ไมโครคอนโทรลเลอร์ แอพพลิเคชัน

MICROCONTROLLER APPLICATION

โดย

นายศุภพล กูลวงษ์วาณิชย์

นายสรวิทย์ เงินยอดรัก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

MICROCONTROLLER APPLICATION

BY

MR. SUPAPOL KULAVONGVANIT

MR. SORAWIT NGURNYOTRAK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENT FOR THE DEGREE OF

BACHELOR IN DEPARTMENT OF INFORMATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2006

หัวข้อโครงงาน	ไมโครคอนโทรลเลอร์ แอพพลิเคชัน	
นักศึกษา	นายศุภพล กุลวงษ์วาณิชย์	รหัสนักศึกษา 46010792
	นายสรวิทย์ เงินยอดรัก	รหัสนักศึกษา 46010809
อาจารย์ที่ปรึกษา	รศ.ดร.อรรถสิทธิ์ หล่าสกุล	
ระดับการศึกษา	ปริญญาตรี วิศวกรรมศาสตรบัณฑิต	
	สาขาวิศวกรรมสารสนเทศ	
ภาควิชา	วิศวกรรมสารสนเทศ	
ปีการศึกษา	2549	

ปริญญานิพนธ์นี้ได้รับความเห็นชอบจากอาจารย์ที่ปรึกษาเป็นที่เรียบร้อยแล้ว

.....

(รศ.ดร.อรรถสิทธิ์ หล่าสกุล)

อาจารย์ที่ปรึกษา

หัวข้อปริญญานิพนธ์	ไมโครคอนโทรลเลอร์ แอพพลิเคชัน	
ชื่อนักศึกษา	นายศุภพล กุลวงษ์วาณิชย์	รหัสประจำตัว 46010792
	นายสรวิทย์ เงินยอดรัก	รหัสประจำตัว 46010809
อาจารย์ที่ปรึกษา	รศ.ดร.อรรถสิทธิ์ หล่าสกุล	
ระดับการศึกษา	ปริญญาตรี วิศวกรรมศาสตรบัณฑิต	
	สาขาวิศวกรรมสารสนเทศ	
ภาควิชา	วิศวกรรมสารสนเทศ	
ปีการศึกษา	2549	

บทคัดย่อ

ปัจจุบัน การตรวจวัดสัญญาณและจัดเก็บสัญญาณต่าง ๆ ที่ติดต่อกันเป็นระยะเวลานาน เพื่อนำไปวิเคราะห์ต่อไปด้วยคอมพิวเตอร์นั้น นับวันจะมีการนำไปใช้งานกันมากขึ้น โดยเฉพาะระบบที่มีขนาดเล็ก สามารถนำติดตัวไปที่ต่าง ๆ ได้สะดวกและราคาถูก ยิ่งมีความต้องการมากขึ้นไปอีก อุปกรณ์ตัวไมโครคอนโทรเลอร์นั้น ถ้าออกแบบให้เหมาะสมก็จะได้คุณลักษณะตามที่กล่าวมาดังข้างต้นและสามารถนำมาใช้งานในส่วนนี้ได้เป็นอย่างดี

งานปริญญานิพนธ์ ฉบับนี้จึงนำเสนอ บอร์ดระบบฝังตัวที่สามารถใช้เก็บข้อมูลได้เป็นจำนวนมาก โดยบอร์ดสามารถนำไปประยุกต์ใช้งานได้มากมาย เช่น วัดอุณหภูมิ, ทำระบบติดตามจากสัญญาณ GPS, เครื่องเล่น MP3 และอุปกรณ์อื่น ๆ อีกเป็นจำนวนมาก โดยระบบที่ทำขึ้นนี้จะมีส่วนเก็บข้อมูลขนาดใหญ่เนื่องจากใช้หน่วยความจำแบบ MMC/SD Card ในขณะที่ระบบเองมีส่วนประกอบอุปกรณ์จำนวนน้อย จึงมีขนาดที่เล็ก ส่วนในการติดต่อย้ายข้อมูลกับคอมพิวเตอร์ก็สามารถทำได้ง่ายเนื่องจากใช้การสื่อสารแบบอนุกรม และใช้ตัวของ MMC/SD Card เอง และในส่วนของราคารวมของระบบที่ทำขึ้นมานี้(ไม่รวมส่วนเซนเซอร์) มีราคาที่ถูกมากเมื่อเทียบกับระบบที่มีอยู่ในท้องตลาด ในส่วนของตัวอย่างการนำไปใช้งานจะได้มีแสดงไว้ในบทท้ายๆต่อไป

Thesis Title MICROCONTROLLER APPLICATION

Student Mr. Supapol Kulavongvanit ID. 46010792

Mr. Sorawit Ngurnyotrak ID. 46010809

Advisor Asst. Prof. Dr Attasit Lasakul

Graduate Level Bachelor Degree of Information Engineering

Department Information Engineering

Academic Year 2006

ABSTRACT

Currently, monitoring of signal and storing for future analysis (Data acquisition system) become very necessary system to wide range of applications. Especially, the system that has the advantage of low cost, small size and moveable system become more useful. To overcome above requirements, well designed of Micro-controller system is the answer.

The purpose of this project is about implementation the embedded board which is used to collect large amounts of sensor data. The board can be configured and programmed for a wide range of data acquisition tasks. For example, temperature monitor, GPS tracking system, animal or object movement (via any kind of sensors), MP3 player etc. The purpose embedded board has large mounts of storage unit (MMC/SD card) where as consist of a few components and small size of board. Data transferring to PC is very easy by RS-232 and MMC/SD Card. Furthermore, total cost of the system is less then (much) to others on the market. Some of applications of using this board will be presented in append chapter.

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้อาจไม่สำเร็จได้ด้วยดีหากไม่ได้รับความช่วยเหลือและร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลสำคัญที่ต้องกล่าวเป็นบุคคลแรกที่มีส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้สำเร็จลงได้ คือ อาจารย์อรรถสิทธิ์ หล่าสกุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลืออย่างต่อเนื่อง ซึ่งต้องขอขอบพระคุณเป็นอย่างสูง

และต้องขอขอบพระคุณบุคคลที่สำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้คือ บิดา มารดา อันเป็นที่เคารพรักยิ่งซึ่งได้เลี้ยงดูพร้อมทั้งให้โอกาสทางการศึกษาอย่างเต็มที่ อีกทั้งยังให้กำลังใจและความเอาใจใส่ในทุกๆด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

ศุภพล กุลวงษ์วาณิชย์
สรวิทย์ เงินยอดรัก

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาของปัญหาและแนวคิดเริ่มต้นในการทำโครงงาน

เนื่องจากรูปแบบข้อมูลในปัจจุบันที่นำมาใช้ในการศึกษาค้นคว้านั้น จำเป็นที่จะต้องใช้ช่วงเวลาเก็บข้อมูลที่มีระยะเวลานาน เพื่อนำมาศึกษาและวิเคราะห์ในการหาแนวโน้มต่างๆที่น่าจะเกิดขึ้นได้ในอนาคต และเมื่อใช้เวลาเก็บข้อมูลเป็นระยะเวลานานก็จะทำให้ข้อมูลที่เก็บนั้นมีขนาดใหญ่มากขึ้นตามเวลาที่ใช้ไป การที่จะเก็บข้อมูลลักษณะนี้ให้เกิดความสะดวกในการเก็บนั้น มีจำเป็นที่จะเก็บใส่เอ็มเอ็มซีเอสดีการ์ด (MMC/SD Card) ซึ่งมีขนาดเล็กโดยผ่านบอร์ดไมโครคอนโทรลเลอร์เอ็มซีเอส 51 (MCS-51) มีผลให้ใช้พลังงานในการทำงานไม่มาก อีกทั้งมีพื้นที่ที่ใช้ในการเก็บข้อมูลได้เป็นจำนวนมาก

ส่วนสาเหตุที่เลือกใช้เอสดีการ์ดนั้น เพราะ การที่จะใช้ ฮาร์ดดิสก์ ในการเก็บข้อมูลจำเป็นที่จะต้องให้พลังงานเป็นจำนวนมากเพื่อให้หัวอ่านข้อมูลมีการทำงานตลอดเวลาที่ใช้ในการเก็บข้อมูล ซึ่งจะมีผลทำให้มีระยะเวลาในการทำงานที่สั้นลง อีกทั้งการที่จะต้องเก็บข้อมูลในที่ๆห่างไกลไม่สามารถที่จะไปดูแลเครื่องได้ตลอดเวลา การที่ใช้ฮาร์ดดิสก์ในการเก็บข้อมูลจะเกิดความไม่สะดวกขึ้นเนื่องจากลักษณะเครื่องมีขนาดใหญ่ ไม่เหมาะที่จะปล่อยทิ้งไว้เป็นระยะเวลานานได้ จึงเกิดเป็นแนวความคิดในการทำโครงการนี้ขึ้นมา

1.2 วัตถุประสงค์ของโครงงาน

- เพื่อความสะดวกในการใช้งาน เพราะเอ็มเอ็มซีเอสดีการ์ดเป็นอุปกรณ์ที่มีขนาดเล็ก
- ตัวระบบที่ใช้งานมีราคาที่ถูกกว่าตามท้องตลาดที่มีการซื้อขาย
- สามารถนำตัวระบบไปประยุกต์ใช้งานในด้านต่างๆได้หลากหลาย
- สามารถเก็บข้อมูลในท้องที่ที่อยู่ห่างไกลได้ เช่น ที่ในป่า หรือ ที่ขั้วโลก
- สร้างอุปกรณ์ต้นแบบเพื่อใช้ในการพัฒนาต่อไปได้
- เก็บข้อมูลได้เป็นระยะเวลานาน และต่อเนื่องกันได้
- ขนย้ายอุปกรณ์ได้สะดวก

1.3 ขอบเขตของโครงงาน

1. ศึกษาหน่วยความจำแบบเอสดีการ์ดว่ามีลักษณะการเก็บข้อมูลอย่างไร เก็บในรูปแบบไหน

2. ศึกษาตัวไมโครคอนโทรลเลอร์เอ็มซีเอส51ว่ามีลักษณะการทำงานเป็นอย่างไรแล้วจะเชื่อมต่อกับตัวเอสดีการ์ดได้อย่างไร

3. ทำการทดสอบโปรแกรมการเชื่อมต่อที่ใช้ในการเชื่อมต่อระหว่างตัวเอสดีการ์ด กับระบบไมโครคอนโทรลเลอร์

4. ศึกษาและทำการแก้ไขปัญหาที่เกิดขึ้นในระบบที่ทำการเชื่อมต่อ

1.4 ขั้นตอนการดำเนินงานของโครงงาน

1 เริ่มจากขั้นตอนการศึกษาข้อมูลเกี่ยวกับเอสดีการ์ดและ ไมโครคอนโทรลเลอร์เอ็มซีเอส 51

2 ทดลองใช้งานตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51 เอซี 3

- ทดลองการใช้งานติดต่อกับ LCD เพื่อแสดงผล
- ทดลองเขียนโปรแกรมติดต่อสื่อสารแบบเอสพีไอ (SPI)

3 ขั้นตอนการทำฮาร์ดแวร์ (Hardware)

4 ขั้นตอนการเขียนโปรแกรมควบคุมเพื่อให้สามารถอ่านเขียนตัวเอสดีการ์ดและนำค่าที่เก็บในเอสดีการ์ดไปแสดงผลได้

5 จัดทำรายงานสรุปผล

ตารางที่ 1.1 แผนการดำเนินงาน

แผนการดำเนินงาน	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.
1. ศึกษาค้นคว้าข้อมูลเอสดีการ์ด								
2. ศึกษาและทดลองตัวไมโครคอนโทรลเลอร์								
3. สร้างฮาร์ดแวร์								
4. เขียนโปรแกรมอ่านเขียนและแสดงผลตัวเอสดีการ์ด								
5. รายงานสรุปผล								

บทที่ 2
ทฤษฎีที่ใช้ในโครงงาน

2.1 หน่วยความจำแบบเอสดีการ์ด (SD Memory Card)

หน่วยความจำแบบเอสดีการ์ด คือ การ์ดหน่วยความจำซึ่งถูกออกแบบมาโดยฉพาะในเรื่องของความปลอดภัย , ความจุของข้อมูล , ประสิทธิภาพและสภาพแวดล้อมรอบข้าง ที่จำเป็นที่มีมาแต่ต้นที่ได้ปรากฏออกมาในเร็วๆนี้

หน่วยความจำแบบเอสดีการ์ดจะรวมการป้องกันทางด้านกลไกที่มีความสำคัญ ซึ่งจะทำตามมาตรฐานความปลอดภัยของเอสดีเอ็มไอ (SDMI) และจะมีความเร็วในการส่งข้อมูลสูงสำหรับการส่งข้อมูลที่มีความจุมากๆ ระบบรักษาความปลอดภัย หน่วยความจำแบบเอสดีการ์ดจะใช้ความเชื่อใจกันทั้งสองฝ่ายและใช้ นิว ไซเฟอร์ อัลกอริทึม (new cipher algorithm) ในการป้องกันจากการใช้การ์ดที่ไม่เหมาะสม

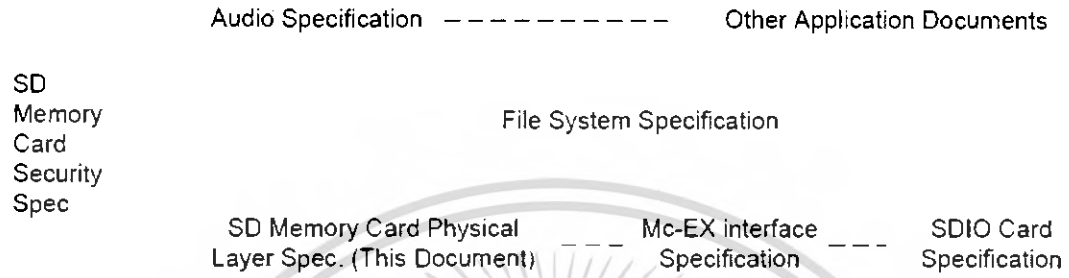
หน่วยความจำแบบเอสดีการ์ดจะมีการสนับสนุนระบบรักษาความปลอดภัย 2 แบบบนมาตรฐานทั่วไปที่มีการใช้งานเช่นไอเอสโอ(ISO)-7816 ซึ่งสามารถถูกใช้ในการติดต่อหน่วยความจำแบบเอสดีการ์ดในเครือข่ายสาธารณะและระบบอื่นๆที่สนับสนุนธุรกิจโทรศัพท์มือถือและอุปกรณ์ทางดิจิตอล

ส่วนประกอบที่เพิ่มขึ้นมาของหน่วยความจำแบบเอสดีการ์ดคือ เอสดีไอโอการ์ด (SDIO Card) จะถูกกำหนดชื่อเป็นพิเศษคือ "เอสดีไอโอการ์ดเฉพาะ" (" SDIO Card Specification ") ซึ่งได้รับมาจากองค์กรเอสดี เอสดีไอโอการ์ดเฉพาะจะกำหนดเอสดีการ์ดที่อาจจะเกี่ยวข้องกับการติดต่อระหว่างหน่วยไอโอ (IO) ที่ต่างกันและเอสดีโฮสท์ (SD Host) เอสดีไอโอการ์ดอาจจะเกี่ยวกับความสามารถในการเก็บหน่วยความจำเช่นเดียวกับหน้าที่ของไอโอ. สัดส่วนของหน่วยความจำของเอสดีไอโอการ์ดจะสามารถเข้ากันได้เต็มที่โดยให้ หน่วยความจำแบบเอสดีการ์ดเป็นพิเศษเอสดีไอโอการ์ดจะสามารถเข้ากันได้กับหน่วยความจำแบบเอสดีการ์ด การเข้ากันได้นี้จะรวมไปถึงทางด้านสภาพภายนอก , วงจรอิเล็คทรอนิคส์ , พลังงาน , สัญญาณ และ ซอฟท์แวร์ (Software) จุดประสงค์ของเอสดีไอโอการ์ดคือจัดให้ความเร็วในการส่งข้อมูลทางด้านไอโอมีค่าสูงและใช้พลังงานที่น้อยสำหรับอุปกรณ์โทรศัพท์มือถือ เป้าหมายหลักคือใส่ค่าไอโอการ์ดในนอนเอสดีไอโอ (non-SDIO) ที่รู้จักกับโฮสท์จะทำให้ไม่ได้รับความเสียหายทางกายภาพหรือเกิดการแตกออกของอุปกรณ์หรือซอฟท์แวร์ในกรณีนี้ ไอโอการ์ดจะถูกละเลยได้ง่ายๆ

การติดต่อสื่อสารของหน่วยความจำแบบเอสดีการ์ดจะอยู่บนพื้นฐานของการติดต่อแบบ 9-ขา (pin) (1 ฐานเวลา (Clock) , 1 ชุดคำสั่ง (Command) , 4 ข้อมูล (Data) และ 3 สายพลังงาน (

Power lines)) ได้ถูกออกแบบให้ทำงานที่ความถี่สูงสุดที่ 50 MHz และมีพลังงานต่ำที่ช่วง 5 โวลต์ โปรโตคอลการติดต่อสื่อสารจะถูกกำหนดให้เป็นส่วนหนึ่งของส่วนนี้การติดต่อของหน่วยความจำแบบเอสดีการ์ดจะสนับสนุนการทำงานของมัลติมีเดียการ์ด (MultiMedia Card)

หน่วยความจำแบบเอสดีการ์ดจะถูกแบ่งออกเป็นหลายส่วนโดยมีโครงสร้างตามในรูปที่ 2.1



รูปที่ 2.1 โครงสร้างของหน่วยความจำแบบเอสดีการ์ด

หน่วยความจำแบบเอสดีการ์ดเฉพาะเสียง (Audio Specification)

ใช้ในงานทางด้านเสียงเป็นส่วนที่สำคัญที่ขาดไม่ได้

หน่วยความจำแบบเอสดีการ์ดเฉพาะระบบไฟล์ (File System Specification)

มีการระบุถึงโครงสร้างรูปแบบไฟล์ของข้อมูลที่มีการบันทึกลงในหน่วยความจำแบบเอสดีการ์ด (ในพื้นที่ที่มีการป้องกัน และ ที่ไม่มีการป้องกัน)

หน่วยความจำแบบเอสดีการ์ดเฉพาะความปลอดภัย (Security Specification)

มีการป้องกันระบบกลไกที่มีความสำคัญและมีคำสั่งพิเศษที่จะสนับสนุนการป้องกันนี้

หน่วยความจำแบบเอสดีการ์ดเฉพาะชั้นกายภาพ (Physical Layer Specification)

มีการระบุถึงการติดต่อทางกายภาพ(physical) และโปรโตคอลชุดคำสั่งที่ใช้โดยหน่วยความจำแบบเอสดีการ์ด จุดประสงค์ของหน่วยความจำแบบเอสดีการ์ดชั้นกายภาพคือการกำหนดสภาพแวดล้อมและจัดการควบคุม

แมคเอ็กซ์ทางการติดต่อโดยเฉพาะทาง (Mc-EX Interface Specification)

ส่วน AI ของ หน่วยความจำแบบเอสดีการ์ดจะทำหน้าที่ขยายไปที่เอสดีการ์ดชั้นกายภาพและจัดหาสิ่งที่ต้องการในการส่งแพ็กเก็จชุดคำสั่งส่วนขยายทางการค้ามือถือ (Mobile Commerce Extension (Mc-EX)) จากแมคเอ็กซ์โฮสท์ ไปยังแมคเอ็กซ์ที่ได้รับสิทธิ์

2.2 ระบบของหน่วยความจำแบบเอสดีการ์ด

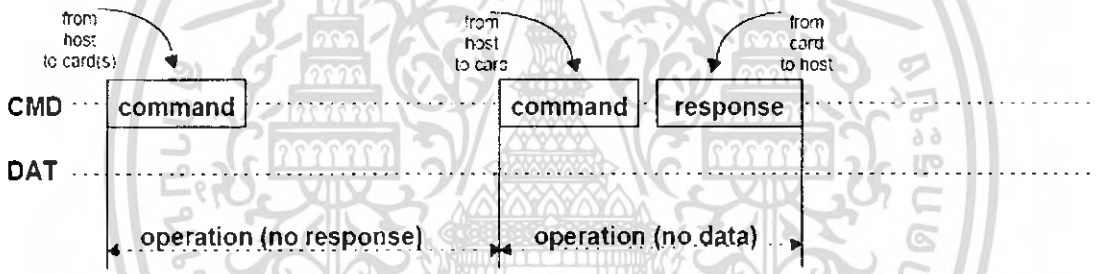
2.2.1 เส้นทางเอสดี (SD bus)

การติดต่อสื่อสารบนเอสดีบัสจะใช้งานอยู่บนพื้นฐานของคำสั่งและกระแสของบิตข้อมูลซึ่งจะเริ่มโดยบิตสตาร์ท (Start) และจะหยุดโดยบิดเอนด์ (End)

คอมมานด์ (Command) : คือชุดคำสั่งที่ให้เริ่มการดำเนินงานคอมมานด์โดยจะส่งจากโฮสท์ไปที่การ์ดเดี่ยว (addressed command) หรือไปที่การ์ดที่มีการเชื่อมต่อทั้งหมด (broadcast command) คำสั่งคอมมานด์นี้จะถูกส่งอยู่บนเส้นทางซีเอ็มดี (CMD)

เรสพ้อนซ์ (Response) : คือคำสั่งที่ส่งจากการ์ดเดี่ยวหรือจากการ์ดที่มีการเชื่อมต่อทั้งหมดส่งไปยังโฮสท์โดยการส่ง เรสพ้อนซ์นี้จะเป็นการตอบต่อคำสั่งคอมมานด์ที่ได้รับก่อนหน้านี้คำสั่งเรสพ้อนซ์จะถูกส่งอยู่บนเส้นทางซีเอ็มดี

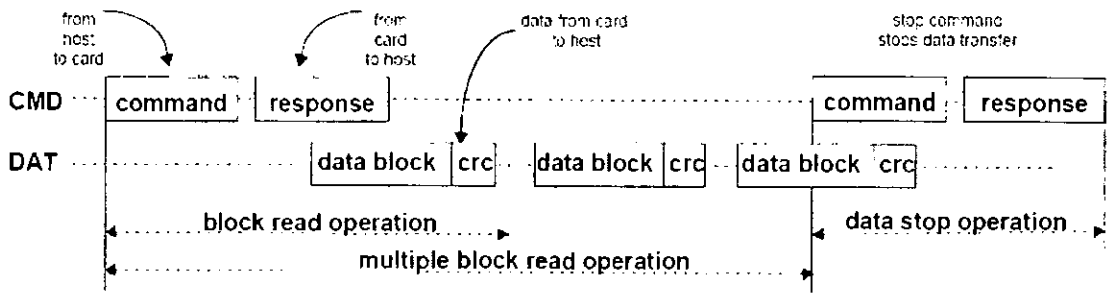
ดาต้า (Data) : ข้อมูลจะถูกส่งจากการ์ดไปยังโฮสท์โดยจะส่งผ่านเส้นทางแดท (DAT)



รูปที่ 2.2 การทำงานที่ไม่มีเรสพ้อนซ์และไม่มีดาต้า

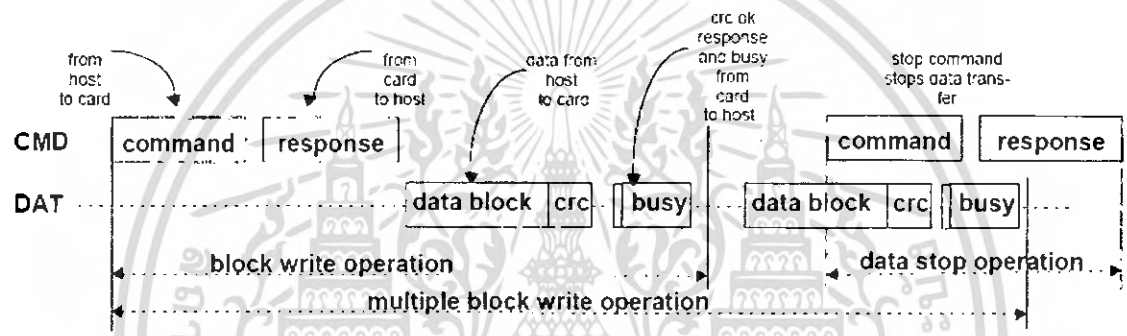
การ์ดแอดเดรสซิ่ง (Card addressing) เป็นเครื่องมือที่ใช้เซสชั่นแอดเดรส (Session address) ไปกำหนดที่การ์ดขณะที่อยู่ในช่วงเริ่มต้น พื้นฐานการจัดการบนเอสดีบัส คือการจัดการคอมมานด์ / เรสพ้อนซ์ บัสชนิดนี้จะจัดการส่งข้อมูลข่าวสารโดยตรงในโครงสร้างของคอมมานด์หรือเรสพ้อนซ์ ในการทำงานแบบอื่นอาจจะมีคำสั่งดาต้าเพิ่มเข้ามาได้

การส่งข้อมูลเข้าไปหรือออกมาจากหน่วยความจำแบบเอสดีการ์ดจะถูกทำอยู่ในบล็อกดาต้า บล็อกดาต้านี้จะถูกทำให้สำเร็จโดยบิตซีอาร์ซี (CRC) ไม่ว่าจะบล็อกเดียวหรือหลายๆบล็อก การทำงานในรูปแบบหลายๆบล็อกจะดีกว่าในเรื่องของความเร็วที่เร็วกว่าในการเขียนข้อมูล การส่งข้อมูลแบบหลายๆบล็อกจะถูกทำให้สิ้นสุดลง เมื่อมีคำสั่งหยุดส่งมาบนเส้นทางซีเอ็มดีการส่งข้อมูลสามารถกำหนดหรือปรับแต่งโดยโฮสท์ที่ใช้ในเส้นทางแดท



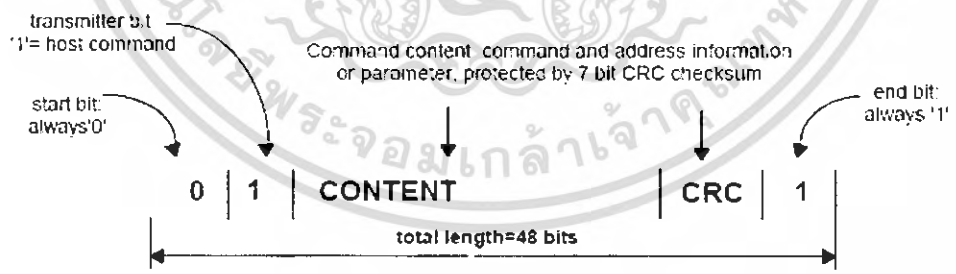
รูปที่ 2.3 รูปหลายๆบล็อกการทำงานแบบการอ่าน

บล็อกที่ใช้เขียนข้อมูลจะใช้ช่วงเวลาบนเส้นทางข้อมูลแดท0 ที่มีสัญญาณของการทำงานแบบการเขียนในการทำงาน (ดูรูปที่ 7) จำนวนของเส้นทางดาต้าจะสัมพันธ์กับการส่งข้อมูล



รูปที่ 2.4 รูปหลายๆบล็อกการทำงานแบบการเขียน

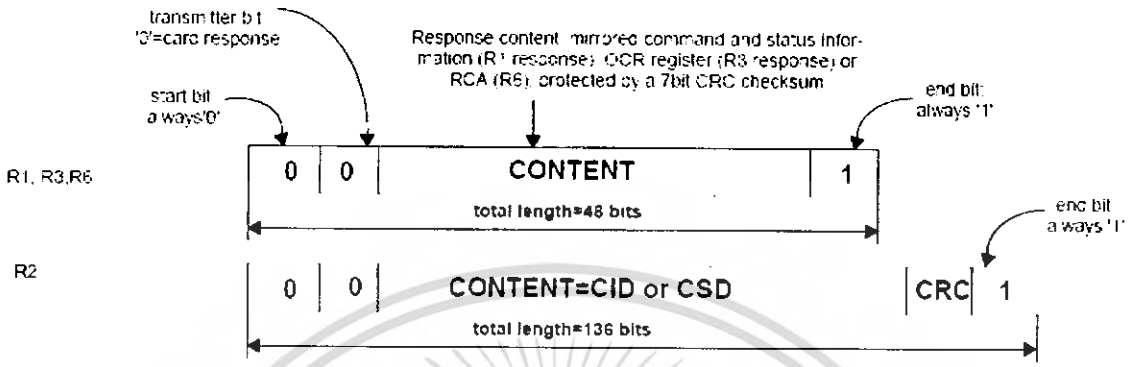
คำสั่งคอมมานด์จะมีเฟรม (Frame) คอมมานด์ คือ



รูปที่ 2.5 รูปแบบเฟรมของคอมมานด์

แต่ละคำสั่งคอมมานด์จะถูกนำหน้าโดยบิตสตาร์ท ('0') และถูกตามหลังโดยบิตเอนด์ ('1') มีความยาวทั้งหมด 48 บิตแต่ละคำสั่งจะถูกป้องกันโดยบิตซีอาร์ซีดังนั้นการส่งข้อมูลที่เกิดการผิดพลาดสามารถที่จะตรวจจับความผิดพลาดได้และข้อมูลก็จะถูกส่งใหม่อีกครั้ง

รูปแบบเรสพ้อนซ์จะมีเฟรมเรสพ้อนซ์อยู่ 4 รูปแบบขึ้นอยู่กับความสำคัญ ความยาวทั้งหมดจะมีอยู่ 48 บิตหรือ 136 บิตอย่างใดอย่างหนึ่ง อัลกอริทึมการป้องกันของซีอาร์ซีสำหรับบล็อกข้อมูล คือ 16 บิตซีซีไอทีที (CCITT)



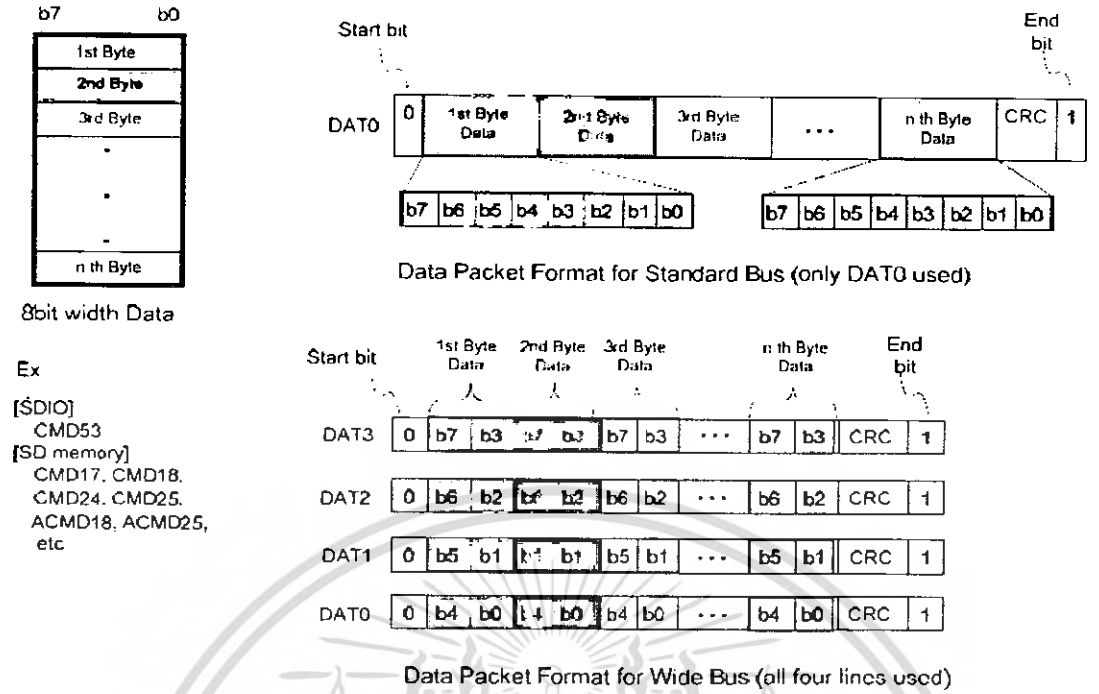
รูปที่ 2.6 รูปแบบเฟรมของเรสพ้อนซ์

ในเส้นทางซีเอ็มดีนั้นบิตเอ็มเอสบี (MSB) จะถูกส่งมาเป็นอันดับแรก และ บิตแอลเอสบี (LSB) จะถูกส่งมาเป็นอันดับสุดท้าย

เมื่อมีการใช้งานตัวเลือกเส้นทางขนาดกว้าง (wide bus) ข้อมูลจะถูกส่งทีละ 4 บิตต่อ 1 ครั้ง (อ้างอิงรูปที่ 8) บิตสตาร์ทและบิตเอนด์จะมีรูปแบบเหมือนกับบิตซีอาร์ซีคือ จะส่งสำหรับทุกๆ เส้นทางแดท บิตซีอาร์ซีจะถูกคำนวณและถูกเช็คสำหรับทุกๆเส้นทางแดท การบอกสถานะการตอบสนองและการทำงานของซีอาร์ซีจะถูกส่งโดยการ์ดไปยังโฮสท์บนแดท 0 เท่านั้น

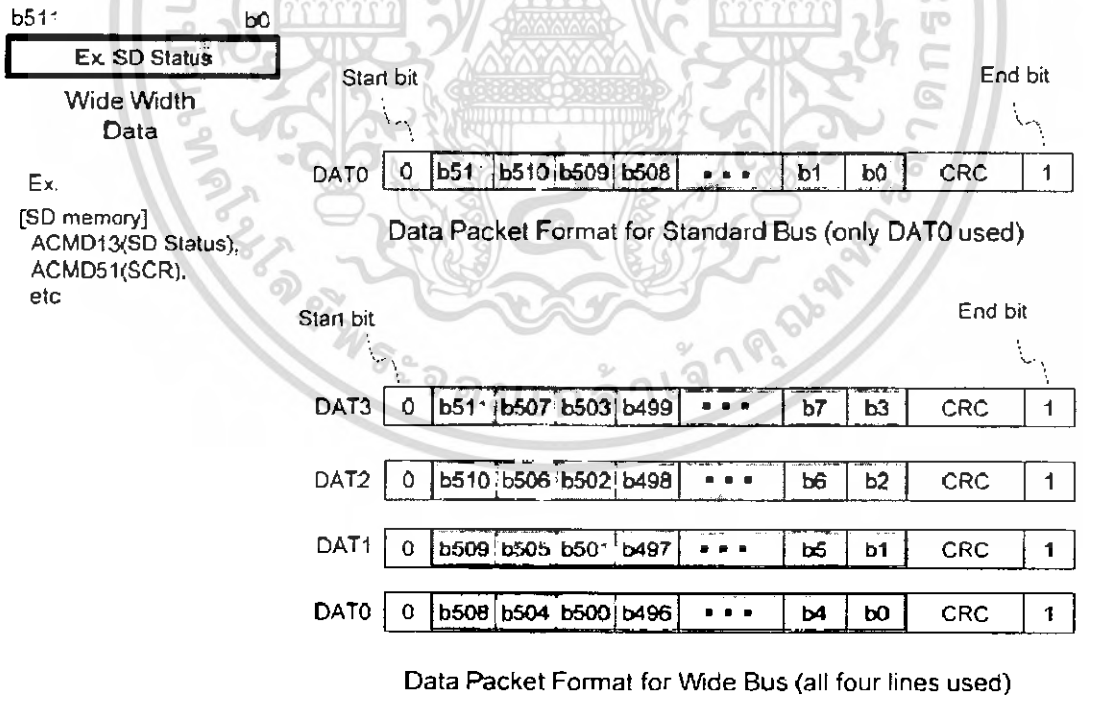
รูปแบบดาต้าแพ็กเก็ท (Data packet) 2 ชนิดของเอสดีการ์ด คือ

(1) ข้อมูลทั่วไป (กว้าง 8 บิต) จะส่ง แอลเอสบี (LSB (Least Significant Byte)) เป็นอันดับแรกและส่ง เอ็มเอสบี (MSB (Most Significant Byte)) เป็นอันดับสุดท้าย



รูปที่ 2.7 รูปแบบแพ็กเก็ทข้อมูลแบบข้อมูลธรรมดา

(2) ข้อมูลขนาดกว้าง (SD Memory Register) จะถูกเลื่อน (Shift) จากบิดเอ็มเอสบี



รูปที่ 2.8 รูปแบบแพ็กเก็ทข้อมูลแบบข้อมูลขนาดกว้าง

2.2.2 เส้นทางเอสพีไอ (SPI Bus)

ขณะที่ช่องสัญญาณเอสดีถูกใช้งานบนพื้นฐานของคำสั่งและกระแสของบิตคาด้าซึ่งถูกเริ่มต้นโดยบิดสตาร์ทและถูกหยุดโดยบิดเอนด์นั้น ช่องสัญญาณเอสพีไอ คือ ไบท์ที่มีการปรับแต่งทุกๆ คำสั่งหรือข้อมูลจะถูกสร้างแบบ 8-บิต ไบท์ และจะถูกรวมเข้ากับสัญญาณชิฟซีเล็ก (Chip Select) เช่น มีความยาว 8 สัญญาณนาฬิกาหลายๆรอบ เหมือนกับโปรโตคอลเอสดี ในคำสั่งของเอสพีไอจะประกอบไปด้วยบล็อกคอมมานด์ , เรสพ้อนซ์ และ คาด้า การติดต่อสื่อสารทั้งหมดระหว่างโฮสท์และการ์ดจะถูกควบคุมโดยโฮสท์มาสเตอร์ (host master) ซึ่งโฮสท์จะเริ่มติดต่อกับทุกๆบัส โดยมีการรักษาสัญญาณขา ชิฟซีเล็กให้มีค่าเป็น '0' ไว้

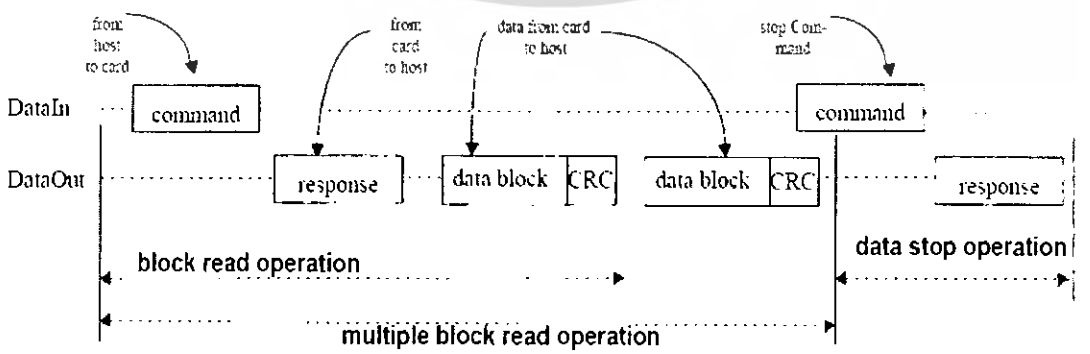
ลักษณะหน้าที่ของเรสพ้อนซ์ในโหมดของเอสพีไอแตกต่างจากโหมดของเอสดี 3 อย่างคือ

- เลือกการ์ดที่ใช้ส่งเรสพ้อนซ์ไปที่คอมมานด์ได้
- ใช้โครงสร้างเรสพ้อนซ์แบบใหม่ 2 ชนิด (8 บิต และ 16 บิต)
- เมื่อการ์ดเผชิญกับปัญหาการกู้ข้อมูลให้คืนสู่สภาพเดิมจะมีการตอบสนองกับ เรสพ้อนซ์ ที่มีการผิดพลาด (ที่เกิดในส่วนของบล็อกดาด้า) โดยใช้การหยุดพักชั่วคราวแทน

สิ่งที่เพิ่มขึ้นมา คือ ในคอมมานด์ , เรสพ้อนซ์ , ทุกๆบล็อกดาด้าจะถูกส่งไปที่การ์ด ในระหว่างที่มีการเขียนจะถูกตอบสนองกับรูปแบบข้อมูลเรสพ้อนซ์พิเศษ

2.2.2.1 อ่านข้อมูล (Data Read)

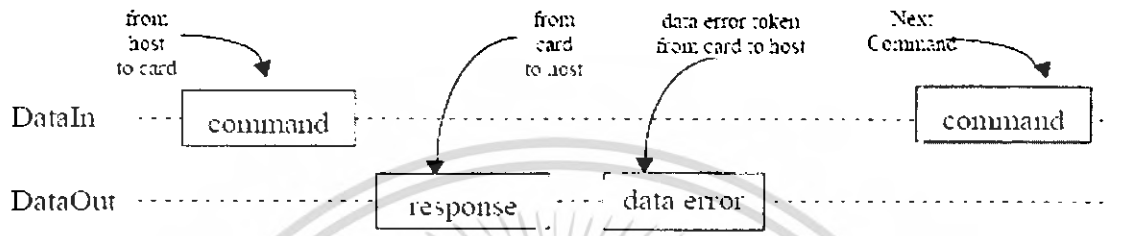
บล็อกต่างๆจะอ่านคอมมานด์ที่ถูกสนับสนุนในโหมดเอสพีไอซึ่งเป็นการกระทำที่เหมาะสมกับมาตรฐานของเอสพีไอ โดยเฉพาะ 2 สัญญาณที่ถูกใช้ (แบบทิศทางเดียว (Unidirectional)) ในเวลาที่มีการรับเข้าของการอ่านคอมมานด์ที่การ์ดจะตอบสนองกับรูปแบบเรสพ้อนซ์ที่ตามมาโดยความยาวของรูปแบบข้อมูลจะถูกกำหนดในคำสั่ง SET_BLOCKLEN (ซีเอ็มดี 16) ก่อนหน้านี้ การยุติการทำงานการอ่านข้อมูลนั้นจะเหมือนกับโปรโตคอลเอสดีคือ มีการส่งคำสั่ง STOP_TRANSMISSION



รูปที่ 2.9 การทำงานแบบการอ่านข้อมูล

ส่วนท้ายของบล็อกข้อมูลคือ 16 บิต ซีอาร์ซี ที่ถูกสร้างโดยมาตรฐานซีซีไอทีที่มีส่วนประกอบคือสมการ $x^{16} + x^{12} + x^5 + 1$

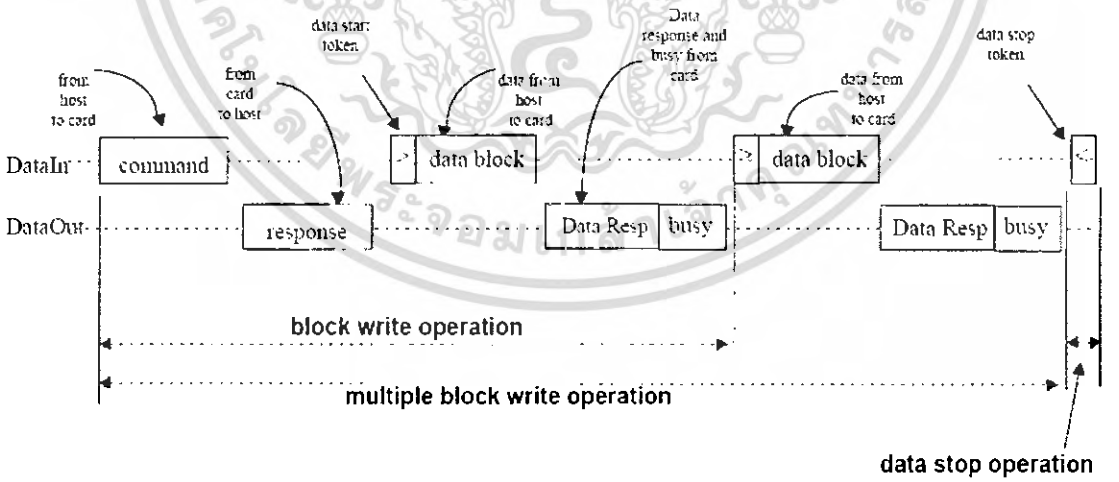
ในกรณีของการกู้คืนดาต้าที่มีความผิดพลาดการ์ดจะไม่ส่งข้อมูลใดๆแทนที่ของข้อมูลที่มีความผิดพลาดพิเศษรูปแบบจะถูกส่งไปที่โฮสท์ รูปที่ 10 จะแสดงการทำงานการอ่านข้อมูลซึ่งถูกยุติลงกับการที่มีความผิดพลาดของบล็อกดาต้า



รูปที่ 2.10 การทำงานแบบการอ่านข้อมูลที่มีความผิดพลาด

2.2.2.2 เขียนข้อมูล (Data write)

บล็อกต่างๆที่ทำการเขียนข้อมูลจะถูกสนับสนุนในการทำงานแบบเอสพีไอในเวลาที่มีการรับคำสั่งการเขียน การ์ดจะตอบสนองกับเรสพ้อนซ์และจะรอบล็อกดาต้าที่ถูกส่งมาจากโฮสท์ ส่วนท้ายของซีอาร์ซี, ความยาวบล็อก และการกำหนดตำแหน่งที่เริ่มจะเหมือนกับการทำงานแบบการอ่าน (ดูรูปที่ 11)

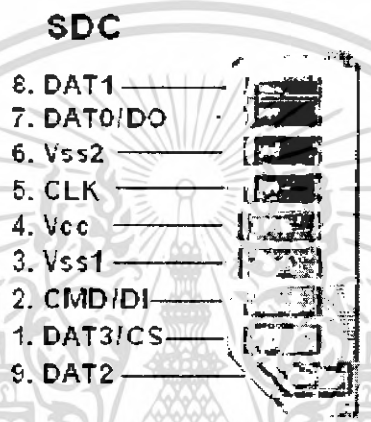


รูปที่ 2.11 การทำงานแบบการเขียนข้อมูล

หลังจากบล็อกดาต้าถูกรับแล้วการ์ดจะตอบสนองกับรูปแบบ ดาต้า-เรสพ้อนซ์ ถ้าบล็อกดาต้าถูกรับโดยไม่มีความผิดพลาดเกิดขึ้น มันจะถูกโปรแกรมตลอดตราบเท่าที่การ์ดยังคงทำงานอยู่ บล็อกบิซี่ (busy) ที่ยังไหลอยู่อย่างต่อเนื่องก็จะถูกส่งไปที่โฮสท์

2.3 โครงสร้างของระบบหน่วยความจำแบบเอสดีการ์ด

2.3.1 การทำงานของหน่วยความจำแบบเอสดีการ์ด



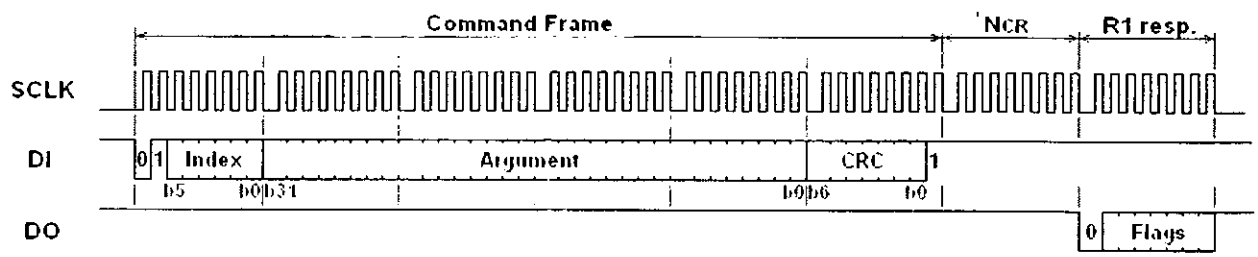
รูปที่ 2.12 ส่วนการติดต่อของเอสดีการ์ด

การทำงานของเอสดีการ์ดในโหมดเอสพีไอ

การทำงานในโหมดเอสดีไอข้อมูลจะถูกส่งเป็นไบท์แบบอนุกรม โดยที่แต่ละคำสั่งที่ใช้สื่อสารในโหมดเอสพีไอ จะมีความยาว 6 ไบท์ เมื่อคำสั่งส่งถึงตัวการ์ดจะมีการส่งข้อมูลตอบกลับมา

ช่วงเวลาการส่งและการตอบสนองต่อคำสั่ง

คำสั่งแต่ละคำสั่งจะใช้เวลาในการส่งทั้งหมด 48 สัญญาณนาฬิกาหลังจากส่งคำสั่งต้องส่ง 1 รอจนกว่าจะมีการตอบสนองกลับมา (ช่วงเวลา * เอ็นซีอาร์ (NCR)) โดยบิตแรกของการตอบสนองจะเป็น 1 ดังรูปที่ 13



รูปที่ 2.13 สัญญาณฐานเวลาของการติดต่อแบบเอสพีไอ

คำสั่ง (Commands)

ชนิดของคำสั่ง (Command Type)

เอสดีการ์ดจะมีคำสั่งทั้งหมด 4 ชนิด ที่ใช้ในการควบคุม

- คำสั่งที่เป็นแบบบรอดคาสท์ (broadcast) ไม่มีการตอบสนอง (bc)
- คำสั่งแบบบรอดคาสท์ที่มีการตอบสนอง (bcr)
- คำสั่งบอกตำแหน่งแอดเดรสที่ไม่มีการส่งข้อมูลไปด้วย
- คำสั่งบอกแอดเดรสที่มีการส่งข้อมูลไปด้วย

รูปแบบของคำสั่ง

คำสั่งแต่ละคำสั่งที่ใช้ติดต่อกับเอสดีการ์ดจะมีขนาด 6 ไบท์ แต่ละคำสั่งจะเริ่มต้นด้วยบิตเริ่มต้น โดยจะมีค่าเป็น '0' และตามด้วยบิตการส่งแสดงทิศทางการส่งจะมีค่าเป็น '1' เพราะส่งจากโฮสท์ อีก 6 บิตต่อมาจะเป็นดัชนีของแต่ละคำสั่งซึ่งคำสั่งทั้งหมดจะมีได้ 64 คำสั่ง(ซีเอ็มดี 0 - ซีเอ็มดี 63) อีก 3 ไบท์ต่อมาจะเป็นอากิวเมนต์ (Argument) และมีบิตที่ใช้ในการตรวจสอบ 7 บิต ตามด้วยบิตปิดท้าย 1 บิต ซึ่งจะมีค่าเป็น '1'

ตารางที่ 2.1 ตารางรายละเอียดแต่ละบิตของคำสั่ง

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

คำสั่งคอมมานด์ที่ใช้งานในการติดต่อกับเอสดีการ์ดในโหมดเอสพีไอทั้งหมดแสดงดังตารางที่ 2

ตารางที่ 2.2 ตารางคำสั่งทั้งหมดที่มีสามารถใช้งานได้ในโหมดเอสพีไอ

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets the SD Card
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No				
CMD3	No				
CMD4	No				
CMD5	Reserved				
CMD6	Reserved				
CMD7	No				
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its card-specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its card identification (CID).
CMD11	No				
CMD12	Yes	None	R1b	STOP _TRANSMISSION	Forces the card to stop transmission during a multiple block read operation
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	No				
CMD15	No				
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).
CMD17	Yes	[31:0] data address	R1	READ_SINGLE _BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE _BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_ TRANSMISSION command
CMD19	Reserved				
CMD20	No				
CMD21 CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop transmission token is sent (instead of start block).

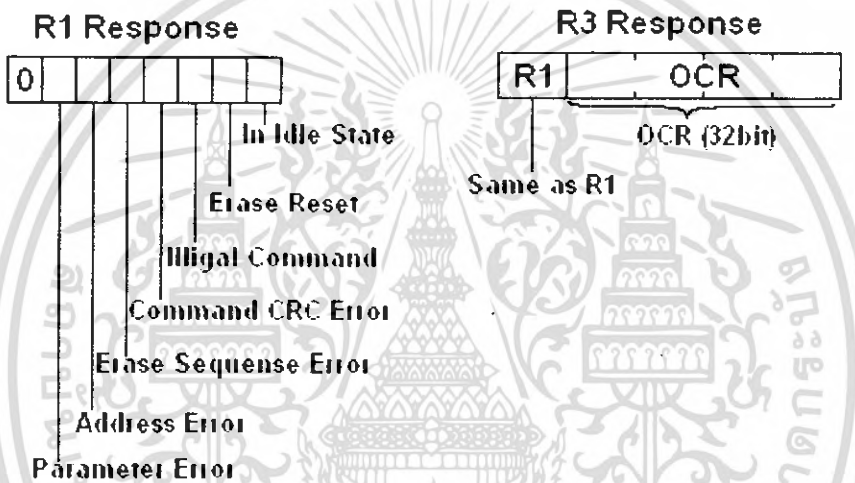
ตารางที่ 2.2 (ต่อ) ตารางคำสั่งทั้งหมดที่มีสามารถใช้งานได้ในโหมดเอสพีไอ

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD26	No				
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28*	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WP_GRP_SIZE).
CMD29*	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	ERASE_WR_BLK_START_ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WR_BLK_END_ADDR	Sets the address of the last write block of a continuous range to be erased.
CMD34 ... CMD37	Reserved				
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.
CMD39	No				
CMD40	No				
CMD41 CMD54	Reserved				
CMD55	Yes	[31:0] stuff bits	R1	APP_CMD	Notifies the card that the next command is an application specific command rather than a standard command.
CMD56	Yes	[31:0] stuff bits [0]: RD/WR	R1	GEN_CMD	Used either to transfer a Data Block to the card or to get a Data Block from the card for general purpose/application specific commands. The size of the Data Block is defined with SET_BLOCK_LEN command.
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR register of a card.
CMD59	Yes	[31:1] don't care* [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on, a '0' will turn it off.
CMD60-63	Not				

การตอบสนอง (Response)

การทำงานในโหมดเอสพีไอจะมีคำสั่งเรสพ้อนซ์อยู่ 3 รูปแบบ คือ อาร์ 1 (R1), อาร์ 2 (R2) และอาร์ 3 (R3) ขึ้นอยู่กับแต่ละคำสั่งว่าเป็นอย่างไร โดยจะมีความยาว 1 ไบท์ คำสั่งส่วนใหญ่จะตอบสนองด้วยอาร์ 1 โดยแต่ละบิตของอาร์1 จะมีความหมายดังรูปที่ 14 ถ้าส่ง 0x00 กลับมาแสดงว่าทำได้สำเร็จ แต่ถ้าเกิดการผิดพลาดขึ้นบิตที่แสดงการผิดพลาดนั้นก็จะมีค่าเป็นหนึ่ง

การตอบสนองอาร์ 3 จะใช้ในการตอบสนองต่อคำสั่งที่ใช้ในการอ่านค่าในริจิสเตอร์โออาร์ซี (ORC) (CMD58) จะเป็น 4 ไบท์ โดยจะมีไบท์แรกเหมือนกับอาร์1 และจะมีส่วนท้ายอีก 3 ไบท์ (32 บิต) คำสั่งแต่ละคำสั่งจะมีระยะเวลาในการตอบสนองประมาณ 8-80 สัญญาณนาฬิกาหรืออาจจะนานกว่า

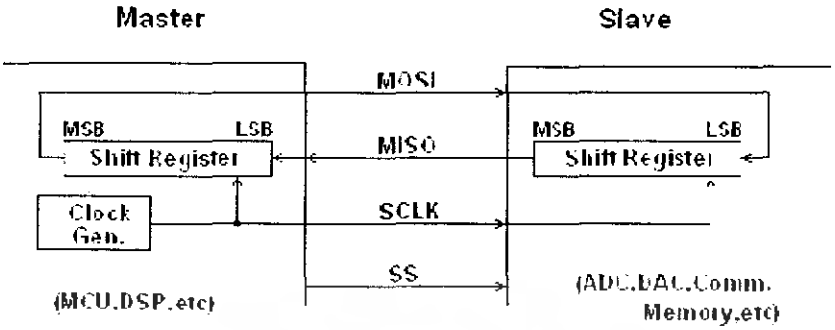


รูปที่ 2.14 รูปแบบเรสพ้อนซ์ของอาร์ 1 และอาร์ 3

2.3.2 การทำงานของหน่วยความจำแบบเอสพีไอ

เป็นโปรโตคอลของการสื่อสารอนุกรมแบบสองทิศทางโดยจะใช้การซิงโครนัส (Synchronous) กันระหว่างเอ็มซียู (MCU) และอุปกรณ์ที่ตัวอยู่ภายนอก รวมไปถึงการติดต่อระหว่างเอ็มซียูตัวอื่นด้วย

โครงสร้างการสื่อสารแบบเอสพีไอ

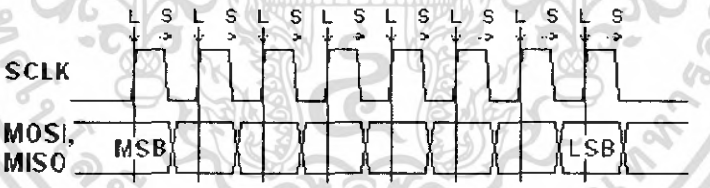


รูปที่ 2.15 โครงสร้างพื้นฐานของการสื่อสารแบบเอสพีไอ

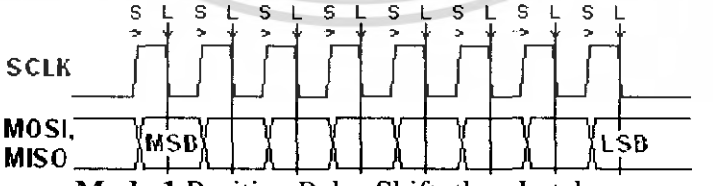
โครงสร้างพื้นฐานของการสื่อสารแบบเอสพีไอจะประกอบด้วยสายสัญญาณ 3 เส้น คือ สายส่งข้อมูล (MOSI), สายรับข้อมูล (MISO) และสายสัญญาณนาฬิกา (SCLK) โดยที่ตัวควบคุม (Master) สามารถต่อกับอุปกรณ์ภายนอกหรือเอ็มซียูได้หลายตัวและใช้สัญญาณจากขาเลือกสัญญาณเอาท์พุท (ขา SS) ในการเลือกสัญญาณเอาท์พุทที่เข้ามาทางสายรับข้อมูล (MISO)

เวลาการส่งข้อมูลของเอสพีไอ (SPI Transfer Timing)

การสื่อสารโดยใช้โปรโตคอลแบบเอสพีไอจะมีอยู่ทั้งหมด 4 โหมดการทำงานโดยแต่ละโหมดการทำงานจะมีการแลช (Latch) และการเลื่อนในช่วงเวลาที่ต่างกันดังรูปที่ 16

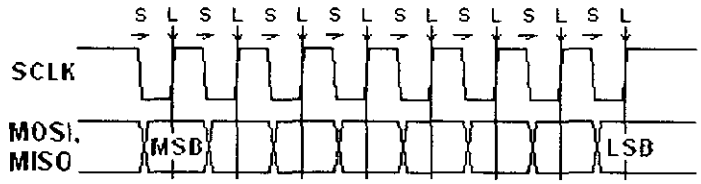


Mode 0 Positive Pulse. Latch, then Shift.

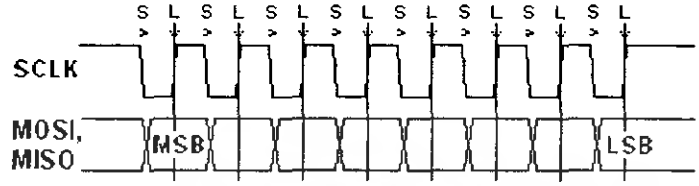


Mode 1 Positive Pulse Shift, then Latch

รูปที่ 2.16 โหมดการทำงานของเอสพีไอ



Mode 2 Negative Pulse. Latch, then Shift

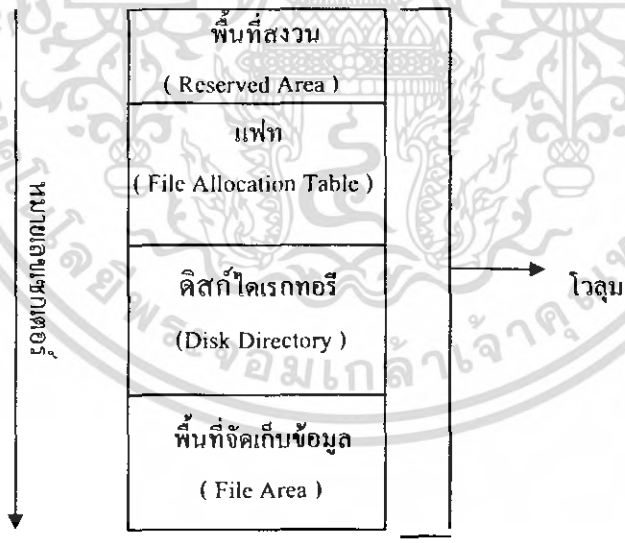


Mode 3 Negative Pulse. Shift, then Latch.

รูปที่ 2.16 (ต่อ) โหมดการทำงานของเอสพีไอ

2.4 ตารางจัดสรรไฟล์ (File Allocation Table)

พื้นที่ส่วนตารางจัดสรรไฟล์ไม่ได้เก็บข้อมูลของไฟล์ต่างๆเอาไว้โดยตรง แต่ทำหน้าที่เป็นตารางที่ทำหน้าที่จัดสรรไฟล์เท่านั้นโดยจะอยู่ถัดจากพื้นที่สงวนดังรูปที่ 2.17



รูปที่ 2.17 ส่วนประกอบหลักๆของโวลุม

โวลุม (Volume) คือ โครงสร้างของดิสก์ หรือ ก็คือไดรฟ์ ซึ่งโวลุมจะถูกแบ่งออกเป็น เซกเตอร์ (Sector) เรียงลำดับกันไปอย่างต่อเนื่อง แต่ละเซกเตอร์จะสามารถบรรจุข้อมูลขนาดคงที่ (โดยทั่วไปเท่ากับ 512 ไบต์) และจะถูกกำกับด้วยหมายเลขโดยจะเริ่มต้นจากเซกเตอร์ 0

2.4.1 พื้นที่สงวน (Reserved Area)

พื้นที่นี้โดยทั่วไปจะมีความยาวเพียงเซกเตอร์เดียวคือ เซกเตอร์แรก (เซกเตอร์หมายเลข 0) ของโวลุม แต่ถ้าใช้มากกว่า 1 เซกเตอร์ก็ต้องเริ่มจากเซกเตอร์ 0 เสมอ เซกเตอร์ 0 ก็คือ บู๊ตเซกเตอร์ (Boot Sector) โดยบู๊ตเซกเตอร์จะเก็บข้อมูลเกี่ยวกับโวลุม รวมถึงรูทีนในการบู๊ตระบบ (Boot Routine หรือ Bootstrap Routine) ไว้

ตารางที่ 2.3 ตารางรายละเอียดของบู๊ตเซกเตอร์

แอดเดรส	รายละเอียด	ขนาด
00h	เป็นคำสั่งให้กระโดดไปยังรูทีนที่ใช้บู๊ตระบบ	3 ไบต์
03h	ชื่อผู้ผลิตและหมายเลขเวอร์ชัน	8 ไบต์
0Bh	จำนวนไบต์ต่อเซกเตอร์	1 เวิร์ด
0Dh	จำนวนเซกเตอร์ต่อคลัสเตอร์	1 ไบต์
0Eh	จำนวนเซกเตอร์ของพื้นที่สงวน	1 เวิร์ด
10h	จำนวน แฟท	1 ไบต์
11h	จำนวนเอ็นทรีของรูทไดเรกทอรี	1 เวิร์ด
13h	จำนวนเซกเตอร์ในโวลุม	1 เวิร์ด
15h	ตัวบ่งชี้ประเภทของสื่อ (Media Descriptor)	1 ไบต์
16h	จำนวนเซกเตอร์ของ แฟท	1 เวิร์ด
18h	จำนวนเซกเตอร์ของแทร็ก	1 เวิร์ด
1Ah	จำนวนหัวอ่าน/เขียน	1 เวิร์ด
1Ch	จำนวนเซกเตอร์ที่ซ่อนไว้	1 เวิร์ด
1Eh-1FFh	รูทีนที่ใช้บู๊ตระบบ (Boot Routine)	

เซกเตอร์ 0 ถูกเรียกว่าบู๊ตเซกเตอร์ เนื่องจากถ้าเป็นโวลุมที่มีสิทธิ์บู๊ตระบบได้ จะเริ่มบู๊ตจากเซกเตอร์นี้โดยปฏิบัติตามคำสั่งที่อยู่ ณ แอดเดรส 00h ของบู๊ตเซกเตอร์ ซึ่งแอดเดรส 00h ได้บรรจุคำสั่งกระโดดไปดำเนินงานยังแอดเดรสที่กำหนด แอดเดรสปลายทางที่ถูกกำหนดอยู่ในส่วนท้ายของบู๊ดเซกเตอร์ช่วง 1Eh ถึง 1FFh (Boot Routine) คำสั่งกระโดดเมื่อเป็นภาษาเครื่องมีขนาด 1 ไบต์เท่านั้น บวกกับแอดเดรสที่ต้องการกระโดดไปอีก 2 ไบต์ก็เท่ากับ 3 ไบต์ที่ระบุในตาราง

ถัดจากแอดเดรส 00h จะประกอบด้วยฟิลด์ข้อมูลต่างๆ อาทิเช่น ชื่อผู้ผลิต ซึ่งทำการฟอร์แมดโวลุมนี้ , จำนวนไบต์ต่อเซกเตอร์ , จำนวนเซกเตอร์ต่อคลัสเตอร์ ฯลฯ จำนวนเซกเตอร์ของพื้นที่สงวนสามารถดูได้จากแอดเดรส 0Eh ของบู๊ดเซกเตอร์ หากมีค่าเป็น 1 หมายความว่าพื้นที่สงวนใช้เนื้อที่ 1 เซกเตอร์ ซึ่งก็คือตัวบู๊ตเซกเตอร์นั่นเอง

2.4.2 พื้นที่แฟท (FAT Area)

แฟท (FAT) ในที่นี้ย่อมาจาก File Allocation Table หรือ ตารางจัดสรรไฟล์ ข้อมูลในแฟทสามารถบ่งบอกสิ่งต่างๆได้ เช่น เซกเตอร์ใดบ้างที่ยังไม่มีไฟล์จับจองเป็นเจ้าของ , เซกเตอร์ใดบ้างที่ได้รับความเสียหายทางกายภาพ (Bad Sector)

บริเวณในโวลุมที่ถูกกำหนดให้เป็นแฟทจะอยู่ถัดจากพื้นที่สงวน หากพื้นที่สงวนใช้เซกเตอร์แรก (เซกเตอร์หมายเลข 0) เพียงเซกเตอร์เดียวเพื่อเป็นบู๊ตเซกเตอร์แฟท พื้นที่แฟทก็จะเริ่มตั้งแต่เซกเตอร์หมายเลข 1 เรื่อยไป พื้นที่แฟทไม่ได้เก็บข้อมูลของไฟล์ต่างๆเอาไว้โดยตรง แต่ทำหน้าที่จัดสรรไฟล์เท่านั้น แฟทประกอบขึ้นจากข้อมูลขนาดคงที่ซึ่งเรียกว่า เอ็นทรี (Entry) 1 เอ็นทรีในแฟท สัมพันธ์กับพื้นที่จัดเก็บข้อมูลได้มากกว่า 1 เซกเตอร์ โดยเซกเตอร์เหล่านั้นต้องต่อเนื่องกันและมีจำนวนเป็นเลขยกกำลังของ 2 กลุ่มของเซกเตอร์ที่สัมพันธ์กับเอ็นทรีในแฟท เรียกว่า คลัสเตอร์ (Cluster) หรืออาจจะกล่าวได้อีกนัยหนึ่งว่า แต่ละเอ็นทรีในแฟทจะสัมพันธ์กับ 1 คลัสเตอร์ ซึ่งอาจประกอบด้วยพื้นที่จำนวน 1, 2, 4, ... เซกเตอร์ จำนวนเซกเตอร์ต่อคลัสเตอร์จะบันทึกข้อมูลในบริเวณออฟเซตที่ 0Dh ของบู๊ตเซกเตอร์ ขนาดของคลัสเตอร์แบ่งตามขนาดของไดร์ฟ และชนิดของ แฟท ที่ใช้แสดงได้ดังนี้

ตารางที่ 2.4 ตารางจำนวนเซกเตอร์และขนาดคลัสเตอร์ตามขนาดของไดรฟ์และแฟท

ขนาดของไดร์ฟ	แฟทต่อคลัสเตอร์	จำนวนเซกเตอร์	ขนาดคลัสเตอร์
0 MB to 1.9 MB	12-บิต	1	512 ไบท์
2 MB to 3.9 MB	12-บิต	2	1 KB
4 MB to 7.9 MB	12-บิต	4	2 KB
8 MB to 15.9 MB	12-บิต	8	4 KB
16 MB to 31 MB	16-บิต	1	512 ไบท์
32 MB to 63 MB	16-บิต	2	1 KB
64 MB to 127 MB	16-บิต	4	2 KB
128 MB to 255 MB	16-บิต	8	4 KB
256 MB to 511 MB	16-บิต	16	8 KB
512 MB to 1023 MB	16-บิต	32	16 KB
1024 MB to 2047 MB	16-บิต	64	32 KB
2048 MB to 4095 MB	16-บิต	128	64 KB
4096 MB to 8191 MB	16-บิต	256	128 KB
8192 MB to 16384 MB	16-บิต	512	256 KB
512 MB to 8191 MB	32-บิต	8	4 KB

ตารางที่ 2.4 (ต่อ) ตารางจำนวนเซกเตอร์และขนาดคลัสเตอร์ตามขนาดของไดรฟ์และแฟท

ขนาดของไดร์ฟ	แฟทต่อคลัสเตอร์	จำนวนเซกเตอร์	ขนาดคลัสเตอร์
8192 MB to 16383 MB	32-บิต	16	8 KB
16384 MB to 32767 MB	32-บิต	32	16 KB
32768 MB or larger	32-บิต	64	32 KB

จะเห็นได้ว่าถ้าขนาดของไดร์ฟที่มากกว่า 4 GB จะต้องใช้ขนาดของคลัสเตอร์ 128K หรือ 256K (แฟท 16 บิต) และขนาดของเซกเตอร์จะต้องมากกว่า 512 ไบท์ดังนั้นจึงมีการเพิ่มขนาดของ แฟท เป็น 32 บิตซึ่งจะมีผมดีทำให้ขนาดของคลัสเตอร์เล็กลง ทำให้การจัดสรรพื้นที่ในดิสก์มีประสิทธิภาพมากขึ้น

โครงสร้างของแฟท

แฟทมีโครงสร้างเป็นแบบลิงค์ลิสต์ (Linked List) ทำหน้าที่เป็นตัวชี้บอกว่าข้อมูลในไฟล์ต่างๆ บนดิสก์นั้นจัดเก็บอยู่ที่ใดบ้างแฟท เป็นส่วนสำคัญมากจึงมีการสำรองไว้อีกชุดหนึ่ง ดังนั้นในดิสก์ 1 แผ่นจึงมีแฟท อยู่ 2 ชุด กรณีที่ชุดหนึ่งเสียหายก็ยังมีแฟทอีกชุดหนึ่งสำรองไว้ใช้แทนได้ แฟทจะเป็นตัวชี้คลัสเตอร์ขนาด 12 บิต สำหรับแผ่นดิสก์ และขนาด 16 บิตหรือ 32 บิต สำหรับฮาร์ดดิสก์ รายละเอียดของค่าต่างๆใน แฟท เป็นดังนี้

ตารางที่ 2.5 ตารางรายละเอียดของฮาร์ดดิสก์

แฟท 12 บิต	แฟท 16 บิต	ความหมาย
000h	0000h	คลัสเตอร์ว่าง ใช้งานได้
FF0h – FF6h	FFF0h – FFF6h	คลัสเตอร์สงวน
FF7h	FFF7h	คลัสเตอร์ใช้การไม่ได้
FF8h - FFFh	FFF8h - FFFFh	คลัสเตอร์สุดท้ายของไฟล์
xxxh (ค่าอื่นๆ)	xxxxh (ค่าอื่นๆ)	คลัสเตอร์ถัดไปที่เก็บข้อมูล

เนื้อที่ในแฟทจะถูกแบ่งเป็นส่วนๆตามขนาดของแฟทถ้าเป็นแฟท 12 บิต ขนาดของแต่ละส่วนที่ถูกแบ่งก็จะเป็นขนาด 12 บิต ซึ่งจะใช้ทำหน้าที่เป็นลิงค์ลิสต์เพื่อชี้ตำแหน่งคลัสเตอร์ต่อไปของไฟล์ข้อมูลที่ถูกเก็บต่อจากคลัสเตอร์ปัจจุบัน โดยที่ตำแหน่งไบต์แรกของแฟทจะแสดงรายละเอียดชนิดของดิสก์ (media descriptor) ในไบต์ที่ 2 , 3 สำหรับ แฟท 12 บิต จะมีค่าเป็น FF กรณีเป็นแฟท 16 บิตจะมีค่า FF ในไบต์ที่ 4 ด้วย แต่ถ้าเป็นแฟท 32 บิดจะมีค่า FF ไปจนถึงไบต์ที่ 8 ซึ่งทั้งหมดนี้จะตรงกับส่วนของคลัสเตอร์หมายเลข 0 และ 1

ดังนั้นในการเก็บไฟล์ข้อมูลจะเริ่มเก็บในคลัสเตอร์ที่ 2 เป็นต้นไปและตรงส่วนของคลัสเตอร์หมายเลย 2 ในแฟทก็จะทำหน้าที่เป็นลิงค์เพื่อชี้ไฟล์ข้อมูลที่ถูกเก็บต่อจากคลัสเตอร์หมายเลข 2 เช่น ถ้าเป็นเลข 3 แสดงว่าคลัสเตอร์ที่ 3 เก็บข้อมูลถัดจากหมายเลข 2 แต่ถ้าเป็น FFF (แฟท 12 บิต) แสดงว่าเป็นคลัสเตอร์สุดท้ายของไฟล์ไม่มีคลัสเตอร์ที่ต่อจากนี้แล้ว ตัวอย่างสมมุติไฟล์ข้อมูลที่ถูกเก็บจะมีชื่ออยู่ในไดเรกทอรีมีคลัสเตอร์แรกของไฟล์เริ่มต้นที่หมายเลข 2 ส่วนคลัสเตอร์ที่เหลือจะต้องมาดูใน แฟท ดังนั้นจะมีคลัสเตอร์ทั้งหมดที่เก็บคือ 2 , 3 , 6 , 7 ดังแสดงในรูป 2.18

หมายเลข Cluster	0	1	2	3	4	5	6	7	8
ค่าที่เก็บ			003	006			007	FFF	

รูปที่ 2.18 ไฟล์ข้อมูลที่มีการเชื่อมโยงแบบลิงค์ลิสต์

2.4.3 ดิสก์ไดเรกทอรี (Disk Directory)

ดิสก์ไดเรกทอรี เป็นส่วนที่เก็บรายละเอียดต่างๆของไฟล์ที่บันทึกในดิสก์ ภายใต้ระบบปฏิบัติการจะมีไดเรกทอรีอยู่ 2 แบบคือ

1. รูทไดเรกทอรี (Root Directory)
2. ไดเรกทอรีย่อย (Subdirectory)

2.4.3.1 รูทไดเรกทอรี

เนื่องจากลักษณะของแฟท เป็นโครงสร้างข้อมูลประเภทที่เรียกกันว่าลิงค์ลิสต์ ซึ่งเอ็นทรีในแฟท ที่ได้รับการจัดสรรให้กับไฟล์หนึ่งๆจะเชื่อมโยงกันเพื่อให้สามารถติดตามข้อมูลในไฟล์ได้ครบถ้วน โดยข้อมูลของไฟล์จะเก็บอยู่ในคลัสเตอร์ที่สัมพันธ์กับเอ็นทรีนี้ ยกตัวอย่างเช่นรูปที่ 2.18 ใช้เอ็นทรี 2 , 3 , 6 และ 7 ข้อมูลในเอ็นทรีหมายเลข 2 จะบอกให้รู้เอ็นทรีถัดไปของไฟล์ข้อมูลนี้ว่าอยู่ที่เอ็นทรีหมายเลข 3 โดยการชี้ลำดับข้อมูลจะสิ้นสุดที่เอ็นทรีหมายเลข 7 เนื่องจากเก็บค่า FFFh ไว้ซึ่งมีความหมายคือเป็นเอ็นทรีสุดท้ายของไฟล์นี้แล้ว โดยเนื้อหาของไฟล์จะถูกเก็บอยู่ในคลัสเตอร์ที่สัมพันธ์กับเอ็นทรีเหล่านี้ หรือก็คือคลัสเตอร์หมายเลย 2 , 3 , 6 และ 7 ตามลำดับ

รูทไดเรกทอรี ถูกออกแบบมาให้เก็บข้อมูลหมายเลขคลัสเตอร์เริ่มต้นสำหรับไฟล์ต่างๆ เปรียบได้กับสารบัญหนังสือที่เมื่อต้องการจะอ่านไฟล์ไหนก็เปิดไปตูที่รูทไดเรกทอรีแล้วก็ไปตามตำแหน่งที่อยู่ของไฟล์ได้ทันที รูทไดเรกทอรีประกอบด้วยเอ็นทรีขนาด 32 ไบต์ แต่ละเอ็นทรีทำหน้าที่เก็บรายละเอียดของไฟล์ๆหนึ่ง เช่น ชื่อ , ส่วนขยายหรือนามสกุล , หมายเลขคลัสเตอร์แรกของไฟล์ ฯลฯ (ดูรายละเอียดจากตารางที่ 2.6) จำนวนเอ็นทรีสูงสุดที่ยอมให้มีได้ในรูทไดเรกทอรีถูกกำหนดไว้ในบู๊ตเซกเตอร์ บริเวณออฟเซต 11h

ตารางที่ 2.6 ตารางรายละเอียดข้อมูลของไฟล์แต่ละไดเรกทอรี

ออฟเซต	รายละเอียด	ขนาด
00h	ชื่อไฟล์	8 ไบต์
08h	ส่วนขยายของไฟล์	3 ไบต์
0Bh	แอตทริบิวต์ของไฟล์	1 ไบต์
0Ch	สงวนไว้	10 ไบต์
16h	เวลาที่ไฟล์ได้รับการปรับปรุงล่าสุด	1 เวิร์ด (2 ไบต์)
18h	วัน/เดือน/ปีที่ไฟล์ได้รับการปรับปรุงล่าสุด	1 เวิร์ด (2 ไบต์)
1Ah	หมายเลขคลัสเตอร์แรกของไฟล์	1 เวิร์ด (2 ไบต์)
1Ch	ขนาดของไฟล์	1 ดับเบิ้ลเวิร์ด(4 ไบต์)

ชื่อและส่วนขยายของไฟล์

ที่ไบต์แรกของชื่อไฟล์จะใช้เป็นตัวบ่งสถานะบางอย่างคือ

ตารางที่ 2.7 ตารางไบต์แรกของชื่อไฟล์

ค่า	ความหมาย
00H	เป็นเอ็นทรีสุดท้าย
05H	อักขระตัวแรกของชื่อไฟล์มีรหัสแอสกี E5H
2EH	เป็นเอ็นทรีที่หมายถึงตัวไดเรกทอรีปัจจุบัน
E5H	ไฟล์นี้ถูกลบทิ้งไปแล้ว

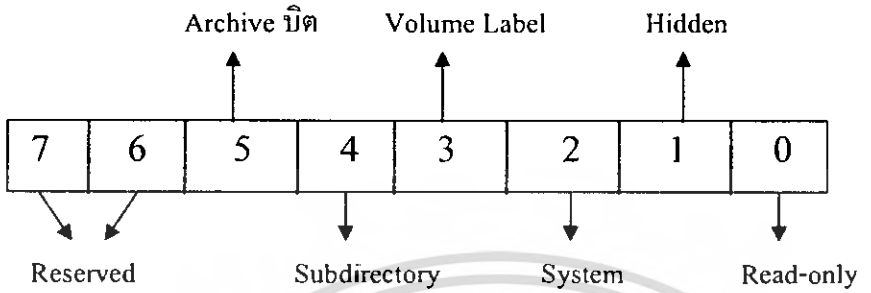
ไบต์แรกที่ออฟเซต 00h นอกจากจะเป็นอักษรตัวแรกของชื่อไฟล์แล้วยังเป็นเอ็นทรีสุดท้ายของไดเรกทอรีนั้นๆ

เอ็นทรีที่มีส่วนของชื่อไฟล์เริ่มต้นด้วยรหัส E5h จะหมายถึงไฟล์ที่ถูกลบทิ้งแล้ว ซึ่งก็คือกระบวนการลบไฟล์

กรณีที่ค่า 2Eh หรือตัวอักษรจุด (.) เป็นไบต์แรกในเอ็นทรีจะมีความพิเศษคือ หมายถึงตัวไดเรกทอรีที่เอ็นทรีนั้นอาศัยอยู่ และถ้าไบต์ถัดไปยังคงเป็น 2Eh (2 จุดติดกัน) ตัวเอ็นทรีจะอ้างถึงไดเรกทอรีที่อยู่เหนือขึ้นไป 1 ระดับ

แอตทริบิวต์

ออฟเซต 0Bh ขนาด 1 ไบต์ของทุกๆเอ็นทรี แต่ละบิตในไบต์ดังกล่าวบ่งชี้สถานะ "ใช่" หรือ "ไม่ใช่" อันเกี่ยวข้องกับคุณลักษณะในรูปที่ 2.19



รูปที่ 2.19 รูปแต่ละบิตในไบต์ของแอตทริบิวต์

- เป็นไฟล์ที่ยอมให้อ่านได้อย่างเดียว (Read-only File)
- เป็นไฟล์ที่ซ่อนไว้ (Hidden File)
- เป็นไฟล์ระบบ (System File)
- เป็นโวลุมลาเบล
- เป็นไดเรกทอรีย่อย
- เป็นไฟล์ที่เพิ่งผ่านการเปลี่ยนแปลงแก้ไขมา

จากรูปที่ 2.19 แสดงตำแหน่งบิตที่สัมพันธ์กับแอตทริบิวต์หรือคุณลักษณะต่างๆข้างต้น ค่า 1 และ 0 ของบิตหมายถึง "มี" และ "ไม่มี" คุณลักษณะนั้นๆตามลำดับ

บิต 0

ผลลัพธ์ : ไฟล์ได้รับการป้องกันการเขียน (Write Protected)

เนื้อหาของไฟล์จะไม่สามารถเปลี่ยนแปลงแก้ไขได้ รวมทั้งไม่สามารถลบไฟล์ได้ แต่จากการทดลอง พบว่าแอตทริบิวต์นี้ไม่เกิดผลอย่างที่ควรจะเป็น โดยสามารถที่จะทำการลบได้

บิต 1

ผลลัพธ์ : ไฟล์ถูกมองข้ามจากหลายๆคำสั่ง

สำหรับไดเรกทอรีย่อยเมื่อกำหนดให้ Hidden แม้คำสั่ง dir จะไม่แสดงรายชื่อออกมา แต่ก็ยังสามารถใช้คำสั่งกับไดเรกทอรีได้ตามปกติ

บิต 2

ผลลัพธ์ : กลายเป็นไฟล์ระบบ (System File)

นิยมใช้แอตทริบิวต์นี้ร่วมกับ Read-only และ Hidden

บิต 3

ความหมาย : เป็นโวลุมลาเบล

โวลุมลาเบลเปรียบเหมือนชื่อเสียงเรียงนามของโวลุม โดยข้อความที่ต้องการให้เป็นลาเบลจะถูกใส่ไว้ในฟิลด์ชื่อและส่วนขยายของเอ็นทรีในรูทไดเรกทอรี (ออฟเซต 00h ถึง 0Ah) จำนวน 11 ไบต์ ซึ่งมีผลทำให้โวลุมลาเบลมีความยาวได้ไม่เกิน 11 อักขระ โวลุมลาเบลเป็นเพียงเอ็นทรีหนึ่งในรูทไดเรกทอรีเท่านั้น ไม่มีเนื้อหาอยู่ภายใน

บิต 4

ความหมาย : เป็นไดเรกทอรีย่อย หรือก็คือไฟล์ๆหนึ่ง

บิต 5

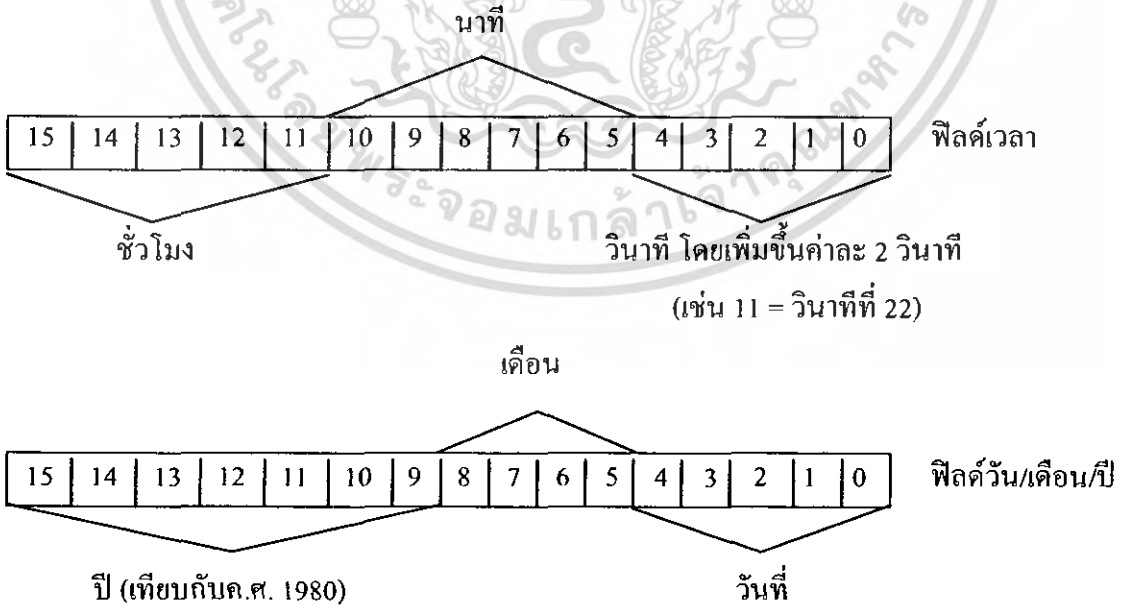
ความหมาย : ไฟล์ถูกแก้ไขหลังจากการทำสำรองครั้งสุดท้าย

บิด 6-7 สงวนไว้

วัน/เวลาที่ไฟล์ได้รับการปรับปรุงครั้งล่าสุด

ข้อมูล 1 เวิร์ดจำนวน 2 ชุด เริ่มต้นที่ออฟเซต 18h และ 16h ตามลำดับ บ่งบอกวัน/เดือน/ปี (Date) และเวลา (Time) ที่ไฟล์ถูกแก้ไขครั้งล่าสุด การแก้ไขนี้ดูที่ตัวเนื้อหาในไฟล์เท่านั้น ไม่รวมการอ่าน การคัดลอก หรือ แม้กระทั่งการเปลี่ยนชื่อไฟล์

วัน/เวลาถูกแปลงหรือเข้ารหัสจากเลขฐาน 10 เป็นเลขฐาน 2 โดยจะเก็บอยู่ในรูปของฟิลด์วัน/เดือน/ปี และ เวลาดังรูปที่ 2.20



รูปที่ 2.20 รูปแสดงความหมายของบิตในฟิลด์ วัน/เดือน/ปี และ เวลา

บิต 9 ถึง 15 (7 บิตซ้ายสุด) ของฟิลด์วัน/เดือน/ปีเก็บตัวเลขเทียบกับปีค.ศ. 1980 กล่าวคือ ต้องนำตัวเลขที่คพนวณได้จาก 7 บิตนี้มาบวกกับ 1980 เพื่อทราบปี ค.ศ.ที่ถูกต้อง ในทางกลับกันเมื่อสร้างไฟล์ขึ้นมาใหม่ดอสจะนำปีค.ศ.ปัจจุบัน (พิจารณาจากวันเวลาในเครื่อง) ลบด้วย 1980 แล้วบันทึกตัวเลขผลลัพธ์ในบิต 9 ถึง 15 ของฟิลด์วัน/เดือน/ปี

สำหรับฟิลด์เวลา จุดที่น่าสนใจอยู่ที่บิต 0 ถึง 4 (5 บิตขวาสุด) ซึ่งเก็บข้อมูล "วินาที" ไว้ ซึ่งใน 1 นาทีมี 60 วินาที แต่ 5 บิตให้ค่าได้ต่างๆกัน 32 ค่าเท่านั้น คิดเป็นจำนวนเต็มแบบไร้เครื่องหมาย (Unsigned Integer) ตั้งแต่ 0 ถึง 31 อย่างน้อยที่สุดต้องใช้ 6 บิตถึงจะเพียงพอ ซึ่งไม่สามารถทำได้ จึงพยายามใช้ประโยชน์จาก 5 บิต โดยหาร "วินาที" ด้วย 2 เพื่อให้ค่าไม่เกิน 31 แล้วค่อยบันทึกในบิต 0 ถึง 4

หมายเลขคลัสเตอร์แรกของไฟล์

เป็นข้อมูลขนาด 1 เวิร์ดที่ออฟเซต 1Ah บอกหมายเลขคลัสเตอร์แรกของไฟล์

ขนาดของไฟล์

ออฟเซต 1Ch (1 ดับเบิ้ลเวิร์ด) คือตัวเลขระบุขนาดของไฟล์ แม้ดอสจะใช้รูทไดเรกทอรีและแฟท ติดตามเนื้อหาของไฟล์จากคลัสเตอร์หมายเลขต่างๆ แต่ขนาดที่แท้จริงของไฟล์อาจไม่ลงตัวกับคลัสเตอร์พอดี ซึ่งวิธีเดียวที่ดอสจะเก็บรายละเอียดของไฟล์ได้โดยไม่ขาดไม่เกินคือ การพิจารณาจากตัวเลขที่บันทึกไว้ในเอ็นทรี และเนื่องจากฟิลด์นี้ใช้ 1 ดับเบิ้ลเวิร์ด (4 ไบต์) จึงรองรับไฟล์ขนาดสูงสุด 4,294,967,295 ไบต์ หรือประมาณ 4 กิกะไบต์

2.4.3.2 ไดเรกทอรีย่อย

ดอสจะเก็บไดเรกทอรีย่อยในรูปแบบเดียวกับไฟล์ โดยบิต 4 ของไบต์แอตทริบิวต์ในรูปที่ 2.19 จะถูกเซตเป็น 1 หมายเลขคลัสเตอร์แรกชี้ไปยังคลัสเตอร์ที่เป็นจุดเริ่มต้นของข้อมูลในไดเรกทอรีย่อย (ในส่วนของพื้นที่จัดเก็บข้อมูล (File Area)) ข้อมูลของไดเรกทอรีย่อยเป็นโครงสร้างที่เลียนแบบมาจากรูทไดเรกทอรี ประกอบด้วยเอ็นทรีขนาด 32 ไบต์ที่ใช้เก็บรายละเอียดของไฟล์ (หรือไดเรกทอรีย่อยอีกที)

ไดเรกทอรีย่อยจึงบรรจุเอ็นทรีอ้างถึงไฟล์ (หรือไดเรกทอรีย่อย) ได้ไม่จำกัดจำนวนแล้วแต่ความจุของดิสก์จะเอื้ออำนวย หากเอ็นทรีในไดเรกทอรีย่อยมีจำนวนมากจน 1 คลัสเตอร์รับมือไม่ไหว ดอสจะจองคลัสเตอร์ว่างถัดไปเข้ามาร่วมโดยใช้แฟท เป็นตัวเชื่อมโยงเหมือนเคยทั้งนี้ขนาดของไดเรกทอรีย่อยถูกกำหนดเป็น 0 เสมอ

2.4.4 พื้นที่จัดเก็บข้อมูล

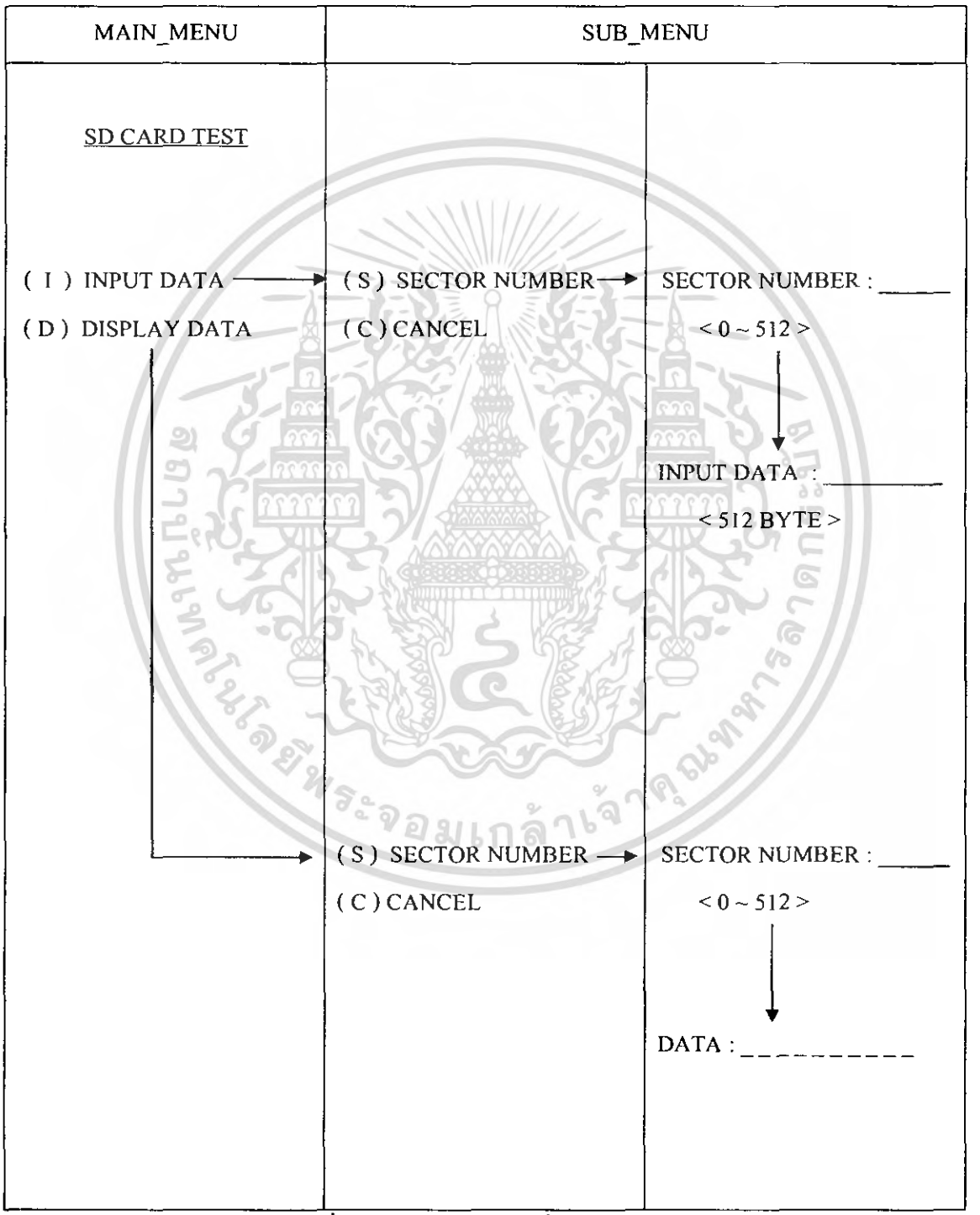
พื้นที่จัดเก็บข้อมูล (File Area หรือ Data Area) เป็นพื้นที่ส่วนที่มีอาณาบริเวณใหญ่โดกว้างขวางที่สุด โดยข้อมูลทั้งหมดของไฟล์ต่างๆรวมทั้งไดเรกทอรีย่อยทั้งหมดจะถูกจัดเก็บอยู่ในส่วนนี้ เมื่อดอสต้องการเขียนหรืออ่านไฟล์/ไดเรกทอรีย่อย จะกระทำกับกลุ่มของลอจิคัลเซกเตอร์ที่รวมตัวกันเป็นคลัสเตอร์ ไม่ได้จัดการกับเซกเตอร์ใดโดยเฉพาะ

ถ้าต้องการอยากรู้ว่าพื้นที่จัดเก็บข้อมูลประกอบด้วยเซกเตอร์จำนวนเพียงใด จะต้องอ่านข้อมูลขนาด 1 เวิร์ดจากออฟเซต 13h ของบู๊ตเซกเตอร์ แล้วนำค่าที่ได้มาลบด้วยจำนวนเซกเตอร์ที่ถูกใช้ไปกับพื้นที่ 3 ส่วนแรก (พื้นที่สงวน , แฟท และ รูทไดเรกทอรี) ผลลัพธ์ที่ได้ก็คือจำนวนเซกเตอร์ของพื้นที่จัดเก็บข้อมูล

บทที่ 3

การออกแบบ

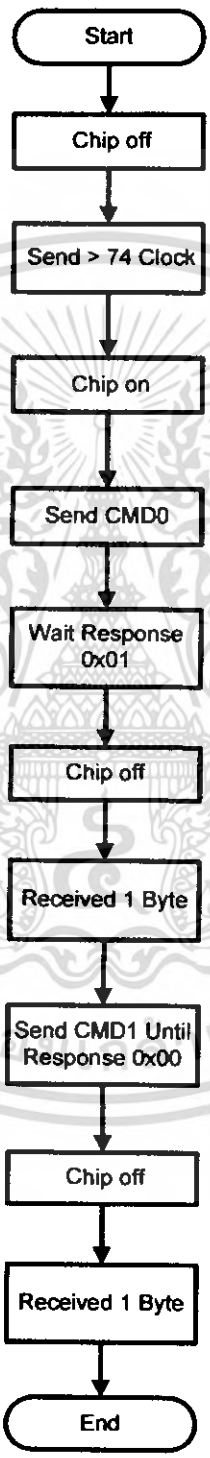
3.1 รูปแบบหน้าจอที่ติดต่อกับผู้ใช้



รูปที่ 3.1 รูปแบบหน้าจอที่ติดต่อกับผู้ใช้

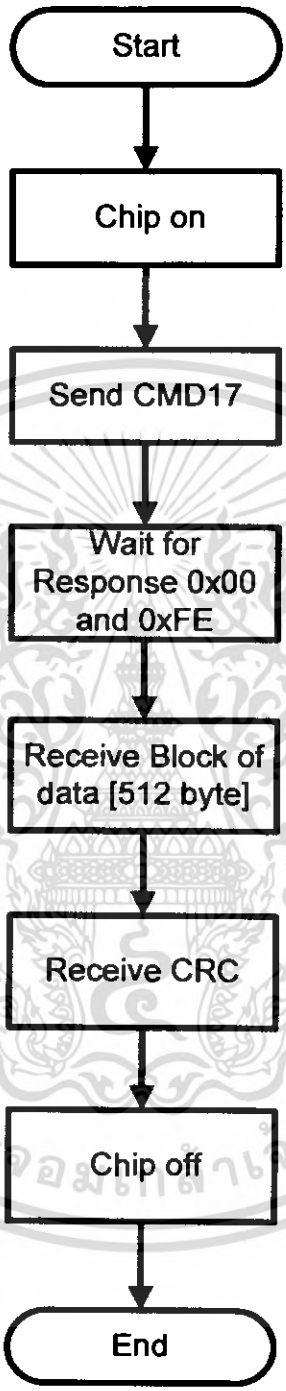
3.2 โฟลว์ชาร์ท (Flow chart) แสดงการทำงาน

3.2.1 โฟลว์ชาร์ทการตั้งค่าเริ่มต้นให้เอสดีการ์ดทำงานในโหมดเอสพีไอ



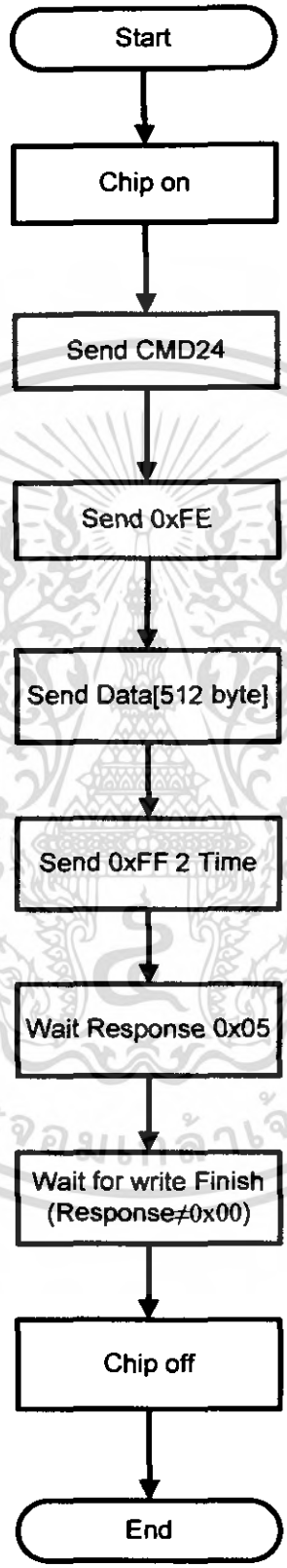
รูปที่ 3.2 โฟลว์ชาร์ทการตั้งค่าเริ่มต้นให้เอสดีการ์ดทำงานในโหมดเอสพีไอ

3.2.2 โฟลว์ชาร์ทการตั้งค่าให้เอสดีการ์ดทำงานการอ่านข้อมูล



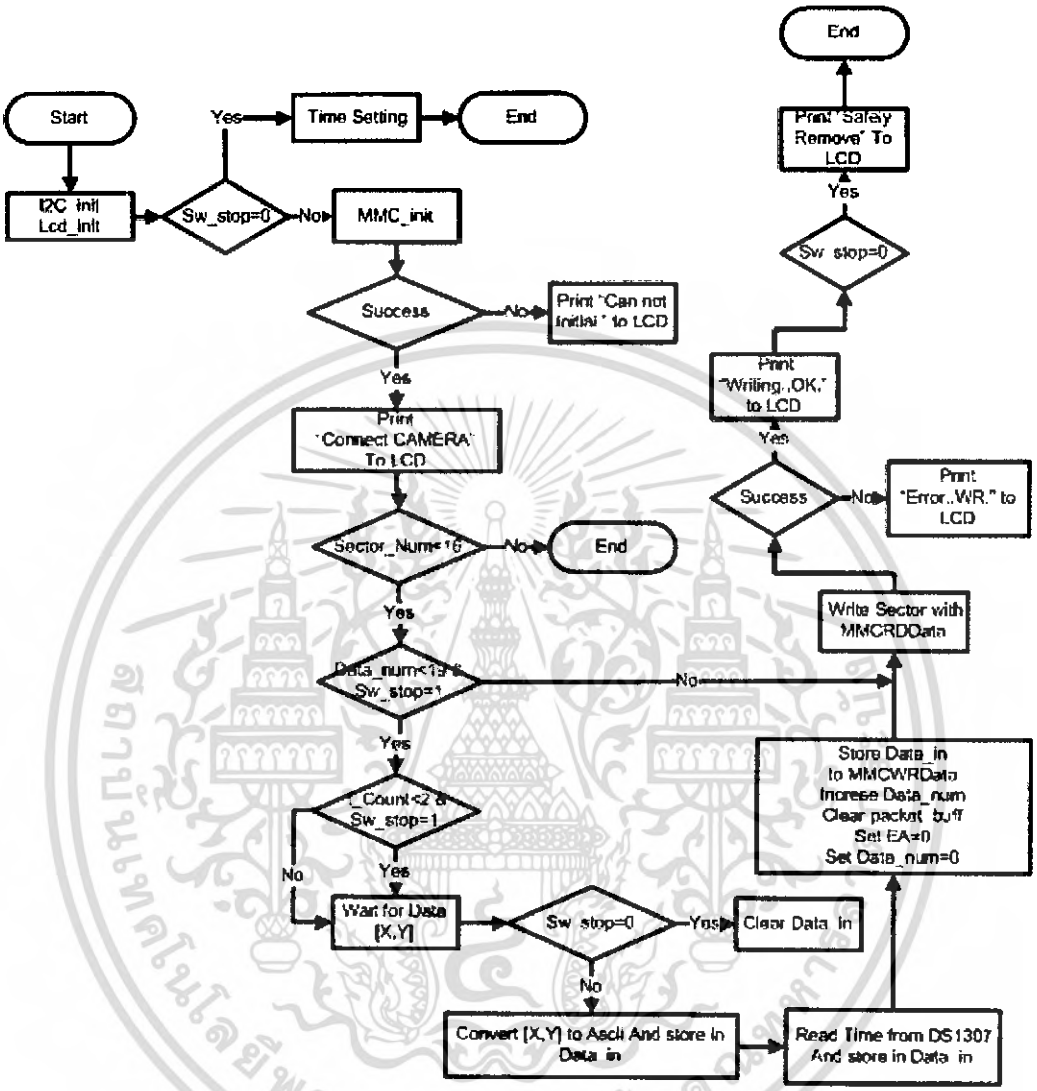
รูปที่ 3.3 โฟลว์ชาร์ทการตั้งค่าให้เอสดีการ์ดทำงานการอ่านข้อมูล

3.2.3 โฟลว์ชาร์ทการตั้งค่าให้เอสดีการ์ดทำงานการเขียนข้อมูล



รูปที่ 3.4 โฟลว์ชาร์ทการตั้งค่าให้เอสดีการ์ดทำงานการเขียนข้อมูล

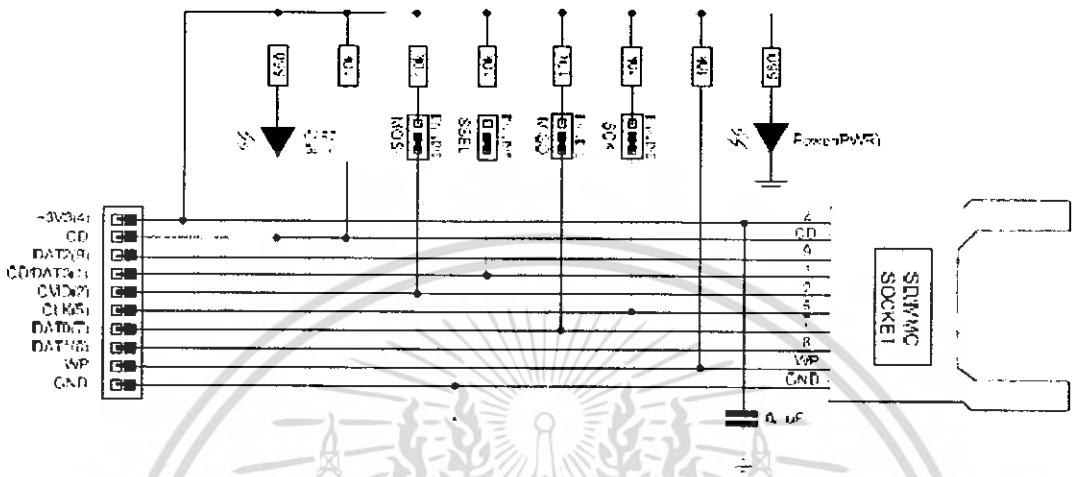
3.2.4 โฟลว์ชาร์ทแสดงภาพรวมการทำงานของโปรแกรม



รูปที่ 3.5 โฟลว์ชาร์ทแสดงภาพรวมการทำงานของโปรแกรม

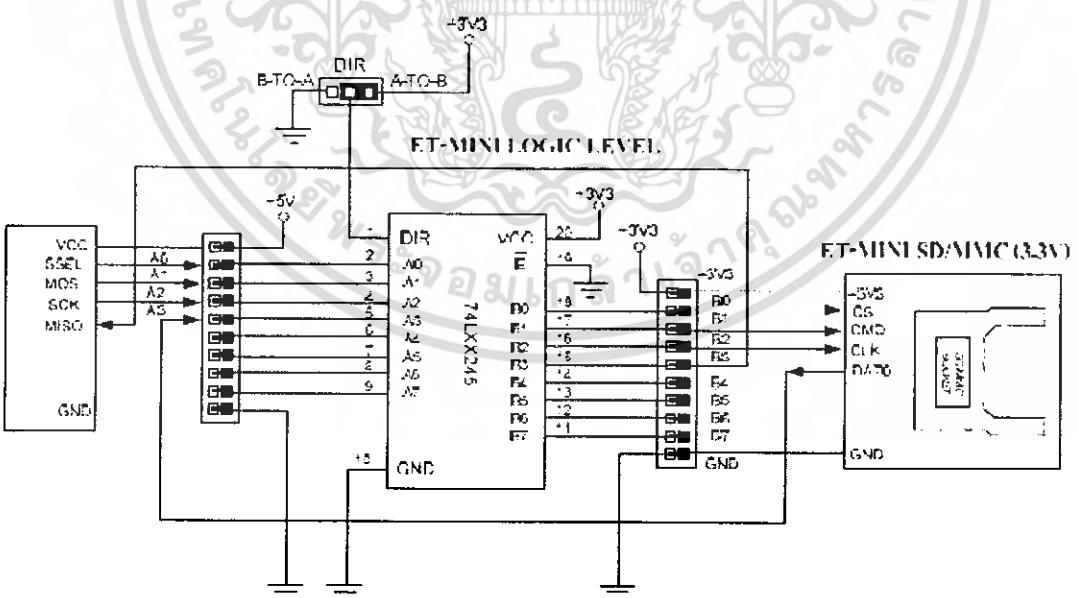
3.3 การออกแบบวงจร

3.3.1 รูปแสดงรายละเอียดระบบวงจรของเอสดีการ์ด



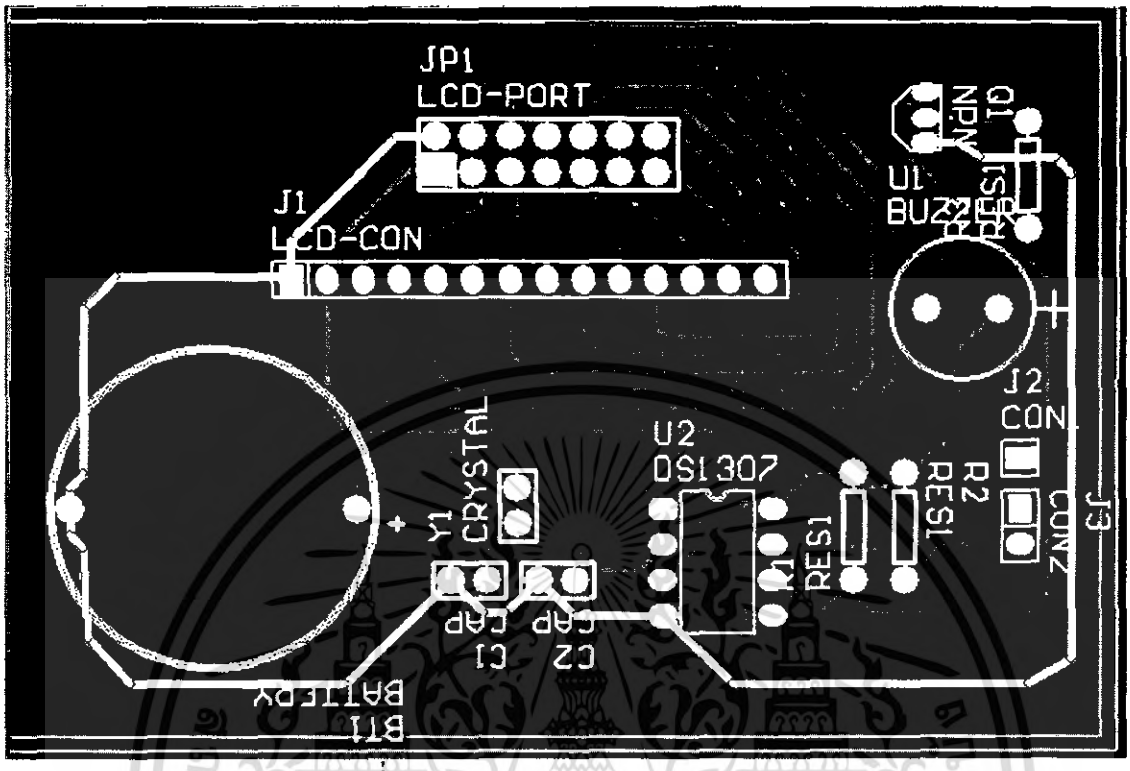
รูปที่ 3.6 ระบบวงจรของเอสดีการ์ด

3.3.2 การเชื่อมต่อสัญญาณระหว่างไมโครคอนโทรลเลอร์ (อุปกรณ์ 5V) กับ SD/MMC CARD (อุปกรณ์ 3V) โดยใช้ชุด ET-MINI LOGIC LEVEL และ ET-MINI SD/MMC ในโหมดเอสพีไอ

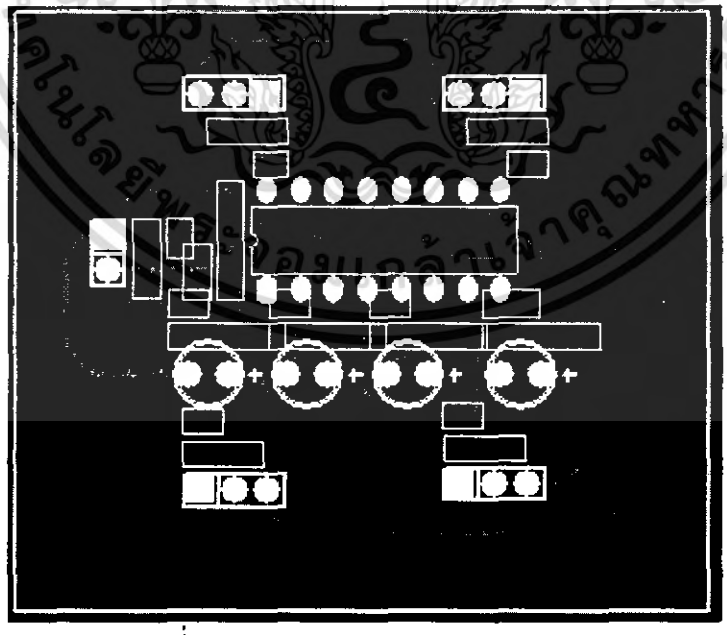


รูปที่ 3.7 การเชื่อมต่อสัญญาณระหว่างไมโครคอนโทรลเลอร์กับเอ็มเอ็มซีเอสดีการ์ด

3.3.3 ลายวงจรที่ใช้โปรแกรมโปรเทล (Protel) ในการเขียนขึ้นมา



รูปที่ 3.8 ลายวงจรดีเอส 1307 (DS 1307)



รูปที่ 3.9 ลายวงจรแมซ 232 (MAX 232)

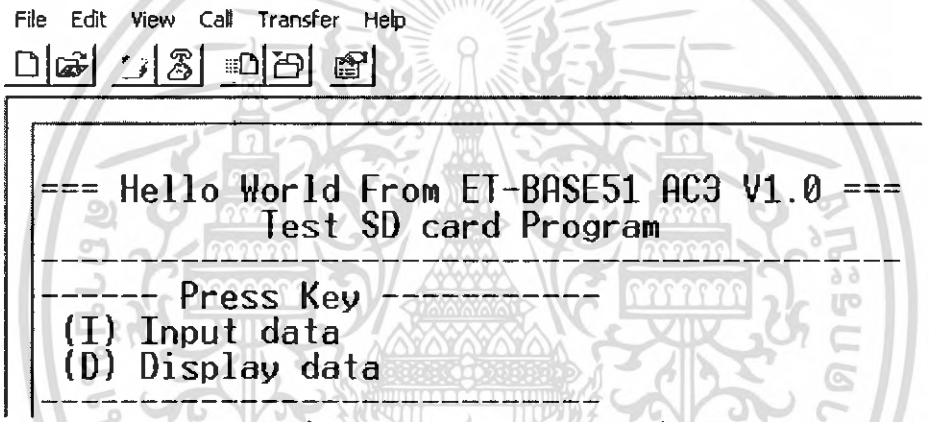
บทที่ 4

ผลการทดลอง

4.1 ผลการทดลอง

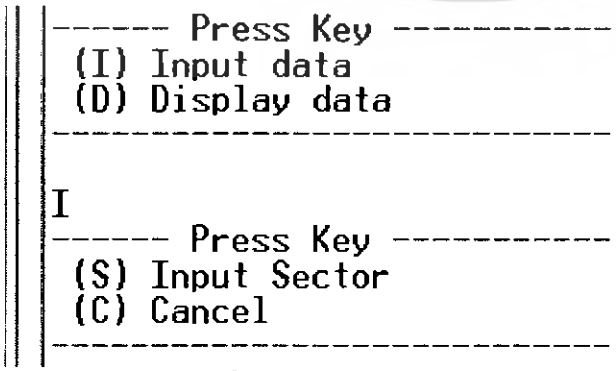
งานปริญญานิพนธ์นี้คือการทำระบบไมโครคอนโทรลเลอร์เอ็มซีเอส 51 ให้มีความสามารถในการจัดเก็บข้อมูลได้โดยจะเก็บลงในตัวเอ็มเอ็มซีเอสดีการ์ดโดยป้อนค่าผ่านอาร์เอส 232 ทางคอมพิวเตอร์ ซึ่งผลการทดลองที่ได้คือ

4.1.1 ผลการทดลองที่ใช้การป้อนค่าเข้าเอสดีการ์ดโดยผ่านทางคอมพิวเตอร์



รูปที่ 4.1 หน้าจอเมนูเริ่มการทำงาน

เมื่อเริ่มติดต่อข้อมูลของไมโครคอนโทรลเลอร์ กับ คอมพิวเตอร์ โดยใช้โปรแกรมไฮเปอร์เทอมินอล (Hyper Terminal) จะปรากฏหน้าจอดังรูปที่ 4.1 จากรูปนี้จะมีการให้ใส่ค่าเข้าไป 2 ค่า คือค่า 'I' เพื่อไปสู่เมนูการใส่ข้อมูลเข้าไป และ ค่า 'D' เพื่อไปสู่เมนูการแสดงผลข้อมูลออกมา



รูปที่ 4.2 แสดงเมนูการเลือกใส่ข้อมูลลงตำแหน่ง

หลังจากที่มีการใส่ค่า 'I' เพื่อต้องการใส่ค่าข้อมูลเข้าไปจะปรากฏเมนูขึ้นมาดังรูปที่ 4.2 ซึ่งจะมีการให้เลือกว่าจะมีการใส่ค่าข้อมูลลงไปในตำแหน่ง โดยที่จะต้องกด 'S' หรือจะยกเลิกคำสั่งทั้งหมดให้กลับไปที่เมนูเริ่มแรกอีกครั้งได้โดยการกด 'C'

```
I
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
S
Sector number<0-7FFFF>:  20
```

รูปที่ 4.3 หน้าจอการใส่ค่าตำแหน่งที่ต้องการใส่ค่าข้อมูล

จากการที่เลือกใส่ค่า 'S' เป็นการเลือกตำแหน่งที่เราต้องการที่จะใส่ค่าข้อมูลเข้าไปโดยในรูปที่ 4.3 เราได้เลือกตำแหน่งที่ 20 ที่เราต้องการใส่ค่าข้อมูลเข้าไป โดยที่ในเมนูนี้นั้นมีการใส่ค่า 'C' เพื่อยกเลิกให้กลับไปสู่ที่หน้าจอเมนูหลักเริ่มแรกอีกครั้ง

```
Input Data<not over 512 bytes>: TEST_SD_CARD
------ Press Key -----------
 (I) Input data
 (D) Display data
----------------------------
```

รูปที่ 4.4 หน้าจอการใส่ข้อมูลเข้าไปในเอสดีการ์ด

หลังจากเราเลือกตำแหน่งที่จะใส่ค่าเข้าไปแล้วต่อมาจะมีการให้ใส่ค่าเข้าไปได้เลย โดยในรูปที่ 4.4 เราได้ใส่ค่าเข้าไปว่า " TEST_SD_CARD" เมื่อใส่ค่าข้อมูลเข้าไปเป็นที่เรียบร้อยแล้วหน้าจอเมนูจะกลับมาที่หน้าจอเมนูหลักเริ่มแรกอีกครั้ง โดยมีการให้เลือกใส่ค่า 'I' กับค่า 'D' เข้าไป

```
D
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
```

รูปที่ 4.5 หน้าจอเมนูการเลือกการแสดงผลของข้อมูลในเอสดีการ์ด

จากหน้าจอเมนูหลักเริ่มแรกนั้นเราได้เลือก 'D' เพื่อให้มีการแสดงผลของข้อมูล หลังจากเลือกการแสดงผลแล้วจะปรากฏหน้าจอเมนูย่อยคือ ให้ใส่ค่า 'S' เพื่อเลือกตำแหน่งที่จะเข้าไปนำผลของข้อมูลที่จะแสดงออกมา และ ค่า 'C' เพื่อยกเลิกคำสั่งทั้งหมดแล้วกลับสู่หน้าจอเมนูหลักอีกครั้ง

```
D
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
S
Sector number<0-7FFFF>:  20


========================================
Data: TEST_SD_CARD
========================================

------ Press Key -----------
 (I) Input data
 (D) Display data
----------------------------
```

รูปที่ 4.6 หน้าจอแสดงผลของข้อมูลที่แสดงออกมา

เมื่อเลือกตำแหน่งที่จะเข้าไปนำผลของข้อมูลที่จะแสดงออกมาแล้วโดยในรูปที่ 4.6 นี้เราได้เลือกตำแหน่งคือ 20 เมื่อเลือกแล้วข้อมูลที่อยู่ในตำแหน่งนี้นั้นก็ได้แสดงผลออกมาคือ " TEST_SD_CARD " หลังจากที่แสดงผลออกมาแล้วนั้น หน้าจอก็ได้กลับไปสู่หน้าจอเมนูหลักอีกครั้งนึง

```
I
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
5
 Not number, Try again..
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
```

รูปที่ 4.7 หน้าจอแสดงผลการใส่ค่าคำสั่งผิดจากที่กำหนดให้

ในรูปที่ 4.7 นี้เป็นรูปที่แสดงผลจากการที่ใส่ข้อมูลที่มีผิดพลาดเข้าไปโดยระบบจะทำการปฏิเสธค่า แล้วให้มีการใส่ค่าข้อมูลเข้าไปอีกครั้งนึง

```
S
Sector number<0-7FFFF>:  524288

 Over range, Insert again..
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
```

รูปที่ 4.8 หน้าจอแสดงผลการใส่ค่าตำแหน่งที่ผิดจากที่กำหนดให้

ในรูปที่ 4.8 นี้เป็นการที่ใส่ค่าตำแหน่งผิดโดยใส่เกินจากการที่มีการกำหนดค่าตำแหน่งที่ใส่ได้ ซึ่งระบบจะทำการปฏิเสธค่าตำแหน่งนี้แล้วจะมีการให้ใส่ค่าตำแหน่งใหม่อีกครั้งนึง

```
I
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
S
Sector number<0-7FFFF>:  20

Input Data<not over 512 bytes>:
------ Press Key -----------
 (I) Input data
 (O) Display data
----------------------------
```

รูปที่ 4.9 หน้าจอแสดงผลวิธีการลบข้อมูลในตำแหน่งที่ต้องการ

จากรูปที่ 4.9 เป็นการลบข้อมูลในตำแหน่งที่เราต้องการจะลบข้อมูล โดยในรูปนี้จากหน้าจอเมนูหลักเราจะทำการเลือก 'I' เพื่อให้ใส่ค่าเข้าไป ต่อมาก็จะเลือก 'S' เพื่อเลือกตำแหน่งที่เราต้องการที่จะลบข้อมูลออกมาในรูปนี้เราได้ทำการเลือกตำแหน่งที่ 20 ที่ต้องการจะลบข้อมูล เมื่อเลือกตำแหน่งแล้วจะมีคำสั่งให้ใส่ข้อมูลเข้าไป ในขั้นตอนนี้เราได้ทำการเคาะ สเปซบาร์ (SPACE BAR) ไป 1 ครั้งเพื่อให้เกิดช่องว่างขึ้นแล้วต่อมาก็กดเอ็นเทอร์ (ENTER) เพื่อให้โปรแกรมทำงานต่อไปยังขั้นตอนหลังจากนี้โดยที่จะไปสู่หน้าจอเมนูหลักอีกครั้งนึง

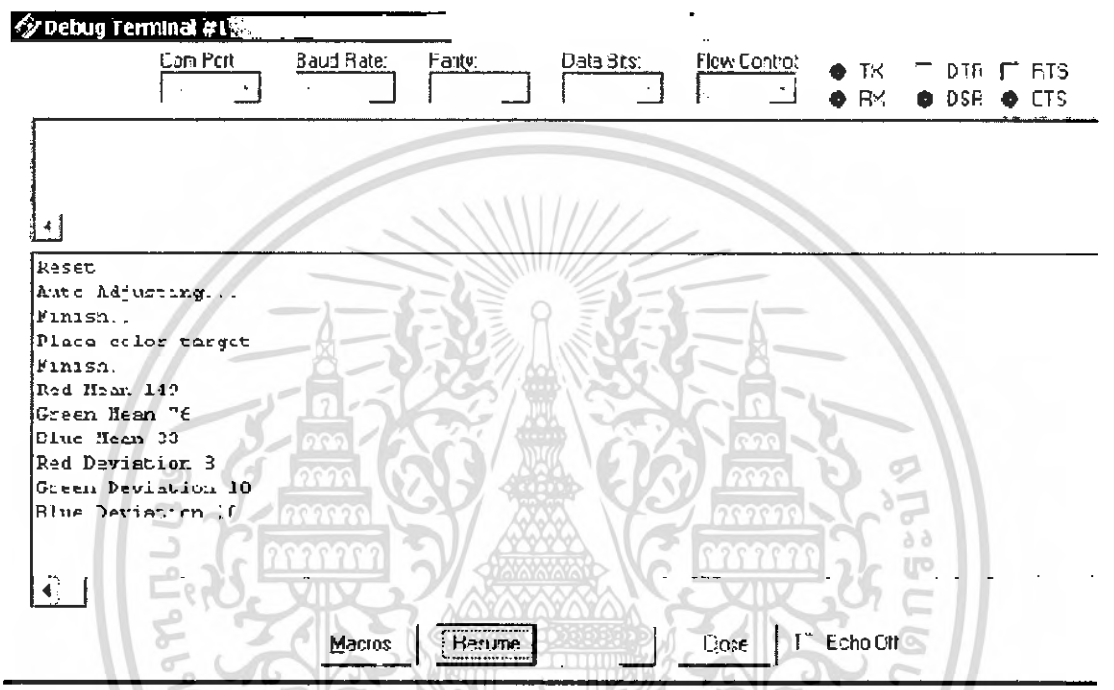
```
D
------ Press Key -----------
 (S) Input Sector
 (C) Cancel
----------------------------
S
Sector number<0-7FFFF>:  20


========================================
Data:
========================================

------ Press Key -----------
 (I) Input data
 (D) Display data
----------------------------
```

รูปที่ 4.10 หน้าจอแสดงผลจากการที่ได้ลบข้อมูลออกไปแล้ว

จากรูปที่ 4.10 นี้เป็นการแสดงถึงผลจากการที่เราลบข้อมูลแล้วหลังจากขั้นตอนในรูปที่ 4.9 ซึ่งในรูปนี้เราได้ทำการเลือก 'D' เพื่อเลือกแสดงผลข้อมูลออกมาจากนั้นก็มีการเลือกตำแหน่งที่เราจะเข้าไปดูข้อมูลในตำแหน่งที่ 20 แล้วผลลัพธ์ที่ออกมาก็จะไม่มีข้อมูลออกมา ซึ่งก็คือข้อมูลได้ถูกลบออกไปเรียบร้อยแล้ว เมื่อแสดงผลเสร็จก็จะกลับไปสู่หน้าจอเมนูหลักอีกครั้งนึง

4.1.2 ผลการทดลองการเก็บค่าข้อมูลของโมดูลกล้องลงในเอสดีการ์ด

ในการทำโครงงานนี้เราได้ใช้โมดูลกล้องคือไอวีแคม (IV-CAM) ในการจับตำแหน่งของวัตถุให้อยู่ในรูปของข้อมูลค่า x และ ค่า y แล้วข้อมูลที่ได้มานี้จะถูกบันทึกลงในตัวเอสดีการ์ดที่ได้ทำการเชื่อมต่อกับกล้องผ่านตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51แล้วนำข้อมูลที่ได้นี้ไปพล็อต(plot)กราฟลงในโปรแกรมแมทแลป (MATLAB) เพื่อแสดงให้เห็นถึงการเปลี่ยนแปลงของตำแหน่งวัตถุ

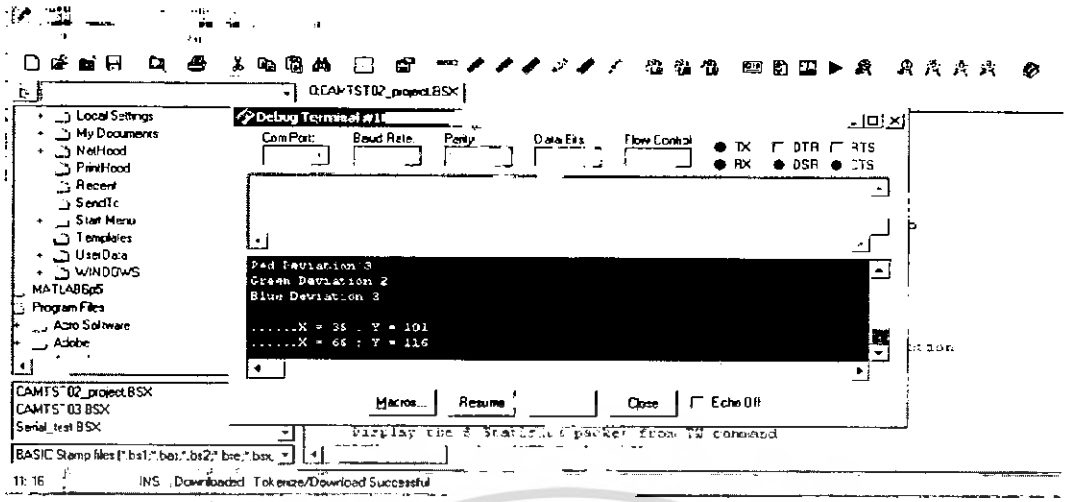


รูปที่ 4.11 รูปหน้าจอแสดงผลที่ส่งมาจากกล้องไอวีแคม

จากรูปที่ 4.11 แสดงถึงหน้าจอ Debug Terminal ที่แสดงผลของการติดต่อกับกล้องไอวีแคมโดยใช้โปรแกรมที่ป้อนเข้าไปเพื่อหาตำแหน่งของวัตถุ โดยการทำงานของโปรแกรมจะเริ่มจากการตั้งค่ากล้องแล้วรอรับการตอบกลับจากกล้องเมื่อได้รับการตอบกลับ I-Stamp จะแสดงข้อความ Reset ที่หน้าต่าง

จากนั้นกล้องจะปรับค่าความสว่างของจอภาพอัตโนมัติซึ่งใช้เวลา 5 วินาทีโดยแสดงการทำงานผ่านหน้าต่างด้วยข้อความ Auto Adjusting เมื่อครบ 5 วินาที I-Stamp จะปิดการทำงานอัตโนมัติโดยส่งข้อความ Finish เมื่อกล้องตอบรับ I-Stamp จะส่งข้อความ Place color target มาแสดงที่หน้าต่างซึ่งจังหวะนี้จะมีการวางวัตถุเป้าหมายวางไว้หน้ากล้อง 5 วินาที แล้ว I-Stamp จะส่งข้อความ Finish เพื่อแจ้งว่าครบ 5 วินาทีแล้วต่อไปจะเตรียมเข้าสู่กระบวนการตรวจจับภาพต่อไป

เมื่อกล้องตรวจจับภาพแล้วจะส่งข้อมูลออกมาที่หน้าต่างซึ่งเป็นค่าเฉลี่ยของสีแดง , สีเขียว , สีน้ำเงิน และ ค่าเบี่ยงเบนของสีแดง , สีเขียว , สีน้ำเงิน ออกมาดังรูปที่ 4.11

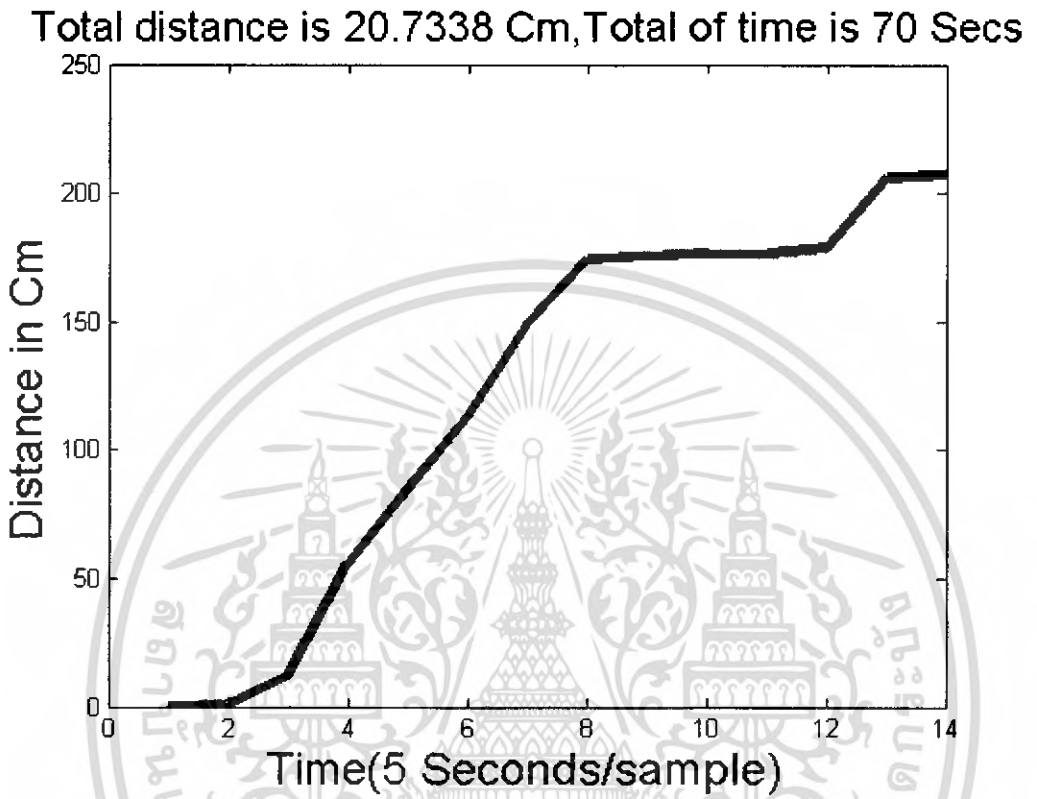


รูปที่ 4.12 รูปแสดงค่าเฉลี่ยของตำแหน่งวัตถุที่จับได้จากกล้องไอวีแคม

หลังจากที่เราวางวัตถุเพื่อกำหนดค่าของสีที่เราต้องการจับภาพแล้ว หลังจากนั้นโปรแกรมจะทำการส่งค่าเฉลี่ยของตำแหน่งของวัตถุออกมาทางซีเรียลพอร์ต โดยในการทดลองนี้แสดงผ่านทางหน้าจอดีบักของโปรแกรมเบสิคสแตมป์อิดิตเตอร์ (BASIC-STAMP EDITOR)

4.1.3 ผลการทดลองการเก็บค่าข้อมูลของโมดูลกล้องที่แสดงผลผ่านโปรแกรมแมทแลป

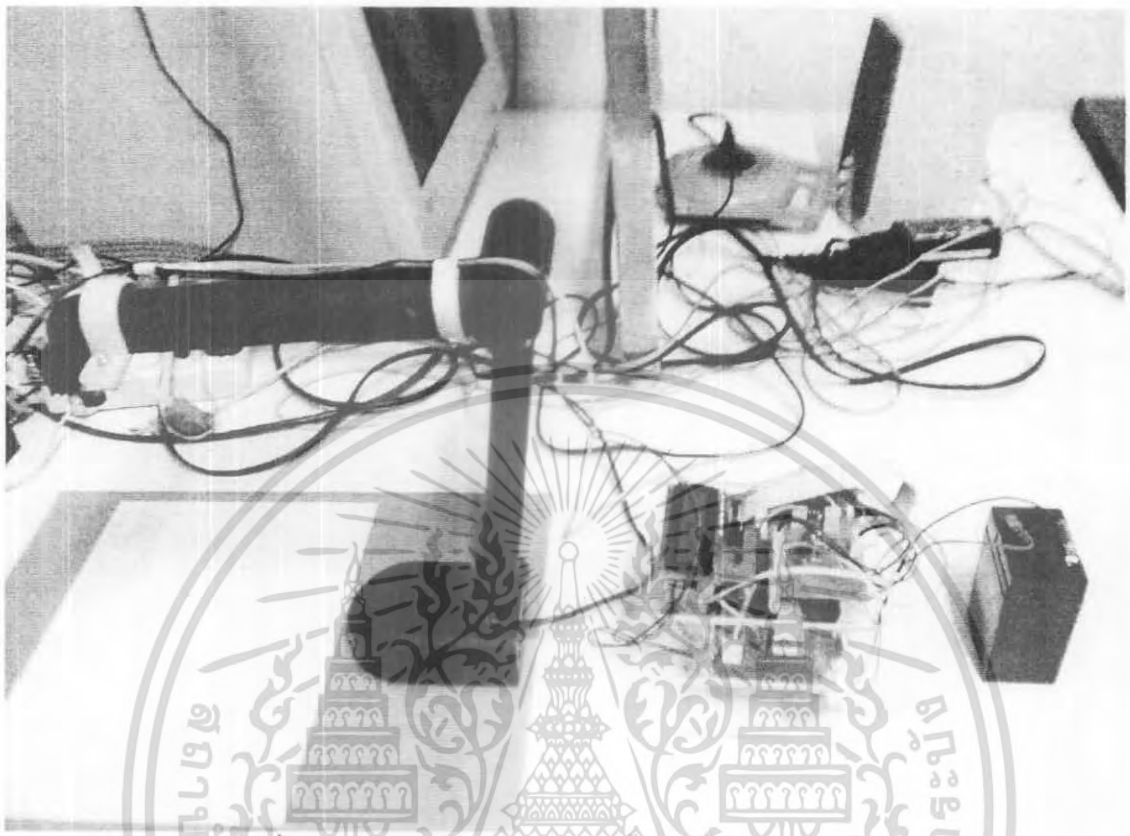
เมื่อเรานำข้อมูลตำแหน่งของวัตถุที่กล้องจับได้ที่ถูกบันทึกอยู่ในเอสดีการ์ดมาผ่านกระบวนการแสดงผลทางโปรแกรมแมทแลปก็จะได้รูปออกมาคือ



รูปที่ 4.13 รูปแสดงข้อมูลที่ถูกนำมาประมวลผลโดยผ่านทางโปรแกรมแมทแลป

จากรูปที่ 4.13 นี้แสดงให้เห็นถึงข้อมูลตำแหน่งของวัตถุโดยจากรูปนี้ ถ้าวัตถุมีการเคลื่อนที่ไปมากก็จะทำให้เส้นกราฟมีความชันมากยิ่งขึ้น และเมื่อวัตถุไม่มีการเคลื่อนที่ก็จะทำให้เส้นกราฟมีลักษณะเป็นเส้นแนวนอน โดยในโปรแกรมนี้เราจะทำการหาตำแหน่งของวัตถุเป็นเวลาทุกๆ 5 วินาที และค่าที่แสดงผลออกมานอกจากเส้นกราฟแล้วยังมีการประมวลผลถึงระยะทางที่วัตถุได้เคลื่อนที่ไปทั้งหมด อีกทั้งยังมีการระบุถึงเวลาที่ใช้ไปทั้งหมดในการจับวัตถุอีกด้วย

4.2 ตัวอย่างแอพพลิเคชันที่สร้างขึ้นมาเพื่อใช้ในโครงงาน



รูปที่ 4.14 รูปแอพพลิเคชันที่สร้างขึ้นมาเพื่อใช้ในโครงงาน

บทที่ 5
สรุปผลการทดลอง

5.1 สรุปผลการทดลอง

โครงงานนี้ได้ทำขึ้นเพื่อนำเสนอการจัดเก็บข้อมูลลงในตัวเอ็มเอ็มซีเอสดีการ์ด โดยใช้ตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51 เอซี 3 เป็นตัวเชื่อมต่อ ซึ่งตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51 นี้เป็นตัวมาตรฐานที่มีการใช้งานอย่างแพร่หลาย อีกทั้งยังนำมาประยุกต์ใช้กับงานด้านต่างๆได้โดย ในโครงงานได้ทำการติดต่อระหว่างตัวเอสดีการ์ดกับตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51 โดยทำการเก็บข้อมูลลงไปในตัวเอสดีการ์ดแล้วสามารถนำไปเชื่อมต่อกับเครื่องคอมพิวเตอร์ เพื่อนำข้อมูลที่ทำการเก็บไปแสดงค่าในงานด้านต่างๆได้

ในตัวอย่างการใช้งานโครงงานนี้มีการในการเก็บตำแหน่งของวัตถุ เพื่อนำไปประมวลผลทางสถิติโดยใช้ไมโครคอมพิวเตอร์ต่อไป โดยตัวไมโครคอนโทรลเลอร์เอ็มซีเอส 51 ได้ทำการเก็บค่าตำแหน่งของวัตถุที่ได้มาจากกล้องไอวีแคม (IV-CAM) ลงไปในตัวเอสดีการ์ดแต่เนื่องจากไม่สามารถสร้างไฟล์ด้วยตัวไมโครคอนโทรลเลอร์เพราะมีการป้องกันการเขียนรูทไดเรกทอรี จากตัวเอสดีการ์ดข้อมูลที่ได้จึงถูกเขียนลงไปในส่วนของพื้นที่จัดเก็บข้อมูล เรียงต่อกันไปเรื่อยๆเพื่อให้เครื่องคอมพิวเตอร์สามารถมองเห็นข้อมูลได้ จากนั้นทำการถอดตัวเอสดีการ์ดนำไปเชื่อมต่อกับเครื่องคอมพิวเตอร์แล้วนำไปแสดงผลในโปรแกรมแมทแลป (MATLAB) เพื่อแสดงถึงค่าตำแหน่งของวัตถุที่มีการเปลี่ยนแปลงไปจากตำแหน่งเดิมทุกๆ 5 วินาที

5.2 ปัญหาที่เกิดขึ้นในการทดลอง

5.2.1 ตัวไมโครคอนโทรลเลอร์ไม่สามารถสร้างไฟล์ขึ้นมาเองได้

5.2.2 เนื่องจากไม่ถนัดในการเขียนโปรแกรม VB จึงต้องใช้โปรแกรมแมทแลปในการแสดงผล

5.3 แนวทางการพัฒนาโครงงาน

5.3.1 ปรับปรุงวงจรให้มีขนาดกะทัดรัดมากยิ่งขึ้น

5.3.2 เพิ่มเติมในส่วนของการใช้งานให้สามารถใช้งานได้รูปแบบหลากหลายยิ่งขึ้น

บรรณานุกรม

นคร ภักดีชาติ และชัยวัฒน์ ลิ้มพรจิตรวิไล. **เรียนรู้และสร้างหุ่นยนต์อัตโนมัติกับไมโครคอนโทรลเลอร์ MCS-51.** บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด.

สันติ นุราช และอุกฤษฏ์ ตันทสุทธานนท์. **เรียนรู้ไมโครคอนโทรลเลอร์ MCS-51 ฉบับภาษาC.** พร้อมเลิศ หล่อวิจิตร. 2541. **DOS FOREVER**. กรุงเทพ : คอมพิวเตอร์ เอจ เทคโนโลยี.

ภาคผนวก

ภาคผนวก ก : โปรแกรมการทำงาน

1 โปรแกรมการทำงานของไมโครคอนโทรลเลอร์ AC3

Ac3_2sx.c

```
/*---------------------------------------------------------------------------*/
// Program : Data aquisition
// Description : Receive X,Y from CAMERA and write to sd card on ac3's board
// Filename : Ac3_2sx.c
// C compiler : Keil 51 V7.10
/*---------------------------------------------------------------------------*/
/* Include Section */
#include < REG51AC2.h> // Declare T89C51AC2's register (replace for AC3)
#include < delay_ac3.h> // Declare Delay library
#include < sound_ac3.h> // Declare Sound generator library
#include < lcd_2line7_ac3.h> // Declare LCD library
#include < spi_mmc.h> // Declare SD CARD library
#include < i2c.h>
/* Include for Standard C */
#include <absacc.h>
#include <assert.h>
#include <ctype.h>
#include <intrins.h>
#include <math.h>
#include <setjmp.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>


//// for DS1307 //////////////
/* * Data type **/
typedef unsigned char byte;
typedef unsigned int word;
#define TRUE 1
#define DS1307 0xD0

typedef struct{
byte sec;
byte min;
byte hour;
byte day;
byte date;
byte month;
byte year;
} RTC_TYPE;
RTC_TYPE RTC;
byte ds_hour;
byte ds_min;
byte ds_sec;
void loop_delay(int n);
void read_RTC(void);
void write_RTC(void);
void loop_delay(int n)
{
        int i,j;
        for (i=0;i<n;i++)
        for (j=0; j<100;j++);
```

```
void read_RTC(void)
{

        ds1307_read( DS1307, 0x00,&RTC.sec);
        ds1307_read( DS1307, 0x01,&RTC.min);
        ds1307_read( DS1307, 0x02,&RTC.hour);
        ds1307_read( DS1307, 0x04,&RTC.date);
        ds1307_read( DS1307, 0x05,&RTC.month);
        ds1307_read( DS1307, 0x06,&RTC.year);
        loop_delay(100);
}
void write_RTC(void)
{
        int i;
        printf( "\nSet Date/Time \n");
        printf( "Date : "); scanf( "%x",&i); RTC.date = i;
        printf( "Month : "); scanf( "%x",&i); RTC.month = i;
        printf( "Year : "); scanf( "%x",&i); RTC.year = i;
        printf( "Hour : "); scanf( "%x",&i); RTC.hour = i;
        printf( "Minutes : "); scanf( "%x",&i); RTC.min = i;
        printf( "Seconds : "); scanf( "%x",&i); RTC.sec = i;
        // Set time
        ds1307_write( DS1307, 0x00,RTC.sec);
        ds1307_write( DS1307, 0x01,RTC.min);
        ds1307_write( DS1307, 0x02,RTC.hour);
        // Set Date
        ds1307_write( DS1307, 0x04,RTC.date);
        ds1307_write( DS1307, 0x05,RTC.month);
        ds1307_write( DS1307, 0x06,RTC.year);
}
void display_RTC(void)
{
```

```
        printf( "\fDS1307 Real Time Clock\n");
        printf( "Date : ");
        if(RTC.date<10) printf( "0%bx/",RTC.date);
        else printf( "%bx/",RTC.date);
        if(RTC.month<10) printf( "0%bx/",RTC.month);
        else printf( "%bx/",RTC.month);
        if (RTC.year<10) printf( "0%bx",RTC.year);
        else printf( "%bx",RTC.year);
        printf ( "\nTime: ");
        if (RTC.hour<10) printf( "0%bx:",RTC.hour);
        else printf( "%bx:",RTC.hour);
        if (RTC.min<10) printf( "0%bx:",RTC.min);
        else printf( "%bx:",RTC.min);
        if (RTC.sec<10) printf( "0%bx:",RTC.sec);
        else printf( "%bx",RTC.sec);
}
/* User Define Function */
extern unsigned char MMCWRData[ MMC_DATA_SIZE];
extern unsigned char MMCRDData[ MMC_DATA_SIZE];
unsigned char Data_in[26]; // for data packet buffer
unsigned long Sector_Num = 0; // Innital number of sector
unsigned char XY_buff[2]; // buffer Serial in from CAMERA
sbit led = P1^0; // Led on board to show SD can be takken off
sbit sw_stop = P3^2; // Switch input for taking SD CARD off
int t_count=0; // counter 2 bytes X,Y camera

/* --- Some RS232 routine ----------*/
void Start(void)
{
        CKCON = 0x01; // Initial X2 Mode (58.9824 MHz)/* Initial MCS51 Serial Port */
        TMOD &= 0x0F; // Reset old Timer1 Mode Config
        TMOD |= 0x20; // Update Timer1 = 8 Bit Auto Reload
```

```
	SCON = 0x50; // Serial Port Mode 1 (N,8,1)
	ES = 0; // Disable Serial Interupt
	ET1 = 0; // Disable Timer1 Interrupt
	PCON &= 0x7F; // SMOD1 = 0 (Disable Double Baudrate)
	TH1 = 0xF0; // Setup Timer1 Baudrate 9600BPS / 58.9824 MHz
	TL1 = 0xF0;
	TR1 = 1; // Start Timer1 Generate Baudrate
	TI = 1; // Set TI to send First char of UART/* Print Open Message to PC via RS232 */
	printf ( "\n");
	printf ( "=== Hello World From ET-BASE51 AC3 V1.0 ===\n");
	printf ( " Test SD card Program \n");
	printf ( "-----------------------------------------\n");
}
void Start1(void)
{
	CKCON = 0x01; // Initial X2 Mode (58.9824 MHz)/* Initial MCS51 Serial Port */
	TMOD &= 0x0F; // Reset old Timer1 Mode Config
	TMOD |= 0x20; // Update Timer1 = 8 Bit Auto Reload
	SCON = 0x50; // Serial Port Mode 1 (N,8,1)
	ET1 = 0; // Disable Timer1 Interrupt
	ET0 = 0;
	PCON &= 0x7F; // SMOD1 = 0 (Disable Double Baudrate)
	TH1 = 0xF0; // Setup Timer1 Baudrate 9600BPS / 58.9824 MHz
	TL1 = 0xF0;
	EA = 0x1; ES = 0x1; // set EA, ES
	TR1 = 1; // Start Timer1 Generate Baudrate
}
/*****************************************************************************
Function :serial receiver / transmitter interrrupt
Parameters : nothing
Return : nothing
*****************************************************************************/
```

```
seial () interrupt 4 using 2
{
if(RI)
{
XY_buff[ t_count] = SBUF; RI = 0;
t_count++;
}
TI = 0;
}
/*************************************************************************
B_write : Write Data (Hex) to MMCWRData
Offset : Address (Byte)
Data_size : Size of Data
*p : Start addres of Data to be transered
*************************************************************************/
void B_write(unsigned long Offset,int Data_size,unsigned char *p)
{
        while( Data_size--) // Check data pointer = 0?
        {
        MMCWRData[Offset]=(*p); // Send data to MMCWRData
        Offset++; p++; // Increase address both 1 time
        }
}
/*************************************************************************/
/* *************** Function int to ASCII ********************************/
/*************************************************************************/
void inttoascii(unsigned char * out_char,int value)
{
        char index=0,buff[6]; // For keep string send to LCD
        buff[0] = (value/10000) | 0x30; // Convert data to ASCII 10000'th
        buff[1] = ((value%10000)/1000) | 0x30; // Convert data to ascii 1000'th
        buff[2] = (((value%10000)%1000)/100) | 0x30; // Convert data to ascii 100'th
```

```
        buff[3] = ((((value%10000)%1000)%100)/10)|0x30;// Convert data toascii10'th
        buff[4] = ((((value%10000)%1000)%100)%10)|0x30;// Convert data to ascii1'th
        buff[5] = 0; // End of data
        if(buff[0]== 0x30) // if data 1'th = 0 none display
        {index = 1;} // Shift to display 2'th
        if(buff[0]== 0x30 && buff[1]== 0x30) // if data 1'th = 0 and 2'th = 0 none display
        {index = 2;} // Shift to display 3'th
        if(buff[0]== 0x30 && buff[1]== 0x30 && buff[2]== 0x30)
        // if data1'th=0,2'th = 0 and 3'th = 0 none display
        {index = 3;} // Shift to display 4'th
        if(buff[0]== 0x30 && buff[1]== 0x30&&buff[2]==0x30&&buff[3]==0x30) // if
        data1'th=0, 2'th = 0, 3'th = 0 and 4'th = 0 none display
        {index = 4;} // Shift to display 5'th
        * out_char=buff[2]; out_char++;
        * out_char=buff[3]; out_char++;
        * out_char=buff[4]; out_char++;
}
void Bcd_to_int(char f_char[2],unsigned char ch_in)
{
        f_char[0] = ch_in;
        f_char[0] >>=4;
        f_char[1] = ch_in;
        f_char[1] &= 0x0f;
}
/*=============================================================*/
/*==============MAINPROGRAM=====================*/
/*=============================================================*/
void main(void)
{
        int i;
        unsigned char Data_num = 0; // amount of data record/sector
        unsigned long Sector_num = 0;
```

```
unsigned char Char_temp[3];
unsigned char ch, ch_temp[2];
s_bit = 0x00; // off Speaker
delay_ms(100); // Delay power up
i2c_init(); // initial i2c interface
lcd_init(); // Initial LCD in 4-bit mode 1 line
lcd_putstr( line1, "Status: "); // Show Character "Hello" on First line
lcd_putstr( line2, " Initial"); // Show Character "Sd Test" on Seconds line
beep();

//// Input Time ///////////////////////
if (! sw_stop)
{
        lcd_init();
        Start();
        lcd_putstr( line1, "Setting ");
        lcd_putstr( line2, " DS1307 ");
        beep();
        printf( "\fSet Real Time Clock (y or n):");
        ch = getchar();
        if(ch== 'y' || ch == 'Y')
        {
                write_RTC();
        }
        while(TRUE)
        {
                read_RTC();
                display_RTC();
        }
}
////////////////////// end input Time ///////////////////////
for(i=0;i< MMC_DATA_SIZE;i++)// Clear buffer
```

```
{
        MMCWRData[i]= 0x20;
        MMCRDData[i]= 0x20;
}
// Start sector number
SPI_Init(); /* Initialize SPI for MMC card */
if((i= mmc_init()) != 0)
{
        lcd_clear();
        lcd_putstr( line1, " Can not"); // Show Character "ERROR"
        lcd_putstr( line2, "Initial ");
while(1);
}
lcd_clear();
lcd_putstr( line1, "Connect."); // Show success
lcd_putstr( line2, ".CAMERA.");
delay_ms(1000);
Start1();
while( Sector_Num < 16) // now test only 16 sectors (is 2 files)
{
        for(i=0;i< MMC_DATA_SIZE;i++) MMCWRData[i]= 0x20; // Clear buffer
        while( Data_num < 19 & sw_stop) // write 1 sector contain 19 packets (1
        packet=26 bytes)
        {
                lcd_clear();
                while( t_count<2 & sw_stop)
                { ///*********************************///
                        lcd_putstr( line1, "Waitting"); ///*** WAIT FOR
                        DATA(2bytes)***///
                        lcd_putstr( line2, "for DATA");
                        ///*********************************///
                }
```

```
lcd_putstr( line1, ".DATA...");
lcd_putstr( line2, ".ADDING.");
beep();
if(! sw_stop){for (i = 0;i<26;i++){ Data_in[i]= 0x20;}}
else
{
        inttoascii( Char_temp,XY_buff[0]);///*************///
        for(i=0;i<3;i++){ Data_in[i]= Char_temp[i];}
        Data_in[3]= 0x20; /// Insert Data ///
        inttoascii( Char_temp, XY_buff[1]);///*************///
        for(i=4;i<7;i++){ Data_in[i]= Char_temp[i-4];}
        Data_in[7]= 0x20;
        /////////* Time_in *//*** DS1307 ***/////////////////////////////////
        read_RTC();
        ch = 0x3f & RTC.hour;
        Bcd_to_int( ch_temp,ch);
        inttoascii( Char_temp, ch_temp[0]);
        Data_in[8]= Char_temp[2];
        inttoascii( Char_temp, ch_temp[1]);
        Data_in[9]= Char_temp[2];
        Data_in[10]= 0x20;
        ch = 0x7f & RTC.min;
        Bcd_to_int( ch_temp,ch);
        inttoascii( Char_temp, ch_temp[0]);
        Data_in[11]= Char_temp[2];
        inttoascii( Char_temp, ch_temp[1]);
        Data_in[12]= Char_temp[2];
        Data_in[13]= 0x20;
        ch = 0x7f & RTC.sec;
        Bcd_to_int( ch_temp,ch);
        inttoascii( Char_temp, ch_temp[0]);
        Data_in[14]= Char_temp[2];
```

```
                inttoascii( Char_temp, ch_temp[1]);
                Data_in[15]= Char_temp[2];
                Data_in[16]= 0x20;
                ch = 0x3f & RTC.date;
                Bcd_to_int( ch_temp,ch);
                inttoascii( Char_temp, ch_temp[0]);
                Data_in[17]= Char_temp[2];
                inttoascii( Char_temp, ch_temp[2]);
                Data_in[18]= Char_temp[2];
                Data_in[19]= 0x20;
                ch = 0x1f & RTC.month;
                Bcd_to_int( ch_temp,ch);
                inttoascii( Char_temp, ch_temp[0]);
                Data_in[20]= Char_temp[2];
                inttoascii( Char_temp, ch_temp[2]);
                Data_in[21]= Char_temp[2];
                Data_in[22]= 0x20;
                ch = RTC.year;
                Bcd_to_int( ch_temp,ch);
                inttoascii( Char_temp, ch_temp[0]);
                Data_in[23]= Char_temp[2];
                inttoascii( Char_temp, ch_temp[1]);
                Data_in[24]= Char_temp[2];
                Data_in[25]= 0x20;
                ////////////////////////////////////////////////////////////////////////////////
            }
            B_write(26* Data_num,26, Data_in);//Appending packet(26bytes)
            for (i = 0;i<26;i++){ Data_in[i]= 0x20;} //clear data_in
            delay_ms(500);
            t_count=0; // Start data again
            Data_num++;
        }
```

```
            EA = 0; // Disable interrupt
            Data_num = 0;
            Sector_num = 0x210+ Sector_Num;
            if( mmc_write_block( Sector_num)==0) // Write To SD (one Sector)
            {
                  lcd_clear();
                  lcd_putstr( line1, "Writing.");
                  lcd_putstr( line2, "....OK..");
                  delay_ms(1000);
            }
            else
            {
                  Chip_select = 0x01; // Off Chip
                  lcd_clear();
                  lcd_putstr( line1, ".Error..");
                  lcd_putstr( line2, ".WR.....");
                  while(1);
            }
      Sector_Num++;
      EA = 1; // Enable interrupt
      if (! sw_stop)
      {
            lcd_putstr( line1, ".Safely.");
            lcd_putstr( line2, ".Remove.");
            led = 0x00; // led ON
            delay_ms(500);
            beep();
            while(1);
      }
      }
}
```

2 โปรแกรมการทำงานของ CMUCAM

CAMTST02.BSX

```
'{$STAMP BS2sx}
RcvData VAR Byte(10)
baud CON 240
tx CON 1
rx CON 2
tx1 CON 6
rx1 CON 7
led CON 8
i VAR Byte
X_data VAR Word
Y_data VAR Word
X_d VAR Byte
Y_d VAR Byte
' Pause 1 second for CMUcam startup
PAUSE 1000
' Send "reset" to sync CMUcam and Stamp
DEBUG "Start....",CR
SEROUT tx, baud, ["RS", CR]
SERIN rx, baud, [WAIT (":")]
DEBUG "Reset....",CR
PAUSE 1000
' Track LED on
SEROUT tx, baud, ["L1 1",CR]
'SERIN rx, baud, [WAIT (":")]
PAUSE 100
' Turn on auto adjust for 5 seconds
'DEBUG "Auto Adjusting..." , CR
SEROUT tx, baud, ["CR 18 44",CR]
```

```
'SERIN rx, baud, [WAIT (":")]
PAUSE 1000
' Pause 5 seconds for CMUcam to auto adjust to lighting conditions
DEBUG "Adjust lighting ...",CR
PAUSE 5000
' Turn off auto adjust
SEROUT tx, baud, ["CR 18 44 19 32",CR]
'SERIN rx, baud, [WAIT (":")]
'DEBUG "Finish..", CR
PAUSE 1000
' Track LED auto mode
SEROUT tx, baud, ["L1 2",CR]
'SERIN rx, baud, [WAIT (":")]
PAUSE 1000
DEBUG "Place color target", CR
HIGH led
' Give user time to place color target close in front of camera
PAUSE 5000
' Send command - Set poll mode - only sends one return packet -
' of data after each command - reduces data flow
DEBUG "Finished..." , CR
SEROUT tx, baud, ["PM 1",CR]
'SERIN rx, baud, [WAIT (":")]
PAUSE 1000
' Send command - Set raw data mode - also suppress Ack:/Nak: to -
' further reduce serial data
SEROUT tx, baud, ["RM 3",CR]
PAUSE 100
LOW led
' Track Window command looks at the center of CMUcam image -
' grabs the color information and sends to the Track Color function
' Send command - Track window
```

```
SEROUT tx, baud, ["TW",CR]
' Display the S Statisics packet from TW command
SERIN rx, baud, [STR RcvData\8]
' Raw mode S packet data format:
' 0 Byte always 255
' 1 Byte always Character S
' 2 Byte Red Mean
' 3 Byte Green Mean
' 4 Left Blue Mean
' 5 Left Red Deviation
' 6 Right Green Deviation
' 7 Right Blue Deviation
' Display all returned camera S Statistics packet data to PC debug screen
DEBUG "Red Mean ",DEC RCVData(2) ,CR
DEBUG "Green Mean ",DEC RCVData(3) ,CR
DEBUG "Blue Mean ",DEC RCVData(4) ,CR
DEBUG "Red Deviation ",DEC RCVData(5) ,CR
DEBUG "Green Deviation ",DEC RCVData(6) ,CR
DEBUG "Blue Deviation ",DEC RCVData(7) ,CR
DEBUG " ", CR
' Ignore the first M packet from TW
PAUSE 2000
Main:
PAUSE 1000
X_data = 0
Y_data = 0
HIGH led
FOR i = 0 TO 5
SEROUT tx, baud, ["TC",CR]
SERIN rx, baud, [STR RcvData\10]
DEBUG "."
X_data = X_data+RCVData(2)
```

```
Y_data = Y_data+RCVData(3)
NEXT
LOW led
X_data = X_data/6
Y_data = Y_data/6
X_d = X_data
Y_d = Y_data
DEBUG "X = ",DEC X_data," : Y = ",DEC Y_data,CR
SEROUT tx1,baud, [X_d]
SEROUT tx1,baud, [Y_d]
PAUSE 5000

GOTO Main
```

3 โปรแกรมการทำงานของโปรแกรมแมทแลป

Data.m

```
%Read data(project)
clear all; %close all;
load -ascii text0.txt;
[s1 s2]=size(text0);
% Get value x,y
j=0;
for i=1 : +8 :s2
j=j+1;
x(j)=text0(i);y(j)=text0(i+1);
end
% calculate the distance
for i=1 :(s2/8-1)
x1(i+1)=abs(x(i)-x(i+1));
y1(i+1)=abs(y(i)-y(i+1));
end
x1(1)=0;y1(1)=0;
out=sqrt(x1.^2+y1.^2);
for i=2 : (s2/8)
out(i)=out(i)+out(i-1);
end
plot(out,'linewidt',4);
ylabel('Distance in Cm','FontSize',18);
xlabel('Time(5 Seconds/sample)','FontSize',18);
out(s2/8)=(out(s2/8)/10);
title(['Total distance is ',num2str(out(s2/8)),' Cm,Total of time is ',num2str(s2/8*5),
' Secs'],'BackgroundColor',[.7 .9 .7],'FontSize',16);%
pause;
clear all; close all;
```

ภาคผนวก ข : จอแสดงผล LCD

ปัจจุบัน LCD เป็นที่นิยมกันเป็นอย่างมาก สำหรับการแสดงผลในเครื่องมือเครื่องใช้ต่างๆ ทั้งนี้เนื่องจากมีความเหมาะสมในหลายๆด้าน เช่น การใช้กระแสต่ำ , สามารถแสดงผลเป็นตัวอักษรและตัวเลขหรือแสดงเป็นกราฟฟิก (เฉพาะรุ่น)

ขาสัญญาณของ LCD

PIN	SYMBOL	LEVEL	FUNCTION
1	Vss	-	0 V GND
2	Vcc	-	+5 V Power Supply
3	Vee	-	+V For Liquid Crystal Drive
4	RS	H/L	Register Select H: Data Input L: Instruction Input
5	R/W	H/L	H: Data Read L: Data Write
6	E	H	Enable Signal (L -> H)
7	DB 0	H/L	Data Bus Bit 0
8	DB 1	H/L	Data Bus Bit 1
9	DB 2	H/L	Data Bus Bit 2
10	DB 3	H/L	Data Bus Bit 3
11	DB 4	H/L	Data Bus Bit 4
12	DB 5	H/L	Data Bus Bit 5
13	DB 6	H/L	Data Bus Bit 6
14	DB 7	H/L	Data Bus Bit 7

ชุดคำสั่งควบคุมและการแสดงข้อความ

ขาสัญญาณ Vee มีไว้สำหรับกำหนดความเข้มของตัวอักษรโดยถ้าต่อกับ GND จะมีความเข้มสูงสุด แต่ถ้าต่อกับ Vcc จะมีความเข้มต่ำสุด ปกติ LCD รุ่นธรรมดา อาจจะต่อกับ GND ไว้เลยก็ได้ ไม่ต้องใส่ VR ให้สิ้นเปลือง แต่ถ้าเป็นรุ่น STN (มุมมองกว้าง) ให้ใช้ R 2 K ต่อลง GND อีกทีเพื่อให้ความเข้มมีความเหมาะสม การเขียนหรืออ่านข้อมูลก็คือการกำหนดคุณสมบัติต่างๆในการใช้งานของ LCD ตามชุดคำสั่งควบคุมและรวมไปถึงการเขียนข้อมูลที่เป็นข้อความเพื่อให้ปรากฏบนแผงแสดงด้วย โดยมีรายละเอียดตามตารางต่อไปนี้

INSTRUCTION	RS	R/W	DATA BIT								EXE TIME (MicroS)
			7	6	5	4	3	2	1	0	
CLEAR DISPLAY	0	0	0	0	0	0	0	0	0	1	1640
CURSOR AT HOME	0	0	0	0	0	0	0	0	1	*	1640
ENTRY MODE SET	0	0	0	0	0	0	0	1	I/D	S	40
DISPLAY ON/OFF	0	0	0	0	0	0	1	D	C	B	40
DISPLAY SHIFT	0	0	0	0	0	1	S/C	R/L	*	*	40
FUNCTION SET	0	0	0	0	1	DL	N	F	*	*	40
SET CGRAM ADD	0	0	0	1	CGRAM ADDRESS						40
SET DDRAM ADD	0	0	1	DDRAM ADDRESS							40
BUSY,ADD. READ	0	1	BF	ADDRESS							0
CGRAM,DDRAM WR	1	0	WRITE DATA								40
CGRAM,DDRAM RD	1	1	READ DATA								40

รายละเอียดของแต่ละคำสั่ง

1. CLEAR DISPLAY

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

สำหรับการ Clear Display โดยจะทำการเขียนตัวอักษร Space ลงไปใน DDRAM ทั้งหมดและกำหนดค่า DDRAM Address ให้เป็น 0 พร้อมทั้ง Cursor จะกลับไปที่ตำแหน่งซ้ายบนสุดของจอภาพ

2 CURSOR AT HOME

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

สำหรับกำหนดค่า DDRAM Address ให้เป็น 0 พร้อมทั้ง Cursor จะไปอยู่ที่ตำแหน่งซ้ายบนสุดของจอภาพโดยที่ข้อมูลใน DDRAM ไม่มีการเปลี่ยนแปลง

3. ENTRY MODE SET

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

การกำหนดค่า I/D และ S นี้ให้กำหนดก่อนการเขียนข้อมูลใน DDRAM และเมื่อกำหนดแล้วจะต้องไม่ใช้คำสั่ง Clear Display อีก

I/D = 0 กำหนดทิศทางของเคอร์เซอร์ (Cursor) และ DDRAM ให้เป็นแบบดีครีเมนท์ (Decrement)

I/D = 1 กำหนดทิศทางของเคอร์เซอร์และ DDRAM ให้เป็นแบบอินครีเมนท์ (Increment)

S = 0 เมื่อเขียนข้อมูลแล้วตัวเคอร์เซอร์จะถูกเลื่อนไปตามทิศทางของค่า I/D

S = 1 เมื่อเขียนข้อมูลแล้วตัวเคอร์เซอร์จะอยู่กับที่และตัวอักษรจะถูกดันไปตามทิศทางของค่า I/D

4. DISPLAY ON/OFF

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

D = 0 กำหนดให้ Off แสดงผล(Display)

D = 1 กำหนดให้ On แสดงผล

C = 0 กำหนดให้ Off เคอร์เซอร์

C = 1 กำหนดให้ On เคอร์เซอร์โดยเคอร์เซอร์จะเป็นเส้นขีดใต้ตัวอักษร

B = 0 กำหนดให้ไม่มีการกระพริบที่ตำแหน่งเคอร์เซอร์

B = 1 กำหนดให้มีการกระพริบที่ตำแหน่งเคอร์เซอร์

5. DISPLAY SHIFT

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	*	*

S/C = 0 กำหนดให้เลื่อนเคอร์เซอร์ตามทิศทาง R/L ไป 1 ตำแหน่ง

S/C = 1 กำหนดให้เลื่อนข้อความบนแผงแสดงตามทิศทาง R/L ไป 1 คอลัมน์ (Column) (เลื่อนทุกบรรทัด)

R/L = 0 กำหนดให้มีทิศทางไปทางซ้าย

R/L = 1 กำหนดให้มีทิศทางไปทางขวา

6. FUNCTION SET

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	*	*

DL = 0 กำหนดให้การติดต่อกับ LCD Module เป็นแบบ 4 บิต

DL = 1 กำหนดให้การติดต่อกับ LCD Module เป็นแบบ 8 บิต

N = 0 กำหนดจำนวนบรรทัดแบบ 1/8 Duty และ 1/11 Duty

N = 1 กำหนดจำนวนบรรทัดแบบ 1/16 Duty

F = 0 กำหนดให้ตัวอักษรเป็นแบบ 5*7 Dots

F = 1 กำหนดให้ตัวอักษรเป็นแบบ 5*10 Dots

7. SET CGRAM ADDRESS

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	CGRAM ADDRESS					

สำหรับการกำหนด Address ของ CGRAM เมื่อได้ทำการกำหนดไว้แล้วการอ่านและเขียนข้อมูลที่ต่อจากนี้จะเป็นไปตามแอดเดรสที่กำหนดทันที

8. SET DDRAM ADDRESS

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	DDRAM ADDRESS						

สำหรับการกำหนดแอดเดรสของ DDRAM เมื่อได้ทำการกำหนดไว้แล้วการอ่านและเขียนข้อมูลที่ต่อจากนี้จะเป็นไปตามแอดเดรสที่กำหนดทันที ตำแหน่งของแอดเดรสในแต่ละรุ่นจะมีความแตกต่างกันบ้าง เพราะจำนวนตัวอักษรต่อบรรทัดไม่เท่ากัน ในโครงงานนี้ใช้ LCD รุ่น DMC 161

รุ่น DMC 161

80	81	82	83	84	85	86	87	C0	C1	C2	C3	C4	C5	C6	C7

9. BUSY FLAG AND ADDRESS READ

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	ADDRESS						

สำหรับการอ่านค่า BF (Busy Flag) ซึ่งบอกถึงความพร้อมของ LCD Module ในการรับข้อมูล ถ้า BF = 0 หมายความว่าพร้อมที่จะรับข้อมูลต่อไปได้ แต่ถ้า BF = 1 หมายความว่ายังไม่พร้อม

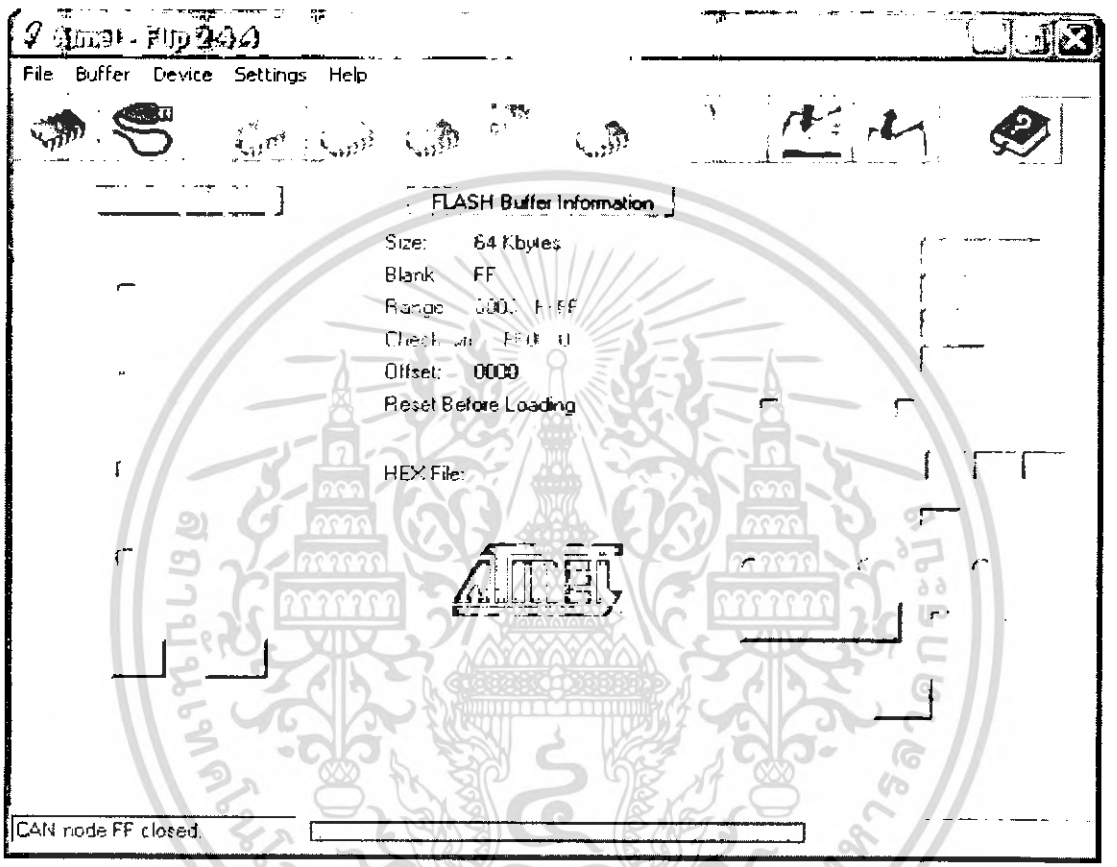
ภาคผนวก ค : โปรแกรม FLIP 2.4.4

โปรแกรม FLIP (Flexible In-system Programmer) เป็นโปรแกรมสำหรับพัฒนาระบบของไมโครคอนโทรลเลอร์ของ ATMEL โดยสามารถใช้สนับสนุนการพัฒนาโปรแกรมของไมโครคอนโทรลเลอร์ตระกูลเอ็มซีเอส 51 ในกลุ่มที่ใช้การพัฒนาแบบ ISP ซึ่งรวมถึงเบอร์ AT89C51AC3 ด้วยโดยโปรแกรมจะทำงานภายใต้ระบบปฏิบัติการของ Window9X/Me/NT/ 2000 และ Window XP โดยสนับสนุนการเชื่อมต่อกับระบบฮาร์ดแวร์ที่ใช้การเชื่อมต่อแบบ RS-232 หรือ CAN หรือ USB ซึ่งวิธีการเชื่อมต่อของโปรแกรม FLIP กับระบบฮาร์ดแวร์ของไมโครคอนโทรลเลอร์นั้น จะขึ้นอยู่กับความสามารถของตัวไมโครคอนโทรลเลอร์ที่จะนำมาทำการพัฒนาว่าสามารถใช้การติดต่อสื่อสารด้วยวิธีใดได้บ้าง แต่สำหรับไมโครคอนโทรลเลอร์เบอร์ AT89C51AC3 นั้นจะสามารถใช้การเชื่อมต่อผ่านทางพอร์ตอนุกรม RS232 เท่านั้นไม่สามารถเชื่อมต่อผ่านระบบการสื่อสารของ CAN หรือ USB ได้ โดยโปรแกรม FLIP จะใช้สำหรับดาวน์โหลดข้อมูลให้กับหน่วยความจำของไมโครคอนโทรลเลอร์ที่ทำงานใน Monitor Mode เพื่อให้ผู้ใช้สั่งจัดการกับหน่วยความจำภายในตัว CPU ไม่ว่าจะเป็นการสั่งลบข้อมูล(Erase) สั่งตรวจสอบข้อมูลในหน่วยความจำ(Blank Check) สั่งโปรแกรมข้อมูลให้กับหน่วยความจำโปรแกรมของ CPU (Program) สั่งเปรียบเทียบข้อมูลจากบัฟเฟอร์กับหน่วยความจำในตัว CPU (Verify) หรือสั่งอ่านข้อมูลจากหน่วยความจำของ CPU (Read) เป็นต้นซึ่งเมื่อต้องการให้โปรแกรม FLIP ติดต่อกับ CPU ใน Monitor Mode นั้น จะต้องสั่ง Reset ให้ CPUเข้าทำงานใน Monitor Mode ก่อนเสียก่อน ซึ่งหลักการสำหรับ Reset ให้ CPU เข้าทำงานใน Monitor Mode จะต้องกำหนดให้ขาสัญญาณ PSEN มีสภาวะเป็น “0” ในขณะที่ CPU หลุดพ้นจากสภาวะของการ Reset ซึ่งตามปรกติแล้วหลังการ Reset ทุกครั้ง CPU จะตรวจสอบสภาวะของขาสัญญาณ PSEN ว่าเป็น “0” หรือไม่ถ้าไม่ใช่ก็จะทำงานในโหมดการทำงานปรกติแต่ถ้าใช่ก็จะตรวจสอบสถานะของสัญญาณอื่นๆที่เกี่ยวข้องกับการทำงานใน Monitor Mode ถ้าเงื่อนไขอื่นๆถูกต้องก็จะเข้าทำงานใน Monitor Mode ทันที สำหรับบอร์ดรุ่น ET-BASE51 AC3 (AT89C51AC3) นั้น การที่จะสั่ง Reset ให้ CPU ของ ATMEL เข้า ทำงานใน Monitor Mode เพื่อสั่ง Download HEX File จาก PC ให้กับบอร์ดจะสามารถทำได้ 2 แบบ คือ

- การ Download แบบ Manual โดยวิธีการนี้จะใช้กับสาย RS232 แบบ 4 Pin ร่วมกับ Switch PSEN และ Switch RESET ในการสั่งดาวน์โหลด
- การ Download แบบ Auto โดยวิธีการนี้ จะใช้สาย ET-DOWNLOAD แบบ 5 Pin ในการสั่งดาวน์โหลด

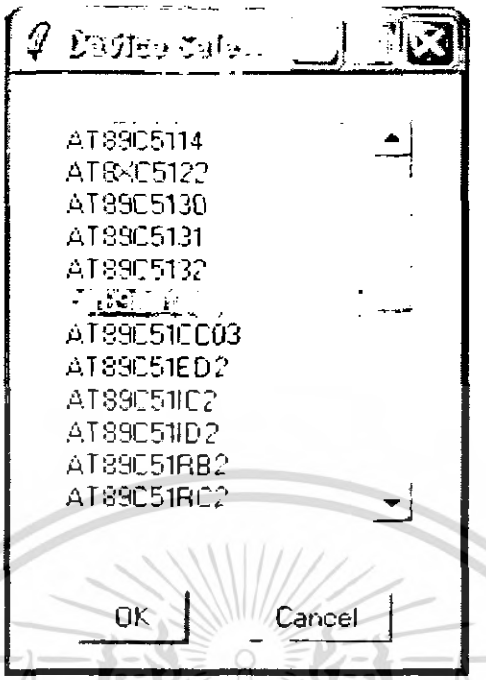
ลำดับขั้นตอนการดาวโหลด HEX File ด้วยโปรแกรม FLIP 2.4.4 แบบ Manuel

1. ต่อสายสัญญาณ RS232 จาก Com Port ของเครื่องคอมพิวเตอร์ PC เข้ากับขั้ว RS232 แบบ 4 Pin ของบอร์ด
2. จ่ายไฟเลี้ยงวงจรให้บอร์ด ซึ่งจะสังเกตุเห็น LED แสดงสถานะของ PWR สีแดงติดสว่างอยู่
3. สั่ง Run โปรแกรม FLIP V2.4.4 ซึ่งจะได้ผลดังรูป



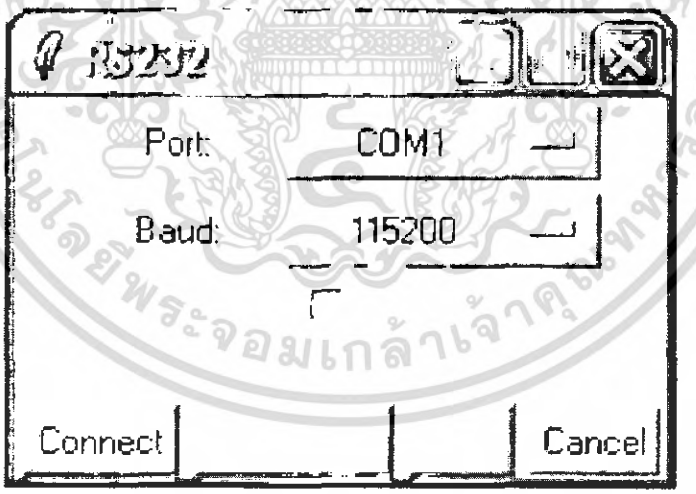
รูปแสดงโปรแกรม FLIP 2.4.4

4. สั่งเลือกกำหนดเบอร์ของ MCU ที่ติดตั้งไว้ในบอร์ด โดยเลือก Device ⟶ Select ซึ่งต้องเลือกกำหนดให้ตรงกับที่ทำการติดตั้งไว้จริงๆในบอร์ดด้วย ดังตัวอย่าง (AT89C51AC3)



รูปแสดง การเลือกกำหนดเบอร์ CPU ของ ET-BASE51 AC3 (AT89C51AC3)

5. คลิกเมาส์ที่คำสั่ง Setting ⟶ Communication ⟶ RS232 จากนั้นเลือกกำหนด Comport ให้ตรงกับที่ต่อสายไว้จริง ดังรูป (ในตัวอย่างใช้ Com1)



6. ทำการรีเซ็ต MCU ให้เข้าทำงานใน Monitor โดยมีลำดับขั้นตอนดังนี้

a) กดสวิตช์ PSEN ค้างไว้เพื่อกำหนดสถานะขาสัญญาณ PSEN ให้เป็น "0"

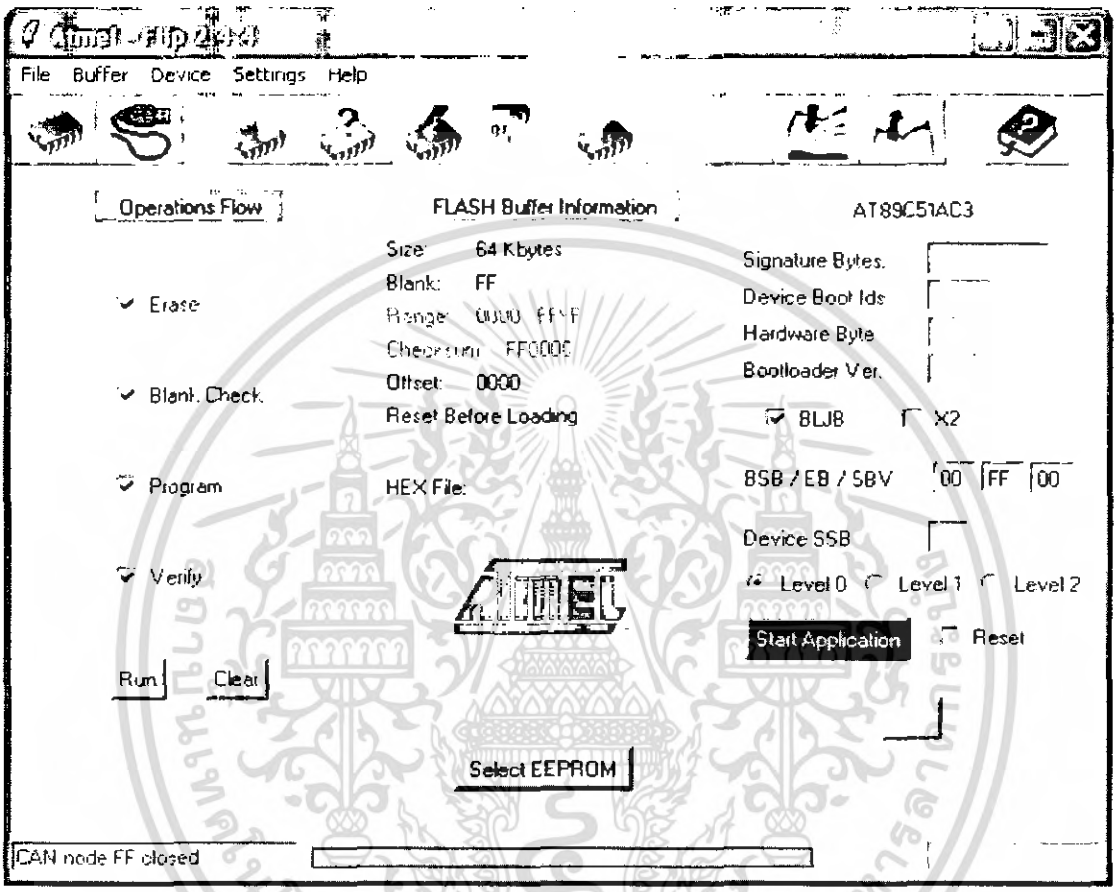
b) กดสวิตช์ RESET เพื่อส่งสัญญาณ RESET ให้กับ CPU โดยสวิตช์ PSEN ต้องกดค้างอยู่เช่นเดิม

c) ปล่อยสวิตช์ RESET เพื่อปล่อยให้ CPU พ้นจากสภาวะการ Reset (สวิตช์ PSEN ยัง

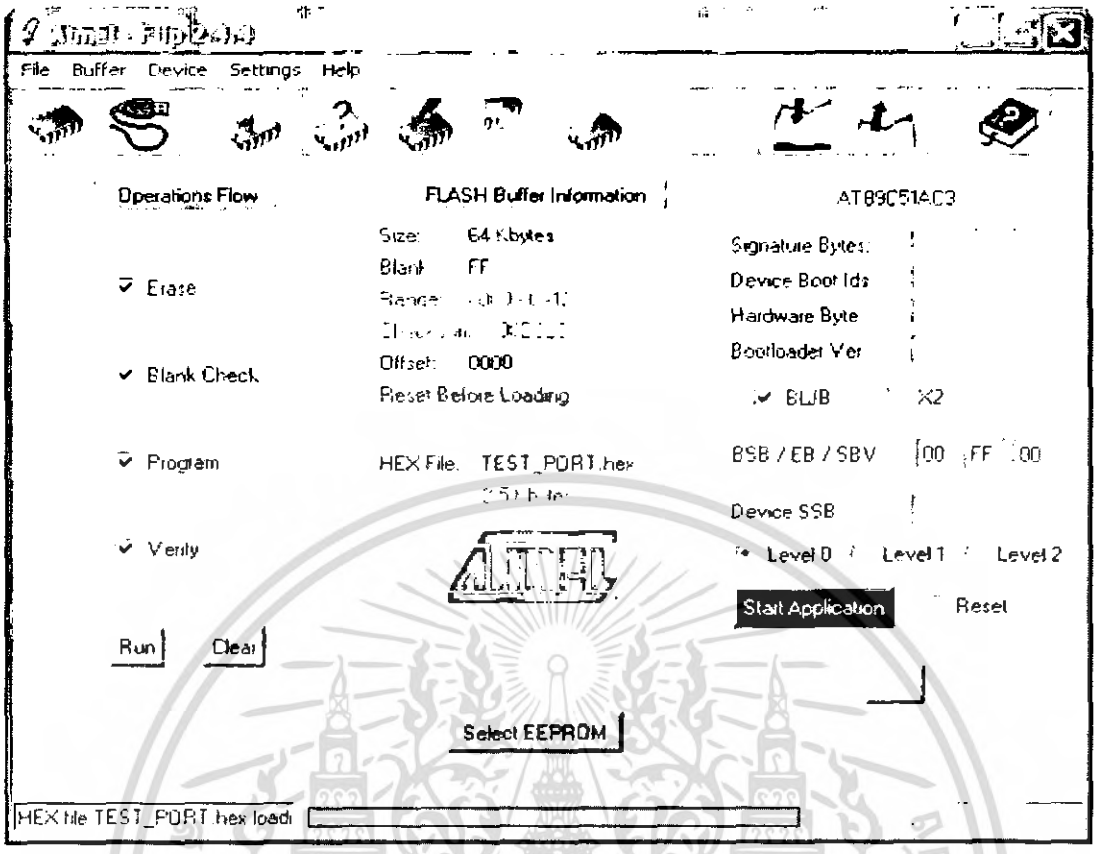
กดค้างอยู่)

d) ปล่อยสวิตช์ PSEN เป็นลำดับสุดท้าย

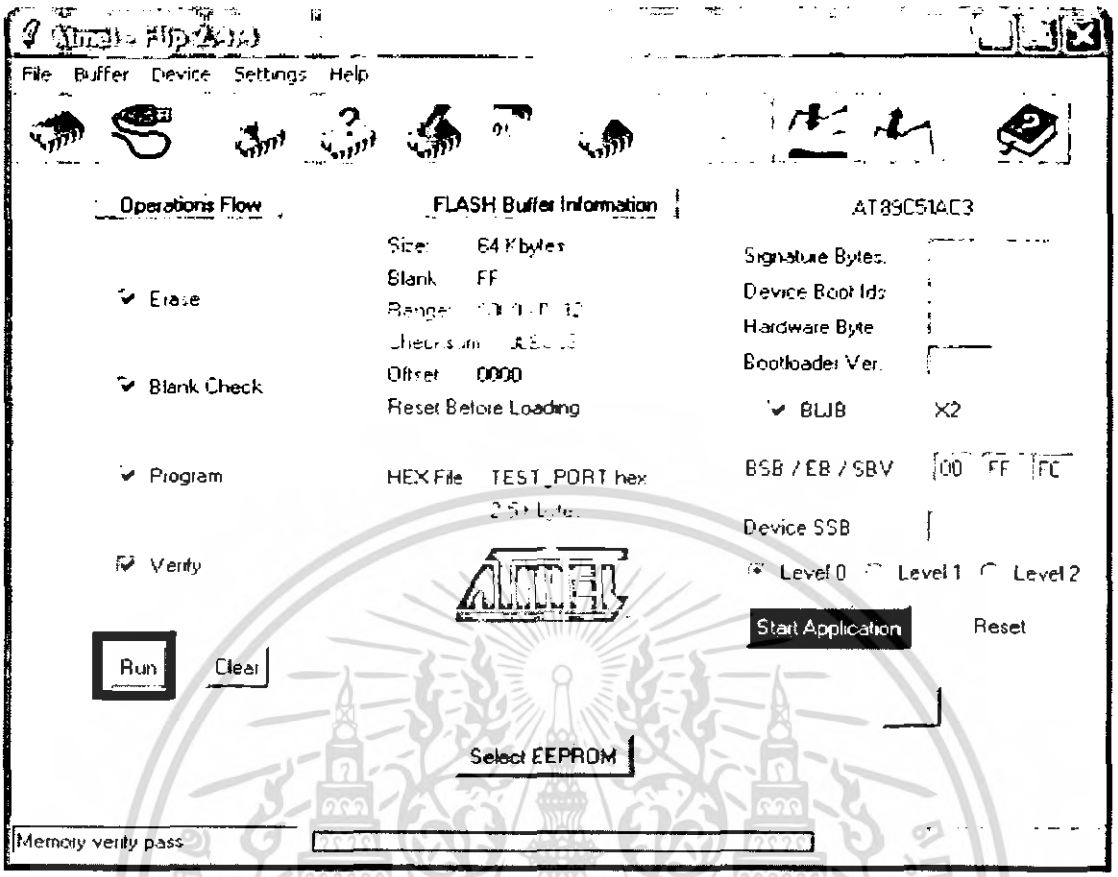
7. คลิกเมาส์ที่ปุ่ม Connect เพื่อทำการติดต่อสื่อสารกับ MCU ใน Monitor Mode ซึ่งจะได้ผลดังรูป



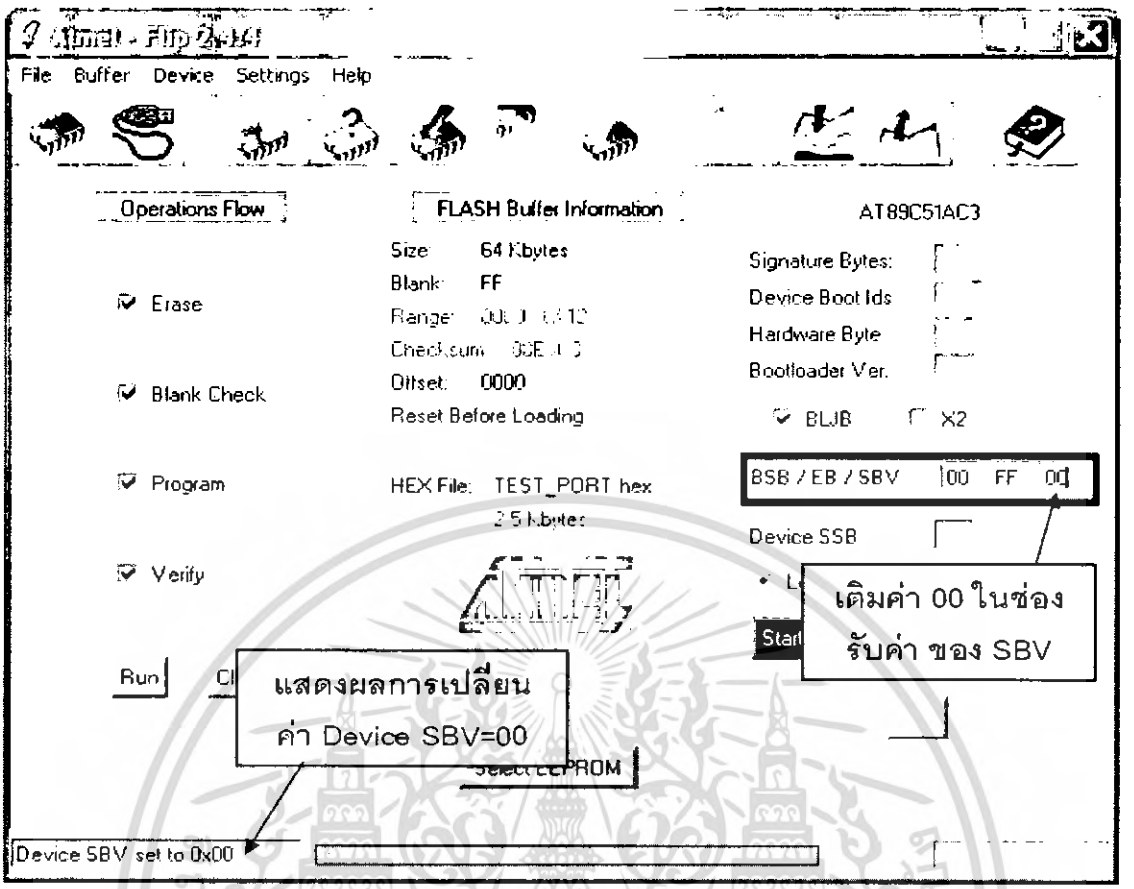
8. สั่งเปิด Hex File ที่ต้องการจะดาวน์โหลดให้กับ MCU มารอไว้ในบัฟเฟอร์ ของโปรแกรม FLIP โดยใช้คำสั่ง "File ⟶ Load Hex File..."



9. คลิกเมาส์ที่หน้าตัวเลือกคำสั่งใน Tab ของ Operation Flow ให้ครบทุกคำสั่ง ซึ่งได้แก่ Erase, Bank Check, Program, Verify จากนั้นคลิกเมาส์ที่ปุ่มคำสั่ง Run และรอจนการทำงานของโปรแกรมเสร็จเรียบร้อยดังรูป



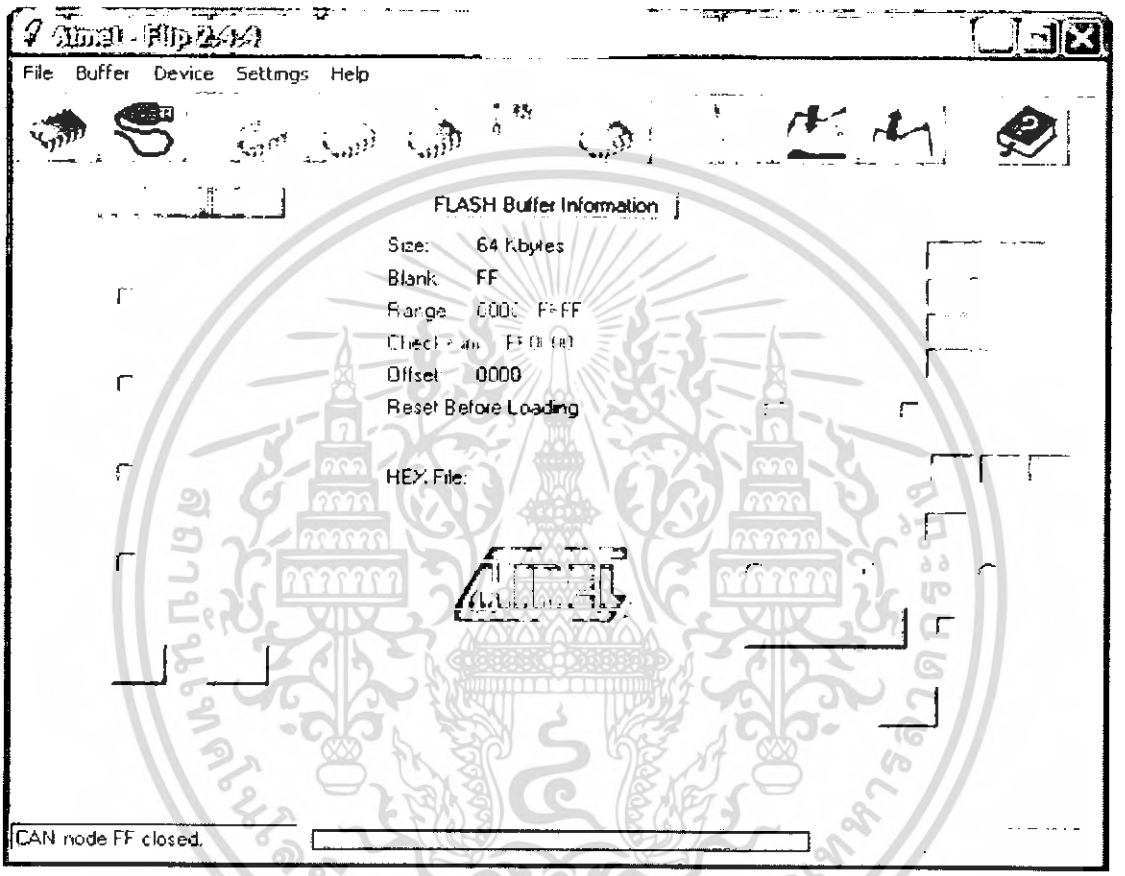
10. ตรวจสอบค่า Device BSB และ SBV ว่ามีค่าเป็น 00 ทั้งหมดแล้วหรือยัง ซึ่งถ้ายังไม่เป็น 00 ให้ทำการแก้ไขค่าให้เป็น 00 โดยคลิกเมาส์ในช่องตัวเลขแล้วพิมพ์ค่า 00 แทนที่ลงไปทั้ง 2 ช่องดังรูป



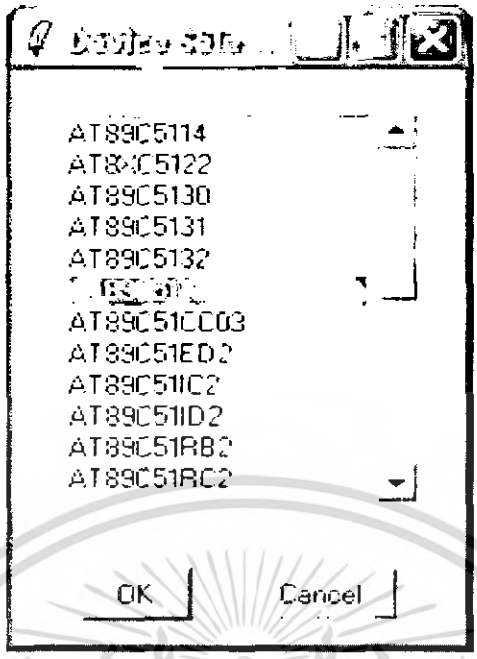
11. ทำการกดสวิตช์ Reset ให้กับบอร์ดเพื่อให้บอร์ดเริ่มต้นทำงานตามโปรแกรมที่ได้ทำการดาวน์โหลดไปให้ ซึ่งถ้าไม่เกิดความผิดพลาดใดๆจะเห็น MCU เริ่มต้นทำงานทันที

ลำดับขั้นตอนการดาวโหลด HEX File ด้วยโปรแกรม FLIP 2.4.4 แบบ Auto

1. ต่อสายสัญญาณ RS232 จาก Com Port ของเครื่องคอมพิวเตอร์ PC เข้ากับขั้ว RS232 แบบ 4 Pin ของบอร์ด
2. จ่ายไฟเลี้ยงวงจรให้บอร์ด ซึ่งจะสังเกตุเห็น LED แสดงสถานะของ PWR สีแดงดิดสว่างอยู่
3. สั่ง Run โปรแกรม FLIP V2.4.4 ซึ่งจะได้ผลดังรูป

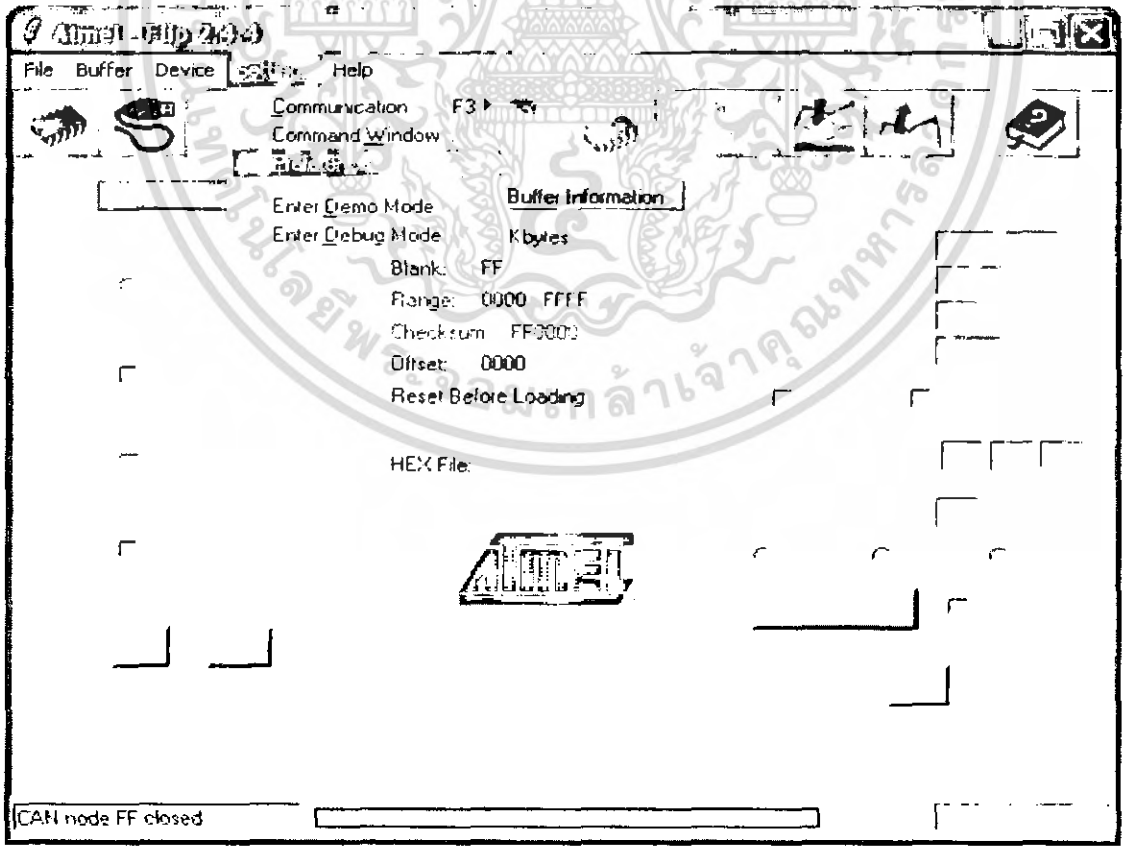


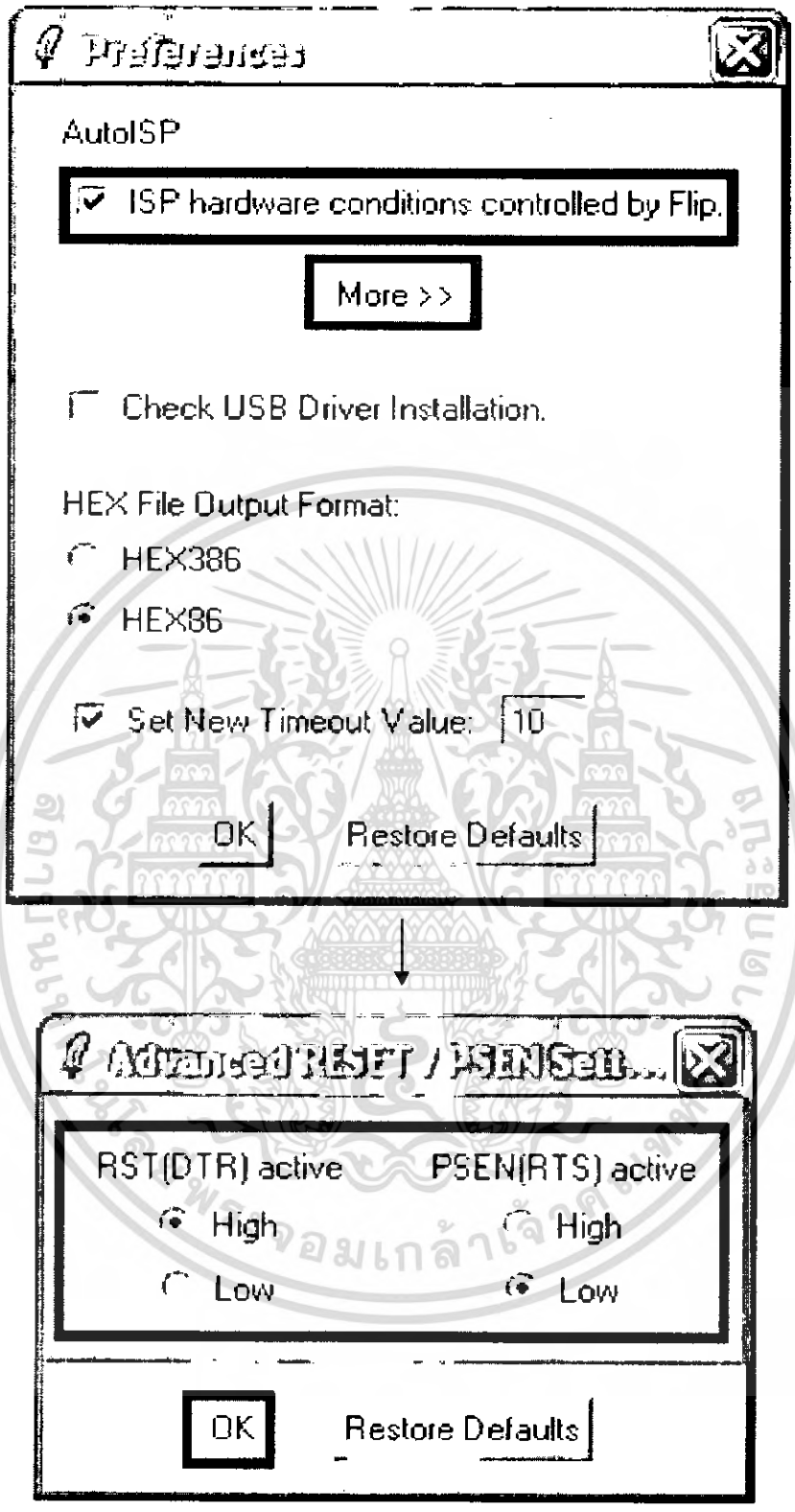
4. สั่งเลือกกำหนดเบอร์ของ MCU ที่ติดตั้งไว้ในบอร์ด โดยเลือก Device ⟶ Select ซึ่งต้องเลือกกำหนดให้ตรงกับที่ทำการติดตั้งไว้จริงๆในบอร์ดด้วย ดังตัวอย่าง (AT89C51AC3)



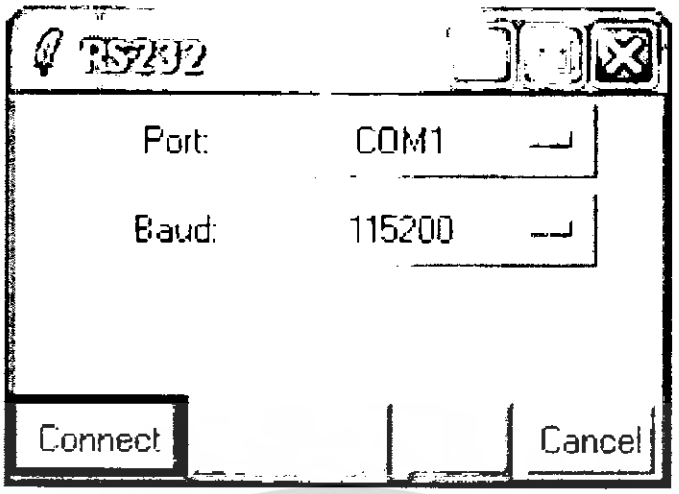
รูปแสดง การเลือกกำหนดเบอร์ CPU ของ ET-BASE51 AC3 (AT89C51AC3)

5. ทำการกำหนดค่า Option ของการสื่อสาร RS232 สำหรับใช้ดาวน์โหลดแบบอัตโนมัติ โดยให้เลือกคลิกเมาส์ที่ Setting → Preferences... แล้วเลือกกำหนดค่าดังรูป

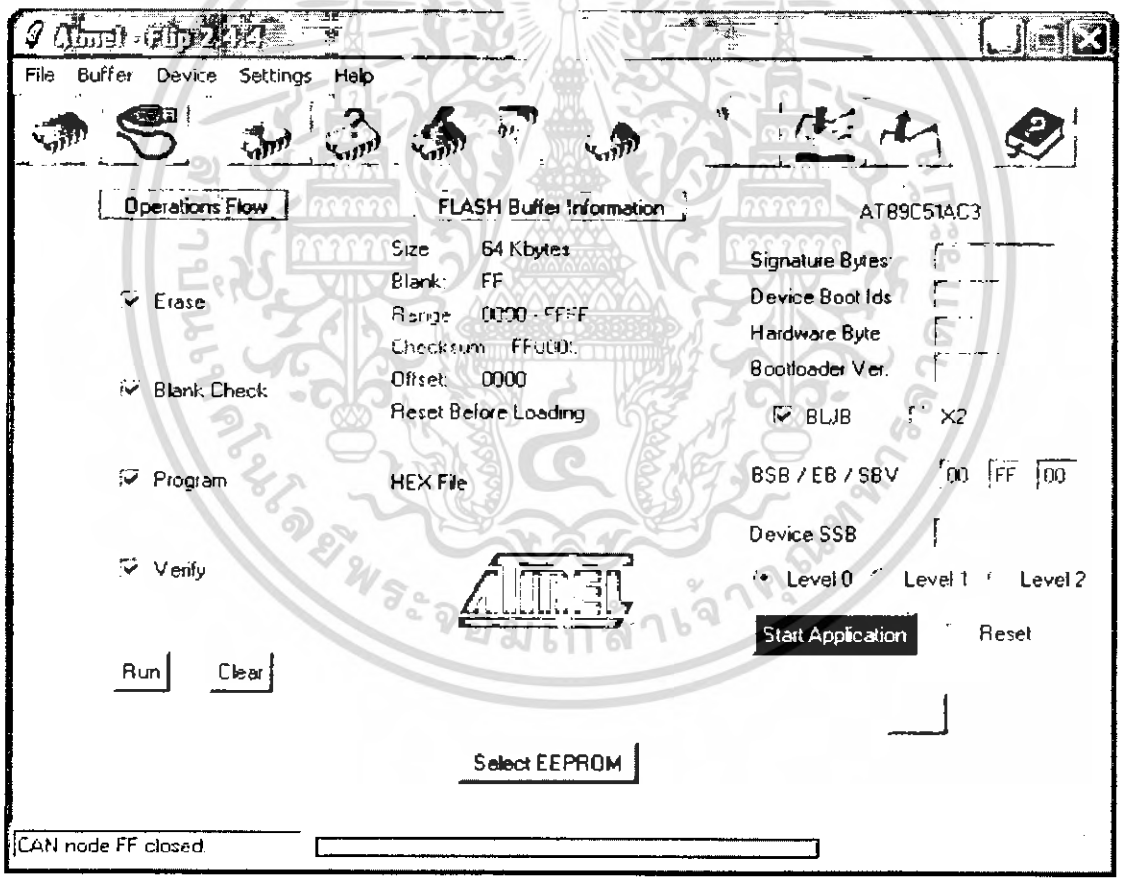




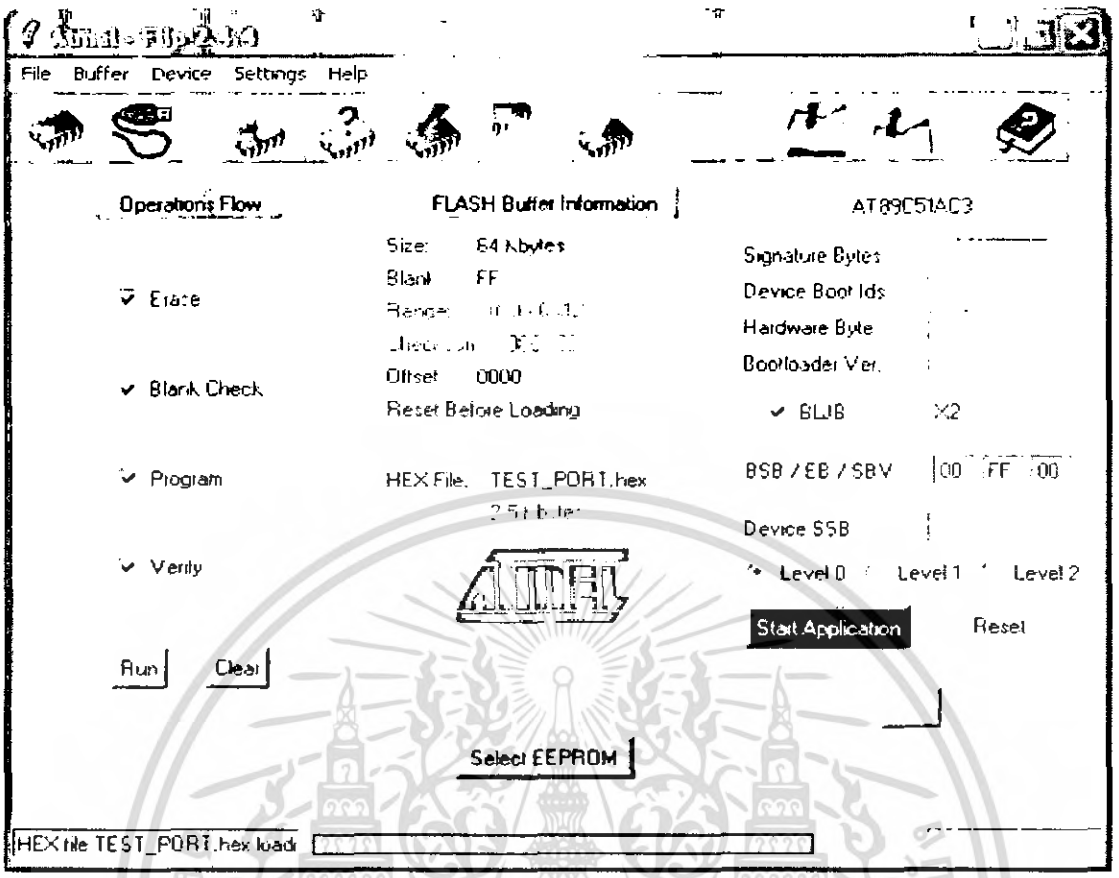
6. คลิกเมาส์ที่คำสั่ง Setting ⟶ Communication ⟶ RS232 จากนั้นเลือกกำหนด Comport ให้ตรงกับที่ต่อสายไว้จริง ดังรูป (ในตัวอย่างใช้ Com1)



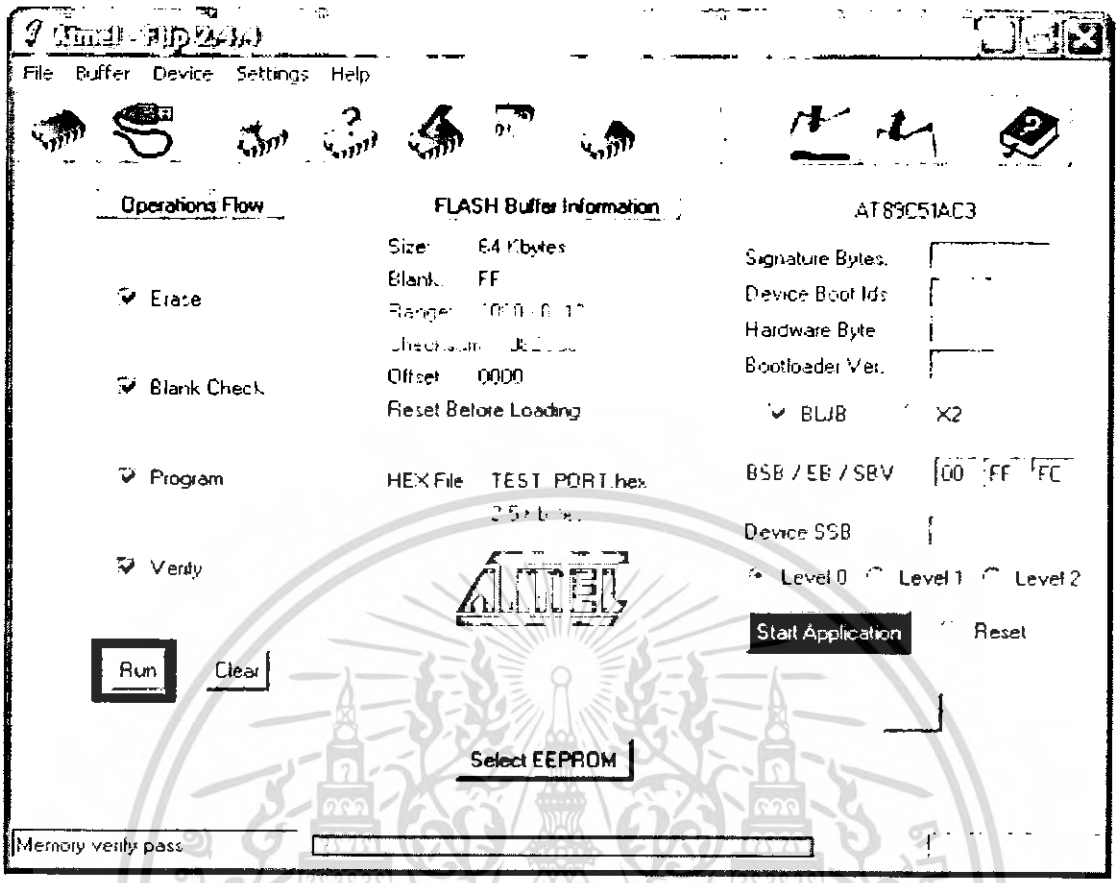
7. คลิกเมาส์ที่ปุ่ม Connect เพื่อทำการติดต่อสื่อสารกับ MCU ใน Monitor Mode ซึ่งจะได้ผลดังรูป



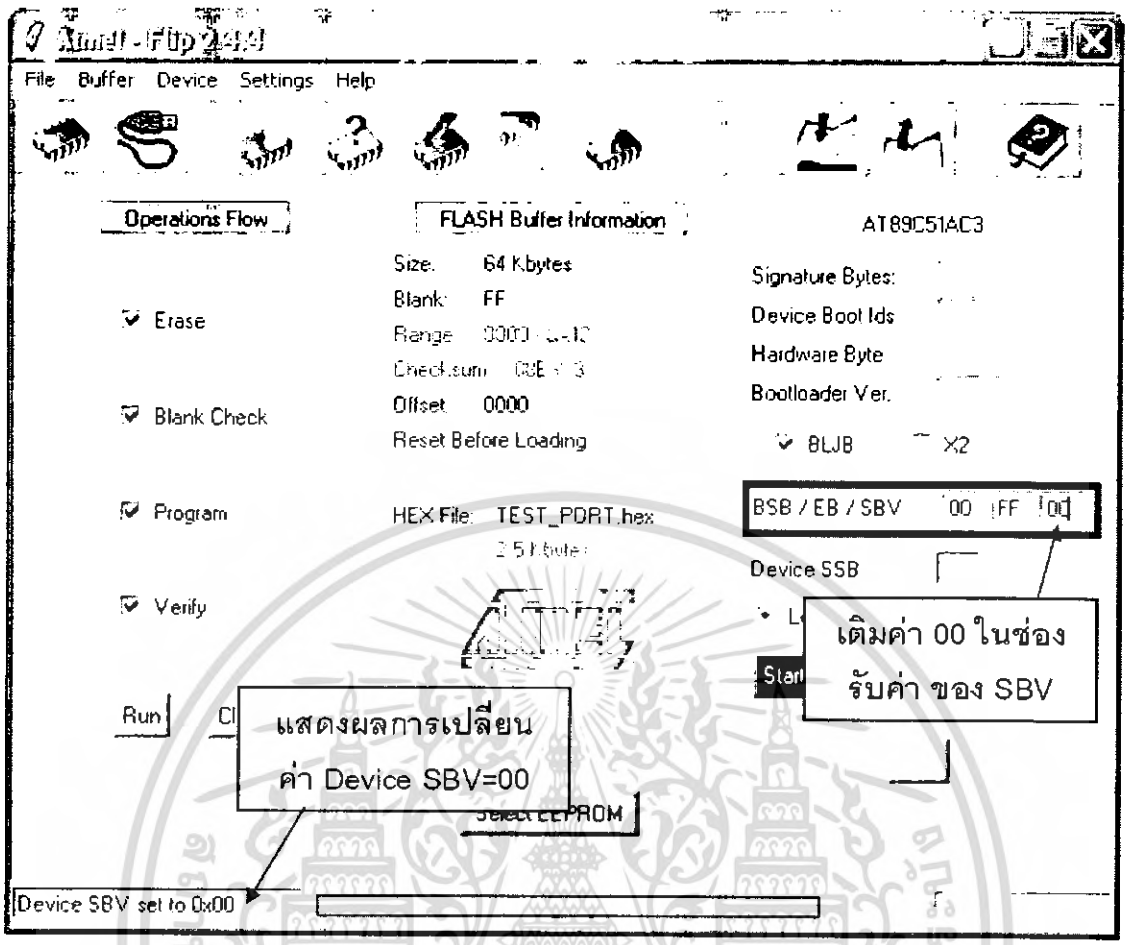
8. สั่งเปิด Hex File ที่ต้องการจะดาวน์โหลดให้กับ MCU มารอไว้ในบัฟเฟอร์ของโปรแกรม FLIP โดยใช้คำสั่ง "File ⟶ Load Hex File..."



9. คลิกเมาส์ที่หน้าตัวเลือกคำสั่งใน Tab ของ Operation Flow ให้ครบทุกคำสั่ง ซึ่งได้แก่ Erase, Bank Check, Program, Verify จากนั้นคลิกเมาส์ที่ปุ่มคำสั่ง Run และรอจนการทำงานของโปรแกรมเสร็จเรียบร้อยดังรูป



10. ตรวจสอบค่า Device BSB และ SBV ว่ามีค่าเป็น 00 ทั้งหมดแล้วหรือยัง ซึ่งถ้ายังไม่เป็น 00 ให้ทำการแก้ไขค่าให้เป็น 00 โดยคลิกเมาส์ในช่องตัวเลขแล้วพิมพ์ค่า 00 แทนที่ลงไปทั้ง 2 ช่องดังรูป



11. ทำการคลิกเมาส์ที่ “Start Application” หรือกดสวิตช์ Reset ให้กับบอร์ดเพื่อให้บอร์ดเริ่มต้นทำงานตามโปรแกรมที่ได้ทำการดาวน์โหลดไปให้ ซึ่งถ้าไม่เกิดความผิดพลาดใดๆ จะเห็น MCUเริ่มต้นทำงานทันที

CMUcam Vision Board

User Manual

Contents

This is the CMUcam Manual v2.00 for the CMUcam v1.12 firmware.
For more information go to http://www.cs.cmu.edu/~cmucam or contact us at cmucam@cs.cmu.edu

Edited by Charles Rosenberg and Illah Nourbakhsh

Introduction

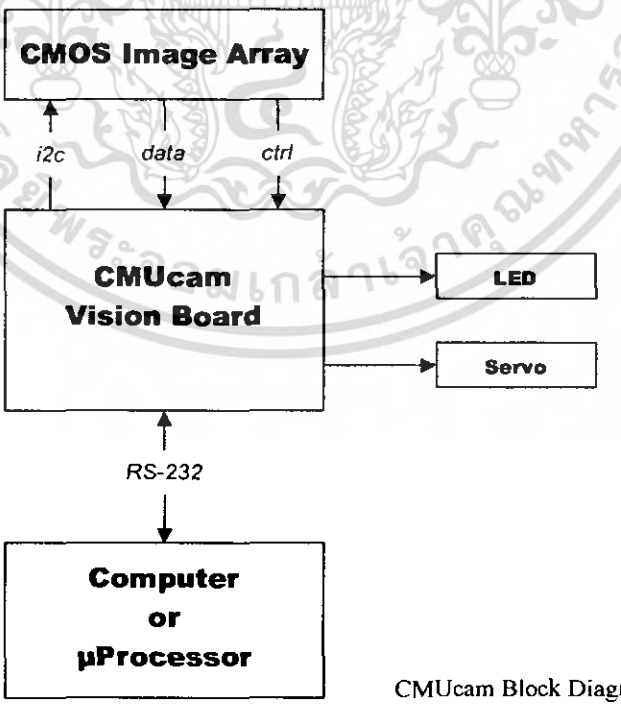
The CMUcam is a SX28 microcontroller (http: www ubicom.com/products/processors/sx28ac.html) interfaced with a OV6620 Omnivision CMOS camera (http: www.ovt.com/omnicmoss.htm) on a chip that allows simple high level data to be extracted from the camera's streaming video. The board communicates via a RS-232 or a TTL serial port and has the following functionality:

- Track user defined color blobs at 17 Frames Per Second
- Find the centroid of the blob
- Gather mean color and variance data
- Transfer a real-time binary bitmap of the tracked pixels in an image
- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image
- 80x143 Resolution
- 115,200 / 38,400 / 19,200 / 9600 baud serial communication
- Slave parallel image processing mode off a single camera bus
- Automatically detect a color and drive a servo to track an object upon startup
- Ability to control 1 servo or have 1 digital I/O pin

Typical Configurations and Uses

The CMUcam can be used to track or monitor color. The best performance can be achieved when there are highly contrasting and intense colors. For instance, it could easily track a red ball on a white background, but it would be hard to differentiate between different shades of brown in changing light. Tracking colorful objects can be used to localize landmarks, follow lines, or chase a moving beacon. Using color statistics, it is possible to monitor a scene, detect a specific color or do primitive motion detection. If the camera detects a drastic color change, then chances are something in the scene changed. Using "line mode", the CMUcam can act as an easy way to get low resolution binary images of colorful objects. This can be used to do more sophisticated line following that includes branch detection, or even simple shape recognition. These more advanced operations would require custom algorithms that would post process the binary images sent from the CMUcam. As is the case with a normal digital camera, this type of processing might require a computer or at least a fast microcontroller.

The most common configuration for the CMUcam is to have it communicate to a master processor via a standard RS232 serial port. This "master processor" could be a computer, PIC, Basic Stamp, Handy Board, Brainstem or similar microcontroller setup. The CMUcam is small enough to add simple vision to embedded systems that can not afford the size or power of a standard computer based vision system. Its communication protocol is designed to accommodate even the slowest of processors. If your device does not have a fully level shifted serial port, you can also communicate to the CMUcam over the TTL serial port. This is the same as a normal serial port except that the data is transmitted using 0 to 5 volt logic. The CMUcam supports various baud rates to accommodate slower processors. For even slower processors, the camera can operate in "poll mode". (See "PM" command pg 22.) In this mode, the host processor can ask the CMUcam for just a single packet of data. This gives slower processors the ability to more easily stay synchronized with the data. It is also possible to add a delay between individual serial data characters using the "delay mode" command. (See "DM" command pg 19.) Due to the communication delays, both poll mode and delay mode will lower the total frame rate that can be processed.



CMUcam Block Diagram

Getting Started

The CMUcam usually comes as a kit or as a fully assembled unit. If the camera is in a kit form, the first steps should be to assemble the kit. This will require that you are already comfortable with soldering electronic boards. Refer to the "Assembled Board View" on page 12 to get a general idea of what your final board should look like. In almost all cases, the Scenix SX28 chip that came with the CMUcam will already be programmed. You do NOT need to buy an SX-Key unless you wish to upgrade the firmware yourself or modify the current CMUcam firmware. Once you have the board constructed, make sure that you have a serial cable and power cable for the camera. The CMUcam can use a supply which produces anywhere from 6 to 9 volts of DC power capable of supplying at least 200mA of current. This can be provided by either an AC adaptor (possibly included) or a battery supply. These should be available at any local electronics store.

The camera consists of two major components, the CMUcam board and the CMOS camera module. This CMOS camera module must be attached to the CMUcam at all times in order for the system to function. Make sure it is connected so that it matches the orientation shown on the cover page of the manual.

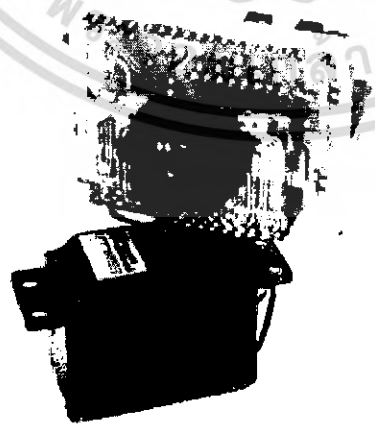
Once you have the board assembled, the next step is to double check all of your connections. Make sure that the positive side of your power plug is facing away from the main components on the board. If the camera came with an AC adaptor, make sure that the connector locks into the socket correctly (It should have a plastic extrusion that makes plugging it in backwards very difficult). Now that the camera has power, connect the serial link between the camera and your computer. This link is required initially so that you can test and focus your camera. See the serial link segment of the "Ports" section on page 13 for more information about connecting the serial cable. Check to make sure that the "Programming Switch" is in the off position. If your board has only one switch, then the programming switch was replaced by a jumper. This switch or jumper should be in the off position (for jumpers, this means that the jumper IS inserted).

Once everything is wired up, try turning the board on. If all is well, the power LED should illuminate. Now you should be ready to test your CMUcam (see next page). If the LED does not illuminate, double check all of your connections and confirm that your power supply is correctly working.

Testing the Board

Once you have set the board up and downloaded the firmware, a good way to test the system is to connect it to the serial port of a computer.

Step 1: If one does not already exist, build a serial and/or power cable (see previous section for help).
Step 2: Plug both of them in.
Step 3: Open the terminal emulator of your choice. Examples of terminal emulators are TERA term (Windows), Hyper Terminal (Windows Built in) and Minicom (Unix)
Step 4: Inside the terminal emulator set the communication protocol to 115,200 Baud, 8 Data bits, 1 Stop bit, no parity, local echo on, and if possible turn on "add line feed" (add \n to a received \r). These setting should usually appear under "serial port" or some other similar menu option.
Step 4: Turn on the CMUcam board; the Power LED should light up and the Track LED should not.
Step 5: You should see the following on your terminal emulator:
CMUcam v1.12
:
Once you have seen this, the board was able to successfully configure the camera and start the firmware.
Step 6: Type gv followed by the enter key. You should see the following:
:gv
ACK
CMUcam v1.12
:
This shows the current version of the firmware. If this is successful, your computer serial port is also configured correctly and both transmit and receive are working.
Step 7: Focus the camera lens using the CMUcam java GUI or by graphically interpreting the Dump Frame Packet yourself. Usually the lens is focused when it sits a few rotations out from its closest position to the CMOS array. Turning the lens and re-dumping the frame incrementally should provide a good feel for when the image is sharpest. (See "Focusing with the CMUcam GUI" on the next page for more information)
Step 8: Try running the camera in demo mode (see page 9).



Focusing with the CMUcam Graphical User Interface (GUI)

When you first run your CMUcam, the lens will most likely not be in focus. In order to focus the camera you need to look at some dumped images. The easiest way to do this is using a graphical user interface that can display the CMUcam frame dump packets. One option is to use the CMUcamGUI, a java program that can be found on the CMUcam website.

Step 1: Testing if you have java installed

The first step is to determine if your computer already has java installed. The easiest way to do this is go to the "start bar" under windows and select "run". Inside the run dialog, type "command" to get a dos prompt. In unix or later versions of the Mac OS, open up a shell. Now try typing "java -version" into your command line. If a message that says "java version "1.x.xx" appears then java is installed! If instead you get "command not found" or some similar message, then you need to go to java.sun.com and download a copy of java (J2SE, JDK, JRE are all valid things to install). Sun should have platform specific instructions on how to install java. Also be sure that your version of java is newer than version 1.3. If it is not, then you will need to download a new copy of java.

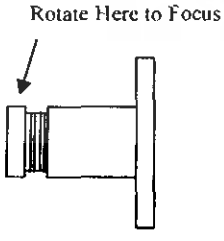
Step 2: Running the CMUcamGUI

Once you have java installed, download a copy of the latest CMUcamGUI java program. Unzip the CMUcamGUI.zip file. Now, go back to the DOS prompt or shell that you used in step 1. Using "cd", navigate to the CMUcamGUI directory that you just unzipped. You can type "dir" (dos) or "ls" (unix) to see the contents of your current directory. Once you are inside the CMUcamGUI directory make sure that you see a file called "CMUcamGUI.class". If you do not see that file, then either you did not decompress the the ZIP file, or you are in the wrong directory. If you see the CMUcamGUI.class file, then type "java CMUcamGUI".

Step 3: Grabbing a Frame

You should now see a dialog box that asks you to select the correct COM port. In windows, type in the number of the COM port that the CMUcam is connected to and press the "okay" button. In unix, make sure that the path to your com port is correct and then press "okay". The CMUcamGUI should now open and display the message "Camera OK and idle..." in the "Output Window" dialog box. That means that the CMUcamGUI found and was able to communicate with the camera. Once this works, go to the "Commands" menu and select "Dump Frame". After a few seconds you should see an image appear in the window.

Step 4: Focusing



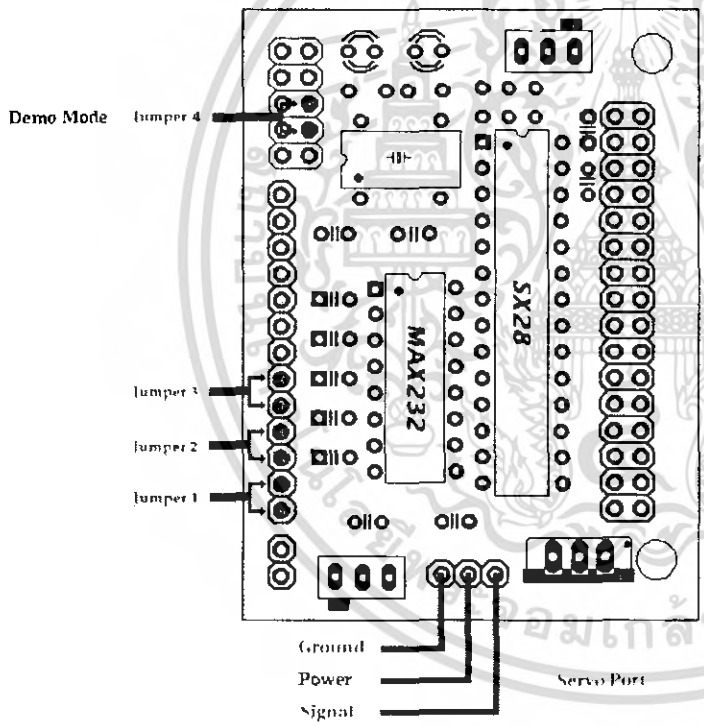
CMUcam Lens Mount

Once you have the ability to grab frames from the camera, you should be able to rotate the front part of the CMUcam lens and see the image change. Try to get the picture to be as sharp as possible by dumping frames and changing the position of the lens a small amount each time. Usually the camera is in focus when the lens is a few rotations away from the base.

Demo Mode

Jumper 4 puts the camera into a demo mode. Demo mode causes the camera to call the track window command and then begin outputting a standard hobby servo PWM signal from the servo output. The servo attempts to drive the camera mounted on it towards the middle mass of the color detected on startup. You need to plug the servo into the CMUcam's servo port as can be found on the diagram below. Try and center the servos range of motion before you attach the CMUcam.

Note that upon powering up in this mode, the camera will wait for 5 seconds before grabbing a color and servoing. When it first starts up auto gain and white balance are both on. Then after they have stabilized for 5 seconds the camera turns them off, switches into YCrCb mode and calls TW. (See page 25 for TW details) The servo tracking can be enabled manually (without the jumper) by sending the MM 2 command and then calling TW. If, when the servo is mounted on the camera, it appears to be tracking in the incorrect direction also set jumper 1. When jumper 1 and jumper 4 are set, the board does not go into slave mode; it runs demo mode and reverses the direction of the servo. Jumper 4 uses the UART port, which inhibits the use of the serial port while in Demo Mode.



The following steps are performed during power up in demo mode:

1. RS is sent to reset the camera.
2. Pause 5 seconds to allow the camera's auto adjusting parameters to stabilize.
3. The camera register string "CR 18 32 19 32" is sent.

   This selects YCrCb mode and turns off auto gain.
4. Execute the "TW" command.

Notes on Better Tracking

Better Tracking with Auto-gain and White Balance

Auto-gain is an internal control that adjusts the brightness level of the image to best suit the environment. It attempts to normalize the lights and darks in the image so that they approximate the overall brightness of a hand adjusted image. This process iterates over many frames as the camera automatically adjusts its brightness levels. If for example a light is turned on and the environment gets brighter, the camera will try and adjust the brightness to dim the overall image.

White balance on the other hand attempts to correct the camera's color gains. The ambient light in your image may not be pure white. In this case, the camera will see colors differently. The camera begins with an initial guess of how much gain to give each color channel. If active, white balance will adjust these gains on a frame-by-frame basis so that the average color in the image approaches a gray color. Empirically, this "gray world" method has been found to work relatively well. The problem with gray world white balance is that if a solid color fills the camera's view, the white balance will slowly set the gains so that the color appears to be gray and not its true color. Then when the solid color is removed, the image will have undesirable color gains until it re-establishes its gray average.

When tracking colors, like in demo mode, you may wish to allow auto-gain and white balance to run for a short period and then shut them off. While on for a period of about 5 seconds, the camera can set its brightness gain and color gains to what it sees as fit. Then turning them off will stop the camera from unnecessarily changing its settings due to an object being held close to the lens or shadows etc. If auto-gain and white balance where not disabled and the camera changed its settings for the RGB values, then the new measured values may fall outside the originally selected color tracking thresholds.

YCrCb is a different color space definition from the more commonly known RGB space. In YCrCb the illumination data is stored in a separate channel. Because of this property, in YCrCb mode the camera may be more resistant to changes in illumination. Because it is a different color space, images in YCrCb do not look like standard RGB images when directly mapped by a frame dump program. The RGB channels map to CrYCb. So in YCrCb mode, the value returned as the red parameter is actually Cr, the green parameter is Y and the blue parameter is Cb. So if you wish to track a red object, you need to look at a dumped frame to see what that object's colors map to in YCrCb. It should then be possible to find the Cr and Cb bounds while giving a very relaxed Y bound showing that illumination is not very important. Below are the transformations used by the camera to convert RGB into YCrCb:

RGB -> CrYCb
Y=0.59G + 0.31R + 0.11B
Cr=R-Y
Cb=B-Y

About the CMOS Camera

From power up, the camera can take up to 15 seconds to automatically adjust to the light. Drastic changes in the environment, such as lights being turned on and off, can induce a similar readjustment time. When using the camera outside, due to the sun's powerful IR emissions, even on relatively cloudy days, it will probably be necessary to use either an IR filter or a neutral density camera filter to decrease the ambient light level. The field of view depends on the lens attached to the camera. It is possible to special order the camera with wider or narrower lenses. Individual lenses can be purchased separately.

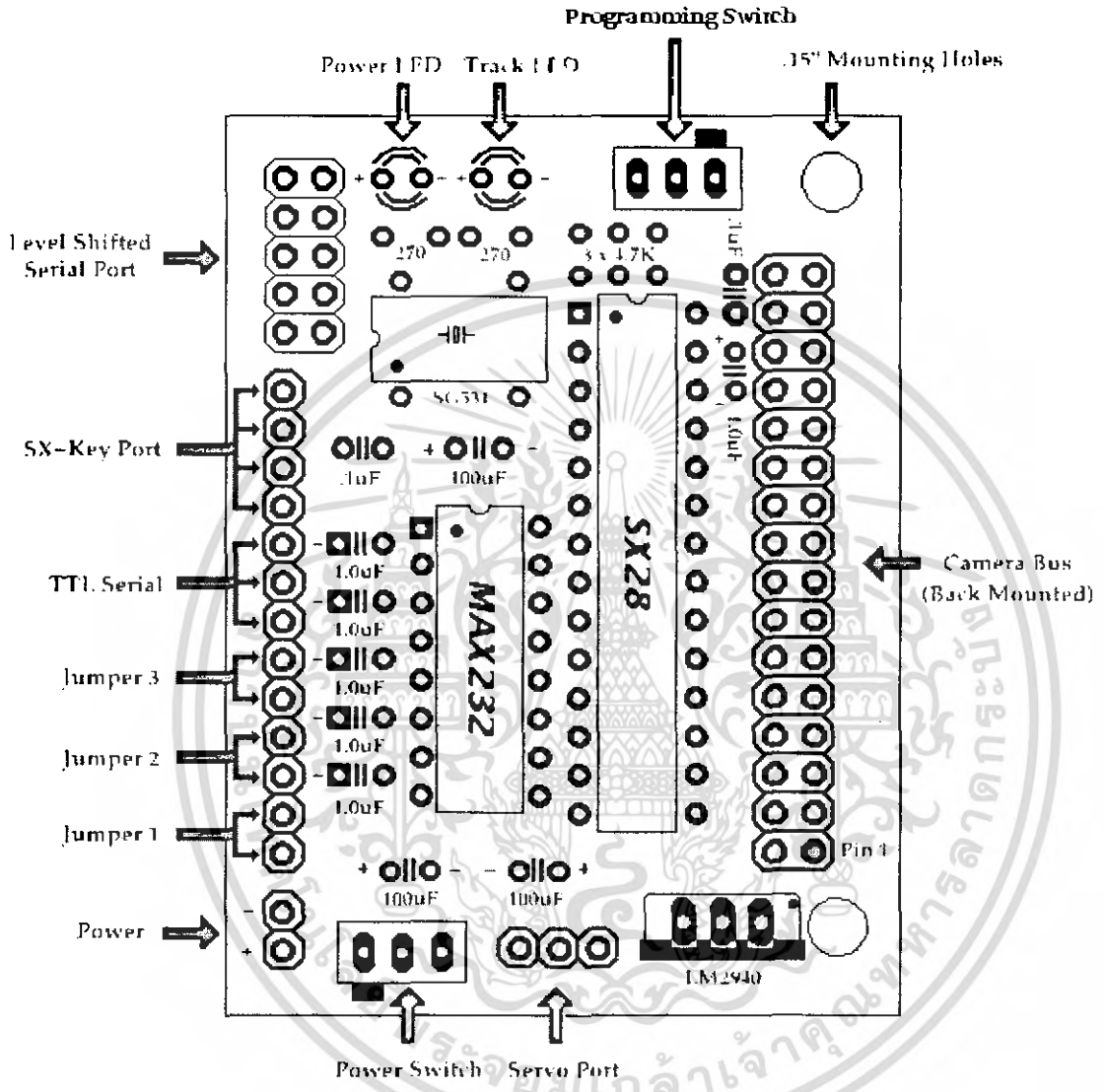
The functions provided by the camera board are meant to give the user a toolbox of color vision functions. Actual applications may greatly vary and are left up to the imagination of the user. The ability to change the viewable window, grab color / light statistics and track colors can be interwoven by the host processor to create higher level functionality.

One notable property of the CMOS sensor array is that it returns values between 16 and 240 for each pixel. This effect is noticeable when the camera is tracking colors, getting mean color data or dumping a frame. This limited range on the data does not depend on the mode of the camera and will still exist in YCrCb mode.

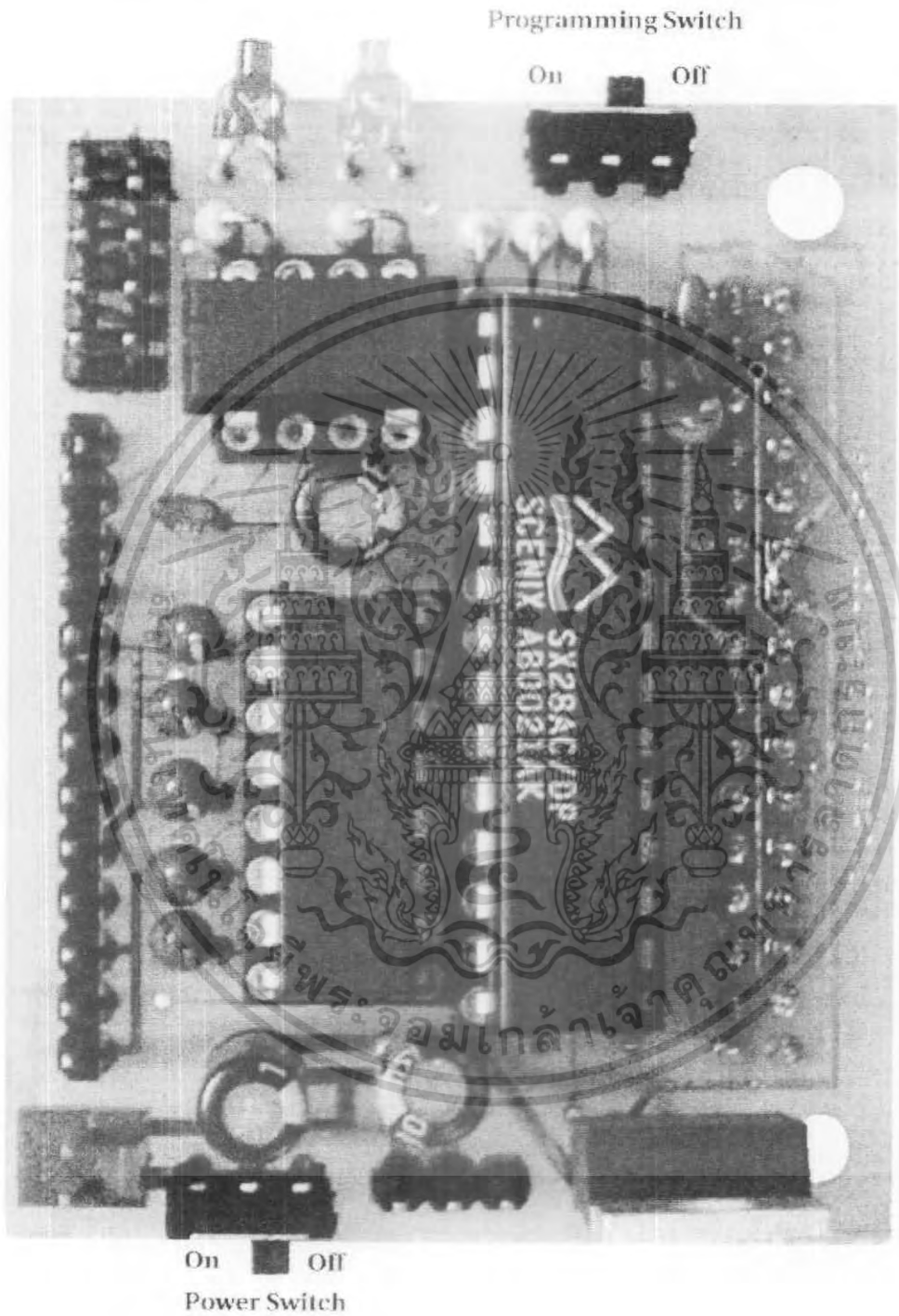
Analog Video Output

Pin 32 on the camera bus is a black and white analog output pin. It is possible to connect the analog output to a TV or multi-sync monitor. Due to the clock rate of the camera, the analog output does not correctly synchronize with standard NTSC or PAL systems. If you have some system that can synchronize with a non-standard signal, it may be possible to monitor the video while processing. On versions 1.22 and lower of the board, it is required that you cut the connection on the CMUcam board between camera bus pins 31 and 32 (this is fixed in version 1.23 and higher). Then, connect the central line on a 75 Ohm coax input plug to pin 32 on the camera bus. Next, connect the outer shield to the camera's common ground (pin 31 on the camera bus). Finally, power up the camera and adjust the frame rate until you see the best results. Most standard TVs can at least see a skewed flickering image when the frame rate clock divider is set to 0 (CR 17 0). Setting frame rates higher than 17 fps will stop the CMUcam's processing from working. So while you are sending data to a monitor, you can not dump frames or perform any processing.

Board Layout



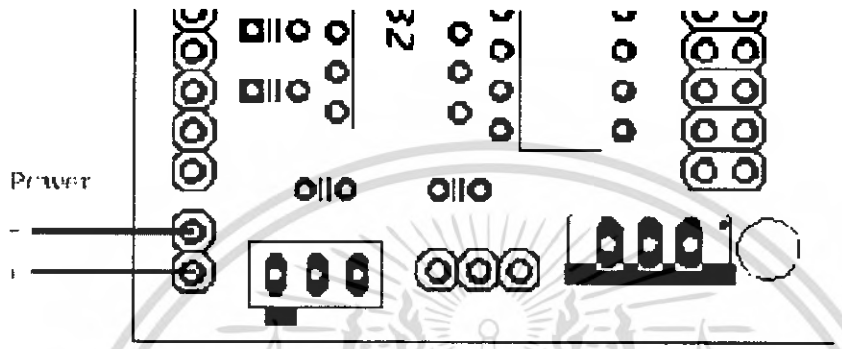
Assembled View



Ports

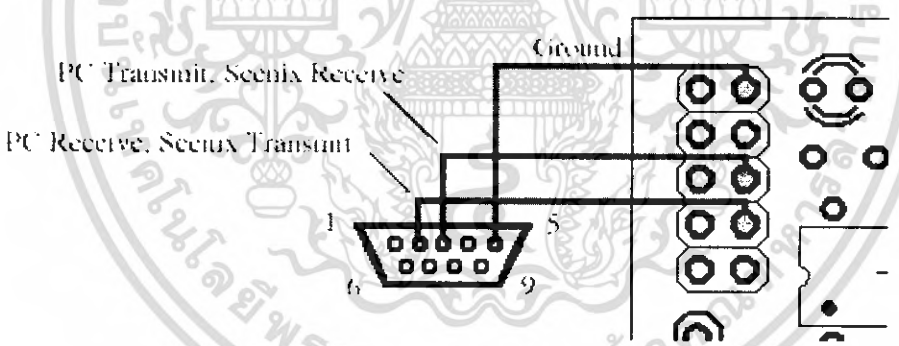
Power

The input power to the board goes through a 5 volt regulator. It is ideal to supply the board with between 6 and 9 volts of DC power that is capable of supplying at least 200 milliamperes of current.



Level Shifted Serial Port

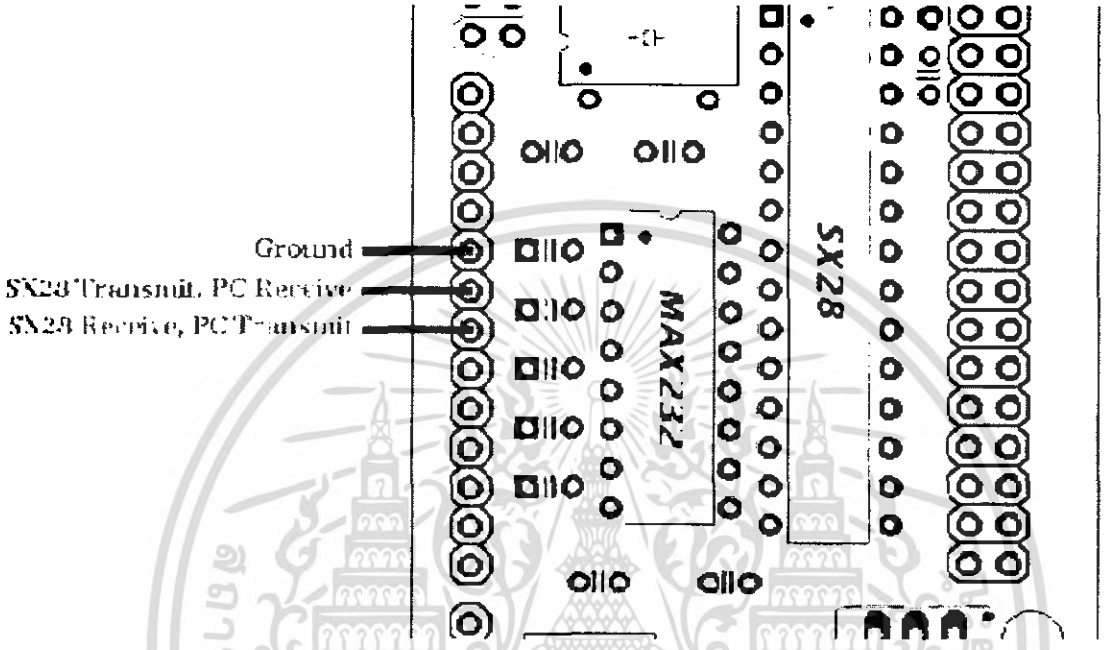
This port provides full level shifting for communication with a computer. Though it only uses 3 of the 10 pins it is packaged in a 2x5 pin configuration to fit standard 9 pin ribbon cable clip-on serial sockets and 10 pin female clip on serial headers that can both attach to a 10 wire ribbon cable. If this initially does not work, try flipping the direction that the ribbon cable connects to the CMUcam board.



The trapezoidal serial connector shown above is what the serial connector on your computer should look like.

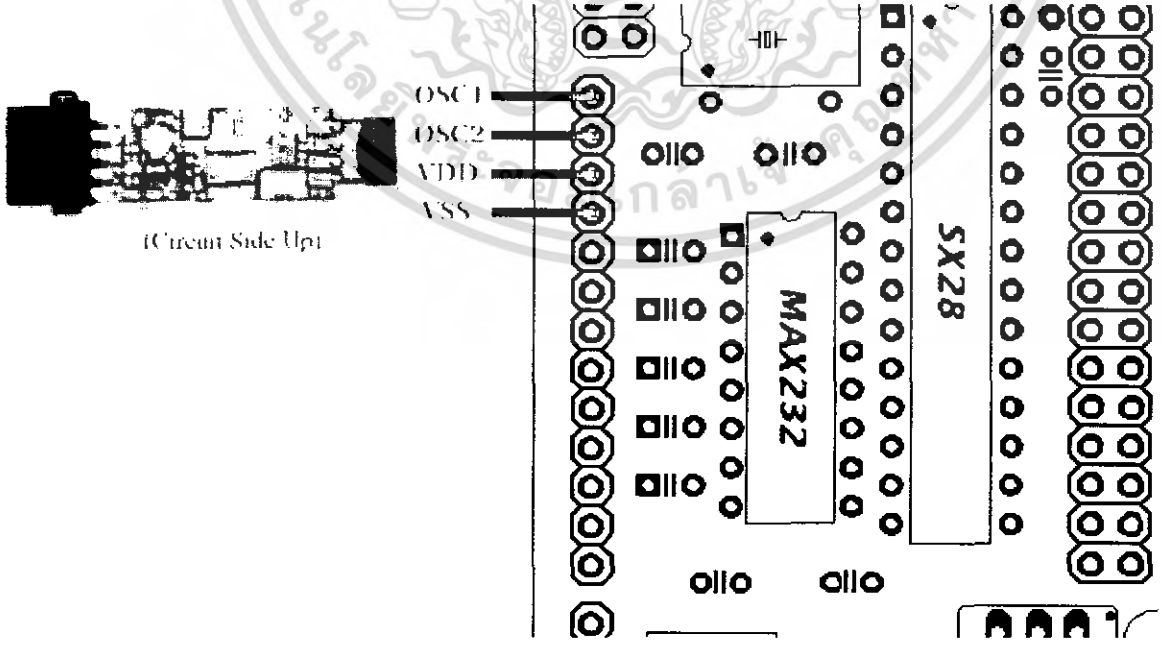
TTL Serial Port

This serial port taps into the serial I/O before it goes through the MAX232 chip. This port may be ideal for communication with a microcontroller that does not have any built in level shifting. Note that, it is necessary to remove the max232 chip to use this port.



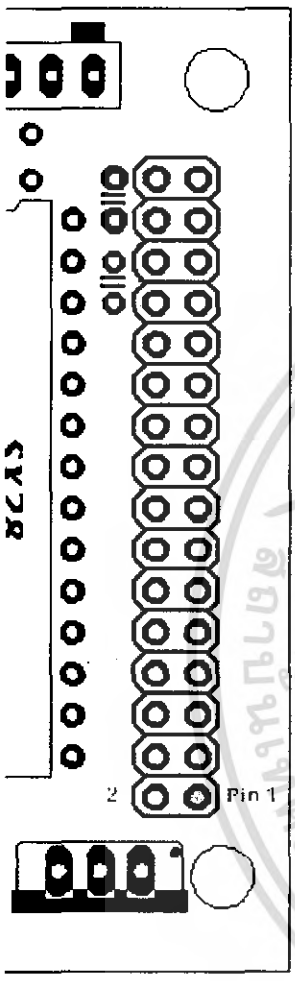
Programming Port

The programming port allows the firmware to be downloaded to the SX28 using a SX-Key / Blitzer or equivalent programmer. These can be purchased from Parallax Inc (www.parallaxinc.com). The SX28 is usually provided preprogrammed.



Camera Bus

This bus interfaces with the CMOS camera chip. The CMOS camera board is mounted parallel to the processing part of the board and connects starting at pin 1. The female camera header should be soldered on the back of the board. (See assembled view.) For information about the Video Analog Output port see page 10.

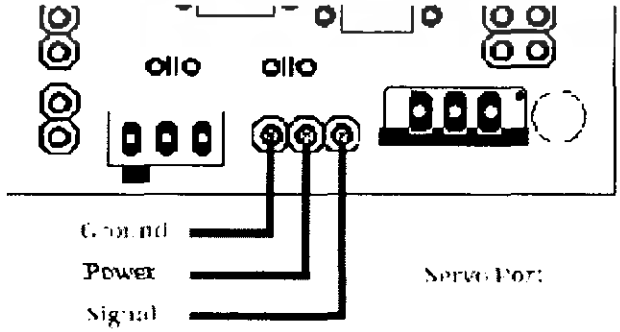


Pin Description

1~8	Y0~Y7	Digital Output Y Bus
9	PWDN	Power Down Mode
10	RST	Reset
11	SDA	I2C Serial Data
12	FODD	Odd Field Flag
13	SCL	I2C Serial Clock
14	HREF	Horizontal Window Reference
15	AGND	Analog Ground
16	VSYN	Vertical Sync
17	AGND	Analog Ground
18	PCLK	Pixel Clock
19	EXCLK	External Clock
20	VCC	+5 VDC
21	AGND	Analog Ground
22	VCC	+5 VDC
23~30	UV0~UV7	Digital output UV bus
31	GND	Common Ground
32	VTO	Video Analog Output (75 Ohm)

Servo Port

This is the output for the servo. The servo power does not go through a regulator from the board's power input. Do not use a servo with the board if the board is being run off of more than 6 volts.



Jumpers

Parallel Processing in Slave Mode (Jumper 1)

The CMUcam supports a mode of operation that allows multiple boards to process data from the same camera. If a PC104 style pass-through header is used instead of the standard double row female header, it is possible to rack multiple boards along the same camera bus. Upon startup, if jumper 1 is set, the camera becomes a slave. Slave mode stops the camera board from being able to configure or interfere with the CMOS camera's settings. Instead it just processes the format setup by the master vision board. When linking the buses together you must only have one master; all other boards should be setup to be in slave mode. In this current version of the system there is no message passing between boards other than the image data from the camera bus. This means you have to communicate to each slave board via a separate serial link. This communication to the board should be identical to using a single CMUcam. For example, you could have the master board tracking some color while the slave board could be told to get mean color data. Each board runs independently of one another and only the master can control camera registers.

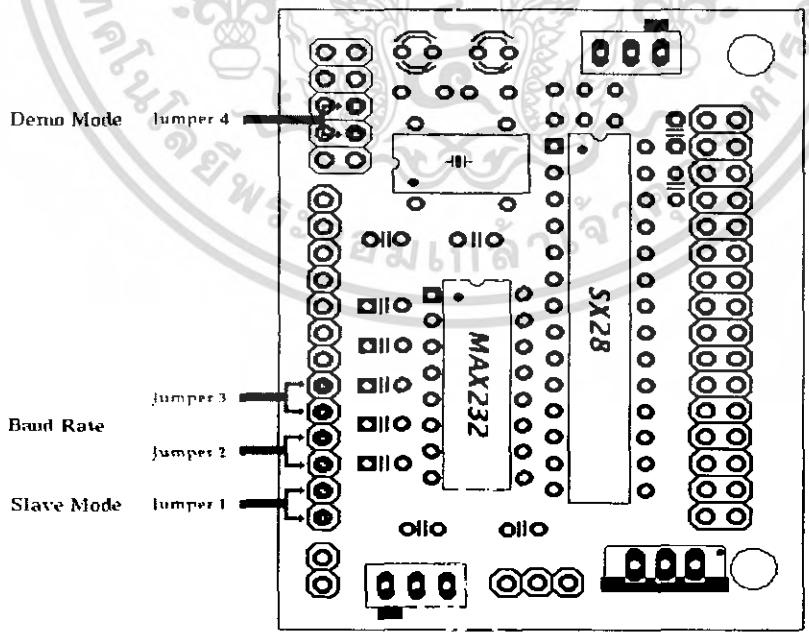
Baud Rate (Jumpers 2 and 3)

115,200 Baud	Jumper 2 Open	Jumper 3 Open
38,400 Baud	Jumper 2 Set	Jumper 3 Open
19,200 Baud	Jumper 2 Open	Jumper 3 Set
9,600 Baud	Jumper 2 Set	Jumper 3 Set

Because of the extra time it takes to transmit data at lower rates, frames may be skipped resulting in a lower frame rate. The slower rate will also cause the bitmap mode and dump frame resolutions to shrink.

Demo Mode (Jumper 4)

Jumper 4 puts the camera into a demo mode. Demo mode causes the camera to call the track window command and then begin outputting a standard hobby servo PWM signal from the servo output. The servo attempts to drive the camera mounted on it towards the middle mass of the color detected on startup. For more information see the "Demo Mode" section on page 8.



Serial Command Set

The serial communication parameters are as follows:

- 115,200 Baud
- 8 Data bits
- 1 Stop bit
- No Parity
- No Flow Control (Not Xon/Xoff or Hardware)

All commands are sent using visible ASCII characters (123 is 3 bytes "123"). Upon a successful transmission of a command, the ACK string should be returned by the system. If there was a problem in the syntax of the transmission, or if a detectable transfer error occurred, a NCK string is returned. After either an ACK or a NCK, a \r is returned. When a prompt ('\r' followed by a ':') is returned, it means that the camera is waiting for another command in the idle state. White spaces do matter and are used to separate argument parameters. The \r (ASCII 13 carriage return) is used to end each line and activate each command. If visible character transmission exerts too much overhead, it is possible to use varying degrees of raw data transfer. (See Raw Mode command, "RM".)

\r

This command is used to set the camera board into an idle state. Like all other commands, you should receive the acknowledgment string "ACK" or the not acknowledge string "NCK" on failure. After acknowledging the idle command the camera board waits for further commands, which is shown by the ':' prompt. While in this idle state a /r by itself will return an "ACK" followed by \r and : character prompt. This is how you stop the camera while in streaming mode.

Example of how to check if the camera is alive while in the idle state

:
ACK
:

CR [reg1 value1 [reg2 value2 ... reg16 value16]]\r

This command sets the **C**amera's internal **R**egister values directly. The register locations and possible settings can be found in the Omnivision CMOS camera documentation. All the data sent to this command should be in decimal visible character form unless the camera has previously been set into raw mode. It is possible to send up to 16 register-value combinations. Previous register settings are not reset between CR calls; however, you may overwrite previous settings. Calling this command with no arguments resets the camera and restores the camera registers to their default state. This command can be used to hard code gain values or manipulate other low-level image properties.

Common Settings:

Register	Values	Effect
5 Contrast	0-255	
6 Brightness	0-255	
18 Color Mode		
	36	YCrCb* Auto White Balance On
	32	YCrCb* Auto White Balance Off
	44	RGB Auto White Balance On
	40	RGB Auto White Balance Off (default)
17 Clock Speed		
	2	17 fps (default)
	3	13 fps
	4	11 fps
	5	9 fps
	6	8 fps
	7	7 fps
	8	6 fps
	10	5 fps
	12	4 fps
19 Auto Exposure		
	32	Auto Gain Off
	33	Auto Gain On (default)

Example of decreasing the internal camera clock speed (default speed is 2)
:CR 17 5
ACK
:

*The red channel becomes Cr which approximates r-g. The green channel becomes Y which approximates intensity, the blue channel becomes Cb which approximates b-g
RGB -> CrYCb
Y=0.59G + 0.31R + 0.11B
Cr=R-Y
Cb=B-Y

DF\r

This command will **D**ump a **F**rame out the serial port to a computer. This is the only command that will by default only return a non-visible ASCII character packet. It dumps a type F packet that consists of the raw video data column by column with a frame synchronize byte and a column synchronize byte. (This data can be read and displayed by the CMUcamGUI java application.) Since the data rate required to send the raw video greatly exceeds the maximum serial port speed. only one column per frame is sent at a time. This allows you to see a slowly updating view of what the camera sees. To get the correct aspect ratio, double each column of pixels. The camera is able to dump a full resolution frame at full speed (17 columns per second) only at 115,200 baud. At lower baud rates, or 115,200 baud with added delays the frame rate must be decreased in order to see a full resolution image. With auto-gain on and at lower frame rates, the image at first may appear much brighter than usual. This is because the camera is getting frames slower than usual and takes longer to adapt. (Try manually setting the brightness and contrast.)

Type F data packet format
1 - new frame
2 - new col
3 - end of frame
RGB (CrYCb) ranges from 16 - 240
1 2 r g b r g b ... r g b r g b 2 r g b r g b r ... r g b r g b ...

Example of a Frame Dump from a terminal program
(WARNING: This may temporarily interfere with a terminal program by sending non-visible characters)
:DF
ACK
*maKP(U A$IU AL<>U A$L*YL%*L L (G AUsonthAYA(KMAy098a34ymawvk....*

DM *value*\r

This command sets the **D**elay before characters that are transmitted over the serial port and can give slower processors the time they need to handle serial data. The value should be set between 0 and 255. A value of 0 (default) has no delay and 255 sets the maximum delay. Each delay unit correlates to approximately the transfer time of one bit at the current baud rate.

GM\r

This command will **G**et the **M**ean color value in the current image. If, optionally, a sub-region of the image is selected, this function will only operate on the selected region. The mean values will be between 16 and 240 due to the limits of each color channel on the CMOS camera (See page 10). It will also return a measure of the average absolute deviation of color found in that region. The mean together with the deviation can be a useful tool for automated tracking or detecting change in a scene. In YCrCb mode RGB maps to CrYCb.

Type S data packet format
S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

Example of how to grab the mean color of the entire window
:SW 1 1 40 143
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2

GV\r

This command **G**ets the current **V**ersion of the firmware from the camera. It returns an ACK followed by the firmware version string.

Example of how to ask for the firmware version
:GV
ACK
CMUcam v1.12

HM *active*\r

This command puts the camera into **H**alf-horizontal resolution **M**ode for the DF command and the LM command when dumping a bitmap image. An *active* value of 1 causes only every odd column to be processed. The default value of 0 disables the mode.

I1 \r

This command uses the servo port as a digital **I**nput. Calling I1 returns either a 1 or 0 depending on the current voltage level of the servo line. The line is pulled high; because of this it is only required to pull it low or let it float to change it's state. The servo line can also be used as a digital output. (See S1 command.)

Example of how to read the digital value of the servo line
:I1
ACK
1

L1 *value*\r

This command is used to control the tracking **L**ight. It accepts 0, 1 and 2 (default) as inputs. 0 disables the tracking light while a value of 1 turns on the tracking light. A value of 2 puts the light into its default auto mode. In auto mode and while tracking, the light turns on when it detects the presence of an object that falls within the current tracking threshold. This command is useful as a debugging tool.

Example of how to toggle the Tracking Light on and then off
:L1 2
ACK
:L1 0
ACK

LM *active\r*

This command turns on **L**ine **M**ode which uses the time between each frame to transmit more detailed data about the image. It adds prefix data onto either **C, M** or **S** packets. This mode is intended for users who wish to do more complex image processing on less reduced data. Since the frame rate is not compromised, the actual processing of the data put out by the vision system must be done at a higher rate. This may not be suitable for many slower microcontrollers.

Line mode's effect on TC and TW:

When line mode is active and TC or TW is called, line mode will send a binary bitmap of the image as it is being processed. It will start this bitmap with an 0xAA flag value (hex value AA not in human readable form). The value 0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 pixels being streamed from the top-left to the bottom-right of the image. The vertical resolution is constrained by the transfer time of the horizontal data so lines may be skipped when outputting data. In full resolution mode, the resulting binary image is 80x48. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **C** or **M** data packet (processed at that lower resolution).

Example of TC with line mode on
:LM 1
:TC
(raw data: AA XX XX XX .... XX XX XX AA AA) C 45 72 65 106 18 51
(raw data: AA XX XX XX .... XX XX XX AA AA) C 46 72 65 106 18 52

Line mode's effect on GM:

When line mode is active and GM is called, line mode will send a raw (not human readable) mean value of every line being processed. These packets are started with an 0xFE and terminate with an 0xFD. Each byte of data between these values represents the corresponding line's mean color value. Similarly to the bitmap mode the vertical resolution is halved, because of the serial transfer time. At 17 fps 115,200 baud every other line is skipped. At any slower frame rate (still 115,200 baud) no lines will be skipped.

Example of GM with line mode on
:LM 1
:GM
(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8
(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

MM *mode*\r

This command controls the **M**iddle **M**ass mode which adds the centroid coordinates to the normal tracking data. A *mode* value of 0 disengages middle mass, a value of 1(default) engages middle mass and a value of 2 engages the mode and turns on the servo PWM signal that tries to center the camera on the center of color mass (see the Demo Mode Jumper description). This mode is good if you want a single point representation of where the object is or if there is too much small background noise to get a good bounding box. The *mode* value acts as a bit field for the following operations. Setting the bits enables the functionality.

$mode = B_3\ B_2\ B_1\ B_0$

Bits	
B_0	Enable / Disable Middle Mass Mode
B_1	Enable / Disable the Servo Output
B_2	Change Servo Direction
B_3	Return N-type packet that includes the current servo position (see page 27.)

Example of how to disable Middle Mass mode
:MM 0
ACK
:TC
ACK
C 38 82 53 128 35 98
C 38 82 53 128 35 98
C 38 82 53 128 35 98

NF *active*\r

This command controls the **N**oise **F**ilter setting. It accepts a Boolean value 1 (default) or 0. A value of 1 engages the mode while a value of 0 deactivates it. When the mode is active, the camera is more conservative about how it selects tracked pixels, it requires 2 sequential pixels for a pixel to be tracked.

Example of how to turn off noise filtering
:NF 0
ACK
:

PM *mode*\r

This command puts the board into **P**oll **M**ode. Setting the mode parameter to 1 engages poll mode while 0 (default) turns it off. When poll mode is engaged only one packet is returned when an image processing function is called. This could be useful if you would like to rapidly change parameters or if you have a slow processor that can't keep up with a given frame rate.

Example of how to get one packet at a time
:PM 1
ACK
:TC 50 20 90 130 70 255
ACK
C 38 82 53 128 35 98
:

RM *bit_flags*\r

This command is used to engage the **R**aw serial transfer **M**ode. It reads the bit values of the first 3 (lsb) bits to configure settings. All bits cleared sets the default visible ascii mode. If bit 0 is set, then all output from the camera is in raw byte packets. The format of the data packets will be changed so as not to include spaces or be formatted as readable ASCII text. Instead you will receive a 255 valued byte at the beginning of each packet, the packet identifying character (i.e. C for a color packet) and finally the packet. There is no \r sent after each packet, so you must use the 255 to synchronize the incoming data. Any 255 valued bytes that may be sent as part of the packet are set to 254 to avoid confusion. If bit 1 is set, the "ACK\r" and "NCK\r" confirmations are not sent. If bit 3 is set, input will be read as raw byte values, too. In this mode, after the two command byte values are sent, send 1 byte telling how many arguments are to follow. (i.e. DF followed by the raw byte value 0 for no arguments) No \r character is required.

bit_flags = $B_2\,B_1\,B_0$

Bits	
B_0	Output to the camera is in raw bytes
B_1	"ACK\r" and "NCK\r" confirmations are suppressed
B_2	Input to the camera is in raw bytes

Example of the new packet for Track Color with Raw Mode output only
(WARNING: This may temporarily interfere with a terminal program by sending non visible characters)
:RM 1
ACK
:TC 50 20 90 130 70 255
ACK
C>%k(ai Ck$&,.L

RS \r

This command **ReS**ets the vision board. Note, on reset the first character is a /r.

Example of how to reset the camera
:rs
ACK

CMUcam v1.12
:

S1 *position* \r

This command lets you **S**et the position of servo 1. 0 turns the servo off and holds the line low. 1-127 will set the servo to that position while it is tracking or getting mean data. Any value 128 and higher sets the line high. In order for the servo to work, the camera must be in either a tracking loop or mean data gather loop. Values 0 and 128 can be useful if you wish to use the servo port as a digital output. The port can also be used as a digital input (see I1 command). The "MM" command can enable or disable the automatic servo tracking.

SM *value* \r

This command is used to enable the **S**witching **M**ode of color tracking. When given a 0 it is in it's default mode where tracking will return its normal C or M color packet. If the value is set to 1, the tracking commands will alternate each frame between color packets and S statistic packets. Each statistic packet is only being sampled from an area one quarter the size of the bounded area returned from the tracking command. If no object was bounded, then no S statistic packets are returned. This can be useful for adaptive tracking or any type of tracking where you would like to get feedback from the currently bound target. After the first tracking packet is returned, the window gets set back to full size for all future packets. Note, you will get only half the number of actual color packets per time interval.

Example of how to Track Color with SM
:SM 1
ACK
:TC 200 255 0 30 0 30
ACK
C 2 40 12 60 10 70
S 225 20 16 2 3 1
C 5 60 20 30 12 100
S 225 19 17 1 2 1
C 0 0 0 0 0 0
C 0 0 0 0 0 0
C 0 0 0 0 0 0
C 5 60 20 30 12 100
S 225 19 17 1 2 1

SW [x y x2 y2] \r

This command **S**ets the **W**indow size of the camera. It accepts the x and y Cartesian coordinates of the upper left corner followed by the lower right of the window you wish to set. The origin is located at the upper left of the field of view. **SW** can be called before an image processing command to constrain the field of view. Without arguments it returns to the default full window size of 1,1,80,143. Do NOT try "SW 0 0 80 144", this is outside of the 1 1 80 143 bounds.

Example of setting the camera to select a mid portion of the view
:SW 35 65 45 75
ACK
:

TC [*Rmin Rmax Gmin Gmax Bmin Bmax*]\r

This command begins to **T**rack a **C**olor . It takes in the minimum and maximum RGB (CrYCb) values and outputs a type M or C or N data packet (set by the MM command). The smaller type C packet encodes a bounded box that contains pixels of the curently defined color, the number of found pixels scaled (actual value is (pixels+4)/8) that fall in the given color bounds and a confidence ratio. The default type M packet also includes the center of mass of the object. The resolution of the processed image is 80x143. The X values will range from 1 to 80 and the y values will range from 1 to 143. A packet of all zeros indicates that no color in that range was detected. The confidence value is a ratio of the pixels counted within the given range and the area of the color bounding box, it returns a value which ranges from 0 to 255. Under normal operations any value greater then 50 should be considered a very confident lock on a single object. A value of 8 or lower should be considered quite poor. With no arguments, the last color tracking parameters will be repeated. Remember that the color values from the CMOS camera will range from between 16 and 240 (See page 10).

Type M packet
M mx my x1 y1 x2 y2 pixels confidence\r

Type C packet
C x1 y1 x2 y2 pixels confidence\r

Example of how to Track a Color with the default mode parameters
:TC 130 255 0 0 30 30
ACK
M 50 80 38 82 53 128 35 98
M 52 81 38 82 53 128 35 98

TW \r

This command will **T**rack the color found in the central region of the current **W**indow. After the color in the current window is grabbed, the track color function is called with those parameters and on the full screen. This can be useful for locking onto and tracking an object held in front of the camera. Since it actually calls track color, it returns the same type of C or M color packet. Note, your set window will only be used for grabbing the color to track and then the window will return to 80x143.

The following internal steps are performed when the "TW" command is called:

1. Shrink the window to 1/4 the size (in each dimension) of the current window. Position the new window at the center of the camera's field of view. (sw 30 54 50 90)
2. Call the get mean command. (gm)
3. Restore the window to the full image size. (sw 1 1 80 143)
4. Set the min and max value for each color channel to be the mean for that channel +/- 30.

Example of how to use Track Window (not that Middle Mass mode is not active):
:TW
ACK
S 240 50 40 12 7 8
C 2 40 12 60 10 70
C 2 41 12 61 11 70

Output Data Packet Descriptions

All output data packets are in ASCII viewable format except for the F frame and prefix packets.

ACK

This is the standard acknowledge string that indicates that the command was received and fits a known format.

NCK

This is the failure string that is sent when an error occurred. The only time this should be sent when an error has not occurred is during binary data packets.

Type **C** packet:

C x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking command.
x1 - The left most corner's x value
y1 - The left most corner's y value
x2 - The right most corner's x value
y2 -The right most corner's y value
pixels –Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8
confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type **F** data packet format:

1 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b ...

1 - new frame 2 - new col 3 - end of frame
RGB (CrYCb) ranges from 16 - 240
RGB (CrYCb) represents a two pixels color values. Each pixel shares the red and blue.
176 cols of R G B (Cr Y Cb) packets (forms 352 pixels)
144 rows
To display the correct aspect ratio, double each column so that your final image is 352x144
It does not begin with an "F" and only sends raw data!

Type **M** packet:

M mx my x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking command with Middle Mass mode on.
mx - The middle of mass x value
m*y* - The middle of mass y value
x1 - The left most corner's x value
y1 - The left most corner's y value
x2 - The right most corner's x value
y2 -The right most corner's y value
pixels –Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8
confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type **N** packet:

N spos mx my x1 y1 x2 y2 pixels confidence\r

This is identical to a type **M** packet with an added value for the servo position.
spos – The current position of the servo

Type **S** data packet format:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

This is a statistic packet that gives information about the camera's view
Rmean - the mean Red or Cr (approximates r-g) value in the current window
Gmean - the mean Green or Y (approximates intensity) value found in the current window
Bmean - the mean Blue or Cb (approximates b-g) found in the current window
R*deviation* - the *deviation of red or Cr found in the current window
G*deviation*- the *deviation of green or Y found in the current window
B*deviation*- the *deviation of blue or Cb found in the current window

*deviation: The mean of the absolute difference between the pixels and the region mean.

Binary bitmap **Line Mode** prefix packet

This packet is in raw byte form only. It starts off with the hex value 0xAA and then streams bytes, with each byte containing a mask for 8 pixels, from the top-left to the bottom-right of the image. (Each binary bit in the byte represents a pixel) The bitmap is then terminated with two 0xAAs. 0xAA is never transmitted as part of the data, so it should be used to signify termination of the binary bitmap. After the binary bitmap is complete, a normal tracking packet should follow.

(raw data: AA XX XX XX .... XX XX XX AA AA) C 45 72 65 106 18 51
(raw data: AA XX XX XX .... XX XX XX AA AA) C 46 72 65 106 18 52

Get mean **Line Mode** prefix packet

This packet prefix outputs the mean color of each row being processed. These packets are started with an 0xFE and terminate with an 0xFD. Each byte of data between these values represents the corresponding line's mean color value. Due to the serial transfer time, the vertical resolution is halved. After all rows have been completed, a normal tracking packet should follow.

(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8
(raw data: FE XX XX XX ... XX XX XX FD) M 45 56 34 10 15 8

Firmware Upgrades

CAUTION: This is not usually necessary because the firmware usually comes preprogrammed!

Once the board has been assembled and powered up, you can modify and download new firmware using the SX-Key downloading program. The SX-Key downloader and manual are available at:
www.parallaxinc.com/html_files/downloads/downloads_sx.htm
The firmware for the camera can be found at:
http://www.cs.cmu.edu/~cmucam/downloads.html

Downloading The Firmware

After downloading and installing the SX-Key software, it is necessary to load the CMUcam.hex firmware to the board.

step 1: Turn on the board's power.
step 2: Ensure that the programming switch is in the program mode (pushed towards the LEDs see Assembled view)
step 3: Plug in the SX-key, text side facing away from the board. (for more details see the hardware port section)
step 4: Open up the SX-Key software
step 5: Select "Device" from under the "Run" menu
step 6: Click on the "Load Hex" button
step 7: Go to the directory of the CMUcam.hex file and type its name into the dialog box. (It may not be visible)
step 8: Press "Open" to load-it
step 9: Press the "Program" button
step 10: Once programming has finished, turn off the power to the board
step 11: Unplug the SX-Key and switch the programming switch into the run position
step 12: Once the power switch is turned on again the firmware should be running and will greet you with

CMUcam v1.xx
:

Download Troubleshooting

If, when you tried to program, you see or experience:

SX-Key not found
-Make sure the comm port is set correctly and power is going to the camera
Vpp Generation Failed
-Ensure that the program switch is in the correct position
Programming Failed
- Try it again and if it still occurs consult the SX-Key manual
No response
-Make sure you switched the board out of program mode. Try power cycling the board.

General Troubleshooting

In Demo Mode, the light turns on for a second and then everything stops

When both the camera and servo are active, the power required is greater. Try using a battery or voltage source rated at a higher current.

When I run java I get: Exception in thread "main" java.lang.NoClassDefFoundError: CMUcamGUI

Chances are you are not in the CMUcamGUI directory. Type "dir" at the command line prompt and make sure that you see the CMUcamGUI.class file.

I see CMUcamGUI.java but I don't see the CMUcamGUI.class file

You should download a new copy of the GUI, because the .class files should be included. If you really need to recompile them, type "javac *.java" .

I get: 'java' is not recognized as an internal or external command, operable program or batch file.

This means that java is not correctly installed in your path. Try re-installing java and reading Sun's documentation about setting the "classpath" variables correctly.

The power LED does not glow

The board either has a fault, or your power supply is not generating enough power. Check the power supply and look over all of the soldier connections. Try unplugging all of the cables except power and turn it on again.

I get garbage output from the camera

Try turning the camera off and unplugging it for 10 seconds. Then plug it back in and try again.

I get wavy lines or a distorted black and white image when I call dumpframe

This is most likely due to power. Make sure that you have a high enough voltage and that you are getting a clean signal. Running the camera off of fresh batteries (not an AC adaptor) is a good way to test if this is the problem.

My processor can not keep up with the serial data stream

Try running the camera in poll mode and setting a delay mode value. See pages 19 and 22 for "PM" and "DM" details.

I don't seem to get any serial data

Make sure that the serial cable is connected on the CMUcam side correctly. If in doubt, try reversing it.

Why does SW keep giving me a NCK?

Make sure you are within the SW 1 1 80 143 bounds.

I see the CMUcam startup message, but then nothing happens

Check to make sure the transmit line on your serial cable is connected correctly.

Parts List

The following is a list of parts needed to assemble the CMUcam.

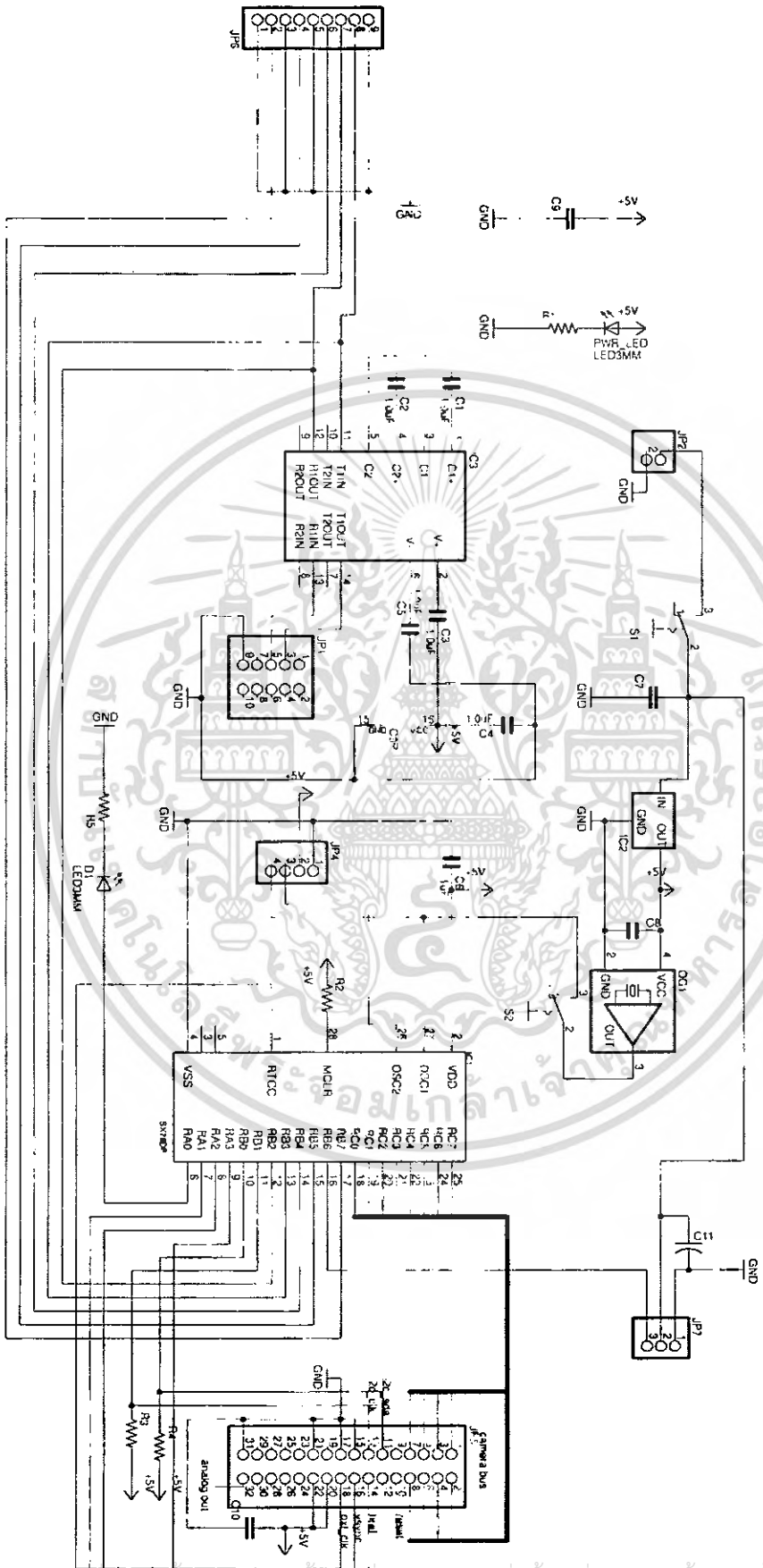
Qty	Item	Part no. (Digi-key)	Schematic Label	Unit Price
1	75 MIPS Ubicom SX28	SX28AC/DP-ND	IC1	5.18
1	OV6620 Eval Board *	BB048*		57.95
1	Maxim 232 Level Shifter	MAX232CPE-ND	IC3	3.31
1	SG-531 75.00 MHz crystal osc	SE2911-ND	QG1	5.85
1	5 Volt Regulator	LM2940CT-5.0-ND	IC2	1.80
3	4.7 Kohm 1/4 Watt Resistor	4.7KQBK-ND	R2 R3 R4	0.06
2	220 ohm 1/4 Watt Resistor	220QBK-ND	R1	0.06
6	1.0 uF Capacitor	P2105-ND	C1 C2 C3 C4 C5 C6	0.42
2	SPST Switch	EG1847-ND	S1 S2	1.11
1	3mm red LED	67-1125-ND	LED3MM	0.16
1	3mm green LED	67-1127-ND	D1	0.16
2	0.1uF Capacitor	P4923-ND	C9 C10	0.55
3	100 uF Capacitor	P5138-ND	C7 C8 C11	0.28
2	14 pin IC socket (to form a 28 pin socket)	AE8914-ND	IC1	0.32
1	8 pin IC socket **	ED3308-ND	QG1	0.42
1	16 pin IC socket	ED3316-ND	IC3	0.83
1	Double row female header	929852-01-36	JP5	3.35
1	Single row male header	929647-09-36-ND	JP1x2 JP3 JP4 JP7	1.83
(1)	Polarized 2 pin terminal housing	WM2700-ND		0.25
(2)	Crimp terminals	WM2200-ND		0.13
(1)	Polarized 2 pin terminal header	WM2000-ND		0.25
(1)	Female serial ribbon cable head	AFS09G-ND		6.14
(1)	Serial ribbon cable socket connector	ASC10G-ND		1.11

*Order Code BB048 at http://www.electronics123.com. Ask them for an evaluation board with an OV6620 from Omnivision and an IR coated lens.

**Use this for the oscillator by cutting off the inner 4 legs

( ) accessories that make interfacing easier, but are not required for functionality

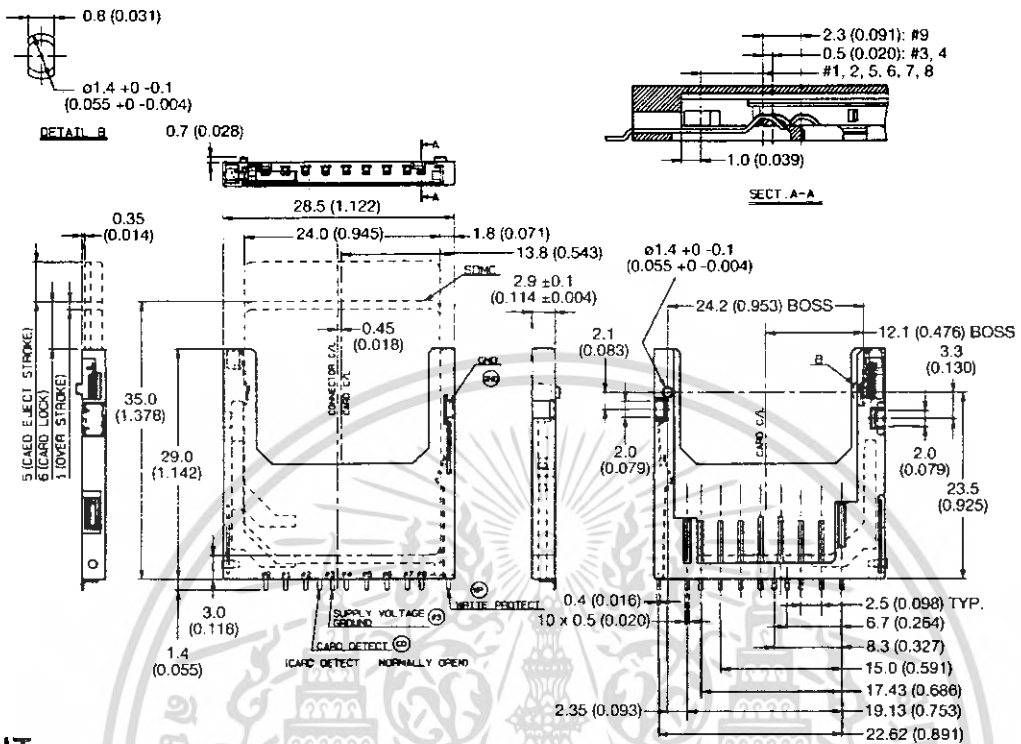
CMUcam Schematic



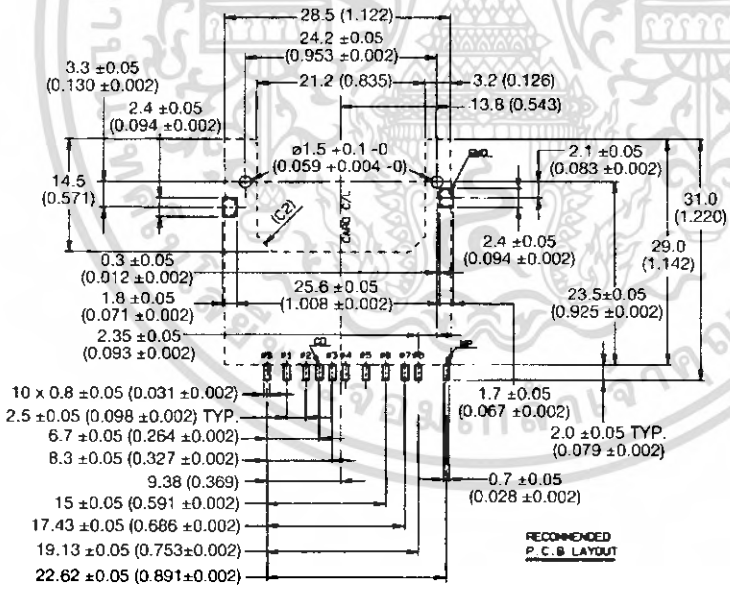
Secure Digital Memory Card Connectors

Series 5638 Spring Eject Type

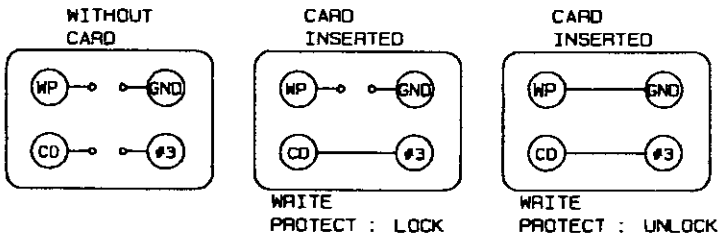
NORMAL



PCB LAYOUT



CIRCUIT

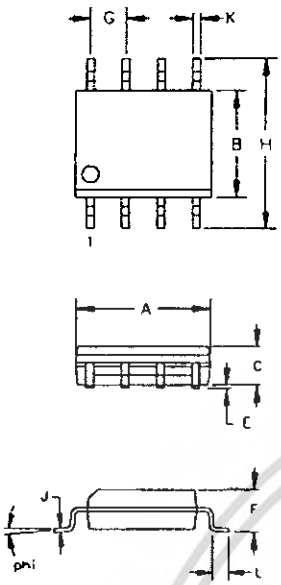


HOW TO ORDER

14 5638 009 511 862

250 pieces per Tape and Reel

DS1307Z 64 X 8 SERIAL REAL-TIME CLOCK
8-PIN SOIC (150-MIL) MECHANICAL DIMENSIONS



PKG	8-PIN (150 MIL)	
DIM	MIN	MAX
A IN.	0.188	0.196
MM	4.78	4.98
B IN.	0.150	0.158
MM	3.81	4.01
C IN.	0.048	0.062
MM	1.22	1.57
E IN.	0.004	0.010
MM	0.10	0.25
F IN.	0.053	0.069
MM	1.35	1.75
G IN.	0.050 BSC	
MM	1.27 BSC	
H IN.	0.230	0.244
MM	5.84	6.20
J IN.	0.007	0.011
MM	0.18	0.28
K IN.	0.012	0.020
MM	0.30	0.51
L IN.	0.016	0.050
MM	0.41	1.27
phi	0°	8°

56-G2008-001