เครื่องช่วยสื่อสารสำหรับผู้พิการทางหู

Communication Aid Machine for Hearing Impaired Person

นาย กรวุฒิ ม่วงไทย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขา วิชาอิเล็กทรอนิกส์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

เครื่องช่วยสื่อสารสำหรับผู้พิการทางหู

Communication Aid Machine for Hearing Impaired Person

จัดทำโดย

นาย กรวุฒิ ม่วงไทย รหัส 45010009

อาจารย์ที่ปรึกษา

ดร. กิติพล ชิตสกุล

ปริญญานิพนธ์นี้สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขา วิชาอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องช่วยสื่อสารสำหรับผู้พิการทางหู

ผู้จัดทำ

1.นาย กรวุฒิ ม่วงไทย รหัส 45010009

.....อาจารย์ที่ปรึกษา

(ดร. กิตติพล ชิตสกุล)

I

เครื่องช่วยสื่อสารสำหรับผู้พิการทางหู

นาย กรวุฒิ ม่วงไทย รหัส 45010009
ดร.กิติพล ชิตสกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้นำเสนอแนวคิดและการสร้างเครื่องช่วยสื่อสารให้กับผู้พิการทางการได้ยินรวมถึงผู้พิการทางการพูด เครื่องมือนี้จะเก็บข้อความสั้น ๆ ซึ่งจำเป็นสำหรับการสื่อสารไว้ในหน่วยความจำ และจะแสดงเป็นภาพกราฟฟิคและข้อความบนหน้าจอกราฟฟิก ภาพกราฟฟิกจะถูกเก็บไว้ ใน SD Memory Card โดยผู้ใช้จะเรียกมาได้โดยการกดสวิทช์บนตัวเครื่อง โครงงานนี้ใช้ไมโครคอนโทรลเลอร์ PSOC ในการติดต่อกับหน่วยความจำ SD Memory Card และการติดต่อกับหน้าจอ กราฟฟิก ของโทรศัพท์ โนเกียร์ 5110 และในการสแกนคีย์สวิทช์ เครื่องต้นแบบเก็บภาพและข้อความได้ทั้งสิ้นไว้ 30 ข้อความ 6 กลุ่มความหมาย

Communication Aid Machine for Hearing Impaired Person

Mr.Korawut Muangthai ID.45010009
Dr. Kitiphol Chitsakul Advisor
Educational Year 2006

Abstract

This thesis presents concept and implementation of a communication aid machine for the hearing impaired person including also the speaking impaired person. The short sentences with graphic symbols are stored in the memories which the sentence is shown on a built in monitor when the buttons are pressed to select. A Nokia, series 5110, display module is used for displaying graphic image, stored in an SD memory card, corresponding the sentence needed to communicate. We using PSOC microcontroller to control all of function of the machine. The prototype stores totally 30 images grouped in 6 groups.

กิตติกรรมประกาศ

ขอขอบคุณอาจารย์ ดร. กิติพล ชิตสกุล (อาจารย์ที่ปรึกษา) ที่คอยให้คำแนะนำในเรื่องต่างๆ รวมทั้งให้โอกาสข้าพเจ้าได้ทำโครงงานนี้ ขอขอบคุณเหล่าคณาจารย์ที่สอนวิชาความรู้ต่างๆ ขอขอบคุณคุณเรณู (พี่ธุรการ) ที่ช่วยในการดำเนินเรื่องและแก้ปัญหาในหลายๆสิ่ง ขอขอบคุณเพื่อนๆที่ร่วมกันแบ่งปันความรู้ช่วยเหลือและแก้ไขข้อบกพร่องและสุดท้ายขอกราบขอบพระคุณคุณแม่ที่คอยเป็นห่วง และให้กำลังใจข้าพเจ้าเสมอมาจนทำให้โครงงานนี้สำเร็จลุล่วงไปด้วยดี

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาของโครงงาน

ในปัจจุบันการติดต่อสื่อสารระหว่างผู้พิการทางเสียงกับคนปกตินั้น ยังไม่ค่อยสะดวกนัก จึงควรมีอุปกรณ์เพื่อช่วยให้ผู้พิการทางเสียงสามารถติดต่อสื่อสารกับคนปกติได้ง่ายขึ้น และดำเนินชีวิตทั่วไปได้อย่างสะดวกสบายยิ่งขึ้น

1.2 วัตถุประสงค์

สร้างอุปกรณ์แสดงภาพบนหน้าจอกราฟฟิก ของโทรศัพท์ โนเกียร์ 3310 โดยนำข้อมูลภาพที่เก็บไว้ใน SD Card มาแสดงได้อย่างถูกต้อง และสามารถทำการปรับเปลี่ยนข้อมูลภาพใน SD Card ได้

1.3 ขอบเขต

ทำการเขียนโปรแกรมอ่านข้อมูลที่อยู่ใน SD SARD โดยไมโครคอนโทรลเลอร์ (PSOC) แล้วทำการส่งข้อมูลที่ได้ให้มาปรากฏเป็นรูปภาพบนหน้าจอกราฟฟิก ของโทรศัพท์ โนเกียร์ 5110 ให้ได้ตามวัตถุประสงค์

1.4 ประโยชน์ที่คาดว่าจะได้รับ

เนื่องจากปัจจุบันผู้พิการทางการได้ยินมักพบปัญหาในการติดต่อสื่อสารกับผู้คนทั่วไป จึงเกิดแนวความคิดทำโครงงานนี้ เพื่อใ ห้ผู้พิการทางการได้ยินมีความสะดวกในการติดต่อสื่อสารมากขึ้น ผู้จัดทำก็หวังว่าโครงงานนี้จะเกิดประโยชน์ไม่มากก็น้อย

ปริญญานิพนธ์นี้ได้อธิบายขั้นตอน และวิธีคิดในการออกแบบอุปกรณ์แสดงภาพบนหน้าจอกราฟฟิก โดยใช้ไมโครคอนโทรลเลอร์ควบคุมการทำงานต่างๆและผลการทดสอบการทำงาน โดยจะมีเนื้อหาแบ่งออกเป็นบทต่างๆดังนี้

บทที่ 2 ทฤษฎี โดยจะกล่าวถึงการทำงาน และการใช้งานต่างๆของไมโครคอนโทรลเลอร์ หน้าจอแสดงผลที่เป็นหน้าจอกราฟฟิก ของโทรศัพท์ โนเกียร์ 5110 และหน่วยความจำ SD Card เพื่อนำมาใช้ในการออกแบบวงจร และออกแบบการเขียนโปรแกรม

บทที่ 3 การออกแบบ โดยจะกล่าวถึงการออกแบบส่วนต่างๆของวงจร และโปรแกรมที่ใช้ในการควบคุม

บทที่ 4 ผลการทดลอง โดยจะกล่าวถึงผลการทดสอบจากการเขียนโปรแกรม

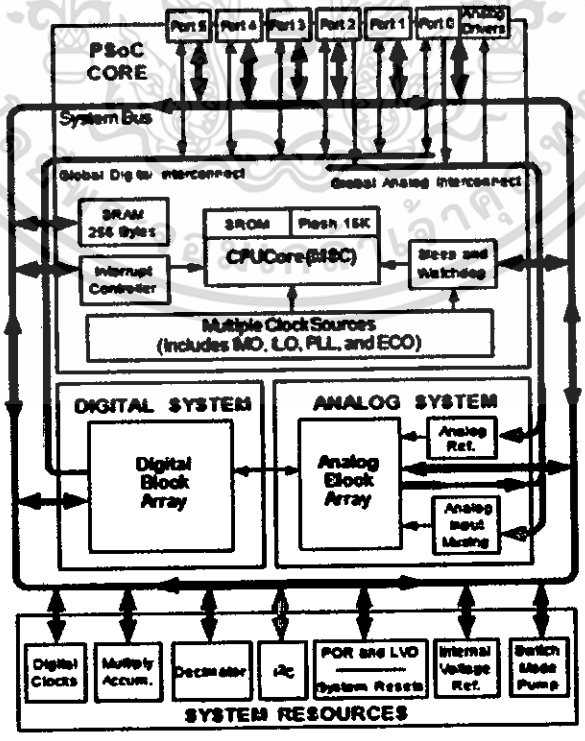
บทที่ 5 สรุปและวิจารณ์ กล่าวถึงปัญหาที่พบในการทำงาน เพื่อนำไปพัฒนาและปรับปรุงต่อไปในอนาคต

บทที่ 2
ทฤษฎีและหลักการทำงานส่วนต่างๆของไมโครคอนโทรลล์เลอร์

2.1 ทฤษฎีและหลักการทำงานส่วนต่างๆของไมโครคอนโทรลล์เลอร์

2.1.1 PSoC MCU

โครงงานนี้เป็นการประยุกต์ใช้อุปกรณ์ควบคุมไมโครคอนโทรเลอร์ PSoC (Programmable system on chip) ซึ่งเป็นไมโครคอนโทรลเลอร์ของ Cypress Microsystems โดยเป็นไมโคร คอนโทรลเลอร์ ที่มีการประมวลผลข้อมูลแบบ 8 บิต เหมือนไมโครคอนโทรลเลอร์ทั่ว ๆ ไป แต่ PSoC มีจุดเด่นหลายประการที่แตกต่างจากอุปกรณ์ไมโครคอนโทรลเลอร์ตระกูลอื่น ๆ คือ PSoC MCU จะทำการรวมเอาการออกแบบทั้งทางดิจิตอล และด้านอนาลอกมาไว้ด้วยกันภายในตัว PSoC MCU ทำให้การออกแบบในการใช้งานทางดิจิตอล และทางด้านอนาลอกสามารถทำได้ง่ายขึ้นและสะดวกสบายยิ่งขึ้น อีกทั้ง ยังทำให้การออกแบบในด้านต่าง ๆ มีขนาดที่เล็กลงโดยเฉพาะด้านอนาลอก ซึ่งมักจะมีขนาดค่อนข้างใหญ่ แต่เมื่อถูกรวมอยู่ภายใน PSoC MCU แล้วทำให้ไม่ต้องทำการต่อวงจรภายนอกเพิ่มขึ้นอีกจึงทำให้วงจรมีขนาดที่เล็กลง นอกจากนี้ยังมีฟังก์ชัน In-System Serial Programming (ISSP) ที่สามารถทำการโปรแกรมซอร์สโค้ดที่ได้ออกแบบลงไปในหน่วยความจำโปรแกรม (Flash Memory) ภายในตัวชิพได้ ซึ่งช่วยให้การพัฒนาโปรแกรมโดยใช้ PSoC MCU มีความสะดวกสบายยิ่งขึ้น



รูปที่ 2.1 แสดงบล็อกการทำงานภายในของ PSoC MCU

จุดเด่นของ PSoC MCU เมื่อเทียบกับไมโครคอนโทรลเลอร์ชนิดอื่น ๆ มีดังนี้

1. User Modules สามารถเลือกใช้ทรัพยากรของระบบได้ตามที่ต้องการทั้ง อนาลอกและดิจิตอล ซึ่งจะไม่ถูกจำกัดด้วยโครงสร้างทางฮาร์ดแวร์เหมือนกับไมโครคอนโทรลเลอร์ชนิดอื่น ๆ

2. API (Application Programming Interface) สนับสนุนการพัฒนาโปรแกรมด้วยฟังก์ชั่น API ซึ่งช่วยให้ผู้พัฒนาโปรแกรมสามารถเขียนออกแบบโปรแกรมได้โดยง่าย

3. ISRs (Interrupt Service Routines) รองรับการทำงานแบบอินเตอร์รัพท์

4. Interconnect device interface สามารถทำการเชื่อมต่อสัญญาณต่าง ๆ ได้อย่างอิสระไม่ถูกกำหนดตายตัวตามฮาร์ดแวร์ เหมือนกับไมโครคอนโทรลเลอร์ชนิดอื่น ๆ

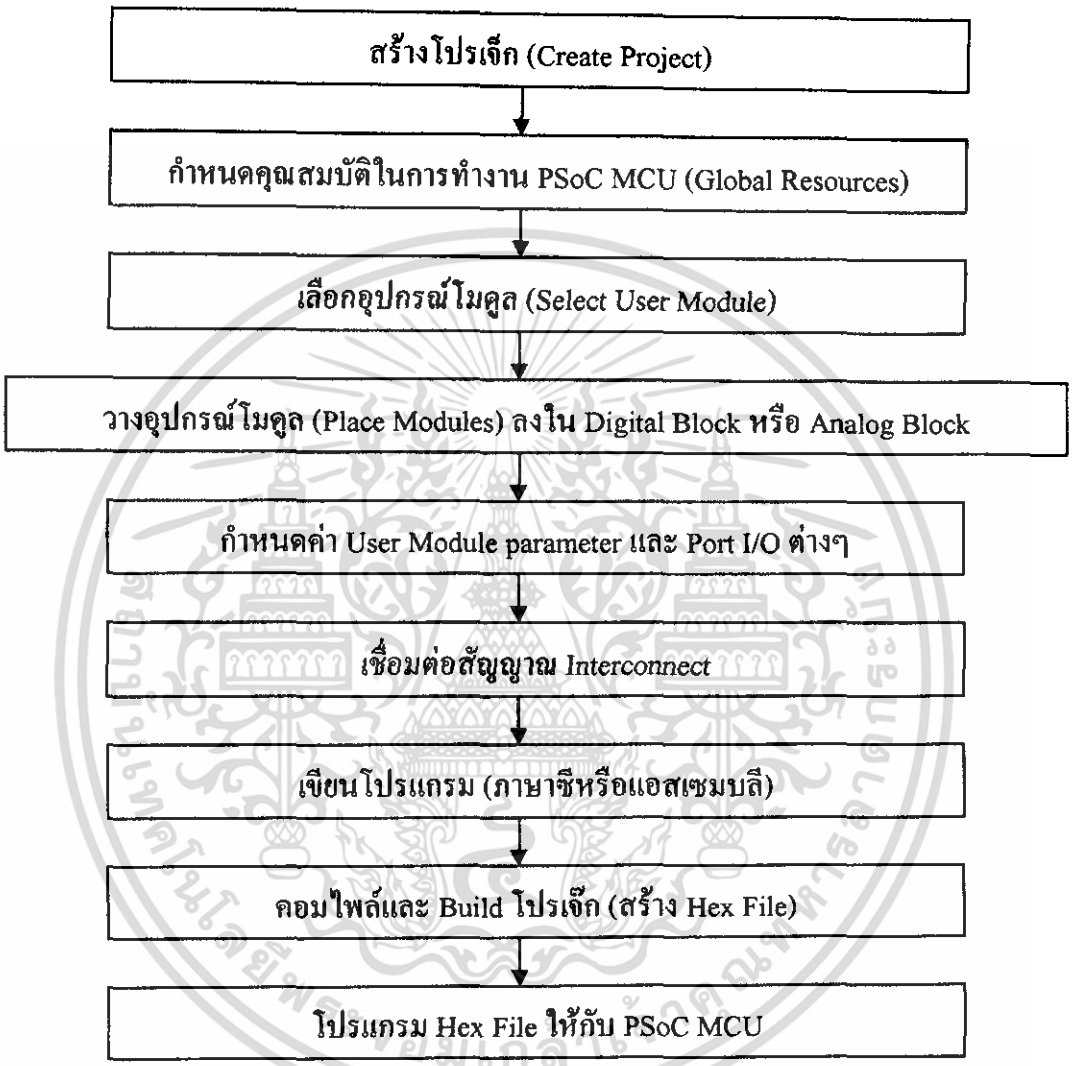
จากที่กล่าวมาเบื้องต้นทั้งหมดเป็นเพียงคุณสมบัติคร่าว ๆ เท่านั้น ซึ่งคุณสมบัติอื่น ๆ ที่เหลือก็จะคล้าย ๆ กับไมโครคอนโทรลเลอร์ชนิดอื่นๆ เช่น Sleep, watchdog, Power on Reset (POR), SPI, UART และ I^2C เป็นต้น จะเห็นได้ว่า PSoC MCU นั้นไม่ด้อยไปกว่าไมโครคอนโทรลเลอร์อื่น ๆ เลย

2.1.2 รูปแบบการใช้งานและการพัฒนา PSoC MCU

ไมโครคอนโทรลเลอร์ PSoC นั้น สนับสนุนระบบการทำงานทั้งทางด้านดิจิตอล และอนาลอก โดยในระบบของดิจิตอล (Digital System) และอนาลอก (Analog System) ได้ถูกออกแบบเป็นบล็อกโมดูลซึ่งจะเรียกว่า บล็อกดิจิตอล (Digital Blocks) และบล็อกอนาลอก (Analog Blocks) โดยบล็อกเหล่านี้จะรองรับการนำเอาโมดูลต่างๆ มาใช้งานเปรียบเสมือนกับเป็นพื้นที่ว่างๆ สำหรับต่อจิ๊กซอว์ โดยชิ้นส่วนของจิ๊กซอว์ก็คือโมดูลต่างๆ เช่น ADC, DAC, I^2C, PWM, UART, SPI เป็นต้น โดยผู้ใช้สามารถกำหนดได้เองว่าจะนำเอาโมดูลใดมาใช้งานและนอกจากนี้ผู้ใช้งานยังสามารถกำหนดการเชื่อมต่อสัญญาณต่างๆ (Programmable Interconnect) ภายในได้เองอีกด้วย เสมือนกับว่าผู้ใช้งานสามารถทำการออกแบบได้เองตั้งแต่ ฮาร์ดแวร์ ไปจนถึง ซอฟต์แวร์ ซึ่งถือได้ว่าเป็นความสามารถหนึ่งที่เหนือกว่าไมโครคอนโทรลเลอร์ชนิดอื่น ๆ ที่ทรัพยากรทุกอย่างถูกกำหนดไว้ตายตัวไม่สามารถเปลี่ยนแปลงได้

ภาษาที่ใช้ในการออกแบบพัฒนาการทำงานของ PSoC MCU ปัจจุบันจะมีอยู่ด้วยกัน 2 ภาษา คือ ภาษาแอสเซมบลีและภาษาซี เนื่องจากการพัฒนาโปรแกรมของ PSoC MCU ส่วนใหญ่จะทำโดยการเรียกใช้งานฟังก์ชัน API และการกำหนดคุณสมบัติต่าง ๆ เช่นความถี่สัญญาณนาฬิกา Sleep, Watchdog, Supply Voltage และอื่นๆ รวมทั้งยังสามารถทำได้จากหน้าต่าง Device Editor ของซอฟต์แวร์ PSoC Designer ทำให้เราไม่จำเป็นต้องทราบรีจิสเตอร์ต่าง ๆ มากนักจึงจะไม่ขอกล่าวถึงรายละเอียดในการใช้งานรีจิสเตอร์มากนัก แต่จะกล่าวถึงเฉพาะในส่วนที่

เกี่ยวข้องกับโครงงานนี้เท่านั้น และจะอธิบายการใช้งานในการพัฒนาโปรแกรมเป็นขั้นตอนดังต่อไปนี้ คือ



รูปที่ 2.2 แสดงลำดับขั้นตอนการพัฒนาโปรแกรมให้กับ PSoC MCU ด้วย PSoC Designer แบบคร่าว ๆ

2.1.3 คุณสมบัติของ PSoC MCU ตระกูล CY8C29466

1. สถาปัตยกรรมแบบ Havard Architecture Processor
2. ความเร็วของ M8C Processor สูงสุด 24 MHz
3. 8x8 Multiply, 32Bit Accumulate
4. Low Power at High Speed

5. ทำงานในช่วงแรงดัน 3.0 V ถึง 5.25 V

6. สามารถทำงานที่แรงดันต่ำสุดที่ 1.0V โดยใช้วงจร Switch Mode Pump (SMP)

7. 12 Analog PSoC Blocks รองรับการใช้งานทางด้านอนาลอก เช่น

- ADCs ความละเอียดสูงสุดถึง 14 บิต
- DACs ความละเอียดสูงสุดถึง 9 บิต
- วงจรเกณฑ์การขยาย (Programmable Gain Amplifier)
- วงจรฟิลเตอร์และวงจรคอมพาราเตอร์ (Programmable Filters and Comparater

8. 8 Digital PSoC Blocks รองรับการใช้งานทางด้านดิจิตอล

- Timers, Counters และ PWMs ขนาด 8 ถึง 32 บิต
- CRC และ PRS โมดูล
- UARTs แบบ Full – Duplex สูงสุด 2 ช่อง
- SPI โมดูลเป็นได้ทั้งแบบ Master และ Slave
- ดิจิตอลบล็อกต่างๆ สามารถเชื่อมต่อไปยังขาสัญญาณ GPIO ได้ทุกขาสัญญาณ

9. สามารถกำหนดขนาดความถี่ของสัญญาณนาฬิกาภายในได้หลายระดับ

10. สัญญาณนาฬิกาภายในขนาด 24/48 MHz ค่าความคลาดเคลื่อน +/-2.5%

11. สามารถเลือกแหล่งกำเนิดสัญญาณ 24/48 MHz จากออสซิลเลเตอร์ 32 kHz ภายในหรือภายนอกได้

12. สามารถรับสัญญาณออสซิลเลเตอร์จากภายนอกได้สูงสุด 24 MHz

13. มีแหล่งกำเนิดสัญญาณนาฬิกาภายในให้กับ Watchdog และ Sleep

14. หน่วยความจำโปรแกรมแบบ Flash ขนาด 16K Byte สามารถ Erase/Write ได้ถึง 5,000 ครั้ง

15. หน่วยความจำข้อมูล SRAM ขนาด 256 Byte

16. ฟังก์ชันการโปรแกรมภายใน ISSP (In-System Serial Programming)

17. สามารถเปลี่ยนแปลงข้อมูล Flash Memory เฉพาะบางส่วนได้

18. สามารถตั้งค่าระบบป้องกันข้อมูลได้ (Flash Security)

19. หน่วยความจำ EEPROM (ใช้จาก Flash Memory)

20. ขาสัญญาณ GPIO จำนวน 24 ขาสัญญาณ

21. สามารถกำหนดคุณสมบัติของขาสัญญาณต่างๆ ได้ (GPIO Pin Configurations)

22. GPIO สามารถจ่ายกระแสสูงสุดได้ถึง 24 mA

23. สามารถกำหนด Drive Mode ของสัญญาณ GPIO ได้ เช่น Pull up, Pull down, HighZ, Strong หรือ Open Drain

24. ขาสัญญาณอินพุตอนาลอกสูงสุด 12 สัญญาณ (จากขาสัญญาณของ GPIO)

25. ขาสัญญาณอนาลอกเอาต์พุตสูงสุด 4 ขาสัญญาณ (จากขาสัญญาณ GPIO) จ่ายกระแสได้ถึง 40 mA

26. สามารถกำหนดการอินเตอร์รัพท์ได้ทุกขาสัญญาณของ GPIO

27. I^2C โหมด Slave, Master และ Multi-Master ความเร็วสูงสุด 400 kHz

28. Wahctdog และ Sleep Timer

29. สามารถตั้งค่าระดับการตรวจจับแรงดันต่ำได้ (Low Voltage Detection)

30. Integrated Supervisory Circuit

31. On-Chip Precision Voltage Reference

จะเห็นได้ว่า PSoC MCU เป็นไมโครคอนโทรลเลอร์ที่มีคุณสมบัติที่ถือได้ว่าครบถ้วนมากไม่ได้ด้อยไปกว่าไมโครคอนโทรลเลอร์ตระกูลอื่นๆ เลย ในการที่มีคุณสมบัติมากมายอาจถูกมองว่าการออกแบบยุ่งยาก แต่ในความจริงแล้ว PSoC MCU ไม่ได้ออกแบบยุ่งยากอย่างที่คิด เนื่องจากสามารถใช้ซอฟต์แวร์ PSoC Designer ในการออกแบบได้ซึ่งซอฟต์แวร์ดังกล่าวนี้จะมีการออกแบบเป็นลักษณะของกราฟิกหรือเป็นแบบวิสชวล (Visual) ส่วนในการเขียนโปรแกรมก็จะเป็นลักษณะการใช้งานฟังก์ชัน API (Application Programming Interface) ที่ PSoC Designer ได้จัดเตรียมไว้ให้แล้ว ทำให้ลดความยุ่งยากไปได้มากเลยทีเดียว

2.1.4 ฟังก์ชันและโครงสร้างของไมโครคอนโทรลเลอร์ PSoC

ไมโครคอนโทรลเลอร์ PSoC นั้นมีฟังก์ชันการใช้งานมากมายหลายฟังก์ชันด้วยกันในหัวข้อนี้จะขอกล่าวถึงเฉพาะฟังก์ชันและโครงสร้างสถาปัตยกรรมของไมโครคอนโทรลเลอร์ PSoC ที่เกี่ยวข้องกับโครงงานนี้เท่านั้น โดยในส่วนที่โครงงานนี้จะขอกล่าวถึงโครงสร้างภายในเพียงบางส่วนของไมโครคอนโทรลเลอร์ PSoC เท่านั้น คือ

PSoC Core

เป็นส่วนหลักของการประมวลผล ซึ่งจะดูแลส่วนต่าง ๆ เช่น การประมวลผลคำสั่ง, การจัดเก็บข้อมูลในหน่วยความจำ SRAM, ควบคุมการอินเตอร์รัพท์, Sleep, Watchdog times และการเลือกแหล่งสัญญาณนาฬิกา (Clock sources) เป็นต้น ซึ่งจะเรียกส่วนที่จัดการส่วนต่าง ๆ เหล่านี้ว่า M8C ซึ่งเป็นสถาปัตยกรรมไมโครโปรเซสเซอร์ 8 บิต แบบ Harvard นอกจากนี้

ภายใน PSoC Core ยังมีหน่วยความจำ SROM และ FLASH Memory สำหรับใช้เก็บโปรแกรมคำสั่งอีกด้วย

Digital System

เป็นระบบของดิจิตอลบล็อกที่วางในรูปแบบของบล็อกอาเรย์ (array block) ในส่วนของ DIGITAL SYSTEM การวางของบล็อกอาเรย์จะวางแถวละ 4 บล็อก สำหรับ CY8C27443 จะมีอาเรย์บล็อกจำนวน 2 แถวรวมแล้วจะมีทั้งหมด 8 ดิจิตอลบล็อก ส่วน PSoC MCU เบอร์อื่น ๆ จะมีขนาดแตกต่างกันออกไป โดยบล็อกเหล่านี้ก็จะรองรับการใช้งานในส่วนของโมดูลดิจิตอลต่าง ๆ

Analog System

ระบบของอนาลอกจะประกอบไปด้วยส่วนต่างๆ คือ อนาลอกบล็อก (Analog Blocks), แรงดันอ้างอิงอนาลอก (Analog Ref) และวงจรเลือกสัญญาณอินพุตอนาลอก (Analog Input Muxing) โดยอนาลอกบล็อกจะมีอยู่ 2 ประเภทด้วยกัน คือ CT (Continuous Time) และ SC (Switched Capacitor) ซึ่งอนาลอกบล็อกจะถูกจัดเรียงเป็น 4 คอลัมน์ (Column) ในหนึ่งคอลัมน์ก็จะประกอบด้วย CT หนึ่งบล็อก และ SC อีกสองบล็อก ซึ่งบล็อกเหล่านี้จะมีไว้สำหรับรองรับการทำงานของโมดูลอนาลอกโดยการจัดวางโมดูลลงในบล็อกต่าง ๆ นั้น ก็ขึ้นอยู่กับความเหมาะสมของแต่ละโมดูล

System Resources

System Resources ก็คือ ฮาร์ดแวร์ต่างๆ ของระบบ เช่น Digital clocks, Multiply accumulate (MAC), Decimator, I^2C, System reset, Internal voltage reference, Switch mode pump (SPM) เป็นต้น

หน่วยความจำ (Memory)

หน่วยความจำเป็นองค์ประกอบหนึ่งที่มีความจำเป็นต่อการทำงานของไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ โดยจะมีการแบ่งหน่วยความจำออกเป็นประเภทต่าง ๆ ตามคุณสมบัติและการใช้งาน ซึ่งในส่วนของสถาปัตยกรรม M8C ก็จะแบ่งหน่วยความจำออกเป็น 3 ส่วน คือ ROM, RAM และ Register

ขาสัญญาณอินพุต/เอาต์พุต (GPIO : General Purpose IO)

ในหัวข้อนี้จะกล่าวถึงคุณสมบัติทางด้านขาสัญญาณ I/O (Input/Output) ของ PSoC MCU ซึ่งขาสัญญาณของ PSoC MCU นั้น สามารถทำงานได้ทั้งในส่วนของ ขาสัญญาณดิจิตอล และขาสัญญาณอนาลอกซึ่งในการใช้งานจะผ่านทางรีจิสเตอร์ต่างๆ

Address	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Access
0,xxh	PRTxDR	Data Register								RW:00
0,xxh	PRTxIE	Bit Interrupt Enables								RW:00
0,xxh	PRTxGS	Global Select								RW:00
0,xxh	PRTxDM2	Drive Mode 2								RW:FF
1,xxh	PRTxDM0	Drive Mode 0								RW:00
1,xxh	PRTxDM1	Drive Mode1								RW:FF
1,xxh	PRTxIC0	Interrupt Control 0								RW:00
1,xxh	PRTxIC1	Interrupt Control 1								RW:00

ตารางที่ 2.1 แสดงขาสัญญาณอินพุต/เอาต์พุต GPIO

- หมายเหตุ เนื่องจากในการพัฒนาจริง จะอาศัยซอฟต์แวร์ PSoC Designer ในการออกแบบ ทำให้เราไม่จำเป็นต้องทราบรายละเอียดของรีจิสเตอร์ต่างๆ เหล่านี้มากนัก

ขาสัญญาณอินพุต/เอาต์พุต หรือขาสัญญาณ GPIO ของไมโครคอนโทรลเลอร์ PSoC แต่ละขาสัญญาณนอกจากการทำงานในโหมด I/O ปกติแล้ว บางขาสัญญาณยังสามารถทำงานในหน้าที่อื่นๆ ได้อีกด้วย ดังเช่น

ชื่อขาสัญญาณ	คำอธิบาย	Input/Output
SMP	Switch Mode Pump	Power
Vdd	Supply Voltage	Power
Vss	Ground	Power
XRES	External Reset (Active High)	Input/Output
P0[0]-P0[1]	Port 0[0], Port0[1], Analog Input	Input/Output
P0[2]-P0[5]	Port 0[2],[3],[4],[5] Analog Input/Output	Input/Output
P0[6]-P0[7]	Port 0[6],[7] Analog Input	Input/Output
P1[0]	Port1[0],XTAL Out/SDATA/I2C SDA	Input/Output
P1[1]	Port 1[1], XTAL In/SCLK/I2C SCL	Input/Output

ชื่อขาสัญญาณ	คำอธิบาย	Input/Output
P1[2]	Port1[2]	Input/Output
P1[3]	Port1[3]	Input/Output
P1[4]	Port1[4],EXTCLK	Input/Output
P1[5]	Port1[5],I2C SDA	Input/Output
P1[6]	Port1[6]	Input/Output
P1[7]	Port1[7],I2C SCL	Input/Output
P2[0]-P2[3]	Port2[0],[1],[2],[3],Non-Multiplexed Analog Input (Switch capacitor)	Input/Output
P2[4]	Port2[4],External AGND	Input/Output
P2[5]	Port2[5]	Input/Output
P2[6]	Port2[6],External VREF	Input/Output
P2[7]	Port2[7]	Input/Output
P3[0]-P3[7]	Port3[0],[1],[2],[3],[4],[5],[6],[7]	Input/Output
P4[0]-P4[7]	Port4[0],[1],[2],[3],[4],[5],[6],[7]	Input/Output
P5[0]-P5[3]	Port5[0],[1],[2],[3]	Input/Output

ตารางที่ 2.2 แสดงหน้าที่ขาสัญญาณต่างๆ ของ Cy8C29466

จะเห็นว่าโครงสร้างขาสัญญาณ GPIO ของ PSoC MCU มีความซับซ้อนมาก ทั้งนี้เพราะ PSoC MCU ได้ออกแบบโครงสร้างขาสัญญาณ ให้สามารถทำงานได้ในหลากหลายคุณสมบัติ โดยผู้ใช้งาน (User) สามารถกำหนดความต้องการในการใช้งานได้ โดยเราสามารถกำหนดคุณสมบัติการทำงานของขาสัญญาณในโหมดต่าง ๆ ได้ดังนี้

- Pull Down (Resistive pull down) โหมดนี้ขาสัญญาณ I/O จะถูกต่อผ่านตัวต้านทานลงกราวด์
- Strong (Strong Drive) โหมดนี้เหมาะสำหรับใช้งานเป็นเอาต์พุตดิจิตอล
- High Z (High Impedance) โหมดนี้ที่ขาสัญญาณ I/O จะมีความต้านทานสูงเหมาะสำหรับการใช้งานเป็นอินพุต
- Pull Up (Resistive Pull Up) โหมดนี้ขาสัญญาณ I/O จะถูกต่อผ่านตัวต้านทานไปยัง Vcc

- Open Drain High สถานะของสัญญาณ I/O เป็นแบบ Open Drain High
- Strong Slow (Slow Strong drive) สถานะสัญญาณ I/O เป็นแบบ Strong Slow
- High Z Analog เป็นสถานะความต้านทานสูงแบบอนาลอก ซึ่งจะเป็นค่าสถานะเริ่มต้น (Default) หลังจากเกิดการ รีเซต (reset state)

DM2	DM1	DM0	Drive Mode	Data=0	Data=1
0	0	0	Resistive Pull Down	Resistive	Strong
0	0	1	Strong Drive	Strong	Strong
0	1	0	High Impedance	HI-Z	HI-Z
0	1	1	Resistive Pull Up	Strong	Resistive
1	0	0	Open Drain, Drive High	HI-Z	Strong (Slow)
1	0	1	Slow Strong Drive	Strong (Slow)	Strong (Slow)
1	1	0	High Impedance Analog	HI-Z	HI-Z
1	1	1	Open Drain, Drive Low	Strong (Slow)	HI-Z

ตารางที่ 2.3 การกำหนด Drive Mode ของ GPIO โดยผ่านรีจิสเตอร์ PRTxDMx[2:0]

การกำหนดคุณสมบัติของขาสัญญาณเหล่านี้ สามารถทำได้สองวิธี คือ การกำหนดที่ค่ารีจิสเตอร์ (PRTxDMx) และอีกวิธีก็คือการกำหนดโดยใช้โปรแกรม PSoC Designer ตรงส่วนของ Device Editor Configuration ซึ่งจะเรียกว่าการกำหนด Configure I/O Pins ซึ่งจะช่วยลดความยุ่งยากในการเขียนโปรแกรมลงได้เป็นอย่างดี

ขาสัญญาณ I/O ของ PSoC MCU จะประกอบไปด้วยบัฟเฟอร์อินพุต และวงจรขับทางด้านเอาต์พุตโดยขาสัญญาณ I/O เหล่านี้จะถูกจัดไว้เป็นพอร์ต ซึ่งปกติ 1 พอร์ต จะมีทั้งหมด 8 บิต แต่จะมีบางกรณีที่พอร์ตนั้นมีขาสัญญาณไม่ถึง 8 บิต ขาสัญญาณ I/O ต่างๆ เหล่านี้สามารถทำงานในลักษณะต่าง ๆ ดังนี้

- Digital IO : เป็นขาสัญญาณดิจิตอล อินพุต/เอาต์พุต สามารถควบคุมการทำงานได้โดยการผ่านค่าให้กับรีจิสเตอร์ PRTxDR
- Global IO : เป็นขาสัญญาณดิจิตอล อินพุต/เอาต์พุต ที่เชื่อมโยงระหว่าง Digital PSoC Block
- Analog IO : เป็นขาสัญญาณ อินพุต/เอาต์พุต ที่เชื่อมโยงระหว่าง Analog PSoC Block

2.1.5 การใช้งาน SPIM (Serial Peripheral Interconnect Master)

ในโครงงานนี้เป็นการนำโมดูลในส่วนของ SPIM มาใช้ในการติดต่อกับการแสดงผลบนหน้าจอ LCO NOKIA 5110 ที่เป็นหน้าจอกราฟิกขนาดความละเอียด 48x84 จุด ซึ่งจะกล่าวถึงในหัวข้อต่อไป ในส่วนของโมดูล SPIM (Serial Peripheral Interconnect Master) จะขออธิบายรายละเอียดดังต่อไปนี้

โมดูล SPIM คือ โมดูลการสื่อสารอนุกรมที่มีโปรโตคอลการสื่อสารเป็นแบบ SPI โดยจะเป็นการส่งแบบเข้าจังหวะ 8 บิต ข้อมูลฟูลดูเพล็กซ์ (Full-duplex) โมดูลนี้จะทำงานเป็นแบบมาสเตอร์ซึ่งสามารถที่จะนำไปต่อเข้ากับอุปกรณ์ SPI ที่เป็นสลาฟมากกว่าหนึ่งตัว ขึ้นอยู่กับการนำไปใช้งานโดยภายในโมดูล SPIM จะประกอบไปด้วย Tx Buffer, Rx Buffer, Control, และ shift registers การใช้งานโมดูลจะเรียกใช้งานผ่านฟังก์ชัน API โดยค่าเริ่มต้นของโมดูล SPIM จะถูกเซตมีการส่งข้อมูลแบบส่งบิตนัยสำคัญต่ำสุดก่อน (LSB First) และจะสนับสนุนการทำงานของสัญญาณในโหมดต่าง ๆ ตั้งแต่โหมด 0,1,2 และ 3 ซึ่งควรจะมีการเซตโหมดของสัญญาณนาฬิการะหว่างอุปกรณ์ PSI Master และอุปกรณ์ SPI Slave ให้มีสัญญาณนาฬิกาเป็นโหมดเดียวกัน โดยโหมดการทำงานของ SPI จะเป็นดังตารางที่ 2.4

SPI Mode			
Mode	**SCLK Edge Performing Data Latch**	**Clock Polarity**	**Notes**
0	Leading	Non-inverting	Leading edge latches data. Data changes on trailing edge of clock.
1	Leading	Inverted	
2	Trailing	Non-inverting	Trailing edge latches data. Data changes on leading edge.
3	Trailing	Inverted	

ตารางที่ 2.4 แสดงโหมดการทำงานของฟังก์ชัน SPI

ขาสัญญาณการเชื่อมต่อ

- SCLK คือ ขาสัญญาณนาฬิกาที่ให้จังหวะในการรับส่งข้อมูลของ SPI
- MOSI (Master Out Slave In Data) คือ ขาสัญญาณเอาต์พุตข้อมูลของตัวอุปกรณ์มาสเตอร์ซึ่งจะนำไปเป็นขาสัญญาณอินพุตของอุปกรณ์สลาฟ

คุณสมบัติของโมดูล SPIM (Serial Peripheral Interconnect Master)

- สนับสนุนการเชื่อมต่อในโหมดมาสเตอร์ ด้วยโปรโตคอลการสื่อสารแบบ SPIM (Serial Peripheral Interconnect Master)
- สนับสนุนโหมดสัญญาณนาฬิกาของ SPI คือ โหมด 0,1,3 และ 3
- สามารถกำหนดแหล่งสัญญาณนาฬิกาได้
- สามารถเลือกการเชื่อมต่อเอาต์พุตของสัญญาณ MOSI และ SCLK ได้
- สามารถสร้างสัญญาณอินเตอร์รัพได้
- ความเร็วในการรับส่งข้อมูลสูงสุด (Maximum bit rate) 12 MHz

การใช้งานฟังก์ชัน API ของโมดูล SPIM

การใช้งานโมดูลจะกระทำผ่านการเรียกใช้ฟังก์ชัน SPI ซึ่งการใช้งานฟังก์ชันนี้อาจมีผลทำให้ค่าของรีจิสเตอร์ A และ X มีการเปลี่ยนแปลงไปด้วยโดยจะมีฟังก์ชันต่าง ๆ ให้ใช้งานดังต่อไปนี้

SPIM_Start

ฟังก์ชันการกำหนดโหมดการทำงานของ SPI พร้อมทั้งเปิดการทำงานของโมดูล โดยการกำหนดโหมดการทำงานจะต้องผ่านค่าให้กับฟังก์ชันผ่านทางรีจิสเตอร์ A ซึ่งจะมีโหมดต่างๆ ตามตารางต่อไปนี้

Symbolic Name	Value	ความหมาย
SPIM_MODE_0	0x00	การทำงานของ SPI ในโหมด 0
SPIM_MODE_0	0x02	การทำงานของ SPI ในโหมด 1
SPIM_MODE_0	0x04	การทำงานของ SPI ในโหมด 2
SPIM_MODE_0	0x06	การทำงานของ SPI ในโหมด 3
SPIM_LSB_FIRST	0x80	ส่งบิตข้อมูลทางด้านบิตต่ำก่อน
SPIM_MSB_FIRST	0x00	ส่งบิตข้อมูลทางด้านบิตสูงก่อน

ตารางที่ 2.5 แสดงค่าของโหมดการทำงานต่าง ๆ ของ SPI

ตัวอย่าง

Assembly :

Mov A, SPIM_MODE_2|SPIM_LSB_FIRST ; เลือกโหมด 2 ส่งข้อมูลทางด้านบิตต่ำก่อน

C Prototype :

Void SPIM_Start (BYTE bConfiguration)

- หมายเหตุ เราสามารถกำหนดโหมดการทำงานร่วมกับการเลือกรูปแบบการส่ง (SPIM_LSB_FIRST) หรือ (SPIM_MSB_FIRST) โดยใช้การ OR โดยใช้เครื่องหมาย "|" คั่นกลางดังตัวอย่าง

SPIM_STOP

ฟังก์ชันการปิดการทำงานของโมดูล SPIM ฟังก์ชันนี้จะไปเคลียร์ค่าในบิต Enable ของรีจิสเตอร์ Control ทำให้โมดูล SPIM หยุดการทำงาน

ตัวอย่าง

Assembly :

Call SPIM_Stop

C Prototype

Void SPIM_Stop (void)

SPIM_EnableInt

ฟังก์ชันเปิดการอินเตอร์รัพของ SPIM ตามเงื่อนไขของการอินเตอร์รัพท์

ตัวอย่าง

Assembly :

Call SPIM_EnableInt

C Prototype :

Void SPIM_EnableInt (void)

SPIM_DisableInt

ปิดการอินเตอร์รัพท์ของ SPIM

ตัวอย่าง

Assembly :

```
Call SPIM_DisableInt
```

C Prototype :

```
Void SPIM_DisableInt(void)
```

SPIM_SendTxData

ส่งข้อมูลแบบ SPI ไปยังอุปกรณ์สลาฟที่ต่อร่วมด้วย

ตัวอย่าง

Assembly :

```
Mov A, bSPIMData
Call SPIM_SendTxData
```

C Prototype :

```
BOOL SPIM_SendTxData (BYTE bSPIMData)
```

SPIM_bReadRxData

ฟังก์ชันการอ่านรับค่าจากไบต์ข้อมูลจากอุปกรณ์สลาฟ โดยควรทำการเช็คสถานะของบัฟเฟอร์ (Rx Buffer) ก่อนว่าว่างหรือไม่ ก่อนที่จะทำการเรียกใช้ฟังก์ชันนี้

ตัวอย่าง

Assembly :

```
Call SPIM_bReadRxData
Mov bRxData, A
```

C Prototype :

```
BOOL SPIM_ bReadRxData(void)
```

SPIM_bReadStatus

เป็นคำสั่งอ่านค่าสถานะจากรีจิสเตอร์ควบคุม (ControlRegister) ของโมดูล SPIM รีจิสเตอร์จะประกอบด้วยสถานการณ์ทำงานต่างๆ ของโมดูล โดยค่าสถานะต่างๆ จะรีเทิร์นค่าออกมาผ่านทางรีจิสเตอร์ A ซึ่งค่าต่าง ๆ จะเป็นดังตารางต่อไปนี้

โหมดการทำงานของ SPI

Symbolic Name	Value	ความหมาย
SPIM_DONE	0x20	การทำงานของ SPIM เสร็จสิ้น
SPIM_RX_OVERRUN	0x40	เกิดการ Overrun Error ของการรับข้อมูล
SPIM_TX_BUFFER_EMPTY	0x10	บัฟเฟอร์ของการส่องสว่าง
SPIM_RX_BUFFER_FULL	0x08	บัฟเฟอร์การรับข้อมูลเต็ม

ตารางที่ 2.6 แสดงค่าสถานะรีจิสเตอร์ของการเก็บข้อมูลของฟังก์ชัน SPI

สามารถใช้การ OR โดยใช้เครื่องหมาย '|' เพื่อเพิ่มเงื่อนไขการตรวจสอบได้ ดังเช่นตัวอย่างต่อไปนี้

ตัวอย่าง

Assembly :

```
Call SPIM_bReadStatus
And A, SPIM_DONE|SPIM_RX_BUFFER_FULL
Jnz SpimCompleteGetRxData
```

C Prototype :

```
BYTE SPIM_bReadStatus(void)
```

- หมายเหตุ บิตสถานะต่างๆ ในรีจิสเตอร์ควบคุม (Control register) จะถูกเคลียร์ หลังเสร็จสิ้นฟังก์ชันนี้

2.1.6 การใช้งานโมดูล SDCard

Resources Only parts in the CY7C29xxx family are currently supported with this user module.	Build Configurations									
	Full File System		Full File System		Full File System		Full File System		Full File System	
	Approximate PSoC Memory Use (bytes)									
	Flash	Flash	Flash	Flash	Flash	Flash	Flash	Flash	Flash	Flash
	19633	19633	19633	19633	19633	19633	19633	19633	19633	19633
SDCard Functions Included in the Configuration										
SDCard_clearerr	X		X		X		X		X	
SDCard_fclose	X		X		X		X			
SDCard_feof	X		X		X		X			
SDCard_ferror	X		X		X		X			
SDCard_fflush	X		X		X				X	
SDCard_fgetc	X		X		X		X		X	
SDCard_fbgetc	X		X		X		X		X	
SDCard_fopen	X		X		X		X			
SDCard_fputBuff	X		X		X				X	
SDCard_fputcBuff	X		X		X				X	
SDCard_fputc	X		X		X				X	
SDCard_fputcs	X		X		X				X	
SDCard_fputs	X		X		X				X	
SDCard_fseek	X		X		X		X		X	
SDCard_ftell	X		X		X		X		X	
SDCard_Copy	X		X							
SDCard_GetFileCount	X		X		X		X			
SDCard_GetFilename	X		X		X		X			
SDCard_GetFileSize	X		X		X		X			
SDCard_InitCard	X		X		X		X		X	
SDCard_Present	X		X		X		X		X	
SDCard_Remove	X		X		X					
SDCard_Rename	X		X							

Resources Only parts in the CY7C29xxx family are currently supported with this user module.	Build Configurations									
	Full File System		Standard File System		Basic File System		Read Only File System		Basic Read Write	
	Approximate PSoC Memory Use (bytes)									
	Flash	**RAM	Flash	*RAM	Flash	*RAM	Flash	*RAM	Flash	RAM
	19633	629	17050	613	16437	613	11751	613	4245	575
SDCard_Select	X		X		X		X		X	
SDcard_Start	X		X		X		X		X	
SDard_Stop	X		X		X		X		X	
SDCard_WriteProtect	X		X		X				X	
Comments	Supports FAT16/32		Supports FAT16		Supports FAT16		Supports FAT16		Not PC Compatible	

**เพิ่ม RAM 24 bytes สำหรับการเปิดไฟล์แต่ละไฟล์ในเวลาเดียวกัน (สำหรับระบบไฟล์ FAT32)

*เพิ่ม RAM 20 bytes สำหรับการเปิดไฟล์แต่ละไฟล์ในเวลาเดียวกัน (สำหรับระบบไฟล์ FAT16)

ตารางที่ 2.7 ตารางแสดงค่าฟังก์ชันต่างๆของ SD card module

จุดเด่นและลักษณะโดยรวม

- รองรับ SD, mini SD, Micro SD/TransFlash, MMC, RS-MMC/MMCmobile and MMCplus
- ปฏิบัติการบนระบบ PC FAT 16/32, DOS และ Windows โดยใช้ชื่อไฟล์สั้น (DOS 8.3 format)
- สามารถอ่านหรือเขียนไฟล์ได้ทีละหลายๆไฟล์
- รองรับการเลือกไฟล์ทีละหลายๆไฟล์โดยแบบสุ่ม
- สามารถใช้ระบบปฏิบัติการ PSoC ในระบบการจัดเก็บข้อมูลได้สูงสุดถึง 2 Gb

SD Card User Module สามารถใช้กับ flash card ถึง 6 ชนิดโดยไม่เกิดปัจจัย "nuts and bolt" ของการเข้าถึงข้อมูลหรือ flash card interface

SD Card User Module สามารถใช้งานกับ PSoC Pins อย่างน้อย 4 ตัวในระบบปฏิบัติการพื้นฐาน ทั้งนี้ขึ้นอยู่กับชนิดของการ์ดและเต้าเสียบการ์ดซึ่งคุณสามารถใช้พินเพิ่มเติมเพื่อรองรับการ write protect, card insert และอื่นๆ

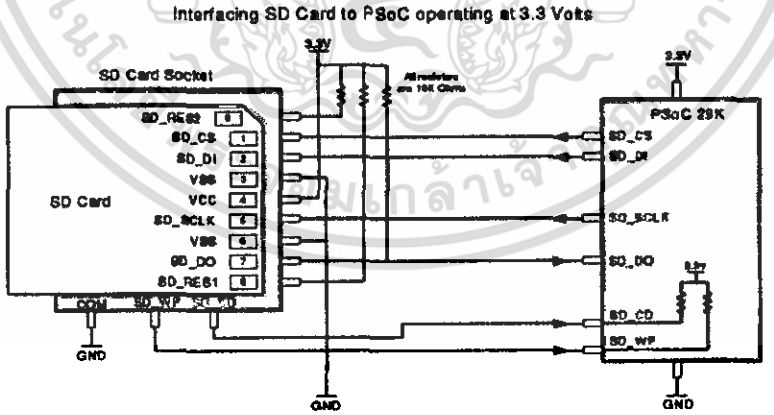
โมดูลตัวนี้สามารถเข้าถึง SD card และ MMC card โดยใช้ภาษา C ในการติดต่อได้โดยไม่จำเป็นต้องทราบว่าตัวไหนเป็น SPI bus หรือ SD/MMC command ซึ่งสามารถใช้ได้ตลอดเมื่อใช้ฟังก์ชัน SD card module

โดยสามารถใช้โมดูลตัวนี้กับ SD หรือ MMC ได้โดยมีเงื่อนไขดังนี้

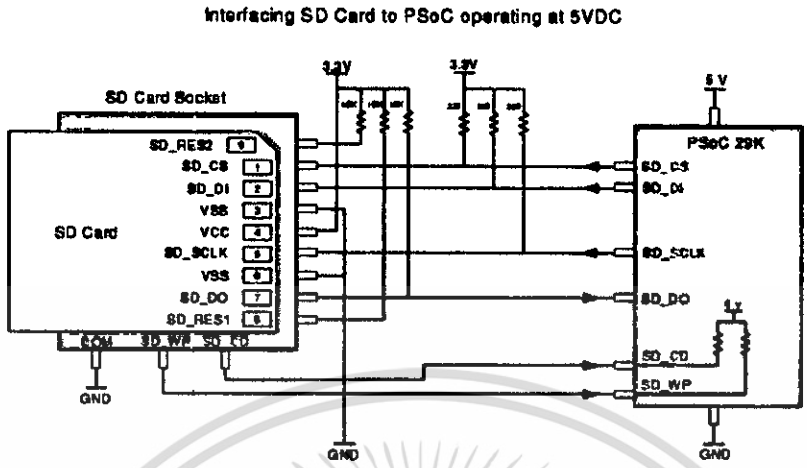
- เกิดภาวะไฟตกโดยอยู่ในระดับที่โมดูลตัวนี้ยอมรับได้
- ใช้ SPI data mode ในการเข้าถึงการ์ด
- สามารถดู Spec ได้ที่เวบไซต์ sdcard.org หรือ mmca.org
- โมดูลตัวนี้จะสามารถใช้ได้ดียิ่งขึ้นเมื่อมีลักษณะเพิ่มเติมดังนี้
  - การ์ดฟอร์แมตด้วย Windows/DOS ให้เป็น FAT16 หรือ FAT32 ได้
  - การ์ดฟอร์แมตนี้เปรียบเสมือน Hard disk ส่วนแรกใน partition table
  - สามารถอ่านไฟล์ได้ในสารบบรากเท่านั้นไม่สามารถอ่านสารบบย่อยได้

ข้อแนะนำ โดยทั่วไปการ์ดที่มีความจุน้อยกว่า 32 MB จะฟอร์แมตเป็น FAT12 ส่วนการ์ดที่มีความจุมากกว่า 32 MB ขึ้นไป (สูงสุดที่ขนาด FAT16 ความจุ 16 GB) จะฟอร์แมตเป็น FAT16 อย่างไรก็ตามก็ยังมีข้อยกเว้นสำหรับวินโดว์หรือการ์ดชนิดอื่นๆ อาจฟอร์แมตวิธีอื่นได้ อาทิเช่น การ์ดความจุที่มีขนาดใหญ่อาจฟอร์แมตเป็น FAT32 แต่ส่วนใหญ่การ์ดจะฟอร์แมตเป็น FAT16

ข้อแนะนำ SDCard User Module สามารถเขียนได้ทั้งสองภาษา คือ Assembly และ C ซึ่งคุณจะต้องมี license compiler C ในโปรแกรม PSoC designer จึงจะสามารถขียนภาษา C ได้ ถ้าหากว่าโปรแกรมที่คุณเขียนมีความซับซ้อนให้ใช้ภาษา Assembly เขียน



รูปที่ 2.3 แสดงการอินเทอร์เฟสระหว่าง SD Card ไปยัง PSoC ที่ 3.3 โวลต์

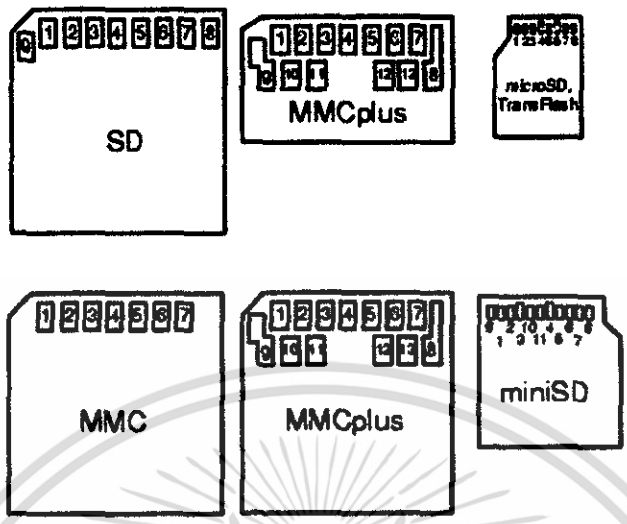


รูปที่ 2.4 แสดงการอินเทอร์เฟสระหว่าง SD Card ไปยัง PSoC ที่ 5 VDC

PSoC Signals to Flash Card Pins								
SD Module Name	**Description**	**SD**	**mini SD**	**microSD, TransFlash**	**MMC**	**RS_MMS, MMC plus, MMCmobile**	**PSoC Drive Type**	
							3.3V	**5V**
SD_CS	Card Select	1	1	2	1	1	Strong	*Open Drain low
SD_MOSI	SD_IN PSoC-to-Card Commands and Data	2	2	3	2	2	Strong	*Open Drain low
SD_MISO	SD_OUT	7	7	7	7	7	High Z	High Z
SD_CLK	SD_CLK	5	5	5	5	5	Strong	*Open Drain low
VDD	3.3V	4	4	4	4	4		
VSS	Ground	3,6	3,6	6	3,6	3,6		
Reserved		8.9	8.9	1,8	-	-		
NC	No Connect	-	10,11	-	-	-		
SD_CI	Optional						Pull Up	Pull Up
SD_WP	Optional (SD cards only)		-	-	-	-	Pull Up	Pull Up
Supply Voltage	VDC	2.7-3.6V	2.7-3.6V	2.7-3.6V	2.7-3.6V	2.7-3.6V or 1.65-1.95V		

*เมื่อ PSoC ทำงานที่ 5 โวลต์จะต้องต่อกับ pull up 330 โอห์มและ power supply 3.3 โวลต์

ตารางที่ 2.8 แสดงสัญญาณ PSoC ที่ส่งไปยังพินของแฟลชการ์ด



รูปที่ 2.5 แสดงแผนผังการ์ด (มองจากด้านล่าง)

Functional Description

SDCard (Secure Digital Memory Card) User Module เป็นโมดูลที่ใช้ในการติดต่อกับ SD/MMC Module นี้จะใช้หนึ่ง Digital Communication Block (DCB) ในโหมด SPI ในการติดต่อสื่อสารกับ SD card และสามารถใช้หนึ่งพอร์ตหรือมากกว่าหนึ่งพอร์ตในการประกาศใช้ chip select, card detection และ write protect

ความสัมพันธ์ของสัญญาณระหว่าง PSoC และ SD memory card มี 4 สัญญาณที่มีฟังก์ชันสัมพันธ์กับ SPI มาตรฐานและสัญญาณ SD card อีก 2 สัญญาณเพิ่มเติม ดังตาราง

ชื่อสัญญาณ	รายละเอียด	SPI Equivalent
SD_CS	Card Select (Active Low)	SS
SD_DI	Data Input	MOSI
SD_DO	Data Output	MISO
SD_SCLK	Interface clock	SCLK
SD Card specific signals (optional)		
SD_CD	Card Detect (Active Low)	NA
SD_WP	Card Write Protection (Active Low)	NA

ตารางที่ 2.9 แสดงความสัมพันธ์ของสัญญาณระหว่างสัญญาณ SD card และ SPI มาตรฐาน

SD_SCLK เป็นสัญญาณ SPI ที่ใช้ในการรับส่งสัญญาณนาฬิกา โดยอัตราส่วนครึ่งหนึ่งเป็นสัญญาณขาเข้าของสัญญาณนาฬิกา ผลของการรับส่งสัญญาณ บิตขาเข้าจะถูกหารด้วยสอง ซึ่งจะอยู่ใน Device Editor Window โดยเบื้องต้น SD_SCLK จะถูกตั้งค่าให้ใช้งานที่ 32 kHz หลังจากที่ผู้ใช้ได้เลือกสัญญาณนาฬิกาและ MMC/SD Card จะสามารถใช้งานกับสัญญาณนาฬิกาได้ไม่เกิน 400 kHz ซึ่งนั่นเป็นข้อจำกัดของ Module และ PSoC เอง ดังนั้นอัตราการถ่ายโอนสูงสุด 20 MHz สำหรับ MMC หรือ 25 MHz สำหรับ SD นั้นเป็นไปไม่ได้

สัญญาณ SD_DI ใช้ในการถ่ายโอนข้อมูลจาก PSoC ไปยัง MMC/SD Card ซึ่งมีค่าเท่ากับสัญญาณ SPI Master Out Slave In (MOSI) ส่วน SD_DO เป็นสัญญาณที่ใช้ในการถ่ายโอนข้อมูลจาก MMC/SD Card ไปยัง PSoC ซึ่งมีค่าเท่ากับ สัญญาณ SPI Master In Slave Out (MOSI) สัญญาณนาฬิกา SD_SCLK เป็นสัญญาณที่มีการรับส่งข้อมูล 2 ทิศทาง และขับเคลื่อนการทำงานด้วยโปรแกรม PSoC โดยมีค่าเท่ากับ SPI Serial Clock (SCLK) (สัญญาณลำดับที่ 4) SD_CS เป็นสัญญาณที่ทำงานก็ต่อเมื่อ MMC/SD Card เกิดการเชื่อมต่อซึ่งสัญญาณนี้จะมีค่าสัญญาณต่ำก่อนที่จะทำการส่งสัญญาณคำสั่ง และ/หรือ ข้อมูลและเมื่อการส่งสัญญาณ (การถ่ายโอนข้อมูล) เสร็จสิ้นสมบูรณ์แล้วค่าสัญญาณจะกลับไปอยู่ในระดับสูง SD_CD (Card Detect) และ SD_WP (Write Protect) เป็นสัญญาณตัวเลือกที่มักใช้เมื่อการ์ดปรากฏขึ้นและการ์ดนั้นเป็น Write Protect

<u>ข้อแนะนำ</u> ควรให้ฟังก์ชัน API ทั้งหมดทำงานอย่างสมบูรณ์ก่อนที่จะปิดโมดูลในระบบซึ่งจะเป็นการรับประกันได้ว่าข้อมูลทั้งหมดจะไม่สูญหายหรือ MMC/SD card จะไม่เกิดการเสียหายได้ และไม่ควรปิดไฟล์ใดๆ ก่อนที่จะทำการปิดโมดูลหรือเอาการ์ดออกเพื่อให้แน่ใจว่าการถ่ายโอนข้อมูลนั้นได้เสร็จสมบูรณ์

คุณสมบัติทางไฟฟ้าของ DC และ AC

Parameter	เงื่อนไขและบันทึก	Typical	ขีดจำกัด	หน่วย
SD_SCLK	จำนวนสูงสุดของอัตรา bit	-	4	MHz
ความเร็วในการเขียน	12 MHz CPU Clock	2250		Bytes/Second
ความเร็วในการอ่าน	12 MHz CPU Clock	2800		Bytes/Second

ตารางที่ 2.10 แสดงคุณสมบัติทางไฟฟ้าของ AC และ DC

ควรตั้งค่า VCC Supply ทุกครั้งเมื่อมีการใช้งาน MMC/SD card โดยทั่วไปแล้วจะอยู่ที่ 2.7 – 3.3 VDC ซึ่งทั้งการ์ดและ PSoC microcontroller จะปฏิบัติการได้ที่ 3.3V VCC แต่คุณก็สามารถ run PSoC ที่ 5V ได้หากต้องการ แต่โปรแกรมต้องการกำลังไฟ 5V – 3.3V ในการ interface กับ MMC/SD card

ตำแหน่ง

SDCard User Module จะติดตั้งอยู่บน single PSoC block และอาจอยู่ที่ตำแหน่ง Digital Communication blocks สำรอง port pin เพื่อสัญญาณ SD_WP, SD_CD และ SD_CS

ตัวแปรและแหล่งที่มา

Build_Configuration

เป็นตัวที่เปลี่ยนขนาดของ Flash และ RAM โดย SDCard User Module

Build_Configuration Value	**ข้อบ่งใช้**
Full FileSystem	ใช้งานกับ Flash และ RAM ที่มีขนาดใหญ่ที่สุด ส่วนใหญ่แล้วจะใช้งานได้ดีทั้งกับ FAT16 และ FAT32 โดยส่งผลให้ Flash card ทำงานอย่างมีประสิทธิภาพในคำสั่งระดับสูงเช่น การ Copy การเปลี่ยนชื่อและการลบ
Standard FileSystem	การใช้งานเหมือนกับ Full File System ต่างตรงที่จะไม่มีระบบรองรับ FAT32 จึงเหมาะกับการ์ดที่ฟอร์แมตเป็น FAT16 อีกทั้งยังลดพื้นที่ของ Code ลงกว่า 2K
Basic FileSystem	เป็นตัวที่ลดพื้นที่หน่วยความจำลงโดยการเอา FAT32 ออก (เอา FileRename และ FileCopy ออก) คงเหลือไว้แต่ FileRemove เนื่องจากจะทำให้ลดพื้นที่ลงได้ประมาณ 55 bytes นั่นก็หมายความว่าแอพลิเคชันต่างๆจะสามารถสร้างและลบได้เมื่อต้องการ
ReadOnly FileSystem	จะคงรักษาไว้เฉพาะระบบ FAT16 และยังคงสามารถอ่านไฟล์ระบบของ PC ได้ อย่างไรก็ตามระบบนี้จะสามารถอ่านไฟล์ในการ์ดได้เพียงอย่างเดียวเท่านั้น ดังนั้นระบบจึงทำงานได้อย่างดีในการอ่านไฟล์โดยเปรียบเสมือนกับ bootloader เลยทีเดียว

Build_Configuration Value	ข้อบ่งใช้
Basic ReadWrite NoFileSystem	เป็นระบบที่เหมาะกับ PC ที่ไม่ต้องการ On Board flash memory เหมือนกับ iNAND Module โดยการนำระบบ API ออกจะสามารถลดปริมาณของ Flash และ RAM ลงโดย SDCard User Module ไม่น้อยกว่า 5K bytes
Custom Configuration	ตัวเลือกนี้เหมาะสำหรับผู้ที่ใช้งานในระดับสูง ในการสร้าง Custom Configuration สำหรับ SDCard User Module เพื่อให้เหมาะกับความต้องการของผู้ใช้งาน โดยทำการปรับแต่งที่ SDCard_Config.h

ตารางที่ 2.11 แสดงค่าฟังก์ชั่นต่างๆของ SD card user module

Maximum_Open_File

การตั้งค่าจำนวนไฟล์สูงสุดนั้นจะทำให้แอพลิเคชันต่างๆสามารถเปิดได้พร้อมๆกัน หากมีพื้นที่ RAM มากเพียงพอที่จะเปิดไฟล์เหล่านั้น เมื่อตั้งค่า Build_Configuration เป็น Full File System โดยแต่ละไฟล์ที่ถูกเปิดจะใช้พื้นที่ RAM 24 bytes ส่วน Build_Configuration แบบอื่นๆอีก 4 แบบ จะใช้ 20 bytes ต่อไฟล์ และเมื่อตั้งค่า Build_Configuration เป็น Custom โดยใช้ SDCard_Config.h ไฟล์สูงสุดที่เปิดได้เท่ากับปริมาณพื้นที่ของ RAM ที่ใช้ในแต่ละไฟล์ที่ถูกตั้งค่าเป็น SDCard_Config.h

Clock

SDCard User Module สามารถเป็นนาฬิกาได้จากแหล่งกำเนิดเพียง 1 แหล่ง โดยใช้ global I/O buses ในการเชื่อมต่อสัญญาณนาฬิกาขาเข้าสู่พินภายนอกหรือฟังก์ชันนาฬิกาซึ่งมาจาก block PSoC ที่ต่างกัน คือ VC1, VC2, VC3 หรือตัวหนึ่งตัวใดของแหล่งนาฬิกาอื่นๆ ซึ่งค่าที่เหมาะสมที่สุดคือการตั้งค่านาฬิกาไปยังแหล่งกำเนิดนาฬิกาที่ 4 MHz ตั้งค่าอัตรานาฬิกา 2 ครั้งเพื่อให้ได้อัตราบิตที่ต้องการ โดย 1 ข้อมูลบิตจะส่งหรือรับข้อมูลทุกๆ 2 input clocks

SD_DO

สัญญาณ SD_DO (Data Output) ควรจะเป็นเส้นทางเดียวกับเส้นทางขาเข้าสู่ PSoC ซึ่งมี 1 สัญญาณที่จะเข้าสู่พินโดยตั้งค่า drive mode สำหรับ pin ที่ "Hi Z"

SD_DI

สัญญาณ SD_DI (Data Input) ควรจะเป็นเส้นทางเดียวกับเส้นทางขาออกสู่ PSoC จะมีสัญญาณขาออกของพินเพียง 1 สัญญาณที่ถูกเลือกโดยตั้งค่าที่ drive mode ในระบบเมื่อทั้ง SD Card Power Supply และ PSoC Supply อยู่ที่ 3.3 volts ให้ตั้งค่า drive mode เป็น "Strong" ถ้า PSoC อยู่ที่ 5 volts และ SD Card อยู่ที่ 3.3 volts ให้ตั้งค่า drive mode เป็น "Open Drain Low" และต่อกับวงจร pull up ภายนอก 330 โอห์ม โดยต่ออยู่ระหว่างสัญญาณ SD_DI และ power supply 3.3 volt

SD_SCLK

SD_SCLK เป็นสัญญาณนาฬิกาที่ใช้สำหรับสัญญาณ SD_DI และ SD_DO พินตัวนี้เป็น output จาก PSoC ไปยัง SDCard ซึ่ง Output pin เพียงตัวเดียวที่ถูกเลือกและสัญญาณนั้นจะถูกส่งเข้าไปโดยตั้งค่าที่ drive mode ในระบบเมื่อทั้ง SD Card Power Supply และ PSoC Supply อยู่ที่ 3.3 volts ให้ตั้งค่า drive mode เป็น "Strong" ถ้า PSoC อยู่ที่ 5 volts และ SD Card อยู่ที่ 3.3 volts ให้ตั้งค่า drive mode เป็น "Open Drain Low" และต่อกับวงจร pull up ภายนอก 330 โอห์ม โดยต่ออยู่ระหว่างสัญญาณ SD_DI และ power supply 3.3 volt

SD_CS_Port

เลือกพอร์ตนี้สำหรับสัญญาณ SD_CS

SD_CS_Pin

สามารถเลือกพินจากพอร์ตที่ต้องการโดยใช้พารามิเตอร์ SD_CS_Port SD_CS(Card Select) ใช้เพื่อให้ SD card สามารถปฏิบัติการได้ โดยใช้สัญญาณนี้เป็นสัญญาณขาเข้าจาก PSoC ไปยัง SD card ซึ่ง drive mode จะตั้งค่าอัตโนมัติเป็น internal pull up

SD_CD_Port

เลือกพอร์ตนี้สำหรับสัญญาณ SD_CD

SD_CD_Pin

สามารถเลือกพินจากพอร์ตที่ต้องการโดย SD_CD_Port สำหรับสัญญาณ SD_CD ซึ่งสัญญาณ SD_CD (Card detect) นี้จะเกิดขึ้นเมื่อเสียบ SD Card

SD_WP_Port

เลือกพอร์ตนี้สำหรับสัญญาณ SD_WP

SD_WP_Pin

สามารถเลือกพินจากพอร์ตที่ต้องการโดยใช้พารามิเตอร์ SD_WP_Port สำหรับสัญญาณ SD_WP ซึ่ง SD_WP (Write Protect) พินจะทำงานบน SD Card โดยผ่านช่องเสียบ SD Card

Invert SD_DO

พารามิเตอร์นี้จะอยู่ใน "Normal setting"

Application Programming Interface (API)

API library function เป็นหัวใจของ SD/MMC card interface ซึ่ง API นี้เขียนโดยใช้พื้นที่ RAM และ Flash น้อยที่สุดเท่าที่จะทำได้ ซึ่งจะคล้ายกับ Standard C function สำหรับการเข้าถึงไฟล์หรือหน่วยความจำ ในขณะที่มันไม่สามารถเป็นไปได้ที่จะทำให้ standard file IO function สัมฤทธิ์ผลทั้งหมด สำหรับ SD/MMC card interface ใน subset พื้นฐานจะประกอบด้วยฟังก์ชันเพิ่มเติมที่จะช่วยในการจัดการไฟล์

<u>หมายเหตุ</u>

แม้ว่า library function จะมีความคล้ายคลึงกับ Standard C function ก็ตามแต่ก็ไม่ได้หมายความว่าจะเหมือนกันทุกประการ ซึ่งคำสั่งนั้นจะใช้ได้หรือไม่นั้นขึ้นกับพารามิเตอร์ของ Build Configuration

Basic Read/Write Command

ฟังก์ชัน	รายละเอียด
void **SDCard_Start**(void);	เริ่มการทำงานของ SDCard Module
void **SDCard_Stop**(void);	หยุดการทำงานของ SDCard Module
void **SDCard_Select**(uchar Enable);	เลือกหรือไม่เลือก SD card
uchar **SDCard_InitCard**(void);	Run ทุกคำสั่งเพื่อให้การ์ดทำการเชื่อมต่อ
uchar **SDCard_fseek**(uchar Fptr, ulong offset);	ค้นหา Offset เฉพาะที่อยู่ในไฟล์

ฟังก์ชัน	รายละเอียด
uchar **SDCard_fgetc**(uchar Fptr);	กลับสู่ character ตัวถัดไปจากไฟล์ที่กำหนด
uchar **SDCard_fbgetc**(uchar Fptr);	กลับสู่ buffered character ตัวถัดไปจากไฟล์ที่กำหนด ซึ่งทำให้เวลาที่อ่านรวดเร็วกว่า fgetc เมื่ออ่านเพียง 1 ไฟล์ใน 1 ครั้ง
void **SDCard_clearerr**(uchar Fptr);	เคลียร์ error flags สำหรับไฟล์
uchar **SDCard_ferror**(uchar Fptr);	กลับสู่ศูนย์ถ้าไม่มีไฟล์ผิดพลาดบนไฟล์ที่กำหนดไว้
ulong **SDCard_ftell**(uchar Fptr);	กลับสู่ file offset ของ character ตัวถัดไปเพื่ออ่านหรือเขียน
uchar **SDCard_ReadSect**(ulong address);	สำหรับอ่าน sector
uchar **SDCard_Present**(void);	กลับสู่ '1' ถ้าปรากฏการ์ดใน drive ถ้าไม่ปรากฏจะมีค่าเป็น '0'
uchar **SDCard_WriteProtect**(void);	กลับสู่ '1' ถ้าการ์ดเป็น write protect ถ้าไม่ปรากฏจะมีค่าเป็น '0'
uchar **SDCard_WriteSect**(ulong address);	สำหรับเขียน sector
uchar **SDCard_fputc**(uchar Data, uchar Fptr);	สำหรับเขียน character ไปยังไฟล์
uchar **SDCard_fputs**(char*str,uchar Fptr);	เขียน null terminate string ไปยังไฟล์
uchar **SDCard_ fputcs** (const char*str, uchar Fptr);	เขียน null terminate constant string ไปยังไฟล์
uchar **SDCard_fputBuff**(uchar*buff, uint count, uchar Fptr);	เขียน Count character จาก RAM buffer ไปยังไฟล์
uchar **SDCard_ fputcBuff** (const uchar*buff, uint count, uchar Fptr);	เขียน Count character จาก ROM buffer ไปยังไฟล์
void **SDCard_fflush**(uchar Fptr);	Flush write buffer ไปยังไฟล์

ตารางที่ 2.12 แสดงคำสั่งพื้นฐานที่ใช้ในการอ่าน/เขียนไฟล์

Top Level File Functions

ฟังก์ชัน	รายละเอียด
uchar **SDCard_fclose**(uchar Fptr);	ปิดไฟล์ที่ต้องการแล้วปล่อยสู่ pointer
uchar **SDCard_fopen**(uchar *Filename, const uchar *Mode);	เปิด supplied file name โดยใช้โหมดที่ระบุและกลับสู่ไฟล์ pointer
uchar * **SDCard_GetFilename**(uint Entry);	กลับสู่ชื่อไฟล์สำหรับการบันทึกสารบบที่กำหนด
uint **SDCard_GetFileCount**(void);	กลับสู่จำนวนของไฟล์ในสารบบราก
ulong **SDCard_GetFileSize**(uchar Fptr);	กลับสู่ขนาดของไฟล์ที่กำหนด
uchar **SDCard_feof**(uchar Fptr);	กลับสู่ non-zero ถ้าไฟล์ที่ระบุไว้เป็น EOF หรือ '0'

ตารางที่ 2.13 แสดงค่าฟังก์ชันของไฟล์ที่นิยมใช้

Top Level Writing Functions

ฟังก์ชัน	รายละเอียด
uchar **SDCard_Remove**(uchar *Filename);	ลบชื่อไฟล์
uchar **SDCard_Rename**(uchar *OldFilename, uchar *NewFilename);	เปลี่ยนชื่อไฟล์
uchar **SDCard_Copy**(uchar *OldFilename, uchar *NewFilename);	Copy ชื่อไฟล์

ตารางที่ 2.14 แสดงค่าฟังก์ชันของการเขียนไฟล์ที่นิยมใช้

SDCard_clearerr

รายละเอียด:

Clears the error flags สำหรับไฟล์ที่กำหนดโดย file pointer

C Prototype:

void SDCard_clearerr(uchar Fptr)

Parameters:

Fptr: เป็นพารามิเตอร์ที่ไฟล์ pointer ใช้สำหรับการเข้าถึงไฟล์

ค่าที่ได้:

None.

ใช้ร่วมกับ:

SD_feof, SDCard_ferror

SDCard_fclose

รายละเอียด:

ปิดไฟล์ที่ต้องการโดยไฟล์ pointer ซึ่งจะทำให้ไฟล์ pointer ถูกปล่อยออกมาเพื่อสามารถนำไฟล์กลับมาใช้ซ้ำและ clear the file control variable

C Prototype:

uchar SDCard_fclose(uchar Fptr)

Parameters:

Fptr: เป็นพารามิเตอร์ที่ไฟล์ pointer ใช้สำหรับการเข้าถึงไฟล์

ค่าที่ได้:

The file error flagsที่กำหนดไว้ใน SDCard_ferror.

ใช้ร่วมกับ:

SDCard_fopen, SDCard_ferror

SDCard_feof

รายละเอียด:

กลับสู่ EOF (End Of File) file error flag สำหรับไฟล์ที่กำหนดไว้โดยไฟล์ pointer

C Prototype:

uchar SDCard_feof(uchar Fptr)

Parameters:

Fptr: เป็นพารามิเตอร์ที่ไฟล์ pointer ใช้สำหรับการเข้าถึงไฟล์

ค่าที่ได้:

The EOF (End Of File) file error flag สำหรับไฟล์ที่กำหนดไว้โดยไฟล์ pointer

ใช้ร่วมกับ:

SDCard_clearerr, SDCard_ferror

SDCard_ferror

รายละเอียด:

กลับสู่ file error flags สำหรับไฟล์ที่กำหนดไว้โดยไฟล์ pointer.

C Prototype:

uchar SDCard_ferror(uchar Fptr)

Parameters:

Fptr: เป็นพารามิเตอร์ที่ไฟล์ pointer ใช้สำหรับการเข้าถึงไฟล์

ค่าที่ได้:

file error flags จะใช้พื้นที่ 1byte สำหรับไฟล์ที่ระบุโดยไฟล์ pointer ซึ่งไฟล์ที่เปิดแต่ละไฟล์จะใช้ไฟล์ error status byte กับ bit ที่กำหนด

SDCard_ferror Bits

Bit 7	**Bit**	**Bit 5**	**Bit**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
EOF	**6WE**	**FPE**	**FFE**	**CE**	**PRE**	**FNF**	**IFN**
End of file	Write error	File pointer error	File format error	Card error	Parameter range error	File not found	Invalid file name

ตารางที่ 2.15 แสดงค่า SDCard_ferror Bits

ใช้ร่วมกับ:

SDCard_clearerr, SDCard_feof

SDCard_fgetc

รายละเอียด:

จะได้รับไฟล์ character ที่กำหนดไว้โดยไฟล์ offset และกลับสู่สถานะ unsigned character ซึ่งจะเพิ่มไฟล์ offset และปรับไฟล์ control variables เพื่อให้เหมาะสมกันโดยจะต้องเปิดไฟล์และมีการจัดการไฟล์ที่ยอมรับได้

หมายเหตุ ใน Basic read/write mode, Fptr เป็น index ที่ออกแบบโดยตัวผู้ใช้เองซึ่งน้อยกว่า MAXFILES ที่กำหนดโดยผู้ใช้ที่บ่งชี้การใช้งานของ offset เอง

C Prototype:

uchar SDCard_fgetc(uchar Fptr)

Parameters:

Fptr: เป็นพารามิเตอร์ที่ไฟล์ pointer ใช้สำหรับการเข้าถึงไฟล์

ค่าที่ได้:

The current character จะอ่านจากไฟล์

ใช้ร่วมกับ:

SDCard_fopen, SDCard_fclose, SDCard_fputc, SDCard_fseek, SDCard_ftell, SDCard_GetFileSize

SDCard_fopen

รายละเอียด:

ค้นหารายชื่อไฟล์และเปิดไฟล์โดยใช้โหมดที่กำหนด(จะมีไฟล์ที่ใช้ได้อยู่หนึ่งไฟล์) ไฟล์ control variables จะถูกตั้งค่าและกลับสู่ฟังก์ชันแรกโดยไม่ใช้ไฟล์พอยน์เตอร์เป็นไฟล์อ้างอิง ถ้าพอยน์เตอร์กลับสู่ค่าที่เท่ากับ MAXFILES ค่าผิดพลาดที่เกิดขึ้นหรือพอยน์เตอร์ทั้งหมดที่หาได้ถูกใช้งานอยู่ ให้ใช้ SDCard_ferror function สำหรับตรวจสอบไฟล์ผิดพลาดก่อนที่จะเข้าถึงไฟล์

หมายเหตุ ถ้าคุณใช้ SDCard_GetFilename หรือชื่อไฟล์อื่นๆซึ่งใช้ Buffer2 สำหรับการจัดเก็บชื่อไฟล์ คุณไม่ควรใช้ฟังก์ชันซึ่งปรับเปลี่ยนคอนเทนต์ของ Buffer2 ก่อนที่จะเรียก SD_fopen.

C Prototype:

*uchar SDCard_fopen(uchar *Filename, const uchar *Mode)*

Parameters:

Filename: ชื่อของไฟล์ที่จะเปิด

Mode: โหมดดังตารางต่อไปนี้เป็นโหมดที่ได้รับการอนุญาต

โหมด	ผลที่ได้รับ
r	เปิดไฟล์ที่มีอยู่เพื่อนำเข้าข้อมูล
w	สร้างไฟล์ใหม่หรือตัดไฟล์ที่มีอยู่หนึ่งไฟล์เพื่อนำข้อมูลออก
a	สร้างไฟล์ใหม่หรือเพิ่มไฟล์ที่มีอยู่หนึ่งไฟล์เพื่อนำข้อมูลออก
r+	เปิดไฟล์ที่มีอยู่เพื่ออัพเดท (ทั้งอ่านและเขียน), เริ่มการทำงาน ณ จุดเริ่มต้นของไฟล์
w+	สร้างไฟล์ใหม่หรือตัดไฟล์ที่มีอยู่หนึ่งไฟล์เพื่ออัพเดท
a+	สร้างไฟล์ใหม่หรือเพิ่มไฟล์ที่มีอยู่หนึ่งไฟล์เพื่ออัพเดท

ตารางที่ 2.16 แสดงค่าโหมดที่อนุญาตให้ใช้ใน SDCard_fopen

ค่าที่ได้:

ไฟล์ pointer แรกที่หาได้ ถ้า pointer กลับสู่ค่าที่เท่ากับ MAXFILESนั่นหมายถึงค่า error จะปรากฏขึ้นซึ่งจะไม่พบ filename หรือไฟล์ pointers ที่ใช้อยู่ทั้งหมด

ใช้ร่วมกับ:

SDCard_fclose, SDCard_GetFileName, SDCard_fputc, SDCard_fgetc, SDCard_fflush, SDCard_fseek, SDCard_ftell

SDCard_InitCard

รายละเอียด:

SDCard_InitCard เป็นการเชื่อมต่อระดับพื้นฐานที่สุดของ SD Card โดยทำการตั้งค่า card data mode ไปยัง SPI จากนั้นตรวจสอบชนิดของแล้วทำการ install ระบบไฟล์และอัพเดทตัวแปร

ส่วนกลางตามลำดับ กรุณาเรียกหาฟังก์ชันนี้ทุกครั้งเมื่อเสียบการ์ดหรือเมื่อตรวจเจอการ์ดที่ผิดปกติ ซึ่ง 1 byte ของฟังก์ชันนี้จะมีข้อมูลของชนิดการ์ดและ format type

หมายเหตุ FatType ต้องมีค่า 0x20 (FAT16 formatting) ก่อนการใช้งานการ์ด

C Prototype:

uchar SDCard_InitCard(void)

Parameters:

None.

ค่าที่ได้:

ใน 1byte ประกอบด้วย:

Card (lower nibble) 0=None detected, 1=MMC, 2=SD.

FAT (upper nibble) 00=None, 0x10=FAT12, 0x40=FAT16, 0xB0=FAT32.

ใช้ร่วมกับ:

SD_Present

SDCard_Present

รายละเอียด:

ใช้ SDCard_Present ในการตรวจสอบว่ามีการเสียบการ์ดหรือนำการ์ดออกและมีการเรียกฟังก์ชัน SDCard_InitCard หรือไม่หรือมีข้อผิดพลาดใดๆเกิดขึ้น

หมายเหตุ ฟังก์ชันนี้เป็นเพียงตัวเลือกเสริมดังนั้นจึงไม่ใช่ช่องเสียบการ์ดทุกชนิดที่จะสามารถรองรับสัญญาณนี้ได้

C Prototype:

uchar SDCard_Present(void)

Parameters:

None.

ค่าที่ได้:

1 for card present, 0 for card absent.

ใช้ร่วมกับ:

SDCard_InitCard

SDCard_Select

Description:

ใช้เพื่อทำให้สามารถหรือไม่สามารถเชื่อมต่อการ interface ระหว่าง PSoC และ SD Card.

C Prototype:

void SDCard_Select(uchar Enable)

Parameters:

Enable: เลือกให้ enable หรือ disable โดย (1 = Enable, 0 = Disable)

ค่าที่ได้:

None.

ใช้ร่วมกับ:

SDCard_Stop, SDCard_Start

SDCard_Start

รายละเอียด:

เริ่มการทำงานของ User Module โดยเริ่มที่ library code จากนั้น run SDCard_Start ก่อนที่จะใช้ SDCard library functions อื่นหรือการเข้าถึงโดย SD hardware

C Prototype:

void SDCard_Start(void)

Parameters:

None.

ค่าที่ได้:

None.

ใช้ร่วมกับ:

SDCard_Stop

SDCard_WriteProtect

รายละเอียด:

หาก SD card's write protect tab เป็น enabled ก็จะป้องกันการเขียนลงบน SD card.

หมายเหตุ ฟังก์ชันนี้เป็นเพียงฟังก์ชันเสริมและสามารถรองรับกับ SD Card มาตรฐานเท่านั้น และต้องแน่ใจว่าช่องเสียบการ์ดนั้นรองรับการตรวจสอบ write protection ด้วย

C Prototype:

uchar SDCard_WriteProtect(void)

Parameters:

None.

ค่าที่ได้:

เป็น 1 ถ้าป้องกันการเขียน, เป็น 0 ถ้าสามารถเขียนได้

ใช้ร่วมกับ:

SD_Present

File System

ข้อความเหล่านี้จะช่วยให้วิศวกรรู้ฟังก์ชันต่างๆเพื่อที่จะนำไปประยุกต์ใช้ในแอพพลิเคชันหรือการดีไซน์ได้

Disk Structure

โครงสร้างของ MMC/SD เปรียบเสมือนกับ Hard disk drive ตัวหนึ่งซึ่งรวมถึงคุณสมบัติโดยทั่วไปไม่ว่าจะเป็นรูปแบบในระบบ DOS/Windows FAT16/32 อีกทั้งยังแบ่งส่วนแยกออกเป็นส่วนเดี่ยวเหมือน drive อื่นๆโดยมีพื้นที่มาตรฐานอยู่ที่ 512 byte

Partition Table

ส่วนแรกของดิสก์จะประกอบด้วย partition table ซึ่งในตารางดังกล่าวจะประกอบไปด้วยข้อมูลที่แบ่งออกเป็น 4 drive โดยแต่ละหน่วยบันทึกตรรกะจะเริ่มต้นบันทึกข้อมูลขนาด 16 bytes ที่ offset 1BE hex และข้อมูลที่เข้ามาในแต่ละหน่วยบันทึกตรรกะก็จะปรากฏขนาดในแต่ละส่วน

รวมถึงที่ตั้งของ boot sector และข้อมูลอื่นๆ ซึ่งพื้นที่ 2 bytes สุดท้ายของส่วนนี้จะเป็น classic 55AA signature โดยทั่วไป SD/MMC Card ส่วนใหญ่จะมีหน่วยบันทึกตรรกะเพียง 1 drive และ SDCard User Module ก็จะยอมรับว่ามีเพียง 1 เช่นกัน

Boot Sector

Boot Sector ประกอบด้วยข้อมูลที่จำเป็นเกี่ยวกับ drive หรือข้อมูลที่อ้างถึง เช่น drive, sector และขนาดของกลุ่มข้อมูล อีกทั้งยังประกอบด้วยตัวเลข, ขนาดและชนิดของ File Allocation Table (FAT) และข้อมูลอื่นๆอีกมากมาย พื้นที่ส่วนแรกของจำนวน 62 bytes (100 bytes สำหรับ FAT32) ของ boot sector ประกอบด้วยข้อมูลที่น่าสนใจส่วนที่เหลือจะเป็น Code ที่ใช้ในการ boot ระบบเมื่อมีพลังงานเกิดขึ้น ส่วนพื้นที่ 2 bytes สุดท้ายจะเป็น classic 55AA signature

FAT Table

File Allocation Table ประกอบด้วยกลุ่มที่ต่อกันต่อเนื่องไปสู่ไฟล์แต่ไม่ใช่ sector กลุ่มข้อมูลนี้เป็นกลุ่มของ sector ต่อเนื่องซึ่งถูกแบ่งมา 1 หน่วย ขนาดของกลุ่มข้อมูลนี้ถูกตั้งค่าในข้อมูลของ boot sector ข้อมูล FAT ที่เข้ามาแต่ละข้อมูลต้องการพื้นที่ 2 bytes (little endian) สำหรับ FAT16 โดย FAT จะเริ่มทำงานกับ entry2 ส่วน entry 0 และ 1 จะเป็นตัวตัดสินชนิดของ FAT รายการของไฟล์ที่เข้ามาประกอบด้วยขนาดของไฟล์และ staring entry ใน FAT ซึ่ง staring entry จะมีเป้าหมายที่กลุ่มที่ต่อกันต่อเนื่องกลุ่มต่อไปหรือ FFFF เพื่อบ่งชี้จุดปลายของสาย โดยไม่จำเป็นที่จะต้องต่อกันอย่างต่อเนื่องซึ่งเราเรียกสายที่ต่อกันอย่างไม่ต่อเนื่องนี้ว่า file fragmentation และ slow file access โดยปกติ 2copies ของ FAT จะตาม boot sector

Directory

ส่วนใหญ่จะประกอบด้วย 512 entries ซึ่งแต่ละ entry จะใชพื้นที่ 32 bytes เมื่อใช้ Short filename (DOS 7.3 format) จะหมายความว่าจะมีไฟล์สูงสุดอยู่ 511 ไฟล์เนื่องจากตำแหน่งแรกนั้นได้สำรองไว้ให้สำหรับ volume label (Long filename ไม่สามารถเปลี่ยนขนาดของรายการเมื่อไฟล์ดังกล่าวใช้ entry หลายตัวในการรวมเป็น 1 filename, มากกว่า 2 filename จะถูกแบ่งส่วน) entry แต่ละ entry จะใช้เหมือน filename การลบ entry, subdirectory, directory name/volume label หรือ blank entry ข้อมูลที่มีอยู่ในแต่ละ entry คือ filename และส่วนที่เพิ่มเข้ามา (หรือ directory name/volume label), ชนิดของไฟล์, ขนาดของไฟล์, starting FAT entry, วัน/เวลา เป็นต้น filename จะมี 8 character, right padded with space, ติดตามโดยส่วนที่เพิ่มเข้าไป, also right padded with

space ตัวอักษรพิมพ์ใหญ่ทั้งหมด Long filename มักจะถูกเปลี่ยนเป็น short filename โดยอัตโนมัติ ถ้า filename มีมากกว่า 8 character มันจะตัดเป็น 6 character กับ tilde(~) ที่เพิ่มเข้ามาและตัวเลขที่ทำให้เป็นหนึ่งเดียว (เช่น 'FOO.BAR', 'MYFILE~1.TXT', MANUFA~2.XLS'.) directory จะเป็นไปตาม FAT Table บน disk

หมายเหตุ FAT32 directories เปรียบเสมือนไฟล์และความยาวก็จะแปรผัน ซึ่ง Module จะมีขีดจำกัดของ 0xFFFE entries.

File Area

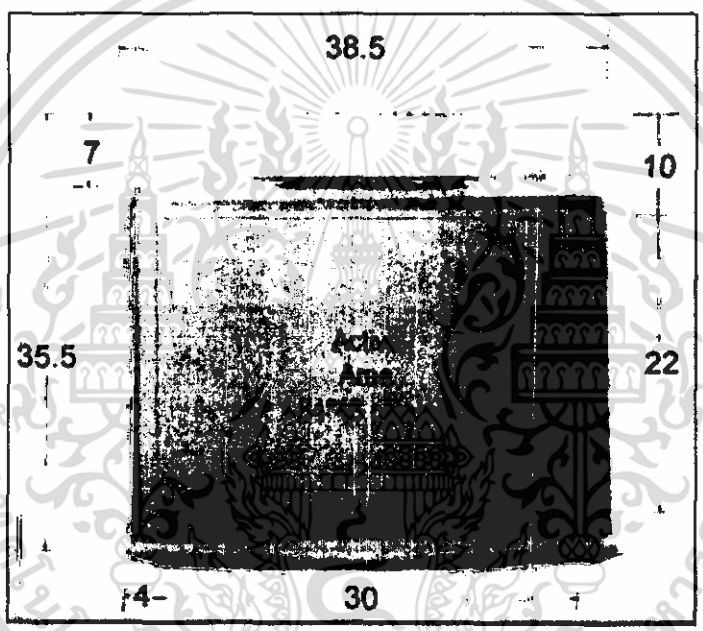
เป็นส่วนที่เหลือของ partition หลังจากจุดสิ้นสุดของ directory ซึ่งจะถูกแบ่งเข้าสู่กลุ่มข้อมูลซึ่งสอดคล้องกับ entries ใน FAT table พึงระลึกเสมอว่าต้องการจับคู่กันกับ FAT Table *entries* ดังนั้นกลุ่มข้อมูลจึงมักมี 2 กลุ่มข้อมูลเสมอ ไม่มี 0 หรือ 1

หมายเหตุ FAT12 ใช้ 3 nibbles/entry และดังนั้น entry ทุกๆ 2 ตัว จะแชร์สู่ middle byte ซึ่งมีความเก่ากว่า, ซับซ้อนกว่า, จัดการยากกว่าและ User Module ตัวนี้ไม่รองรับ FAT12 นี้ แต่ FAT32 จะมีลักษณะคล้าย FAT16 ยกเว้นในแต่ละ FAT entry จะมี 4 bytes long

2.2 หน้าจอแสดงผล LCD Nokia 5110

2.2.1 ลักษณะทางกายภาพของหน้าจอ LCD

โครงงานนี้ใช้หน้าจอแสดงผลแบบกราฟิกที่มีขนาดเล็กซึ่งเป็นหน้าจอ LCD ของ Nokia 5110 ซึ่งใช้ชิปไอซีในการเชื่อมต่อเบอร์ PCD8544 ซึ่งชิปไอซีนี้จะประกอบติดมากับหน้าจอโดยตรง PCD 8544 ออกแบบมาเพื่อควบคุมการแสดงผลข้อมูลบนจอ LCD Display ขนาด 48 row x 84 column โดยจัดเตรียมฟังก์ชันที่จำเป็นในการทำงานไว้ในชิพเดี่ยวต้องการอุปกรณ์ในการต่อเพิ่มน้อย ประหยัดพลังงาน มีขาใช้งานทั้งหมด 8 ขา ซึ่งมีลักษณะเป็นแบบ Surface Mount



รูปที่ 2.6 แสดงลักษณะหน้าจอแสดงผล LCD ของ Nokia 5110

รูปที่ 2.7 แสดงตำแหน่งขา LCD ของ Nokia 5110

ตำแหน่งขา	สัญลักษณ์	รายละเอียด
A	VDD	ขารับแรงดัน 2.7 – 3.3 V
B	GND	กราวด์
C	SCE'	ขาเซ็ทการทำงานของ LCD
D	RES'	ขารีเซ็ต
E	D/C'	ขาควบคุมการเขียนคำสั่งหรือข้อมูล
F	SDIN	ขารับข้อมูลอนุกรม
G	SCLK	ขารับสัญญาณนาฬิกา
H	LED	ขาสัญญาณควบคุมการทำงานของหลอดไฟ LED

ตารางที่ 2.17 แสดงหน้าที่ขาต่าง ๆ ของ LCD

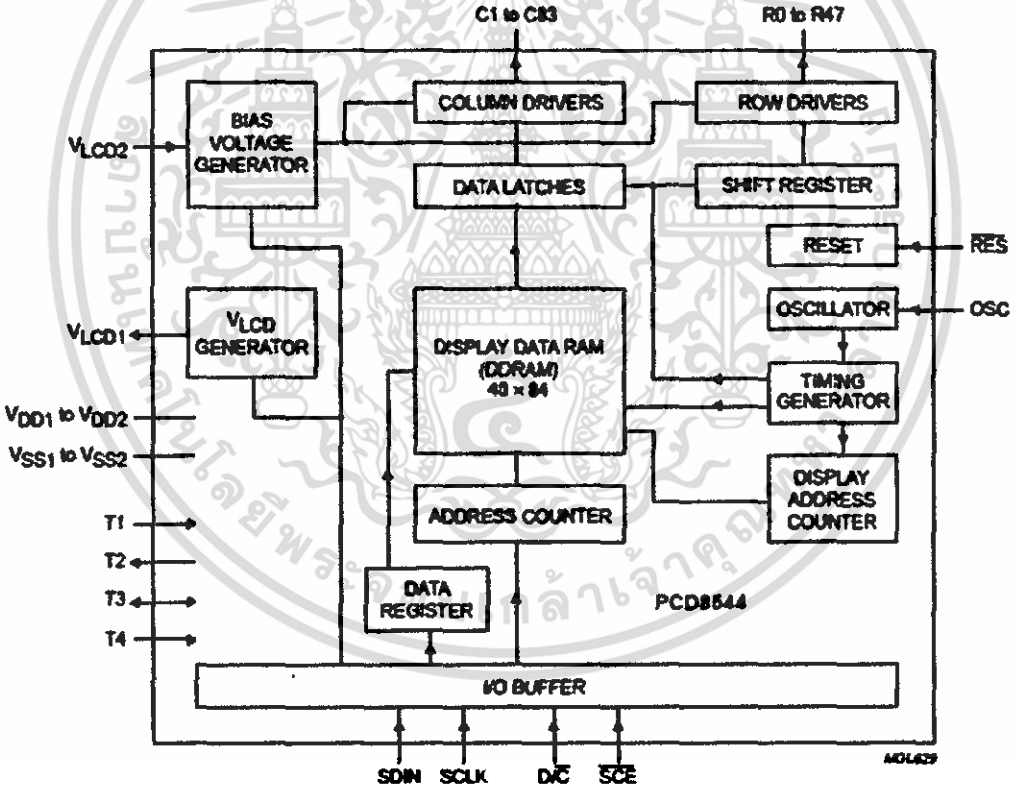
2.2.2 คุณสมบัติของหน้าจอ PCD 5844

- ใช้ชิปเดี่ยวเป็นตัว controller/driver
- มีขนาดความละเอียดของหน้าจอ 48 แถว 84 หลัก
- หน่วยความจำในการแสดงผล 48x84 บิต
- ใช้ชิปในการควบคุมการทำงาน
  - จ่ายแรงดันให้หน้าจอจากภายนอก
  - จ่ายแรงดันไบอัสให้หน้าจอแบบปานกลาง
  - ไม่จำเป็นต้องต่อสัญญาณคล็อกภายนอก เนื่องจากมีสัญญาณคล็อกบนชิพอยู่แล้ว
- รีเซตผ่านขา RES
- เชื่อมต่อแบบ Serial ส่งข้อมูลด้วยความเร็วสูงสุด 4.0 Mbits/s
- วงจรภาครับแบบ CMOS
- Mux rate : 48
- แรงดัน V_{DD} to V_{SS} ประมาณ 2.7 ถึง 3.3 โวลท์
- แรงดันไฟเลี้ยงหน้าจอ จาก V_{LCD} to V_{SS} 2 แบบ
- กินไฟน้อย เหมาะกับการใช้งานที่แรงดันไฟต่ำ
- ค่าอุณหภูมิชดเชยของ V_{LCD}
- ช่วงการทำงานในย่านอุณหภูมิ -25 ถึง 70°C

2.2.3 โครงสร้างของ PCD 8544

หน้าจอแสดงผลแบบกราฟิกที่ใช้ชิป PCD 8544 นั้น เป็นอุปกรณ์ที่ใช้กำลังขนาดต่ำและใช้ CMOS เป็นตัวควบคุมและเป็นตัว Driver โดยจะมีการแสดงผลออกหน้าจอที่มีขนาด 48 แถว 84 หลัก ในการใช้งานนั้นจะทำการส่งฟังก์ชันการใช้งานผ่านไปยังชิปซึ่งเป็นชิปเดี่ยว หลังจากนั้นชิปจะประมวลผลออกทางหน้าจอ LCD

ส่วนการใช้งานเชื่อมต่อกับไมโครคอนโทรลเลอร์นั้น จะทำการเชื่อมต่อแบบ Serial Bus ดังรูปแสดงบล็อกการทำงานภายในของ LCD



รูปที่ 2.8 แสดงบล็อกภายในชิปของ PCD 8544

SYMBOL	DESCRIPTION
R0 to R47	สัญญาณเอาต์พุตทางด้านแถวของ LCD
C0 to C83	สัญญาณเอาต์พุตทางด้านหลักของ LCD
V_{SS1} , V_{SS2}	Ground
V_{DD1} , V_{DD2}	ต่อไฟเลี้ยงของชิป
V_{LCD1} , V_{LCD2}	ต่อไฟเลี้ยงของ LCD
T1	Test 1 input (จะต่อขา V_{SS})
T2	Test 2 output (noconnect)
T3	Test 3 input/output (จะต่อขา V_{SS})
T4	Test 4 input (จะต่อขา V_{SS})
SDIN	Serial data input (ขาสัญญาณข้อมูล อินพุต)
SYMBOL	**DESCRIPTION**
SCLK	Serial clock input (ขาสัญญาณนาฬิกา 0.0 : to 4.0 Mbits/s)
D/C	Data / command (ขาเลือกส่ง Data/command)
SCE	Chip enable (เลือกให้ชิปทำงาน)
OSE	Oscillator
RES	External reset input
Dummy 1,2,3,4	Not connected

ตารางที่ 2.18 การทำงานของขาต่าง ๆ ภายในชิป PCD 8544

2.2.4 ฟังก์ชั่นในการทำงาน

1. ตัวกำหนดสัญญาณความถี่ (Oscillator) ไม่จำเป็นต้องต่อสัญญาณ Clock ภายนอก เนื่องจากมีสัญญาณ Clock บนชิพอยู่แล้ว แต่ถ้าหากต้องการต่อ Clock จากภายนอกก็สามารถต่อกับขานี้ได้เช่นกัน

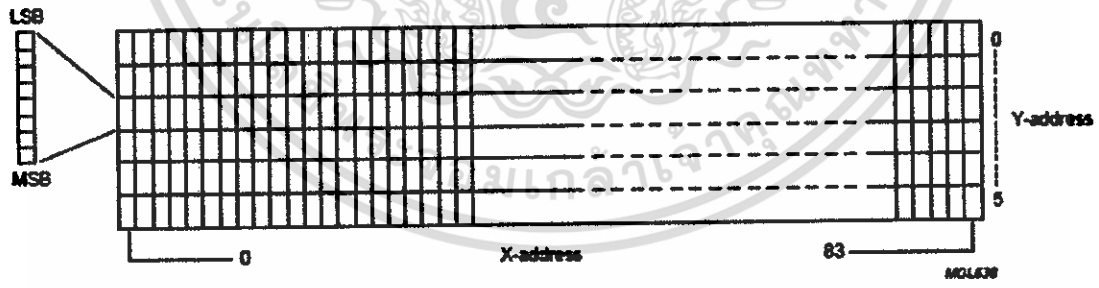
2. ตัวนับตำแหน่ง (Address Counter : AC) ตัวนับตำแหน่งเป็นตัวกำหนดค่าที่จะใช้เขียนข้อมูลจากหน่วยความจำข้อมูล โดยจะประกอบไปด้วยตำแหน่งในแกน X (X - address) X6 ถึง X0 และตำแหน่งในแกน Y (Y - address) Y2 ถึง Y0 ซึ่งสามารถแยกเซ็ตเป็นอิสระต่อกันได้ หลังจากทำการเขียนข้อมูลเสร็จแล้ว ตัวนับตำแหน่งจะเพิ่มค่าขึ้น 1 อัตโนมัติตามค่า V flag

3. หน่วยความจำข้อมูล (Display Data RAM : DDRAM) DDRAM เป็นแบบ static Ram ขนาด 48x84 bits ใช้สำหรับเก็บค่าข้อมูลที่จะทำการแสดง โดยแรมจะถูกแบ่งเป็น bank ขนาด 84 bit (6x8x84 bits) จำนวน 6 bank ในขณะที่แรมรับข้อมูลเข้ามา ข้อมูลจะถูกส่งเข้ามาทาง serial interface โดยค่าตำแหน่งในแกน X จะมีความสัมพันธ์โดยตรงกับตำแหน่งของ output column

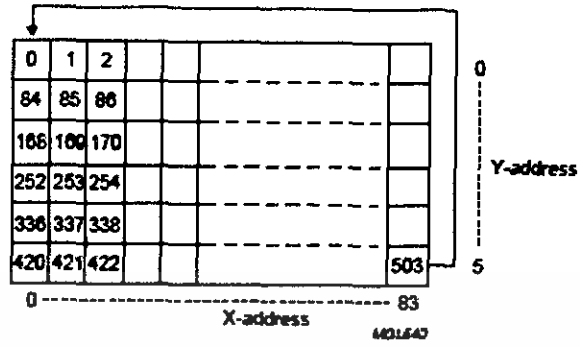
4. ตัวกำเนิดเวลา (Timing generator) ตัวกำเนิดเวลาเป็นตัวสร้างสัญญาณที่จำเป็นในการขับวงจรภายในชิพ โดยกระบวนการทำงานภายในชิพจะไม่ได้รับผลกระทบจากกระบวนการใด ๆ ที่เกิดในบัสข้อมูลเลย

5. ตัวนับตำแหน่งแสดงผล (Display Address counter) ข้อมูลที่แสดงบนจอจากการเลื่อนข้อมูลในหน่วยความจำไปยังเอาท์พุตคอลัมน์แบบจุดบนหน้าจอ LCD

6. ตำแหน่งแสดงผล (Addressing) ข้อมูลจะถูกโหลดทีละไบต์เข้าไปในหน่วยความจำ โดยจะใส่ทีละคอลัมน์จนเต็ม โดยคอลัมน์จะถูกรพบุตำแหน่งโดย address pointer ซึ่งจะมีช่วงตำแหน่งแกน X อยู่ระหว่าง 0 ถึง 83 ส่วน Y อยู่ระหว่าง 0 ถึง 5 โดยการโหลดข้อมูลอาจจะเป็นไปโดย Y address จะเพิ่มขึ้นทุก ๆ ไบต์ที่มีการโหลดเข้ามาเมื่อเต็มทั้งคอลัมน์แล้ว และก็จะเพิ่มค่าตำแหน่งแกน X หรืออาจจะทำการเพิ่มค่าในแกน X ก่อนเมื่อเต็มลัว ค่อยเพิ่มค่าในแกน Y ต่อไป จำนวนของแรมและการส่งข้อมูลออกหน้าจอเป็นดังรูป

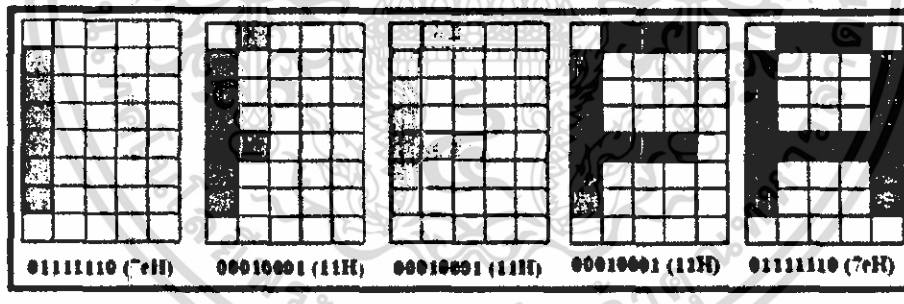


รูปที่ 2.9 แสดงรูปแบบของแรมและตำแหน่งของข้อมูล



รูปที่ 2.10 แสดงลำดับในการเขียนข้อมูลลงในหน่วยความจำแบบ horizontal addressing

7. การเก็บข้อมูลเพื่อใช้แสดงผลในการส่งข้อมูลไปแสดงเป็นอักขระบนจอเราจะทำการส่งไป 5 ไบต์ โดยจะทำการส่งบิต most ไปก่อน เพื่อเก็บในตำแหน่งบิตล่างสุดในแต่ละไบต์ ตัวอย่างการส่งข้อมูลไปแสดงผลอักขระดังแสดงในรูป



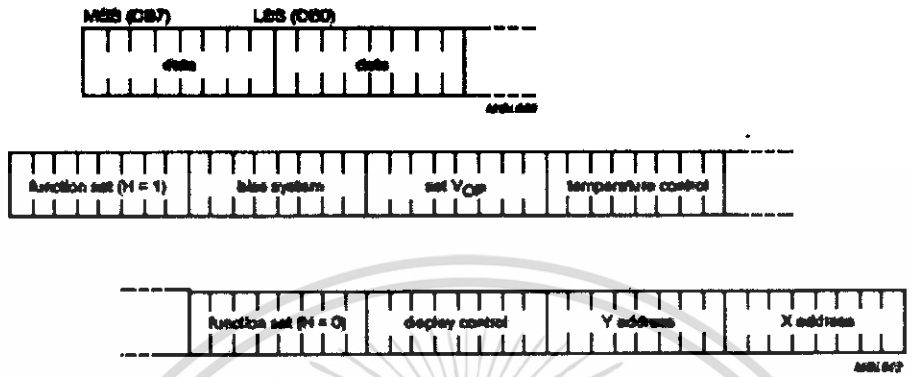
รูปที่ 2.11 แสดงลำดับขั้นการส่งข้อมูลไปแสดงผลอักขระ

2.2.5 หลักการทำงานของหน้าจอ LCD

โครงสร้างรูปแบบการทำงานจะถูกแบ่งออกเป็นสองโหมด คือ ถ้ามีการส่งค่า LOW มาจะถือว่าเป็นการทำงานในโหมด Command แต่ถ้ามีการส่ง HIGH มาก็จะเป็นการทำงานในโหมดส่งข้อมูล DATA จาก RAM โดยจะมีการนับตำแหน่งเพิ่มขึ้นอย่างอัตโนมัติ

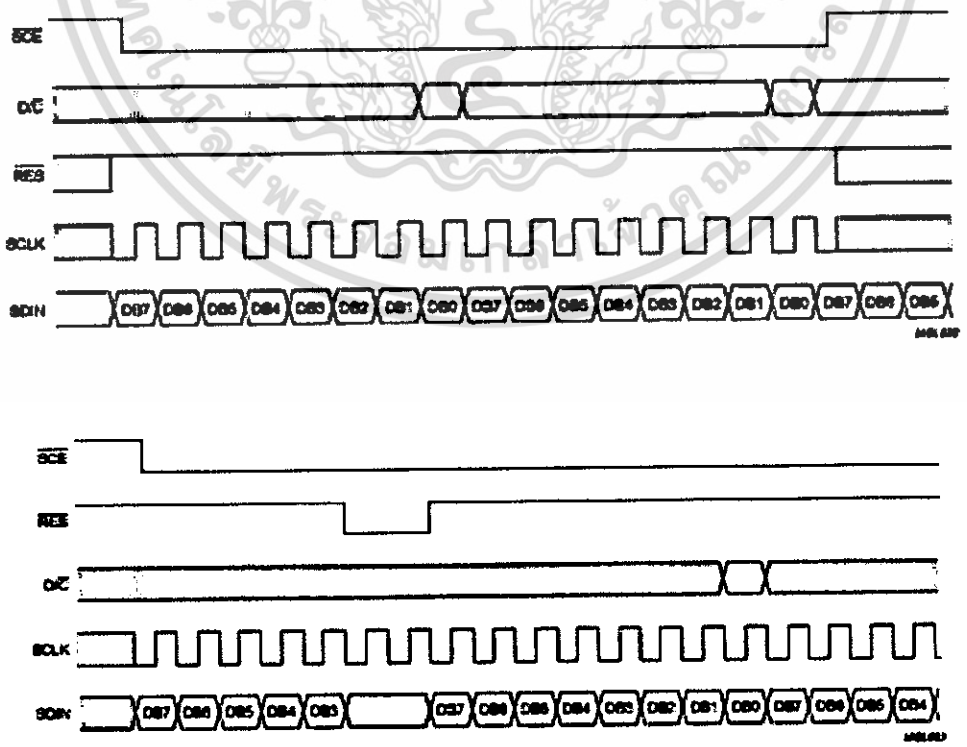
ในการส่งค่า D/$\overline{C}$ นั้นจะส่งระหว่างบิตสุดท้ายของข้อมูลในแต่ละไบต์ โดยในการส่งข้อมูลนั้น PCD 8544 จะทำการส่งค่าด้าน MSB ก่อนแล้วค่อยส่งค่าด้าน LSB และในการส่ง

แบบ Serial นั้น เริ่มต้นค่าของ -SCE จะเป็น HIGH สัญญาณนาฬิกาจะไม่มีผลและจะไม่มีการสูญเสียพลังงาน แต่เมื่อขา SCE ถูกเซตเป็น LOW จะเป็นการส่งค่าของข้อมูลได้ดังรูป



รูปที่ 2.12 แสดงหลักการส่งข้อมูลของ PCD 8544

เมื่อจะทำการส่งข้อมูลจะต้องทำการส่งค่า LOW ให้กับขา SCE และป้อนสัญญาณ SCLK และป้อนสัญญาณขา D/$\overline{C}$ ว่าจะส่งข้อมูลหรือส่งค่าคำสั่งในการทำงาน (0 = Command, 1 = DATA) หลังจากนั้นก็ทำการส่งข้อมูล โดยสามารถส่งข้อมูลผ่านขา SDIN ในการทำงานนั้นจะทำงานที่ขอบขาลงยกเว้นการส่งข้อมูลที่จะทำงานที่ขอบขาขึ้นของ SCLK และเมื่อมีการรีเซตข้อมูลจะเป็นผลให้ข้อมูลในช่วงนั้นไม่ถูกส่งไป ดังรูป



รูปที่ 2.13 แสดงการส่งข้อมูลพร้อมสัญญาณ Clock

2.2.6 รูปแบบคำสั่ง COMMAND ในการทำงานของ LCD

INSTRUCTION	D/c	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H=0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	No Operation
Function set	0	0	0	1	0	0	PD	V	H	Power down control ; extended instruction set control (H)
Write data	1	D7	D6	D5	D4	D3	D2	D1	D0	Write data to display RAM
(H=0)										
Reserved	0	0	0	0	0	0	1	X	X	Do not use
Display control										Set display configuration
Reserved	0	0	0	0	1	X	X	X	X	Do not use
Set Y address of RAM										Set Y address of RAM $0 \le Y \le 5$
Set X address of RAM	0	2	X6	X5	X4	X3	X2	X1	X0	Set X address part of RAM $0 \le X \le 83$
(H=1)										
Reserved	0	0	0	0	0	0	0	0	1	Do not use
	0	0	0	0	0	0	0	1	X	Do not use
Temperature Control	0	0	0	0	0	0	1	TC_1	TC_0	Set temperature coefficient (TC_x)
Reserved	0	0	0	0	0	1	X	X	X	Do not use
Bias system	0	0	0	0	1	0	BS_2	BS_1	BS_0	Set bias system (BS_x)
Reserved	0	0	1	X	X	X	X	X	X	Do not use
Set Vop	0	1	Vop6	Vop5	Vop4	Vop3	Vop2	Vop1	Vop0	Write Vop to register

ตารางที่ 2.19 แสดงรูปแบบคำสั่งต่างๆ ใน PCD 8544

BIT	0	1
PD	Chip is active	Chip is in Power-Down mode
V	Horizontal addressing	Vertical addressing
H	Use basic instruction set	Use extended instruction set
D and E		
00	Display blank	
10	Normal mode	
01	All display segment on	
11	Inverse video mode	
TC_1 and TC_1		
00	VIcd temperature coefficient0	
01	VIcd temperature coefficient1	
10	VIcd temperature coefficient2	
11	VIcd temperature coefficient3	

ตารางที่ 2.20 แสดงความหมายและค่าของตัวแปรในคำสั่ง COMMAND ของ PCD 8544

บทที่ 3
การออกแบบและการทดลอง

3.1 ขอบข่ายของโครงงาน

เป้าหมายในการทำโครงงานคือ การเข้าถึงข้อมูลใน SD Card ที่เป็นข้อมูลแสดงผลเป็นภาพหน้าจอแสดงผล LCD Nokia 5110 โดยเดิมที่จะใช้หน่วยความจำของไมโครคอนโทรเลอร์ในการเก็บข้อมูล แต่เนื่องจากหากจะต้องมีการเปลี่ยนแปลงข้อมูลภาพที่อยู่ในหน่วยความจำของไมโครคอนโทรเลอร์ จะต้องทำการถอดไม่โครคอนโทรเลอร์มาลงโปรแกรมที่เขียนขึ้นมาใหม่ จึงมีความยุ่งยากให้การใช้งาน จึงต้องการอ่านค่าข้อมูลภาพที่อยู่ในหน่วยความจำภายนอก โดยใช้ SD Card เป็นหน่วยความจำภายนอก

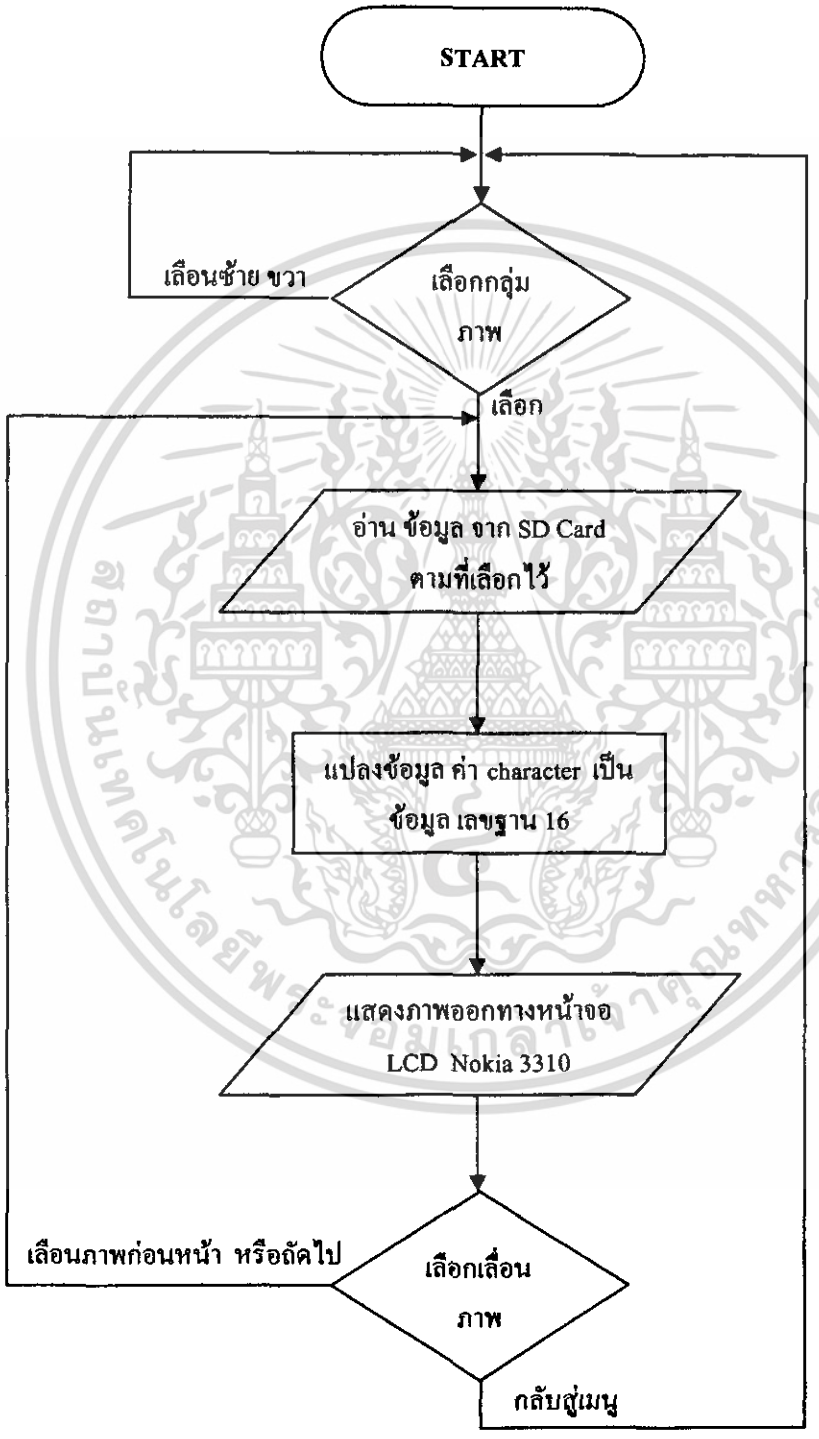
แนวความคิดในการออกแบบโครงงานนี้จะแบ่งออกเป็น 2 ส่วนคือ ส่วนแรกส่วนของโปรแกรมในไมโครคอนโทรลเลอร์ที่ใช้ในการติดต่อการเข้าถึงเข้ามูลใน SD CARD และส่วนที่สองคือ ส่วนของโปรแกรมในไมโครคอนโทรลเลอร์ที่ใช้ในการติดต่อ และส่งข้อมูลไปที่หน้าจอแสดงผล LCD Nokia 5110 ที่จะต้องรับข้อมูลเข้ามาจากโปรแกรมไมโครคอนโทรลเลอร์ที่ใช้ในการติดต่อการเข้าถึงข้อมูลใน SD CARD

จะใช้ไมโครคอนโทรลเลอร์ PSoC ในการควบคุมการติดต่อกับหน่วยความจำเอสดีการ์ด นำข้อมูลที่ได้มาประมวณผล แล้วนำไปแสดงผลทางหน้าจอแสดงผล โดยใช้ไมโครคอนโทรลเลอร์ PSoC ในการติดต่อกับหน้าจอแสดงผล

รูป 3.1 แผนผังการทำงานของโครงงาน

การติดต่อกับหน่วยความจำ SD Card นั้น ใช้การรับส่งข้อมูลแบบ SPI โดยในโปรแกรมไมโครคอนโทรลเลอร์ PSoC มี API (Application Programming Interface)ฟังก์ชันของโมดูล SD_Card ในการติดต่อกับหน่วยความจำ SD Card เป็นการช่วยให้การเขียนโปรแกรมในการเข้าถึงข้อมูลใน SD Card ง่ายขึ้น และใช้โมดูล SPIM ในโปรแกรมไมโครคอนโทรลเลอร์ PSoC ในการติดต่อกับหน้าจอแสดงผล

3.3 การออกแบบโปรแกรม



รูปที่ 3.3 โฟลว์ชาร์ทโปรแกรมการทำงานของในไมโครคอนโทรลเลอร์

ลักษณะการปฏิบัติการในโครงงานนี้ได้จัดให้มีจำนวนข้อมูลใน SD Card จำนวนทั้งหมด 30 ข้อมูล ซึ่งข้อมูลทั้งหมดจะถูกแบ่งออกเป็นกลุ่มข้อมูลจำนวน 6 กลุ่ม เพื่อให้ง่ายต่อการใช้งาน

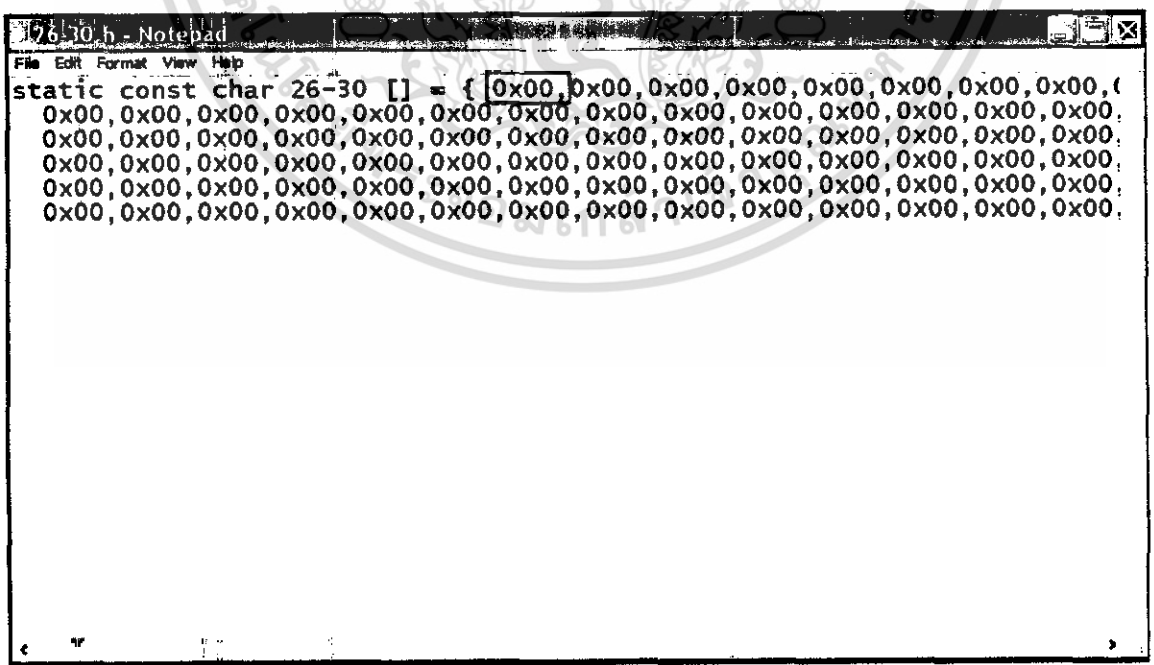
การออกแบบทางด้านโปรแกรมนอกจากจะต้องออกแบบโปรแกรมการใช้งานหลักแล้วจะต้องออกแบบการเข้าถึงข้อมูลใน SD Card และ การประมวลผลข้อมูลจาก SD Card ด้วย

3.3.1. โปรแกรมการเข้าถึงข้อมูลใน SD Card

โปรแกรมการเข้าถึงข้อมูลใน SD Card ใช้โมดูล SD_Card ในการติดต่อ และใช้ ฟังก์ชัน SDCard_fgetc ในการอ่านค่าออกมาเป็นค่า character ที่ละค่า แล้วฟังก์ชัน SDCard_fgetc จะชี้ตำแหน่งค่า character ตัวถัดไป แล้วทำการเขียนโปรแกรมวนลูปเพื่อมาอ่านค่า character ในตัวถัดไป และเก็บไว้ในตัวแปล อะเรย์(array) จนครบชุด เช่น 0x00, นำค่าที่ได้มาแปลงเป็นค่าเลขฐาน 16

3.3.2. การประมวลผลข้อมูลจาก SD Card

การอ่านข้อมูลที่เป็นไฟล์ชนิด .txt , .h , .c ที่อยู่ใน SD Card ไมโครคอนโทรเลอร์จะอ่านข้อมูลที่อยู่ในไฟล์ในลักษณะเป็นค่า Character และในโครงงานนี้ข้อมูลที่ใช้ในการประมวลผลเป็นในลักษณะ รูปที่ 3.2 ค่าที่ต้องนำมาใช้คือค่าที่เป็นค่าเลขฐาน 16 แต่การอ่านมานั้นจะเป็นค่า Character จึงต้องนำมาแปลงค่าเป็นเลขฐาน 16 ก่อน แล้วจึงส่งข้อมูลที่อ่านได้ทั้งหมดมาแสดงผลทางหน้าจอแสดงผล



รูปที่ 3.4 แสดงข้อมูลที่นำมาประมวลผล

3.4 การทดลอง

3.4.1 การทดลองในส่วนของหน้าจอกราฟฟิกแอลซีดีของ Nokia 5110

วัตถุประสงค์

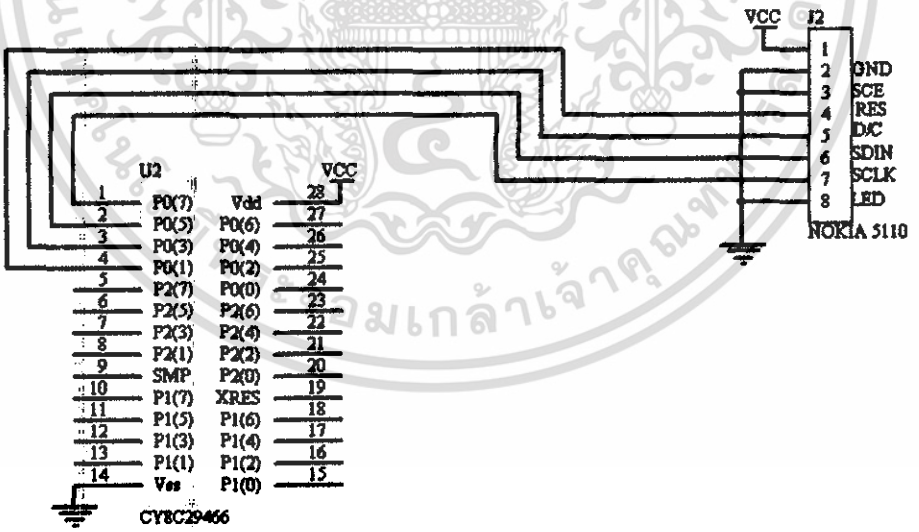
1. เพื่อเป็นการตรวจสอบดูว่าหน้าจอกราฟฟิกแอลซีดีสามารถตอบสนองการทำงานได้หรือไม่
2. เพื่อศึกษาการใช้งานหน้าจอกราฟฟิกแอลซีดี
3. เพื่อศึกษาการเขียนโปรแกรมโดยใช้ไมโครคอนโทรลเลอร์ PSoC มาควบคุมการแสดงผลหน้าจอกราฟฟิกแอลซีดี

เครื่องมือและอุปกรณ์ที่ใช้ในการทดลอง

1. หน้าจอกราฟฟิกแอลซีดีของโทรศัพท์โนเกียร์ 5110
2. ไมโครคอนโทรลเลอร์ PSoC พร้อมชุดโปรแกรม
3. แหล่งจ่ายไฟ 3.3 โวลต์

ขั้นตอนการทดลอง

1. ต่อวงจรของจอกราฟฟิกแอลซีดีเข้ากับไมโครคอนโทรลเลอร์ PSoC ดังรูป



รูปที่ 3.5 วงจรการต่อหน้าจอกราฟฟิกเข้ากับไมโครคอนโทรลเลอร์ PSoC

2. ทำการเขียนโปรแกรมเพื่อเขียนคำสั่งไปควบคุม โดยขั้นแรกเราต้องทำการกำหนดค่าเริ่มต้นต่างๆของหน้าจอ หรือที่เราเรียกว่าการ Set Initial หลังจากนั้นก็จะทำการส่ง Command ไปก่อนแล้วตามด้วย Data

3. สังเกตที่หน้าจอกราฟฟิกแอลซีดีจะมีการตอบสนองเกิดขึ้นคือ เมื่อเราเขียนคำสั่งพื่อที่ให้จอแสดงผลเป็นกลุ่มข้อมูลตัวเลข ก็จะปรากฏขึ้นที่หน้าจอตามที่เราส่งไป

รูปที่ 3.6 การแสดงผลของกราฟฟิกแอลซีดีเมื่อเขียนคำสั่งควบคุม

3.4.2 การทดลองการเข้าถึงข้อมูลใน SD Card

วัตถุประสงค์

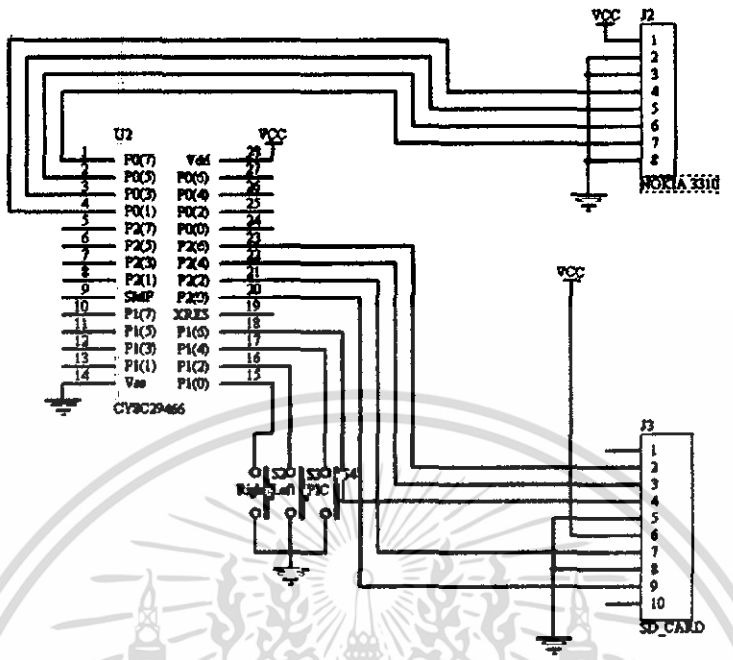
1. เพื่อศึกษาการใช้งาน SD Card
2. เพื่อศึกษาการเขียนโปรแกรมโดยใช้ไมโครคอนโทรลเลอร์ PSoC มาควบคุมการเข้าถึงข้อมูลและการประมวลผลข้อมูลใน SD Card
3. เพื่อนำข้อมูลที่ได้จากการประมวลผลมาแสดงออกทางหน้าจอกราฟฟิกแอลซีดี

เครื่องมือและอุปกรณ์ที่ใช้ในการทดลอง

1. SD Card
2. ไมโครคอนโทรลเลอร์ PSoC พร้อมชุดโปรแกรม
3. หน้าจอกราฟฟิกแอลซีดีของโทรศัพท์โนเกียร์ 5110
4. แหล่งจ่ายไฟ 3.3 โวลต์

ขั้นตอนการทดลอง

1. ต่อวงจรไมโครคอนโทรลเลอร์ PSoC ดังรูป



รูปที่ 3.7 วงจร SD Card และหน้าจอกราฟฟิกเข้ากับไมโครคอนโทรลเลอร์ PSoC

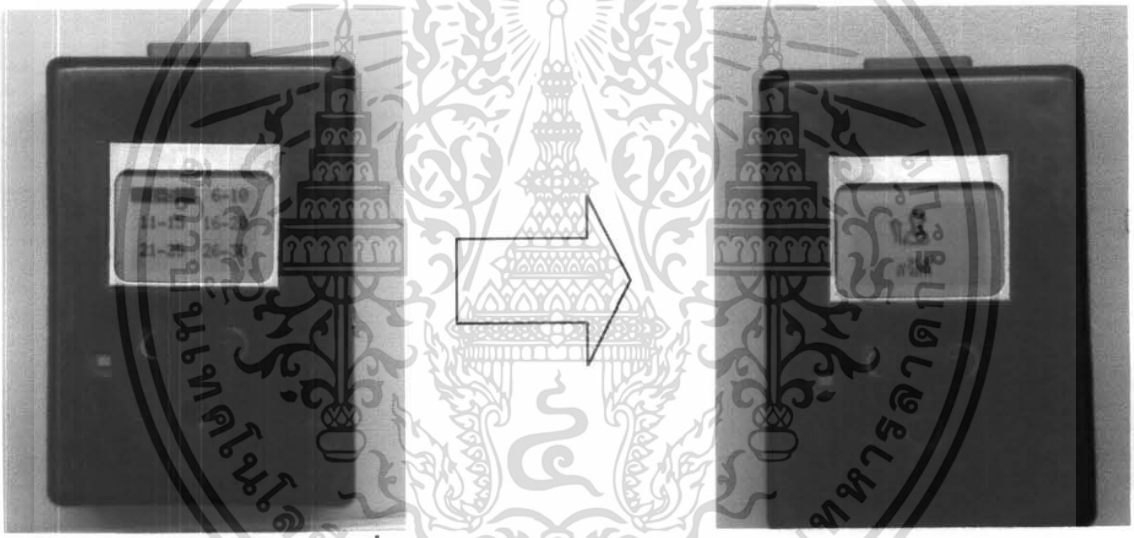
2. ทำการเขียนโปรแกรมส่งคำสั่งเซ็ตค่าไปที่ SD Card เพื่อให้ SD Card เริ่มทำงานแล้วจึงเขียนคำสั่ง SDCard_fgetc เพื่อเข้าไปอ่านข้อมูลไฟล์ใน SD Card ที่ตรงกับเงื่อนไขการประมวลผลข้อมูล แล้วจึงเก็บข้อมูลลงในตัวแปรอะเรย์ (Array)
3. นำข้อมูลในตัวแปรอะเรย์ที่ได้จากการประมวลผลมาแสดงผลออกทางหน้าจอกราฟฟิกแอลซีดี

 สวัสดี **p1**	 ดีใจจังที่คุณมาหา **p2**	 ขอบคุณนะ **p3**	 ขอโทษนะ **p4**	 ฝันดีนะ **p5**
 เป็นห่วงนะ **p6**	 ทำไมช้าจัง **p7**	 รู้สึกดีจัง **p8**	 เบื่อจังเลย **p9**	 ยินดีด้วยนะ **p10**
 หยิบของให้หน่อย **p11**	 หิวจังเลย **p12**	 อยากกินข้าว **p13**	 อยากดื่มน้ำ **p14**	 ขอดื่มน้ำหน่อย **p15**
 พาไปห้องน้ำ **p16**	 เปิดไฟให้หน่อย **p17**	 อยากพักผ่อน **p18**	 ช่วยอะไรไหม? **p19**	 วันนี้อากาศดีนะ **p20**
 อากาศร้อนจัง **p21**	 คุณไปไหนมา **p22**	 มีธุระที่ไหน? **p23**	 ใช่มั้ย? **p24**	 ใคร? **p25**
 ทำอะไร? **p26**	 ที่ไหน? **p27**	 เมื่อไหร่? **p28**	 อย่างไร? **p29**	 ทำไม? **p30**

รูปที่ 3.8 รูปที่ใช้ทั้งหมดในโครงงาน

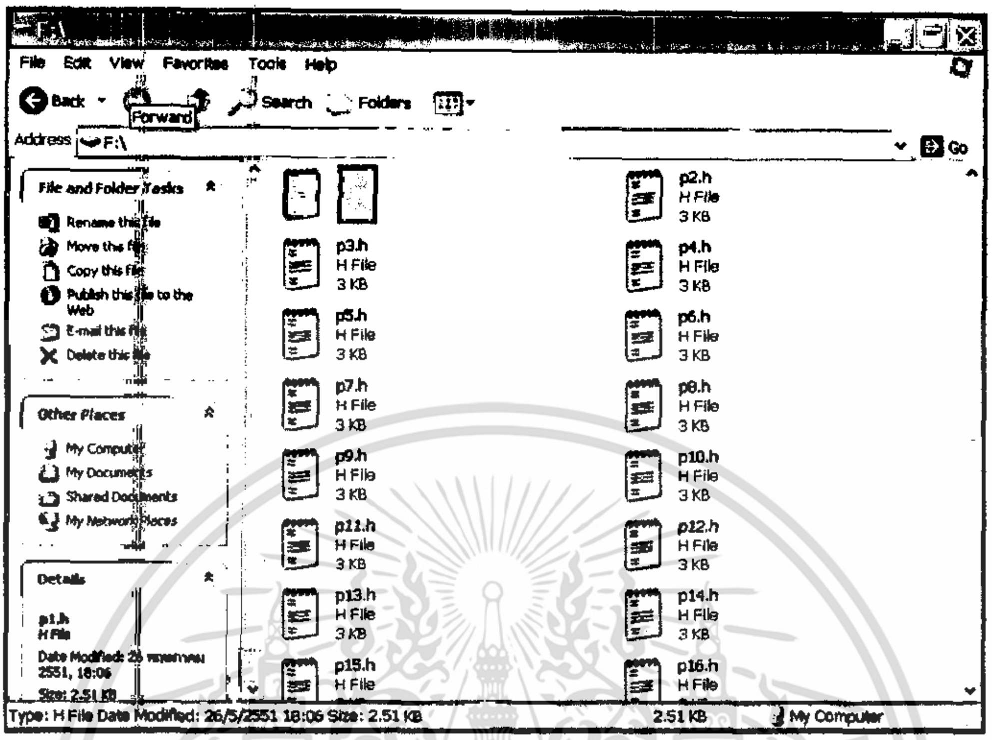
บทที่ 4
ผลการทดลอง

จากการทดลองของโครงงานนี้พบว่า การทำงานของหน้าจอกราฟฟิกแอลซีดีโนเกียร์ 5110 ข้อมูลจะถูกโหลดทีละไบต์เข้าไปในหน่วยความจำโดยจะใส่ทีละคอลัมน์จนเต็ม คอลัมน์จะถูกระบุตำแหน่งด้วย address pointer ซึ่งจะมีช่วงตำแหน่ง X อยู่ระหว่าง 0 ถึง 83 ส่วน Y อยู่ระหว่าง 0 ถึง 5 ดังนั้นการส่งข้อมูลที่เป็นภาพแบ็คกราวน์จะต้องทำการส่งข้อมูลไปทุกตำแหน่ง X และตำแหน่ง Y ทั้งหมด รวมแล้วต้องส่งข้อมูลเป็นจำนวน 504 ไบต์ ข้อมูลนั้นเป็นข้อมูลที่ได้จากการประมวณผลแล้วเก็บไว้ในตัวแปรอะเรย์ จนครบจำนวน 504 ไบต์ ด้วยเหตุนี้มีผลให้การแสดงภาพออกทางหน้าจอกราฟฟิกใช้เวลาค่อนข้างนาน

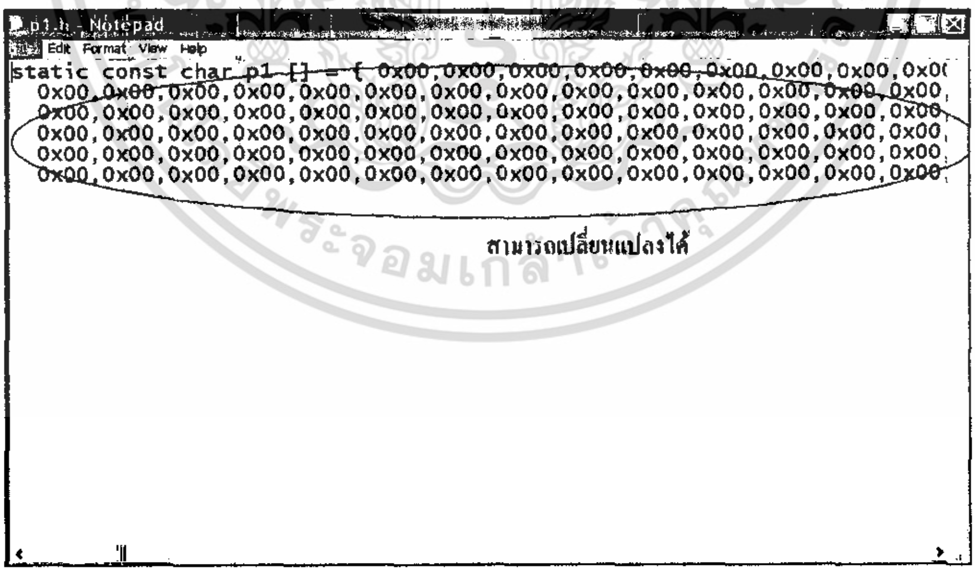


รูปที่ 4.1 แสดงลำดับการเลือกแสดงภาพ

การเข้าไปอ่านข้อมูลไฟล์ใน SD Card นั้นจะต้องระบุชื่อไฟล์ข้อมูลที่ต้องการอ่าน จึงจะสามารถเข้าไปอ่านข้อมูลในไฟล์นั้นๆได้ ดังนั้นต้องกำหนดชื่อไฟล์ทั้งหมดไว้ในขั้นตอนการเขียนโปรแกรม มีผลทำให้ต้องจำกัดจำนวนไฟล์ข้อมูลและชื่อไฟล์ ใน SD Card เราสามารถเปลี่ยนแปลงข้อมูลในไฟล์ได้ โดยที่จำนวนไฟล์ข้อมูลและชื่อไฟล์ข้อมูลใน SD Card ยังคงเหมือนเดิม



รูปที่ 4.2 ไฟล์ข้อมูลภาพใน SD Card



รูปที่ 4.3 แสดงข้อมูลในไฟล์

บทที่ 5
บทสรุป

ในปริญญานิพนธ์ได้กล่าวถึง แนวคิดและทฤษฎีในการสร้างเครื่องมือช่วยสื่อสารอย่างง่าย สำหรับผู้พิการทาการได้ยินและการพูด อุปกรณ์ที่นำมาใช้ถูกเลือกมาให้กินกำลังงานต่ำ ซึ่งทำให้เครื่องมีขนาดเล็กใช้งานและพกพาได้สะดวก ภาพและข้อความที่ใช้ เป็นภาพมาตรฐานของศูนย์อิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ นำมาเขียนเป็นไฟล์ลงใน SD Cardโดยใช้โปรแกรม Paint bush และใช้ไมโครคอนโทรเลอร์ PSoc อ่านออกมาแสดงผลบนหน้าจอกราฟฟิก ของโทรศัพท์โนเกียร์ 5110 โดยการกดเลือกคีย์สวิทช์ สามคีย์ ทั้งสามารถปรับเปลี่ยนข้อมูลภาพใน SD Card ได้ตรงตามจุดประสงค์ของโครงงาน อุปกรณ์อิเล็กทรอนิกทั้งหมดบรรจุได้ในกล่องพลาสติก ขนาด 70x114x38 มม. มีช่อเสียบการ์ดด้านบน ด้านล่างเป็นเต้ารับขั้วจ่ายไฟสำหรับ ชาร์จแบตเตอรีภายใน ด้านหน้าเป็นจอแสดงผลและคีย์สวิทช์สี่ตัว น้ำหนักเครื่องประมาณ 400 กรัม การพัฒนาต่อสามารถทำได้โดย ให้มีการแสดงผลทางเสียง และ เพิ่มจำนวนไฟล์ข้อมูล

บรรณานุกรม

1. http://www.cypress.com/portal/server.pt?space=CommunityPage&control=SetCommunity&CommunityID=285&PageID=552&shortlink=SD1070&ref=shk&CID=ILC-shortlinks&shk=SD1070 , "PSoC Designer New User Module Extension Packs".

2. www.semiconductors.philips.com/acrobat/datasheet/PCD8544_1.pdf , "PCD8544".

3. http://www.thaieasyelec.com/index.php?lay=show&ac=article&Id=420836&Ntype=1 , "แกะรอย LCD 3310 NOKIA".

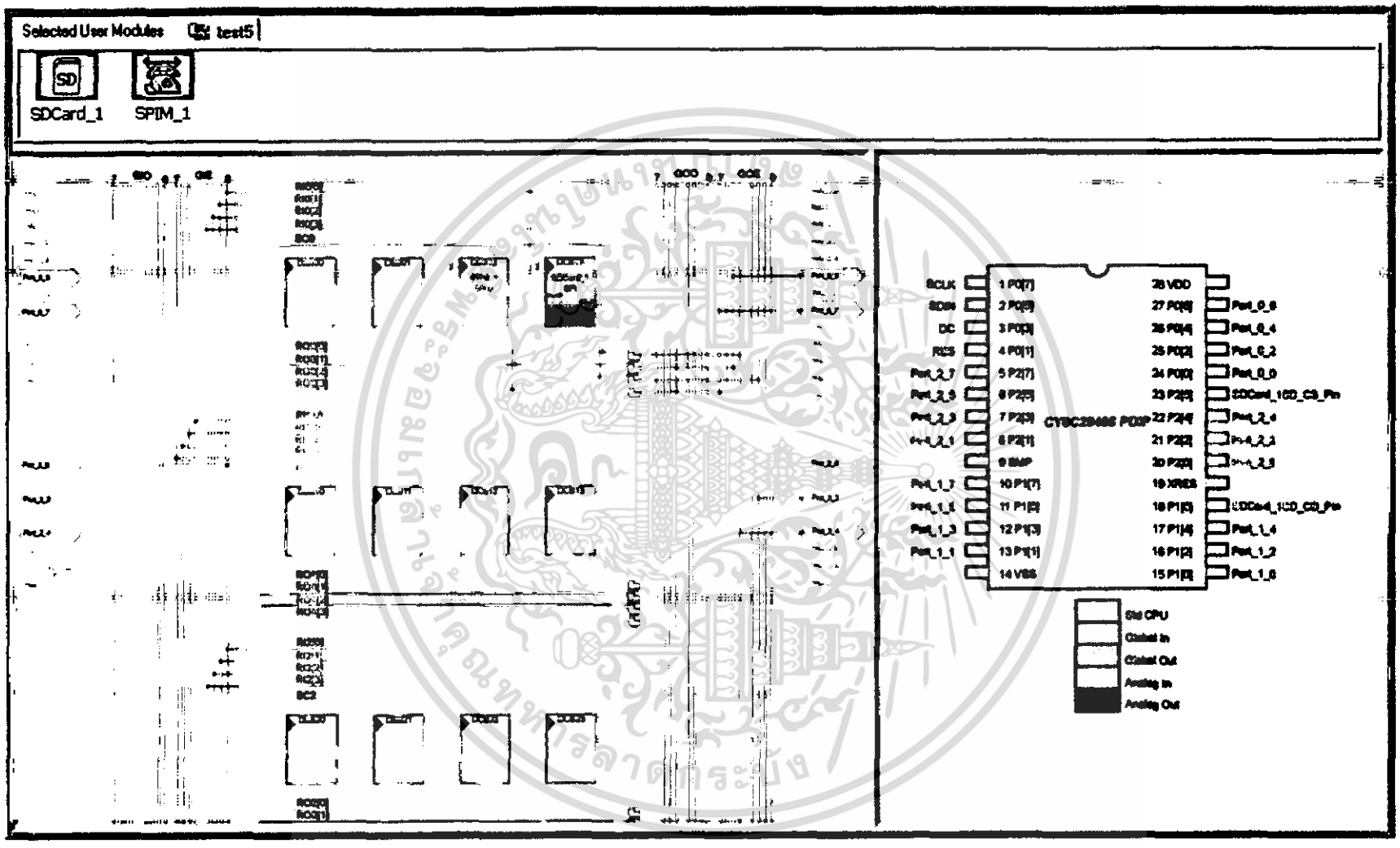
4. www.100acre.org/elec/nokia_lcd/nokia_lcd.jal , "Nokia 3310 LCD Driver Library".

5. http://www.ett.co.th/product/intf/ET-NOKIA-5110-LCD.pdf , "ขอแสดงผลกราฟิก ET-NOKIA LCD 5110".

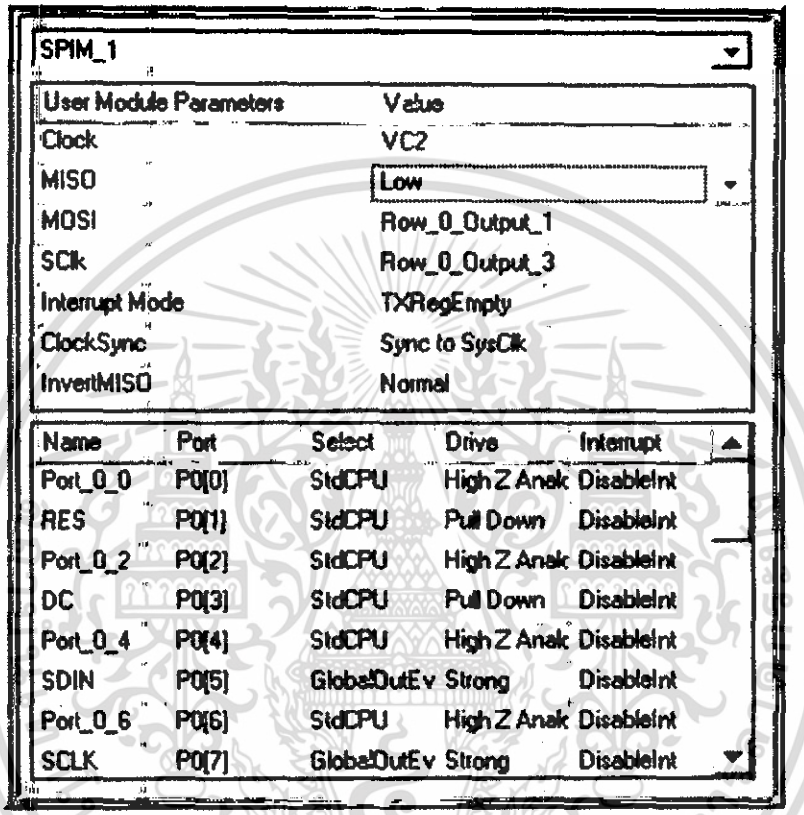
6. วัชรินทร์ เคารพ "เรียนรู้และเข้าใจ PSoC Microcontroller ด้วยภาษา Assembly และภาษา C" อีทีที กรุงเทพ 2548

7. อุกฤษฏ์ ตันทสุทธานนท์ " การเขียนโปรแกรมไมโครคอนโทรลเลอร์ PSoC ด้วยภาษาซี" MRT กรุงเทพ 2548

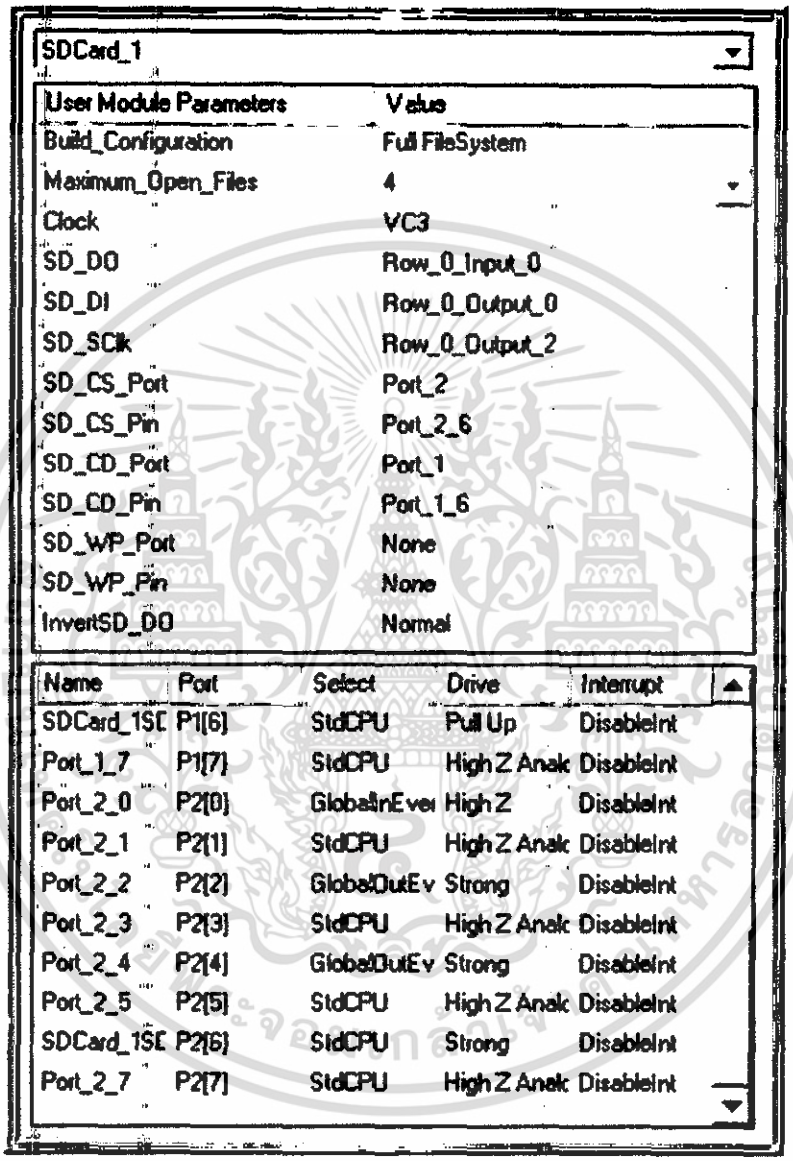
ภาคผนวก



การวางโมดูล SPIM และ SD Card ลงบล็อก



การกำหนดค่าโมดูล SPIM



การกำหนดค่าโมดูล SD Card

โปรแกรม

MAIN PROGRAM

```
#include "m8c.h"
#include "PSoCAPI.h"
#include "stdlib.h"
#include "delay.h"
#include "ioport.h"
#include "display5.h"

#define  CARD_PRESENT     1

char FileReadp1[30][6]  =
{"p1.h","p2.h","p3.h","p4.h","p5.h","p6.h","p7.h","p8.h","p9.h","p10.h","p11.h","p12.h","p13.h",
"p14.h","p15.h","p16.h","p17.h","p18.h","p19.h","p20.h","p21.h","p22.h","p23.h","p24.h","p25.h
","p26.h","p27.h","p28.h","p29.h","p30.h" };

void main()
{
  int i,count,move,background;
  char   cardInfo;                  // Card information


  SDCard_1_Start( );                  // initalize hardware and SDCard_lib buffers
  SetBit1_0;
  SetBit1_2;
  SetBit1_4;
  SetBit1_6;
  count=0;
  move=0;

  while(1)
```

```
{

    if(SDCard_1_Present() == CARD_PRESENT)    // Card inserted
    {

      SDCard_1_Select(SDCard_1_ENABLE);
      cardInfo = 0;
      while ( ! cardInfo )        // Wait for card to communicate
      {
        cardInfo = SDCard_1_InitCard();   // initialize card, determine card type and file system
type

      }
      SetScreen ();
      Delay1mS(1000);

      while(move==0)
      {
         if(SDCard_1_Present() == CARD_PRESENT)
         {
                  if(Bit1_0==0){count++; SetScreen ();}
                  if(Bit1_2==0){count--; SetScreen ();}
                           if(count>5){count=0;}
                           if(count<0){count=5;}
                           switch (count)
                           {
                             case 0:
                                       GotoXY(0,0);
                                  for(i=0; i<12; i++)

                                  {
```

```
                SendData(0xFF);
}
InvertString("1-5");
for(i=0; i<12; i++)
{
                SendData(0xFF);
}
background=0;
        break;
 case 1:
        GotoXY(42,0);
for(i=0; i<12; i++)
{
                SendData(0xFF);
}
InvertString("6-10");
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
background=5;
        break;
 case 2:
        GotoXY(0,2);
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
InvertString("11-15");
for(i=0; i<6; i++)
```

```
{
                SendData(0xFF);
}
background=10;
        break;
 case 3:
        GotoXY(42,2);
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
InvertString("16-20");
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
background=15;
        break;
 case 4:
        GotoXY(0,4);
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
InvertString("21-25");
for(i=0; i<6; i++)
{
                SendData(0xFF);
}
background=20;
```

```
                    break;
                case 5:
                    GotoXY(42,4);
                for(i=0; i<6; i++)
                {
                        SendData(0xFF);
                }
                InvertString("26-30");
                for(i=0; i<6; i++)
                {
                        SendData(0xFF);
                }
                background=25;
                    break;
                default:SetScreen ();
                }
            if(Bit1_4==0){move=1; NokiaReadFile(FileReadp1[background]);}
        }
        else{move=1; count=0; SDCard_1_Select(SDCard_1_DISABLE);}
    }
    while(move==1)
        {
        if(SDCard_1_Present() == CARD_PRESENT)
        {
        if (Bit1_0==0){background++; if (background>29){background=0;}
NokiaReadFile(FileReadp1[background]);}
        if (Bit1_2==0){background--; if (background<0){background=29;}
NokiaReadFile(FileReadp1[background]);}
        if (Bit1_4==0){move=0; count=background/5;}
        }
```

```
                else{move=0; count=0; SDCard_1_Select(SDCard_1_DISABLE);}
               }

      SDCard_1_Select(SDCard_1_DISABLE);

    }
    else
    {
      NokiaStart(test1);
      GotoXY(20,2);
      SuperString("NO CARD");
    }
  }
}
```

DISPLAY AND SD CARD PROGRAM

```
#include "m8c.h"
#include "spim_1.h"
#include "PSoCAPI.h"
#include "display5.h"
#include "ioport.h"

unsigned char data1[252];
unsigned char data2[252];
int Background_cursor;


//Send Data
void SendData ( char data )
{
 PRT0DR |= DC_PIN;
 SPIM_1_SendTxData(data);
 while ( ! (bSPIM_1_ReadStatus() & SPIM_1_SPIM_TX_BUFFER_EMPTY) );
}

//Send Command
void SendCommand ( char command )
{
 PRT0DR &= ~DC_PIN;
 SPIM_1_SendTxData(command);
 while ( ! (bSPIM_1_ReadStatus() & SPIM_1_SPIM_TX_BUFFER_EMPTY) );
}

static void wait ( void )
{
```

```
 int time;
 for (time= -32000; time< 32000; time++ );
}


void NokiaSetting ( void )
{
 int i;
 wait();
 PRT0DR = RES_PIN;
 SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
 SendCommand(0x21); // LCD Extended Commands.
 SendCommand(0x88); // Set LCD Vop (Contrast).
 SendCommand(0x06); // Set Temp coefficent.
 SendCommand(0x13); // LCD bias mode 1:48.
 SendCommand(0x20); // LCD Standard Commands, Horizontal addressing mode.
 SendCommand(0x0C); // LCD in normal mode.
 SendCommand(0x40); //clear display
 SendCommand(0x80); //clear display
 for(i=0; i<504; i++)
 {
        SendData(0x00);
 }
}
//-----------------------------------------------------------------------


#ifdef CHEN
void NokiaStart (const char * background)
#else
```

```
void NokiaStart ( void )
#endif
{
 int i;
 wait();
 PRT0DR = RES_PIN;
 SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
 SendCommand(0x21); // LCD Extended Commands.
 SendCommand(0x88); // Set LCD Vop (Contrast).
 SendCommand(0x06); // Set Temp coefficent.
 SendCommand(0x13); // LCD bias mode 1:48.
 SendCommand(0x20); // LCD Standard Commands, Horizontal addressing mode.
 SendCommand(0x0C); // LCD in normal mode.
 SendCommand(0x40); //clear display
 SendCommand(0x80); //clear display
#ifdef CHEN
 Background = background;
 for(i=0; i<504; i++)
 {
        SendData(Background[i]);
 }
 Background_cursor = 0;
#else
 for(i=0; i<504; i++)
 {
        SendData(0x00);
 }
#endif
}
```

```
//Set Screen

void SetScreen (void)
{
 NokiaStart(test1);
 GotoXY(12,0);
 SuperString("1-5");
 GotoXY(54,0);
 SuperString("6-10");

 GotoXY(6,2);
 SuperString("11-15");
 GotoXY(48,2);
 SuperString("16-20");

 GotoXY(6,4);
 SuperString("21-25");
 GotoXY(48,4);
 SuperString("26-30");
}

//Goto XY

void GotoXY ( char x, char y )
{
 // SendCommand(0x40 | y);
```

```
 // SendCommand(0x80 | x);
  SendCommand(0x40|(y&0x07));
  SendCommand(0x80|(x&0x7f));
#ifdef CHEN
  Background_cursor = y*84+x;
#endif
}

//CString

#ifdef CHEN
void SuperString ( const char *dataPtr)
{
  int i;
  while ( *dataPtr )
  {
    char ch = *dataPtr++;

     for(i=0; i<5; i++, Background_cursor++)
     {
          SendData(Font[ch-32][i] ^ Background[Background_cursor]);
     }
    SendData(Background[Background_cursor++]);
  }
}
#else
void SuperString ( const char *dataPtr )
{
  int i;
  while ( *dataPtr )
```

```
  {
   char ch = *dataPtr++;
   for(i=0; i<5; i++)
   {
        SendData(Font[ch-32][i]);
   }
   SendData(0x00);
  }
}
#endif


// InvertString

#ifdef CHEN
void InvertString ( const char *dataPtr)
{
 int i;
 while ( *dataPtr )
 {
  char ch = *dataPtr++;

   for(i=0; i<5; i++)
   {
        SendData(Invert_Font[ch-32][i]);
   }
  SendData(0xFF);
 }
}
#else
```

```
void InvertString ( const char *dataPtr )
{
 int i;
 while ( *dataPtr )
 {
  char ch = *dataPtr++;
  for(i=0; i<5; i++)
  {
        SendData(Invert_Font[ch-32][i]);
  }
  SendData(0xFF);
 }
}
#endif

//-------------------------------------------------------------------

unsigned char ASCII_2_DEC_1 (char Read)
{       char Result;
        switch (Read)
        {       case '0': Result= 0x00;
                                        break;
                case '1': Result= 0x01;
                                        break;
                case '2': Result= 0x02;
                                        break;
                case '3': Result= 0x03;
                                        break;
                case '4': Result= 0x04;
                                        break;
```

```
        case '5': Result= 0x05;
                                break;
        case '6': Result= 0x06;
                                break;
        case '7': Result= 0x07;
                                break;
        case '8': Result= 0x08;
                                break;
        case '9': Result= 0x09;
                                break;
        case 'A':
        case 'a': Result= 0x0A;
                                break;
        case 'B':
        case 'b': Result= 0x0B;
                                break;
        case 'C':
        case 'c': Result= 0x0C;
                                break;
        case 'D':
        case 'd': Result= 0x0D;
                                break;
        case 'E':
        case 'e': Result= 0x0E;
                                break;
        case 'F':
        case 'f': Result= 0x0F;
                                break;
}
return Result;
```

```
}

unsigned char ASCII_2_DEC_2 (char Data_H,char Data_L,char Data_Ent)
{       unsigned char Result;

      if ((Data_Ent==',')||(Data_Ent==' '))
      {
         Result = ASCII_2_DEC_1 (Data_L);
            Result |= (ASCII_2_DEC_1 (Data_H)<<4);
      }

  else
  {
    Result = 0x00;
    Result |= (0x00<<4);
  }

        return Result;
}
//---------------------------------------------------------------------

void NokiaReadFile(char * ReadFile)
{
 unsigned char status[5]={0};
 unsigned char Fptr;                  // Temp file pointers
 int i , m , n;
  Fptr = SDCard_1_fopen(ReadFile, "r");    // Open file by filename for Read
  if(Fptr < SDCard_1_MAXFILES)              // Did the open succeed?
  {
```

```
SDCard_1_clearerr(Fptr);          // Clear error flags
NokiaSetting();

status[0]=SDCard_1_fgetc(Fptr);

if (status[0]='s')
{
while(status[0]!='{')
{

  status[0]=SDCard_1_fgetc(Fptr);

}
  m=0;
  n=0;
  while(status[0]!=';')
  {

    if (status[0]=='0')
    {
      status[1]=SDCard_1_fgetc(Fptr);

      if (status[1]=='x')
      {
        for(i=2; i<5; i++)
        {
          status[i]=SDCard_1_fgetc(Fptr);
        }
        if (m==0)
        {
```

```
            data1[n]=ASCII_2_DEC_2 (status[2],status[3],status[4]);
            n++;
            if (n==252)
            {
              m=1;
              n=0;
            }
          }
          else if (m==1)
          {
            data2[n]=ASCII_2_DEC_2 (status[2],status[3],status[4]);
            n++;
          }
        }
      }
      status[0]=SDCard_1_fgetc(Fptr);

  }
  for (n=0; n<252; n++)
  {
    SendData(data1[n]);
  }
  for (n=0; n<252; n++)
  {
    SendData(data2[n]);
  }
  }

}
```

```
            data1[n]=ASCII_2_DEC_2 (status[2],status[3],status[4]);
            n++;
            if (n==252)
            {
              m=1;
              n=0;
            }
          }
          else if (m==1)
          {
            data2[n]=ASCII_2_DEC_2 (status[2],status[3],status[4]);
            n++;
          }

        }
      }
        status[0]=SDCard_1_fgetc(Fptr);
    }
    for (n=0; n<252; n++)
    {
      SendData(data1[n]);
    }
    for (n=0; n<252; n++)
    {
      SendData(data2[n]);
    }
    }
  }
  SDCard_1_fclose(Fptr);              // Close the files when finished
}
```

Display5.h

```
#ifndef _DISPLAY1_H_
#define _DISPLAY1_H_

#define CHEN

#define DC_PIN 0x08 // PRT0_3
#define RES_PIN 0x02 // PRT0_1
#include "Invert_Font.h"
#include "small_font.h"
#include "test1.h"

#ifdef CHEN
static const char *Background;

/*typedef enum
{
 SHW = 0,
 CLR = 1
} D_type;*/
#endif

//void Clrscr (void);
//void Contrast (char contrast);
void SetScreen (void);
void GotoXY (char x, char y);
void SendData (char data);
void SendCommand (char command);
void NokiaSetting ( void );
unsigned char ASCII_2_DEC_1 (char Read);
```

```
unsigned char ASCII_2_DEC_2 (char Data_H,char Data_L,char Data_Ent);
void NokiaReadFile(char * ReadFile);


#ifdef CHEN
void NokiaStart (const char * dataPtr);
//void SetBackground (const char * dataPtr);
void SuperString (const char * dataPtr);
void InvertString ( const char *dataPtr);


#else
void NokiaStart (void);
void SuperString (const char * dataPtr);
void InvertString ( const char *dataPtr);


#endif

#endif
```