ่นยนต์

2.1.14 แบตเตอร์รี่

หุ่นยนต์ทุกตัวต้องการพลังงานไฟฟ้าเพื่อไปขับเคลื่อนวงจรอิเล็กทรอนิกส์สำหรับโมบายโรบอทนั้นจำเป็นต้องใช้แบตเตอร์รี่เพราะเป็นแหล่งจ่ายที่สามารถเคลื่อนที่ไปกับตัวหุ่นได้ ซึ่งการเลือกใช้แบตเตอร์รี่นั้นควรดูจากโวลต์เทจปรกติและแอมป์ชั่วโมง (เป็นหน่วยวัดพลังงานไฟฟ้า) ที่ต้องการซึ่ง แบตเตอร์รี่ที่สามารถชาร์จใหม่ได้ (rechargeable battery) มีหลายชนิด เช่น SLA , NiCad , NiMH , LiIon , LiPol เป็นต้น ซึ่งแต่ละชนิดจะมีข้อดีข้อเสียต่างกันไป ได้แก่

- เอสเอลเอ (SLA : sealed lead acid) เก็บพลังงานได้น้อยถ้าเทียบกับน้ำหนักและขนาด แต่ข้อดีคือ ราคาถูก และชาร์จได้ง่าย

- นิแคด (NiCad : nickel-cadmium) และ NiMH (nickel metal hydride)ประสิทธิภาพปานกลาง แต่ราคาแพงกว่า SLA
- ลิโพล (LiPol : lithium-polymer) และ LiIon (lithium-ion) ประสิทธิภาพสูงเมื่อเทียบกับน้ำหนักและขนาด แต่ราคาแพงมาก

2.1.15 การติดต่อสื่อสารภายใน

หุ่นยนต์ต้องมีการใช้อุปกรณ์ที่หลากหลายภายในตัว เช่น เซ็นเซอร์, มอเตอร์ และอื่นๆ จึงจำเป็นต้องมีการติดต่อกับอุปกรณ์นั้นๆ เพื่อควบคุมและสื่อสารกันภายใน เช่น การควบคุมความเร็วมอเตอร์ของหุ่นยนต์ถ้าไม่มีการสื่อสารกันระหว่างคอนโทรลเลอร์และมอเตอร์แล้วก็จะทำให้การเคลื่อนที่นั้นผิดพลาดไป เป็นต้น โดยการสื่อสารกันภายในนี้เราอาจใช้เน็ทเวิร์คโปรโตคอล (network protocol) มาเป็นตัวกำหนดกฎเกณฑ์ในการติดต่อสื่อสารได้ โดยนิยามที่จำเป็นต้องทราบในการติดต่อสื่อสารได้แก่

- โหนด (Node) เป็นอุปกรณ์หรือสิ่งที่เชื่อมต่อกับเน็ทเวิร์ค เช่น บอร์ดลินุกซ์ (Linux Motherboard) หรือ มอเตอร์คอนโทรลเลอร์ (motor controller) เป็นต้น
    - มาสเตอร์ (Master) เป็นโหนดที่สามารถควบคุมเน็ทเวิร์คได้ เช่น บอร์ดลินุกซ์ เป็นต้น
    - สเลฟ(Slave) เป็นโหนดที่สามารถสื่อสารได้เฉพาะเมื่อถูกร้องขอ (request) โดยมาสเตอร์ เท่านั้น เช่น มอเตอร์คอนโทรลเลอร์ เป็นต้น
- การสื่อสารแบบเพียร์ทูเพียร์ (Peer-to-peer communication) เป็นการติดต่อสื่อสารระหว่างอุปกรณ์ที่ไม่มีลักษณะเป็นมาสเตอร์หรือสเลฟทุกๆโหนดมีสิทธิ์ที่จะเริ่มต้นการสื่อสารได้ทำให้มีโอกาสที่ 2 โหนดจะส่งข้อมูลพร้อมๆ กัน
- การสื่อสารแบบมาสเตอร์สเลฟ (Master-slave communication) เป็นการติดต่อสื่อสารที่เกิดขึ้นเมื่อมี 1 โหนดเป็นมาสเตอร์ และโหนดที่เหลือเป็นสเลฟและทุกการติดต่อจะต้องเริ่มต้นจากตัวมาสเตอร์เสมอ
- การสื่อสารแบบมัลติมาสเตอร์(Multimaster communication) เป็นการติดต่อสื่อสารที่คล้ายๆ กับมาสเตอร์สเลฟแต่มีได้มากกว่า 1 โหนดที่เป็นมาสเตอร์ ซึ่งก็จะเกิดปัญหาเดียวกับเพียร์ทูเพียร์

- โมดูล (module) เป็นคอนโทรลเลอร์หรืออุปกรณ์เพียงตัวเดียว เช่น อิฟราเรดเป็นต้น เป็นต้น
- พ็อด (pod) เป็นคอเล็กชั่นที่เกิดจากการรวมโมดูลต่างๆ เข้าด้วยกัน ซึ่งพ็อดจะติดต่อสื่อสารกับตัวอื่น โดยผ่านโปรโตคอลซีเรียล (serial protocol) เช่น โมโจบัส (MojoBus) หรือยูเอสบีทำให้ง่ายต่อการทำอุปกรณ์แบบ พลั๊กแอนด์เพล (plug-and-play) เช่น การทำพ็อดตรวจจับความระยะทาง (Distance-sensing pod) โดยการรวมโซน่า, เซอร์โว, อินฟราเรด และไมโครคอนโทรลเลอร์เข้าด้วยกันแล้วติดต่อผ่านยูเอสบี เป็นต้น

ตัวอย่างการติดต่อสื่อสาร ได้แก่

- อีเทอร์เน็ท ส่วนมากแล้วบอร์ดลินุกซ์ในปัจจุบันมักจะมีอีเทอร์เน็ทติดตั้งภายในอยู่แล้ว โดยอีเทอร์เน็ทมีลักษณะดังนี้
    - ความเร็วสูง
    - เป็น ทีซีพี/ไอพีโปรโตคอล (TCP/IP Protocol) มี การแก้ไขข้อผิดพลาด (error correction) และมีการการันตีว่าส่งข้อมูลถึง (guaranteed delivery)
    - แต่ละดีไวซ์ (device) มีแอดเดรส (address) เป็นของตัวเอง และติดตั้งได้ง่าย
    - มีการลดทอนสัญญาณรบกวน
    - ในบอร์ดมีอย่างน้อย 1 พอร์ต

  แต่ข้อเสียของอีเทอร์เน็ทเพียงอย่างเดียวคือ ราคาแพง (ประมาณ 50$) เนื่องจากโครงงานนี้ใช้การติดต่อสื่อสารภายในเพียงแค่การควบคุมมอเตอร์ และเซ็นเซอร์ จึงไม่มีความจำเป็นที่ต้องใช้อีเทอร์เน็ท
- ยูเอสบี (USB : Universal Serial Bus) เป็นทางเลือกที่ดี โดยเฉพาะยูเอสบี 2.0 มีความเร็วสูงและส่วนใหญ่มีไดร์เวอร์ที่เป็นมาตรฐาน (standard driver) อยู่ในตัวระบบปฏิบัติการอยู่แล้ว รวมทั้งเรายังสามารถสร้างดีไวซ์ของตนเองได้โดยการใช้ชิพ เช่น เอฟทีดีไอ (FTDI) ที่ใช้แปลงจาก ยูเอสบี เป็น อาเอส 485 (asynchronous serial protocol : RS 485) หรือถ้าเป็นอุปกรณ์ที่มีอยู่แล้ว (ถ้าเป็นยูเอสบี) เช่น เว็ปแคม, เมาส์, คีย์บอร์ด, ฮาร์ดไดร์ฟ เป็นต้น ก็สามารถนำมาเชื่อมต่อใช้งานได้ โดยในที่นี้เราจะนำมาใช้กับการควบคุมมอเตอร์ และเซ็นเซอร์ต่างๆ ซึ่งในไมโครคอนโทรลเลอร์บางประเภทก็มียูเอสบีในระดับฮาร์ดแวร์อยู่แล้ว แม้ว่ายูเอสบียังมีข้อเสียตรงที่โปรโตคอลมีความซับซ้อนมากรวมทั้งมีแหล่งข้อมูลให้ศึกษาน้อย แต่มีข้อดีที่สำคัญคือ

มีความเร็วสูงและง่ายต่อการใช้งาน ทำให้ยูเอสบีเป็นการติดต่อสื่อสารที่เหมาะสมที่สุดที่จะนำมาใช้ในโครงงานนี้

- โปรโตคอลอนุกรมแบบไม่สอดคล้อง (Asynchronous Serial Protocol) เป็นโปรโตคอลที่เก่าแก่ที่สุดที่ใช้ในบอร์ดของคอมพิวเตอร์มันเหมือนกับโปรโตคอลที่ใช้กับพอร์ตอนุกรมซึ่งโปรโตคอลอนุกรมนี้ครอบคลุมตั้งแต่ อาเอส 232 (RS232), อาเอส 484 และอื่นๆ โดยมีลักษณะโดยรวมดังนี้

  มี 1 สตาร์ทบิท (start bit)

  มี 7 หรือ 8 บิทที่เป็นข้อมูล

  มี 5 อ็อพชั่น (option) ของ พาริตี้บิท (parity bit) คือ

  - ไม่มีพาริตี้บิท (No parity)
  - พาริตี้บิทจะเท่ากับ 1 เมื่อ ผลรวมของทุกบิทเป็นเลขคี่ (Odd parity)
  - พาริตี้บิทจะเท่ากับ 1 เมื่อ ผลรวมของทุกบิทเป็นเลขคู่ (Even parity)
  - พาริตี้บิทเป็น 1 ตลอด (Mark parity)
  - พาริตี้บิทเป็น 0 ตลอด (Space parity)

  มี สต๊อพบิท (stop bit) 1 บิทหรือมากกว่านั้น

  โดยปกติแล้วจะใช้ 8N1 (8 ดาต้าบิท, ไม่มีพาริตี้บิท, 1 สต๊อพบิท)

- อาเอส 232 ใช้ในพอร์ตอนุกรมของคอมพิวเตอร์ใช้โวลต์เทจไฮ (voltage high) เป็นบิท 1 และใช้โวลต์เทจโลว์ (voltage low) แทนบิท 0 ข้อดี คือ มีดีไวซ์มากมายที่ใช้โปรโตคอลนี้ ข้อเสีย คือ มีการลดทอนสัญญาณสูง ถูกรบกวนจากสัญญาณรบกวนได้ง่าย และแล็พท๊อพบางรุ่นไม่มีพอร์ทอนุกรมอยู่
- อาเอส 485 คล้ายกับอาเอส 232 แต่มีสายเป็นแบบ สายคู่บิดเกลียว (twisted pair) 1 คู่ต่อ 1 ทิศทาง และมีสาย กราวด์ อาเอส 485 โดยทั่วไปจะเป็นแบบฮาล์ฟดูเพล็กซ์ (half-duplex) ทำให้บางสิ่งง่ายและบางสิ่งก็ซับซ้อนขึ้น ค่าของแต่ละบิทขึ้นอยู่กับผลต่างของระดับโวลต์เทจที่ อาเอส 485 ใช้สายคู่บิดเกลียวเพราะช่วยลดสัญญาณรบกวนและลดการลดทอนสัญญาณจากสายอีกเส้นหนึ่ง อาเอส 485 อาจใช้กับสายแคท 5 (CAT5) และหัวอาเจ 45 (RJ45) ได้

- ไอสเควซี (I2C : Intra-IC communication) ถูกพัฒนาขึ้นมาเพื่อใช้ติดต่อภายในบอร์ดเดียวกัน เป็นซิงโครนัสโปรโตคอล (synchronous protocol) และเป็นโปรโตคอลที่ต่อเชื่อมได้มากกว่า 2 โหนด (multidrop protocol) ซึ่งในหุ่นยนต์ตัวอย่างนี้ใช้ในการติดต่อกับโซน่า, เครื่องตรวจจับความร้อน และเข็มทิศ เป็นต้น ไอสเควซีบางครั้งถูกเรียกว่าเอสเอ็มบัส (SMBus) และไมโครคอนโทรลเลอร์หลายรุ่นสามารถติดต่อแบบไอสเควซีโดยตรงได้
- เอสพีไอ (SPI : Serial Peripheral Interface) พัฒนาขึ้นมาด้วยเหตุผลเดียวกับ ไอสเควซี (ใช้เพื่อสื่อสารกันระหว่าง IC) เป็นโปรโตคอลที่ต่อเชื่อมได้มากกว่า 2 โหนด แต่เปลี่ยนจากการใช้แอดเดรส (address) ใส่ไปกับแพ็กเกจข้อมูล (data packet) มาเป็นการใช้สายชิพซีเล็ก (chip select) แทน เมื่อชิพซีเล็กของสเลฟตัวไหนเป็นโลว์ (ในเวลาหนึ่งมีได้เพียงตัวเดียว) สเลฟตัวนั้นก็จะคอยฟังการติดต่อจากมาสเตอร์และตอบสนองกลับไป เอสพีไอเหมาะกับการติดต่อสื่อสารในระยะสั้นๆ ไม่ควรใช้ส่งในระยะไกลๆ
- โรบิน (ROBIN : Robot Independent Network) เป็นแบบมัลติมาสเตอร์ที่มีข้อดีคือเขียนโปรแกรมง่าย
- โมโจบัส (MojoBus) เป็นเท็กซ์เบสโปรโตคอล (text-based protocol) โดยใช้ อาเอส 485 ข้อดีคือง่ายต่อการตรวจสอบข้อผิดพลาดเพราะตัว protocol เป็น text-based

2.2 ยูเอสบี (USB : Universal serial bus)

จุดกำเนิดและความหมาย

พอร์ทแต่ละชนิดในคอมพิวเตอร์และในอุปกรณ์ต่างๆโดยทั่วไปได้รับการออกแบบมาให้ใช้งานเฉพาะ ทำให้อุปกรณ์แต่ละตัวต้องเลือกใช้พอร์ตต่างชนิดกันไป ส่งผลให้การนำอุปกรณ์ต่างๆ มาต่อเชื่อมมีการใช้พอร์ตในหลายลักษณะ เช่น พอร์ตพีเอส 2 (PS2 port) ใช้ติดต่อกับเมาส์ (mouse) และคีย์บอร์ด (keyboard), พาราเลลพอร์ต (Parallel port) ใช้ติดต่อกับปริ้นเตอร์ (printer) เป็นต้น ทำให้ผู้ใช้ที่ไม่มีความรู้ความชำนาญพบความยากลำบากในการเรียนรู้เรื่องราวเหล่านี้ นอกจากนั้นการติดตั้งอุปกรณ์ต่างๆ เพิ่มเข้าไปจะทำให้เกิดปัญหาการแย่งกันใช้สัญญาณไออาคิว (IRQ : Interrupt request) ซึ่งเป็นตัวจำกัดจำนวนอุปกรณ์ที่จะมาเชื่อมต่อโดยปัญหาทั้งสองนี้สามารถแก้ไขได้โดยการใช้การติดต่อสื่อสารแบบยูเอสบี

ยูเอสบี แปลว่า บัสอนุกรมเอนกประสงค์ คุณสมบัติที่สำคัญมีดังนี้

- ใช้คอนเน็กเตอร์เพียงชนิดเดียวซึ่งมี 2 รูปแบบสำหรือเชื่อมต่ออุปกรณ์ทุกๆ ชนิด
- สามารถเชื่อมต่ออุปกรณ์หลายๆ ชนิดรวมเข้าสู่คอนเน็กเตอร์ตัวเดียว สูงสุด 127 ตัว
- ไม่เกิดการขัดแย้งกันในการใช้ทรัพยากรณ์ของระบบและไม่มีปัญหากับไออาคิว
- ตรวจสอบการเชื่อมต่อและตั้งค่าการทำงานต่างๆ อัตโนมัติ ระหว่างที่เครื่องกำลังทำงานอยู่ (hot attachment)
- ความเร็วในการถ่ายทอดข้อมูลจะขึ้นอยู่กับมาตรฐาน ซึ่งมีรายละเอียดดังนี้
  - มาตรฐานยูเอสบี 1.0/1.1 มีอัตราการถ่ายทอดข้อมูลความเร็วต่ำ (low speed) ประมาณ 1.5 เมกะบิทต่อวินาที และมีอัตราการถ่ายทอดความเร็วเต็มที่ (full speed) เท่ากับ 12 เมกะบิทต่อวินาที
  - มาตรฐานยูเอสบี 2.0 จะมีอัตราการถ่ายทอดข้อมูลเพิ่มขึ้นอีก 1 ระดับคือ ความเร็วสูง (high speed) ซึ่งมีความเร็วสูงถึง 480 เมกะบิทต่อวินาที
- ที่ขาพอร์ตยูเอสบี มีแรงดันไฟฟ้ากระแสตรง 5 โวลต์ จ่ายออกมาด้วย ทำให้อุปกรณ์ต่อพ่วงที่ใช้พลังงานไม่มากนักสามารถใช้แรงดันจากไฟเลี้ยงยูเอสบีนี้ไปทำงานได้โดยไม่ต้องอาศัยแหล่งจ่ายไฟภายนอกเพิ่มเติมอีก

2.2.1 การส่งข้อมูลภายในบัสยูเอสบี 1.0/1.1

ยูเอสบีเป็นการส่งข้อมูลและเชื่อมต่อในรูปแบบของบัส คือ อุปกรณ์ทุกๆ ตัวที่ต่อเชื่อมจะต้องส่งสัญญาณรวมกันไปในสายส่งสัญญาณเพียงคู่เดียว ดังนั้นอุปกรณ์ทุกๆ ตัวจะต้องส่งข้อมูลเรียงลำดับเพื่อป้องกันการชนกันของข้อมูล และทำให้ในช่วงเวลาหนึ่งๆ จะมีข้อมูลวิ่งไปได้เพียงทิศทางเดียวเท่านั้น

จังหวะการรับส่งข้อมูลของระบบบัสยูเอสบีทั้งหมดจะถูกควบคุมโดยโฮสต์ซึ่งก็คือตัวแพลตฟอร์มที่เป็นจุดรวมของอุปกรณ์ต่างนั่นเอง ดังนั้นจึงไม่สามารถเชื่อมต่อโฮสต์ 2 ตัวเข้าด้วยกันโดยตรงได้ เพราะจะทำให้เกิดการชนกันของข้อมูลภายในบัสเนื่องจากแต่ละโฮสต์ก็จะพยายามกำหนดจังหวะในการรับส่งของตนเองขึ้นมา ดังนั้นจึงต้องมีอุปกรณ์ตัวกลางเพื่อซิงโครไนซ์ตัวเองเข้ากับโฮสต์ทั้ง 2 ฝั่งให้ได้

การรับส่งข้อมูลจะถูกกำหนดเป็นเฟรม (frame) โดยทุกๆ 1 มิลลิวินาทีจะเกิดการรับส่งข้อมูลขึ้น 1 เฟรมในแต่ละเฟรมจะถูกแบ่งแยกออกเป็นแพ็กเกจและเริ่มต้นการทำงานของแต่ละเฟรมโดยโฮสต์จะส่งสัญญาณเริ่มต้นเฟรม (SOF : start of frame) ออกไปให้อุปกรณ์ทุกตัวรู้จังหวะการเริ่มเฟรม หลังจากนั้นโฮสต์ก็จะเริ่มส่งหรือรับข้อมูลต่างๆ ตามที่ได้จัดลำดับความสำคัญไว้ อุปกรณ์ต่างๆ ที่อยู่ภายในบัสจะต้องทำตามจังหวะที่โฮสต์กำหนดไว้เท่านั้น การส่งข้อมูลกลับไปยังโฮสต์จะทำได้เมื่อได้รับการถามหรือร้องขอจากโฮสต์

แต่เนื่องจากแต่ละเฟรมจะต้องรับส่งข้อมูลให้เสร็จภายใน 1 มิลลิวินาทีนั่นหมายความว่าข้อมูลของอุปกรณ์ทุกๆ ตัวที่เชื่อมต่อกับบัสจะต้องถูกกำหนดขนาดไม่ให้ใหญ่เกินกว่าที่จะสามารถรับส่งได้ภายใน 1 มิลลิวินาทีและเล็กพอที่จะทำให้อุปกรณ์ทุกๆ ตัวสามารถใช้งานบัสไปพร้อมๆ กันได้ ดังนั้น ยูเอสบีจึงจำเป็นต้องอาศัยซอฟแวร์ที่เข้ามาจัดการในด้านนี้ และต้องอาศัยฮาร์ดแวร์ที่จะคอยกระจายการส่งและรวบรวมการรับข้อมูลจากทุกๆ อุปกรณ์ในระบบ ซึ่งซอฟแวร์และฮาร์ดแวร์ที่จำเป็นกับยูเอสบีมีดังนี้

ส่วนซอฟแวร์

- ไดรเวอร์อุปกรณ์ยูเอสบี (USB device drivers)
- ไดรเวอร์ยูเอสบี (USB driver)
- ไดรเวอร์โฮสต์คอนโทรลเลอร์ (USB host controller driver)

ส่วน hardware

- ยูเอสบีโฮสต์คอนโทรลเลอร์ (USB host controller) / รูทฮับ (root hub)
- ยูเอสบีฮับ (USB hub)
- อุปกรณ์ยูเอสบี (USB device)

2.2.2 ส่วนประกอบทาง Software

- **ไดรเวอร์อุปกรณ์ยูเอสบี คือ โปรแกรมเก็บข้อมูลที่จำเป็นในการติดต่อไปยังอุปกรณ์แต่ละตัว เมื่อโปรแกรมใดมีความต้องการจะติดต่อกับอุปกรณ์ต่างๆ จะต้องแจ้งความต้องการนั้นๆ มายังไดรเวอร์อุปกรณ์ยูเอสบี เนื่องจากตัวไดร์เวอร์นี้จะรู้ว่าถ้าต้องการติดต่ออุปกรณ์จะต้องติดต่อผ่านเอ็นพอยท์ (endpoint) ใด ด้วยรูปแบบใด (การทำงานของอุปกรณ์ยูเอสบีจะติดต่อสั่งงาน**

ผ่านเอ็นพอยท์ของตัวอุปกรณ์ ซึ่งอุปกรณ์ต่างๆ จะมีชนิดและจำนวนเอ็นพอยท์ที่ต่างกัน) ดังนั้นอุปกรณ์แต่ละตัวจะต้องมีไดรเวอร์อุปกรณ์ยูเอสบีเฉพาะตัว เช่น ถ้าต้องการติดต่อกับอุปกรณ์คีย์บอร์ดตัวไดรเวอร์อุปกรณ์ยูเอสบีก็จะรู้ว่าต้องรับส่งข้อมูลแบบความเร็วต่ำ โดยใช้รูปแบบการถ่ายทอดข้อมูลแบบอินเทอร์รับผ่านเอ็นพอยท์ตัวหนึ่งของคีย์บอร์ดและตรวจสอบข้อมูลการกดเป็นช่วงๆ ระยะห่างค่าหนึ่ง

แต่ในบางอุปกรณ์พื้นฐานของเครื่องคอมพิวเตอร์ เช่น เมาส์, คีย์บอร์ด จะมีการบรรจุไดรเวอร์อุปกรณ์ยูเอสบีเหล่านั้นไว้ในไบออส (BIOS) ของคอมพิวเตอร์เรียบร้อยแล้วจึงไม่ต้องติดตั้งไดรเวอร์เพิ่มเติมสำหรับอุปกรณ์นี้

- ไดรเวอร์ยูเอสบี การทำงานของยูเอสบีนั้นเป็นการต่อร่วมกันของอุปกรณ์หลายๆ ชนิดบนสายสัญญาณเพียงคู่เดียว ดังนั้นการส่งข้อมูลของอุปกรณ์แต่ละชนิดจะต้องมีการแบ่งสรรปันส่วนกันไปอย่างพอเหมาะพอดี เพื่อให้อุปกรณ์ทุกตัวสามารถทำงานไปได้พร้อมๆ กัน และแน่นอนว่าต้องมีซอฟแวร์ที่เข้ามาทำหน้าที่นี้ ซึ่งก็คือ ไดรเวอร์ยูเอสบีนั่นเอง

ไดรเวอร์อุปกรณ์ยูเอสบีของอุปกรณ์แต่ละตัวจะส่งการร้องขอเพื่อการติดต่อ (request) ลงมายังไดรเวอร์ยูเอสบี และเมื่อไดรเวอร์ยูเอสบีรับทราบความต้องการการติดต่อของอุปกรณ์ครบทุกตัวที่เชื่อมอยู่บนบัสแล้ว ก็จะพิจารณาว่า ในรอบการรับส่งข้อมูลหนึ่งๆ นั้นอุปกรณ์แต่ละตัวสามารถรับส่งข้อมูลได้มากเท่าใด หากปริมาณข้อมูลที่ต้องการรับส่งมีขนาดใหญ่มากก็จะตัดแบ่งออกเป็นส่วนๆ แล้วเก็บไว้เพื่อรอส่งในรอบถัดไป โดยปริมาณข้อมูลที่ส่งได้ของอุปกรณ์แต่ละตัวจะถูกพิจารณาจากชนิดของการถ่ายทอดข้อมูล (transfer type) ว่าอุปกรณ์ใดใช้การถ่ายทอดข้อมูลแบบใดและการรับส่งข้อมูลชนิดนั้นมีลำดับความสำคัญมากน้อยเพียงใด

- ไดรเวอร์โฮสต์คอนโทรลเลอร์ หลังจากไดรเวอร์ยูเอสบีพิจารณาแล้วว่าอุปกรณ์แต่ละตัวส่งข้อมูลได้เท่าใดบ้าง มันจะส่งข้อมูลของอุปกรณ์แต่ละตัวที่จะติดต่อในรอบการติดต่อนั้นๆ มายังไดรเวอร์โฮสต์คอนโทรลเลอร์ จากนั้นไดรเวอร์โฮสต์คอนโทรลเลอร์จะจัดเรียงลำดับข้อมูลของอุปกรณ์แต่ละชนิดลงในเฟรมข้อมูล เพิ่มเติมส่วนประกอบต่างๆ ของเฟรมข้อมูลให้ครบตามมาตรฐานการถ่ายทอดข้อมูลแบบยูเอสบีแล้วส่งข้อมูลทั้งหมดไปยัง ฮาร์ดแวร์ยูเอสบีโฮสต์คอนโทรลเลอร์เพื่อส่งข้อมูลทั้งหมดออกไปยังอุปกรณ์ต่างๆ

2.2.3 ส่วนประกอบทางฮาร์ดแวร์

- **ยูเอสบีโฮสต์คอนโทรลเลอร์ มีหน้าที่สร้างสัญญาณข้อมูลทางไฟฟ้า แล้วส่งต่อไปยังรูทฮับ(root hub) เพื่อกระจายออกไปยังอุปกรณ์ต่างๆ โดยมันจะสร้างสัญญาณการติดต่อข้อมูลรูปแบบต่างๆตามที่ไดรเวอร์โฮสต์คอนโทรลเลอร์กำหนดมาให้ จากนั้นแปลงข้อมูลที่จะส่งจากแบบขนานเป็นแบบอนุกรมเพื่อใช้ในการส่งต่อไป เมื่อสัญญาณมาถึงรูทฮับ รูทฮับก็จะส่งสัญญาณนั้นออกไปยังบัสเพื่อส่งต่อไปยังอุปกรณ์ต่างๆนอกจากนั้นรูทฮับยังทำหน้าที่สำคัญอีก 4 อย่าง คือ**
    - **ควบคุมการใช้พลังงานของอุปกรณ์ที่มาต่อ**
    - **ตรวจสอบการเชื่อมต่อของอุปกรณ์ว่ามีอุปกรณ์ต่ออยู่หรือไม่**
    - **เปิดการใช้งานพอร์ต (enable) เมื่อมีอุปกรณ์ต่ออยู่ และปิดการใช้งาน (disable) เมื่อปลดอุปกรณ์ออกไปแล้ว**
    - **รายงานสถานะของแต่ละพอร์ต เมื่อไดรเวอร์โฮสต์คอนโทรลเลอร์ร้องขอมา**
- **ยูเอสบีฮับ หน้าที่หลักๆคือ ขยายการเชื่อมต่อให้อุปกรณ์จำนวนมากๆ สามารถเชื่อมต่อเข้ากับระบบบัสได้ โดยการทำงานหลักมี 2 ส่วน คือ ทำหน้าที่เป็นตัวทวนสัญญาณ (repeater) และตัวจัดการพลังงาน (power management) ในส่วนการทวนสัญญาณ ยูเอสบีฮับจะต้องรับสัญญาณจากโฮสต์มา แล้วส่งกระจายออกไปยังพอร์ตทุกๆ พอร์ต และรับสัญญาณจากแต่ละพอร์ต แล้วจับมารวมกันเพื่อส่งกลับไปที่โฮสต์สำหรับส่วนของการจัดการพลังงานนั้นมีหน้าที่เหมือนกับรูทฮับก็คือ ตรวจสอบว่ามีการเชื่อมต่ออยู่ของอุปกรณ์ที่พอร์ตใดบ้าง หากมีอุปกรณ์ต่ออยู่ก็เปิดการใช้งานพอร์ตนั้นๆ หากไม่มีอุปกรณ์ต่ออยู่ก็ปิดการใช้งาน ตรวจสอบการเชื่อมต่อหรือปลดออกของอุปกรณ์เพื่อรายงานผลให้โฮสต์คอนโทรลเลอร์ร้องขอ และป้องกันอุปกรณ์ที่ต่ออยู่ในแต่ละพอร์ตไม่ให้ดึงกระแสไฟฟ้าเกินกว่าที่กำหนด**
- **อุปกรณ์ยูเอสบี คือ อุปกรณ์ต่างๆ ที่เชื่อมต่อกับคอมพิวเตอร์ด้วยพอร์ตยูเอสบี นั่นเอง สามารถแบ่งเป็น 2 ชนิดตามความเร็วในการถ่ายทอดข้อมูลคือ**
    - **อุปกรณ์ความเร็วต่ำ (low speed devices) ถ่ายทอดข้อมูลด้วยความเร็ว 1.5 เมกะบิทต่อวินาที**

- อุปกรณ์ความเร็วเต็มที่ (full speed devices) รับส่งข้อมูลด้วยความเร็ว 12 เมกะบิทต่อวินาที

ในปัจจุบันนี้อุปกรณ์ยูเอสบีที่มีจำหน่ายอยู่ในท้องตลาดมีอยู่เป็นจำนวนมาก เช่น คีย์บอร์ด , เมาส์ , จอยสติ๊ก เหล่านี้คืออุปกรณ์ยูเอสความเร็วต่ำ ส่วนจอมอนิเตอร์ , ลำโพง , เครื่องพิมพ์ , กล้องถ่ายรูปดิจิตอล , ซีดีรอมไดรฟ , เครื่องเล่นเอ็มพี3 จัดเป็นอุปกรณ์ยูเอสบีความเร็วสูง อุปกรณ์บางตัวจะบรรจุความสามารถของยูเอสบีฮับ เข้าไปด้วยทำให้สามารถนำอุปกรณ์อื่นๆ มาเชื่อมต่อได้เหมือนกับการต่อเข้ากับฮับอุปกรณ์เหล่านี้ (Compound USB devices) ตัวอย่างของอุปกรณ์ที่มีฮับอยู่ภายในได้แก่ จอมอนิเตอร์ , เครื่องพิมพ์ เป็นต้น

นอกจากการแบ่งชนิดของอุปกรณ์ตามความเร็วในการถ่ายทอดข้อมูลแล้วยังแบ่งอุปกรณ์ยูเอสบีตามการใช้พลังงานของตัวอุปกรณ์เองก็ได้ซึ่งสามารถแบ่งได้อีก 2 ชนิด คือ

- อุปกรณ์ที่ใช้ไฟเลี้ยงจากบัส (bus powered device) คือ อุปกรณ์ที่ใช้ไฟเลี้ยงจากบัสโดยตรง ไม่ต้องมีแหล่งจ่ายไฟภายนอกเพิ่มเติม
- อุปกรณ์ที่ใช้ไฟเลี้ยงจากตัวเอง (self powered device) คือ อุปกรณ์ที่มีแหล่งจ่ายไฟในตัวไม่ต้องอาศัยไฟเลี้ยงจากบัส

2.2.4 โครงสร้างการเชื่อมต่อ (Topology)

โครงสร้างการเชื่อมต่อของยูเอสบีนั้นเป็นแบบสตาร์ (Star) โดยมีฮับเป็นศูนย์กลางจุดเชื่อมต่อไปยังอุปกรณ์ต่างๆ ในแต่ละระดับชั้น สายเชื่อมต่อแต่ละเส้นเป็นการต่อแบบจุดต่อจุด (point to point) ดังนั้นต้องระวังว่าพอร์ตยูเอสบีมีรูปแบบการเชื่อมต่อเป็นระบบบัส แต่มีโครงสร้างการเชื่อมต่อแบบสตาร์

2.2.5 การติดต่อระหว่างอุปกรณ์และโฮสต์

การถ่ายทอดสัญญาณ (transfer type) ของยูเอสบีนั้นแบ่งออกเป็น 4 ชนิดตามขนาดชนิดของข้อมูลและจังหวะการส่งข้อมูลดังนี้

- **การถ่ายทอดสัญญาณแบบไอโซโครนัส (Isochronous transfer) ใช้กับการรับส่งข้อมูลที่ต้องการความต่อเนื่องสูง มีอัตราเร็วในการส่งข้อมูลคงที่ ไม่มีการรับประกันความถูกต้องของ**

ข้อมูล หากเกิดความผิดพลาดจะไม่มีการพยายามส่งข้อมูลนั้นไปใหม่ การรับส่งแบบนี้จะต้องจองการรับส่งเฟรมข้อมูลในแต่ละ 1 มิลลิวินาที ไว้ตลอดเวลาด้วย เช่น ข้อมูลเสียงเพลง , ลำโพง เป็นต้น โดยมีการซิงโครไนซ์จังหวะการรับส่งเฟรมข้อมูล แบ่งเป็น 3 วิธี คือ

- **อะซิงโครนัส (Asynchronous)** สัญญาณนาฬิกาของอุปกรณ์ไม่จำเป็นต้องเข้าจังหวะกับสัญญาณข้อมูลของยูเอสบีแต่อัตราการส่งข้อมูลจะต้องถูกกำหนดไว้ตายตัวที่ค่าใดค่าหนึ่ง ซึ่งกำหนดตอนเริ่มต้นการเชื่อมต่อ ไม่สามารถเปลี่ยนแปลงได้ เช่น ไมโครโฟนที่อัตราการสุ่มข้อมูล (sampling rate) เป็นอิสระไม่ขึ้นกับสัญญาณเอสโอเอฟ (SOF) ของยูเอสบี เป็นต้น
- **ซิงโครนัส (Synchronous)** สัญญาณนาฬิกาของอุปกรณ์จะต้องเข้าจังหวะกับสัญญาณข้อมูลและเอสโอเอฟของยูเอสบี เช่น โมเด็ม (modem ISDN 64 kb/s) จะต้องเกิดการซิงโครไนซ์ สัญญาณเอสโอเอฟของโฮสต์เข้ากับสัญญาณนาฬิกาของระบบไอเอสดีเอ็น (ISDN) ก่อนจึงจะถ่ายทอดข้อมูลที่ความเร็วคงที่ 64 กิโลบิทต่อวินาทีได้
- **อะแดพทีฟ(Adaptive)** วิธีนี้เป็นวิธีที่ตัวอุปกรณ์มีความสามารถมากที่สุด เพราะสามารถปรับความเร็วในการถ่ายทอดข้อมูลได้ในช่วงเวลาที่กว้างมาก ไม่ถูกกำหนดตายตัวไว้ค่าใดค่าหนึ่ง สามารถเปลี่ยนแปลงความเร็วในระหว่างการใช้งาน เช่น ซีดีรอม (CDRom) ที่สามารถเปลี่ยนอัตราการสุ่มข้อมูลได้

- **การถ่ายทอดสัญญาณแบบบัลก์ (Balk transfer)** ใช้กับการถ่ายทอดข้อมูลที่มีปริมาณมากๆ แต่ไม่ต้องการความต่อเนื่องของข้อมูล และไม่ต้องการอัตราการส่งข้อมูลที่คงที่ การส่งช้าหรือเร็วจะส่งผลกระทบในด้านเวลาที่ต้องใช้มากขึ้นเท่านั้น แต่ข้อมูลไม่ได้มีความเสียหายแต่อย่างใด เพราะในการส่งข้อมูลแต่ละครั้งจะมีการตรวจสอบความถูกต้องและจะเกิดการส่งใหม่เมื่อมีความผิดพลาดเกิดขึ้น เช่น ซีดีรอม (ส่วนที่ไม่ใช่การอ่านข้อมูลเป็นเพลง) , เครื่องพิมพ์ หรือ สแกนเนอร์
- **การถ่ายทอดสัญญาณแบบอินเทอร์รัปต์ (Interrupt transfer)** ใช้กับการถ่ายทอดข้อมูลที่มีจำนวนน้อยครั้งและปริมาณของข้อมูลไม่มาก โดยอาศัยวิธีการวนอ่านข้อมูลจากอุปกรณ์เป็นระยะๆ หรือการโพลลิ่ง (Polling) โดยอัตราการวนอ่านนี้จะต้องไม่ช้าเกินไปเพราะจะทำให้เกิดการสูญเสียข้อมูล และไม่เร็วเกินไปเพราะจะทำให้ใช้แบนด์วิดท์ (bandwidth) มากเกินความ

จำเป็น ถ้าหากอุปกรณ์ไม่มีข้อมูลที่จะส่งก็จะส่งแน็ค (NAK : Negative Acknowledgement) ตอบกลับไป เช่น คีย์บอร์ด เป็นต้น

- การถ่ายทอดสัญญาณควบคุม (Control transfer) มีความสำคัญมาก เพราะใช้สำหรับรับส่งคำสั่งควบคุมการทำงานทั้งหมดของอุปกรณ์ทุกๆ ตัวตั้งแต่เริ่มแรกเมื่ออุปกรณ์ถูกต่อเข้ากับระบบ เกิดการอ่านเดสคริปเตอร์ ต่างๆ เมื่อพิจารณาแล้วว่าสามารถรองรับการทำงานได้หรือไม่ ถ้าได้ก็จะทำการตั้งค่าการทำงานต่างๆ และเริ่มการทำงาน ทั้งหมดที่กล่าวมานี้จะใช้การถ่ายทอดสัญญาณควบคุมเพื่อติดต่อกับการควบคุมเอ็นพอยท์ (endpoint control) ซึ่งเป็นกุญแจหลักทั้งสิ้น

2.2.6 เดสคริปเตอร์, ความหมาย , ชนิดและรูปแบบการใช้งาน

อุปกรณ์แต่ละตัวมีคุณสมบัติและการทำงานที่แตกต่างกัน โฮสต์จำเป็นต้องรู้คุณสมบัติทั้งหมดของอุปกรณ์แต่ละตัวเพื่อให้การสั่งงานเป็นไปได้อย่างถูกต้อง และเนื่องจากบัสข้อมูลทั้งหมดถูกใช้งานรับส่งข้อมูลร่วมกันระหว่างอุปกรณ์ทุกๆ ตัว สิ่งที่โฮสต์จำเป็นต้องรู้อีกอย่างก็คือ ปริมาณที่ต้องการส่งหรือแบนด์วิดท์ของอุปกรณ์แต่ละตัวในบัส ดังนั้นเมื่อมีอุปกรณ์ตัวใหม่ต่อเข้ากับบัสก็สามารถใช้ข้อมูลเหล่านี้พิจารณาได้ว่าจะรองรับอุปกรณ์นั้นๆ ได้หรือไม่ ข้อมูลเหล่านี้รวมเรียกว่า ดีไวซ์เดสคริปเตอร์ (Device descriptor)

- ดีไวซ์เดสคริปเตอร์ ในอุปกรณ์แต่ละตัวจะมี ดีไวซ์เดสคริปเตอร์เพียงชุดเดียวเท่านั้น ภายในจะระบุข้อมูลที่ใช้ในการเชื่อมต่อขั้นแรก (default descriptor pipe) เพื่อใช้ในการกำหนดข้อมูลสถานะของอุปกรณ์เข้ากับโฮสต์ นอกจากนั้นยังเก็บข้อมูลของข้อกำหนดในโหมดการทำงานต่างๆ ของอุปกรณ์รวมถึงจำนวนคอนฟิกกูเรชั่น (configuration) ด้วย เพราะในครั้งแรกที่อุปกรณ์เชื่อมต่อเข้ากับบัสนั้นโฮสต์ ไม่มีทางรู้ได้เลยว่า ต้องติดต่อกับอุปกรณ์ที่เอ็นพอยท์ใด จึงจำเป็นต้องขอข้อมูลส่วนนี้ก่อนที่จะติดต่อกับส่วนอื่น
- คอนฟิกกูเรชั่นเดสคริปเตอร์ (Configuration descriptor) ใช้เก็บข้อมูลที่จำเป็นของการทำงานในแต่ละโหมดการทำงานและเก็บจำนวนอินเทอร์เฟซ ที่ใช้งานในโหมดนั้นๆ เช่น อุปกรณ์บางตัวที่มีการทำงาน 2 โหมด คือ โหมดใช้พลังงานสูง และโหมดประหยัดพลังงานเดสคริป

เตอร์ ตัวนี้จะเก็บข้อมูลที่จำเป็นในการตั้งค่าต่างๆ ของโฮสต์ที่ต้องการเลือกใช้โหมดการทำงาน แต่โหมดของอุปกรณ์

- **อินเทอร์เฟซเดสคริปเตอร์(Interface descriptor)** ในแต่ละโหมดการทำงานหรือ คอนฟิกกูเรชั่น อาจจะมีการเชื่อมต่อหรืออินเทอร์เฟซ 1 แบบหรือมากกว่าเพื่อใช้งานในหน้าที่ต่างๆ เช่น ซีดีรอม โดยในตัวซีดีรอมจะมีการรับส่งข้อมูลปริมาณมากๆ (mass storage), การส่งข้อมูลเสียงออดิโอ และการส่งข้อมูลภาพจะเห็นได้ว่า มีอินเทอร์เฟซที่ใช้งานแยกกัน ภายในอินเทอร์เฟซ เดสคริปเตอร์จะบรรจุข้อมูลการใช้งานอินเทอร์เฟซนั้นๆ โดยข้อมูลเหล่านี้จะระบุว่าอุปกรณ์ถูกจัดอยู่ในคลาสหรือซับคลาสใดและมีเอ็นพอยท์ จำนวนเท่าใดบ้างที่ใช้งานในอินเทอร์เฟซนี้บ้าง
- **เอ็นพอยท์เดสคริปเตอร์ (Endpoint descriptor)** ใช้เก็บข้อมูลคุณสมบัติของแต่ละเอ็นพอยท์ เช่น ใช้การถ่ายทอดสัญญาณแบบใด และถ่ายทอดข้อมูลได้มากที่สุดครั้งละเท่าใด
- **สตริงเดสคริปเตอร์ (String descriptor)** เป็นเดสคริปเตอร์ที่ไม่ได้อยู่ในโครงสร้างหลัก เพราะเดสคริปเตอร์ชนิดนี้จะเก็บข้อมูลตัวอักษรที่สามารถอ่านเข้าใจได้ไว้อธิบายส่วนต่างๆ ของเดสคริปเตอร์ทั้งสี่ตัวข้างต้น เช่น เก็บชื่อบริษัทผู้ผลิต และ/หรือ หมายเลขประจำตัวของอุปกรณ์ เป็นต้น
- **คลาสสเปกซิฟิกเดสคริปเตอร์ (Class-specific descriptor)** เป็นเดสคริปเตอร์พิเศษที่มีเฉพาะอุปกรณ์บางตัวที่จัดอยู่ในบางคลาสเท่านั้น ภายในเก็บข้อมูลเฉพาะของคลาสนั้นๆ ที่อยู่นอกเหนือจากเดสคริปเตอร์พื้นฐาน 4 ชนิดแรก
- **สรุปการใช้งานของเดสคริปเตอร์**
    - **ข้อมูลทั่วไปของตัวอุปกรณ์จะเก็บอยู่ในเดสคริปเตอร์ของอุปกรณ์ หรือ ดรไวซ์เดสคริปเตอร์**
    - **ข้อมูลการทำงานของแต่ละโหมดจะถูกเก็บอยู่ในคอนฟิกกูเรชั่นเดสคริปเตอร์**
    - **ข้อมูลหน้าที่การทำงานเก็บอยู่ในอินเทอร์เฟซเดสคริปเตอร์**
    - **ข้อมูลการทำงานของแต่ละเอ็นพอยท์จะถูกเก็บอยู่ในเอ็นพอยท์เดสคริปเตอร์**

2.2.7 การตรวจสอบการเชื่อมต่อของอุปกรณ์

ใช้การตรวจสอบจากการเปลี่ยนแปลงของระดับสัญญาณในสายสัญญาณดีลบ (D-) และดีบวก (D+) โดยสายข้อมูลทั้งสองที่ด้านฮับจะถูกต่อพูลดาวน์ (pull-down) ด้วยความต้านทาน 15 กิโลโอห์ม ซึ่งจะส่งผลให้สายสัญญาณทั้งสองมีระดับแรงดันเป็น 0 โวลต์ในขณะที่ไม่มีอุปกรณ์ใดๆ มาเชื่อมต่อ

ที่ด้านอุปกรณ์ สำหรับอุปกรณ์ความเร็วต่ำสายสัญญาณดีลบ จะต่อตัวต้านทาน 1.5กิโลโอห์ม พูลอัพ (pull-up) กับแรงดัน 3.0-3.6 โวลต์ ส่วนอุปกรณ์ความเร็วสูงสายสัญญาณดีบวก จะถูกพูลอัพแทน ดังนั้นเมื่อมีการต่ออุปกรณ์ตัวใหม่เข้าไปจะทำให้มีการเปลี่ยนแรงดันของสายสัญญาณเพิ่มจาก 0 โวลต์ ขึ้นไปที่ 90% ของไฟเลี้ยงจึงทำให้ฮับรู้ว่ามีการเชื่อมต่ออุปกรณ์เข้าไป

2.2.8 เทคนิคการเข้ารหัสสัญญาณ

การเข้ารหัสสัญญาณของยูเอสบี นั้นใช้เทคนิค 2 อย่างประกอบกันคือ เอ็นอาแซดไอ (NRZI : Non Return To Zero Inverted) และการเติมบิทสตัฟ (bit stuffing)

- เอ็นอาแซดไอ การเปลี่ยนแปลงระดับสัญญาณจากระดับสูงไประดับต่ำหรือระดับต่ำไประดับสูงจะถือเป็นข้อมูล "0" แต่ถ้าไม่เกิดการเปลี่ยนแปลงจะถือเป็นข้อมูล "1" ซึ่งกรณีนี้ถ้าข้อมูลเป็น "1" ติดต่อกันหลายๆ ตัวอาจทำให้เกิดความผิดพลาดในจังหวะการอ่านข้อมูลได้จึงต้องมีการเติมบิทสตัฟเข้ามาช่วย
- การเติมบิทสตัฟ เมื่อข้อมูลเป็น "1" ติดต่อกัน 6 บิท จะมีการเติมบิท "0" เข้าไปข้างหลัง และเวลาอ่านข้อมูลเมื่ออ่านเจอ "1" ทั้งหมด 6 บิทก็จะทำการอ่านบิท ที่ตามหลังมาว่าเป็น "0" หรือไม่ ถ้าเป็น "0" จะไม่นำมาคิด แต่ถ้าไม่ใช่ "0" จะถือว่าข้อมูลนั้นผิดพลาด

2.2.9 แนวทางเบื้องต้นในการพัฒนาอุปกรณ์สำหรับเชื่อมต่อกับพอร์ตยูเอสบี

เป็นที่ทราบทั่วไปว่ายูเอสบีสามารถต่อพ่วงอุปกรณ์ได้มาสุดถึง 127 ตัว มีความเร็วสูงผนวกเข้ากับคุณสมบัติปลั๊กแอนด์เพลอันเป็นปัจจัยช่วยเสริมให้ผู้ใช้งานสามารถใช้อุปกรณ์ที่ต่อผ่านทางพอร์ตยูเอสบีได้สะดวกสบายมากยิ่งขึ้น แต่กระบวนการในการพัฒนานั้นมีความซับซ้อนและต้องผ่านการพิจารณาอย่างครบถ้วนรอบด้านมากตามไปด้วย ทำให้มีขั้นตอนในการพัฒนาค่อนข้างมากทั้งทางด้านซอฟแวร์ และฮาร์ดแวร์

2.2.10 แนวทางในการพัฒนาฮาร์ดแวร์ของอุปกรณ์ยูเอสบี

ส่วนประกอบหลักของอุปกรณ์ยูเอสบีทางด้านฮาร์ดแวร์มี 5 ส่วนคือ

- ส่วนรับส่งข้อมูลกับพอร์ตยูเอสบี (Transceiver)
- วงจรเชื่อมต่อข้อมูลอนุกรม (Serial Interface Engine : SIE)
- ส่วนจัดการเชื่อมต่อกับเอสไออี (SIE interface)
- ส่วนควบคุมรูปแบบการเชื่อมต่อ (Protocol controller)
- ส่วนเชื่อมต่ออุปกรณ์อินพุตหรือเอาท์พุตพอร์ต (Input/Output : I/O)

ในปัจจุบันมีอุปกรณ์ที่รองรับการพัฒนาอุปกรณ์ยูเอสบีทั้งในแบบแยกส่วนและแบบรวมทั้งหมดอยู่ในอุปกรณ์เพียงตัวเดียว ซึ่งแบบแยกส่วนมักจะรวมเอาส่วนรับส่งข้อมูล , เอสไออี และส่วนจัดการเชื่อมต่อเอสไออีเข้าด้วยกันเป็นอุปกรณ์ 1 ตัว ในขณะที่ส่วนควบคุมรูปแบบการเชื่อมต่อและพอร์ตจะใช้ไมโครคอนโทรลเลอร์มาดำเนินการ เช่น พีดีไอยูเอสบี11 (PDIUSBD11) ของฟิลลิพ (Philips) สามารถเชื่อมต่อกับไมโครคอนโทรลเลอร์ ภายนอกผ่านระบบบัสไอสเควซี, ยูเอสบีเอ็น9603 (USBN9603) ของ เนชั่นเนลเซมิคอนดักเตอร์ (National Semiconductor) ใช้การเชื่อมต่อกับไมโครคอนโทรลเลอร์ในระบบบัส 3 สายคือ สายข้อมูลอนุกรมเข้า , สายข้อมูลอนุกรมออก และสายสัญญาณนาฬิกา เป็นต้น

สำหรับในแบบรวมส่วนประกอบทั้งหมดในอุปกรณ์เดียวกันนั้น จะเรียกอุปกรณ์นี้ว่า ยูเอสบีไมโครคอนโทรลเลอร์ (USB microcontroller) นั่นคือภายในไมโครคอนโทรลเลอร์จะรวมเอาอุปกรณ์ภาคที่ใช้ในการเชื่อมต่อกับพอร์ทยูเอสบีเข้าไว้ด้วย พร้อมกับจัดสรรฟังก์ชั่นพิเศษเพื่อรองรับการทำงานกับ พอร์ตยูเอสบี เพื่อช่วยให้การใช้งานสะดวกขึ้น ตัวอย่างของไมโครคอนโทรลเลอร์ชนิดนี้ได้แก่ พีไอซี16ซี745 (PIC16C745), พีไอซี18เอฟ2410 (PIC18F2410), พีไอซี18เอฟ4550 (PIC18F4550) ของไมโครชิพ (Microchip), ซีวาย7ซี63001 (CY7C63001) และโมดูลอีแซดยูเอสบี (module EZ-USB) ของ ไซเปรส (Cypress) เป็นต้น

2.2.11 สิ่งที่ต้องดำเนินการในการพัฒนาอุปกรณ์ยูเอสบี

หัวใจสำคัญในส่วนของฮาร์ดแวร์ ของอุปกรณ์ยูเอสบี คือ ไมโครคอนโทรลเลอร์โดยอุปกรณ์ยูเอสบีตัวหนึ่งๆจะมีการติดต่อกับโฮสต์ซึ่งใช้ยูเอสบีโปรโตคอลจากนั้นการถ่ายทอดข้อมูลจะเริ่มต้นขึ้นด้วย ดังนั้นอุปกรณ์ยูเอสบีจึงไม่อาจหลีกเลี่ยงการเขียนโปรแกรมและต้องเขียนมากกว่า 1 โปรแกรมด้วยเสมอ

งานที่จะต้องทำในการพัฒนาอุปกรณ์ยูเอสบี สามารถแบ่งออกได้เป็น 2 ส่วนคือ

- **งานที่ต้องทำในอุปกรณ์ยูเอสบี** มีอยู่ 3 อย่างคือ
    - ออกแบบฮาร์ดแวร์ของอุปกรณ์
    - เขียนโปรแกรมกำหนดรูปแบบการติดต่อกับพอร์ตยูเอสบีลงในไมโครคอนโทรลเลอร์
    - เขียนโปรแกรมควบคุมและใช้งานลงในไมโครคอนโทรลเลอร์
- **งานที่ต้องทำบนโฮสต์ (แพลตฟอร์มที่จะพัฒนา) มี** 3 อย่างเช่นกันคือ
    - ออกแบบฮาร์ดแวร์สำหรับการเชื่อมต่อ
    - **เขียนโปรแกรมไดร์เวอร์ของอุปกรณ์ ซึ่งโปรแกรมนี้จะทำงานในระดับเคอร์เนลสเปซ** (kernel space)
    - **เขียนแอปพลิชั่นโปรแกรมสำหรับใช้งาน โปรแกรมนี้จะอยู่ในระดับยูสเซอร์สเปซ** (user space)

2.2.12 ขั้นตอนโดยรวมในการพัฒนาอุปกรณ์ยูเอสบี

ขั้นตอนที่ 1 ออกแบบฮาร์ดแวร์ เป็นการออกแบบวงจรโดยรวมของอุปกรณ์ยูเอสบี ซึ่งผลลัพธ์ที่ได้ควรออกมาเป็นพีซีบีพร้อมกับลงอุปกรณ์

ขั้นตอนที่ 2 จัดการกับเดสคริปเตอร์ เป็นการเตรียมข้อมูลเพื่อกำหนดให้แก่ เดสคริปเตอร์ทั้งหมดที่ต้องใช้ในอุปกรณ์

ขั้นตอนที่ 3 เขียนโปรแกรมให้กับไมโครคอนโทรลเลอร์บนอุปกรณ์ยูเอสบีและยูเอสบีไดร์เวอร์บนโฮสต์ เมื่อได้วัตถุดิบและข้อมูลทั้งหมดแล้วขั้นตอนต่อไปคือการเขียนโปรแกรมให้กับ

ไมโครคอนโทรลเลอร์ หรือที่เรียกกันว่าเฟิร์มแวร์ (firmware) โดยสามารถแบ่งแยกออกเป็นส่วนๆ ได้ดังนี้

- ดีแคร์โมดูล (declare module) สำหรับกำหนดตัวแปรทั้งหมดลงในโปรแกรม
- เดสคริปเตอร์โมดูล (descriptor module) ใช้สำหรับกำหนดข้อมูลของเดสคริปเตอร์ทั้งหมด
- โมดูลเมน (main module) ใช้เป็นส่วนของโปรแกรมควบคุมหลัก
- โมดูลอินเทอร์รับ (interrupt module) เป็นโมดูลโปรแกรมย่อยสำหรับตอบสนองการอินเทอร์รับ
- โมดูลถอดระหัด (decode Module) เป็นโมดูลโปรแกรมย่อยสำหรับจัดการกับคำสั่งตั้งค่าต่างๆในโปรแกรมควบคุม

ส่วนการเขียนไดร์เวอร์นั้นคล้ายกับการทำให้โฮสต์รู้จักอุปกรณ์และสามารถทำงานร่วมกันได้

ขั้นตอนที่ 4 ติดต่อกับโฮสต์ ในขั้นตอนนี้เป็นการกำหนดรูปแบบข้อมูลที่ฮาร์ดแวร์ (ในที่นี้คือไมโครคอนโทรลเลอร์) จะติดต่อกับโฮสต์และทำการทดลองว่าอุปกรณ์ยูเอสบีสามารถติดต่อกับโฮสต์ได้หรือไม่ แต่จะยังไม่สามารถใช้งานหรือควบคุมอุปกรณ์ยูเอสบีตัวนี้ได้ จนกว่าจะมีโปรแกรมสำหรับใช้งาน

ขั้นตอนที่ 5 เขียนโปรแกรมใช้งาน ในที่นี้หมายถึงการพัฒนาแอปพลิเคชั่นโปรแกรมขึ้นมาเพื่อให้ผู้ใช้สามารถใช้งานอุปกรณ์ได้ ซึ่งโปรแกรมส่วนนี้จะถูกใช้บนโฮสต์

2.3 เอ็มเบดเดดลินุกซ์

ที่โครงงานเลือกใช้เอ็มเบดเดดลินุกซ์ก็เพราะความประหยัด และประโยชน์ทางด้านเทคนิค เราเล็งเห็นความเติบโตของการนำเอ็มเบดเดดลินุกซ์มาใช้สำหรับอุปกรณ์แบบสมองกลฝังตัว และเป็นแนวทางของการตลาดและเทคโนโลยีอย่างแท้จริง ลินุกซ์ก็เลยกลายเป็นส่วนหนึ่งของผลิตภัณฑ์ต่างๆ ไปทั่วโลก ลินุกซ์ถูกนำไปเป็นส่วนหนึ่งของ แอปพลิเคชั่น มือถือ, เกม และ พีดีเอ, ปริ้นเตอร์, สวิตช์ และเร้าเตอร์ (router) และผลิตภัณฑ์ อีกหลายตัว เอ็มเบดเดดลินุกซ์จึงเข้ามาและเติบโตอย่างต่อเนื่อง เหตุผลที่ทำให้เอ็มเบดเดดลินุกซ์เติบโตดังต่อไปนี้

- ลินุกซ์มีประสิทธิภาพสูง, สถานะคงตัว ซึ่งเป็นคุณสมบัติที่จะนำมาเป็น ระบบปฏิบัติการ

- ลินุกซ์เป็นฟรีแวร์
- ลินุกซ์สนับสนุน แอปพลิเคชั่น และ โปโตรคอลเน็ทเวิร์ค ที่หลากหลาย
- การเพิ่มขึ้นของจำนวนฮาร์ดแวร์และซอร์ฟแวร์ซึ่งจะมีโครงสร้างพื้นฐานขณะนี้สนับสนุนตัวลินุกซ์
- ลินุกซ์มีความน่าสนใจ ดึงดูดนักพัฒนา, สนับสนุน สถาปัตยกรรมฮาร์แวร์ชนิดใหม่, แพลตฟอร์ม, และอุปกรณ์อื่นๆ

และยังมีเหตุผลอื่นๆอีกมาก เราจึงเล็งเห็นแล้วว่าโครงงานนี้จะต้องใช้ลินุกซ์ เป็นส่วนหนึ่งของโครงงาน

2.3.1 โครงสร้างสถาปัตยกรรมเคอร์เนลของลินุกซ์

ถึงแม้ลินุกซ์เคอร์เนล จะออกมาหลายตัวอาทิเช่น เรดแฮด (Red Hat), ซูซี (SuSE), เดเบียน (Debian) แต่สถาปัตยกรรมโครงสร้างพื้นฐานนั้นก็คล้ายคลึงกัน หรือจะแตกต่างกันเพียงเล็กน้อยลิกนุกส์เคอร์เนลสามารถถูกแบ่งออกเป็นส่วนๆดังต่อไปนี้

- ชั้นฮาร์ดแวร์นามธรรม (Hardware abstraction layer)
- หน่วยจัดการหน่วยความจำ (Memory manager)
- หน่วยจักการตารางเวลา (Scheduler)
- ไฟล์ซิสเต็ม (File system)
- ระบบอินพุทเอาต์พุท (IO subsystem)
- ระบบเครือข่าย (Networking subsystem)
- ไอพีซี (IPC)

จึงสรุปแต่ละส่วน และรายละเอียดในการใช้งานในระบบฝังตัว สั้นๆดังต่อไปนี้

1. ชั้นฮาร์ดแวร์นามธรรม

โดยแท้จริงแล้วชั้นฮาร์ดแวร์นามธรรมนั้นก็คือฮาร์ดแวร์แพลตฟอร์ม นั่นเอง ดังนั้นไดร์เวอร์ต่างๆ สามารถถูกใส่ลงไปอย่างง่ายในฮาร์ดแวร์ที่ต่างๆกันไป สถาปัตยกรรมที่ดังๆก็ได้แก่ อาร์ม (ARM) และพาวเวอร์พีซี (PowerPC) ซึ่งมีสัญลักษณ์ ของโครงสร้างข้อมูลที่ดี และเอพีไอทำให้ง่ายในการพอร์ตสู่บอร์ดใหม่อย่างง่าย

แสดง โพรเซสเซอร์ฝังตัว ซึ่งสนับสนุน ลินุกซ์เคอร์เนล 2.6

- MIPS
- PowerPC

- ARM
- M68K
- CRIS
- V850
- SuperH

ชั้นฮาร์ดแวร์นามธรรมนั้นจะมี ส่วนประกอบฮาร์ดแวร์ต่อไปนี้

- **หน่วยประมวลผล, เคช และเอ็มเอ็มยู** (Processor, cache, and MMU)
- **การเซ็ทอัพเมมโมรี่แม็พ**(Setting up the memory map)
- **ส่วนสนับสนุนการยกเว้นและการตอบสนองอินเทอร์รับ** (Exception and interrupt handling support)
- **ดีเอ็มเอ** (DMA)
- **ไทม์เมอร์** (Timers)
- **ซิสเต็มคอนโซล** (System console)
- **การจัดการบัส** (Bus management)
- **การจัดการพลังงาน** (Power management)

2. **หน่วยจัดการหน่วยความจำ**

หน่วยจัดการหน่วยความจำในลินุกซ์จะมีหน้าที่รับผิดชอบสำหรับการควบคุมการเข้าถึงทรัพยากรหน่วยความจำฮาร์ดแวร์ หน่วยจัดการหน่วยความจำจะรับผิดชอบเตรียมการจัดหาหน่วยความจำที่เป็นไดนามิก ให้กับเคอร์เนลซัปซิสเต็ม อาทิเช่น ไดร์เวอร์, ไฟล์ซิสเต็ม, และชั้นของเน็ทเวิร์ค หน่วยจัดการหน่วยความจำจะมีซอร์ฟแวร์ที่จำเป็นเพื่อจัดการ หน่วยความจำเสมือน ให้กับโปรแกรมแอปพลิเคชั่น กระบวนการหนึ่งๆใน ลินุกซ์ซัปซิสเต็มจะกระทำบนตำแหน่งเสมือน ซึ่งตำแหน่งเสมือน นี้ กระบวนการ หนึ่งๆ สามารถจะทับซ้อนกับกระบวนการอื่นๆได้

ลินุกซ์เคอร์เนลแบ่งหน่วยความจำทั้งหมดออกเป็นเพจ ซึ่งมีขนาดแต่ละเพจเป็น 4 กิโลไบต์ ถึงแม้ว่าเพจจะถูกเข้าถึงโดยเคอร์เนล แต่ก็มีบางอันถูกใช้ในโดยเคอร์เนลเอง นอกจากนั้นก็จะถูกใช้โดยแอปพลิเคชั่น

3. **หน่วยจัดตารางเวลา**
4. **หน่วยจัดตารางเวลา**

จะจัดการการทำมัลติทาส และได้ถูกปล่อยออกมาโดยได้มุ่งไปที่หลักการจัดตารางเวลาแบบคาดเดาได้ (deterministic scheduling policy) ก่อนจะมาพูดถึงความเป็นมา มาทำความเข้าใจความหมายของรายการต่อไปนี้

- เคอร์เนลเธรด (Kernel thread) : ก็คือกระบวนการซึ่งไม่ได้ทำในส่วนยูซเซอร์คอนเท็กซ์ แต่จะทำงานในส่วนของเคอร์เนลสเปซ
- ยูสเซอร์โพรเซส (User process) : ยูสเซอร์โพรเซสหนึ่งจะมี พื้นที่ตำแหน่ง ซึ่งเป็นหน่วยความจำเสมือนจะสามารถเข้าสู่โหมดเคอร์เนลก็ต่อเมื่อเกิด อินเตอรัพต์, เอ็กเซ็บชั่น, หรือ ซิสเต็มคอลล์ถูกทำงาน
- ยูสเซอร์เธรด (User thread) : เธรดจะทำงานแตกต่างกันไปซึ่งจะถูกแม็ปเข้ากับ ยูสเซอร์โพรเซส หนึ่งๆ ยูสเซอร์เธรดจะแชร์ คอมมอนเท็กซ์, ข้อมูล, และพื้นที่ฮีป (heap) มีพื้นที่สแต็ก (stack) ที่แยกต่างหาก ทรัพยากรอื่นๆอาทิเช่น ไฟล์, ซิกแนลแฮนเดิล จะถูกแชร์กันข้ามเธรด

5. ลินุกซ์ เริ่มต้นเป็นที่นิยม เป็นที่ต้องการสำหรับงาน ระบบทันเวลา ผลลัพธ์ก็คือ หน่วยจัดตารางเวลาจึงปรับปรุงแก้ไขและกลายมาเป็นแบบคาดเดาได้ (deterministic) ต่อไป เป็นการแสดง ไมล์สโตน(mile stone) ที่สำคัญของการพัฒนาการในลินุกซ์เคอร์เนล และลักษณะสำคัญ
   - เริ่มจาก 1.3.55 เคอร์เนล สนับสนุนการทำราวด์โรบิน (round robin) และ การจัดตารางเวลาโดยใช้ไฟโฟ่เป็นหลัก (FIFO-based scheduling) พร้อมด้วย การจัดตารางแบบแบ่งเวลา (classic time-sharing scheduler) ง่ายในการดิสเอเบิลเพจจิ้ง (disable paging) สำหรับเลือกขอบเขตหน่วยความจำของแอปพลิเคชั่นอีกด้วย เป็นเหตุให้หน่วยความจำถูกล็อก ก็เพราะเพจจิ้งสร้างจากระบบ ที่คาดเดาไม่ได้ (nondeterministic)
   - 2.0 เคอร์เนล มีฟังก์ชันใหม่คือ nanosleep() อนุญาตให้โพรเซสหยุดพักเป็นเวลาสั้นๆ
   - 2.2 เคอร์เนล สนับสนุนโพซิกเรียลไทม์ซิกเนล (Posix real-time signals)
   - 2.4 เคอร์เนล ถูกปรับปรุงอย่างมาก และมุ่งไปที่ การจัดตารางเวลาที่ทันเวลา และในที่สุดก็กลายมาเป็น2.6 เคอร์เนล

- 2.6 เคอร์เนล มีตารางเวลาขึ้นมาใหม่ซึ่งนำการคาดเดาได้ เข้าไปใน นโยบาย การจัดการเวลา และ ลักษณะทันเวลา อาทิเช่น โพซิกไทม์เมอร์ (POSIX timers) อีกด้วย

6. ไฟล์ซิสเต็ม

บนลินุกซ์ ไฟล์ซิสเต็มทั้งหลายถูกจัดการ โดยวีเอฟเอส (VFS : Virtual File System) ซึ่งวีเอฟเอส จะจัดการมุมมองข้อมูลที่ถูกเก็บไว้ในแต่ละอุปกรณ์บนระบบ สิ่งที่ทำคือทำการแยกมุมมองของผู้ใช้งานโดยใช้ ซิสเต็มคอลล์พื้นฐาน แต่ก็อนุญาต ให้นักพัฒนาเคอร์เนล ทำการอิมพรีเม้นท์ ลอจิคอลไฟล์ซิสเต็ม บนอุปกรณ์

อุปกรณ์ที่เป็นลินุกซ์ใดๆ หรือ ระบบฝังตัว ใดๆจะต้องมีไฟล์ซิสเต็ม อย่างน้อย 1 ไฟล์ซิสเต็ม อันนี้จะไม่เหมือนกับระบบทันเวลาที่ไม่จำเป็นจะต้องมีไฟล์ซิสเต็มใดๆ ลินุกซ์จะต้องมีไฟล์ซิสเต็ม ก็มีสาเหตุมาจาก

- แอปพลิเคชั่นต่างก็มีหลายตัว เนื่องจากเหตุนี้จึงจำเป็นต้องมีพื้นที่สตอเรจ (starage space) ในไฟล์ซิสเต็ม
- ดีไวซ์ส่วนมากเข้าถึงแบบเป็นไฟล์

พร้อมด้วยลินุกซ์ สนับสนุนไฟล์ซิสเต็มแต่ละชนิดเช่นแฟลชและรอมสำหรับ ระบบฝังตัว ยังสนับสนุนเอ็นเอฟเอส (NFS) อีกด้วย อันซึ่งอนุญาตให้ ไฟล์ซิสเต็ม บนเครื่องโฮสถูกเม้าท์สู่ ระบบฝังตัว ลินุกซ์สนับสนุนเมโมรี่เบสไฟล์ซิสเต็ม (memory-base file system) อันซึ่งใช้ใน ระบบฝังตัวอีกเช่นกัน ต่อไปเป็นการแสดงชนิดต่างๆที่ใช้ใน ไฟล์ซิสเต็มฝังตัว

- EXT2
- CRAMFS
- ROMFS
- RAMFS
- JFFS2
- PROCFS
- DEVFS

7. ระบบอินพุทเอาต์พุท

ระบบอินพุทเอาต์พุทบนลินุกซ์จัดการการเชื่อมต่ออย่างง่ายๆ และแบบยุ่งยากกับ ดีไวซ์บนบอร์ด มีสามชนิดดีไวซ์ที่ถูกสนับสนุนโดย หน่วยอินพุทเอาต์พุทบน

- ชาเล็กเตอร์ดีไวซ์ (Character devices) สนับสนุนดีไวซ์แบบลำดับ

- บล๊อกดีไวซ์ (Block devices) สนับสนุนดีไวซ์แบบแรนดอม, บล๊อกดีไวซ์ เป็นหัวใจในการสร้างไฟล์ซิสเต็ม
- เน็ทเวิร์คดีไวซ์ (Network devices) สนับสนุน การเชื่อมต่อที่เป็นเน็ตเวิร์ค

8. ระบบเครือข่าย

หนึ่งในหลายๆจุดแข็งของลินุกซ์ ก็คือการมีการสนับสนุน โปโตคอลเน็ทเวิร์ค ที่หลากหลาย ตารางต่อไปเป็นการแสดงคุณลักษณะของแต่ละเคอร์เนลเวอร์ชั่น

ตารางที่ 2.2 เปรียบคุณลักษณะทางเน็ตเวิร์กของลินุกซ์

	Kernel Availability		
Feature	*2.2*	*2.4*	*2.6*
Layer 2			
Support for bridging	Yes	Yes	Yes
X.25	Yes	Yes	Yes
LAPB	Experimental	Yes	Yes
PPP	Yes	Yes	Yes
SLIP	Yes	Yes	Yes
Ethernet	Yes	Yes	Yes
ATM	No	Yes	Yes
Bluetooth	No	Yes	Yes
Layer 3			
IPV4	Yes	Yes	Yes
IPV6	No	Yes	Yes
IP forwarding	Yes	Yes	Yes
IP multicasting	Yes	Yes	Yes
IP firewalling	Yes	Yes	Yes
IP tunneling	Yes	Yes	Yes
ICMP	Yes	Yes	Yes
ARP	Yes	Yes	Yes
NAT	Yes	Yes	Yes
IPSEC	No	No	Yes
Layer 4 (and above)			
UDP and TCP	Yes	Yes	Yes
BOOTP/RARP/DHCP	Yes	Yes	Yes

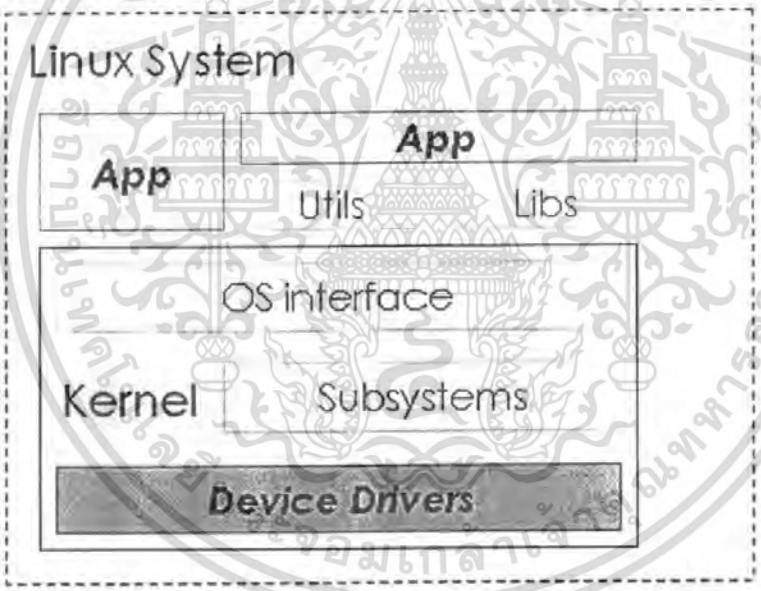
9. ไอพีซี (IPC : Interprocess communication)

ไอพีซีภายในลินุกซ์นั้นจะประกอบด้วยสัญญาน(ใช้สำหรับ asynchronous communication), ไปป์, และ ซ็อกเก็ต เช่นเดียวกับ System V IPC mechanisms ประกอบด้วย การแชร์หน่วยความจำ, แมสเซส คิว (message queues), และ เซมาฟอร์(semaphores) ซึ่งเคอร์เนล 2.6ได้เพิ่มสนับสนุน POSIX-type message queues เข้าไป

2.3.2 ยูสเซอร์ สเปซ (User Space)

ยูสเซอร์ สเปซ ในลินุกซ์จะมีหลักการดังต่อไปนี้

- โปรแกรม : จะเป็น อิมเมจ และอยู่บนไฟล์ซิสเต็ม เมื่อแอปพลิเคชั่นต้องการขึ้นมาทำงาน อิมเมจจะถูกโหลดลงสู่หน่วยความจำและรัน
- หน่วยความจำเสมือน: อนุญาตให้แต่ละโพรเซสมี พื้นที่ตำแหน่ง และยังอนุญาตคุณสมบัติพิเศษอาทิเช่น แชร์ไลบรารี่ (shared library) โพรเซสหนึ่งๆจะมี เมมโมรี่แม็ปใน ตำแหน่งพื้นที่เสมือน ซึ่งจะต้องมีหนึ่งเดียว และไม่อิสระแยกจากเคอร์เนลเมโมรี่แม็ป (kernel memory map)
- ซิสเต็มคอลล์ : เป็นการส่งให้เคอร์เนลทำงาน ดังนั้นเคอร์เนลสามารถประมวลผลบริการต่างๆ ในนามของแอปพลิเคชั่นที่เรียกใช้



รูปที่ 2.4 แสดงส่วนประกอบของระบบลินุกซ์

2.3.3 ความหมายของระบบสมองกลฝังตัว

เป็นระบบอิเล็กทรอนิกส์ที่ใช้สำหรับงานควบคุมรวมถึงการแสดงผลการทำงานต่าง ๆ โดยที่ระบบเหล่านี้ถูกใช้เป็นส่วนหนึ่งของระบบและอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรต่าง ๆ การที่ใช้คำว่า "ระบบสมองกลฝังตัว" เนื่องจากระบบเหล่านี้เป็นส่วนหนึ่งของระบบใหญ่ ในหลายกรณีที่ผู้ใช้ทั่วไปอาจไม่ทราบว่าอุปกรณ์ควบคุม เครื่องมือ เครื่องจักรรวมถึงระบบใดที่ใช้งานเป็นประจำเหล่านั้น

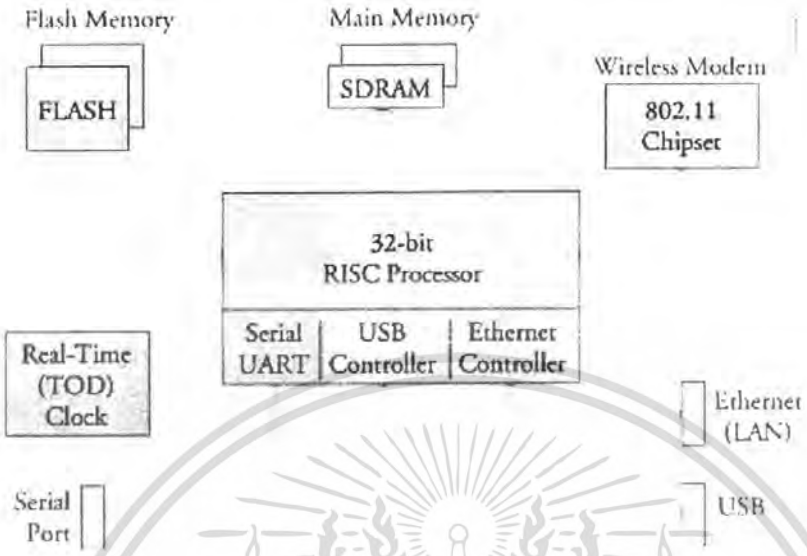
เป็นระบบสมองกลฝังตัว ในบางครั้งแม้แต่ผู้ที่มีความรู้ทางด้านเทคนิคก็ไม่สามารถระบุได้แน่ชัดว่าสิ่งใดมีระบบสมองกลฝังตัวอยู่ จนกว่าจะมีการทำงานและตรวจสอบกับระบบและอุปกรณ์ควบคุมนั้นระยะหนึ่งเลยที่เดียว ระบบสมองกลฝังตัว นี้แม้ไม่ใช่เครื่องคอมพิวเตอร์ แต่ก็มีระบบคอมพิวเตอร์อยู่ภายใน อาจจะเป็นเพียงไมโครโพรเซสเซอร์ (Microprocessor) หรือชิป์ (chip) ธรรมดาหรือโพรเซสเซอร์ (Processor) ที่ประกอบด้วย ชิป (Chip) ที่มีวงจรซับซ้อน โดยจะมีหลักการทำงาน คือ มีสัญญาณข้อมูลเข้า (Input) จากอุปกรณ์ เซนเซอร์ (Sensor) เข้าสู่ระบบ และมีสัญญาณผลลัพธ์ (Output) ของระบบไปควบคุมบังคับสวิตช์เครื่องควบคุมต่าง ๆ เช่นสวิตช์เครื่องจักร หรือ วาล์วควบคุมทิศทางการไหลของท่อทางต่าง ๆ

ระบบฝังตัว นั้นมีรูปร่าง ขนาด ที่หลากหลาย จากขนาดที่ใหญ่สุด มัลติเพิล-แรค ดาต้า สตอเรจ (multiple-rack data storage) หรือ เน็ทเวิร์ค พาเวอร์เฮาส(network powerhouse) ไปจนถึงขนาดเล็กๆ อาทิเช่น เอ็มพีสาม (MP3) ส่วนบุคคล หรือ เซลลูล่า แฮดเซ็ต (cellular handset) ของนั้นมักจะมีคุณสมบัติของ ระบบฝังตัว ประกอบด้วย

- มีเครื่องประมวลผม
- ระบบฝังตัว ส่วนใหญ่ต้องทันเวลา
- พัฒนา และตรวจสอบข้อผิดพลาดยาก
- ถูกดีไซน์ เฉพาะเจาะจงใน แอปพลิเคชั่น หรือวัตถุประสงค์
- มีขนาดทรัพยากรที่จำกัด อาทิเช่น หน่วยความจำ(ROM, RAM) อินพุทเอาต์พุท
- อาจจะมีพลังงาน ที่จำกัด
- มักจะมุ่งที่งานนั้นๆ โดยที่ปราศจากมนุษย์เข้ามาเกี่ยวข้อง

ตัวอย่างระบบฝังตัว

แสดงไดอะแกรมของระบบฝังตัวซึ่งเป็นตัวอย่างอย่างง่ายๆของสถาปัตยกรรมฮาร์ดแวร์ ซึ่งอาจจะเจอในวายเลสแอ็คเซสพอยท์ (wireless access point) ระบบนี้มี 32 บิท ริสโพรเซสเซอร์ เป็นหัวใจของระบบ, มีหน่วยความจำแฟลชใช้ในการเก็บข้อมูล มีหน่วยความจำหลักเป็นเอสดีแรม (SDRAM),โมดูล เรียว ไทม์ ค็อก (real-time clock), อีเทอร์เน็ท (Ethernet) , วายเลส โมเด็ม (Wirless modem), พอร์ตอนุกรม (Serial port) และ ยูเอสบี อินเทอร์เฟส (USB interface)

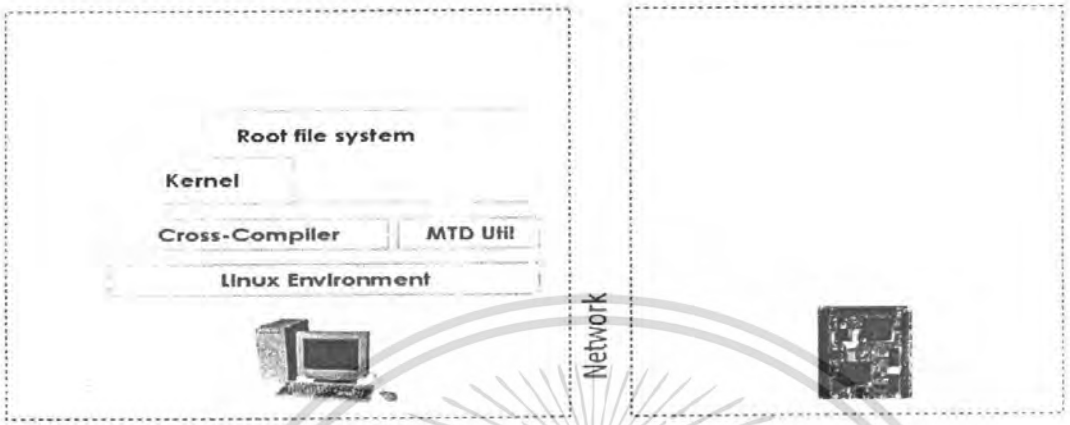


รูปที่ 2.5 แสดงตัวอย่างระบบฝังตัว

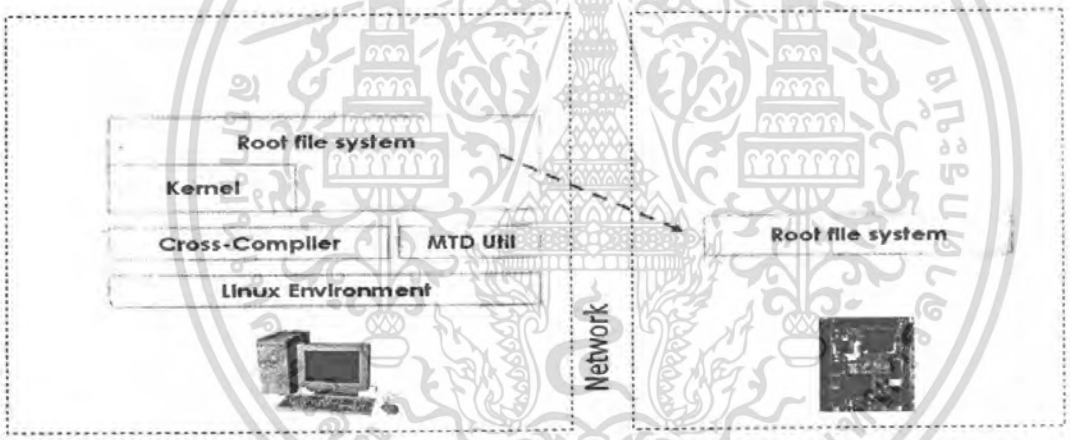
2.3.4 ลำดับการพัฒนา

เป็นแนวทางลำดับของการพัฒนาระบบฝังตัวซึ่งจะต้องสร้างสิ่งต่างที่เครื่องพีซีประกอบด้วย ลินุกซ์ เอ็นวิรอนเม็นท์ (linux environment), คอสคอมไพล์ (cross-compiler), เอ็มทีดี ยูทิริตี้ (MTD Utilities), ลินุกซ์เคอร์เนล (Linux kernel), รูทไฟล์ซิสเต็ม (Root file system)

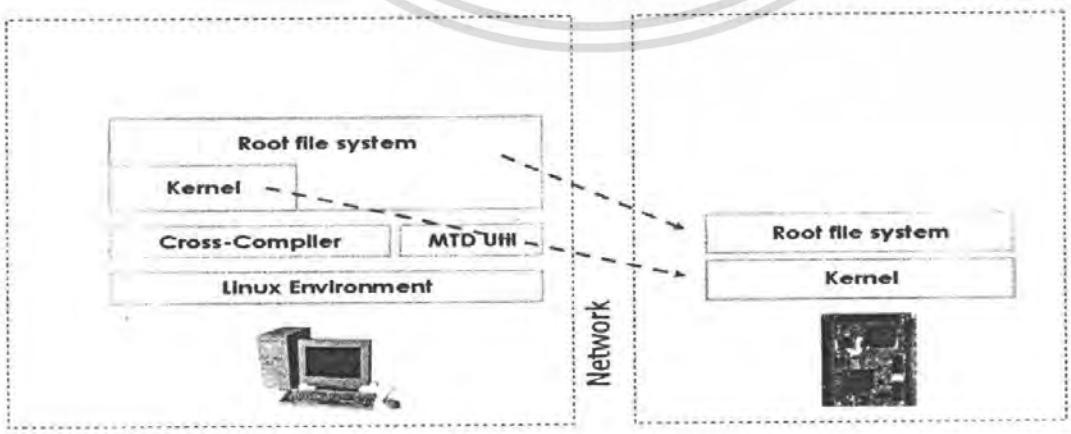
จากนั้นก็ทำการโหลดรูทไฟล์ซิสเต็ม, ลินุกซ์เคอร์เนล สู่บอร์ดตามลำดับ ในการสร้างแอปพลิเคชั่น และโมดูล ต่างๆ อาทิเช่นดีไวซ์ไดร์เวอร์นั้นจะต้องเขียน และคอมไพล์ที่เครื่องพีซีโดยใช้ คอสคอมไพล์ จากนั้นจึงโหลดลงบอร์ดต่อไป



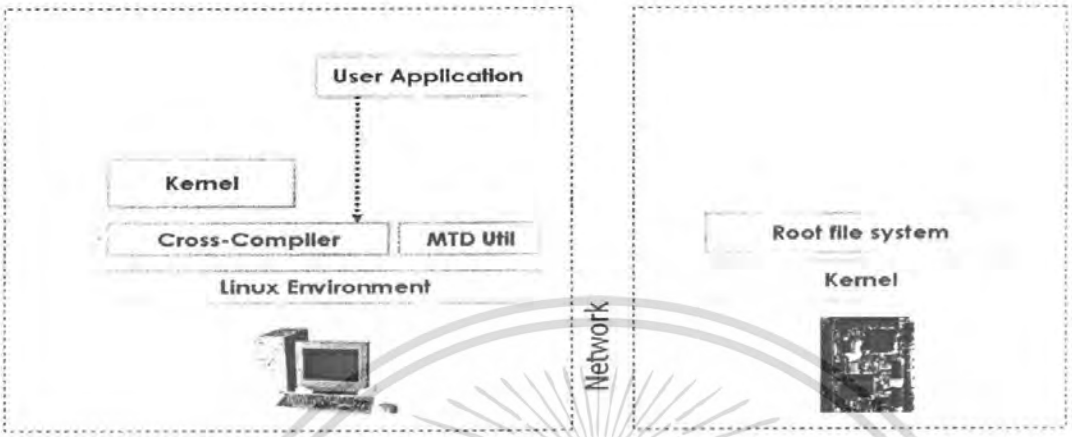
(ก)



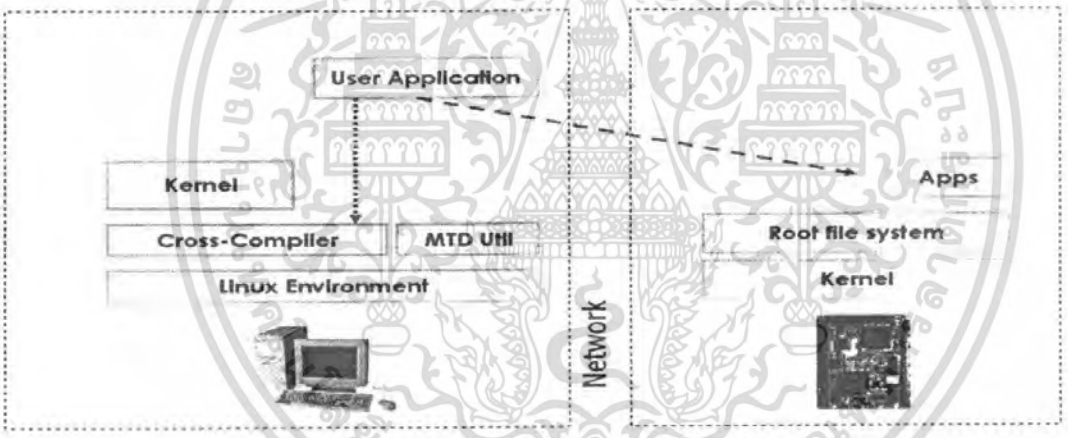
(ข)



(ค)

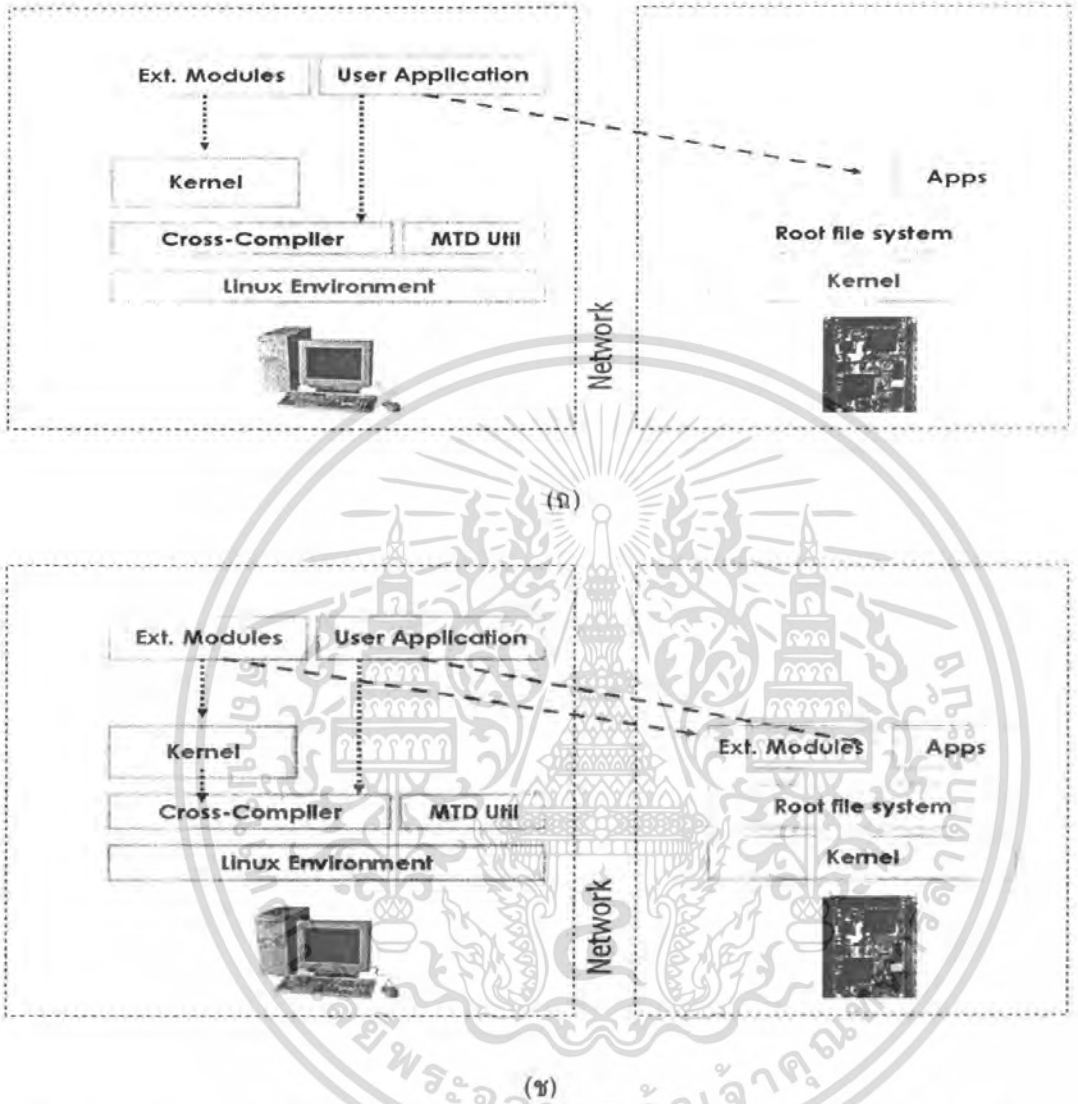


(ง)



(จ)

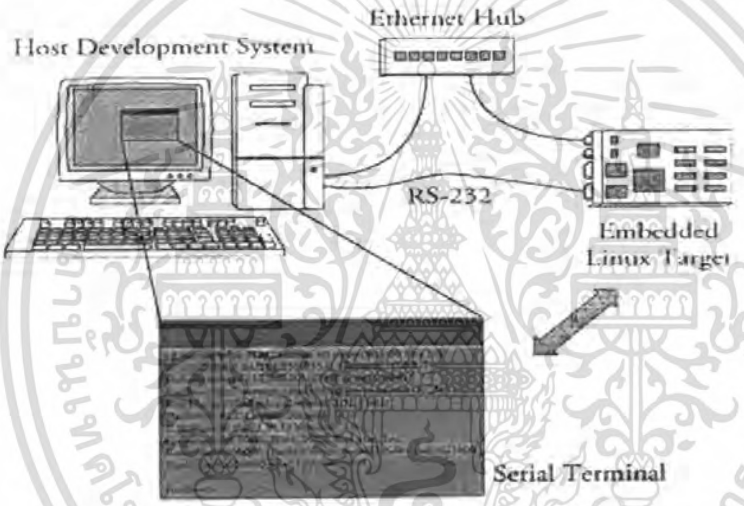
6.



รูปที่ 2.6 แสดงขั้นตอนในการพัฒนา (ก) ลำดับที่ 1, (ข) ลำดับที่ 2, (ค) ลำดับที่ 3, (ง) ลำดับที่ 4, (จ) ลำดับที่ 5, (ฉ) ลำดับที่ 6, (ช) ลำดับที่ 7

2.3.5 การติดตั้ง เอ็มเบดเดด ลินุกซ์ (Embedded Linux development setup)

รูปนี้แสดงการติดตั้งแบบง่ายๆ มีเครื่องที่ใช้ในการพัฒนา(host development) ซึ่งลินุกซ์ จะรันอยู่แล้วแต่ชนิดของลินุกซ์ ที่ชื่นชอบ อาทิเช่น เช่น เรดแฮด, ซูซี, เดเบียน เป็นต้น บอร์ดเอ็มเบดเดดลินุกซ์ นั้นๆ จะถูกเชื่อมต่อกับพีซีผ่านทางพอร์ตอนุกรม (RS-232 serial) หรือจะเชื่อมต่อกับบอร์ดผ่านทางอีเทอร์เน็ทผ่านทาง ฮับ หรือ สวิตช์ เครื่องพัฒนาจะประกอบไปด้วย ทูล (tools) และ ยูทิริตี้ (utilities) ตามแล้วแต่ ชนิดของ เอ็มเบดเดด ลินุกซ์ นั้น



รูปที่ 2.7 การติดตั้งบอร์ด

การเริ่มทำงานของบอร์ด

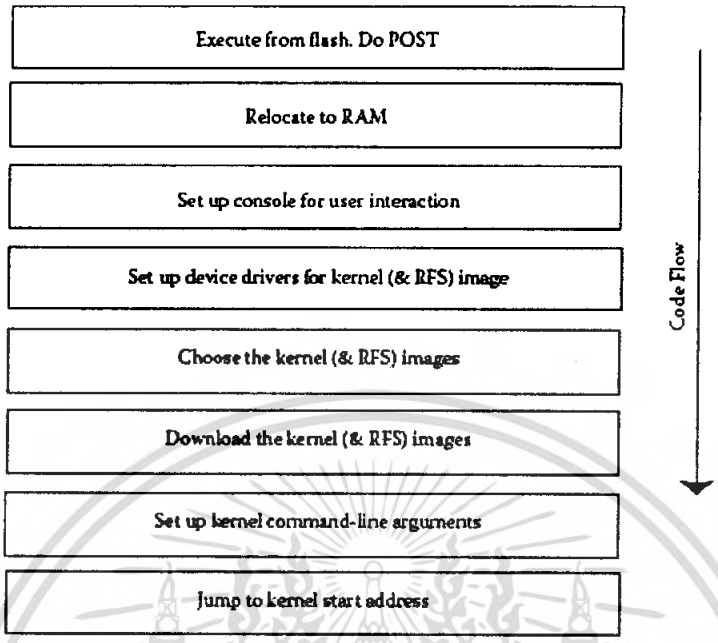
เมื่อเริ่มทำงานเปิดสวิตช์พาเวอร์ขึ้นมานั้น ในช่วงขณะนี้ บูทโหลดเดอร์ จะเป็นตัวที่ควบคุมการทำงานของโพรเซสเซอร์ทั้งหมด กระทำการให้ฮาร์ดแวร์ถูกกำหนดค่าเริ่มต้นประกอบด้วย โพรเซสเซอร์ และหน่วยความจำ ถูกติดตั้ง, กำหนดค่ายูอาร์ท (UART)ไปควบคุมพอร์ตอนุกรม และกำหนดค่าเริ่มต้นอีเทอร์เน็ทคอนโทรเลอร์
และ จะแสดงค่าต่างๆออกทาง พอร์ตอนุกรม

```
U-Boot 1.1.4 (Mar 18 2006 - 20:36:11)

AMCC PowerPC 440EP Rev. B
Board: Yosemite - AMCC PPC440EP Evaluation Board
         VCO: 1066 MHz
         CPU: 533 MHz
         PLB: 133 MHz
         OPB: 66 MHz
         EPB: 66 MHz
         PCI: 66 MHz
I2C:   ready
DRAM:  256 MB
FLASH: 64 MB
PCI:   Bus Dev VenId DevId Class Int
In:    serial
Out:   serial
Err:   serial
Net:   ppc_4xx_eth0, ppc_4xx_eth1

=>
```

รูปที่ 2.8 ภาพแสดงการกำหนดค่าเริ่มต้นในส่วนพอร์ตอนุกรม

การกำหนดค่าเริ่มต้นของฮาร์ดแวร์

- **คอนฟิคความเร็วซีพียู**
- **หน่วยความจำถูกกำหนดค่าเริ่มต้น, และหาขนาดหน่วยความจำของบอร์ด**
- **เซ็ตพอร์ตอนุกรม สำหรับ บูทคอนโซล์**
- **ตรวจสอบฮาร์ดแวร์ (POST : Power ON Self-Test)**



รูปที่ 2.9 ขั้นตอนทำงานของบูทโหลดเดอร์

หลังจากทำขั้นตอนข้างบนเสร็จเรียบร้อยแล้วนั้น ขั้นตอนต่อไปเป็นการ โหลด ลินุกซ์เคอร์เนล บูทเคอร์เนลหลังจากที่บูทโหลดเดอร์ทำการกำหนดค่าเริ่มต้นเสร็จสิ้น ต่อไปบูทโหลดเดอร์ก็จะโหลด และบูท ลินุกซ์เคอร์เนล ซึ่งบูทโหลดเดอร์ ทั้งหลาย จะมี คำสั่งในการโหลด และ เอ็กซีคิว (execute) ตัว โอเอส อิมเมจ รูปต่อไปเป็นการแสดงตัวอย่างหนึ่ง ที่บูทโหลดเดอร์(u-boot) โหลด และ บูทลินุกซ์เคอร์เนล

```
=> tftpboot 200000 uImage-440ep
ENET Speed is 100 Mbps - FULL duplex connection
Using ppc_4xx_eth0 device
TFTP from server 192.168.1.10; our IP address is 192.168.1.139
Filename 'uImage-amcc'.
Load address: 0x200000
Loading: ###################################################################
         ##########################################
done
Bytes transferred = 962773 (eb0d5 hex)

=> bootm 200000
## Booting image at 00200000 ...
   Image Name:   Linux-2.6.13
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    962709 Bytes = 940.1 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.6.13 (chris@junior) (gcc version 4.0.0 (DENX ELDK 4.0 4.0.0))
  #2 Thu Feb 16 19:30:13 EST 2006
AMCC PowerPC 440EP Yosemite Platform
...
< Lots of Linux kernel boot messages, removed for clarity >
...
amcc login:    <<< This is a Linux kernel console command prompt
```

รูปที่ 2.10 แสดงการโหลด ลินุกซ์เคอร์เนล

บูทโหลดเดอร์จะต้องหาที่อยู่ของเคอร์เนล อิมเมจอันซึ่งอาจจะอยู่ในระบบแฟลชหรือ อาจจะอยู่บนเน็ทเวิร์กไม่ว่าจะอยู่ไหนอิมเมจจะต้องถูกโหลดสู่หน่วยความจำในกรณีของอิมเมจที่ถูกบีบอัดมา อิมเมจจะต้องถูกคลายก่อน และถ้าอินนิเชียลแรมดิส (initial randisk) บูทโหลดเดอร์จะต้องโหลดอิมเมจของอินนิเชียลแรมดิส สู่หน่วยความจำอีกด้วย ตำแหน่งที่อยู่หน่วยความจำที่จะถูกเคอร์เนลอิมเมจโหลดนั้น บูทโหลดเดอร์จะเป็นผู้ตัดสินว่าจะอยู่ตรงไหน โดยอ่านจากอีเอ็ฟเอ็ฟ เฮดเดอร์(EFF header) ของเคอร์เนลอิมเมจ

ก่อนที่จะถึงช่วงลอกอินช่วงซีเรียล เทอร์มินอล (serial terminal) ลินุกซ์จะทำการเม้าท์รูทไฟล์ซิสเต็ม รูทไฟล์ซิสเต็มนั้นจะบรรจุ โปรแกรมแอปพลิเคชั่น, ระบบ ไลบรารี, และ ยูทิริตี้ ที่สร้างขึ้นจากระบบจีเอ็นยู/ลินุกซ์ (GNU/Linux system)

มีรูทไฟล์ซิสเต็มสามชนิดที่ใช้ในระบบฝังตัว

- อินนิเชียลแรมดิส
- Network-based file system using NFS

- Flash-based file system

จนกระทั่งมาถึงตรงนี้เคอร์เนลกำลังเอ็กซีคิวโค้ดกระทำการกำหนดค่าเริ่มต้นต่างๆมากมาย ภายใน คอนเท็กท์ (context) ที่เรียกกันว่า เคอร์เนลคอนเท็กท์ เคอร์เนลนั้นจะเข้าถึงหน่วยความจำแบบฟิซิคอล และ อินพุทเอาต์พุท ซับซิสเต็ม ทั้งหมด

เมื่อลินุกซ์เคอร์เนลกระทำการกำหนดค่าเริ่มต้นเสร็จสิ้น และทำการเม้าท์รูทไฟล์ซิสเต็ม โดยปกติจะเรียกโปรแกรมแอปพลิเคชั่น ขึ้นมาทำงาน ก็คือ init เมื่อเคอร์เนลล์รัน init จะรันอยู่ในส่วนของยูซเซอร์ หรือยูซเซอร์คอนเท็กท์โหลดการทำงานจะอยู่ในส่วนของยูซเซอร์การเข้าถึงซิสเต็มจะต้องใช้ k เคอร์เนลซิสเต็มคอลล์ในการเรียกใช้บริการเคอร์เนลอาทิเช่น ดีไวซ์และ ไฟล์ อินพุตเอาต์พุทเป็นต้น ยูซเซอร์สเปซโพรเซสหรือโปรแกรมจะทำงานบนพื้นที่หน่วยความจำเสมือน

2.3.6 ระบบไฟล์

ในหัวข้อนี้จะกล่าวถึงระบบไฟล์ที่เป็นที่นิยมกัน โดยส่วนใหญ่จะใช้ แฟลช (flash-based) และนอกจากนั้นก็จะใช้แรม (momory-based) แบบใช้แรมสามารถถูกใช้ช่วงเวลาบูทเพื่อที่จะถึงรูทไฟล์ซิสเต็ม และสำหรับเก็บข้อมูลที่เป็น โวเลนไทล์ (volatile) ที่ไม่จำเป็นต้องบันทึก

- แรมดิสค์ (Ramdisk)

แรมดิสค์เป็นสิ่งทางเลือกหนึ่งของระบบไฟล์ของลินุกซ์ โดยการจำลองให้มีฮาร์ดดิสค์โดยใช้หน่วยความจำ จะใช้แรมดิสค์เมื่อไม่มีอุปกรณ์ความจำอาทิเช่นฮาร์ดดิสค์ หรือ แฟลช สำหรับเก็บรูทไฟล์ซิสเต็ม โดยแท้จริงแล้วแรมดิสค์ไม่ใช้ไฟล์ซิสเต็มแต่ใช้กลไกการทำงานโดยสามารถโหลดไฟล์ซิสเต็มลงสู่หน่วยความจำและใช้งานรูทไฟล์ซิสเต็ม

อินนิทอาร์ดี (Initrd) ได้จัดเตรียมกลไกกระบวนการโดยบูทโหลดเดอร์ทำการโหลดเคอร์เนลอิมเมจ กับ รูทไฟล์ซิสแต็มลงสู่หน่วยความจำการใช้อินนิทอาร์ดีจะต้องมีขั้นตอนที่จะต้องดำเนินการดังต่อไปนี้

- สร้างอินนิทอาร์ดีอิมเมจ และแพ็คมันไว้กับเคอร์เนล อิมเมจคุณจะต้องสร้างแรมดิสค์อิมเมจบนเครื่องโฮส ต่อไปคุณจะต้องแพ็คแรมดิสค์อิมเมจกับเคอร์เนล และออปชั่นจะต้องตรงกับตัวแพลตฟอร์มเมื่อทำการตอนบูทเคอร์เนล โดยทั่วไปจะมีอีแอลอี เซ็คชั่น (ELE section) เรียกว่า .initrd ซึ่งจะเป็นตัวจับแรมดิสค์อิมเมจ อันซึ่งสามารถทำให้บูทโหลดเดอร์ใช้ได้โดยตรง
- เปลี่ยนจากบูทโหลดเดอร์ เป็นโหลดอินนิทอาร์ดี

อินนิจอาร์ดียังมีกลไกให้คุณสามารถที่จะสลับเปลี่ยนไปใช้รูทไฟล์ซิสเต็มใหม่ หลังจากช่วงของระบบทำงาน คุณสามารถใช้อินนิทอาร์ดีสำหรับการกู้ และอัพเกรดอีกด้วยเมื่อผลิตภัณฑ์ถูกปล่อยออกไป

- แรมเอ็ฟเอส (RAMFS)

บ่อยครั้งที่ระบบฝังตัวมีไฟล์ และไม่ต้องการมาใช้อีกเมื่อรีบูต และเพียงต้องการมาเก็บไว้ในไดเรคทรอรี /tmp การที่เก็บไฟล์ไว้ในหน่วยความจำแทนที่จะเก็บไว้ในแฟรชก็เพราะว่าการใช้งานแฟรชนั้นมีราคาสูง คุณควรใช้ แรมเอ็ฟเอส แทนไม่มีการจำกัดขนาด

- คอมเพรชแรมเอ็ฟเอส (CRAMFS(Compressed RAM File System))

นี่เป็นไฟล์ซิสเต็มที่มีประโยชน์มากสำหรับแฟรชและได้ออกมาในเคอร์เนล2.4 การอ่านมีการบีบอัดสูง คอมเพรชแรมเอ็ฟเอส เป็นไฟล์ซิสเต็มที่มีกฎเกณฑ์กล่าวคือมันจะต้องกระทำผ่านบัฟเฟอร์ แคช (buffer cache)เพื่อบอกบล็อก ดีไวซ์ (block device)ในการเข้าถึงข้อมูล คุณจะต้องเปิดการใช้งานเอ็มทีดี (MTD) บล็อก ดีไวซ์ ไดร์เวอร์ โหมด คอมเพรชแรมเอ็ฟเอส ใช้แซสลิบ (zlib) สำหรับการบีบอัด และบีบอัดทุกๆ 4 กิโลไบต์(page size) ในการจะเบิร์นคอมเพรชแรมเอ็ฟเอสอิมเมจสู่แฟรชสามารถทำได้โดยใช้โปรแกรมที่เรียกว่า mkcramfs

- เจเอ็ฟเอ็ฟเอส (Journaling Flash File System-JFFS และ JFFS2)

เราต้องการแฟรชไฟล์ซิสเต็มก็เพราะ

- ข้อมูลไม่สูญหายเมื่อระบบไม่ทำงาน
- ใช้ MTD-level เอพีไอในการเข้าถึงชั้นของแฟรชได้โดยตรง

ในปี1999 Axis Communicationได้ปล่อย เจเอ็ฟเอ็ฟเอส สำหรับ ลินุกซ์เคอร์เนล 2.0 และมีคุณลักษณะข้างบนทั้งหมด และได้ใส่ลงมาใน 2.2 และ 2.4 ในเวลาต่อมา แต่มันยังขาดการสนับสนุนการบีบอัด โครงการ เจเอ็ฟเอ็ฟเอสทู (JFFS2) จึงได้เริ่มกำเนิดขึ้นมา เจเอ็ฟเอ็ฟเอสทูถูกปล่อยลงสู่เคอร์เนล2.4 และแซงหน้า เจเอ็ฟเอ็ฟเอส เพราคุณลักษณะที่โดดเด่นกว่า ทั้ง เจเอ็ฟเอ็ฟเอส และ เจเอ็ฟเอ็ฟเอสทู เป็น log-structure file system ก็คือเปลี่ยนจาก เรคคอร์ด (recorded)กลายเป็น ล๊อก (log)

ภายในล๊อกจะบรรจุสิ่งต่อไปนี้

- การระบุตัวตน (Identification)
- เวอร์ชัน ซึ่งจะเป็นหนึ่งเดียวกับล๊อก เป็นส่วนหนึ่งของรายละเอียดไฟล์
- เมตาดาต้า (Metadata) อาทิเช่น ไทม์สแตมป์ (timestamp)

- ข้อมูล และขนาดของข้อมูล
- ออปเซ็ต (Offset)ของข้อมูลในไฟล์

การเขียนไฟล์จะสร้างล๊อก และเมื่อต้องการที่จะอ่านล๊อกจะกลับมากลายเป็นไฟล์โดยใช้ ขนาดข้อมูล และออปเซ็ตไฟล์ก็จะถูกสร้างขึ้นมาใหม่ จากนั้นล๊อกก็จะกลายมารอการทำลายโดยการ์เบก คอลเลคชั่น (garbage collection)

คุณลักษณะสำคัญของระบบไฟล์แบบเจเอ็ฟเอ็ฟเอสทูดังต่อไปนี้

- การจัดการการลบบล็อก (Management of erase block):ใน เจเอ็ฟเอ็ฟเอสทู การลบบล็อกมีสามลิสต์ คลีน (clean), เดอร์ตี้ (dirty), และฟรี (free) ลิสต์clean จะบรรจุเพียงแต่ วาลิด ล๊อก(valid log) ลิสต์dirtyบรรจุ เพียงหนึ่งหรือหลายล๊อกที่ข้อมูลกำลังจะหมดไปเมื่อการ์เบก คอลเลคชั่นถูกเรียกทำงาน ลิสต์freeบรรจุ โนล๊อก (no log) และจากนี้ไปสามารถที่จะถูกเรียกใช้ไปเป็น ล๊อกใหม่
- การ์เบก คอลเลคชั่น:เจเอ็ฟเอ็ฟเอสทู การ์เบก คอลเลคชั่น เกิดขึ้นในเธรดที่แยกต่างหาก อันซึ่งจะเริ่มมาคุณเม้าท์ เจเอ็ฟเอ็ฟเอสทูไฟล์ซิสเต็ม เจเอ็ฟเอ็ฟเอสทูจะสงวนห้าบล็อกสำหรับการทำการ์เบก คอลเลคชั่น
- การบีบอัด (Compression):ข้อแตกต่างระหว่าง เจเอ็ฟเอ็ฟเอส กับ เจเอ็ฟเอ็ฟเอสทู ก็ตรงที่ เจเอ็ฟเอ็ฟเอสทูมีการบีบอัดโดยใช้ แซสลิบ (zlib) และ รูบิน (rubin)

ทั้งเจเอ็ฟเอ็ฟเอสและเจเอ็ฟเอ็ฟเอสทูไฟล์ซิสเต็มอิมเมจ สามารถถูกสร้างขึ้นจากคำสั่ง mkfs.jffs และ mkfs.jffs2 คำสั่งทั้งสองจะกระทำบนเครื่องโฮสโฮสจะต้องโหลดสู่บอร์ดแล้วเบิร์นสู่แฟรชในภายหลัง

- เอ็นเอ็ฟเอส (NFS-Network File System)

เอ็นเอ็ฟเอสสามารถใช้ในการเม้าท์ไฟล์ซิสเต็มบนเน็ทเวิร์กในช่วงของการพัฒนา นักพัฒนามันจะใช้เอ็นเอ็ฟเอสเนื่องจาก การใช้แฟลชมีราคาค่อนข้างสูงเนื่องจากแฟรชมีข้อจำกัดของการเขียนที่จำกัด แต่เอ็นเอ็ฟเอส ไม่จำกัดขนาดเนื่องจากสตอเรจ (storage)ทั้งหมดจะอยู่ที่รีโมท (remote) เซิร์ฟเวอร์หรือเครื่องโฮส

ลินุกซ์เคอร์เนล ได้จัดเตรียมกลไกสำหรับ การเม้าท์แบบอัติโนมัติ (automatically mounting) โดยใช้เอ็นเอ็ฟเอสซึ่งจะต้องทำตามขั้นตอนดังต่อไปนี้

- คอนฟิคออปชั่น CONFIG_NFS_FS อันซึ่งจะอนุญาตให้สามารถเม้าท์ไฟล์ซิสเต็มจาก รีโมทเซิร์ฟเวอร์ และ CONFIG_ROOT_NFS อันซึ่งอนุญาตให้ใช้ เอ็นเอ็ฟเอส ในตอนเวลาที่สร้างเคอร์เนล

- คอนฟิคหมายเลขไอพีหรืออาจจะใช้ BOOTP, RARP, หรือ DHCP ในการ คอนฟิคแบบอัตโนมัติผ่านเน็ตเวิร์ก (turn on network auto configuration)
- ตัวแปรเคอร์เนลคอมมานด์ไลน์ (Kernel command-lind parameter) จะต้องเป็นชนิดเอ็นเอ็ฟเอส

สำหรับรายละเอียดของรูปแบบคำสั่งของตัวแปรอาร์กิวเมนท์ คอมมานด์ไลนด์ นั้นจะถูกอธิบายไว้ในไฟล์ Docomentation/nfsroot.txt ใน kernel source tree

2.3.7 ยูบูท (U-boot)

ยูบูท ได้ให้คำสั่งมากมายสำหรับการโปรแกรม การลบ การป้องกันหน่วยความจำแฟลชซึ่งในจะขอกล่าวถึงและชี้นำการใช้งานคำสั่งและแก้ไขตัวแปรสภาพแวดล้อม

-ข้อมูลข่าวสารของแฟลช

จะใช้คำสั่ง 'flinfo' ในการที่จะเข้าถึงอุปกรณ์แฟลช,ข้อมูลอุปกรณ์ซึ่งจะประกอบไปด้วย ขนาด,ขนาดการลบในแต่ละบล๊อก และการแม็ปหน่วยความจำ

ตัวอย่าง:

```
U-Boot>  flinfo
Bank # 1: CFI conformant FLASH (32 x 16)  Size: 8 MB in 32 Sectors
 Erase timeout 16384 ms, write timeout 3 ms, buffer write timeout 3 ms, buffer size 32
  Sector Start Addresses:
    28000000 (RO) 28040000 (RO) 28080000    280C0000    28100000
    28140000      28180000      281C0000    28200000    28240000
    28280000      282C0000      28300000    28340000    28380000
    283C0000      28400000      28440000    28480000    284C0000
    28500000      28540000      28580000    285C0000    28600000
    28640000      28680000      286C0000    28700000    28740000
    28780000      287C0000
U-Boot>
```

รูปที่ 2.11 แสดงคำสั่ง flinfo

ผลลัพธ์จะแสดงข้อมูลจำนวนมาก ตัวอย่างเช่น เลย์เอาต์ เซกชั่นของแฟลช, ตำแหน่งเริ่มต้นของเซกชั่น สำหรับในตัวอย่างนี้จะมี 256 กิโลไบต์ 32 เซกชั่นเริ่มที่ตำแหน่ง 0x28000000 รวมเป็น 8 เมกะไบต์.

-แสดงตัวแปรสภาพแวดล้อม

จะใช้คำสั่ง 'printenv' ซึ่งคำสั่งนี้จะแสดงค่าตัวแปลสภาพแวดล้อม

ตัวอย่าง:

```
OMAP5912 OSK # printenv
baudrate=115200
bootdelay=20
ethaddr=00:0E:99:xx:xx:xx
bootcmd=bootp; bootm
stdin=serial
stdout=serial
stderr=serial
```

รูปที่ 2.12 แสดงคำสั่ง printenv

-เซ็ตค่าตัวแปรสภาพแวดล้อม

จะใช้คำสั่ง 'setenv' เป็นการแก้ไขค่าของตัวแปรสภาพแวดล้อมซึ่งถ้าคุณใส่อาร์กิวเมนท์ไปตัวเดียวจะเป็นการลบตัวแปรสภาพแวดล้อมนั้นออกไป

ตัวอย่าง:

```
OMAP5912 OSK # setenv bootcmd 'cp 0x00040000 0x10000000 0x00200000;bootm'
```

รูปที่ 2.13 แสดงคำสั่ง setenv

-เซฟค่าตัวแปรสภาพแวดล้อม

จะใช้คำสั่ง 'saveenv' เมื่อใดก็ตามที่คุณทำการเปลี่ยนแปลงค่าของตัวแปรสภาพแวดล้อมการกระทำนั้นๆจะกระทำบนแรมเพียงอย่างเดียว เมื่อใดก็ตามที่คุณทำการรีบูทสิ่งที่ได้เปลี่ยนแปลงนั้นก็จะหายหมด ถ้าคุณต้องการที่จะเปลี่ยนแปลงอย่างถาวร คุณควรที่จะใช้คำสั่งนี้ ซึ่งจะเป็นการบันทึกค่าตัวแปรสภาพแวดล้อมเมื่อเราทำการแก้ไขโดยใช้คำสั่ง 'setenv'

ตัวอย่าง:

```
OMAP5912 OSK # saveenv
Saving Environment to Flash...
Un-Protected 1 sectors
Erasing Flash...
Erasing sector  1 ...  done
Erased 1 sectors
Writing to Flash.../done
Protected 1 sectors
```

รูปที่ 2.14 แสดงคำสั่ง saveenv

-ทีเอ็ฟทีพี (Tftp) Data

จะใช้คำสั่ง 'tftp' เป็นการโอนถ่ายข้อมูลโดยใช้โปรโตคอลทีเอ็ฟทีพี ซึ่งจะต้องโหลดไฟล์มาเก็บไว้ที่แรมก่อน

ตัวอย่าง:

```
OMAP5912 OSK # tftp 0x10000000 osk1/u-boot-1.1.2.bin
Using MAC Address 00:0E:99:xx:xx:xx
TFTP from server 192.168.1.10; our IP address is 192.168.1.56
Filename 'osk1/u-boot-1.1.2.bin'.
Load address: 0x10000000
Loading: ###################
done
Bytes transferred = 87764 (156d4 hex)
```

รูปที่ 2.15 แสดงคำสั่ง tftp

-ลบแฟลช

จะใช้คำสั่ง 'erase' ซึ่งก็คือการลบข้อมูลออกจากแฟลชนั้นเอง

ตัวอย่าง:

```
OMAP5912 OSK # erase 0x01000000 0x001fffffff
Erasing sector 128 ...  done
...
```

รูปที่ 2.16 แสดงคำสั่ง erase

-โปรแกรมแฟลช

จะใช้คำสั่ง 'cp' ซึ่งก็คือการก๊อปปี้ข้อมูลโดยจะต้องระบุตำแหน่งจุดตั้งต้น, จุดก๊อปปี้ และก็จำนวนครั้ง ตัวอย่าง:

```
OMAP5912 OSK # cp 0x10000000 0x00040000 0x00200000
Copy to Flash...\done
```

รูปที่ 2.17 แสดงคำสั่ง cp

-บูทหน่วยความจำ

จะใช้คำสั่ง 'bootm' ซึ่งคำสั่งนี้จะเป็นการเริ่มทำงานของซิสเต็มอิมเมจ คำสั่งนี้จะโหลดไฟล์อิมเมจไปไว้นะที่ตำแหน่งหนึ่งๆแล้วทำการแตก(uncompressing)

คำสั่งที่กล่าวถึงนี้เป็นคำสั่งที่ควรที่จะรู้ไว้ใช้งานโดยคำสั่งที่กล่าวถึงไปนั้นเป็นคำสั่งที่ต้องใช้งาน จึงได้ยกตัวอย่างคำสั่งเพียงเท่านี้ คำสั่งและข้อมูลเกี่ยวกับยูบูทนั้นยังมีอีกมากมาย

2.3.8 แอปพลิเคชั่นสำหรับเอ็มเบดเดดลินุกซ์

ในหัวข้อนี้จะขอกล่าวถึงบางแอปพลิเคชั่น ที่จะใช้ในเอ็มเบดเดดลินุกซ์ซิสเต็ม

บิวซีบล็อก (Busybox)

โปรแกรมบิวซีบล็อกเป็นมัลติคอลล์ (multicall) โปรแกรม บิวซีบล็อกบรรจุโปรแกรมหลักๆที่เป็นที่รู้จักดังต่อไปนี้

- เชลล์ (Shells) อาทิเช่น the ash, lash, hush, และอื่นๆ
- คอร์ยูทิริตี้ (Core utilities) อาทิเช่น cat, chmod, cp, dd, mv, ls, pwd, rm,
- โพรเซสคอนโทร และ มอนิทอริ่ง ยูทิริตี้ (Process control and monitoring utilities) อาทิเช่น ps, kill, และอื่นๆ
- โมดูลโหลดดิ่งยูทิริตี้ (Module-loading utilities) อาทิเช่น lsmod, rmmod, modprobe, insmod,

  and depmod
- ซิสเต็มทูล (System tools) อาทิเช่น reboot, init, และอื่นๆ
- เน็ตเวิร์ก ยูทิริตี้ (Networking utilities) อาทิเช่น ifconfig, route, ping, tftp, httpd,

telnet, wget, udhcpc (dhcp client), และอื่นๆ

- ยูทิริตี้สำหรับการลอกอิน และพลาสเวิร์ด (Log-in and password management utilities) อาทิเช่น login, passwd,
  adduser, deluser, และอื่นๆ
- อาร์คฮิเวอร์ ยูทิริตี้ (Archival utilities) อาทิเช่น ar, cpio, gzip, tar, และอื่นๆ
- ซิสเต็ม ล็อกอินยูทิริตี้ (System logging utilities) อาทิเช่น syslogd

การสร้างบิวซีบล็อกแบ่งเป็นสองขั้นตอนดังนี้

- คอนฟิค:ใช้คำสั่ง make menuconfig แล้วเลือกว่าจะเอาอะไรบ้าง
- สร้างบิวซีบล็อก:ใช้คำสั่ง make

ขั้นตอนต่อไปเป็นการติดตั้ง บิวซีบล็อก ลงสู่เป้าหมายของคุณ กระทำได้โดยการเรียก บิวซีบลียก กับออปชั่น –intall ยกตัวอย่าง

busybox mount –n –t proc /proc /proc

busybox –install –s

คำสั่งการติดตั้งบิวซีบล็อกนั้นจะสร้าง ซอร์ฟ ลิงค์ (soft link) ยกตัวอย่างเช่นหลังจากติดตั้งเสร็จแล้วใช้คำสั่ง ls –l ในไดเรคทรอรี /bin จะแสดงผลลัพธ์ออกมาดังนี้

```
-rwxr-xr-x  1 0      0        1065308 busybox
lrwxrwxrwx  1 0      0              7 init -> busybox
lrwxrwxrwx  1 0      0             12 ash -> /bin/busybox
lrwxrwxrwx  1 0      0             12 cat -> /bin/busybox
lrwxrwxrwx  1 0      0             12 chmod -> /bin/busybox
lrwxrwxrwx  1 0      0             12 cp -> /bin/busybox
lrwxrwxrwx  1 0      0             12 dd -> /bin/busybox
lrwxrwxrwx  1 0      0             12 echo -> /bin/busybox
```

รูปที่ 2.18 แสดงคำสั่ง ls -l

ทีนีล็อกอิน(Tinylogin)

ทีนีลอกอิน เป็นมัลติคอลล์โปรแกรม เช่นเดียวกับบิวซีบล็อกและใช้สำหรับการพัฒนายูนิกซ์ล็อกอิน(UNIX log-in) และ เข้าถึงแอปพลิเคชั่น แสดงฟังก์ชันที่ใช้ทีนีล็อกอิน

- เพิ่ม และ ลบ ผู้ใช้งาน
- ล็อกอิน และ เกตตี้แอปพลิเคชั่น (getty application)
- เปลี่ยนพลาสเวิร์ด

เอ็ฟทีพี เซิร์ฟเวอร์ (Ftp Server)

เอ็ฟทีพี เซิร์ฟเวอร์ ใช้ในการก๊อปปี้ไฟล์จากหรือไปที่เอ็มเบดเดดซิสเต็ม

2.3.9 เคอร์เนล โมดูล

เคอร์เนล โมดูลจะถูกเพิ่มเข้าไปแบบไดนามิคขณะเคอร์เนลกำลังทำงาน มีส่วนประกอบสามส่วนก็คือ

- โมดูล อินเทอร์เฟส/เอพีไอ (Module interface/APIs)
- การสร้างโมดูล(Module building)
- การโหลดโมดูลเข้าและนำโมดูลออก (Module loading and unloading)

ทั้งสามส่วนได้มีการเปลี่ยนแปลงจาก 2.4 เคอร์เนล ไปสู่ 2.6 เคอร์เนล

โมดูลเอพีไอ

ภาพข้างล่างแสดงตัวอย่างเคอร์เนลของ 2.4 และ 2.6 kernel โมดูลทั้งสองจะแสดงสตริง Hello world ทุกๆครั้งเมื่อทำการโหลด และจะแสดงสตริง Bye world ทุกๆครั้งที่โมดูลถูกเอาออก

- การเข้า (Entry) และการออก (exit) ฟังก์ชัน

ทุกๆโมดูลจะต้องมีฟังก์ชัน ลงชื่อ และ ออก ซึ่งจะถูกเรียกอย่างอัตโนมัติโดยเคอร์เนล เมื่อโมดูลถูกโหลด และ ถูกเอาออกโดยลำดับ ใน 2.4 เคอร์เนล ใช้ฟังก์ชัน init_module() และ cleanup_module()

ในการ ลงชื่อ และเอาออก ส่วนใน 2.6 เคอร์เนลจะใช้มาโคร module_init() และ module_exit()

- การส่งผ่านตัวแปร (Parameter passing)

ทุกๆโมดูลสามารถถูกส่งตัวแปรได้ โดยการส่งทางคอมมานไลค์ เมื่อโมดูลถูกโหลด ใน2.4 เคอร์เนล จะใช้มาโคร MODULE_PARM ส่วน2.6 เคอร์เนล จะใช้มาโครmodule_param()

```
/* 2.4 kernel based module */

static int excount = 1;
MODULE_PARM(excount,"i");
static int init_module(void)
{
  int i;
  if(excount <= 0) return -EINVAL;
  for(i=0; i<excount;i++)
    printk("Hello world\n");
  return 0;
}

static void cleanup_module(void)
{
  printk("Bye world\n");
}

/* 2.6 kernel based module code */

MODULE_LICENSE("GPL");
module_param(excount, int, 0);
static int init_module(void)
{
  int i;
  if(excount <= 0) return -EINVAL;
  for(i=0; i<excount;i++)
    printk("Hello world\n");
  return 0;
}

static void cleanup_module(void)
{
printk("Bye world\n");
}

module_init(init_module);
module_exit(cleanup_module);
```

รูปที่ 2.19 แสดงโปรแกรมไดร์เวอร์ตัวอย่าง

- ทุกๆโมดูลมีการใช้การนับแสดงจำนวนการอ้างถึงโมดูล จำนวน เท่ากับ 0 หมายความว่าโมดูลยังไม่ได้ถูกโหลด
- ทุกๆโมดูลต้องประกาศ กระทำได้โดยใช้มาโคร MODULE_LICENSE

2.3.10 ชาเล็กเตอร์ดีไวซ์ไดร์เวอร์ (Charactor device driver)

ชาเล็กเตอร์ดีไวซ์ไดร์เวอร์จะถูกเข้าใช้งานโดยผ่านทางชื่อในไฟล์ซิสเต็ม ชื่อเหล่านี้เรียกว่า สเปเชียลไฟล์ (special file) หรือ ดีไวซ์ไฟล์ หรือ โหนด ซึ่งอยู่ในไดเรคทรอรี /dev ซึ่งจะมีเครื่องหมายเป็น "c" คอลัมท์ แรกของผลลัพธ์เมื่อใช้คำสั่ง ls –l Block device ก็อยู่ใน/devเช่นกัน แต่จะมีเครื่องหมายเป็น

"b" เมื่อคุณใช้คำสั่ง ls –l จะเห็นเลขสองตัว ก่อน วันเดือนปี เลขทั้งสองก็คือ หมายเลขดีไวซ์หลัก และ หมายเลขดีไวซ์รอง ซึ่งจะเป็นเฉพาะแต่ละดีไวซ์ ต่อไปเป็นการแสดงเลขทั้งสอง

```
crw-rw-rw-    1 root     root       1,   3 Apr 11  2002 null
crw-------    1 root     root      10,   1 Apr 11  2002 psaux
crw-------    1 root     root       4,   1 Oct 28 03:04 tty1
crw-rw-rw-    1 root     tty        4,  64 Apr 11  2002 ttyS0
crw-rw----    1 root     uucp       4,  65 Apr 11  2002 ttyS1
crw--w----    1 vcsa     tty        7,   1 Apr 11  2002 vcs1
crw--w----    1 vcsa     tty        7, 129 Apr 11  2002 vcsa1
crw-rw-rw-    1 root     root       1,   5 Apr 11  2002 zero
```

รูปที่ 2.20 แสดงคำสั่ง ls –l ใน /dev

หมายเลขดีไวซ์หลัก จะมีความสัมพันธ์กับดีไวซ์ลินุกซ์เคอร์เนล สมัยใหม่จะอนุญาตให้ดีไวซ์หลายตัวแชร์หมายเลขดีไวซ์หลักได้ แต่โดยจะยังคงหนึ่งหมายเลขดีไวซ์หลักต่อหนึ่งดีไวซ์ ส่วนหมายเลขดีไวซ์รองถูกใช้โดยเคอร์เนลเพื่อหาดีไวซ์ที่แท้จริง ภายในkernelจะมีชนิดข้อมูล dev_t ซึ่งถูกประกาศใน linux/types.h ถูกใช้เพื่อการดึงหมายเลขdevice ทั้งในส่วนของ หมายเลขดีไวซ์หลัก และหมายเลขดีไวซ์รอง ในเวอร์ชัน 2.6.0 เคอร์เนล dev_t จะมีทั้งหมด 32บิท โดยแบ่งออเป็น 12บิท สำหรับ หมายเลขดีไวซ์หลัก และ 20บิทสำหรับ หมายเลขดีไวซ์รอง ภายใน linux/kdev_t.h จะมีมาโครที่ใช้ในการดึงเลข หมายเลขดีไวซ์หลัก และ หมายเลขดีไวซ์รอง โดยใช้

MAJOR(dev_t dev);

MINOR(dev_t dev);

MKDEV(int major, int minor); จะได้ค่าออกมาเป็นชนิดข้อมูล dev_t

สิ่งแรกที่จะต้องทำเมื่อทำการติดตั้งชาเล็กเตอร์ดีไวซ์ไดร์เวอร์จำเป็นต้องมีการใช้ฟังก์ชัน register_chrdev_region ซึ่งถูกประกาศอยู่ใน linux/fs.h สุดท้ายแล้วชื่อควรจะปรากฏใน /proc/devices และ sysfs register_chrdev_region จะส่งค่าคืนเป็น 0 ถ้าทำการติดตั้งอย่างเสร็จสมบูรณ์ ในส่วนของกรณีผิดพลาด ค่าที่ส่งคืนจะเป็นเลขติดลบ

int register_chrdev_region(dev_t first, unsigned int count,char *name);

register_chrdev_region จะเป็นฟังก์ชันที่ใช้เมื่อรู้หมายเลขดีไวซ์ ที่ต้องการ แต่ถ้าไม่รู้จะใช้หมายเลขดีไวซ์หลักอะไร ลินุกซ์ก็มีฟังก์ชันที่ใช้ในการจองหมายเลขดีไวซ์แบบไดนามิกก็คือฟังก์ชัน alloc_chrdev_region

int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,unsigned int count, char *name);

ฟังก์ชันนี้ dev เป็นผลลัพธ์อย่างเดียว และจะได้ค่าหมายเลขดีไวซ์หลักออกมา และค่าหมายเลขดีไวซ์รองที่ใส่เข้าไปนั้นโดยปกติแล้วจะใช้ 0

ส่วนการปล่อยหมายเลขดีไวซ์ จะใช้ฟังก์ชัน unregister_chrdev_region

void unregister_chrdev_region(dev_t first, unsigned int count);

2.3.11 ไฟล์ โอเปอเรชั่น (File Operation)

จากที่ผ่านมาได้กล่าวถึงแต่หมายเลขดีไวซ์ที่จะเอาไปใช้ แต่ไม่ได้ยังไม่ได้กล่าวว่าจะติดต่อกระทำการใดๆกับดีไวซ์ได้อย่างไร file_operatoins structureเป็นตัวที่จะจัดการติดตั้งการเชื่อมต่อ สตรักเจอร์ (structure) ซึ่งถูกประกาศไว้ใน linux/fs.h จะเป็นรวบรวม พอยท์เตอร์ (pointer) ของฟังก์ชันตัวอย่าง ไฟล์ โอเปอเรชั่นสตรักซึ่งภายในจะมีฟิวที่เป็นพอยท์เตอร์ใช้ไปยังฟังก์ชันที่ถูกเขียนขึ้นเพื่อปฏิบัติงานหนึ่งๆ หรือถ้าเป็น นูล (NULL) ก็จะหมายความว่าไม่สนับสนุนการปฏิบัติงานชนิดนั้นๆ

```
struct file_operations scull_fops = {
    .owner =    THIS_MODULE,
    .llseek =   scull_llseek,
    .read =     scull_read,
    .write =    scull_write,
    .ioctl =    scull_ioctl,
    .open =     scull_open,
    .release =  scull_release,
};
```

รูปที่ 2.21 แสดงไฟล์ โอเปอเรชั่น

การรีจิสเตอร์ชาเล็กเตอร์ดีไวซ์ไดร์เวอร์

ก่อนที่เคอร์เนลจะเรียกโอเปอเรชั่นต่างๆของดีไวซ์นั้น ก่อนอื่นจะต้องจอง และ รีจิสเตอร์สตรักก่อน ก่อนอื่นจะต้องinclude <linux/cdev.h>และจะมีสองฟังก์ชันในการกำหนดค่าเริ่มต้นคือ

void cdev_init(struct cdev *cdev, struct file_operations *fops);

และสุดท้ายใช้ฟังก์ชัน

int cdev_add(struct cdev *dev, dev_t num, unsigned int count);

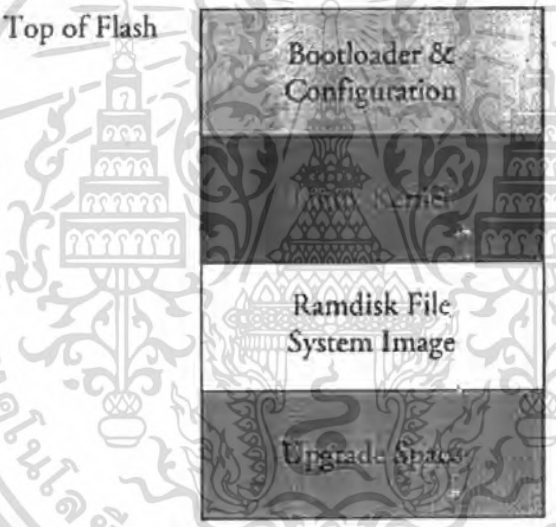
2.3.12 เอ็มเบดเดด สตอเรจ (Embedded Storage)

จากแนวทางที่สืบต่อกันมา สตอเรจ ในระบบฝังตัวจะใช้รอมในการอ่านโค้ดอย่างเดียว และ เอ็นวีแรม (NVRAM) สำหรับการอ่านและเขียนข้อมูล แต่อย่างไรก็ตามได้กลายมาเป็นเทคโนโลยีแฟลช อันซึ่ง มี

ความเป็น นอนวาเลนไทล์ (nonvolatile)อย่างสูง ประกอบกับราคาไม่สูงแฟลช จึงมาใช้ในระบบฝังตัวอย่างมากขึ้น

การแม็ปแฟลช
ในระบบฝังตัวนั้นแฟลชจะถูกใช้โดยทั่วไปดังต่อไปนี้

- ส่วนเก็บบูทโหลดเดอร์
- ส่วนเก็บโอเอสอิมเมจ
- ส่วนเก็บแอปพลิเคชั่นอิมเมจ ไลบรารี่อิมเมจ
- ส่วนเก็บการอ่านเขียนไฟล์



รูปที่ 2.22 แสดงโครงสร้างหน่วยความจำแฟลชทั่วๆไปในระบบฝังตัว

บ่อยครั้งที่บูทโหลดเดอร์จะอยู่ในส่วนบนสุด หรือล่างสุดของหน่วยความจำแฟลชและก็เรียงตามลำดับคือลินุกซ์เคอร์เนลและไฟล์ซิสเต็มเอ็มเจ

หนึ่งในการเพิ่มพูนประสิทธิภาพของแฟลชไฟล์ซิสเต็มนั้นกระทำได้ แฟรชบล็อกนั้นมีระยะเวลาการใช้งานที่ถูกจำกัด จึงต้องใส่วิธีการในการลบบล็อกแฟลชที่แตกต่างกันไป

ข้อจำกัดอื่นอาจเกิดขึ้น เนื่องจากสถาปัตยกรรมของแฟลชซึ่งมีความเสี่ยง ที่ข้อมูลจะสูญหายในระหว่างที่ ไฟไม่ทำงานหรือ พรีมาเตอร์ชัตดาวน์ (premature shutdown) ก็เนื่องจากว่าขนาดแฟลชบล็อก ค่อนข้างที่จะใหญ่ และขนาดของไฟล์โดยเฉลี่ยแล้วจะเล็กมาก เมื่อเทียบกับขนาดของ

บล็อก เมื่อถูกเขียนลงไป ซึ่งในการเขียนแต่ละครั้งจะต้องกระทำเป็นบล็อก เช่นนั้นในการเขียนไฟล์เล็กๆซึ่งมีขนาดเท่ากับ 8กิโลไบต์ จะต้องลบ และเขียนใหม่เป็นบล็อกซึ่งมีขนาด 64กิโลไบต์ หรือ 128กิโลไบต์

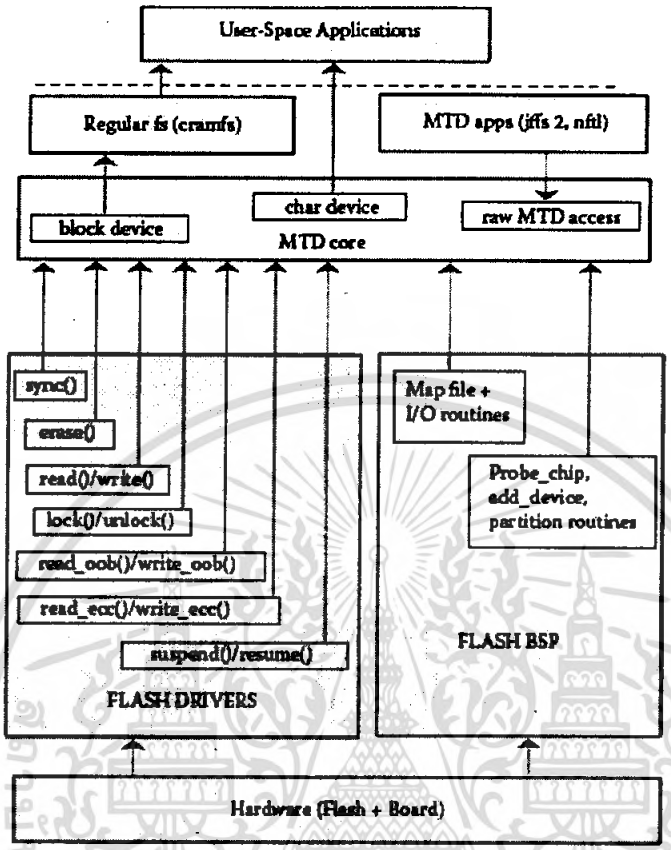
หนึ่งในหลายแฟลชไฟล์ซิสเต็มที่นิยมในวันนี้นั้นจะใช้ เจเอ็ฟเอ็ฟเอสทู หรือ Journaling Flash File System2 ซึ่งมีลักษณะสำคัญหลายอย่างที่ได้มุ่งไปที่การปรับปรุงประสิทธิภาพทั้งหมดเพิ่มอายุการใช้งานและ ลดทอนความเสี่ยงที่ข้อมูลจะหายเมื่อเกิดไฟดับ

2.3.13 สถาปัตยกรรม เอ็มทีดี

เอ็มทีดี เป็นระบบย่อย ที่ใช้จัดการอุปกรณ์สตอเรจบนบอร์ด สถาปัตยกรรม เอ็มทีดีถูกแบ่งออกเป็นส่วนดังต่อไปนี้

- เอ็มทีดี คอล (MTD core):จะทำหน้าที่จัดการการเชื่อมต่อระหว่างไดร์เวอร์แแฟลชแบบระดับ (Low-level flash driver) ต่ำ กับ แอปพลิเคชั่น มันจะอิมพลีเมนท์ เป็น ชาเล็กเตอร์ และ บล็อก ดีไวซ์ โหมด
- ไดร์เวอร์แแฟลชแบบระดับต่ำ: ในส่วนนี้จะกล่าวถึงเพียงแต่ในส่วนที่เกี่ยวกับ นอร์-เบสด์ (NOR-) และ แนนด์-เบสด์ (NAND-based)แฟลชชิป
- บีเอสพี สำหรับ แฟลช (BSP for flash) : แฟลชสามารถเชื่อมต่อกับบอร์ด ยกตัวอย่างเช่น นอร์แฟลช สามารถถูกเชื่อมต่อโดยตรงกับบัสโพรเซสเซอร์หรืออาจจะเชื่อมต่อกับ พีซีไอบัสชั้นบีเอสพีจะทำให้แฟลชไดร์เวอร์ทำงาน กับบอร์ดหรือโพรเซสเซอร์ใดๆได้
- เอ็มทีดีแอปพลิเคชั่น: สามารถให้ โมดูลย่อย อาทิเช่น เจเอ็ฟเอ็ฟเอส2 , เอ็นเอ็ฟทีแอล (NFTL), แอปพลิเคชั่นส่วนยูซเซอร์สเปซ จัดการใดๆอาทิเช่น อัพเกรด

รูปแสดงส่วนประกอบต่างของเอ็มทีดีปฏิสัมพันธ์กัน และส่วนของ เคอร์เนล



รูปที่ 2.23 แสดงสถาปัตยกรรมเอ็มทีดี

2.3.14 หน่วยจัดการอินเทอร์รับ

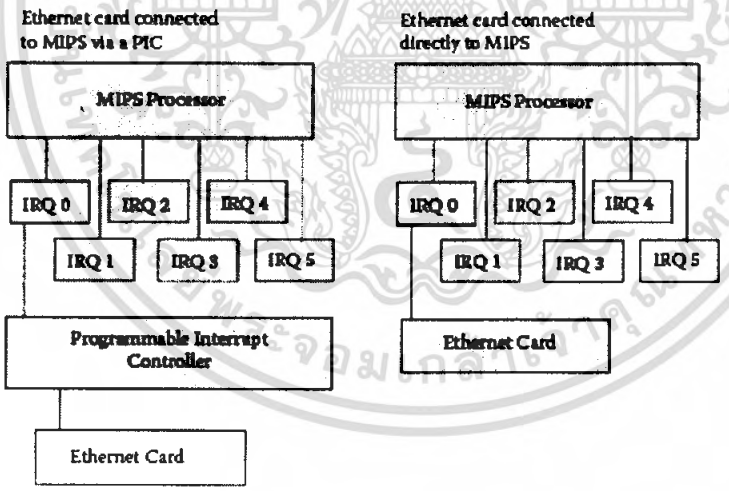
ทุกๆบอร์ดจะมีหน่วยจัดการอินเทอร์รับโดยมากแล้วจะเป็น พีไอซี (PIC : Programmable Interrupt Controller) ในหัวข้อนี้จะกล่าวถึงขั้นตอนอย่างละเอียดนำไปสู่การเขียนโปรแกรมอินเทอร์รับคอนโทรเลอร์ในลินุกซ์ ก่อนที่จะไปถึงการรายละเอียดการเขียนโปรแกรม มาทำความเข้าใจฟังก์ชันการทำงานพื้นฐานของ พีไอซี ก่อน

- ไมโครโปรเซสเซอร์ โดยปกติจะมีจำนวนอินเทอร์รับที่จำกัด และอาจจะไม่พอถ้ามีอุปกรณ์บนบอร์ด และทั้งหมดต้องการการอินเทอร์รับ อินเทอร์รับคอนโทรเลอร์จึงเข้ามาช่วยเหลือในการแก้ปัญหานี้ โดยอนุญาตให้อินเทอร์รับทั้งหลายถูกมัลติเพลคซ์ บนเส้นทางอินเทอร์รับเดียว ทำให้ขยายความสามารถของโพรเซสเซอร์
- พีไอซี จัดการลำดับความสำคัญของอินเทอร์รับทำให้เป็นประโยชน์ในรายของกรณีที่โพรเซสเซอร์ ไม่สนับสนุนลำดับความสำคัญของการอินเทอร์รับเมื่อโพรเซสเซอร์

กำลังจัดการทำงานกับอินเทอร์รับที่มีลำดับความสำคัญสูง พีไอซีจะไม่ส่งมอบอินเทอร์รับที่มีลำดับความสำคัญต่ำให้แก่โพรเซสเซอร์ดังนั้นเมื่อมีสองอุปกรณ์เกิดการอินเทอร์รับ พีไอซีจะดูลำดับความสำคัญและส่งอินเทอร์รับที่มีลำดับความสำคัญสูงแก่ซีพียู

- ลักษณะการทริกเกอร์ ก็คือฮาร์ดแวร์จะตรวจรับการอินเทอร์รับสองวิธีคือ เลเวล (level) และ เอดจ์ ทริกเกอร์ (edge trigger) พีไอซีสามารถแปลงจากเอดจ์ ไปเป็นเลเวลอินเทอร์รับ โดยใช้การแลชอินพุตอินเทอร์รับ 8259A เป็นตัวอย่างของพีไอซี ไอซีตัวนี้เป็นที่นิยม และเป็นฮาร์ดแวร์ที่เก่ง และได้ถูกนำไปใช้ในบอร์ดเอ็มเบดเดด กันอย่างแพร่หลาย แสดงคุณลักษณะพื้นฐานของ 8259A controller ดังต่อไปนี้

ลินุกซ์เคอร์เนล จะถือปฏิบัติว่าอินเทอร์รับ ทั้งหมดเป็นลอจิคอลอินเทอร์รับ ลอจิคอลอินเทอร์รับที่เชื่อมโดยตรงกับโพรเซสเซอร์ หรือผ่านทางพีไอซีในกรณีทั้งสองเมื่ออินเทอรัพต์ถูกรีจิสเตอร์จะต้องทำผ่านฟังก์ชัน request_irq(), ดีไวซ์ไดร์เวอร์ จะต้องส่งหมายเลขอินเตอร์รับ แก่ฟังก์ชัน จำนวนลอจิคอลอินเทอร์รับขึ้นอยู่กับโพรเซสเซอร์ แต่ละตัว การแม็ประหว่าง ลอจิคอลอินเทอร์รับ กับ ฟิซิคอลอินเทอร์รับ เป็นหน้าที่ความรับผิดชอบของบีเอสพี



รูปที่ 2.24 แสดงการเชื่อมต่ออินเตอร์รัพท์

แกนหลักของ บีเอสพีอินเตอร์รับ มีสองโครงสร้างข้อมูลคือ

- Interrupt controller descriptor hw_interrupt_type : โครงสร้างนี้จะถูกประกาศไว้ใน include/linux/irq.h ทุกๆ ฮาร์ดแวร์อินเตอร์รับต้องทำการอิมพลีเมนท์โครงสร้างนี้ ฟิวที่สำคัญในโครงสร้างข้อมูลมีดังต่อไปนี้

  - Start-up: เป็น พอยท์เตอร์ ไปที่ฟังก์ชันที่ใช้ในการร้องขอเมื่ออินเทอร์รับ ถูกโปรพ (probe)หรือถูกเรียก ใช้ฟังก์ชัน request_irq()
  - Shutdown:เป็น พอยท์เตอร์ ไปที่ฟังก์ชันที่ใช้ในการร้องขอเมื่ออินเทอร์รับถูกปลดปล่อย ใช้ฟังก์ชัน free_irq()
  - Enable: เป็น พอยท์เตอร์ ไปที่ฟังก์ชันที่ทำการเปิดอินเทอร์รับ
  - Disable:เป็น พอยท์เตอร์ ไปที่ฟังก์ชันที่ใช้ในการปิดอินเทอร์รับ
  - Ack:เป็น พอยท์เตอร์ไปที่ฟังก์ชัน controller-specific ที่ใช้ในการ ตอบรับอินเทอร์รับ
  - End:เป็น พอยท์เตอร์ ไปที่ฟังก์ชันที่จะเรียกขึ้นมา เมื่ออินเทอร์รับถูกบริการเสร็จสิ้น

- IRQ descriptor irq_desc_t : ถูกประกาศไว้ใน include/linux/irq.h อีกเช่นกัน ทุกๆลอจิคอลอินเทอร์รับ จะต้องใช้โครงสร้างนี้ ฟิวที่สำคัญในโครงสร้างข้อมูลมีดังต่อไปนี้
  - Status : สถานะของแหล่งกำเนิดอินเทอร์รับ
  - Handler : เป็นพอยท์เตอร์ ไปที่อินเทอร์รับคอนโทรเลสคริปเตอร์
  - Action : รายการการทำงานของไออาร์คิว (IRQ action list)

บทที่ 3
แนวทาง และการพัฒนาแพลตฟอร์ม

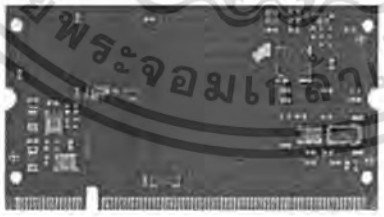
3.1 การพัฒนา ฮาร์ดแวร์ ของ แพลตฟอร์ม

มีขั้นตอนดังต่อไปนี้

- วิเคราะห์หา ฮาร์ดแวร์ ที่จำเป็นในการพัฒนาหุ่นยนต์ใน แพลตฟอร์ม
- ศึกษาการทำงานของ ฮาร์ดแวร์ ส่วนต่างๆ แล้วออกแบบแผนผังวงจร
- ออกแบบ พีซีบี และจัดทำ
- จัดหาอุปกรณ์ อิเล็กทรอนิกส์ และ ส่วนประกอบ ต่างๆ แล้วทำการติดตั้งอุปกรณ์ลงใน พีซีบี
- ทำการทดสอบและค้นหาข้อผิดพลาด

3.1.1 การศึกษาการทำงานของ ฮาร์ดแวร์ และการออกแบบแผนผังวงจร

IntelPXA270



รูปที่ 3.1 อินเทลพีเอ็กซ์เอเอ็กซ์สเกล 270 (IntelPXA270)

ความเร็ว CPU:

312 / 520 MHz

ขนาดหน่วยความจำ:

64 เมกะไบต์ เอสดีแรม (32 บิท)

32 เมกะไบต์ แฟลช (32 บิท)

สามารถ Interface กับ:

32 บิท โพเซสเซอร์ บัส

คอมแพล็คแฟลช / พีซีเอ็มพีไอเอ

แอลซีดี (เอสวีจีเอ)

หน้าจอแบบสัมผัส

ออร์ดิโอ อินพุท/เอาต์พุท (16 บิท สเตอริโอ)

ซีมอส/ซีซีดี

เอ็มเอสแอล (อัตราสูงสุด 416 เมกะบิทต่อวินาที)

ไอสแควร์ซี

เอสพีไอ

เอสดีการ์ด

เมมโมรี สติ๊ก

85 จีพีไอโอ

ยูเอสบี โฮส / ดีไวซ์

100 MBit อีเทอร์เน็ต

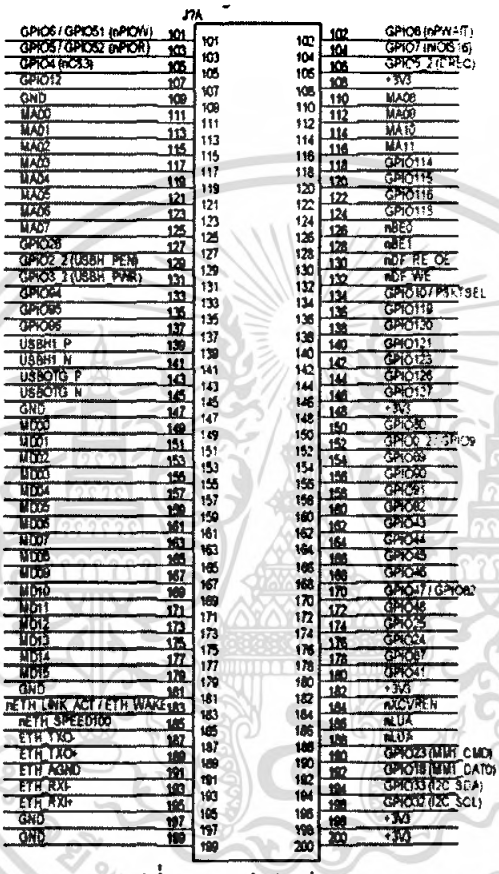
ระบบปฏิบัติการที่สนับสนุน:

วินโดวส์ ซีอี

ลินุกซ์ฝังตัว เคอร์เนล 2.4 และ 2.6

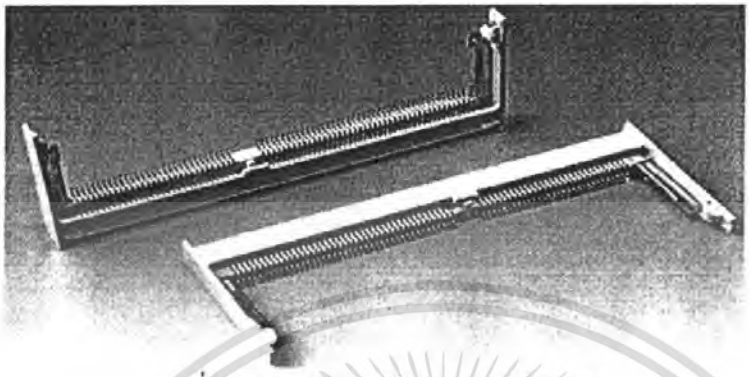
คิวเอ็นเอ็กซ์ (QNX)

รายละเอียดขา (200 ขา) :



รูปที่ 3.2 ขาพีเอ็กซ์เอ 270

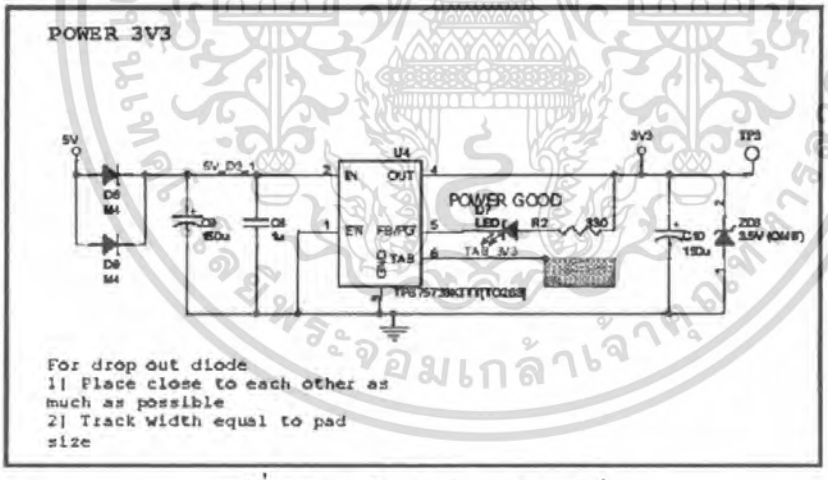
คอนเน็กเตอร์ (SO-DIMM200):



รูปที่ 3.3 คอนเน็กเตอร์โซดิมซอกเก็ต 200

แหล่งจ่ายไฟ 3.3โวลต์

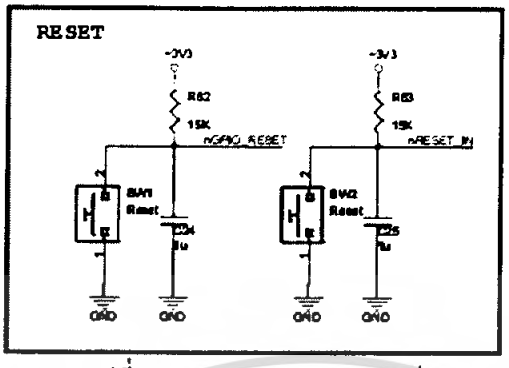
ใช้ ไอซี เรกูเลเตอร์ TPS75733KTTT ซึ่งรับ อินพุต เป็น 5โวลต์ แล้วแปลงออกมาเป็น 3.3โวลต์ ได้แผนผังวงจร ดังนี้



รูปที่ 3.4 แผนผังวงจรของเรกูเลเตอร์

สวิตซ์ รีเซ็ต

ใช้เชื่อมต่อเข้ากับ ขา NRESET_IN ของพีเอ็กซ์เอ 270 โดยปกติแล้วจะต้องต่อพลูอัพไว้ที่ขานี้เพื่อไม่ให้ซีพียู ถูก รีเซ็ต และเมื่อกด สวิตซ์ (NRESET_IN = 0) จะทำให้เกิดการรีเซ็ตตัวซีพียู จึงออกแบบวงจรแผนผังวงจร ได้ดังนี้



รูปที่ 3.5 แผนผังวงจรของสวิตช์

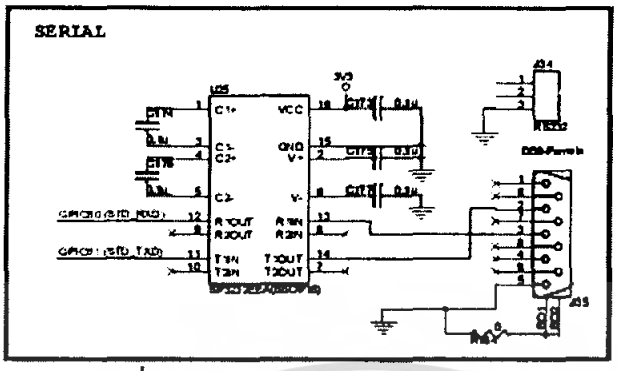
พอร์ตอนุกรม

เป็นวงจรที่ใช้ในการขยายหรือปรับสัญญาณข้อมูลให้สามารถส่งไปตามสายสัญญาณ ได้ตาม ยูอาร์ท (UART) พื้นฐาน ในที่นี้เลือกใช้ ไอซี SP3232EEA ที่ใช้ไฟ 3.3 โวลต์ ซึ่งจะนำมาแปลงสัญญาณระหว่าง PXA270 และ คอนเน็กเตอร์ DB9 ที่ ผู้ใช้งาน จะนำไปเชื่อมต่อกับ คอมพิวเตอร์ เพื่อทำการ คอนฟิค หรือ ตรวจสอบข้อผิดพลาด ซึ่งมี ขา ของ PXA270 ที่จะต้องเชื่อมต่ออยู่ 2 ขา คือ

ตารางที่ 3.1 ขาอนุกรม

ชื่อขา	คำอธิบาย	ชนิด อินพุท/ เอาต์พุท	มัลติเพล็กซ์ (Multiplexed) กับ
STD_RXD	ใช้รับสัญญาณข้อมูลตาม ยูอาร์ทพื้นฐาน (Receive Pin for Standard UART and Slow Infrared Functions)	อินพุท	GPIO46
STD_TXD	ใช้ส่งสัญญาณข้อมูลตาม ยูอาร์ทพื้นฐาน (Transmit Pin for Standard UART and Slow Infrared Functions)	เอาต์พุท	GPIO47

ออกแบบ แผนผังวงจร ได้ดังนี้



รูปที่ 3.6 แผนผังวงจรของพอร์ตอนุกรม

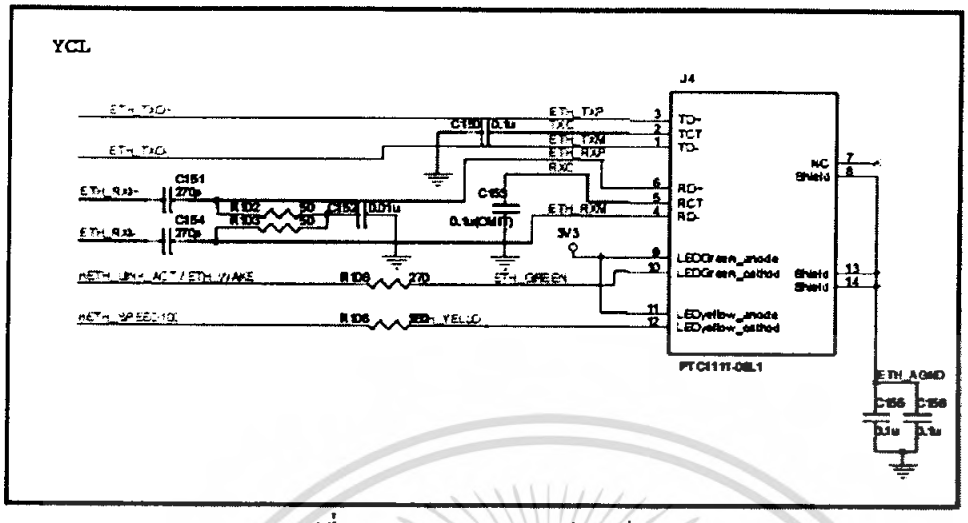
อีเทอร์เน็ท

ในการเชื่อมต่อกับ คอนเน็กเตอร์ อีเทอร์เน็ท นั้นใช้การเชื่อมต่อกับ GPIO ที่สำคัญทั้งหมด 6 GPIO ดังนี้

ตารางที่ 3.2 ขาอีเทอร์เน็ท

ชื่อขา	คำอธิบาย	ชนิด อินพุท/ เอาต์พุท	มัลติเพล็กซ์ (Multiplexed) กับ
nETH_LINK_AKT	เป็นสายสัญญาณที่ใช้ check สถานะการทำงาน ของ อีเทอร์เน็ท(Ethernet Activity Indicator)	เอาต์พุท	-
nETH_SPEED100	เป็นสายสัญญาณที่ใช้ check ความเร็วในการ รับส่งของ ethernet ว่าเป็น 10 หรือ 100 Mbp s(Ethernet Speed Indicator)	เอาต์พุท	-
ETH_TXO-	เป็นสายสัญญาณที่ในการส่งข้อมูลโดยจะใช้คู่ กับ ETH_TXO+ เพื่อหาข้อมูล ที่ถูกต้องและ ลด ข้อผิดพลาด จากการส่ง (Ethernet TX Differential Output (minus))	เอาต์พุท	-
ETH_TXO+	เป็นสายสัญญาณที่ในการส่งข้อมูลโดยจะใช้คู่ กับ ETH_TXO- เพื่อหา ข้อมูล ที่ถูกต้องและ ลด ข้อผิดพลาด จากการส่ง (Ethernet TX Differential Output (plus))	เอาต์พุท	-
ETH_RXI-	เป็นสายสัญญาณที่ในการรับข้อมูลโดยจะใช้คู่ กับ ETH_RXI+ เปรียบเทียบกัน เพื่อหา ข้อมูล ที่ถูกต้องและลด ข้อผิดพลาด จากการส่ง (RX Differential Input (minus))	อินพุท	-
ETH_RXI+	เป็นสายสัญญาณที่ในการรับข้อมูลโดยจะใช้คู่ กับ ETH_RXI- เปรียบเทียบกัน เพื่อหา ข้อมูล ที่ถูกต้องและลด ข้อผิดพลาด จากการส่ง (RX Differential Input (plus))	อินพุท	-

แต่ คอนเน็กเตอร์ อีเทอร์เน็ท นั้นมีทั้งหมด 8 สายสัญญาณ(ไม่รวมไฟแสดงสถานะ) โดยที่เหลืออีก 2 เส้น เป็น กราวด์ และ ไม่มีการเชื่อมต่อ จึงสามารถออกแบบแผนผังวงจร ได้ดังนี้



รูปที่ 3.7 แผนผังวงจรของอีเทอร์เน็ท

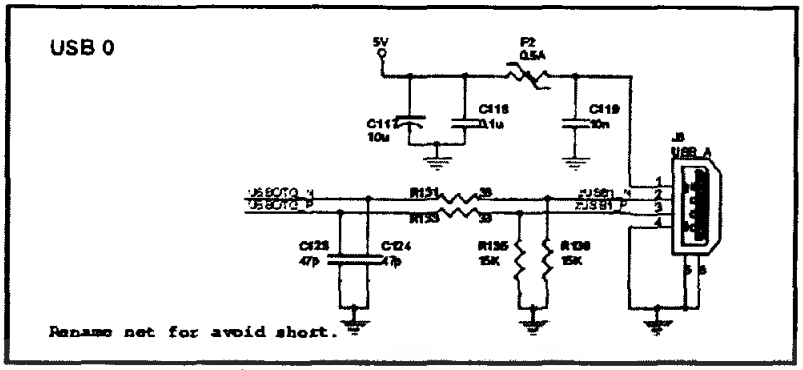
ยูเอสบี โฮส

คอนเน็กเตอร์ยูเอสบี มีสายสัญญาณที่สำคัญทั้งหมด 4 เส้นคือ วีซีซี 5 โวลต์ , กราวด์ , D+ , D- โดยจะต้องเชื่อมต่อกับ ขา SODIMM ซอคเก็ต ของ PXA 270 2 ขาคือ

ตารางที่ 3.3 ขายูเอสบีโฮสต์

ชื่อขา	คำอธิบาย	ชนิด	มัลติเพล็กซ์กับ
USBH_P	เป็น ขา ที่เป็นสัญญาณข้อมูลแบบ + ใช้เปรียบเทียบกับUSBH_N เพื่อหาข้อมูลที่แท้จริง (USB Host Positive Line: Differential signal connects to the USB host interface. (D+))	เอาต์พุท/อินพุท	-
USBH_N	เป็น ขา ที่เป็นสัญญาณข้อมูลแบบ - ใช้เปรียบเทียบกับUSBH_P เพื่อหาข้อมูลที่แท้จริง(USB Host Negative Line: Differential signal connects to the USB host interface. (D-))	เอาต์พุท/อินพุท	-

โดยสามารถออกแผนผังวงจร ได้ดังนี้



รูปที่ 3.8 แผนผังวงจรของยูเอสบีโฮสต์

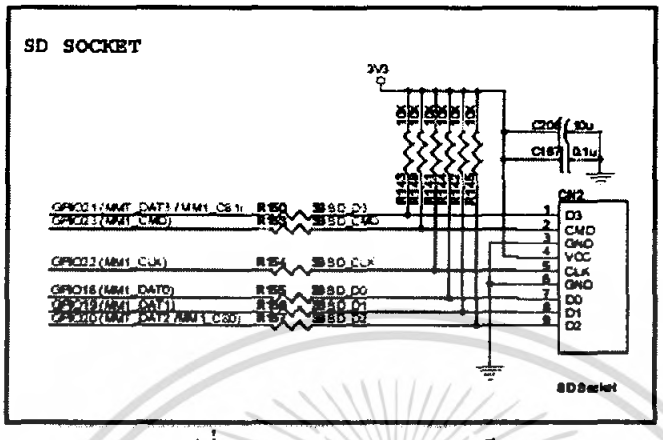
มัลติมีเดียการ์ด (MMC) / ซีเคียวดิจิตอลการ์ด (SD)

เป็นส่วนที่ใช้ในการเก็บข้อมูลของ แพลตฟอร์ม หรือ ดึงข้อมูลจาก การ์ด เข้ามาใช้ใน แพลตฟอร์ม โดย มี ขา ที่จะต้องเชื่อมต่อกับ เอสดี/เอ็มเอ็มซี คอนเน็กเตอร์ ทั้งหมด 6 ขาดังนี้

ตารางที่ 3.4 ขาเอสดี/เอ็มเอ็มซี

ชื่อขา	คำอธิบาย	ชนิด	มัลติเพล็กซ์กับ
MMCLK	เป็น ขา สัญญาณนาฬิกาที่ใช้กับ เอสดี/เอ็มเอ็มซี (MultiMediaCard and SD/SDIO Card Bus Clock)	เอาต์พุท	GPIO32
MMCMD	เป็น ขาอินพุทเอาต์พุท ที่ใช้ในการรับส่งคำสั่ง และการตอบสนองต่อ โทเค็น ของ เอสดี/เอ็มเอ็มซี (MultiMediaCard Command: MMC and SD/SDIO: Bidirectional line for command and response tokens.) เป็น เอาท์พุท ที่ใช้ในการส่งคำสั่งและเขียนข้อมูลของ เอสพีไอ (SPI: Output for command and write data.)	เอาต์พุท/ อินพุท	GPIO112
MMDAT 0	เป็น ขา ที่ใช้ในการอ่านและเขียนข้อมูลของ เอสดี/เอ็มเอ็มซี (MultiMediaCard Data 0: MMC and SD/SDIO: Bidirectional line for read and write data.) เป็น ขา ที่ใช้ในการตอบสนองต่อ โทเค็น ของ เอสพีไอ (SPI: Input for response token and read data.)	เอาต์พุท/ อินพุท	GPIO92
MMDAT 1	เป็นขาที่ใช้ในการอ่านและเขียนข้อมูลของ เอสดี/เอ็มเอ็มซี (MultiMediaCard Data 1: SD/SDIO: Bidirectional line for read and write data. Used only for SD 4-bit data transfers and to signal SDIO interrupts to the controller.) ใช้เป็น ขาอินเทอรัพท์ไปที่ คอนโทรเลอร์ ของ เอสพีไอ (SPI: Used only to signal SDIO interrupts to the controller.)	เอาต์พุท/ อินพุท	GPIO109
MMDAT 2/ MMCS0	เป็น ขา ที่ใช้ในการอ่านและเขียนข้อมูลของ เอสดี/เอ็มเอ็มซี (MMC Chip Select 0: SD/SDIO: Bidirectional line for read and write data. Used only for SD 4-bit data transfers.) ใช้เป็นขา ชิพ ซีเลค 0 ของ เอสพีไอ(SPI: Chip select 0)	เอาต์พุท/ อินพุท	GPIO110

จึงสามารถออกแบบวงจร แผนผังวงจร ได้ดังนี้



รูปที่ 3.9 แผนผังวงจรของเอสดี

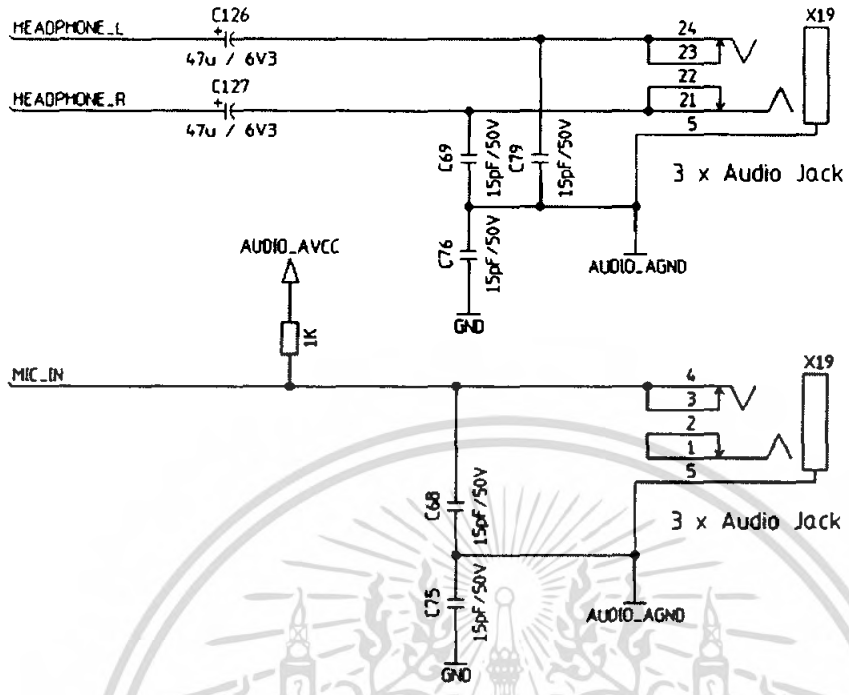
ออดิโอ

ใช้รับและส่งข้อมูล อะนาล็อก ที่เป็นเสียงโดยในพีเอ็กซ์เอ 270 มีลักษณะการเชื่อมต่อกับ ไมโครโฟน คอนเน็กเตอร์ โดยใช้ 2 ขา คือ MIC_IN และ เชื่อมต่อกับ เฮดโฟน หรือ สปีคเกอร์ คอนเน็กเตอร์ โดยใช้ 3 ขา ดังนี้

ตารางที่ 3.5 ขาออดิโอ

ชื่อขา	คำอธิบาย	ชนิด	มัลติเพล็กซ์กับ
MIC_IN	เป็น ขา ที่ใช้รับสัญญาณ อะนาล็อก ที่เป็นข้อมูลเสียงจาก ไมโครโฟน (Microphone Input)	อะนาล็อกอินพุท	-
MIC_GND	เป็น อะนาล็อก กราวด์ ของ ไมโครโฟน (Microphone Ground)	พาเวอร์	-
HEADPHONE_GND	เป็น อะนาล็อก กราวด์ ของ เฮดโฟน หรือ สปีคเกอร์ (Headphone Ground)	พาเวอร์	-
HEADPHONE_L	เป็นขา ที่ใช้ส่งสัญญาณ อะนาล็อก ที่เป็นข้อมูลเสียงไปที่ เฮดโฟน หรือ สปีคเกอร์ ด้านซ้าย (Headphone Output (Left Channel))	อะนาล็อกเอาต์พุท	-
HEAPHONE_R	เป็น ขา ที่ใช้ส่งสัญญาณ อะนาล็อก ที่เป็นข้อมูลเสียงไปที่ เฮดโฟน หรือ สปีคเกอร์ ด้านขวา (Headphone Output (Right Channel))	อะนาล็อกเอาต์พุท	-

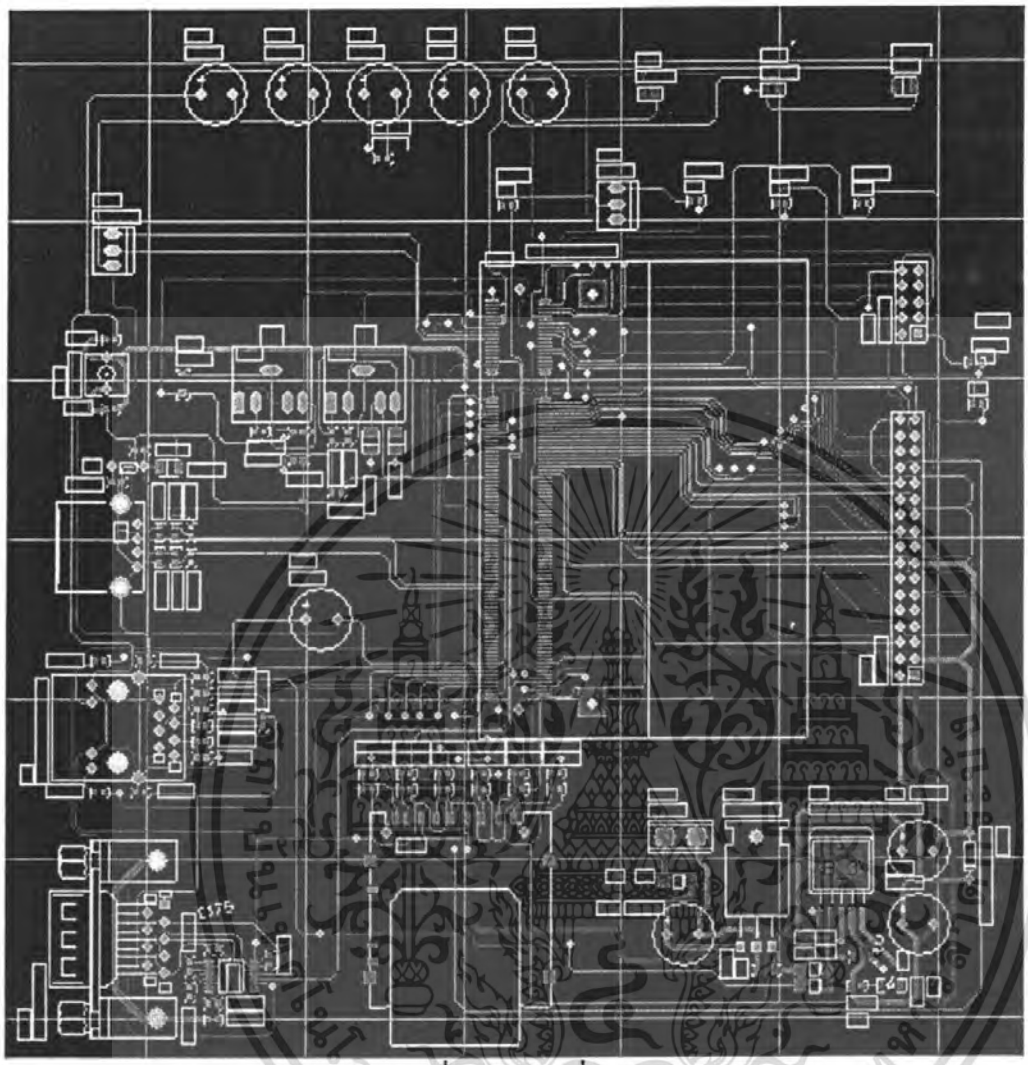
จึงได้ แผนผังวงจร ดังนี้



รูปที่ 3.10 แผนผังวงจรของออดิโอ

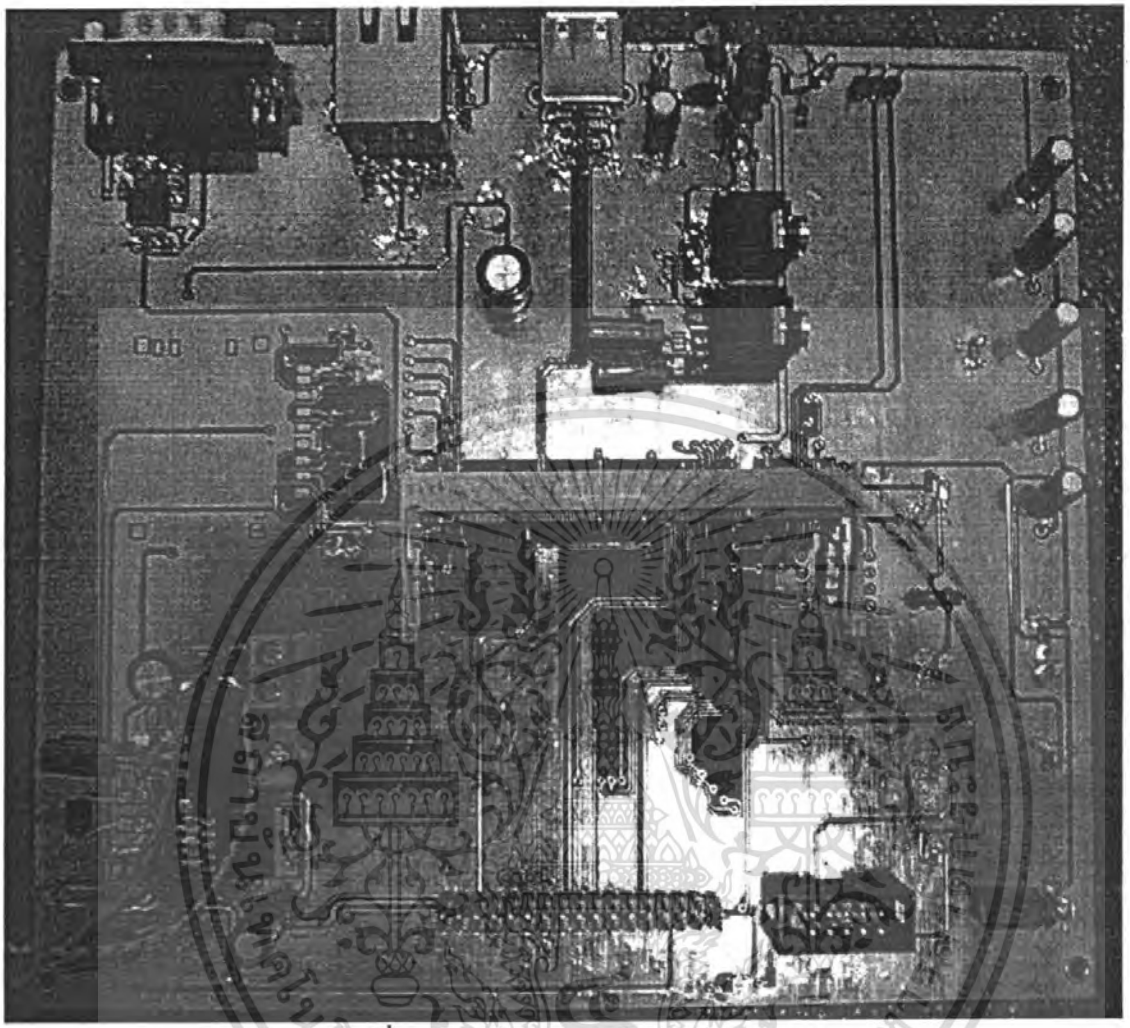
3.1.2 การออกแบบ พีซีบี

หลังจากที่ได้แผนผังวงจร ครบทุกส่วนที่ต้องการแล้วก็นำ แผนผังวงจร นั้น มาออกแบบเป็น พีซีบี เพื่อจะนำไปในเป็น แพลตฟอร์ม ต้นแบบ (โดยใช้ โปรแกรมโพเทล 99 (Potel 99)) ได้ดังนี้



รูปที่ 3.11 พีซีบี ที่ออกแบบ

เมื่อจัดทำ พีซีบี และลงอุปกรณ์แล้วจะได้ บอร์ด ขนาดประมาณ 5.8x6.5 นิ้ว ที่ใช้เป็น แพลตฟอร์ม ต้นแบบดังนี้



รูปที่ 3.12 บอร์ดแพลตฟอร์ม ต้นแบบ

3.2 การติดตั้งบอร์ด

ดีเอชซีพีเซิร์ฟเวอร์ (DHCP) : จะเป็นสิ่งที่บอก หมายเลขไอพีให้กับตัวบอร์ดใน ตอนเวลาบูท,บอกเซิร์ฟเวอร์ว่าจะบูทที่ไหน,ไฟล์ที่จะโหลดสำหรับการบูท,ตัวเซิร์ฟเวอร์ และ พาร์ท สำหรับ รูทไฟล์ ที่จะถูกเมาท์ ในภายหลัง

ทีเอฟทีพีเซิร์ฟเวอร์ (TFTP) : สนับสนุนการโอนย้ายข้อมูลโดยใช้โปรโตคอล ทีเอฟทีพี ซึ่งไม่มีการลอกอินเวลาใช้งานซึ่งจะใช้เป็นตัวโอนถ่ายไฟล์ บูทอิมเมท

เอ็นเอฟเอสเซิร์ฟเวอร์ (NFS) : เพื่อสร้างการเมาท์รูทไฟล์จากตัวบอร์ด

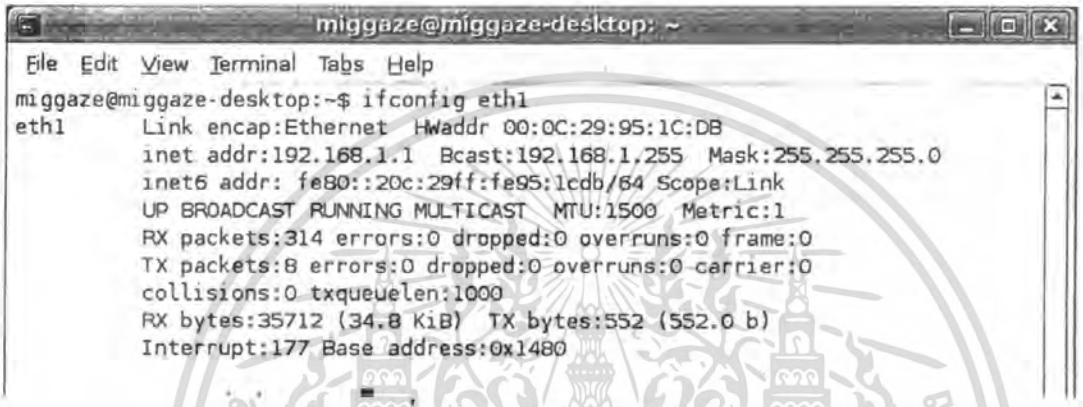
ซึ่งเซิร์ฟเวอร์เหล่านี้อาจเปลี่ยนแปลงได้หรือไม่ต้องใช่ก็ได้แต่ได้กล่าวถึงและแสดงการติดตั้ง เนื่องจากจะสะดวกในการพัฒนา

ต่อไปนี้จะกล่าวถึงขั้นตอนการติดตั้ง

เพื่อให้แน่ใจว่าเครื่อง พีซี มี eth1แล้วควรตรวจสอบก่อนว่ามีหรือไม่เนื่องจากการติดตั้งและใช้งานเซิร์ฟเวอร์ดังกล่าวจะทำบนeth1 ใช้คำสั่งนี้

$ Ifconfig eth1

ตัวอย่าง:



รูปที่ 3.13 แสดงการทดสอบ eth1

-ติดตั้ง DHCP เซิร์ฟเวอร์

ถ้า DHCP เซิร์ฟเวอร์ ยังไม่ได้ถูกติดตั้งบนเครื่อง พีซี ของคุณ คุณควรที่จะทำการติดตั้งเสียก่อน

$ apt-get install dhcpd

ในการคอนฟิกดีเอชซีพี โดยทั่วไปจะทำกันในสองส่วน

- คอมมานไลท์ ซึ่งจะบอกว่าเซิร์ฟเวอร์ว่าอินเตอร์เฟสไหนที่จะเป็นตัวทำงาน
- คอนฟิคไฟล์ซึ่งจะบรรจุข้อมูลของตัวไคล์เอ็นท์

จากนั้นเข้าไปแก้ไขไฟล์ /etc/init.d/dhcp โดยให้ตัวแปล INTERFACE เปลี่ยนเป็น "eth1" จากนั้นก็ไปแก้ไขไฟล์configของ dhcp คือ /etc/dhcpd.conf

คุณควรกำหนดค่าดังต่อไปนี้

```
subnet 192.168.1.0 netmask 255.255.255.0 {
        # --- default gateway
        # option routers 192.168.1.1;
        option subnet-mask 255.255.255.0;
        option broadcast-address 192.168.1.255;
```

```
        option domain-name "development.local";
        option domain-name-servers 192.168.1.1;
        option ip-forwarding off;
        default-lease-time 86400;
        max-lease-time 86400;
        host colibri {
                hardware ethernet 08:00:3e:26:0a:5b;
                fixed-address 192.168.1.99;
                option host-name "colibri";
                next-server 192.168.1.1;
                filename "/work/colibri/boot/image";
                option root-path "192.168.1.1:/work/colibri/rootfs";
        }
}
```

ชื่อและค่าต่างๆอาจแก้ไขได้เมื่อคุณนำไปใช้งาน

เมื่อทำการบันทึกไฟล์แล้วทำการอัพเดทไฟล์ เพื่อให้แน่ใจว่าไฟล์ที่แก้ไขไปนั้นdhcp เซิร์ฟเวอร์ จะใช้งานไฟล์ใหม่ที่ถูกแก้ไขนี้

```
$ touch /var/lib/dhcp/dhcpd.leases
```

เริ่มการทำงานของดีเอชซีพีเซิร์ฟเวอร์

```
$ /etc/init.d/dhcp start
```

-ติดตั้งทีเอฟทีพีเซิร์ฟเวอร์

ถ้าทีเอฟทีพีเซิร์ฟเวอร์ ยังไม่ได้ถูกติดตั้งบน พีซี ของคุณ ควรที่จะติดตั้งเสียก่อน

```
$ apt-get install tftpd
```

ซึ่งทีเอฟทีพีเซิร์ฟเวอร์จะทำงานผ่าน xinetd เซิร์ฟเวอร์ จึ่งจะต้องติดตั้ง xinetd เซิร์ฟเวอร์ ด้วย

```
$ apt-get install xinetd
```

จากนั้นจะต้องไปทำการคอนฟิคไฟล์ /etc/xinetd.d/tftp

ตัวอย่าง:

รูปที่ 3.14 แสดงตัวอย่างการติดตั้งTFTP

เมื่อทำการคอนฟิคแก้ไขเสร็จสิ้นแล้ว จากนั้นก็ควรที่จะเริ่มการทำงานใหม่ของตัว xinetd เซิร์ฟเวอร์

S /etc/init.d/xinetd restart

คุณควรที่จะแน่ใจว่าสิทธิ์การใช้งานการอ่านเขียนและผู้ใช้งานควรเป็น 'nobody'

-ติดตั้งเอ็นเอฟเอสเซิร์ฟเวอร์

โดยปกติแล้ว เอ็นเอฟเอส เซิร์ฟเวอร์จะมีอยู่แล้วในตัวลินุกซ์ เคอร์เนล ของเครื่อง พีซี แต่ถ้าในส่วนของยูเซอร์ สเปส ของ เอ็นเอฟเอส เซิร์ฟเวอร์ ยังไม่ได้ถูกติดตั้งคุณควรที่จะทำการติดตั้งก่อน

S apt-get install nfs-common

S apt-get install nfs-kernel-server

ไฟล์ /etc/exports จะกำหนดตัว ไฟล์ซิสเต็ม อันไหน สำหรับ ไคล์เอ็นท์ไหน ทาง เอ็นเอฟเอส คุณควรเข้าไปคอนฟิคโดยในข้อต่อไปนี้ไปในไฟล์

/work/colibri/rootfs 192.168.1.0/255.255.255.0(rw,no_root_squash)

เมื่อทำการเพิ่มเติมบรรทัดดังกล่างเสร็จสิ้นแล้วก็ควรที่จะเริ่มการทำงาน เซิร์ฟเวอร์ เสียใหม่

S /etc/init.d/nfs-common restart

S /etc/init.d/nfs-kernel-server restart

หลังจากทำการติดตั้ง เซิร์ฟเวอร์ทั้งสามแล้วเครื่อง พีซี ก็พร้อมที่จะพัฒนาแพลตฟอร์ม

การดาวน์โหลด บูทโหลดเดอร์ลงแฟลตฟอร์ม

-ดาวน์โหลด เมื่อแฟลตฟอร์มยังไม่มีลินุกซ์ทำงานอยู่ จะใช้เจแท็ก (JTAG) ในการเชื่อมต่อซึ่งเป็นวิธีที่ปลอดภัยที่สุด,และใช้งานได้อย่างแน่นอน หรือคุณสามารถใช้พอร์ตขนานในการทำงานได้

โดยจะต้องแมปขาสัญญานดังต่อไปนี้

Mapping parallel port		<->	JTAG header X20
function	pin	function	pin
Strobe	1	nTRST	1
D0	2	TDI	3
D1	3	TCK	9
D2	4	TMS	7
PaperError	12	TDO	5
Ground 18-25		Ground 2,4,6,8,10	

ซึ่งถ้าใช้ใช้การเชื่อมต่อโดยใช้ พอร์ตขนานแปลงเป็นเจแท็ก คุณสามารถใช้ jflash utility โดยคุณต้องแน่ใจว่ามีสิทธิ์ในการเข้าถึงและสั่งทำงานได้ ถ้ายังไม่มีสิทธิ์ควรแก้ไขสิทธิ์ก่อนจากนั้นก็ทำคำสั่งต่อไปนี้

```
$ cd /work/colibri-bsp-x.x/bin
$ ./jflash bulbcx u-boot.bin P 0
```

คำสั่งข้างบนจะทำการเขียนไฟล์ u-boot.bin ลงสู่ หน่วยความจำแฟลช ในตำแหน่งที่ 0 และตรวจสอบความถูกต้อง ถ้าคุณแน่ใจว่าการทำงานนั้นปราศจากข้อผิดพลาดอย่างแน่นอนก็
สามารถที่จะไม่ต้องมีการตรวจสอบข้อผิดพลาดได้โดยเปลี่ยนแปลง "P" ไปเป็น "N" ได้ หรือในกรณีที่เราต้องการจะใส่ u-boot.bin ลงไปใหม่นั้นจะให้ บูทโหลดเดอร์ เป็นตัวใหม่นั้นเราต้องทำการลบข้อมูลที่อยู่ในหน่วยความจำแฟลชนั้นเสียก่อนโดยเปลี่ยนแปลง "P" ไปเป็น "E"
ก็จะทำการลบข้อมูลในแต่ละsectionก่อนจากนั้นจึงจะสามารถโหลด u-boot.bin ลงไปใหม่
-ดาวน์โหลด เมื่อแฟลตฟอร์มมีลินุกซ์ทำงานอยู่
โดยการทำงานมีขั้นตอนที่ง่ายโดยคุณต้องทำการดาวน์โหลด บูทโหลดเดอร์ ตัวใหม่มาเก็บไว้ในตัวบอร์ดเสียก่อนโดยใช้การทีเอฟทีพีหรือจาก พอร์ตอนุกรม หรือในกรณีที่เราทำการเมาท์ รูทไฟล์ซิสเต็มดังตัวอย่างก็สามารถที่จะก๊อบปี้ไฟล์บูทโหลดเดอร์(u-boot.bin)มาเก็บไว้ใน /work/colibri/root/data
จากนั้นก็ใช้คำสั่ง dd ทำการโปรแกรมไฟล์ลงสู่flashดังตัวอย่างคำสั่งต่อไปนี้

```
$ dd if =/data/u-boot.bin of=/dev/mtdblock0 conv=notrunc
```

คำสั่งที่ใช้ในการการโหลด รูทไฟล์ซิสเต็ม ลงสู่ แฟลช(mtdblock2)

```
$ dd if =/data/root.jffs2 of=/dev/mtdblock2 bs=256k
```

และจะต้องไปแก้ ตัวแปล ของยูบูทให้เป็น "root=/dev/mtdblock2 rootfstype=jffs2 ip=:::::eth0: console=ttyS0,9600n8" ซึ่งใช้คำสั่งต่อไปนี้

u-boot> setenv bootargs "root=/dev/mtdblock2 rootfstype=jffs2 ip=:::::eth0: console=ttyS0,9600n8"

เมื่อเซ็ตอัพมาถึง ณ ตรงนี้บอร์ดก็จะเริ่มรันเข้าสู่ U-boot และ ทำการ ดาวน์โหลดลินุกซ์ เคอร์เนล (uImage)โดยผ่านทาง TFTP ในกรณีที่ไม่เจอลินุกซ์ เคอร์เนลในหน่วยความจำแฟลช และ เมื่อโหลดเสร็จเรียบร้อยแล้ว ก็จะทำการคลายออกแล้วโหลดไว้ในส่วนของเมมโมรีเพื่อทำการบูทเคอร์เนลต่อไป หลังจากนั้นก็จะทำการเมาท์รูทไฟล์ซิสเต็ม จากโฮส

และจากกรณีที่ ลินุกซ์ เคอร์เนล(uImage) ยังไม่มีอยู่ในหน่วยความจำแฟลช เราสามารทำการดาว์โหลดการใช้ทีเอฟทีพีและบันทึกเก็บไว้ในบอร์ดโดยคำสั่งต่อไปนี้

u-boot>tftp 0xa1000000 work/colibri/boot/uImage

u-boot>cp 0xa1000000 0x00080000 0x11227e

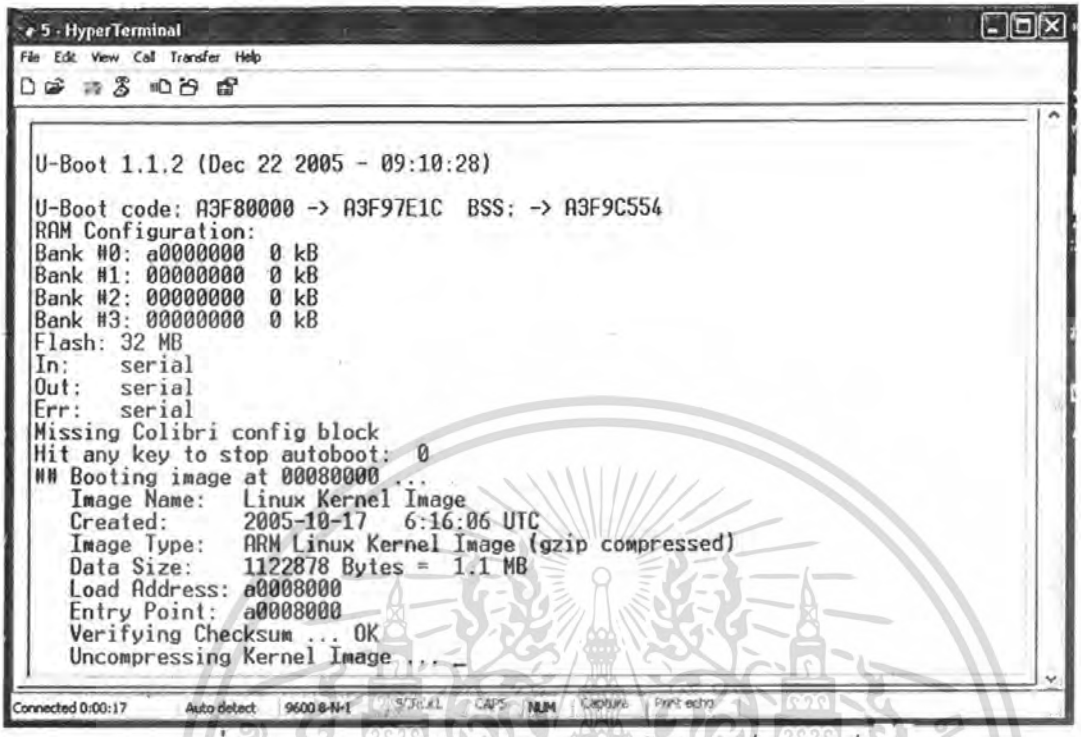
u-boot> bootm 0x80000

ตำแหน่งของ พาร์ทิชั่น ภายในหน่วยความจำแฟลช บนบอร์ด

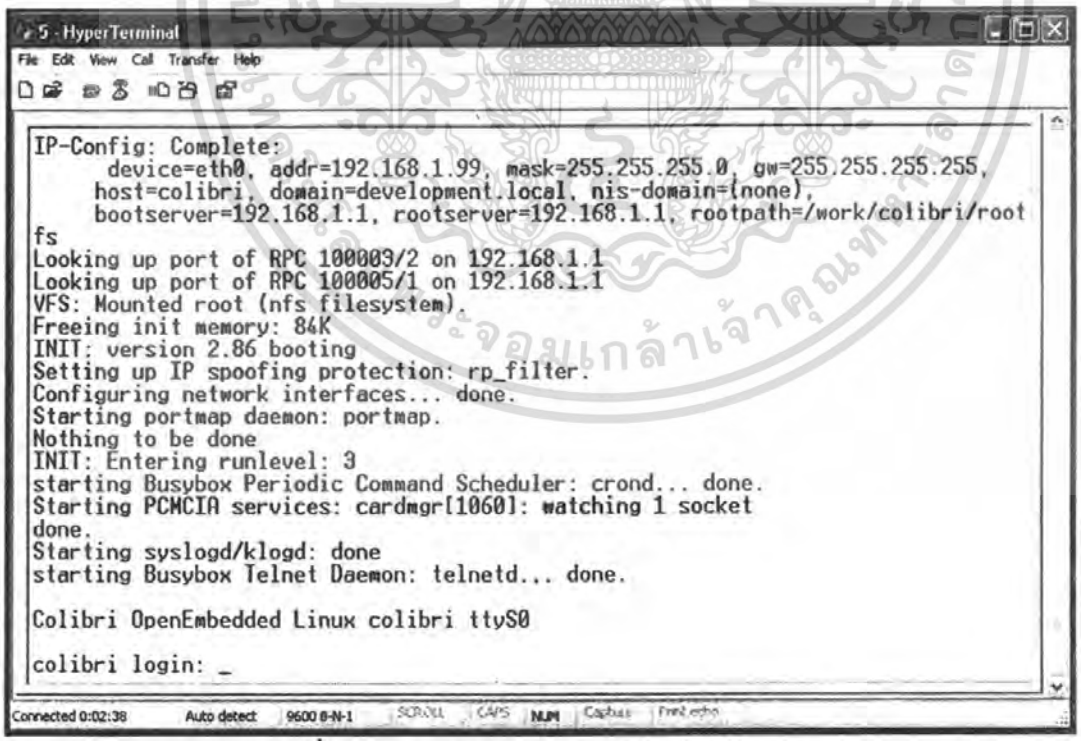
0x00000000 - 0x00080000 : "Bootloader"

0x00080000 - 0x00480000 : "kernel"

0x00480000 - 0x02000000 : "Filesystem"



รูปที่ 3.15 แสดงภาพลำดับการทำงานของบูทโหลดเดอร์ และเคอร์เนล

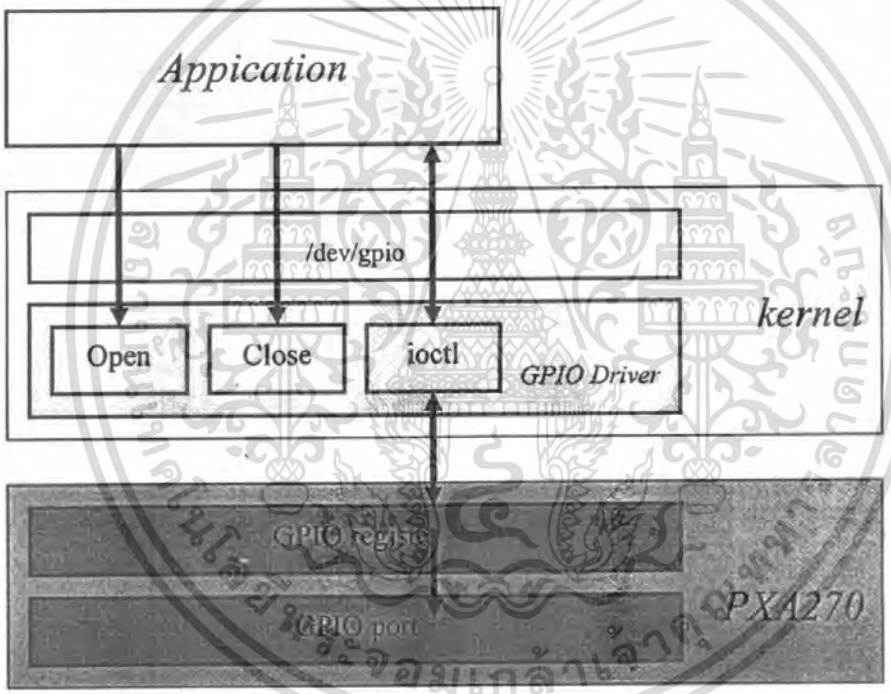


รูปที่ 3.16 แสดงภาพลำดับการเมาท์ รูทไฟล์ซิสเต็ม

3.3 การพัฒนาไดร์เวอร์

3.3.1 จีพีไอโอไดร์เวอร์

แสดงภาพไดอะแกรมการออกแบบของจีพีไอโอ ไดร์เวอร์ซึ่งจะให้แอปพลิเคชั่นกระทำการกับจีพีไอโออาทิเช่นการสั่งเอาต์พุท 0 ,เอาต์พุท 1 หรือ อ่านสถานะสัญญาณของจีพีไอโอ โดยผ่าน ioctl ซึ่งเป็นตัวเชื่อมต่อกับตัวดีไวน์ ไดร์เวอร์



รูปที่ 3.17 แสดงจีพีไอโอไดร์เวอร์

จีพีไอโอรีจิสเตอร์ (GPIO registers)

- GPLR == GPIO Pin Level Register
- GPDR == GPIO Pin Direction Register
- GPSR == GPIO Pin Output Set Register
- GPCR == GPIO Pin Output Clear Register

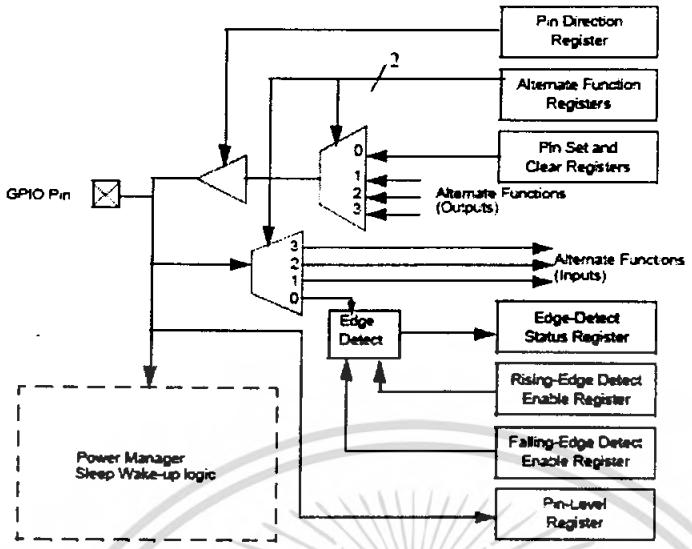
- GRER == GPIO Rising-Edge Detect Register
- GFER == GPIO Falling-Edge Detect Register
- GEDR == GPIO Edge Detect Status Register
- GAFR == GPIO Alternate Function Register

มาโคร ที่ใช้ในการ กำหนดทิศทางและสถานะของสัญญาน ให้มีค่าสัญญาณเป็น ลิอจิก '0' หรือ '1' หรืออ่านสถานะของพอร์ตจากพอร์ต

```
// GPIO is input port
GPDR(gpio) &= ~GPIO_bit(gpio);
// GPIO is output port
GPDR(gpio) |= GPIO_bit(gpio);
// Status of GPIO is high('1')
GPSR(gpio) |= GPIO_bit(gpio);
// Status of GPIO is low ('0')
GPCR(gpio) |= GPIO_bit(gpio);
// Check status of GPIO  : if true status of GPIO is high else status of GPIO is low
GPLR(gpio) & GPIO_bit(gpio);
```

ติดตั้ง จีพีไอโอ ไดร์เวอร์

```
mknod /dev/gpio c 53 0
insmod gpio.ko
```



รูปที่ 3.18 แสดงแผนภาพของจีพีไอโอ

การเขียนแอปพลิเคชั่นเพื่อใช้งาน จีพีไอโอ ไดร์เวอร์

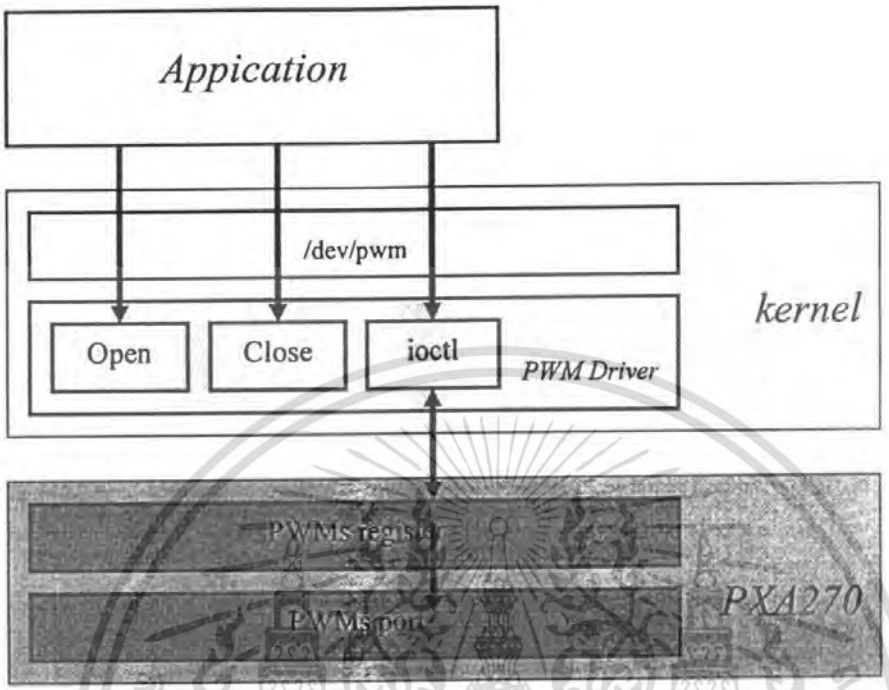
จากตัวอย่างโค้ดแอปพลิเคชั่น ด้านล่างเป็นการใช้งานจีพีไอโอไดร์เวอร์ โดยทำการเปิดโหนดจากนั้นสั่งสัญญานเอาต์พุทเป็น 1 ที่ จีพีไอโอ หมายเลข 16

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <netinet/in/h>
#include < fcntl.h>

#define GPIO_WRITE_HIGH _IOW('k', 0, unsigned int)
#define GPIO_WRITE_LOW _IOW('k', 1, unsigned int)
#define GPIO_READ       _IOR('k', 2, unsigned int)
int my_file;

int main (int argc, char *argv[])
```

```
{
        if((my_file=open("/dev/gpio",0_RDWR)) < 0){
                exit(1);
        }
        Unsigned int port = 16;
        ioctl(my_file, GPIO_WRTE_HIGH, &port);
        close(my_file);
        return 0;
}
```

3.3.2 พีดับเบิลยูเอ็ม ไดร์เวอร์

แสดงภาพไดอะแกรมการออกแบบของพีดับเบิลยูเอ็ม ไดร์เวอร์มีลักษณะเช่นเดียวกับGPIO ไดร์เวอร์ ซึ่งจะให้ แอปพลิเคชั่น กระทำการกำหนดค่าที่ รีจิสเตอร์ ของ พีดับเบิลยูเอ็ม ซึ่งจะใช้ในการกำหนดค่า ความถี่ ดิวตี้ไซเคิล โดยผ่าน ioctl ซึ่งเป็นตัวเชื่อมต่อกับตัวดีไวซ์ ไดร์เวอร์ ซึ่งพีดับเบิลยูเอ็ม ไดร์เวอร์ เวอร์ชันนี้สนับสนุน พีดับเบิลยูเอ็ม 2 พอร์ต คือ PWM0 และ PWM1 และช่วงความถี่ที่มีดิวตี้ไซเคิลเป็น 50%จะอยู่ระหว่างจาก 49.6 กิโลเฮิร์ต จนถึง 1.625เมกะเฮิร์ต



รูปที่ 3.19 แสดงพีดับเบิลยูเอ็มไดร์เวอร์

การคำนาณดิวตี้ไซเคิลของพีดับเบิลยูเอ็ม

Duty cycle time = 76.9nS x (PWMCRx[PRESCALE] + 1) x PWMDCRx[DCYCLE]

PRESCALE : ตัวเลขขนาด 6 บิท

DCYCLE : ตัวเลขขนาด 10 บิท

การคำนวณคาบของpwm

PWM cycle time = 76.9nS x (PWMCRx[PRESCALE] + 1) x (PWMPCRx[PV] + 1)

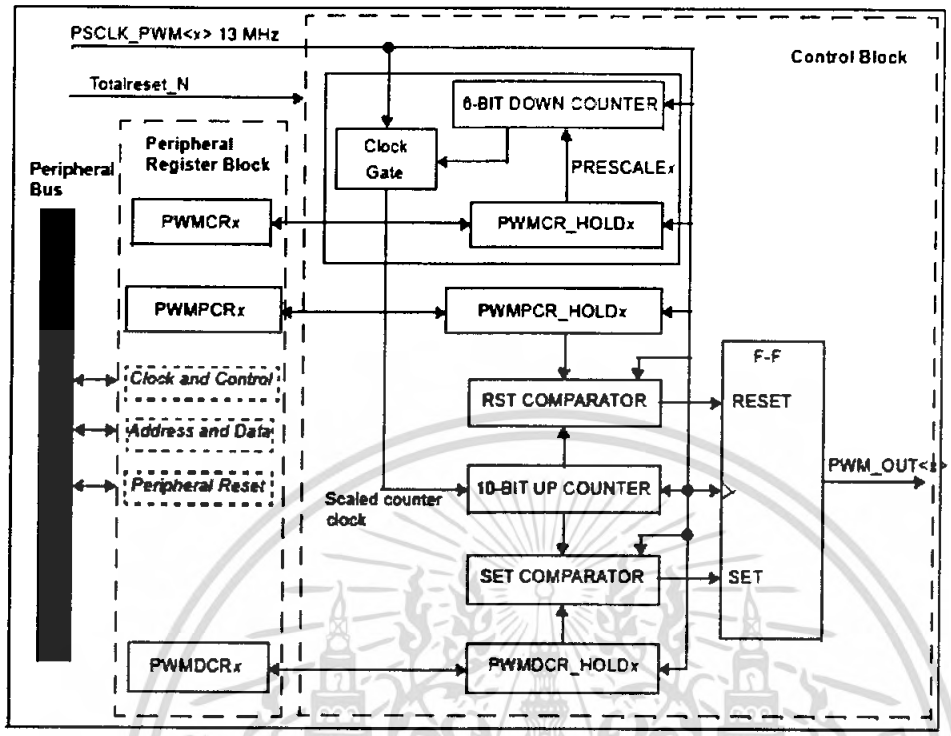
PV: ตัวเลขขนาด 10 บิท

พีดับเบิลยูเอ็ม รีจิสเตอร์

PWMCRx == เป็นรีจิสเตอร์ที่ใช้ในการหารclockความถี่ 13 เมกะเฮิร์ต

PWMDCRx == เป็นรีจิสเตอร์ที่ใช้ในการกำหนดดิวตี้ไซเคิล

PWMPCRx == เป็นรีจิสเตอร์ที่ใช้กำหนดคาบ



รูปที่ 3.20 แสดงไดอะแกรมของพีดับเบิลยูเอ็ม

การเขียนแอปพลิเคชั่น เพื่อใช้งาน พีดับเบิลยูเอ็ม ไดรเวอร์

จากตัวอย่างโค้ดแอปพลิเคชั่น ด้านล่างเป็นการใช้งานพีดับเบิลยูเอ็ม ไดร์เวอร์ โดยทำการเปิดโหนด

จากนั้นทำการกำหนดค่า สตรัก รีจิสเตอร์ เพื่อส่งค่าต่างๆเหล่านี้ผ่านทาง ioctl ไปให้พีดับเบิลยูเอ็ม ไดร์เวอร์ และไดร์เวอร์จะเป็นตัวไปจัดการกำหนดค่าภายในรีจิสเตอร์ของพีดับเบิลยูเอ็มต่อไป หลังจากนั้นจะได้สัญญาณพีดับเบิลยูเอ็ม ออกทาง PWM0

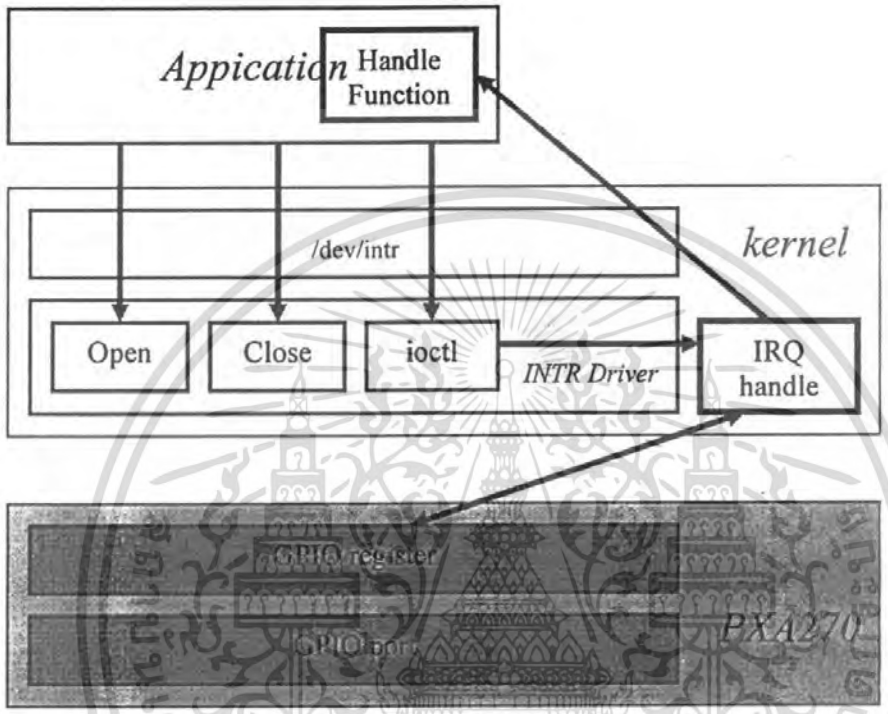
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <netinet/in/h>
#include < fcntl.h>
```

```
#define PWM0_ENABLE _IOW('k', 0, unsigned int)
#define PWM1_ENABLE _IOW('k', 1, unsigned int)
#define PWM0_DISABLE _IOR('k', 4, unsigned int)
#define PWM1_DISABLE _IOW('k', 5, unsigned int)
struct pwm_struct_register{
        unsigned int PWM_CTRLx;
        unsigned int PWM_PWDUTYx;
        unsigned int PWM_PERVALx;
};
int my_file;

int main (int argc, char *argv[])
{
        if((my_file=open("/dev/pwm",0_RDWR)) < 0){
                exit(1);
        }
        struct pwm_struct_register pwm_register;
        pwm_register = x;
        pwm_register = y;
        pwm_register = z;
        ioctl(my_file, PWM_ENABLE, &pwm_register);
        close(my_file);
        return 0;
}
```

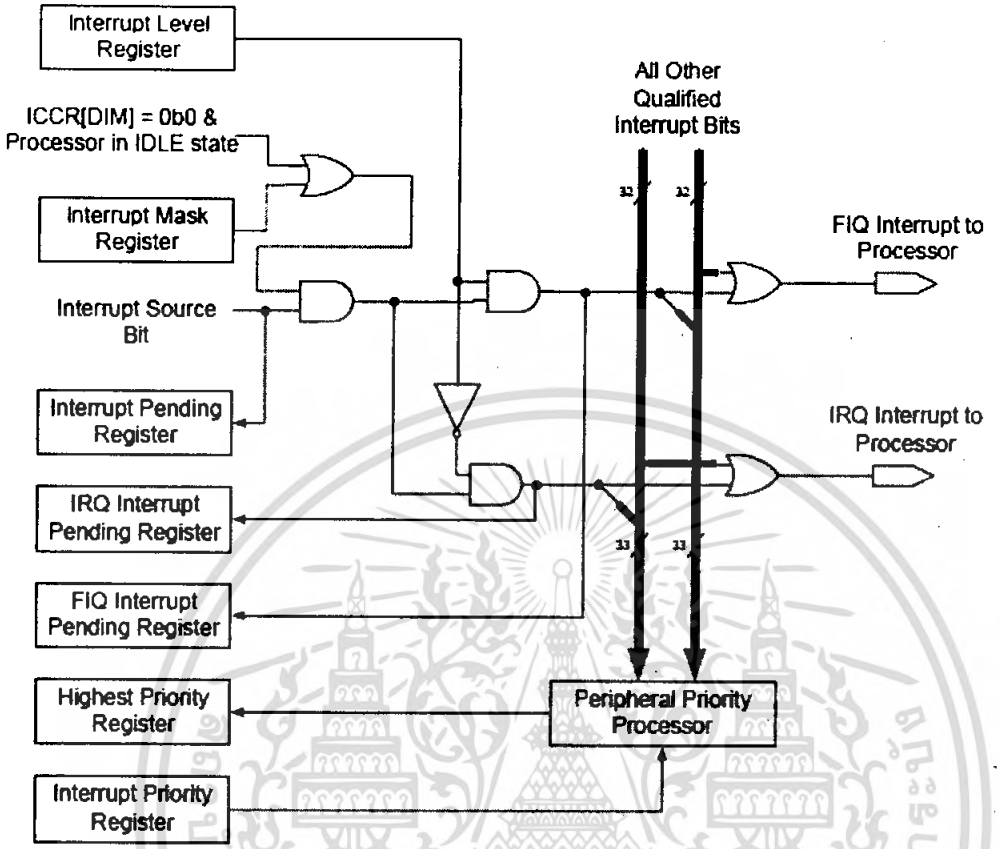
3.3.3 Interrupt ไดร์เวอร์

แสดงภาพไดอะแกรมการออกแบบของInterrupt ไดร์เวอร์ มีลักษณะคล้ายๆกับGPIO ไดร์เวอร์ และพีดับเบิลยูเอ็ม ซึ่งinterrupt ไดร์เวอร์เวอร์ชันแรกนี้จะให้ แอปพลิเคชั่น กระทำการกำหนดค่าต่างๆ อาทิเช่น ลักษณะการตรวจรับสัญญานอินเทอรัพต์ พอร์ตจีพีไอโอ และ แอปพลิเคชั่น สามารถส่ง

ฟังก์ชันให้แก่ไดร์เวอร์ เพื่อกำหนดค่าให้เคอร์เนล เรียกใช้ฟังก์ชันของแอปพลิเคชั่น ทำงานเมื่อเกิดการอินเทอรัพต์



รูปที่ 3.21 แสดงInterrupt ไดร์เวอร์



รูปที่ 3.22 แสดงไดอะแกรม ของสัญญานอินเทอรัพต์

การเขียนแอปพลิเคชั่นเพื่อใช้งาน Interrupt ไดร์เวอร์

จากตัวอย่างโค้ดแอปพลิเคชั่นด้านล่างเป็นการใช้งานInterrupt ไดร์เวอร์โดยทำการเปิดโหนด จากนั้นทำการแปลงค่าแอดเดรสของฟังก์ชันให้เป็นตัวชนิด unsined int เพื่อส่งค่าต่างๆเหล่านี้ผ่านทาง ioctl ไปให้ Interrupt ไดร์เวอร์และไดร์เวอร์ จะเป็นตัวไปจัดการกำหนดค่าและเซ็ต และเมื่อมีสัญญานอินเทอรัพต์เกิดขึ้นในขณะที่ยังวันลูปอยู่ก่อนก่อนส่งioctl เพื่อทำการปลดปล่อยสัญญานอินเทอรัพต์ ภายในตัวเคอร์เนลจะจัดการไปเรียกใช้ฟังก์ชัน application_intr

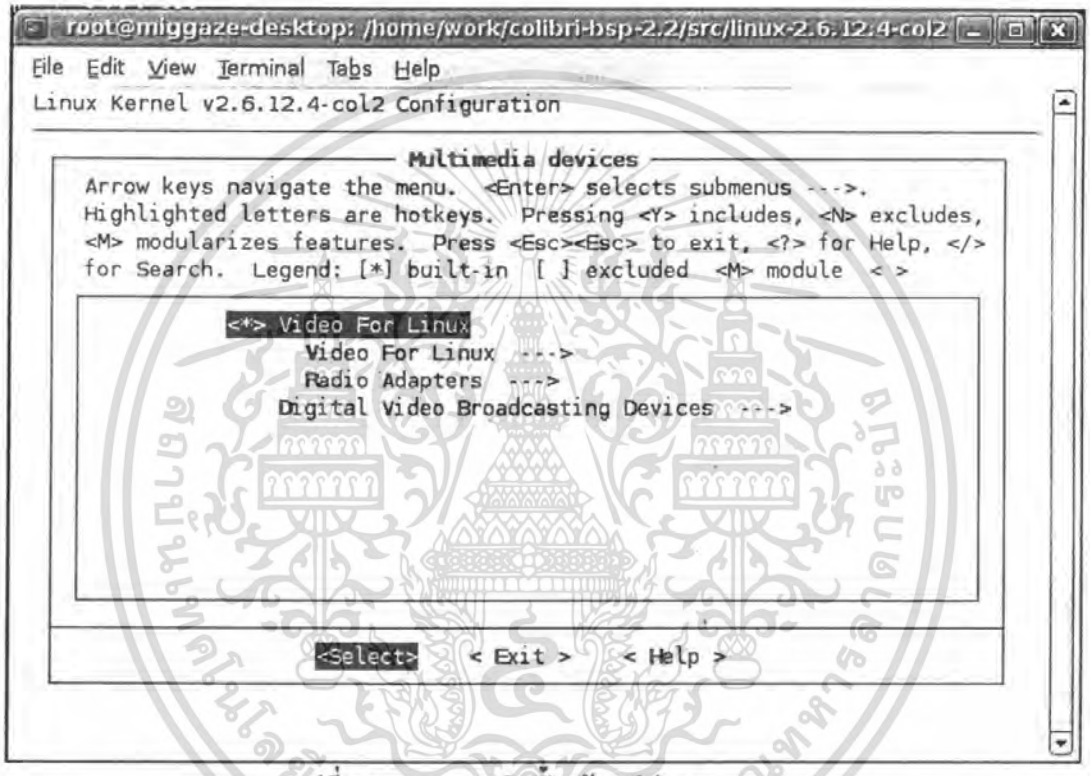
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
```

```
#include <netinet/in/h>
#include < fcntl.h>

#define INTR_CONFIG_F        _IOW('k', 0, unsigned int)
#define INTR_CONFIG_R        _IOW('k', 1, unsigned int)
#define INTR_RELEASE         _IOR('k', 2, unsigned int)
int my_file;

static void application_intr()
{
        printf(" I Love Project\n");
        return;
}
int main (int argc, char *argv[])
{
        if((my_file=open("/dev/intr",0_RDWR)) < 0){
                exit(1);
        }
        unsigned long addr = (unsigned long)application_intr;
        ioctl(my_file, INTR_CONFIG_F, &addr);
        int count = 1000;
        while(count--);
        ioctl(my_file,INTR_RELEASE,&addr);
        close(my_file);
        return 0;
}
```

3.3.4 การทำแพลตฟอร์มให้สนับสนุนการทำงานของเว็ปแคม

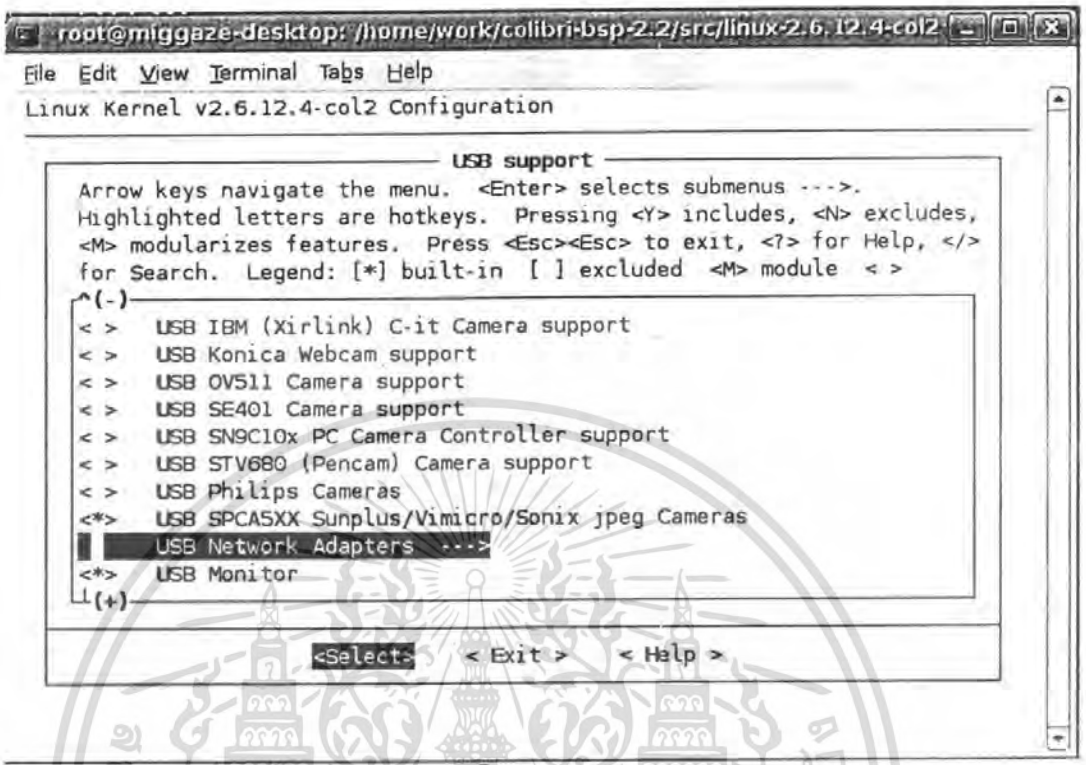
ดาวน์โหลดซอร์ส usb-2.6.12LE06.patch.tar แล้วนั้น เนื่องจากซอร์สยังไม่รองรับกล้องจึงจำเป็นทำให้กล้องรองรับเสียก่อน ดังนี้

- นำไฟล์ แพตช์ (usb-2.6.8.1-2.patch) ไปวางตำแหน่ง /drivers/usb
- เข้าไปตำแหน่ง /drivers/usb ของ เคอร์เนลซอร์สไฟล์
- patch -p1 < usb-2.6.12.patch
- make menuconfig
- เข้าไปใน Multimedia devices แล้วเลือก include Video For Linux



รูปที่ 3.23 แสดงการติดตั้งกล้องเว็ปแคม

- เข้าไปใน USB support แล้วเลือก ชิปเซ็ต ของกล้อง (USB SPCA5XX Sunplus/Vimicro/Sonix jpeg Webcamera)



รูปที่ 3.24 แสดงการติดตั้งกล้องเว็ปแคม(2)

จากนั้นบันทึก และออก

- make

จากนั้นจะได้ vmlinux ใน เคอร์เนลซอร์สไฟล์ ซึ่งนำไปทำเคอร์เนล อิมเมจ ที่จะโหลดลงแพลตฟอร์มต่อไป

3.4 การพัฒนาโมดูลต่างๆของหุ่นยนต์ให้เป็นยูเอสบีดีไวซ์

เป็นลักษณะการพัฒนาชิ้นส่วนต่างๆ ของหุ่นยนต์ให้เป็นรูปแบบของ ยูเอสบีดีไวซ์ ซึ่งสามารถปั๊กอินเข้าทางพอร์ตยูเอสบี เพื่อใช้งาน ซึ่งจะเป็นการนำข้อดีต่างของยูเอสบีมาใช้กับการพัฒนาหุ่นยนต์ เช่น

- มีความเร็วในการรับส่งข้อมูลสูง ยูเอสบี1.0 = 1.5 เมกะบิทต่อวินาที , ยูเอสบี 1.1 = 12 เมกะบิทต่อวินาที , ยูเอสบี 2.0 = 480 เมกะบิทต่อวินาที
- มีลักษณะของ ปั๊กแอนด์เพล์ (สามารถเรียนรู้ได้ว่าเป็นดีไวซ์แบบไหน และทำงานอย่างไร จากการอ่าน คำอธิบาย)

- มีลักษณะเป็นบัสทำให้สามารถเชื่อมต่อได้หลายอุปกรณ์ในเวลาเดียวกัน รวมถึงสามารถต่อยูเอสบีฮับ เพื่อเพิ่มพอร์ตในการเชื่อมต่อได้(สามารถเชื่อมต่ออุปกรณ์ได้สูงสุดถึง 127 ตัว ถ้าการจัดการด้านพลังงานดีพอ)
- ประหยัดข หรือจีพีไอโอที่ใช้ในการเชื่อมต่อ
- เมื่อเปลี่ยนชิ้นส่วนฮาร์ดแวร์ของยูเอสบีดีไวซ์ก็ไม่จำเป็นต้องเปลี่ยนแปลงดีไวซ์ไดร์เวอร์และแอปพลิเคชั่น

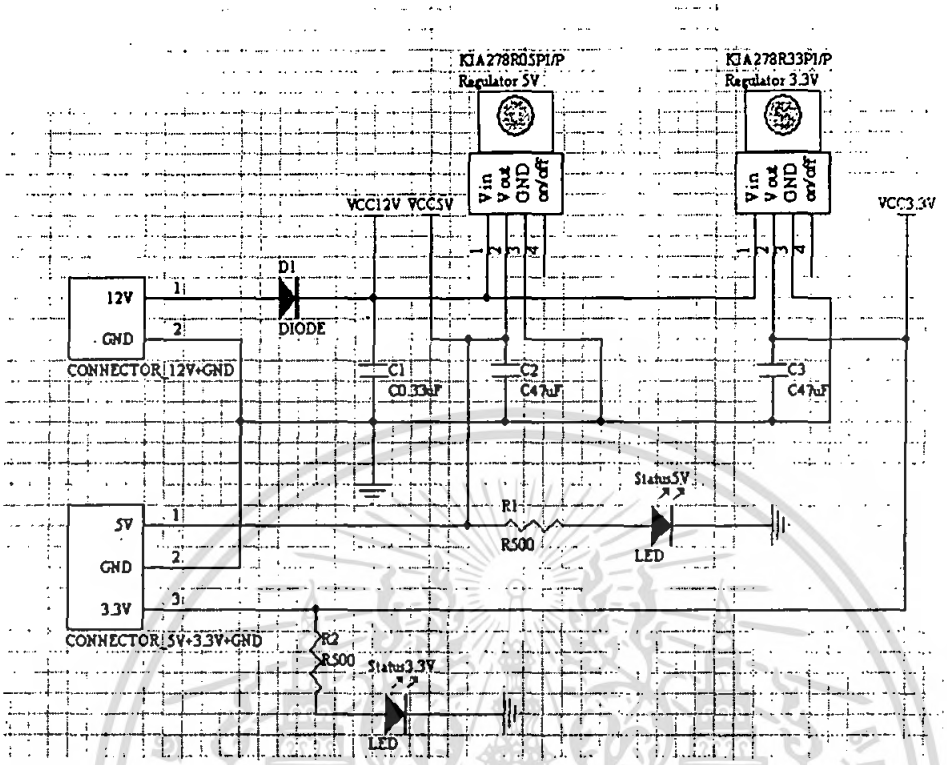
3.4.1 การพัฒนายูเอสบีดีไวซ์

ยูเอสบีดีไวซ์เป็นอุปกรณ์ที่นำมาเชื่อมต่อกับโฮสผ่านทางพอร์ตยูเอสบีซึ่งสามารถพัฒนาจากไมโครคอนโทรเลอร์ที่มีอุปกรณ์ภาคที่ใช้ในการเชื่อมต่อกับพอร์ตยูเอสบีอยู่ด้วย พร้อมกับจัดสรรรีจิสเตอร์ฟังก์ชัน พิเศษเพื่อรองรับการทำงานกับพอร์ตยูเอสบีเพื่อช่วยให้การใช้งานสะดวกขึ้น ตัวอย่างของไมโครคอนโทรเลอร์ชนิดนี้ได้แก่ PIC16C745 , PIC18F2410 , PIC18F4550 ของ Microchip , CY7C63001 และ โมดูล EZ-USB ของ Cypress เป็นต้น ซึ่งในที่นี้เราได้นำ PIC18F4550 มาพัฒนาเป็นตัวอย่าง ยูเอสบีดีไวซ์ ได้โดยขั้นตอนดังนี้

- ออกแบบและพัฒนาวงจรทางด้านฮาร์ดแวร์
- การเขียนโปรแกรมเดสคริปเตอร์ ต่าง ๆ , การ คอนฟิค , การรับส่งข้อมูลผ่านทางยูเอสบี , และการควบคุม อินพุต/เอาต์พุท บน ไมโครคอนโทรเลอร์
- การทดสอบยูเอสบีดีไวซ์

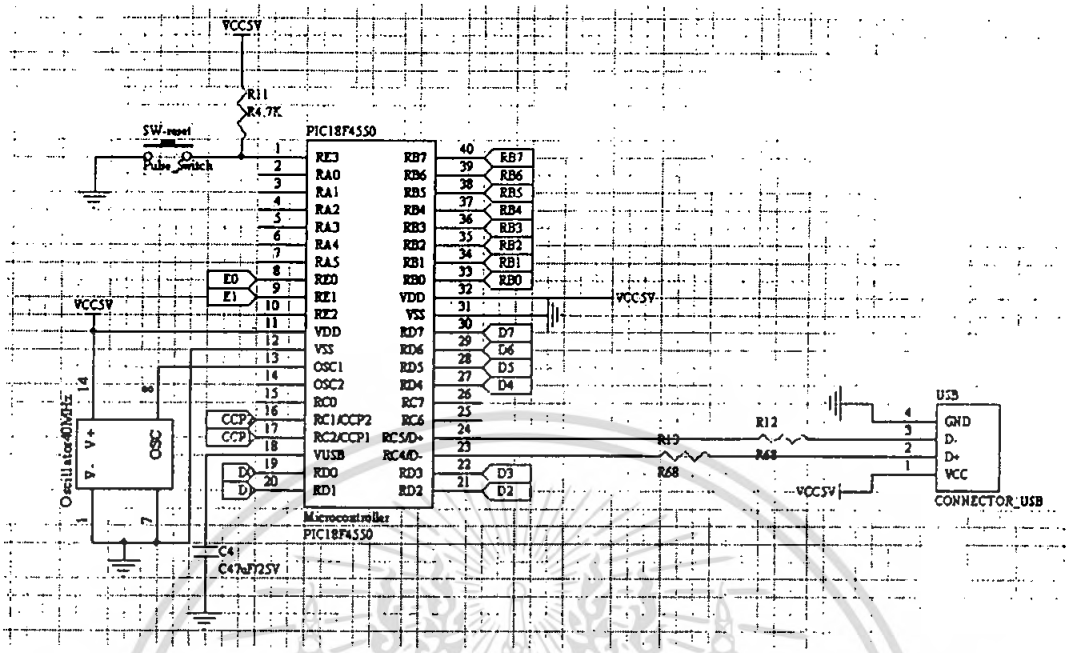
3.4.2 การออกแบบและพัฒนาวงจรยูเอสบีดีไวซ์ทางด้านฮาร์ดแวร์

- วงจร เรกูเลเตอร์: เป็นวงจรที่ทำการแปลงกระแสและความต่างศักย์ให้เหมาะกับอุปกรณ์ที่ใช้ ในที่ใช้ไฟ 5โวลต์(ใช้กับไมโครคอนโทรเลอร์) และ 3.3โวลต์(ใช้ พลูอัพ) ถ้าในกรณีที่ไฟเลี้ยงจากพอร์ตยูเอสบีของโฮสคงที่และมีค่าพอที่จะใช้ในอุปกรณ์ ก็ไม่จำเป็นที่จะต้องมีวงจร เรกูเลเตอร์ ก็ได้



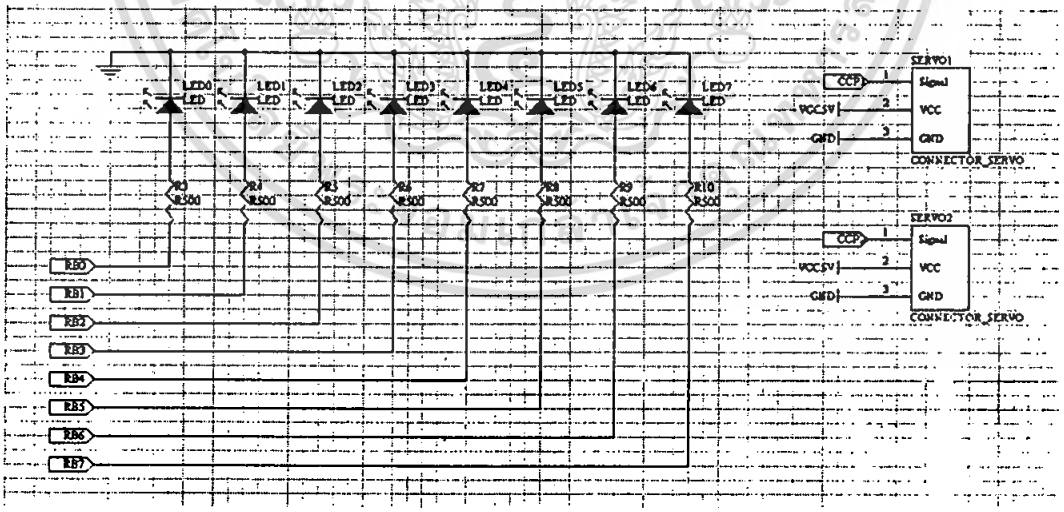
รูปที่ 3.25 วงจร เรกูเลเตอร์

- **วงจรไมโครคอนโทรเลอร์ : เป็นวงจรที่ต่อเพื่อให้ไมโครคอนโทรเลอร์ทำงานได้ รวมถึงการ *เชื่อมต่อกับพอร์ตยูเอสบีด้วย***



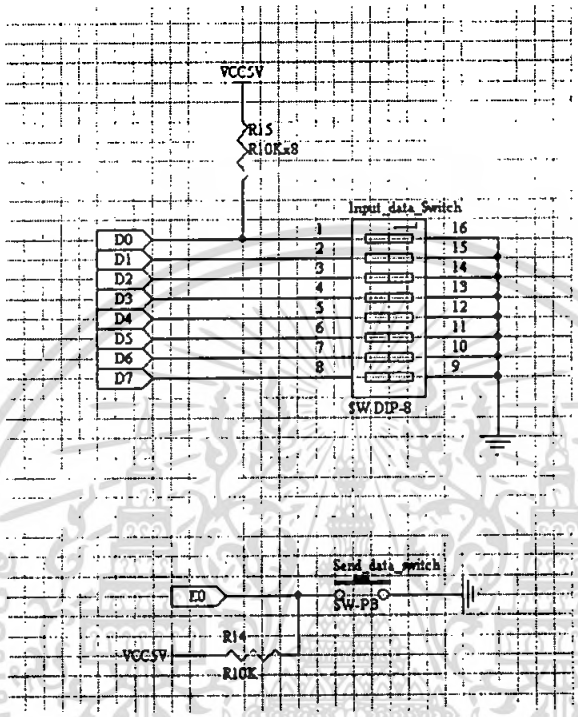
รูปที่ 3.26 วงจร ไมโครคอนโทรเลอร์

- **วงจรเอาต์พุท : เป็นวงจรที่ใช้ในการแสดงผลการเชื่อมต่อและการรับส่งข้อมูลในที่นี้ใช้เป็น แอลอีดี และ เซอร์โวมอเตอร์**



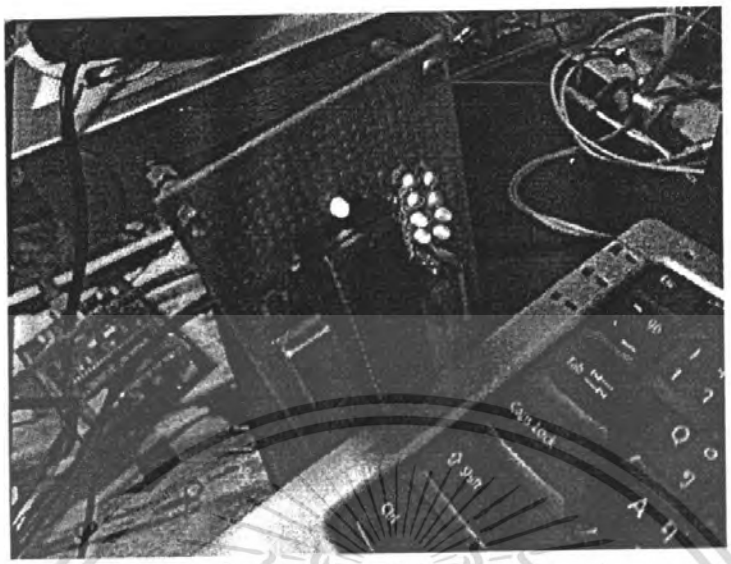
รูปที่ 3.27 วงจรเอาต์พุท

- วงจรอินพุท: เป็นวงจรที่ใช้รับค่าจากผู้ใช้หรือเซ็นเซอร์เพื่อนำมาทำการ คอนฟิคหรือ ส่งข้อมูลให้โฮสประมวลผลผ่านทางพอร์ตยูเอสบี



รูปที่ 3.28 วงจรอินพุท

เมื่อนำวงจรที่ออกแบบไว้มาพัฒนาได้ดังนี้



รูปที่ 3.29 ยูเอสบีดีไวซ์ติดต่อกับพีซี

รูปที่ 3.30 ยูเอสบีไดร์เวอร์ติดต่อกับแพลตฟอร์ม

3.4.3 การเขียนโปรแกรมบนยูเอสบีดีไวซ์

- การเขียนเดสคริปเตอร์ : เป็นการประกาศข้อมูลต่างๆ ของยูเอสบีดีไวซ์รวมถึงการคอนฟิค ช่องทางและรูปแบบในการรับส่งข้อมูล ต้องมีการประกาศทั้ง ดีไวซ์ เดสคริปเตอร์, คอนฟิคกูเรชั่น เดสคริปเตอร์, อินเทอร์เฟส เดสคริปเตอร์และ เอ็นพอยท์ (endpoint) เดสคริปเตอร์ซึ่งตัวอย่างการประกาศเป็นดังนี้

```
char const USB_DEVICE_DESC[] ={
    USB_DESC_DEVICE_LEN,        //the length of this report
    0x01,           //constant DEVICE (0x01)
    0x10,0x01,        //usb version in bcd
    0x00,           //class code (if 0, interface defines class.  FF is vendor defined)
    0x00,           //subclass code
    0x00,           //protocol code
    USB_MAX_EP0_PACKET_LENGTH,      //max packet size for endpoint 0.
                                    //(SLOW SPEED SPECIFIES 8)
    0xD8,0x04,        //vendor id (0x04D8 is Microchip)
    0x11,0x00,        //product id, me gusta el 11 ;)
    0x01,0x00,        //device release number
    0x01,       //index of string description of manufacturer. therefore we point
              //to string_1 array (see below)
    0x02,           //index of string descriptor of the product
    0x00,           //index of string descriptor of serial number
    USB_NUM_CONFIGURATIONS  //number of possible configurations
};
```

- การรับส่งข้อมูลผ่านทางยูเอสบี : เป็นการใช้คำสั่งต่างๆ ที่สร้างโดยพัฒนาไมโครคอนโทรเลอร์ โดยสามารถเข้าไปดูได้ใน file pic18_usb.h , usb.h และ usb.c ซึ่งมี คำสั่งที่สำคัญดังนี้
  - usb_init() : เป็นคำสั่งที่ใช้ในการกำหนดค่าเริ่มต้น การทำงานของโมดูลยูเอสบี
  - usb_wait_for_enumeration() : เป็นคำสั่งที่ใช้ในการรอการติดต่อจากโฮสโดยที่ถ้าไม่มีการติดต่อจากโฮสก็จะไม่ทำคำสั่งถัดไป

- usb_enumerated() : เป็นคำสั่งที่ใช้ในการคอนฟิคการเชื่อมต่อระหว่างโฮสและดีไวซ์ รวมถึงการส่ง เดสคริปเตอร์ ให้กับโฮส
- usb_kbhit() : เป็นคำสั่งที่ใช้ในการตรวจสอบการติดต่อจากโฮส
- usb_get_packet() : เป็นคำสั่งที่ใช้ในการอ่านข้อมูลจากบัฟเฟอร์ขาเข้าของยูเอสบี
- usb_put_packet() : เป็นคำสั่งที่ใช้ในการส่งข้อมูลออกไปที่โฮส

3.4.4 การพัฒนายูเอสบีดีไวซ์ไดร์เวอร์

เป็นการพัฒนาไดร์เวอร์ที่ทำให้ยูเอสบีดีไวซ์สามารถทำงานได้ตามต้องการ ซึ่งในที่นี้เราได้พัฒนายูเอสบีดีไวซ์ไดร์เวอร์ทั้งในโฮสที่เป็นอาร์ม ลินุกซ์ 2.4 และ อาร์ม ลินุกซ์2.6 โดยใช้หลักการของไฟล์โอเปอเรชั่นที่เป็นที่เก็บโอเปอเรชั่นต่างๆ ที่ใช้กับดีไวซ์นั้นๆ ตัวอย่างเช่น

```
static struct file_operations DEVICE_fops = {
    .owner =    THIS_MODULE,
    .read =     DEVICE_read,
    .write =    DEVICE_write,
    .open =     DEVICE_open,
    .ioctl =    DEVICE_ioctl,
    .release =  DEVICE_release,
};
```

ซึ่งเป็นการแมปฟังก์ชันในระดับเคอร์เนลให้สามารถนำขึ้นไปใช้บนแอปพลิเคชั่นได้ซึ่งชื่อที่อยู่ทางขวาเป็นฟังก์ชันในระดับเคอร์เนลและทางซ้ายเป็นฟังก์ชันในระดับแอปพลิเคชั่นโดยฟังก์ชันที่สำคัญได้แก่

- DEVICE_open() : ใช้ในการเริ่มการติดต่อกับดีไวซ์(return file descriptor)
- DEVICE_read() : ใช้ในการรับข้อมูลจากดีไวซ์
- DEVICE_write() : ใช้ในการส่งข้อมูลให้กับดีไวซ์

โดยการทำให้ยูเอสบีคอร์นั้นรู้จักยูเอสบีดีไวซ์ไดร์เวอร์ทำได้โดยการใช้คำสั่ง usb_register() ใน module init function และใช้คำสั่ง usb_register() ใน module exit

ส่วนคำสั่งในการ Complie USB device driver ใช้

```
mknod -m 700 /dev/usb1 180 144
insmod usbdevicedriver.ko
```

3.4.5 การพัฒนาแอปพลิเคชั่นของยูเอสบีดีไวซ์

เป็นการเขียนโปรแกรมโดยใช้หลักการเดียวกับ การสั่งการดีไวซ์อื่นๆ คือ ใช้ฟังก์ชันจากที่ประกาศไว้ในไฟล์โอเปอเรชั่นในดีไวซ์ไดร์เวอร์ตัวอย่างเช่น

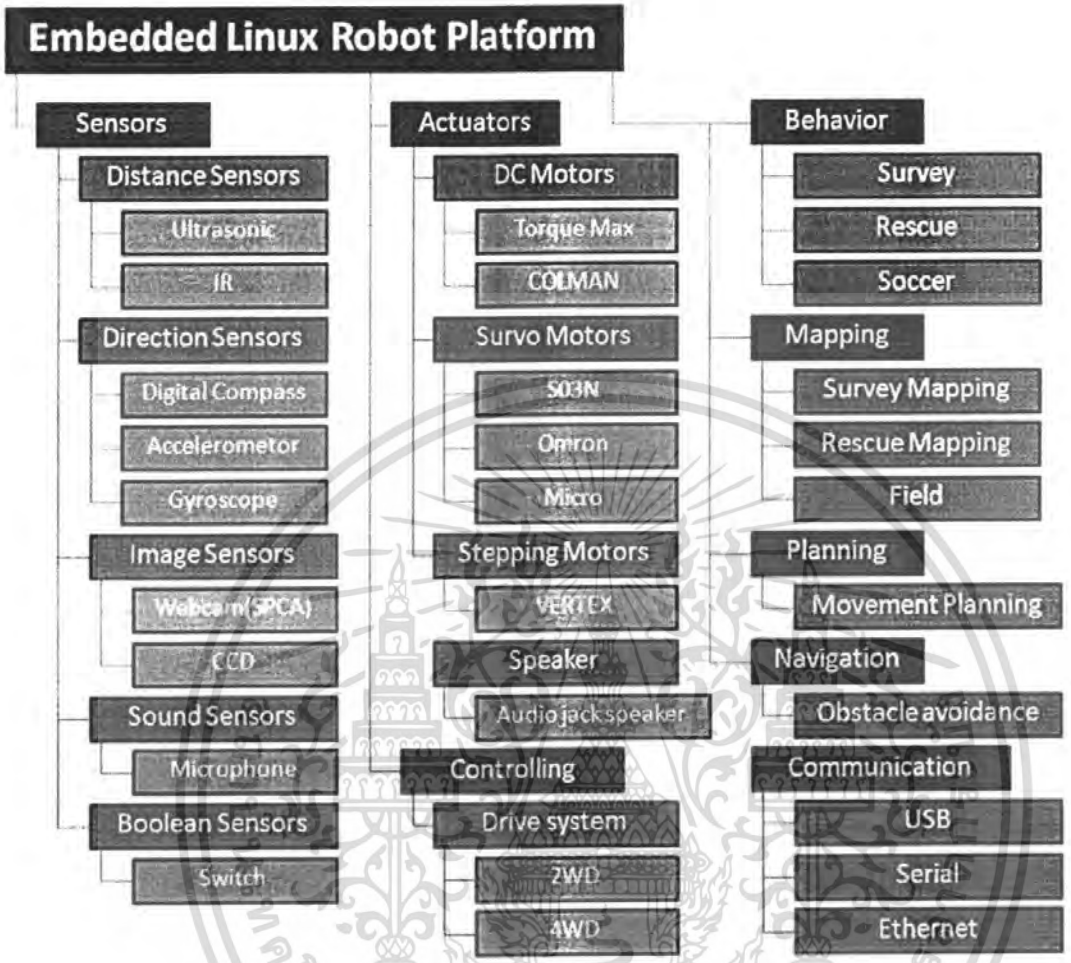
```
open("/dev/usbl",O_RDONLY | O_WRONLY)   //ใช้เพื่อเปิดการทำงานของดีไวซ์ โดย รีเทอร์น
                                        //ไฟล์เดสคริปเตอร์ ออกมา
read(f1,dataout,3);     //เป็นการอ่านข้อมูลจากยูเอสบีดีไวซ์ของไฟล์โอเปอเรชั่น(f1) นั้นๆ จำนวน
                        //3 ไบต์
write(f1,dataout,3);    //เป็นการส่งข้อมูลไปที่ยูเอสบีดีไวซ์ของ ไฟล์โอเปอเรชั่น(f1) นั้นๆ จำนวน
                        //3ไบต์
```

ส่วนคำสั่งในการคอมไพล์แอปพลิเคชั้นของยูเอสบีดีไวซ์ใช้

```
arm-linux-gcc usbdeviceapplication.c –o usbdeviceapplication
```

3.4.6 การออกแบบซอร์ฟแวร์โครงสร้าง และการเชื่อมต่อโมดูล

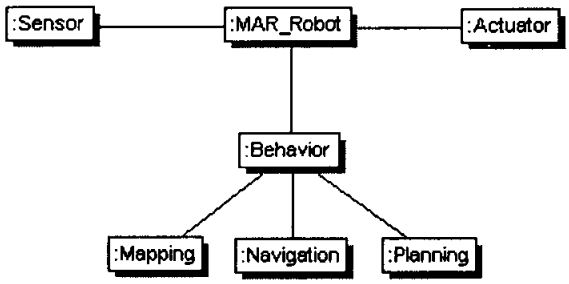
จากที่กล่าวแนะนำมาแล้วว่าโครงงานนี้เป็นการพัฒนาแพลตฟอร์มซึ่งสามารถนำไปพัฒนาโมบายค์โรบอทซึ่งจะได้แนวทางการพัฒนา และรูปร่างของแพลตฟอร์มดังรูป



รูปที่ 3.31 แสดงแนวทางของโครงงาน

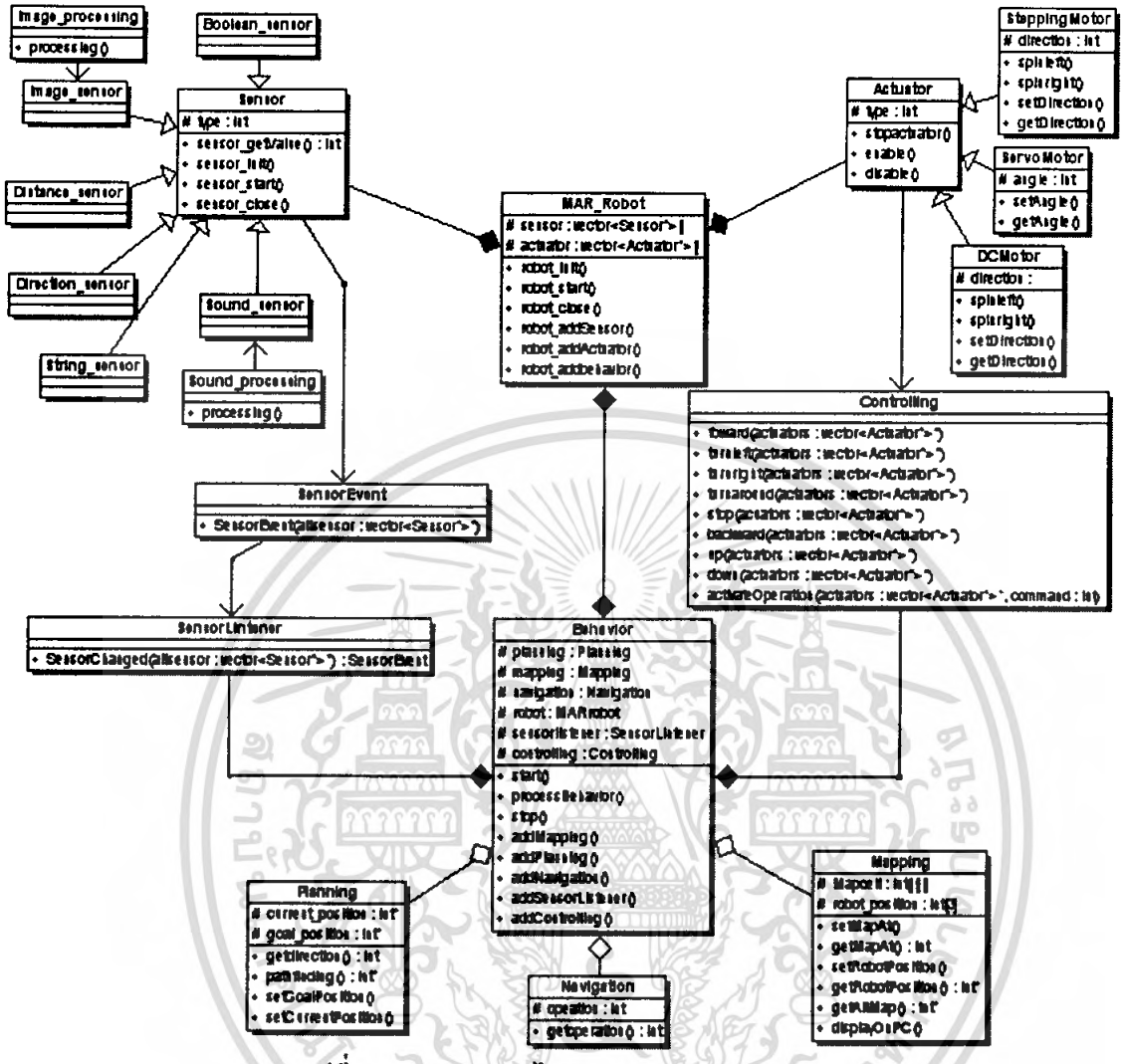
โดยโครงงานนี้จะต้องสร้างการทำงาน และส่วนประกอบโมดูลต่างของหุ่นยนต์ เซ็นเซอร์ (Sensor), ส่วนเคลื่อนที่หรือแสดงผล (Actuator), พฤติกรรม (Behavior), การสร้างแผนที่ (Mapping), การวางแผน (Planning), การนำทาง (Navagation) และการสื่อสาร (Communication) ซึ่งผู้ใช้สามารถนำไปใช้งานเพื่อนำไปหุ่นยนต์ได้อย่างสะดวก และง่ายดาย และโดยโมดูลแต่เรื่องนั้นภายในจะมีโมดูลซอร์ฟแวร์แต่ละเรื่องที่จะต้องพัฒนาขึ้นดังตัวอย่างเช่น โมดูลเซ็นเซอร์ภาพ ก็จะมีอุปกรณ์เว็ปแคม, ซีซีดี และอื่นๆที่จะใส่เพิ่มเข้าไปอีกเรื่อยๆ

หลังจากที่ได้ทำการศึกษา หาข้อมูล และวิเคราะห์เพื่อทำการออกแบบโครงสร้างซอร์ฟแวร์ที่เป็นออบเจ็ค (object) ที่จะใช้ในแพลตฟอร์มแล้วนั้นซอร์ฟแวร์โครงสร้างที่ได้สามารถทำการอธิบายได้ด้วยยูเอ็มเอล (UML : Unified Modeling Language) ดังนี้



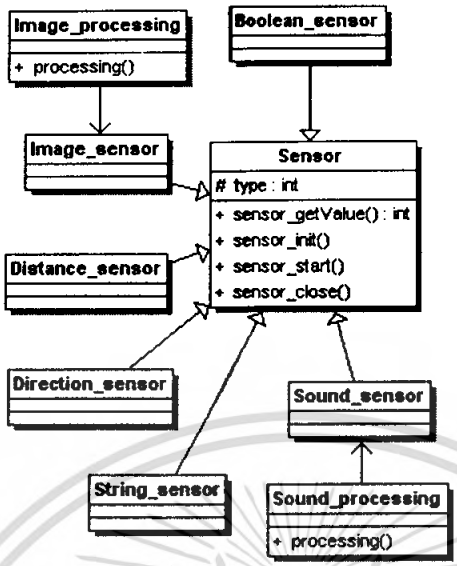
รูปที่ 3.32 แสดงอ็อบเจ็คไดอะแกรมของระบบ

ซึ่งโครงสร้างหลักๆ ของระบบจะเป็น ตัวหุ่นยนต์ (MAR_Robot) ซึ่งจะมีส่วนประกอบเป็นส่วนของการรับข้อมูลจากสิ่งแวดล้อม (Sensor) ส่วนที่กระทำต่อสิ่งแวดล้อม (Actuator) และส่วนที่เป็นพฤติกรรมของหุ่นยนต์ (Behavior) ซึ่งพฤติกรรมของหุ่นยนต์จะประกอบไปด้วย การวางแผน (Planning) การนำทาง (Navigation) และการจดจำเส้นทางหรือแผนที่ (Mapping) ซึ่งหลังจากที่ได้ อ็อบเจ็คไดอะแกรมแล้วจึงทำการออกแบบลักษณะความสัมพันธ์และรูปแบบการเชื่อมต่อต่างๆ ได้ดังนี้



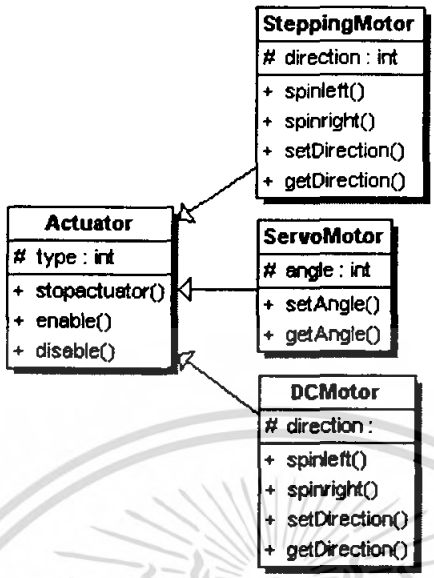
รูปที่ 3.33 แสดงคลาสไดอะแกรมของระบบ

จากคลาสไดอะแกรมจะเห็นได้ว่า แต่ละคลาสจะมีหน้าที่ ความสัมพันธ์และการเชื่อต่อที่แต่ต่างกันไปซึ่งจะอธิบายเป็นส่วนๆ ดังต่อไปนี้



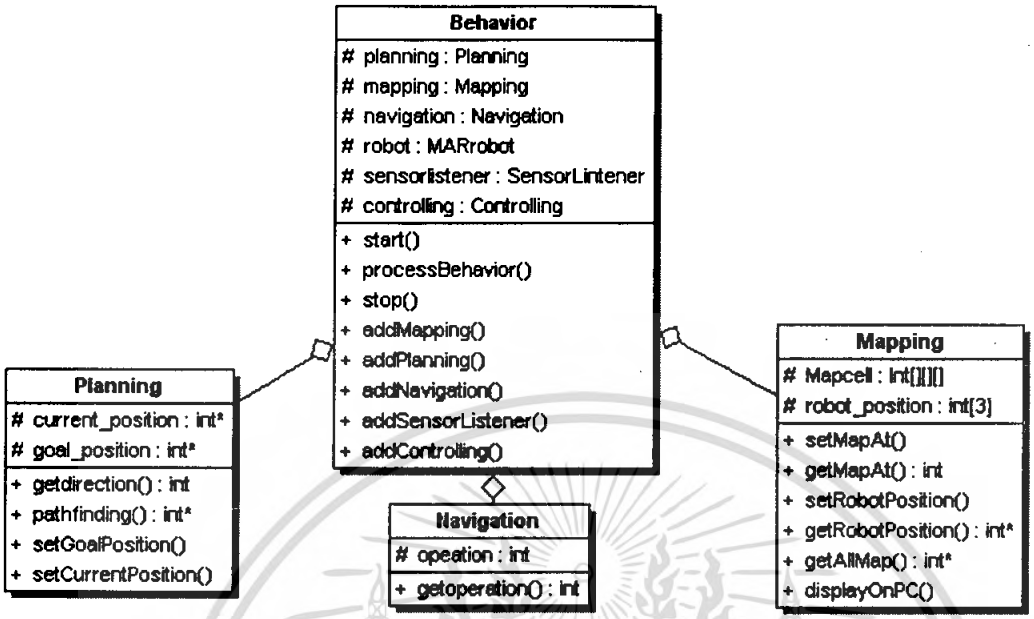
รูปที่ 3.34 แสดงคลาสไดอะแกรมของเซ็นเซอร์

คลาส Sensor เป็นซูเปอร์คลาสที่มีเมทธอดหลักๆ ที่เซ็นเซอร์ ทุกๆอย่างควรจะมี ซึ่งผู้ใช้งานสามารถ ถ่ายทอดคลาส นี้ไปเป็นเซ็นเซอร์ชนิดต่างๆ ที่เจาะจงมากขึ้น ซึ่งในโครงงานนี้ประกาศชนิดของเซ็นเซอร์เป็น 6 ชนิด ได้แก่ boolean sensor, image sensor, distance sensor, direction sensor, string sensor, sound sensor เป็นต้น ซึ่งเซ็นเซอร์แต่ละชนิดสามารถถูกถ่ายทอดไปเป็นเซ็นเซอร์ที่เจาะจงรุ่น ซึ่งในโครงงานนี้ได้ทำ การรวบรวมคลาสของเซ็นเซอร์ต่างๆ ที่พร้อมใช้งานไว้ให้กับผู้ใช้งาน เช่น ultrasonicSRF05 sensor, digital compass ,switch เป็นต้น



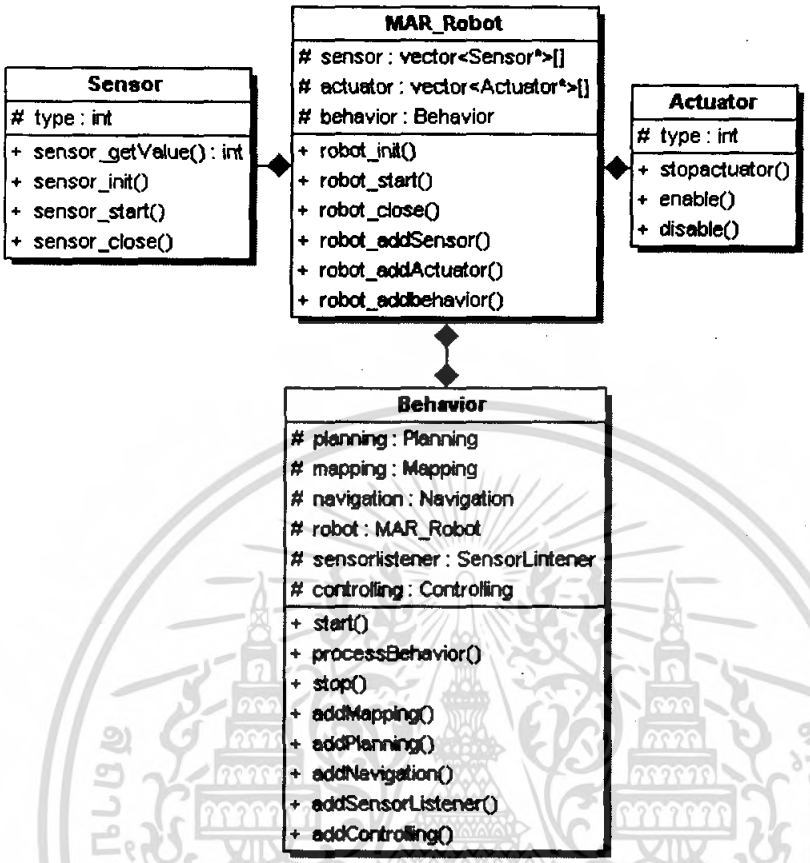
รูปที่ 3.35 แสดงคลาสไดอะแกรมของส่วนเคลื่อนที่หรือแสดงผล

คลาส Actuator เป็นซุปเปอร์คลาสที่มีเมทธอดหลักๆ ที่ส่วนเคลื่อนที่หรือแสดงผลทุกๆอย่างควรจะมี ซึ่ง ผู้ใช้งานสามารถ ถ่ายทอดคุณสมบัติ นี้ไปเป็น ส่วนเคลื่อนที่หรือแสดงผลชนิดต่างๆ ที่เจาะจงมากขึ้น ซึ่งในโครงงานนี้ประกาศชนิดของส่วนเคลื่อนที่หรือแสดงผลเป็น 3 ชนิด ได้แก่ stepping motor, servo motor, DC motor เป็นต้น ซึ่งส่วนเคลื่อนที่หรือแสดงผลแต่ละชนิดสามารถถูกถ่ายทอดไปเป็นส่วนเคลื่อนที่หรือแสดงผลที่เจาะจงรุ่นซึ่งในโครงงานนี้ได้ทำ การรวบรวมคลาสของส่วนเคลื่อนที่หรือแสดงผลต่างๆ ที่พร้อมใช้งานไว้ให้กับ ผู้ใช้งาน เช่น จีพีไอโอ DC motor เป็นต้น



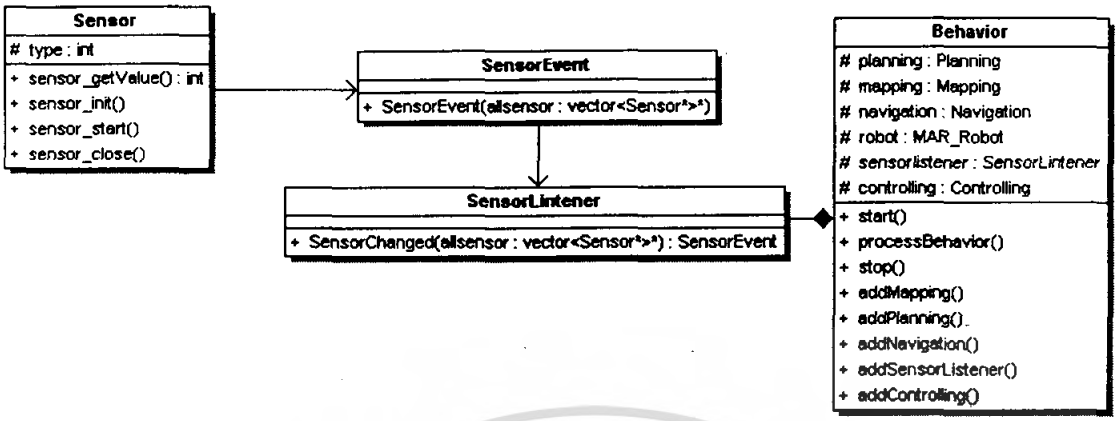
รูปที่ 3.36 แสดงคลาสไดอะแกรมของส่วนพฤติกรรม

คลาส Behavior เป็นซุปเปอร์คลาสของ พฤติกรรมของหุ่นยนต์ ภายในคลาสจะประกอบไปด้วยการวางแผน (planning) , การนำทาง (navigation), การสร้างแผนที่ (mapping), การควบคุม (controlling), และการรับข้อมูลจากสิ่งแวดล้อม (sensorlistener) ซึ่งหุ่นยนต์แต่ละชนิดอาจจะมีหรือไม่มีส่วนใดส่วนหนึ่งในนี้ขึ้นอยู่กับความซับซ้อนของหุ่นยนต์นั้นๆ



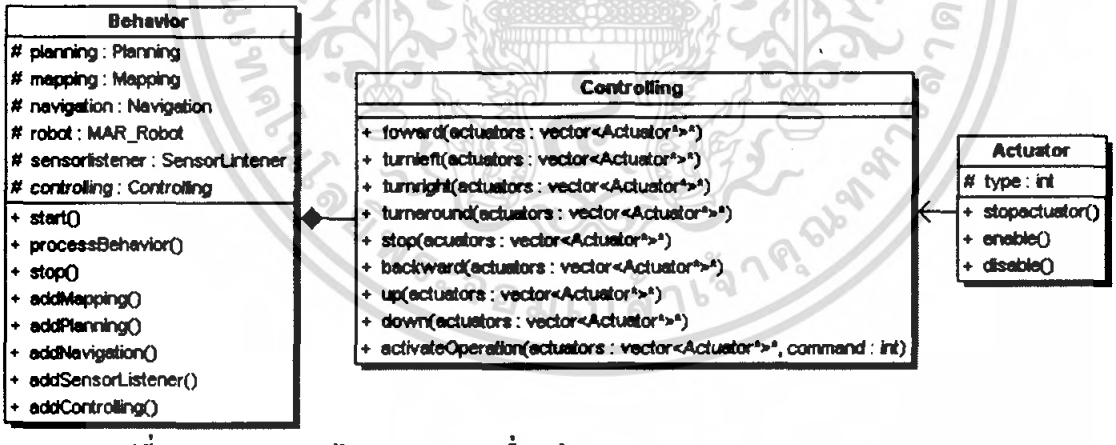
รูปที่ 3.37 แสดงคลาสไดอะแกรมของคลาสหุ่นยนต์

คลาส MAR_Robot เป็น คลาสพื้นฐาน ที่แทน mobile autonomous robot ที่จะประกอบไปด้วย เซ็นเซอร์ , ส่วนเคลื่อนที่หรือแสดงผล และ พฤติกรรม ซึ่ง ผู้ใช้งาน สามารถ ถ่ายทอดคุณสมบัติ คลาส MAR_Robot ไปเป็นคลาสของหุ่นยนต์ชนิดต่างๆ เช่นคลาส Survey_Robot ซึ่งได้จัดทำไว้ใน platform ที่พร้อมเรียกใช้งาน รวมทั้ง คลาส MAR_Robot จะเป็นส่วนประกอบของพฤติกรรม เพื่อให้ทราบว่า พฤติกรรมนั้นๆ เป็นของหุ่นยนต์ประเภทใด



รูปที่ 3.38 แสดงคลาสไดอะแกรม การเชื่อมต่อของ คลาส Behavior และ Sensor

การเชื่อมต่อระหว่าง คลาส Sensor และ Behavior นั้นจะใช้ คลาส SensorEvent เพื่อทำการกำเนิด เหตุการณ์ที่ได้จากเซ็นเซอร์ ต่างๆ ซึ่งคลาส SensorEvent นี้จะถูกดักจับโดยคลาส SensorListener แล้วทำการส่งค่าไปให้ Behavior ทำการจัดการ โดยการทำการต่างๆ เช่น ถ้าเจอสิ่งกีดขวางข้างหน้าตัวหุ่น SensorEvent ก็จะสร้างค่าที่บอกว่ามีสิ่งกีดขวางอยู่ด้านหน้าและ SensorListener ก็จะทำการส่งต่อค่านี้ไปให้ Behavior สั่งการ เป็นต้น

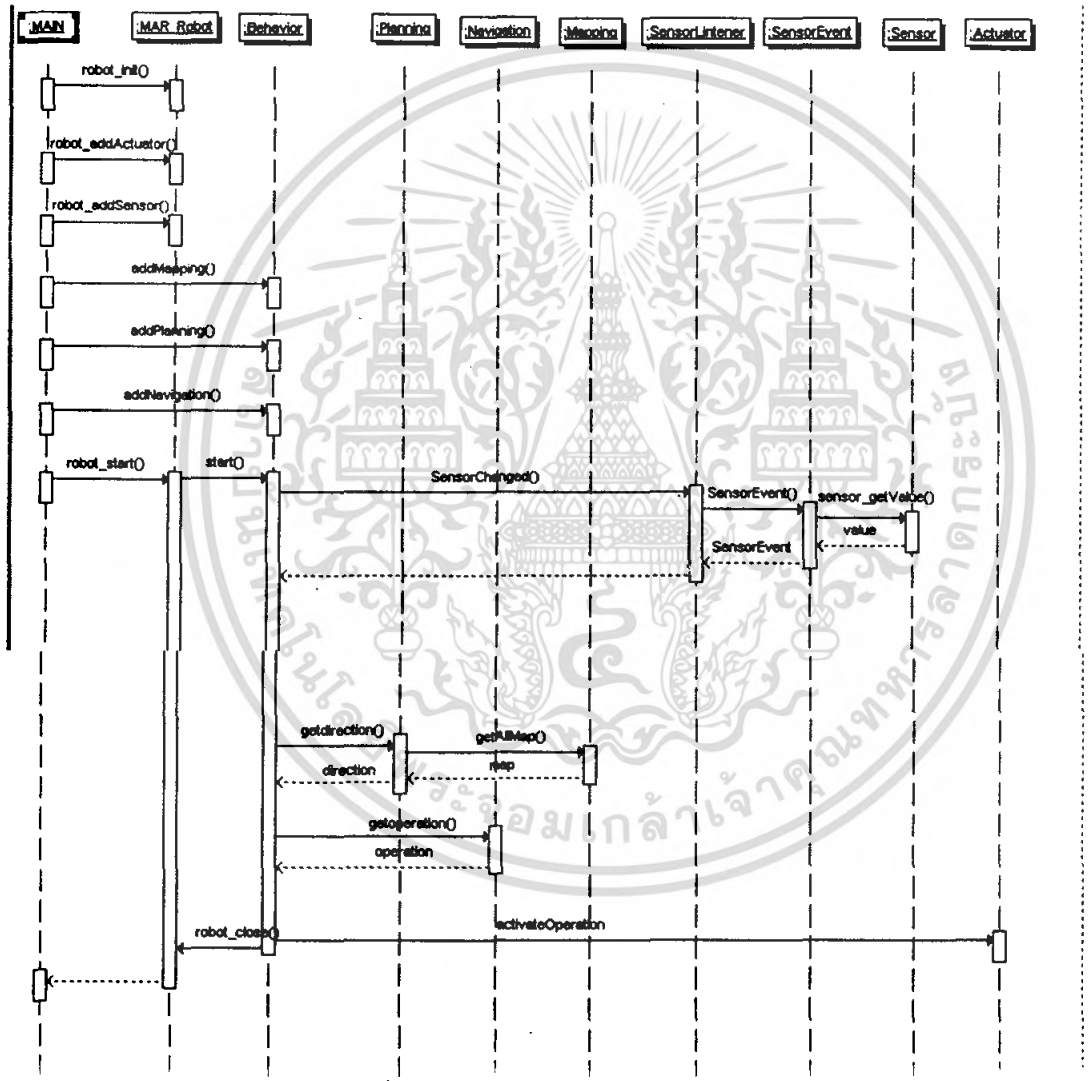


รูปที่ 3.39 แสดงคลาสไดอะแกรมการเชื่อมต่อของ คลาส Behavior และ Actuator

การเชื่อมต่อระหว่าง คลาส Behavior และ Actuator นั้นจะใช้ คลาส ที่เป็นสื่อกลางคือ คลาส Controlling ที่ทำหน้าที่รับคำสั่งจาก Behavior และทำการควบคุมให้ Actuator ต่างๆ ทำงานได้อย่างถูกต้อง เช่น ถ้า Behavior สั่งการมาว่าให้ หุ่นยนต์เลี้ยวซ้าย Controlling ก็จะทำการส่งสั่งการให้มอเตอร์

ทางด้านซ้ายหมุนไปในทิศทางที่ไปข้างหลัง และสั่งการให้มอเตอร์ทางด้านขวาหมุนไปในทิศทางที่ไปข้างหน้า เป็นต้น

หลังจากที่ได้ทำการออกแบบคลาสต่างที่จะต้องมีในระบบแล้ว ต่อไปเป็นการเขียนไดอะแกรมแสดงลำดับการทำงานของโครงสร้างซอร์ฟแวร์ซึ่งจากการวิเคราะห์ถึงการทำงานของMobile Robot ประเภทนี้แล้ว จะได้ลำดับการทำงานตามไดอะแกรมแสดงลำดับการทำงานข้างล่าง



รูปที่ 3.40 ตัวอย่างซีเคว้นซ์ไดอะแกรม

ซึ่งแนวคิดก็คือ ผู้ใช้งานเพียงแต่ทำการอิมพลีเมนท์ (implement) คลาสโดยทำการเพิ่มเติมรายละเอียดการทำงานจำเพาะเข้าไป และสร้างส่วนประกอบของหุ่นยนต์ ตามหลักของโอโอพี (OOP : Object Oriented Programming) ที่ได้ จากนั้นจึงใส่ส่วนประกอบต่างๆเข้าไปภายในหุ่นยนต์ และเรียกขึ้นมาทำงาน โดยการทำงานนั้นจะถูกเรียกอัตโนมัติโดยแพลตฟอร์มของซอร์ฟแวร์ ซึ่งภายในการทำงานจะใช้เทคนิคของโอโอพี ประกอบด้วย เจนเนอรอลไลเซชั่น (Generalization), สเปเชียลไลเซชั่น (Specialization) และ การถ่ายทอดคุณสมบัติ (Inheritance)

ตัวอย่างการนำซอร์ฟแวร์แพลตฟอร์มไปใช้งาน

```
#include .....framework
Class UserImplement.....
.....

int main(){
        MAR_Robot *robot = new MAR_Robot();

        Sensor* s_front = new Ultrasonic(6);
        Sensor* s_left  = new Ultrasonic(7);
        Sensor* s_right = new Ultrasonic(8);
        Sensor* s_direction = new DigitalCompass(5);
        ImageSensor *imagesensor = new WebcamSensor();
        sleep(10);
        robot->robot_addSensor(s_front);
        robot->robot_addSensor(s_left);
        robot->robot_addSensor(s_right);
        robot->robot_addSensor(s_direction);
        robot->robot_addSensor(imagesensor);
        Controlling* drivesystem = new DriveSystem();
        SensorListener* surveylistener = new SurveyListener();
        Behavior*       survey = new Survey();
```

```
        survey->addSensorListener(surveylistener);
        survey->addControlling(drivesystem);

        Navigation* surveyNavigation = new SurveyNavigation();
        survey->addNavigation(surveyNavigation);
        Mapping* surveyMapping = new SurveyMapping();
        survey->addMapping(surveyMapping);

        robot->robot_addBehavior(survey);
        robot->robot_start();
        return 0;
}
```

วิธีการใช้งานซอร์ฟแวร์แพลตฟอร์ม

1.ผู้ใช้งานทำการอิมพลีเมนท์ในไฟล์ mar.cpp ซึ่งเป็นไฟล์หลัก(main) โดยทำการincludeไฟล์เฮดเดอร์ของแพลตฟอร์ม จากนั้นทำการสร้างคลาสลูกที่ถ่ายทอดจากคลาสของแพลตฟอร์ม(หากต้องการอิมพลีเมนท์รายละเอียดการทำงานเพิ่มเติมตามหลักของ OOP) จากนั้นทำการอิมพลีเมนท์ส่วนประกอบของหุ่นยนต์ดังตัวอย่างโค้ดหัวข้อที่ผ่านมา

2.คอมไพล์โปรแกรม โดยรันโปรแกรมเชลล์สคริป ซึ่งจะได้ไฟล์ทำงานที่ชื่อว่า mar

```
# ./Crosscompile.bash
```

3.โหลดโปรแกรมลงสู่บอร์ด

บทที่ 4
การทดลอง ผลการทดลอง

การทดสอบบอร์ด

การทดสอบจะใช้การทดสอบทีละโมดูล ตามตรวจสอบรายการจากการใช้งานจริง ได้ดังนี้

ตารางที่ 4.1 การทดสอบบอร์ด

โมดูล	ผลการทดสอบ
PXA270	ซีพียูสามารถทำงานได้ โดยมีการตอบสนองทาง s พอร์ตอนุกรมว่ามีการทำงาน
Serial	สามารถรับอินพุท และส่งเอาต์พุทได้อย่างถูกต้อง
Ethernet	สามารถทำงานได้โดยตรวจสอบจากการนำเอา รูท ไฟล์ซิสเต็มจากเน็ทเวิร์ค มาใช้ในแพลตฟอร์มได้
GPIO	สามารถส่งเอาต์พุท และรับอินพุทได้
USB	สามารถทำการเชื่อมต่อกับยูเอสบีดีไวซ์ได้
SD card	ยังไม่สามารถทำงานได้เนื่องจากไดร์เวอร์ยังไม่สมบูรณ์
Audio	ยังไม่สามารถทำงานได้เนื่องจากยังไม่มีดีไวซ์ไดร์เวอร์ และแอปพลิเคชั่นที่ใช้ในการทดสอบ
PWM	สามารถส่งสัญญาน PWM ได้สองช่อง
Interrupt	สามารถกำหนดให้GPIO ใช้งานอินเทอร์รัพต์ ได้

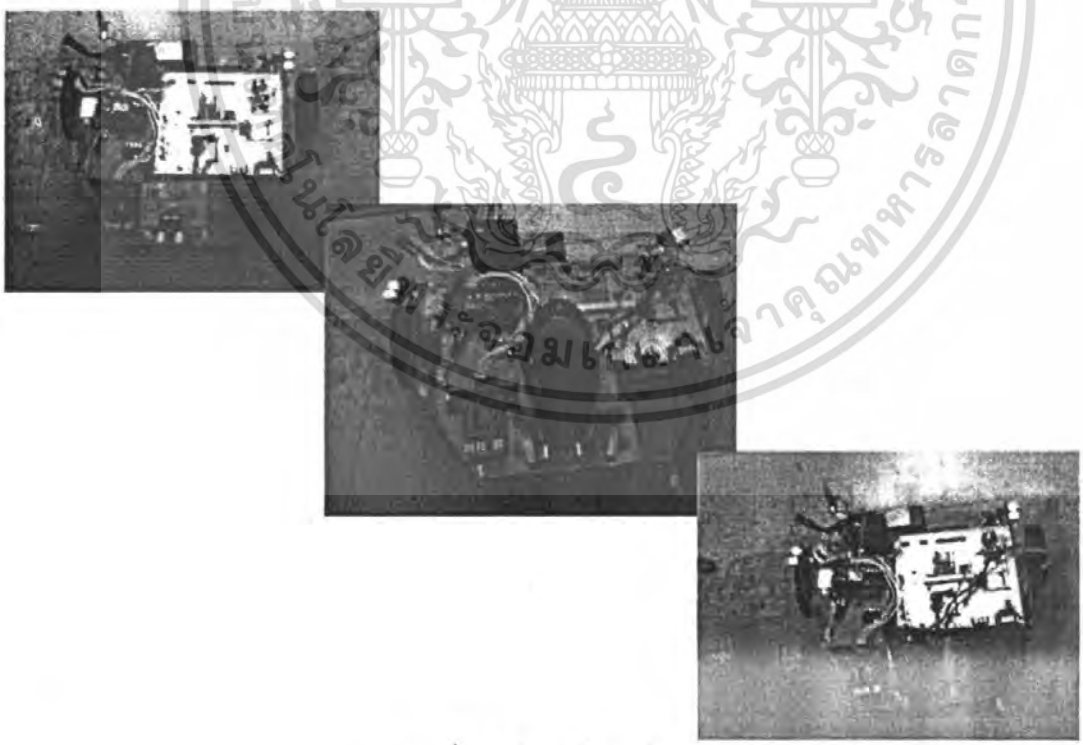
การทดสอบยูเอสบีดีไวซ์

หลังจากที่ได้พัฒนาตัวฮาร์ดแวร์ของยูเอสบีดีไวซ์, ดีไวซ์ไดร์เวอร์และ แอปพลิเคชั่นเสร็จแล้ว ได้ทำการทดสอบยูเอสบีดีไวซ์ตามรายการดังนี้

ตารางที่ 4.2 การทดสอบยูเอสบีดีไวซ์

แพลตฟอร์ม	ผลการทดสอบ
PC ที่ใช้ Windows XP service Pack 2 ทำงาน	สามารถรับส่งข้อมูลผ่านดีไวซ์ไดร์เวอร์และแอปพลิเคชั่น ที่พัฒนาขึ้นได้
PC ที่ใช้ Linux Ubantu 6.06 ทำงาน	สามารถรับส่งข้อมูลผ่านยูเอสบีลิบ และแอปพลิเคชั่น ที่พัฒนาขึ้นได้
Platform Embedded Linux ที่ run Linux 2.4	สามารถรับส่งข้อมูลผ่านดีไวซ์ไดร์เวอร์ และแอปพลิเคชั่นที่พัฒนาขึ้นได้
Platform Embedded Linux ที่ run Linux 2.6	สามารถรับส่งข้อมูลผ่าน ดีไวซ์ไดร์เวอร์ และแอปพลิเคชั่นที่พัฒนาขึ้นได้

การทดสอบแฟลตฟอร์ม



รูปที่ 4.1 หุ่นยนต์ตัวอย่าง

จากรูปเป็นหุ่นยนต์สำรวจอย่างง่ายซึ่งพัฒนาขึ้นมาเพื่อใช้ในการทดสอบแพลตฟอร์มของโครงงาน ว่าสามารถนำไปสร้างเป็นหุ่นยนต์ได้จริง มีความสะดวกจริง และมีประสิทธิภาพจริงหรือไม่ โดยหุ่นยนต์ตัวอย่างนี้มีอุปกรณ์ดังต่อไปนี้

แพลตฟอร์มเอมเบตเดดลินุกซ์โรบอท	1	ตัว
มอเตอร์	2	ตัว
เว็ปแคม	1	ตัว
เซ็นเซอร์วัดระยะทาง	3	ตัว
เซ็นเซอร์วัดทิศทาง	1	ตัว
คอนโทรเลอร์พีไอซี	1	ตัว
อาเอฟ โมดูล	1	ตัว
ยูเอสบีฮับ	1	ตัว
แบตเตอรี่ลิเทียม	1	ก้อน

ลักษณะการทำงานของหุ่นยนต์ตัวอย่าง โดยจะทำการเคลื่อนที่โดยจะค้นหาวัตถุที่มีสีที่สนใจ และมีการหลบหลีกสิ่งกีดขวาง และมีการส่งข้อมูลการเดินทางกลับมายังศูนย์กลางเพื่อแสดงข้อมูลการเคลื่อนที่

รูปที่ 4.2 แผนที่ที่ได้จากการสำรวจของหุ่นยนต์ตัวอย่าง

โดยมีการใช้งานส่วนซอฟแวร์ของแพลตฟอร์มดังต่อไปนี้

ตารางที่ 4.3 การทดสอบแพลตฟอร์ม

ชื่อโมดูล	การนำไปใช้งาน	ผลที่ได้
เว็ปแคม	ดึงภาพจากแพลตฟอร์มไป ประมวลผลหาสีที่สนใจ	ใช้งานแพลตฟอร์มได้
เซ็นเซอร์	วัดระยะทาง 3 ตัว	ใช้งานแพลตฟอร์มได้
เซ็นเซอร์	วัดทิศทาง	ใช้งานแพลตฟอร์มได้
มอเตอร์	ให้หุ่นยนต์เคลื่อนที่	ใช้งานแพลตฟอร์มได้
การสื่อสาร	ส่งข้อมูลพื้นที่ที่สำรวจกลับสู่ ศูนย์กลางเพื่อแสดงการ เคลื่อนที่ผ่านพอร์ตอนุกรม	ใช้งานแพลตฟอร์มได้
การสื่อสาร	ติดต่อกับคอมโทรเลอร์พีไอซี	ใช้งานแพลตฟอร์มได้

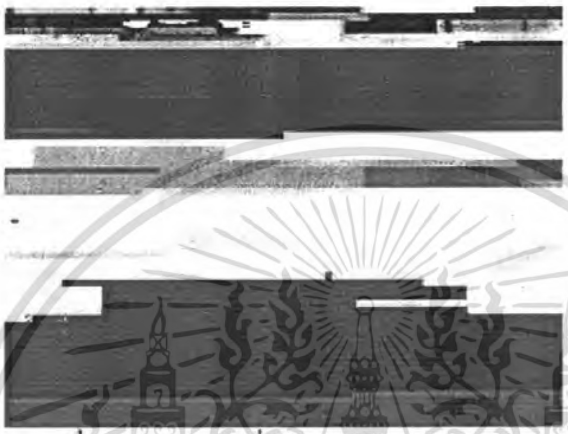
การทดสอบการทำงานหุ่นยนต์ตัวอย่าง

การทดสอบการทำงานของโมดูลเซนเซอร์ : สามารถนำแพลตฟอร์มไปใช้งานได้ ซึ่งแพลตฟอร์มจะทำการติดต่อกับอุปกรณ์เซนเซอร์และรับข้อมูลระยะทางกับมาประมวลผลได้ แต่มีข้อจำกัดทางด้านตัวฮาร์ดแวร์เองซึ่งมีระยะการตรวจสอบได้ไม่ไกลนักและสเกลระยะทางของเซนเซอร์อาจจะไม่ถูกต้องร้อยเปอร์เซ็นต์แต่สามารถยอมรับได้

การทดสอบการทำงานของโมดูลการเคลื่อนที่ : สามารถนำแพลตฟอร์มไปใช้งานได้ โดยหุ่นยนต์สามารถเคลื่อนที่หลบหลีกสิ่งกีดขวางได้ โดยมีส่วนน้อยที่เกินการชนเนื่องจากสเกลการทำงานของการควบคุมมอเตอร์ไม่แม่นยำพอ และมีปัญหาทางด้านแมคคานิกส์ของหุ่นยนต์ ซึ่งทำให้หุ่นยนต์เคลื่อนที่ได้ไม่ตรงทิศทาง

การทดสอบการทำงานการสื่อสารข้อมูล : สามารถนำแพลตฟอร์มไปใช้งานได้ โดยหุ่นยนต์สามารถสื่อสารระหว่างเครื่องพีซีเพื่อทำการส่งข้อมูลพื้นที่ที่ถูกสำรวจกลับไปได้ แต่ยังมีข้อเสียตรงที่เกิดข้อมูลสูญหายเนื่องจากปัญหาของอุปกรณ์ฮาร์ดแวร์เอง

การทดสอบโมดูลกล้องเว็ปแคม : ผลการทำงานของหุ่นยนต์ทดสอบนั้นสามารถทำงานได้ ภายใต้การทำงานในที่ที่มีแสงสว่างไม่มาก มิฉะนั้นจะทำให้หุ่นยนต์ทำงานผิดพลาด โดยหุ่นยนต์ตัวอย่างจะคิดว่าเจอวัตถุสีที่สนใจคือสีแดงทันที



รูปที่ 4.3 แสดงภาพที่ได้จากกล้องขณะแสงมาก

รูปที่ 4.4 แสดงภาพที่ได้จากกล้องขณะแสงพอเหมาะ

จากรูปด้านบนคือภาพแสดงภาพที่ได้จากกล้อง ซึ่งนำไปประมาณผลตรวจสอบสีแดง เมื่อหุ่นยนต์ประมวลผลเสร็จแล้วจะทำการหยุดการทำงานของหุ่นยนต์เพื่อแสดงว่าเสร็จสิ้นภารกิจ

สรุปได้ว่าโมดูลของแพลตฟอร์มนี้สามารถนำไปใช้งานได้จริง ยกเว้นในที่ที่มีสภาวะแสงสว่าง

มาก ซึ่งเป็นปัญหาที่ตัวสาร์ดแวร์กล้องเว็ปแคม

สรุปการทดสอบแพลตฟอร์ม

จากการทดสอบแพลตฟอร์มซึ่งทำการทดสอบโดยการสร้างหุ่นยนต์สำรวจอย่างง่ายขึ้นมา โดยมีการใช้งานส่วนต่างๆของแพลตฟอร์ม

บทที่ 5
บทวิจารณ์ และสรุป

5.1 บทวิจารณ์ และสรุปผล

โครงงานนี้มีความหน้าในระดับที่น่าพอใจ โดยเป็นไปตามจุดประสงค์และเป้าหมายของโครงการที่ได้ว่างแผนไว้ในตอนต้นของโครงการ โดยผลที่ได้คือ ฮาร์ดแวร์แพลตฟอร์มสามารถใช้งานได้ และตัวลินุกซ์และซอพแวร์สามารถทำงานร่วมกับฮาร์ดแวร์ได้อย่างดี แต่ยังมีบางโมดูลที่ยังไม่สามารถทำการทดสอบได้เนื่องจากบางโมดูลไม่มีดีไวซ์ไดร์เวอร์ในการทดสอบและฮาร์ดแวร์ยังมีข้อผิดพลาดอยู่ เฟรมเวิร์คและไลบรารี่ สามารถใช้งานได้อย่างง่ายดาย และสุดท้ายสามารถนำแพลตฟอร์มไปสร้างหุ่นยนต์ได้จริง

5.2 ปัญหาและอุปสรรคในการทำงาน

ปัญหาและอุปสรรคที่พบในการทำโครงงาน ได้แก่

- อุปกรณ์ฮาร์ดแวร์บางอย่างไม่สามารถหาได้ภายในประเทศไทย จะต้องสั่งซื้อจากต่างประเทศ ซึ่งใช้เวลานานในการจัดส่ง จึงทำให้มีผลกระทบต่อตารางเวลาการทำงานของโครงงาน
- การลงอุปกรณ์ทำได้ยาก เพราะอุปกรณ์ส่วนใหญ่มีลักษณะเป็นเซอร์เฟซเมาท์ (surface mount) ซึ่งมีขนาดเล็ก รวมถึงตัวซ็อกเก็ต ซึ่งเป็นซ็อกเก็ต ที่มีระยะห่างระหว่างขาที่เล็กมากทำให้ต้องใช้เวลา และความชำนาญสูง ในการลงอุปกรณ์

5.3 แนวทางการแก้ไข

- ติดตามการจัดซื้อ และจัดส่งอย่างสม่ำเสมอ
- ฝึกฝนการลงอุปกรณ์ การบัดกรีให้มีความชำนาญ แล้วจึงลงมือทำ เพื่อให้ไม่เกิดความสิ้นเปลือง หรืออาจจะทำให้วงจรเสียหายขึ้นมาได้

5.4 แนวทางการพัฒนาต่อ

- ทำการพัฒนาดีไวซ์ไดร์เวอร์เพิ่มเติม เช่น เอดีโอ

- ทำการรวบรวมไลบรารี่ที่สนับสนุนการทำงานของหุ่นยนต์เพิ่มเติม
- นำเฟรมเวิร์คไปพัฒนาโมดูลต่างๆให้มีความหลากหลายมากขึ้น
- นำแพลตฟอร์มไปใช้สร้างหุ่นยนต์ที่มีประสิทธิภาพ

บรรณานุกรม

Raghavan, P. Lad, Amol. and Neelakandan, Sriram. 1997. **EMBEDDED LINUX SYSTEM DESIGN AND DEVELOPMENT.**

Hallinan, Christopher. **Embedded Linux Primer : A Practical Real-World Approach.**

Newman, *D.jay. 2000.* ***Linux Robotics.***

Corbet, Jonathan. Rubini, Alessandro. And Kroah-Hartman, Greg. 2002. **Linux Device Drivers.** Third Edition.

How to write a Makefile. [online] Available: http://www.hsrl.rutgers.edu/ug/make_help.html

The DENX U-Boot and Linux Guide (DULG) for TQM8xxL. [online] Available: http://www.denx.de/wiki/bin/view/DULG/Manual

ลภน สุภาพ, อรรถพล บุญยะโภคา, วรพจน์ กรแก้ววัฒนกุล และ ชัยวัฒน์ ลิ้มพรจิตรวิไล. Simple PC-USB prot Interface Experiment text-lab manual.

PXA270 manual1. [online] Available : www.intel.com/design/pca/applicationsprocessors/datashts/280000.htm

PXA270 manual2. [online] Available : www.intel.com/design/pca/applicationsprocessors/manuals/280001.htm

PXA270 manual3. [online] Available : www.intel.com/design/pca/applicationsprocessors/manuals/280002.htm