เครื่องแสดงภาพบนจอขนาดเล็ก

Digital Image Viewer

นาย ทักษากร สัตยากวี

นาย ปิยะพงษ์ พิสิษฐ์กุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

เครื่องแสดงภาพบนจอขนาดเล็ก

Digital Image Viewer

โดย

นาย ทักษากร สัตยากวี

นาย ปิยะพงษ์ พิสิษฐ์กุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

เครื่องแสดงภาพบนจอขนาดเล็ก

Digital Image Viewer

โดย

นาย ทักษากร สัตยากวี รหัส 47010281

นาย ปิยะพงษ์ พิสิษฐ์กุล รหัส 47010460

อาจารย์ที่ปรึกษา

ดร. กสิน วิเชียรชม

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องแสดงภาพบนจอขนาดเล็ก

(Digital Image Viewer)

ผู้จัดทำ 1.นายทักษากร สัตยากวี รหัส 47010281 ชั้นปีที่ 4

2. นายปิยะพงษ์ พิสิษฐ์กุล รหัส 47010460 ชั้นปีที่ 4

.....อาจารย์ที่ปรึกษา

(ดร. กสิน วิเชียรชม)

เครื่องแสดงภาพบนจอขนาดเล็ก

นาย ทักษากร สัตยากวี รหัส 47010281
นาย ปิยะพงษ์ พิสิษฐ์กุล รหัส 47010460
ดร. กสิน วิเชียรชม อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

เครื่องแสดงภาพบนจอขนาดเล็กที่ได้ทำการออกแบบ แบ่งออกเป็น 3 ส่วน คือ ส่วนเก็บข้อมูล ซึ่งเป็นไฟล์ภาพใน SD Card โดยมีการจัดเก็บข้อมูลแบบ FAT ส่วนแสดงผลจะใช้จอ LCD ซึ่งเป็นจอสีขนาดเล็ก สามารถแสดงผลได้หลากหลายสี ส่วนสุดท้ายเป็นส่วนควบคุม จะใช้ ไมโครคอลโทรเลอร์ มาประมวลผล โดยใช้การรับส่งข้อมูลแบบ SPI ซึ่งเครื่องนี้สามารถแสดง ไฟล์ Bitmap

Digital Image Viewer

Mr. Taksakron Sattayakawee ID.47010281

Mr. Piyapong Pisitkul ID.47010460

Dr. Kasin Vichienchom Advisor

Education Year 2007

Abstract

This Digital Image Viewer with built-in Secure Digital Card (SD Card) interface includes three parts which are a SD Card reader, a LCD display and a microcontroller. The image data is stored in SD Card using FAT file system. The LCD can be shown in various colors. A controller is served as a controller unit. It transfers the image data from the SD Card to the LCD display by SPI interface. This Digital Image Viewer can display bitmap files.

กิตติกรรมประกาศ

โครงงานนี้สามารถลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือจากหลายๆท่าน โดยเฉพาะอย่างยิ่ง ดร.กสิน วิเชียรชม (อาจารย์ที่ปรึกษา) ที่คอยให้คำปรึกษาเกี่ยวกับปริญญานิพนธ์ อีกทั้งพี่ ที่คอยช่วยเหลือในการปฏิบัติงานเป็นอย่างดีมาโดยตลอด จนทำให้โครงงานนี้สำเร็จโดยสมบูรณ์ได้

จึงขอขอบคุณมา ณ ที่นี้

นายทักษากร สัตยากวี รหัส 46010281

นายปิยะพงษ์ พิสิษฐ์กุล รหัส 46010460

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความเป็นมาของโครงงาน

ปัจจุบันเทคโนโลยีของการแสดงผลด้วยจอแอลซีดีมีวิวัฒนาการที่ดีขึ้น โดยได้มีจอแอลซีดีที่สามารถแสดงผลเป็นสีได้และมีราคาที่ไม่แพงมาก การนำมาเชื่อมต่อกับไมโครคอนโทรลเลอร์ในการควบคุมการแสดงผลก็ไม่ยุ่งยากมากนัก โดยแอลซีดีที่ว่านี้ คือ แอลซีดีที่ถูกใช้กับเครื่องโทรศัพท์มือถือที่มีขายอยู่ในปัจจุบันนี้ ซึ่งมีการแสดงผลได้หลายสี โดยความคมชัดของภาพจะขึ้นอยู่กับความละเอียดของจอภาพและความละเอียดของสีที่จอนั้นแสดงได้ ซึ่งโครงงานนี้เราเลือกใช้หน้าจอแสดงผลแอลซีดีที่ทางบริษัทโนเกียได้นำมาใช้ ซึ่งสามารถหาข้อมูลเกี่ยวกับคุณสมบัติต่างๆของจอ ได้จากบทความทางอิเล็กทรอนิกส์ ข่าวสารทางอินเตอร์เน็ต

การ์ดหน่วยความจำ (Memory Card) ได้มีการพัฒนาออกสู่ตลาดมากมายหลายรูปแบบ เช่น Memory Stick, Smart Media, Compact Flash, USB, SD/MMC Card เป็นต้น การใช้งานก็จะมีลักษณะแตกต่างกัน ในส่วนของผู้จัดทำได้เลือกใช้ SD Card เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไมแพงมากและหาซื้อได้ตามท้องตลาดทั่วไป

ในส่วนของไมโครคอนโทรลเลอร์ ได้เลือกใช้ไมโครคอนโทรลเลอร์ตระกูล PIC เนื่องจากผู้จัดทำต้องการไมโครคอนโทรลเลอร์ที่ต้องการกำลังไฟฟ้าค่อนข้างต่ำ มีการประมวลผลที่รวดเร็ว ผู้จัดทำจึงเลือกใช้ไมโครคอนโทรลเลอร์ตระกูล PIC (PIC16F4620) ในการควบคุมการทำงานทั้งหมด และใช้ภาษาซีในการเขียนโค้ดโปรแกรมควบคุมส่วนต่างๆ

1.2 วัตถุประสงค์

เพื่อศึกษาการเขียน-อ่านข้อมูลการ์ดหน่วยความจำแบบ SD Card ศึกษาการถอดรหัสไฟล์ภาพแบบต่างๆ ศึกษาการติดต่อสื่อสารข้อมูลในแบบ SPI ศึกษาการใช้งานเกี่ยวกับหน้าจอแอลซีดีมือถือที่มีการแสดงผลได้หลายสี ซึ่งแสดงภาพต่างๆที่ได้จากไฟล์ใน SD Card ได้ ตลอดจนเพิ่มทักษะในการใช้งานไมโครคอนโทรลเลอร์ตระกูล PIC

1.3 ขอบเขตของโครงงาน

โครงงานเครื่องแสดงภาพขนาดเล็กชนิดพกพาที่มี SD Card นี้ ได้ออกแบบและสร้างวงจรในการเขียน อ่านการ์ดหน่วยความจำแบบ SD Card ในส่วนของการแปลงไฟล์ภาพให้เป็น Hex Byte จะใช้โปรแกรมจากในคอมพิวเตอร์ช่วย แล้วนำไฟล์ที่แปลงนั้นมาแสดงผลทางหน้าจอ ซึ่งการดิดต่ออุปกรณ์ทั้งสองจะเป็นการดิดต่อสื่อสารในแบบ SPI ซึ่งจะแสดงผลออกทางหน้าจอแอลซีดีมือถือ และใช้ไมโครคอนโทรลเลอร์ PIC เบอร์ 18F4620 ในการควบคุม

1.4 โครงสร้างของรายงาน

รายงานฉบับนี้ได้อธิบายขั้นตอน วิธีในการออกแบบ รวมทั้งวงจร และผลการทดลองทดสอบคุณสมบัติต่างๆของเครื่องแสดงภาพขนาดเล็กชนิดพกพาที่มี SD Card โดยมีเนื้อหาแบ่งเป็นบทต่างๆ ดังนี้

บทที่ 2 ทฤษฎี กล่าวถึงทฤษฎี และหลักการพื้นฐานต่างๆที่เกี่ยวข้องกับการออกแบบและสร้างเครื่องแสดงภาพขนาดเล็กชนิดพกพาที่มี SD Card

บทที่ 3 การออกแบบ กล่าวถึงขั้นตอนในการออกแบบและโค้ดโปรแกรมในการติดต่อกับการ์ดหน่วยความจำแบบ SD Card การติดต่อกับหน้าจอแอลซีดีมือถือแสดงผล โดยใช้ไมโครคอนโทรเลอร์ PIC

บทที่ 4 การทดลอง และผลการทดลอง กล่าวถึงการทดลอง และผลการทดลอง เมื่อทำการทดสอบสั่งงานเครื่องแสดงภาพขนาดเล็กชนิดพกพาที่มี SD Card นี้ด้วยอินพุทต่างๆ

บทที่ 5 สรุปผลการทดลองและวิจารณ์ผลการทดลอง

บทที่ 2

ทฤษฎี

2.1 LCD (Liquid Crystal Display)

เทคโนโลยีของจอภาพแบบ LCD หรือ Liquid Crystal Display ซึ่งเป็นจอภาพที่เป็นการแสดงภาพแบบดิจิตอล (Digital) การทำงานนั้นอาศัยหลักของการใช้ความร้อนที่ได้จากขดลวด มาทำการเปลี่ยนและ บังคับให้ผลึกเหลวแสดงสีต่างๆ ออกมาตามที่ต้องการซึ่งการแสดงสีนั้นจะเป็นไปตามที่กำหนด โดยภาพที่ได้นั้นเกิดจากการปรากฏ ขึ้นจากแสงที่ปล่อยออกมาจากหลอดไฟด้านหลังของจอภาพ (Back light) และแสงนั้นก็จะผ่านชั้นกรองแสง (Polarized filter) แล้วแสงนั้นก็จะทำการผ่านต่อไปยังชั้นที่ผลึกคริสตัลเหลวที่เรียง ตัวกันเป็น 3 เซลล์ด้วยกัน นั้นคือ แสงสีแดง แสงสีเขียว และแสงสีน้ำเงิน โดยแสงที่ได้นั้นจะกลายเป็นแต่ล่ะพิกเซล (Pixel) และรวมกันจนกลายเป็นภาพที่ได้ ออกมาทางหน้าจอ โดยจอภาพแบบ LCD นั้นได้มีการพัฒนามาอย่างต่อเนื่องจนสามารถที่จะแบ่งออกได้เป็น 2 ประเภทนั้นคือ

จอภาพที่ใช้เทคโนโลยีSTN(Super-TwistedNematic)

จะมีสายไฟขึงพาดอยู่ในแนวตั้งและแนวนอน ทำหน้าที่จ่ายกระแสไฟแต่ละพิกเซลของจอเพื่อทำปฏิกิริยาแก่ตัวผลึก (Passive Matrix) ทำให้ประหยัดพลังงานซึ่งเป็นเทคโนโลยีที่ให้ความคมชัด และแสงสว่างไม่มากนักจึงทำให้นิยมนำไปใช้งานกับอุปกรณ์ประเภทเคลื่อนที่ขนาดเล็กๆ อย่างโทรศัพท์มือถือ เกมเคลื่อนที่ หรือจอภาพของ Palm ที่เป็นแบบขาวดำ ข้อเสียคือ เมื่อใช้งานกลางแดด หรือที่มีแสงจ้า ความคมชัดจะลดลง เพราะจออาศัยไฟจากตัวจ่ายไฟ ทำให้เมื่อโดนแดดพิกเซลจะซีด ลางและมองเห็นได้ไม่ชัดเจน

จอภาพที่ใช้เทคโนโลยีTFT(ThinFilmTransistor)

ตัวนี้จะแตกต่างจาก STN ตรงที่ จะมีตัวส่งสัญญาณจ่ายไฟกำกับอยู่ทุกๆช่องพิกเซล ไม่ต้องอาศัยการจ่ายไฟจากสายไฟ (Active Matrix) ทำให้ปฏิกิริยาตอบสนองไว ให้สีสันสดใสอยู่ตลอดเวลา แม้ในที่ที่มีแสงจัด แดดจ้าเป็นเทคโนโลยีที่นิยมนำมาใช้งานทั้งจอของเครื่องโน้ตบุ๊ก (Notebook) และจอภาพที่นำมาใช้งานกับเครื่องคอมพิวเตอร์ทั่วไปเป็นอย่างมากเนื่องจากว่าภาพที่ได้จากเทคโนโลยีนี้นั้นจะมีความคมชัด และแสงสว่างกว่าแบบแรกเป็นอย่างมาก

จอภาพที่ใช้เทคโนโลยีTFD(ThinFilmDiode)

เป็นหน้าจอที่พัฒนาต่อเนื่องมาจากหน้าจอแบบ STN และ TFT ซึ่งหน้าจอแบบ TFD นี้จะเอาข้อดีของหน้าจอทั้งสองแบบข้างต้นมารวมเข้าด้วยกัน และพยายามตัดข้อด้อยที่มีอยู่เดิมออกไปด้วย กล่าวคือหน้าจอแบบ TFD จะมีความสามารถในการแสดงผลที่คมชัด สีสันสดใส มีการตอบสนองสัญญาณได้รวดเร็ว เหมือนกับหน้าจอแบบ TFT แต่ใช้พลังงานน้อยเทียบเท่ากับหน้าจอแบบ STN นั่นเอง ทำให้แบตเตอรี่สามารถใช้งานได้ยาวนานขึ้นไม่ต้องมีการประจุไฟบ่อยครั้ง

2.2 คุณสมบัติของจอสี

หัวใจหลักของจอสี คือ Chip set ที่บรรจุอยู่ภายใน เนื่องจากจอแอลซีดีที่นำมาใช้งานนี้มีผู้ผลิตอยู่ด้วยกันสองบริษัท คือ Epson, Phillip ซึ่งการกำเนิดของสีต่างๆจะขึ้นอยู่กับการผสมสีของแม่สีที่สัดส่วนแตกต่างกันไปทำให้เกิดความแตกต่างของสีในหนึ่งดอท (หรือตำแหน่ง) ซึ่งในปัจจุบันได้มีเทคโนโลยีการแสดงความแตกต่างของสีที่มากมาย เช่นสามารถแสดงได้ 256สี 4,096 สี 65,536 สี 1 ล้านสี หรือ 2 ล้านสี เป็นต้น แต่ถ้าตัวเลขยิ่งมากความแตกต่างของเฉดสีก็จะมีระดับความแตกต่างที่ละเอียดมากขึ้นตามไปด้วย

2.3 การแปลงไฟล์ภาพให้เป็น Hex byte

การเปลี่ยนรูปแบบของไฟล์รูปให้เป็น Hex byte เพื่อนำไปใช้แสดงผลบนหน้าจอแอลซีดี สามารถทำได้สองวิธี

- การใช้โปรแกรมสำเร็จรูปทำหน้าที่แปลงไฟล์จากคอมพิวเตอร์
- การใช้ไมโครคอนโทรลเลอร์(CPU)ทำหน้าที่ในการแปลงไฟล์

ซึ่งการใช้ไมโครคอนโทรลเลอร์ในการอ่านไฟล์ภาพแล้วใช้ฟังก์ชันแปลงไฟล์รูปภาพให้เป็น Hex byte นั้นจะต้องใช้ไมโครคอนโทรลเลอร์ที่มีการคำนวณหรือประมวลผลค่อนข้างสูง และต้องมี RAM ภายในที่สูงเช่นกัน เนื่องจากโปรแกรมแปลงไฟล์รูปจะมีขนาดค่อนข้างใหญ่มาก ซึ่งอาจต้องต่อ RAM ภายนอกเพิ่มขึ้น ถ้าใช้ร่วมกับการ Interface จาก SD_Card

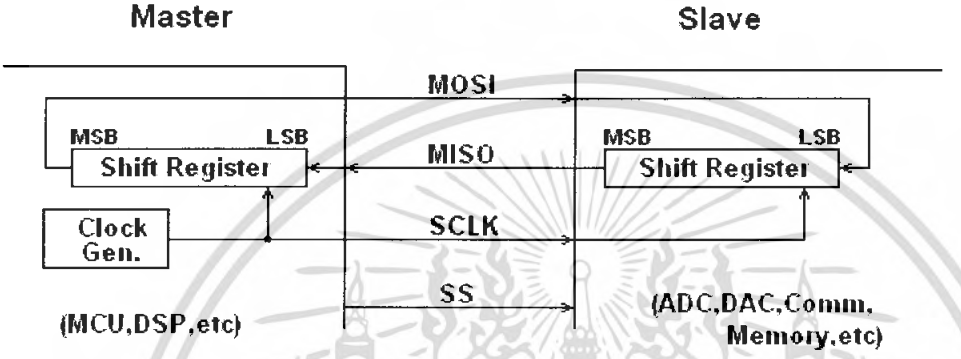
2.4 โหมดการติดต่อแบบSPI (Serial Peripheral Interface)

SPIเป็นอีกหนึ่งโหมดการติดต่อที่ใช้สำหรับSDCและMMCนอกเหนือจากNative mode ซึ่งโหมดSPI นี้จะใช้งานได้ง่ายกว่าเมื่อเทียบกับโหมดNative เนื่องจากในไมโครคอน-โทรลเลอร์ส่วนใหญ่จะมีport SPIหรือGPIOที่ไว้ใช้สำหรับการเชื่อมต่อโหมดนี้อยู่ในตัวแล้ว ดังนั้นการเชื่อมต่อแบบSPIจึงเหมาะกับงานที่มีราคาไม่สูง

2.4.1 โครงสร้างของSPI

โครงสร้างของการเชื่อมต่อแบบSPIแสดงดังรูปที่ 2.3 ตัวMasterและตัวSlave จะติดต่อกันด้วยสายสัญญาณสามเส้น คือ SCLK (Serial Clock), MISO (Master-In Slave-Out) และMOSI (Master-Out Slave-In) ส่วนสายSS (Slave Selected) ที่เพิ่มขึ้นมานั้นจะใช้เป็นสายสำหรับเลือกตัวSlaveที่จะมาติดต่อกับตัวMaster

ในโหมดSPIนี้ การส่งข้อมูลจะทำการส่งMSB (Most Significant Bit) ออกไปก่อน

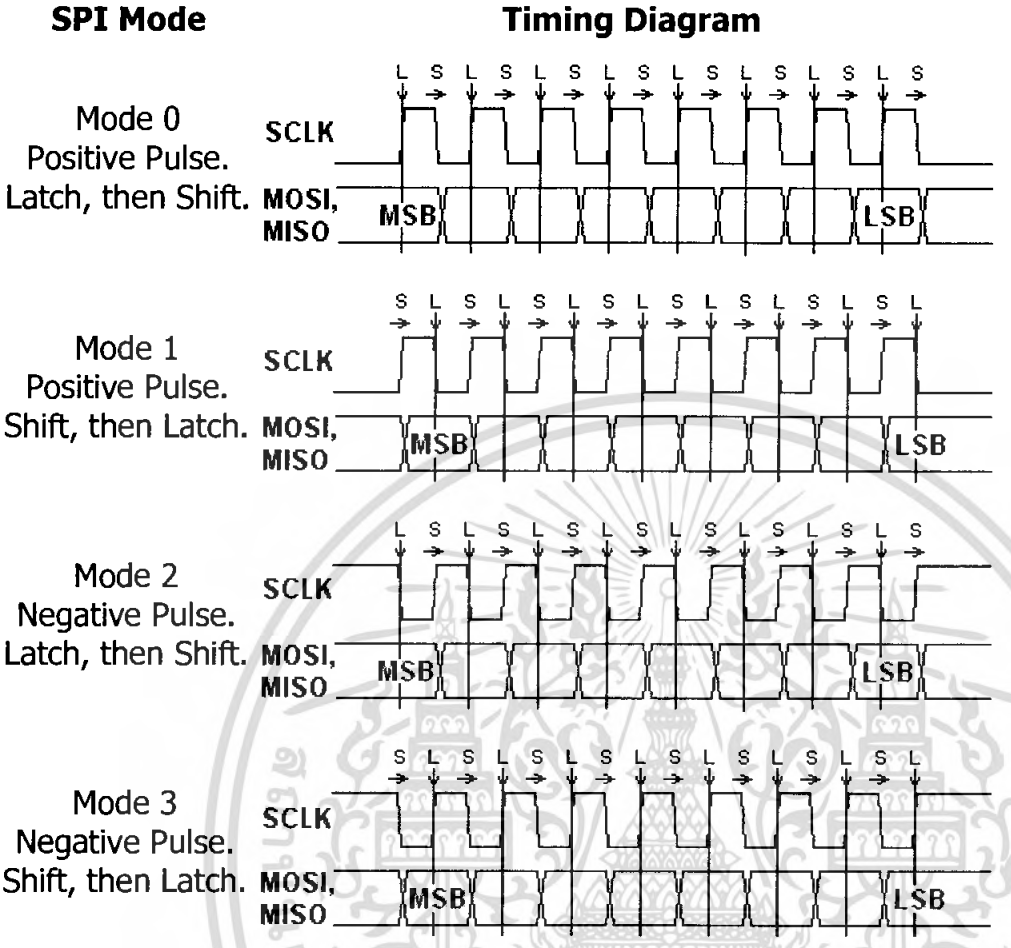


รูปที่ 2.1 โครงสร้างอย่างง่ายของการติดต่อในโหมดSPI

2.4.2 จังหวะเวลาในการถ่ายโอนข้อมูลของโหมดSPI

ในโหมด SPI การshiftและlatchของข้อมูลจะกระทำในช่วงที่ตรงกันข้ามของขอบสัญญาณclock ซึ่งจะสามารถแบ่งออกได้เป็น 4 โหมดดังรูปที่ 2.4

สำหรับ SDC เมื่อมีการติดต่อแบบSPIควรใช้ 'SPI Mode 0' แต่สำหรับMMC นั้นสามารถใช้โหมดใดก็ได้ ดังนั้นโดยทั่วไปจึงมักจะใช้ 'SPI Mode 0' (positive clock, front edge latch, back edge shift) ในการติดต่อกับSDC/MMC อย่างไรก็ตาม 'SPI Mode 3' ก็สามารถใช้งานได้ดีเช่นเดียวกัน



รูปที่ 2.2 Timing diagram ของการติดต่อแบบSPIทั้ง4โหมด

2.5 ประเภทของไฟล์ภาพ

2.5.1 BMP (Bitmap)

Bitmap เป็นภาพแบบ Resolution Dependent ประกอบขึ้นด้วยจุดสีต่างๆ ที่มีจำนวนคงที่ตายตัวตามการสร้างภาพที่มี Resolution หรือความละเอียดของภาพต่างกันไป หากขยายภาพ Bitmapจะเห็นว่ามีลักษณะเป็นตารางเล็กๆซึ่งแต่ละบิตคือส่วนหนึ่งของข้อมูลคอมพิวเตอร์

เนื่องจาก Bitmap มีค่า Pixel จำนวนคงที่จึงทำให้มีข้อจำกัดในเรื่องการขยายขนาดภาพ การเปลี่ยนขนาดภาพทำโดยเพิ่มหรือลด Pixel จากที่มีอยู่เดิม เมื่อขยายภาพให้ใหญ่ขึ้น ความละเอียดของภาพจึงลดลง และถ้าเพิ่มค่าความละเอียดมากขึ้นก็จะทำให้ไฟล์มีขนาดใหญ่และเปลืองเนื้อที่หน่วยความจำมากขึ้นตามไปด้วย ภาพที่ขยายโตขึ้นจะมองเห็นเป็นตารางสี่เหลี่ยมเรียงต่อกัน ทำให้ขาดความสวยงาม

Bitmap เป็นข้อมูลภาพที่เก็บอยู่ในหน่วยความจำซึ่งเป็นภาพที่เก็บในลักษณะ Pixel ต่อ Pixel โดยเก็บข้อมูลสีของแต่ละ Pixel เรียงกันไปในหน่วยความจำ ข้อมูลสีของแต่ละ Pixel จะมีขนาดที่ใช้ในการเก็บขึ้นอยู่กับความละเอียดสีที่ใช้ เช่นถ้าเป็น Bitmap แบบ 256 สี 1 Pixel จะมีขนาด 8 Bit (2 กำลัง 8) ดังนั้น 1 Pixel ของ Bitmap 256 สี จะมีขนาด 1 Byte โดยลักษณะของ Bitmap 256 สี ก็เป็น Array ของข้อมูลขนาด 1 Byte ในลักษณะเดียวกัน ถ้าเป็น 32 Bit ต่อ Pixel จะได้ว่า 1 Pixel มีขนาด 4 Byte ต่อ Pixel และจะเก็บได้ 2 ยกกำลัง 32 สี

เนื่องจาก Bitmap ที่เก็บอยู่ในหน่วยความจำนั้นจะเป็นข้อมูล Pixel เรียงต่อกันไปเรื่อยๆ จนหมดภาพ ดังนั้นจึงต้องมีการกำหนดความกว้างและความยาวของ Bitmap ด้วย เพื่อให้โปรแกรมส่วนที่นำ Bitmap ไปใช้ทราบว่า Bitmap มีความกว้าง – ยาว เท่าไหร่

2.5.2 JPEG (Joint Graphics Expert Group)

JPEG (Joint Photographic Experts Group - "เจเพ็ก") คือรูปแบบการบีบอัดแฟ้มภาพแบบสูญเสียโดยยังให้เสียความละเอียดน้อยที่สุด รูปแบบแฟ้มสำหรับวิธีการนี้ได้แก่ .jpeg, .jpg, .jpe, .jfif, .jfi (อาจจะเป็นตัวเล็กหรือตัวใหญ่ก็ได้)

รูปแบบแฟ้ม JPEG นี้ เป็นรูปแบบแฟ้มที่ใช้กันในการจัดเก็บและแลกเปลี่ยนรูปภาพบนเวิลด์ไวด์เว็บมากที่สุด โดยเฉพาะภาพถ่าย เนื่องจากสามารถเก็บความละเอียดสูงได้โดยใช้ขนาดไฟล์ที่เล็ก สามารถเก็บภาพสีได้หลากหลายระดับความแม่นยำของสี(Bit Depth) ความสามารถในการย่อขนาดไฟล์ของแฟ้ม JPEG นั้นเกิดจากการใช้เทคนิคการย่อขนาดภาพแบบการบีบอัดคงข้อมูลหลัก (Lossy Compression) หรือการบีบอัดแบบมีความสูญเสียทำให้ไม่นิยมใช้กับภาพที่เป็นลายเส้น หรือไอคอนต่าง ๆ เนื่องจากจะไม่ได้ประสิทธิภาพเท่าการเก็บในรูปแบบอื่น อย่าง PNG หรือ GIF การบีบอัดของ JPEG นั้นจะใช้เทคนิคที่เรียกว่า DCT (Discrete Cosine Transform) ซึ่งเป็นการแปลงค่าความสว่างของภาพให้อยู่ในรูปแบบเชิงความถี่ (Frequency Domain) ทำให้สามารถเลือกแทนค่าของสัมประสิทธิ์หรือในที่นี้คือแอมพลิจูดของค่าความถี่ต่างๆ ได้โดยอาศัยตัวแปรที่มีนัยสำคัญที่ต่างกันได้ การที่สามารถลดนัยสำคัญของค่าตัวเลขลงไปได้ทำให้สามารถลดขนาดของหน่วยความจำหรือขนาดไฟล์ที่ใช้เก็บตามไปได้

2.5.3 GIF (Graphics Interchange Format)

GIF (Graphics Interchange Format) เป็นรูปแบบแฟ้มภาพและแฟ้มภาพเคลื่อนไหว รูปแบบ GIF ถูกออกแบบโดย CompuServe ซึ่งเป็นระบบเครือข่ายข่าวสารแบบออนไลน์ เพื่อให้บริการแลกเปลี่ยนกราฟิกในรูปแบบบิตแมป ภาพแบบGIF มีข้อจำกัดอยู่ตรงด้านแผงสีแบบ Index ภาพสีแบบ 24bit (RGB) ไม่สามารถใช้ได้ แผงสีสามารถบรรจุได้ 2-256 สี ซึ่งสร้างจากข้อมูล

สี 24 บิท แฟ้มแบบ GIF โดยในการบีบอัดของ GIF จะเป็นประเภทไม่สูญเสีย (lossless compression) GIF จะใช้ได้ดีกับภาพกราฟฟิก สัญลักษณ์ต่างๆ หรือภาพอนิเมชันเคลื่อนไหวง่ายๆ ที่มีสีไม่ต่อเนื่องและมีจำนวนสีไม่มากนัก

Pixel เป็นหน่วยพื้นฐานของสีในระบบโปรแกรมบนจอภาพหรือภาพ ซึ่งหน่วยที่มีลักษณะเป็นหน่วยทางตรรกะมากกว่ากายภาคขนาดของ Pixel ขึ้นกับการกำหนดความละเอียด (Resolution) ของจอภาพ ถ้าตั้งค่าความละเอียดสูงสุดขนาดของ Pixel จะทำกับขนาดทางกายภาพของ dotpitch (ขนาดของจุด) ของจอภาพ การกำหนดสีของ pixel ใช้การกำหนดผสมของสเปคตรัม RGB ข้อมูลขอสีสามารถคำนวณไบต์ได้ถึง 3 ไบต์ ซึ่ง I สำหรับแต่ละสี true Color หรือระบบสี 24 บิต จะใช้จำนวนไบต์ทั้ง 3 ไบต์ อย่างไรก็ตามระบบสีส่วนใหญ่ใช้ 8 บิต ซึ่งไฟล์สีได้ 256 สี bitmap เป็นไฟล์ที่ใช้ชี้สีในแต่ละ Pixel ตามแกนนอนหรือแถว และสีสำหรับแต่ละ Pixel ในแกนตั้ง เช่น ไฟล์ GIF (Graphics Interchange Format) เก็บ bitmap ของภาพ ความคมชัดภาพบนจอภาพในบางครั้งแสดงในรูปของจุดต่อไว้ (dots per inch) จำนวนจุดต่อนิ้วจะหาได้โดยขนาดทางกายภาพของจอ และการตั้งค่าความละเอียด ถ้าตั้งค่าความละเอียดไว้ต่ำ ทำให้จุดต่อนิ้วต่ำด้วย ซึ่งจอภาพที่ใหญ่กว่าแต่มีค่าความละเอียดเท่ากัน จะทำให้ความคมชัดลดลง

RGB (red, green and blue) อ้างถึงระบบ สำหรับนำเสนอสีที่ใช้บนจอภาพคอมพิวเตอร์ red, green และ blue สามารถรวมในสัดส่วนต่าง ๆ เพื่อทำให้เป็นสีต่างๆ ภายในช่วงที่มองเห็น ระดับของ R, G และ B มีช่วงตั้งแต่ 0 ถึง 100 เปอร์เซ็นต์ ของความหนาแน่นเต็มที่ แต่ละระดับแสดง โดยช่วงของเลขฐานสิบจาก 0 ถึง 255 (256 ระดับของแต่ละสี) เทียบเท่ากับช่วงของเลขฐานสองจาก 00000000 ถึง 11111111 หรือ เลขฐานหก 00 ถึง FF จำนวนสีที่มีให้ทั้งหมดคือ 256*256*256 หรือ 16,777,216 สี ในภาษา Hypertext Markup Language (HTML) จำนวนค่าของ RGB ได้แนะนำการใช้โดยพิจารณาให้ลด ประการแรก โดยข้อเท็จจริงภาพส่วนมากสามารถใช้สีได้ 256 สี เพื่อที่จะทำให้สีที่มีอยู่เป็นจริงบน browser ทั้ง 2 ประเภท ให้เลือกใช้ร่วมกัน 216 สี สีอื่นนอกจากนี้ให้ทำการแปลงเป็นสีใกล้เคียง

2.6 ตารางการจัดเรียงไฟล์ (File Allocation Table: FAT)

ระบบปฏิบัติการของคอมพิวเตอร์แต่ละแพล็ตฟอร์ม (Platform) จะมีการจัดการระบบไฟล์ในฮาร์ดดิสก์ที่แตกต่างกัน บางระบบสามารถใช้ระบบไฟล์ได้หลายรูปแบบ โดยระบบไฟล์นั้นเป็นตารางที่ใช้บอกตำแหน่งของข้อมูลต่างๆที่อยู่บนฮาร์ดดิสก์ว่าอะไรอยู่ตรงไหน ปกติเมื่อซื้อฮาร์ดดิสก์มาใหม่ต้องทำการจัดข้อมูล (Format) ให้ฮาร์ดดิสก์ก่อนที่จะนำไปบรรจุข้อมูลการจัด

ข้อมูลในฮาร์ดดิสก์เป็นการแบ่งฮาร์ดดิสก์ออกเป็นส่วนๆ เพื่อให้คอมพิวเตอร์รู้ว่าตำแหน่งของข้อมูลอยู่ตรงไหน

FATเป็นระบบไฟล์ที่ใช้ในระบบปฏิบัติการในตระกูลMicrosoftและเป็นระบบไฟล์ที่มีพัฒนาการมาอย่างต่อเนื่อง ระบบไฟล์ในตระกูลนี้มีลักษณะคือ เป็นการกำหนดหมายเลขให้กับทุกๆคลัสเตอร์ (Cluster) ในแต่ละส่วนแบ่งของฮาร์ดดิสก์ (Partition) แล้วทำการสร้างตารางที่มีจำนวนช่องตามจำนวนคลัสเตอร์ เพื่อเป็นการระบุสถานที่หรือคลัสเตอร์ที่ทำการเก็บข้อมูลของไฟล์แต่ละไฟล์ และมีตารางอีกตารางหนึ่งที่เรียกว่าไดเรคทอรี่ (Directory) สำหรับเก็บข้อมูลรายละเอียดของไฟล์ เช่น คุณลักษณะ (Attribute) ต่าง ๆ และ หมายเลข Cluster เริ่มต้นที่เก็บตัวข้อมูลจริง ๆ

ระบบจัดการไฟล์แบบ FAT เป็นระบบที่ไม่ยุ่งยากซับซ้อน ดังนั้นจึงถูกรองรับจากทุกระบบปฏิบัติการที่มีอยู่สำหรับคอมพิวเตอร์ส่วนบุคคล และยังสะดวกในการแลกเปลี่ยนข้อมูลระหว่างระบบปฏิบัติการที่แตกต่างกันซึ่งถูกติดตั้งบนคอมพิวเตอร์เครื่องเดียวกันข้อด้อยของระบบจัดการไฟล์แบบ FAT คือ เมื่อไฟล์ถูกลบและไฟล์ใหม่ถูกเขียนลงไป แฟรกเมนต์ (fragment) ของแต่ละไฟล์จะมีโอกาสกระจัดกระจายออกไปอยู่ทั่วทั้งหน่วยความจำ ส่งผลให้การอ่านและการเขียนไฟล์ทำได้ช้า การจัดเรียงข้อมูลเป็นหนึ่งในวิธีการทำให้การจัดเรียงไฟล์แบบ FATเป็นระเบียบ แต่จะต้องทำการจัดเรียงข้อมูลอยู่บ่อยๆ และเป็นกระบวนการที่ใช้เวลามาก

ระบบไฟล์ FAT มีหลายรุ่นดังต่อไปนี้

2.6.1 FAT12

ระบบ FAT12 เป็นรุ่นที่เก่าแก่ที่สุดของตระกูล FAT ใช้กับระบบปฏิบัติการ DOS ซึ่งใช้ 12 บิตในการอ้างอิงถึงหมายเลขคลัสเตอร์ เพราะฉะนั้นสามารถอ้างถึงคลัสเตอร์ได้มากที่สุด 4086 คลัสเตอร์ (ประมาณ 2 ยกกำลัง 12 = 4096 และมีเนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัว FAT เอง) ระบบ FAT12จึงเหมาะกับหน่วยความจำที่มีเนื้อที่ไม่มาก เช่น Floppy Disk หรือดิสก์ขนาดเล็น3585. ที่มีเนื้อที่ไม่เกิน 16 เมกกะไบต์ และระบบ FAT12 นี้สามารถใช้งานกับไฟล์ที่มีชื่อยาวเพียง 8.3 ตัวอักษรเท่านั้น

2.6.2 FAT16

สามารถใช้งานร่วมกับระบบปฏิบัติการที่หลากหลายได้ เช่น DOS, Windows 95, Windows 98,Windows ME, OS/2, Linux ซึ่งใช้ 16 บิต เพื่ออ้างถึงหมายเลขคลัสเตอร์ เพราะฉะนั้นสามารถอ้างถึงได้ทั้งหมด 65526 คลัสเตอร์ (ประมาณ 2 ยกกำลัง 16 = 65536 และมี

เนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัวFAT เอง) FAT16 นั้นถูกออกแบบมาเพื่อใช้งานกับไฟล์ต่างๆบนดิสก์ขนาดเล็กหรือขนาดกลาง ซึ่งมีเนื้อที่ประมาณ 2048 เมกะไบต์

ปัญหาของ FAT16 คือ

1. ระบบ FAT16 ใช้งานพื้นที่ของหน่วยความจำสิ้นเปลืองมาก เพราะจำนวนสูงสุดของคลัสเตอร์ต่อพาร์ติชัน (Maximum number of cluster per partition) นั้นถูกกำหนดเอาไว้ตายตัว (65526 คลัสเตอร์) ดังนั้นเมื่อหน่วยความจำมีขนาดที่ใหญ่ขึ้น แต่ว่าจำนวนคลัสเตอร์ยังเท่าเดิม ก็หมายความว่าขนาดของคลัสเตอร์ก็จะใหญ่ขึ้นตามไปด้วย อย่างในกรณีของหน่วยความจำขนาด 2 กิกะไบต์ นั้นจะมีคลัสเตอร์ที่ใหญ่ถึง 32 กิโลไบต์ นั่นก็หมายความว่า ต่อให้เราพิมพ์เอกสารที่มีตัวอักษรเพียง 10 ตัว แต่ขนาดของไฟล์ก็จะมีถึง 32 กิโลไบต์ ซึ่งเป็นขนาดเล็กสุดของคลัสเตอร์เลยทีเดียว

2. ขีดจำกัดเรื่องขนาดสูงสุดของหน่วยความจำที่รองรับได้ เพราะเริ่มต้น FAT16 ถูกออกแบบมาเพื่อจะใช้งานกับดิสก์ที่มีขนาดเล็ก ดังนั้นในยุคแรกๆจึงมีปัญหาตามมาเมื่อระบบ FAT16 ใน MSDOSยุคแรก สามารถรองรับหน่วยความจำได้เพียง 32 เมกะไบต์ เท่านั้น แต่ต่อมาถูกแก้ไขให้รองรับได้เป็น 128 เมกะไบต์ ใน MS-DOS 4.0 และเรื่อยมาเป็น 2 กิกะไบต์ในปัจจุบัน

3. ระบบ FAT16 สามารถใช้งานกับไฟล์ที่มีชื่อยาวเพียง 8.3 ตัวอักษรเท่านั้นเช่นเดียวกับ FAT12 แต่ได้รับการปรับปรุงให้มีความสามารถมากขึ้นใน Windows 95 เพื่อให้สามารถใช้งานกับไฟล์ที่มีชื่อยาวได้ไม่เกิน 256 ตัวอักษร เรียก FAT16 รุ่นนี้ว่า Virtual FAT หรือ VFAT

2.6.3 FAT32

ระบบ FAT32 ใช้กับระบบปฏิบัติการ Windows 95/98, Windows ME, Windows 2000 โดยทำการแก้ไขเพิ่มเติมจาก FAT16 เพื่อที่จะได้มีจำนวนคลัสเตอร์ต่อพาร์ติชันมากขึ้น ดังนั้นเลยมีความสามารถที่จะรองรับหน่วยความจำที่มีขนาดใหญ่สูงสุดได้ถึง 2 เทระไบต์ (2000 กิกะไบต์) ซึ่งจะใช้28 บิต (อีก 4 บิต สำรองเอาไว้) ฉะนั้นสามารถอ้างถึงคลัสเตอร์ได้ทั้งหมด 286 ล้านคลัสเตอร์ (ประมาณ2 ยกกำลัง 28 และมีเนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัว FAT เอง) และระบบ FAT32 ยังสามารถรองรับชื่อไฟล์แบบยาว (Long File Name : LFN) คือ 255 ตัวอักษรได้อีกด้วย

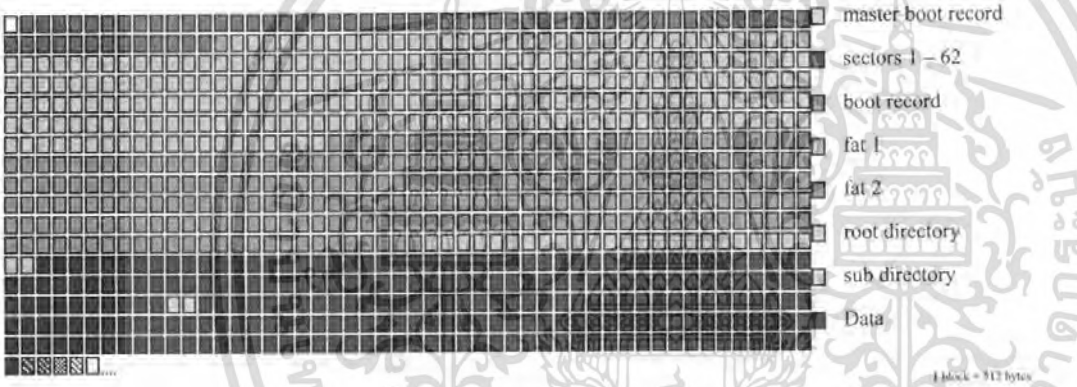
ตารางที่ 2.1 เปรียบเทียบระหว่าง FAT12, FAT16 และ FAT32

	FAT12	**FAT16**	**FAT32**
Developer	Microsoft		
Full Name	File Allocation Table		
	(12-bit version)	(16-bit version)	(32-bit version)
Introduce	1977 (Microsoft Disk BASIC)	July 1988 (MS-DOS 4.0)	August 1996 (Window 95 OSR2)
Structures			
Directory contents	Table		
File allocation	Linked List		
Bad blocks	Linked List		
Limits			
Max file size	32 MB	2 GB	4 GB
Max number of files	4,077	65,517	268,435,437
Max file name size	8.3 or 255 when using LFNs		
Max volume size	32 MB	2 GB 4 GB with some implementation	2 TB
Features			
Dates recorded	Creation, modified, access		
Date range	January 1, 1980 – December 31, 2107		
Forks	Not natively		
Attributes	Read-only, hidden, system, volume label, subdirectory, archive		

Permissions	No	
Transparent compression	Per-volume, stacker, DoubleSpace, DriveSpace	No
Transparent encryption	Per-volume only with DR-DOS	No

2.6.4 โครงสร้างของดิสก์หลัก (Main disk structures)

ระบบไฟล์แบบ FAT ประกอบไปด้วย 4 ส่วนที่แตกต่างกัน ดังนี้ คือ 1). Reserved sectors, 2) FAT Region, 3) Root Directory Region, 4) Data Region ซึ่งสามารถแสดงโครงสร้างของดิสก์ได้ดังรูปที่ 2.1



รูปที่ 2.3 โครงสร้างของดิสก์

1. **เซ็คเตอร์สำรอง (Reserved sectors)** อยู่ในส่วนต้นๆ โดยเซคเตอร์สำรองแรกสุดเป็นบูทเซคเตอร์ (Boot Sector) ซึ่งรวมพื้นที่ที่เรียกว่า BIOS Parameter Block โดยปกติจะประกอบด้วยโค้ดเริ่มต้นของระบบปฏิบัติการ ซึ่งจำนวนของเซคเตอร์สำรองทั้งหมดจะถูกระบุอยู่ในส่วนของนี้ ข้อมูลสำคัญต่างๆจากบูทเซคเตอร์สามารถเข้าถึงผ่านโครงสร้างระบบปฏิบัติการ ที่เรียกว่า Drive Parameter Block ใน DOS และ OS/2

2. **บริเวณตารางจัดเรียงไฟล์ (FAT Region)** ส่วนนี้ประกอบไปด้วยตารางการจัดเรียงไฟล์ 2 ชุด ชุดหนึ่งสำหรับใช้งานจริงและอีกชุดเป็นการสำรองไว้ใช้ในเวลาจำเป็น โดยบริเวณตารางจัดเรียงไฟล์นี้จะสอดคล้องกับบริเวณข้อมูล (Data Region) ทำหน้าที่ระบุตำแหน่งคลัสเตอร์ของไฟล์และไดเร็คทอรี่ต่างๆ

3. **บริเวณไดเร็คทอรี่ (Root Directory Region)** นี่คือตารางไดเรคทอรี่ซึ่งเก็บข้อมูล

เกี่ยวกับไฟล์และไดเรคทอรี่ต่างๆ ในระบบจัดการไฟล์แบบ FAT12 และ FAT16 ตารางไดเรคทอรี่นี้จะอยู่ในบริเวณไดเรคทอรี่เท่านั้น และมีขนาดสูงสุดที่แน่นอน แต่ในระบบจัดการไฟล์แบบ FAT32 ตารางไดเรคทอรี่จะอยู่รวมกับไฟล์ต่างๆในบริเวณข้อมูล ซึ่งสามารถขยายขนาดออกไปได้ไม่จำกัด

4. **บริเวณข้อมูล (Data Region)** เป็นส่วนที่มีไฟล์ข้อมูลอยู่จริง ขนาดของไฟล์และไดเร็คทอรี่ย่อยสามารถเพิ่มขึ้นได้เท่าที่มีคลัสเตอร์เหลืออยู่

2.6.4.1 บูทเซ็คเตอร์และโครงสร้างBPB (Bios Parameter Block)

ตารางที่ 2.2 โครงสร้างพื้นฐานของบูทเซ็คเตอร์และBPBซึ่งใช้ในทุกเวอร์ชั่นของFAT

ชื่อ	ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด
BS_jmpBoot	0x00	3	คำสั่งกระโดด(เพื่อที่จะข้ามส่วนหัวของการบูท)
BS_OEMName	0x03	8	ชื่อ OEM ค่าพื้นฐานคือ IBM 3.3 และ MS-DOS 5.0
BPB_BytsPerSec	0x0B	2	จำนวนไบต์ต่อหนึ่งเซ็คเตอร์
BPB_SecPerClus	0x0D	1	จำนวนเซ็คเตอร์ต่อหนึ่งคลัสเตอร์
BPB_RsvdSecCnt	0x0E	2	จำนวนของเซ็คเตอร์สำรองในบริเวณReserved
BPB_NumFATs	0x10	1	จำนวนของตารางการจัดเรียงข้อมูล
BPB_RootEntCnt	0x11	2	จำนวนของไดเร็คทอรี่ในบริเวณ Root directory สำหรับFAT32ค่านี้จะเป็นศูนย์เนื่องจากบริเวณ Root directoryของFAT32จะอยู่รวมกับไฟล์ต่างๆ ในบริเวณข้อมูล
BPB_TotSec16	0x13	2	จำนวนเซ็คเตอร์รวมทั้งหมดของบริเวณทั้งสี่ สำหรับ FAT32ค่านี้จะเป็นศูนย์
BPB_Media	0x15	1	ตัวบรรยายมีเดีย 0xF8 ด้านเดียว, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xF9 สองด้าน, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อ

			แทรก 0xFA ด้านเดียว, 80 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFB สองด้าน, 80 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFC ด้านเดียว, 40 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xFD สองด้าน, 40 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xFE ด้านเดียว, 40 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFF สองด้าน, 40 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก
BPB_FATSz16	0x16	2	จำนวนฌซ็คเตอร์ต่อหนึ่งตารางการจัดเรียงข้อมูล
BPB_SecPerTrk	0x18	2	จำนวนเซ็คเตอร์ต่อหนึ่งTrack
BPB_NumHeads	0x1A	2	จำนวนของเฮดสำหรับการอินเทอรัป0x13
BPB_HiddSec	0x1C	4	จำนวนเซ็คเตอร์ที่ถูกซ่อน
BPB_TotSec32	0x20	4	จำนวนเซ็คเตอร์รวมทั้งหมดของบริเวณทั้งสี่ สำหรับ FAT16ค่านี้จะเป็นศูนย์

ตารางที่ 2.3 โครงสร้างของบูทเซ็คเตอร์และBPBที่เพิ่มขึ้นมาในFAT32

ชื่อ	ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด
BPB_FATSz32	0x24	4	จำนวนเซ็คเตอร์ต่อหนึ่งตารางการจัดเรียงข้อมูล
BPB_ExtFlags	0x28	2	เครื่องหมายตารางการจัดเรียงข้อมูล
BPB_FSVer	0x2A	2	เวอร์ชั่น
BPB_RootClus	0x2C	4	หมายเลขของคลัสเตอร์แรกในRoot directory
BPB_FSInfo	0x30	2	หมายเลขเซ็คเตอร์ของเซ็คเตอร์ข้อมูล FS

BPB_BkBootSec	0x32	2	หมายเลขเซ็คเตอร์ของสำเนาบูทเซ็คเตอร์
BPB_Reserved	0x34	12	ส่วนสำรอง
BS_DrvNum	0x40	1	หมายเลขฟิสิคอลไดรฟ์
BS_Reserved1	0x41	1	ส่วนสำรอง
BS_BootSig	0x42	1	ซิกเนเจอร์ (Signature)
BS_VolID	0x43	4	หมายเลขซีเรียล
BS_VolLab	0x47	11	โวลุมลาเบล (Volume Label)
BS_FilSysType	0x52	8	ชนิดของระบบไฟล์ ต้องเป็นFAT32เสมอ
	0x5A	420	โค้ดบูทระบบปฏิบัติการ
	0x1FE	2	เซ็คเตอร์สุดท้าย (0x55 , 0xAA)

2.6.4.2 ตารางไดเรคทอรี่

ตารางไดเรคทอรี่เป็นไฟล์ชนิดพิเศษที่แสดงไดเรคทอรี่ (ปัจจุบันรู้จักกันในนามของโฟลเดอร์) แต่ละไฟล์หรือไดเรคทอรี่ประกอบไปด้วย 32 ไบต์ แต่ละไบต์จะบันทึกชื่อไฟล์, นามสกุลไฟล์, คุณลักษณะของไฟล์ (ชนิดเอกสารสำคัญ, ไดเรคทอรี่, ถูกซ่อน, อ่านได้อย่างเดียว, ระบบ, และความจุ), วัน เวลาที่ไฟล์ถูกสร้าง, แอดเดรสของคลัสเตอร์แรกของไฟล์หรือไดเรคทอรี่นั้น และสุดท้ายคือขนาดของไฟล์หรือไดเรคทอรี่
จำนวนไดเรคทอรี่ทั้งหมดทั้งในบริเวณไดเรคทอรี่และในไดเรคทอรี่ย่อย มีรูปแบบ ดังนี้

ตารางที่ 2.4 โครงสร้างของตารางไดเรคทอรี่

ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด		
0x00	8	0x00		
0x08	3	เครื่องหมายของตารางจัดเรียงข้อมูล		
0x0B	1	บิต	Mask	รายละเอียด
		0	0x01	อ่านได้อย่างเดียว
		1	0x02	ถูกซ่อน
		2	0x04	ระบบ

		3	0x08	โวลุมลาเบล
		4	0x10	ไดเร็คทอรี่ย่อย
		5	0x20	เอกสารสำคัญ
		6	0x40	อุปกรณ์
		7	0x80	ไม่ใช้
0x0C	1	สำรอง ถูกใช้โดยNT		
0x0D	1	เวลาสร้างไฟล์, ความละเอียด 10 มิลลิวินาที ค่าอยู่ที่ระหว่าง 0-199		
0x0E	2	**บิต**	**รายละเอียด**	
		15-11	ชั่วโมง (0-23)	
		10-5	นาที (0-59)	
		4-0	วินาที/2 (0-29)	
0x10	2	**บิต**	**รายละเอียด**	
		15-9	ปี (0=1980, 127=2107)	
		8-5	เดือน (1=มกราคม, 12=ธันวาคม)	
		4-0	วันที่ (1-31)	
0x12	2	วันที่ใช้งานไฟล์ล่าสุด ดูไบต์ออฟเซ็ต 0x01 ประกอบ		
0x14	2	ดัชนี EA (ถูกใช้โดย OS/2 และ NT) ในFAT12และFAT16 หรือ 2 ไบต์สูงของคลัสเตอร์แรกใน FAT32		
0x16	2	เวลาที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซ็ต 0x0E ประกอบ		
0x18	2	วันที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซ็ต 0x10 ประกอบ		
0x1A	2	คลัสเตอร์แรกใน FAT12 และ FAT16, หรือ 2ไบต์ต่ำของคลัสเตอร์แรกในFAT32		
0x1C	4	ขนาดไฟล์		

2.6.4.3 ตาราง FAT

ขนาดของแต่ละคลัสเตอร์ขึ้นอยู่กับชนิดของระบบไฟล์แบบ FAT ที่ใช้และขนาดของพาร์ติชันโดยปกติขนาดของคลัสเตอร์จะอยู่ระหว่าง 2 กิโลไบต์ถึง 32 กิโลไบต์ แต่ละไฟล์อาจจะกินเนื้อที่มากกว่า 1คลัสเตอร์ขึ้นอยู่กับขนาดของไฟล์นั้น แต่ไม่จำเป็นต้องเป็นคลัสเตอร์ที่ติดกัน

ตารางการจัดเรียงข้อมูลเป็นสมุดรายชื่อของไฟล์ที่สัมพันธ์กับตำแหน่งคลัสเตอร์ของไฟล์ แต่ละคลัสเตอร์ใน FAT จะบันทึกข้อมูล 1 ใน 5 ดังนี้

- แอดเดรสของคลัสเตอร์ถัดไปของไฟล์
- สถานะแสดงจุดสิ้นสุดของไฟล์
- สถานะแสดงว่าเป็นคลัสเตอร์เสีย (bad cluster)
- สถานะแสดงว่าเป็นคลัสเตอร์ที่ถูกสำรองไว้
- ศูนย์เพื่อแสดงว่าเป็นคลัสเตอร์ที่ไม่ถูกใช้

แต่ละเวอร์ชันของระบบไฟล์แบบตารางการจัดเรียงข้อมูล จะมีขนาดของคลัสเตอร์ในตารางการจัดเรียงข้อมูลที่ต่างกัน ซึ่งขนาดจะถูกระบุโดยชื่อของแต่ละเวอร์ชัน เช่น ระบบไฟล์ FAT16 คลัสเตอร์จะมีขนาด 16 บิต ในขณะที่ FAT32 จะใช้ 32 บิต ตามขนาดของพาร์ติชันที่ใหญ่ขึ้น ซึ่ง FAT32 จะมีประสิทธิภาพมากกว่าในกรณี FAT16 เนื่องจาก FAT32 สามารถแบ่งคลัสเตอร์ให้มีขนาดที่เล็กกว่าซึ่งหมายความว่าจะมีพื้นที่สูญเปล่าน้อยกว่าในกรณี FAT16

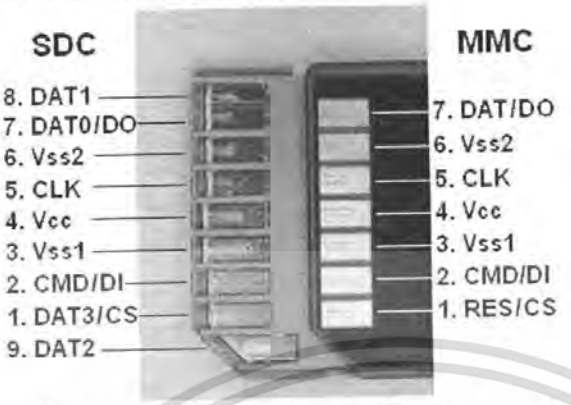
ตารางที่ 2.5 ค่าต่างๆในตารางการจัดเรียงข้อมูล

FAT12	FAT16	FAT32	รายละเอียด
0x000	0x0000	0x?0000000	คลัสเตอร์ว่าง
0x001	0x0001	0x?0000001	คลัสเตอร์สำรอง
0x002-0xFEF	0x0002-0Xffef	0x?0000002- 0x?FFFFFEF	คลัสเตอร์ที่ถูกใช้งาน, ค่าของคลัสเตอร์ถัดไป
0xFF0-0xFF6	0xFFF0- 0xFFF6	0x?FFFFFF0- 0x?FFFFFF6	ค่าสำรอง
0xFF7	0xFFF7	0x?FFFFFF7	คลัสเตอร์เสีย
0xFF8-0xFFF	0xFFF8- 0xFFFF	0x?FFFFFF8- 0x?FFFFFFF	คลัสเตอร์สุดท้ายของไฟล์

สังเกตว่า FAT32 จะใช้เพียง 28 บิต จาก 32 บิตที่สามารถใช้ได้ โดยปกติ 4 บิตบนจะมีค่าเป็นศูนย์

2.7 SD Card และ MMC Card

2.7.1 หน้าสัมผัสสำหรับการเชื่อมต่อของการ์ด



รูปที่ 2.4 หน้าสัมผัสสำหรับการเชื่อมต่อของการ์ด

รูปที่ 2.4 แสดงหน้าสัมผัสสำหรับการเชื่อมต่อของ SDC (Secure Digital Memory Card) และ MMC (Multi Media Card) โดยMMCจะมี7ขา ส่วนSDCจะมี9ขา สามขาเป็นขาของpower supply คือ ขา4 และขา6 ขอบเขตของแรงดันที่ใช้ในการทำงานของการ์ดจะถูกกำหนดโดยรีจิสเตอร์OCRดังตารางที่2.6 แต่อย่างไรก็ตามคุณสมบัติโดยทั่วไปของSDCและMMC จะสามารถทำงานได้ที่แรงดัน 2.7 – 3.6 V.

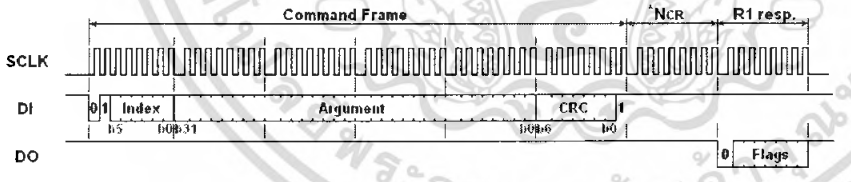
ตารางที่ 2.6 OCR Register Definitions

OCR Bit	VDD Voltage Window
0-7	Reserved
8	2.0-2.1
9	2.1-2.2
10	2.2-2.3
11	2.3-2.4
12	2.4-2.5
13	2.5-2.6
14	2.6-2.7
15	2.7-2.8
16	2.8-2.9
17	2.9-3.0

18	3.0-3.1
19	3.1-3.2
20	3.2-3.3
21	3.3-3.4
22	3.4-3.5
23	3.5-3.6
24-30	Reserved
31	Card power up status bit (Busy)

2.7.2 คำสั่งและการตอบกลับ

ในโหมด SPI นั้น โครงสร้างของคำสั่งจะถูกกำหนดไว้ให้มีความยาว 6 Byte ดังแสดงในรูปที่ 2.6 เมื่อชุดคำสั่งถูกส่งไปที่การ์ด จะมีการตอบกลับจากการ์ดไปที่ Host ในรูปแบบ R1, R2 หรือ R3 เนื่องจากการถ่ายโอนข้อมูลนั้นจะถูกขับโดย Clock แบบอนุกรมที่ Host สร้างขึ้น ดังนั้น Host จะต้องสร้าง Clockต่อไปเรื่อยๆ จนกว่าจะได้รับการตอบกลับจากการ์ด ช่วงระยะเวลาก่อนที่การ์ดจะตอบกลับหลังจากได้รับคำสั่งจาก Host (NCR) คือ 0 ถึง 8 Byte สำหรับ SDC และ 1 ถึง 8 Byte สำหรับ MMC ในช่วงที่มีการโอนถ่ายข้อมูลนี้ ขา SS จะต้องถูกกำหนดให้มีสถานะลอจิกเป็น 0 เสมอ



รูปที่ 2.5 Command Frame ที่ส่งจาก Host ไปหาการ์ด

2.7.3 ชุดคำสั่งที่ใช้ในโหมด SPI

คำสั่นนี้เป็นเพียงคำสั่งโดยทั่วไปที่ใช้สำหรับการอ่าน เขียนและ Initial Card เท่านั้น

ตารางที่ 2.7 แสดงชุดคำสั่งของโหมด SPI

Command Index	Argument	Response	Data	Abbreviation	Description

CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55	None(0)	R1	No	APP_CMD	Application

					specific command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.
*1: ACMD <n> means a command sequence of CMD55-CMD <n>.					

2.7.4 Command Response ในโหมดSPI

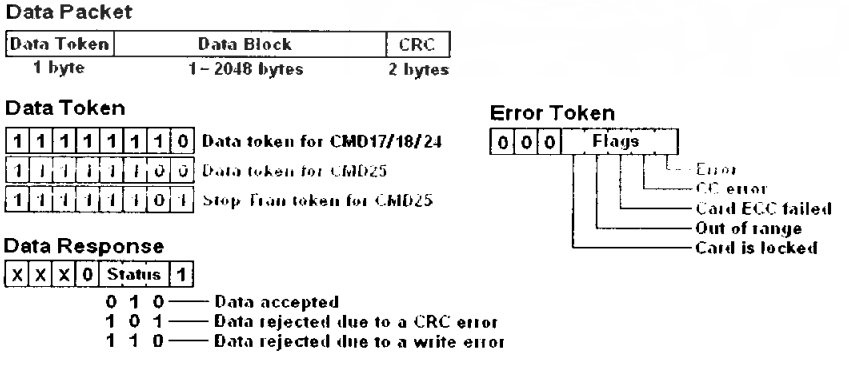
การตอบกลับในโหมด SPI จะมีอยู่สามรูปแบบ คือ R1, R2 และ R3 ดังรูปที่ 2.7 ขึ้นอยู่กับคำสั่งที่Host ส่งมา แต่คำสั่งส่วนมากจะมีการตอบกลับในรูปแบบ R1 ถ้าหาก R1 มีค่าเป็น 0x00 แสดงว่าไม่มีข้อผิดพลาดเกิดขึ้น ส่วนการตอบสนองแบบ R3 นั้นจะตอบกลับเฉพาะ CMD58 เท่านั้น

รูปที่ 2.6 การตอบกลับแบบต่างๆ

2.8 การถ่ายโอนข้อมูล

2.8.1 Data Packet และ Data response

ในการถ่ายโอนข้อมูล จะมีการถ่ายโอนหลังจากมีการส่ง Command Response แล้ว บล็อกข้อมูลจะถูกถ่ายโอนในรูป Data Packet ซึ่งประกอบไปด้วย Data Token, Data Block และ CRC ดังแสดงในรูปที่ 2.8 จะเห็นได้ว่า Data Token จะมีอยู่สามแบบ ซึ่งขึ้นอยู่กับแต่ละคำสั่งที่ส่ง



รูปที่ 2.7 โครงสร้างของData Packet

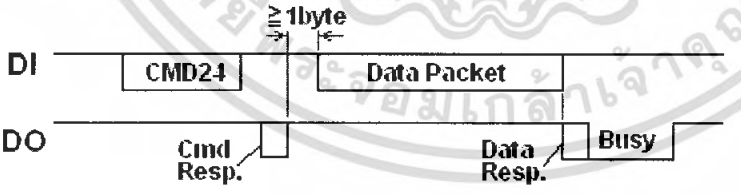
2.8.2 คำสั่ง Single Block Read

Argument ที่ต่อท้ายคำสั่งนี้จะเป็นตัวกำหนดแอดเดรสเริ่มต้นของข้อมูลที่จะอ่าน เมื่อคำสั่งนี้ได้รับการตอบกลับ กระบวนการอ่านข้อมูลก็จะเริ่มขึ้น โดยชุดข้อมูลที่ต้องการอ่านนั้นก็จะถูกส่งไปยัง Host เมื่อ Host ตรวจพบ Data Token ที่ส่งมา Host ก็จะทำการอ่านบล็อกข้อมูลที่ตามหลัง Token นั้น เมื่อมีความผิดพลาดเกิดขึ้นในการทำงาน Token ที่แสดงความผิดพลาดจะถูกส่งมาแทนที่บล็อกข้อมูล

2.8.3 คำสั่ง Multiple Block Read

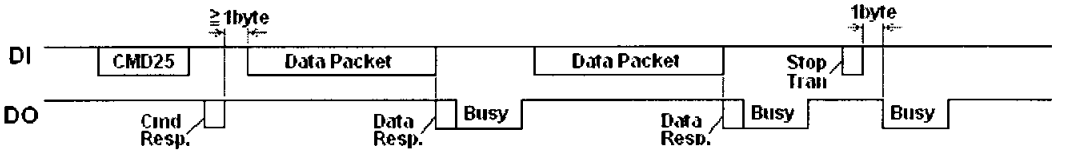
คำสั่งนี้เป็นการอ่านข้อมูลครั้งละหลายๆ บล็อกจากตำแหน่งแอดเดรสที่กำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการอ่านต่อหนึ่งครั้งไว้ กระบวนการอ่านนี้จะดำเนินต่อเนื่องไปเรื่อยๆ จนกว่าจะมีการส่งคำสั่ง CMD12 ไปที่การ์ด การอ่านข้อมูลนี้จึงจะหยุดลง

2.8.4 คำสั่ง Single Block Write



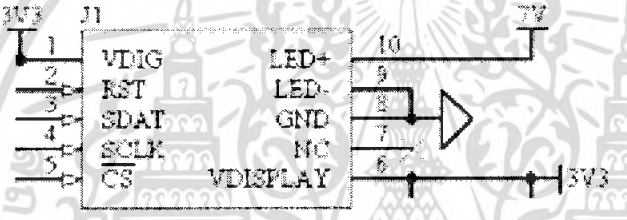
เมื่อคำสั่งนี้ได้รับการตอบรับ Host ก็จะทำการส่งแพ็คเกจข้อมูลไปที่การ์ดหลังจากนั้นเป็นช่วงประมาณ 1 Byte แพ็คเกจของข้อมูลนี้ก็จะมีลักษณะเหมือนกับแพ็คเกจของข้อมูลในคำสั่ง Read เมื่อข้อมูลถูกส่งไปที่การ์ดแล้ว การ์ดก็จะตอบกลับมาด้วย Data Response ในทันที การ์ดโดยส่วนใหญ่จะสามารถเขียนข้อมูลได้บล็อกละ 512 Byte

2.8.5 คำสั่ง Multiple Block Write



คำสั่งนี้จะเขียนข้อมูลครั้งละหลายๆ บล็อกลงไปในแอดเดรสที่เรากำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการถ่ายโอน การถ่ายโอนนี้ก็จะดำเนินต่อไปเรื่อยๆ จนกว่าจะถูกหยุดโดย Stop Tran Token ซึ่งก็จะทำให้เกิด Busy Flag ขึ้นเป็นช่วงระยะ 1 Byte ต่อจาก Stop Tran Token

2.9 องค์ประกอบของ LCD Nokia 6100



รูปที่ 2.8 ตำแหน่งขาที่ใช้ในการเชื่อมต่อกับจอ

ขาของ LCD6100 มี 10 ขา โดยมีรายละเอียดดังนี้

ขา 1 VDIG คือ แรงดันไฟเลี้ยงสำหรับวงจร Digital เช่น ไฟเลี้ยง LCD Controller ที่ฝังอยู่

ขา 2 RST คือ ขารีเซต โดยทำงานที่ลอจิก 0

ขา 3 SDAT คือ ขาสำหรับรับส่งข้อมูลแบบอนุกรม

ขา 4 SCLK คือ สำหรับสัญญาณนาฬิกาสำหรับกำหนดจังหวะการอ่านข้อมูล

ขา 5 CS คือ Chip Select สำหรับ enable การรับส่งข้อมูลอนุกรม ทำงานที่ลอจิก 0

ขา 6 VDISPLAY แรงดันที่ใช้ในการแสดงผล สามารถใช้ได้กับแรงดัน 3.3V

ขา 7 NC ไม่มีการใช้งาน

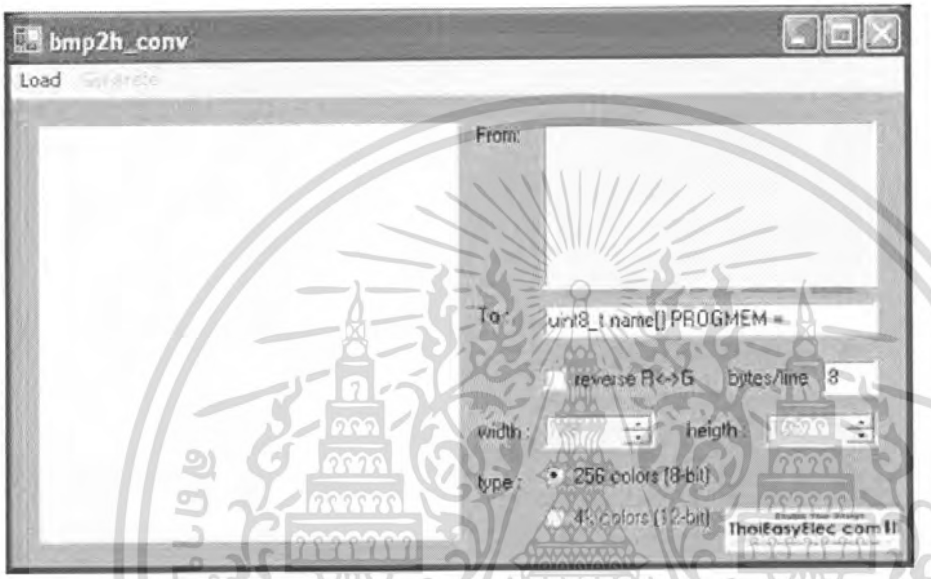
ขา 8 GND

ขา 9 LED- เป็นขาลบ สำหรับไฟ Backlight ถ้าไม่ต้องการควบคุม การจ่ายไฟ สามารถต่อ GND ได้โดยตรง

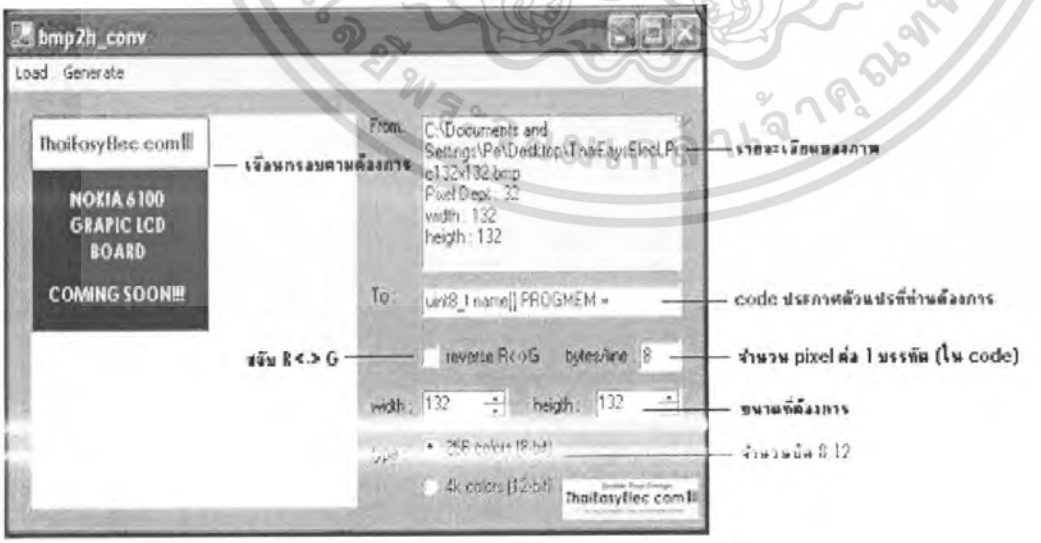
ขา 10 LED+ เป็นขาบวก สำหรับไฟ Backlight ใช้ได้กับแรงดันประมาณ 6-7 V

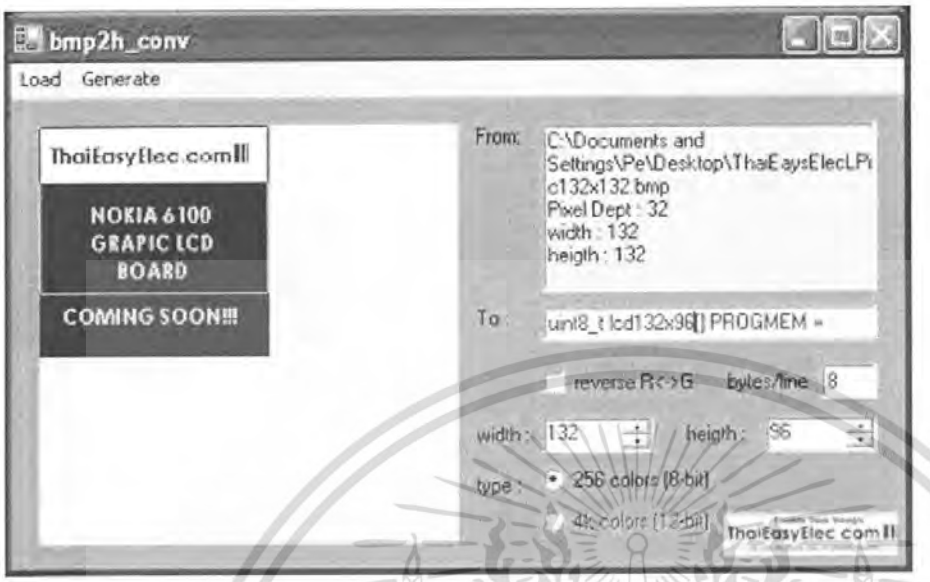
2.10 โปรแกรม convert ไฟล์ภาพ

ถ้าเราต้องการนำภาพจากในคอมของเรามาแสดงบนจอ LCD 6100 ในที่นี้จะใช้การสร้าง .h ไฟล์ มาเพื่อเก็บภาพไว้ในหน่วยความจำ SD Card ก่อนอื่น เราก็ต้องสร้างไฟล์ .h ก่อน โดยทาง ThaiEasyElec.com ได้เขียนโปรแกรมด้วยภาษา C# สำหรับสร้างไฟล์ .h จาก .bmp ให้ใช้งานง่ายๆ ดังนี้



โปรแกรมมีชื่อว่า bmp2h_conv เมื่อเปิดโปรแกรมขึ้นมา ให้เรากด Load เพื่อโหลดภาพ .bmp .jpg





เมื่อ Load มาแล้ว ช่อง From จะแสดงรายละเอียดของภาพ เราสามารถกำหนดขนาดของภาพที่เราต้องการแสดงได้ที่ช่อง width และ height จากนั้นสามารถใช้ mouse เลื่อนกรอบภาพได้

เมื่อเลือกขนาดและบริเวณที่ต้องการได้แล้ว ก็กด Generate เพื่อสร้างไฟล์ .h ก็จะได้ไฟล์ที่มีรูปแบบดังนี้

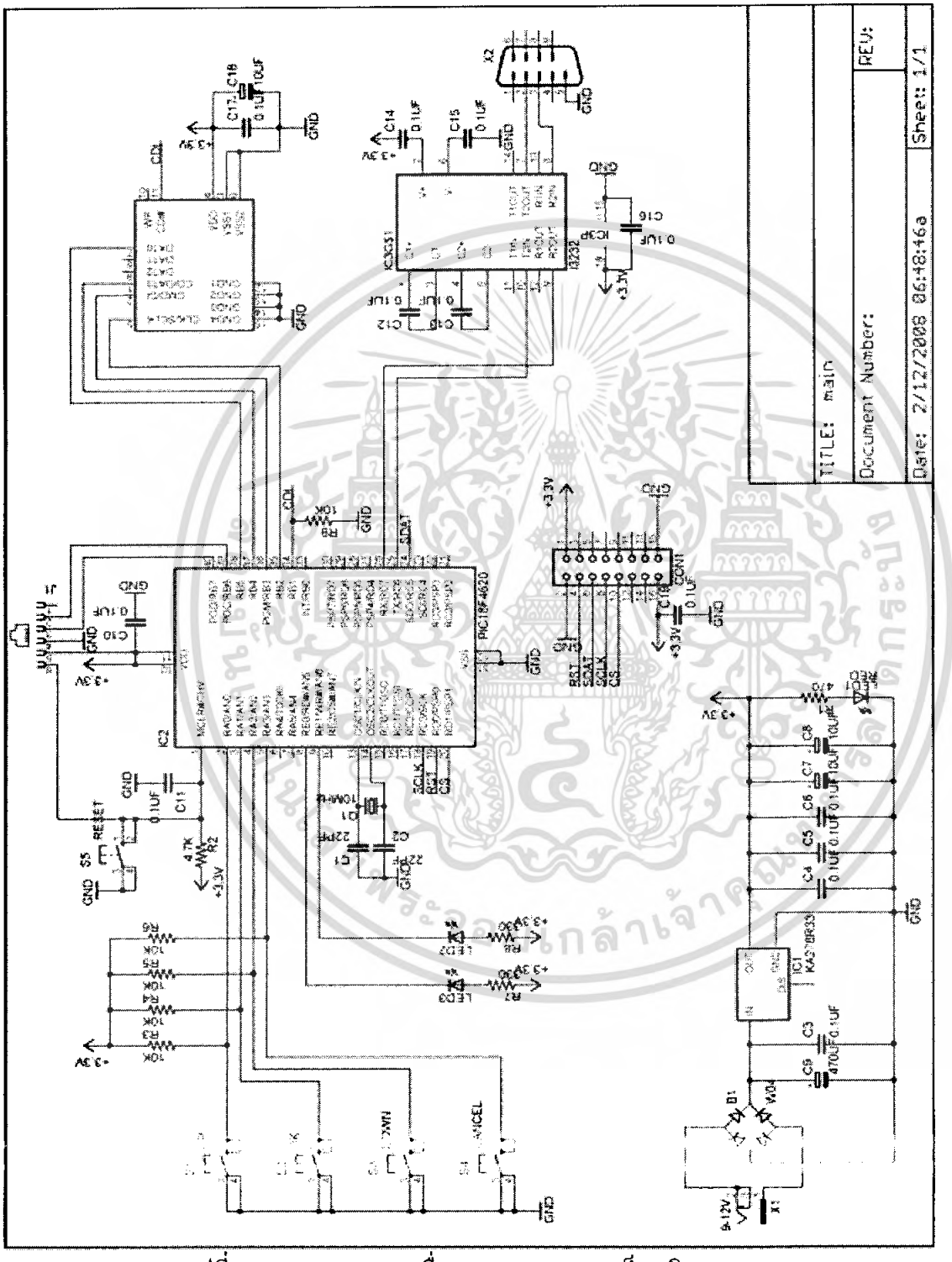
```
//#############################################
// bmp to header file convertor version 1.0
// Author : www.ThaiEasyElec.com
//#############################################
// source bmp file : C:\Documents and Settings\Pe\Desktop\ThaiEaysElecLPic132x132.bmp
// source pixel dept : 32
// target name : C:\Documents and Settings\Pe\Desktop\bmp2h_conv_pic\lcd132x96.h
// target pixel dept : 8
// target width : 132
// target height : 96
// #############################################
uint8_t lcd132x96[] PROGMEM =
{
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
```

```
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF, ...
```

บทที่ 3

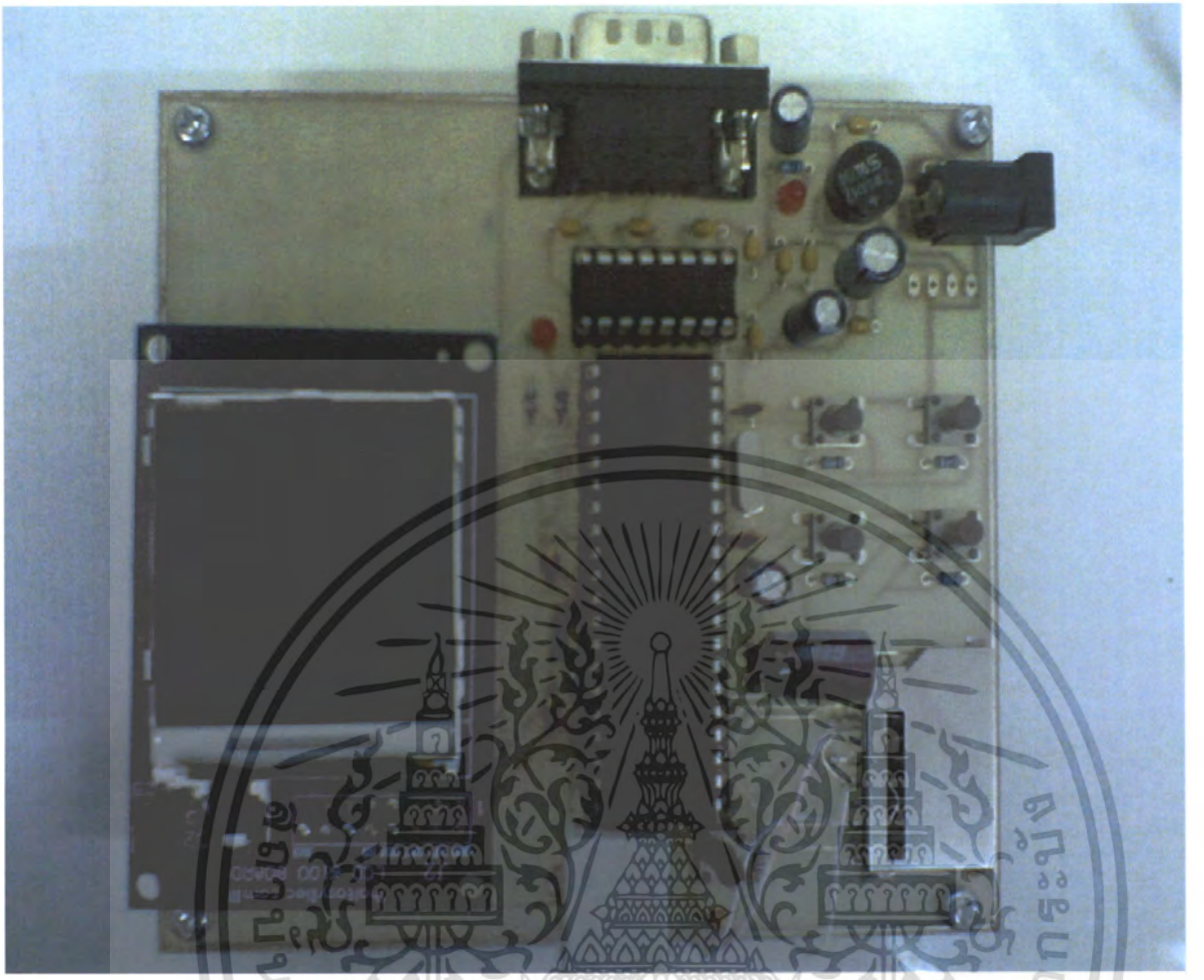
การออกแบบ

3.1 วงจรรวมของเครื่องแสดงภาพขนาดเล็ก



รูปที่ 3.1 วงจรรวมของเครื่องแสดงภาพขนาดเล็กชนิดพกพา

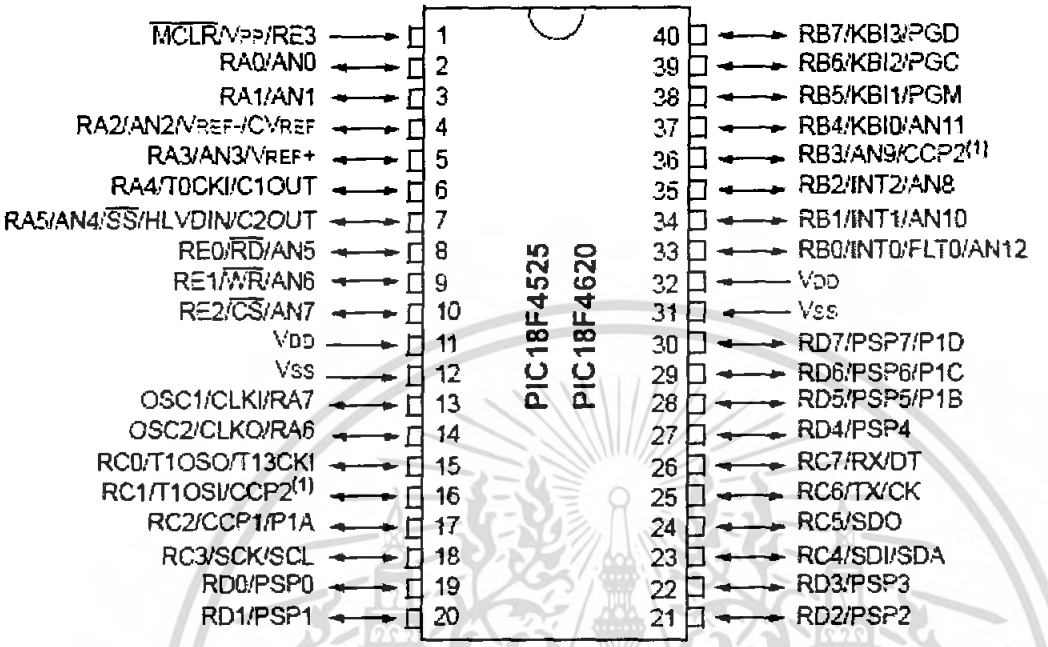
เครื่องแสดงภาพขนาดเล็กชนิดพกพาแบ่งอุปกรณ์เป็น 3 ส่วนดังนี้



รูปที่ 3.2 วงจรเครื่องแสดงภาพขนาดเล็กที่ทำการออกแบบ

เครื่องแสดงภาพขนาดเล็กชนิดพกพาแบ่งอุปกรณ์เป็น 3 ส่วนดังนี้

3.1.1 Microcontroller



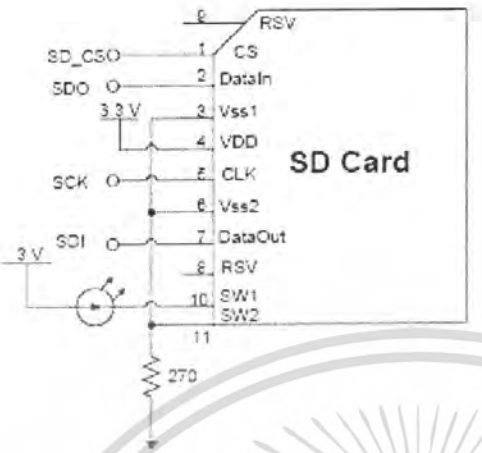
รูปที่ 3.3 Microcontroller PIC 18F4620

เนื่องจากผู้จัดทำต้องการใช้ไมโครคอนโทรลเลอร์ที่มีการประมวลผลรวดเร็ว และรองรับการติดต่อในระบบ SPI BUS จึงเลือกใช้ไมโครคอนโทรลเลอร์ตระกูล PIC เบอร์ 18F4620 และใช้คริสตอลความถี่ 10 เมกะเฮิรตซ์

การส่งข้อมูลจากไมโครคอนโทรลเลอร์ให้ไมโครคอนโทรลเลอร์ของจอ LCD นั้นจะใช้พอร์ต SPI ที่เขียนฟังก์ชันขึ้นมาเองในการส่งข้อมูลแทน ข้อดีของวิธีนี้คือ เราสามารถกำหนดอัตราเร็วในการับ/ส่งข้อมูลระหว่างอุปกรณ์ทั้งสามได้เป็นอิสระต่อกัน และยังง่ายต่อการควบคุมในระบบจริงอีกด้วย ส่วนข้อจำกัดคือ เราต้องเปลืองพอร์ตสำหรับใช้กับฟังก์ชันที่เขียนขึ้นมาเอง และจำเป็นต้องสร้างสัญญาณลักษณะ SPI ขึ้นมาให้ตรงกับมาตรฐานของสัญญาณ SPI

ในส่วนของอินพุทเพื่อแสดงภาพจะมีอยู่ 4 อินพุท คือ Up, Down, OK, Cancel
การติดต่อกับตัวไมโครคอนโทรลเลอร์ของจอมี 4 ขา คือ RST, SDAT, SCLK, CS
ส่วนการติดต่อกับ SD/MMC Card มี 4 ขา คือ $\overline{SD\_CS}$, SCK , SDO , และ SDI

3.1.2 SD Card



รูปที่ 3.4 วงจรในส่วนของ SD Card

ผู้จัดทำได้เลือกใช้ SD Card เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไม่ค่อยสูงมากและเป็นการ์ดที่หาซื้อได้ง่าย โดย SD Card ที่ใช้มีความจุ 1 จิ๊กกะไบต์ รูปที่3.3แสดงขาของ SD Card ที่ใช้

3.1.3 LCD

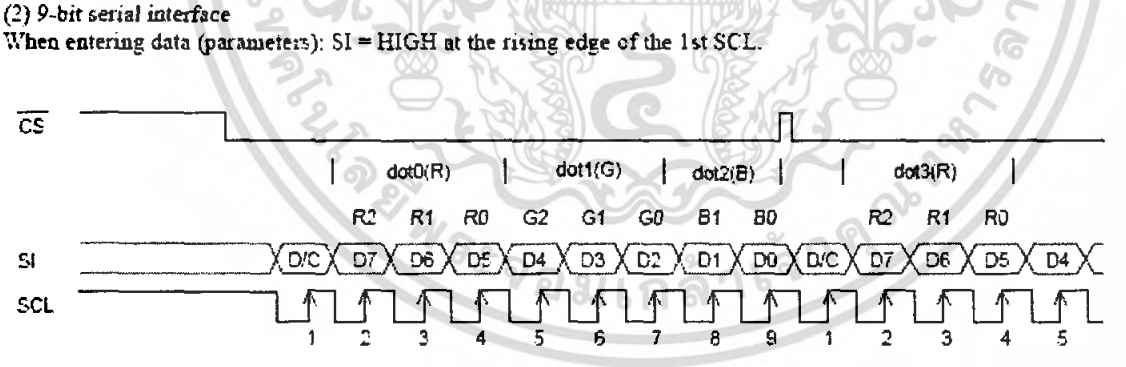
รูปที่ 3.5 LCD ที่นำมาใช้งาน

LCD ที่ใช้ในโครงงานนี้เป็นส่วนสำคัญในการแสดงผลซึ่งจะเลือกใช้ LCD มือถือของ Nokia 6100 ที่ใช้ Chip Set ของบริษัท Epson (S1D15G10) ที่กินกระแสต่ำ เป็นหน้าจอแบบ STN มีการแสดงผลได้ 4,098 สี มีขนาด 128 ดอท (ในแนวตั้ง) และ 128 ดอท (ในแนวนอน) ซึ่งจะมีการ Interface ได้ 3 แบบคือ Serial Interface, Parallel Interface, EEPROM Interface โดยจะเลือกใช้แบบ Serial Interface จากรูปที่ 3.4 เนื่องจากขาที่ต่อออกมาเพื่อใช้งานมี ขา DATA , CLOCK , CHIP ENABLE , RESET ซึ่งการแสดงผลของจอสีจะใช้หลักการเอาแม่สี ซึ่งมีอยู่สามสี คือ สีน้ำเงิน , สีแดง , สีเขียว (RGB) มาผสมในสัดส่วนที่แตกต่างกันเพื่อทำให้เกิดสีของ pixel นั้นๆ ซึ่งความเข้มและความจางของเม็ดสีขึ้นอยู่กับสัดส่วนของการผสมค่าของแม่สี

การติดต่อ LCD Nokia 6100 ด้วยไมโครคอนโทรลเลอร์

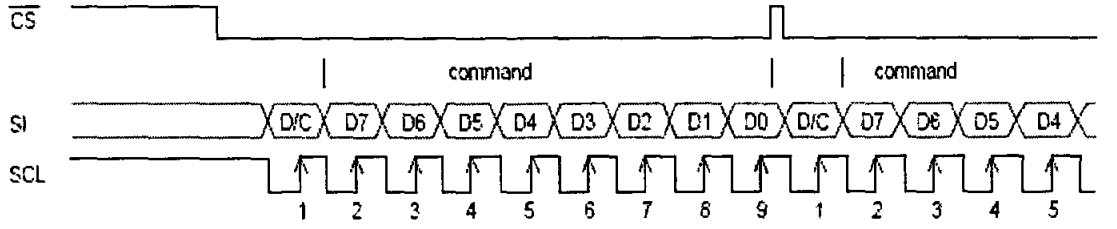
ภายในจอ LCD 6100 จะมี LCD Controller ที่ทำหน้าที่สแกนภาพออกที่จอ โดยจะมีหน่วยความจำ SRAM ในตัว และเวลาเราติดต่อกับจอ ก็คือการติดต่อกับตัว LCD Controller ซึ่งจริงๆแล้ว สามารถติดต่อได้ทั้งแบบ Parallel และแบบอนุกรม แต่ผู้ผลิตจอ LCD 6100 ได้ต่อวงจรในส่วนนี้เรียบร้อยแล้ว และให้เราติดต่อได้ทางอนุกรมเท่านั้น

รูปแบบของการส่งแบบ Serial interface แบบ 9 บิต



รูปที่ 3.6 รูปแบบการส่ง DATAให้กับจอ

When entering commands: SI = LOW at the rising edge of the 1st SCL.

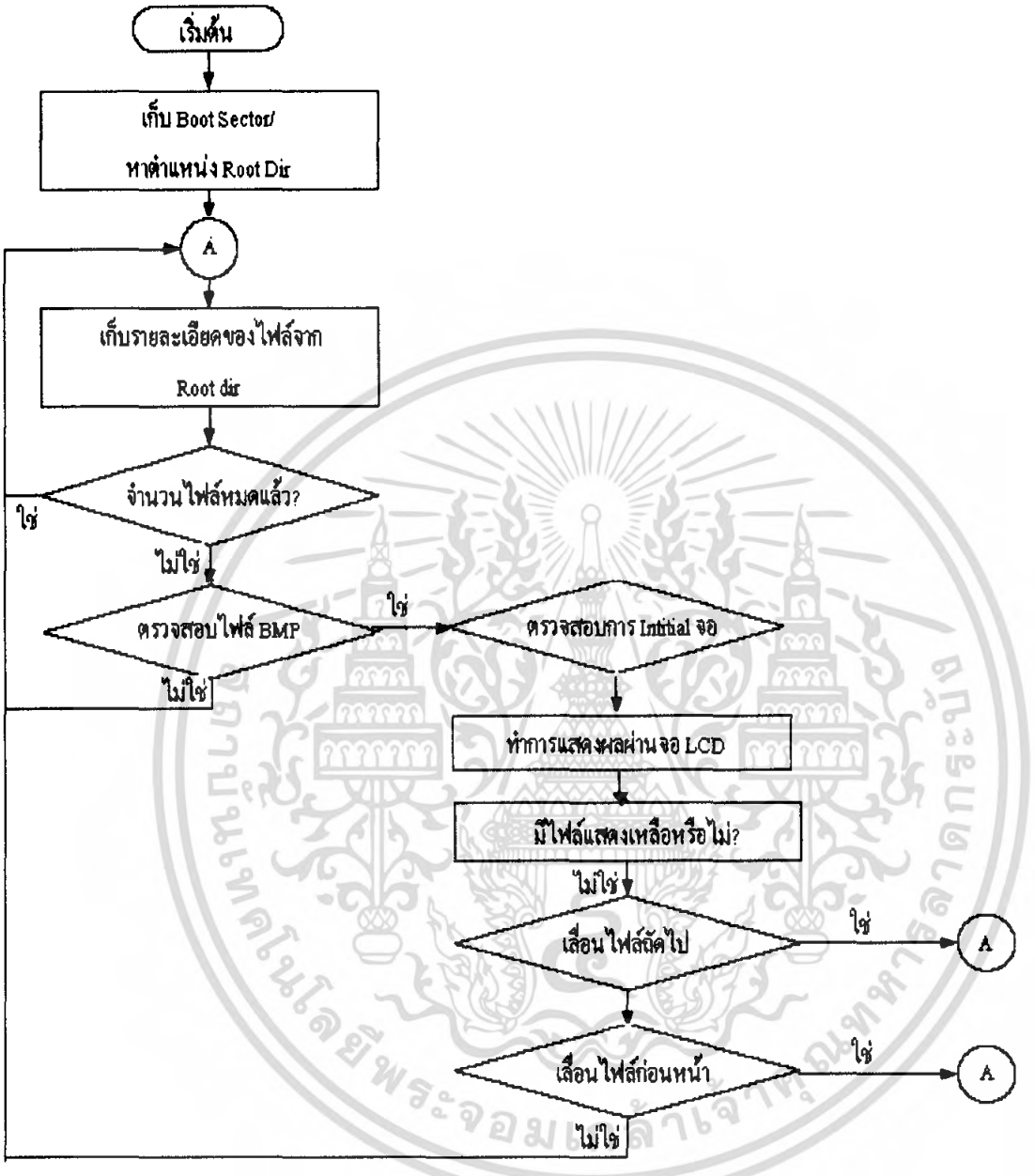


รูปที่ 3.7 รูปแบบการส่ง COMMANDให้กับจอ

หลักการทำงานคือ เมื่อ ขา CS หรือขา Chip Select เป็นขาที่ทำหน้าที่ Enable/Disable การทำงานของจอ หากมีการเลือกที่จะส่งข้อมูลหรือคำสั่งจะต้องให้สถานะลอจิกของขานี้เป็น Low และ High (Toggle) เพื่อทำให้ Chip ภายในรับข้อมูลที่ป้อนเข้ามาที่ขา DATA ขาต่อไปเป็นสัญญาณนาฬิกา (SCK) ใช้เพื่อกำหนดให้มีการส่งข้อมูลมาที่ขา DATA ทีละบิต ซึ่งข้อมูล 1 บิต จะใช้ Clock 1 ลูก และอีกขาคือ ขา RESET เพื่อใช้ในการ Reset Chip ภายใน

จากรูปที่ 3.5 และ 3.6 เนื่องจากจอที่เราใช้มีขา DATA เพียงขาเดียวการที่เราจะบ่งบอกว่าข้อมูลที่ส่งมาว่าเป็น DATA หรือ Command จะดูที่ bit ที่ 8 คือ bit D/C ซึ่งเมื่อบิตนี้เป็นลอจิก low ข้อมูลอีก 8 bit ต่อมาจะเป็น command และ ถ้าบิตนี้เป็นลอจิก high ข้อมูล 8 บิต ต่อมาจะเป็น Data สำหรับสัญญาณ CS ใช้กำหนดจังหวะการเริ่มต้นการส่งข้อมูล เพื่อให้ผู้ส่งและผู้รับ เข้าใจตรงกันว่าให้นับบิตไหนเป็นบิตแรก และแต่ละไบต์ สามารถส่งต่อเนื่องกันโดยที่ไม่ต้องหยุดการส่งข้อมูลแล้วเริ่มใหม่ โดยสามารถเริ่มต้นครั้งเดียวแล้วส่งคำสั่งตามด้วยค่าพารามิเตอร์ แล้วส่งคำสั่งถัดไปได้ทันที

ผังการทำงานของเครื่องแสดงภาพขนาดเล็กมี SD Card

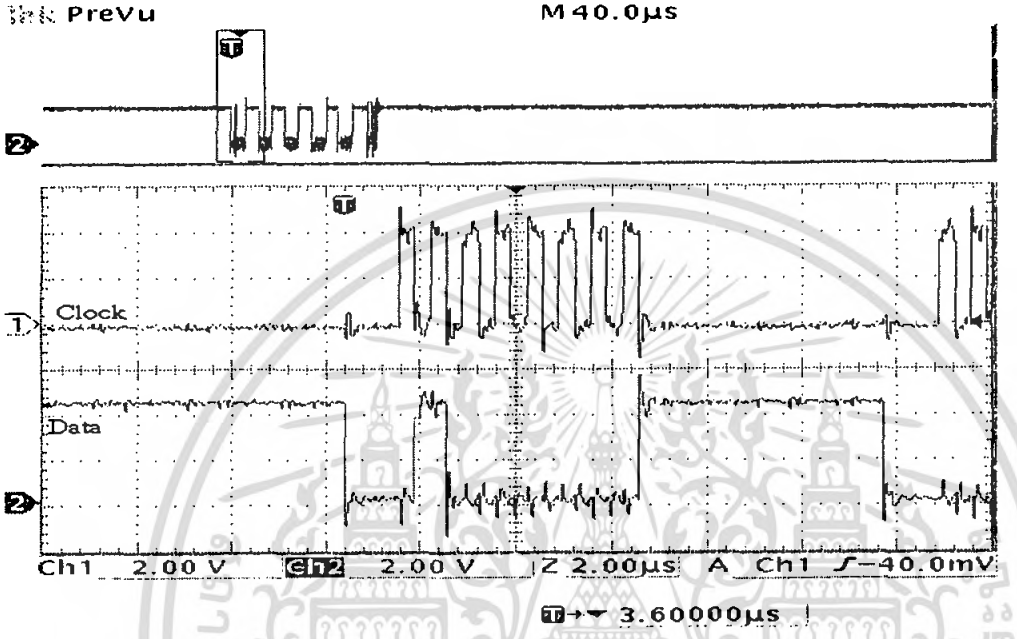


รูปที่ 3.8 ผังการทำงานของเครื่องแสดงภาพนิ่งขนาดเล็กมี SD Card

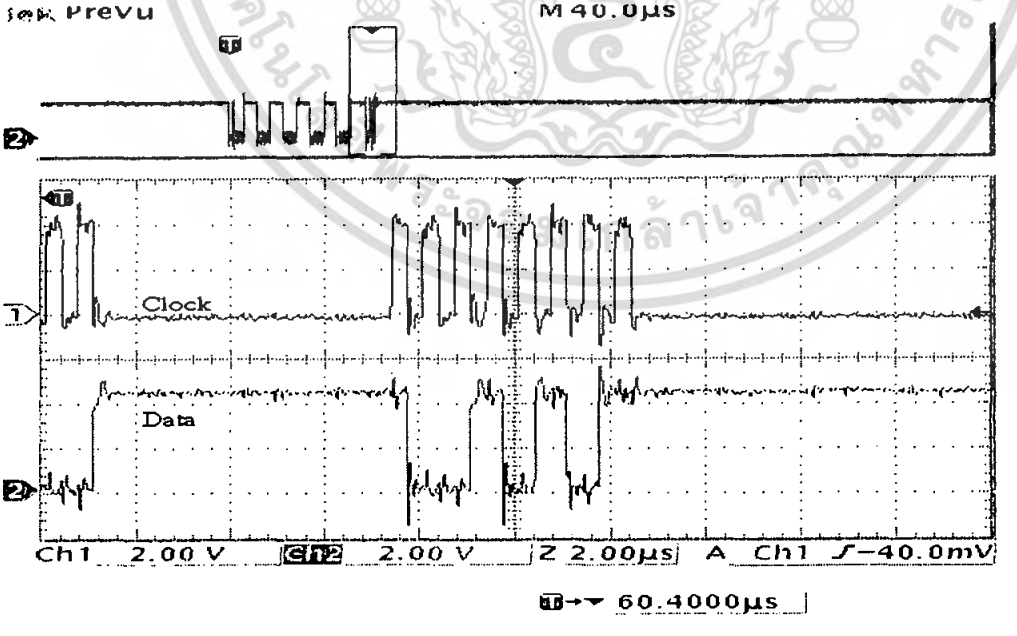
บทที่ 4

ผลการทดลอง

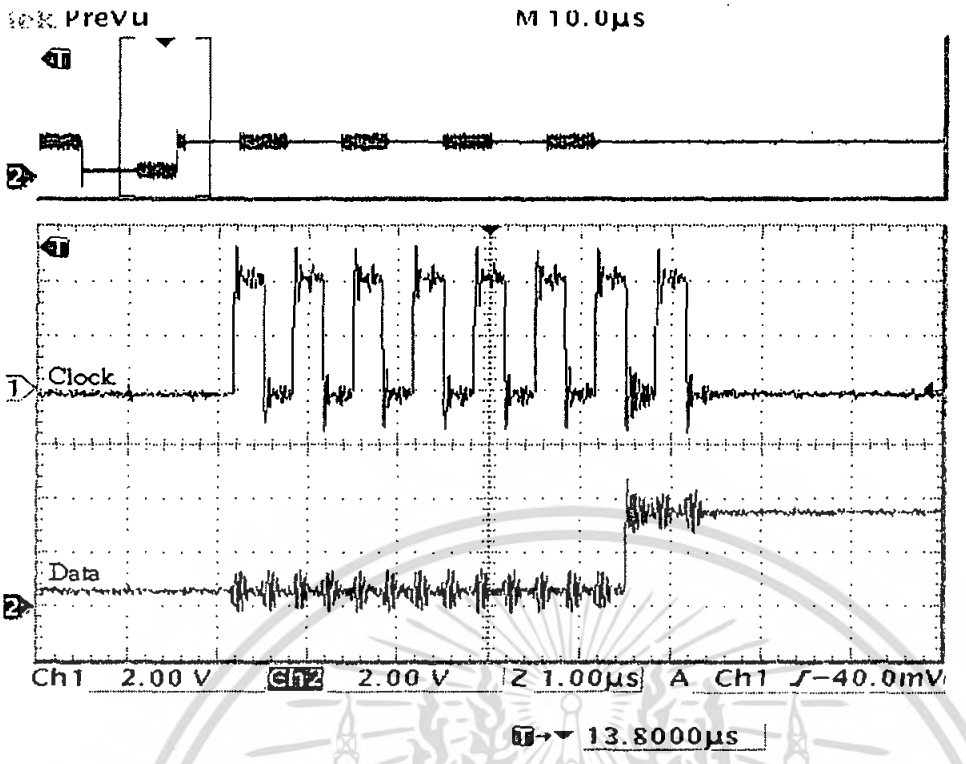
4.1 <u>การทดลองที่ 1</u> วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus โดยเชื่อมต่อจาก Microcontroller กับ SD Card



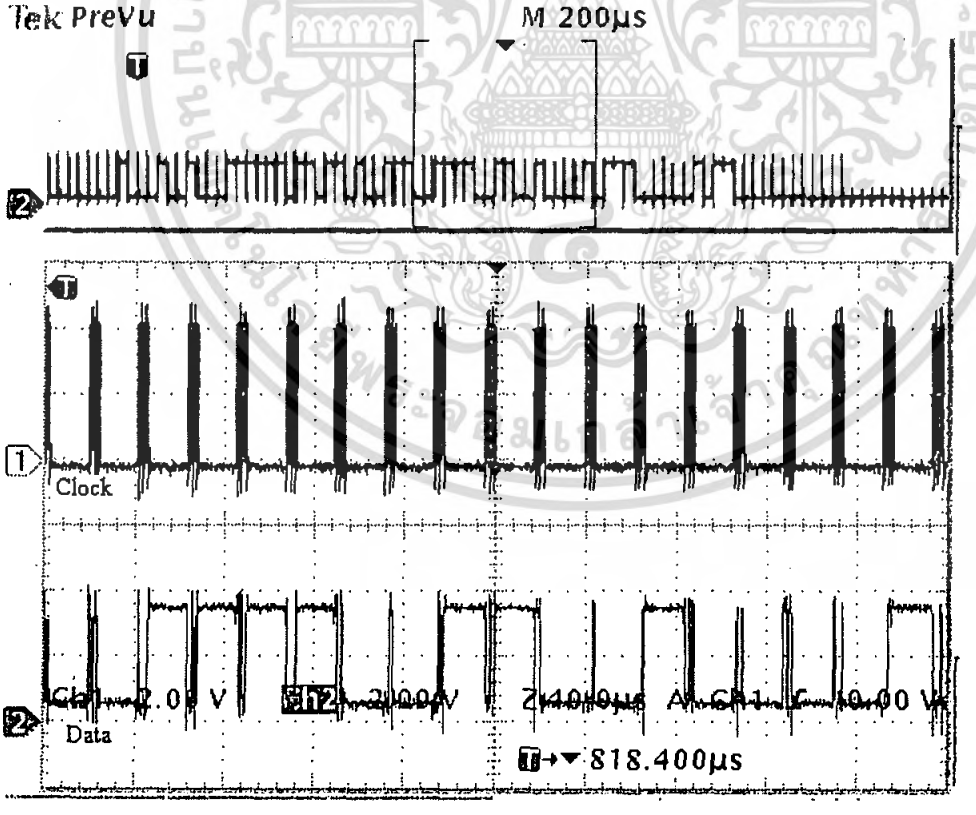
รูปที่ 4.1 รูปสัญญาณคำสั่ง RESET (0x40)



รูปที่ 4.2 รูปสัญญาณคำสั่งRESET (0x95)

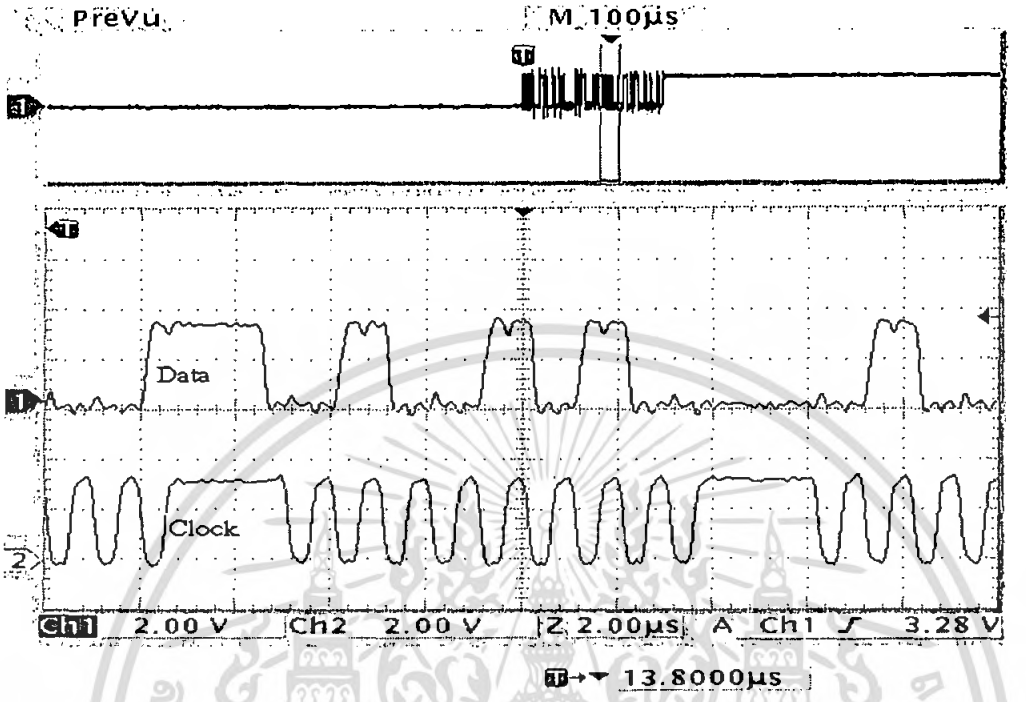


รูปที่ 4.3 รูปกราฟของสัญญาณตอบสนองคำสั่งRESET (0x01)

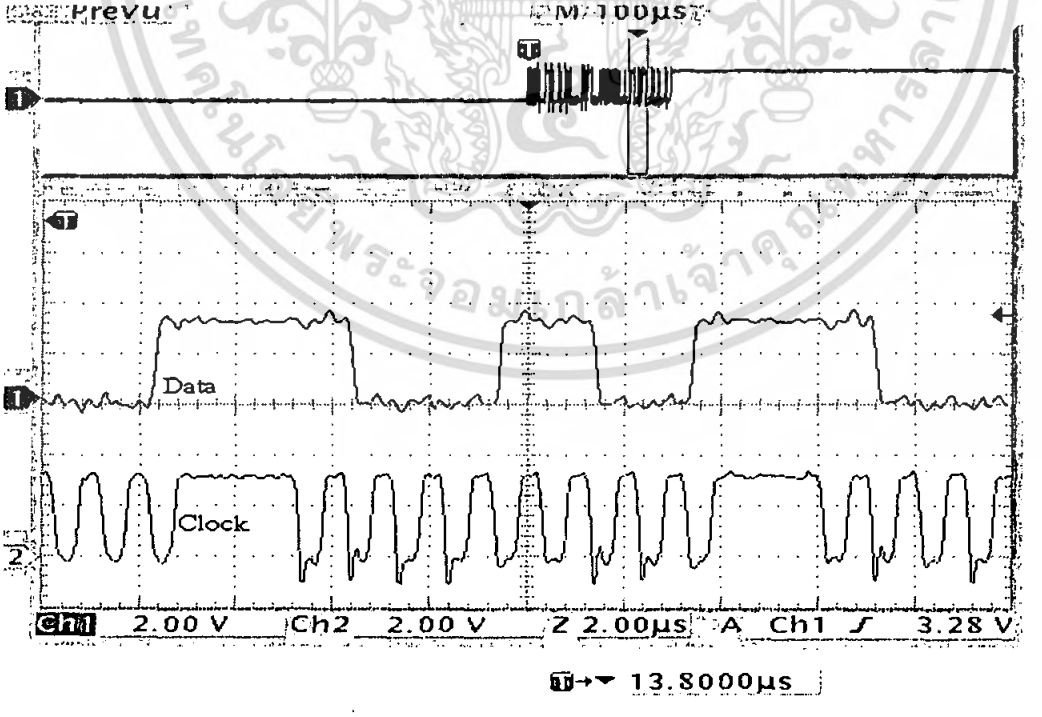


รูปที่ 4.4 สัญญาณของชุดข้อมูลที่ได้รับจากการ์ด

4.2 <u>การทดลองที่2</u> วัดสัญญาณการสื่อสารแบบ SPI ระหว่าง Microcontroller กับ LCD

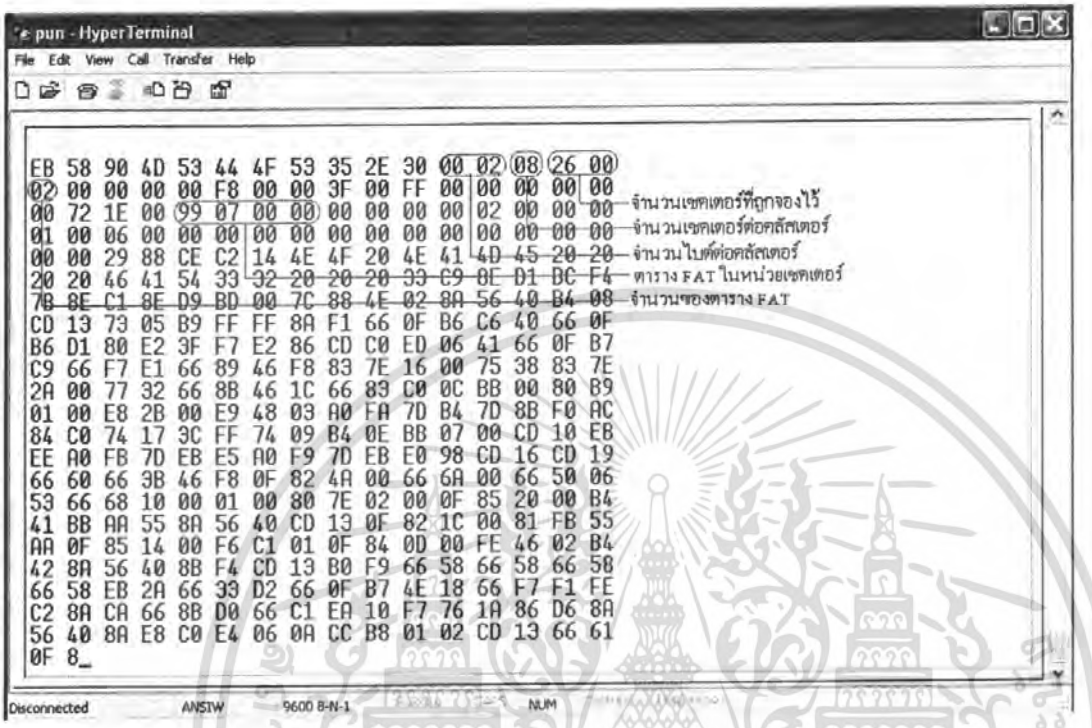


รูปที่ 4.5 สัญญาณของการส่ง COMMAND ให้จอ

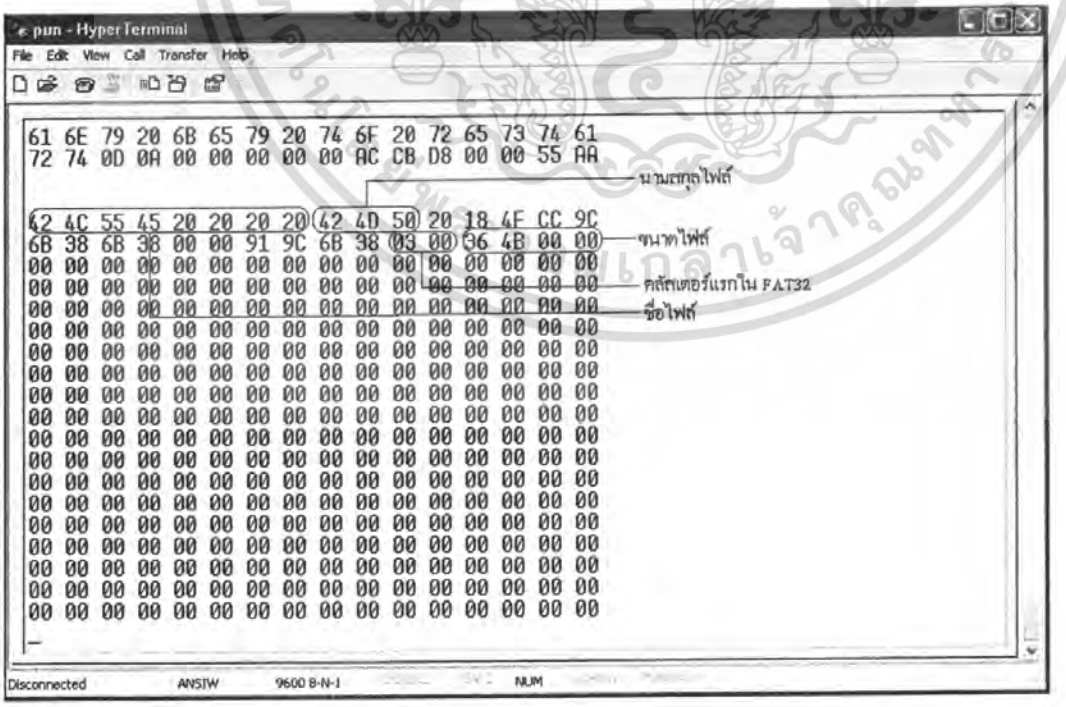


รูปที่ 4.6 สัญญาณของการส่ง DATA ให้จอ

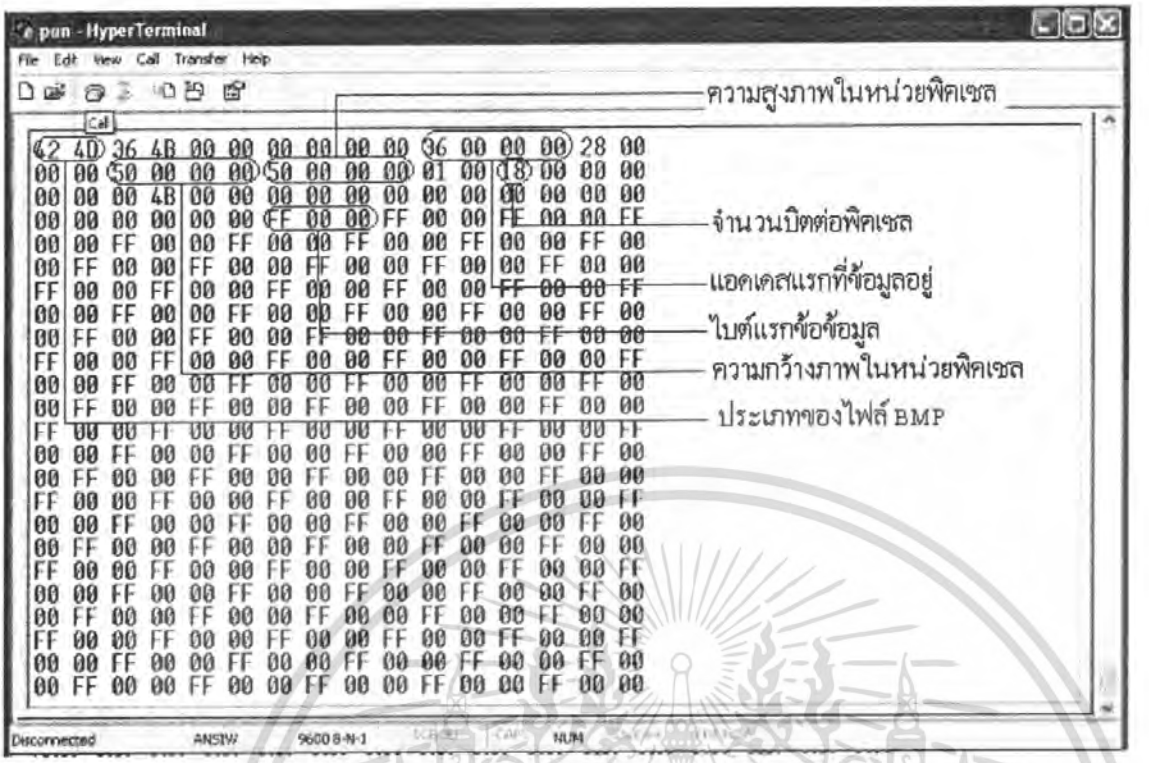
4.3 <u>การทดลองที่3</u> ทดสอบการแสดงข้อมูลรูปสีแดงที่ได้จาก SD Card แสดงผ่าน Hyperterminal โดยเชื่อมต่อ SD Card กับ Microcontroller



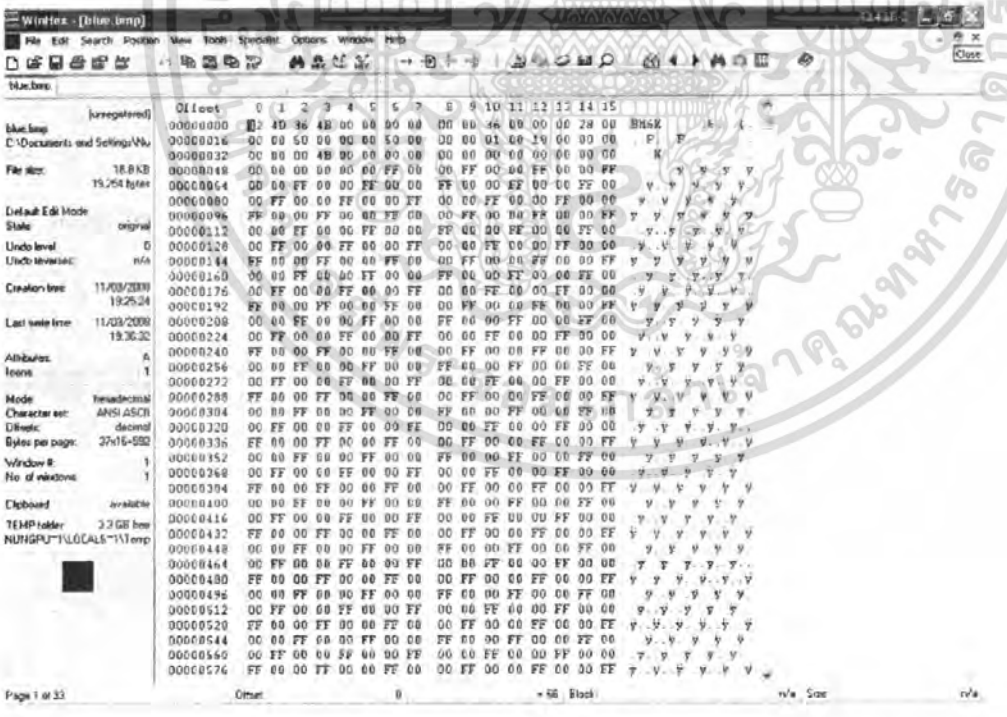
รูปที่ 4.7 ส่วนของ Bios Parameter Block (BPB) ของสีแดง



รูปที่ 4.8 ส่วนของ Root Directory ของสีแดง



รูปที่ 4.9 ส่วนของ Data ของสีแดง



รูปที่ 4.10 ส่วนของ Data สีแดงจาก Winhex ที่นำมาเปรียบเทียบกับ Data ที่ได้

4.4 <u>การทดลองที่4</u> ทดสอบการแสดงภาพจาก LCD โดยเชื่อมต่อ Microcontroller กับ LCD

รูปที่ 4.11 แสดงภาพ Bitmap 24 bit ในคอมพิวเตอร์ก่อนนำมาแสดงผ่าน LCD

รูปที่ 4.12 แสดงภาพที่ถูกแปลงเป็น Bitmap 12 bit ผ่านทาง LCD

จากภาพที่นำมาแสดงเป็นการกำหนดให้ภาพเริ่มแสดงที่ตำแหน่ง แถวแรก(PASET) และหลักแรก (CASET) แต่เนื่องจากภาพที่นำมาแสดงนั้นมีขนาดไม่เต็ม Resolution ของจอ ภาพที่แสดงออกจอจึงได้เป็นดังรูป

4.5 <u>การทดลองที่5</u> นำภาพจาก SD Card แสดงผ่านทาง LCD โดยเชื่อมต่อ SD Card กับ Microcontroller และ LCD

การทดลองส่วนนี้ยังไม่สมบูรณ์ เนื่องจาก ไมโครคอนโทรลเลอร์ PIC ที่ใช้งานมีพอร์ต SPI เพียงพอร์ตเดียว ทำให้ต้องเขียนฟังก์ชั่น SPI เพิ่มขึ้นมาเพื่อทำการส่งข้อมูลออกมาแสดงผล หรือทำการเปลี่ยนตัวไมโครคอนโทรลเลอร์ที่มีพอร์ต SPI 2 พอร์ต เช่นไมโครคอนโทรลเลอร์ ARM และในการโปรแกรมส่วนของ FAT จะใช้พื้นที่ของแรมไปมากจึงควรเพิ่มส่วนของแรมใน ตัวไมโครคอนโทรลเลอร์ให้มากขึ้นและเนื่องจากเวลาในการทำมีจำกัดจึงไม่สามารถทำการทดลองนี้ให้สมบูรณ์ได้

ระบบที่สมบูรณ์ควรมีคุณสมบัติดังนี้

1. สามารถนำภาพ Bitmap จาก SD Card มาแสดงผ่านทาง LCD ที่เป็นภาพสีได้
2. สามารถแสดงชื่อไฟล์ทั้งหมดใน SD Card
3. สามารถ เลือกดูภาพว่าจะดูภาพไหนและเลือกดูภาพที่ต้องการทีละภาพได้
4. ความสามารถพิเศษอื่นๆ เช่น แสดงภาพในแนวตั้งและแนวนอน แสดงมากกว่า 1 ภาพในเวลาเดียวกัน และสามารถย่อหรือขยายภาพได้

บทที่ 5
บทสรุป

5.1 สรุป

ในรายงานได้กล่าวถึงความเป็นมาของโครงงาน แนวคิด ทฤษฎี และรายละเอียดการสร้างเครื่องแสดงภาพชนิดพกพามี SD/MMC Card รวมถึงการทดสอบการทำงานเบื้องต้น ในการทดสอบการติดต่อระหว่างไมโครคอนโทรลเลอร์ (PIC18F4620) กับจอ LCD ที่ใช้ใน มือถือ โดยการติดต่อนั้นสามารถติดต่อได้โดยตรงเลย โดยสร้างสัญญาณจาก ไมโครคอลโทรเลอร์ โดยส่งค่าคำสั่งเป็นการเซตค่าเริ่มต้นให้กับจอ จากนั้นก็ส่งรหัสสีไปพล็อตบน ตำแหน่ง pixel ที่เราต้องการได้ทันที

ส่วนการทดสอบการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD/MMC Card กระทำโดยให้ไมโครคอนโทรลเลอร์อ่านค่ารีจิสเตอร์จาก SD Card 2 รีจิสเตอร์ คือ รีจิสเตอร์ CID และ CSD โดยการติดต่อทั้งหมดระหว่างไมโครคอนโทรลเลอร์กับจอมือถือ และ SD/MMC Card เป็นการติดต่อในระบบ SPI BUS ซึ่งจะต้องใช้การสร้างสัญญาณจากการเขียนฟังก์ชั่น SPI ขึ้นมาเอง เพื่อทำการติดต่อกับ LCD โดยจะต้องให้สัญญาณที่สร้างขึ้นมาตรงกับมาตรฐานของสัญญาณ SPI

5.2 ปัญหาและแนวทางในการแก้ไข

- เนื่องจากไมโครคอนโทรลเลอร์ PIC เบอร์ที่ใช้มีพอร์ต SPI เพียงพอร์ตเดียว และมีแรมภายในน้อยในการเก็บข้อมูล จึงใช้ SPI พอร์ตเดียวกันแล้วทำการ Slave Select เพื่อเลือกว่าจะให้ไมโครคอนโทรลเลอร์ติดต่อกับ Slave ตัวอื่นๆหรือทำการเขียนฟังก์ชันเพื่อสร้างพอร์ต SPI เพิ่มขึ้นมา หรืออาจจะใช้ไมโครคอนโทรลเลอร์ตัวใหม่ที่มีพอร์ต SPI สองพอร์ต และมีแรมภายในสูง เช่น ARM ซึ่งมีแรมภายในที่สูงพอสมควร และมีฟังก์ชั่น SPI ให้เลือกใช้ สองพอร์ต

- เนื่องจาก SD/MMC Card มีการจัดเรียงข้อมูลในระบบ FAT32 ในการตรวจสอบความถูกต้องจากการอ่านข้อมูลใน SD/MMC Card จึงทำได้ยาก ดังนั้นผู้จัดทำจึงใช้ซอฟต์แวร์ (Winhex) มาช่วยตรวจสอบข้อมูลและตำแหน่งแอดเดรสภายใน SD Card

-เนื่องจากจอที่นำมาใช้ในโครงงานนี้ต้องมีคอนเนคเตอร์ เชื่อมออกไปซึ่งมีขนาดเล็กยุ่งยากต่อการออกแบบและ Blacklight ที่จ่ายให้จอสว่างเพื่อที่จะทำให้เห็นสีชัดขึ้น หากจ่ายไฟตรงให้เป็นเวลานานๆจะทำให้เกิดความร้อนทางผู้ทำจึงซื้อบอร์ดสำเร็จรูปมาใช้เพราะสะดวกในการใช้ และมีวงจรช่วยป้องกันไม่ให้จอเกิดความเสียหาย เมื่อจ่ายไฟเป็นเวลานาน

-เนื่องจากไฟล์ Bitmap ทั่วไปที่อ่านได้ เป็นแบบ 24 bit แต่จอ LCD ที่นำมาใช้นั้นสามารถแสดงผลได้เพียง 12 bit RGB หรือแสดงได้ 4096 สี จึงต้องทำการสร้างฟังก์ชั่นเพื่อลดระดับสีให้เป็นเหลือเพียง 12 bit เพื่อที่จะแสดงสีออกมาได้ถูกต้อง

5.3 ประโยชน์ที่ได้รับ

- มีความรู้ความเข้าใจการติดต่อสื่อสารข้อมูลในแบบอนุกรม SPI
- เข้าใจการใช้งานเกี่ยวกับหน้าจอแอลซีดีมือถือที่มีการแสดงผลได้หลายสี
- มีความรู้ในการใช้งานไมโครคอนโทรลเลอร์ตระกูล PIC
- เนื่องจาก SD/MMC Card มีการจัดเรียงไฟล์แบบ FAT32 ทำให้ผู้จัดทำมีความเข้าใจในตัวระบบ FAT32 เพื่อใช้ในการติดต่อกับ SD Card

คุณสมบัติของเครื่องแสดงภาพขนาดเล็ก

สามารถอ่านข้อมูลภาพ BMP (Bitmap) จากหน่วยความจำ SD CARD ไปแสดงผ่านจอ LCD ที่สามารถแสดงสีได้ 4096 สี

แนวทางในการพัฒนาต่อไป

1. สามารถเลือกภาพโดยอ่านชื่อไฟล์ภาพแสดงบนจอ LCD แล้วเลือกภาพที่ต้องการดูได้
2. สามารถนำไปใช้แสดงภาพเคลื่อนไหวได้ เช่น ทำเป็นเกมงูกินหาง
3. อาจนำไปร่วมใช้กับพอต VGA เพื่อเป็นตัวเลือกในการนำภาพออกแสดงทางน่าจอได้ ทั้งพอต VGA และจอ LCD ขนาดเล็กได้

บรรณานุกรม

1. Pic microcontroller programming with CCS C compiler : ประจิน พลังสันติกุล

2. เรียนรู้และพัฒนาไมโครคอนโทรลเลอร์ ARM7 LPC2148 ด้วยภาษาซี : โอภาส ศิริครรชิตถาวร

3. มาเล่นจอ 6610 กันเถอะ , www.electoday.com

4. SparkFun Electronics, Nokia 6100 LCD Display Driver ,Author: James P. Lynch

ภาคผนวก

```
/*********************************************************************
**********
 * File    : SD Card

*********************************************************************
**********/

#include <18F458.h>
#define  TxD       PIN_C6
#define  RxD       PIN_C7
//#define  LED1      PIN_A0
//#define  LED2      PIN_A1
//#define  LED3      PIN_C1
//#define  LED4      PIN_C2
#define  MMC_CS     PIN_B4
#define  CLOCK_SP   10000000
#define MMCSD_PIN_SCL    PIN_C3 //o
#define MMCSD_PIN_SDI    PIN_C4 //i
#define MMCSD_PIN_SDO    PIN_C5 //o
//#define MMCSD_PIN_SELECT  PIN_C2 //o


#fuses HS
#fuses NOLVP, NOWDT         // No Low Voltage Program, No Watchdog timer
#fuses NOPROTECT            // Code no protection
#use delay (clock=CLOCK_SP)   // Use built-in function: delay_ms() & delay_us()
#use rs232(baud=9600, xmit=TxD,rcv=RxD) // Use serial I/O port (RS232)

int fat,file_ad[4],root_ad[4],numfile,sec_per_clus,FileName;
int16 l,i,m,byte_per_sec,sec_num,sec_total,rsv_size,fat_size;
unsigned char count;
unsigned char resp[10],data[512],detail[512];
int32 file_size,root_start,start_addr;

//void blink1();
//void blink2();
void pic_init();
void mmc_select()  {output_low(mmc_cs);}
void mmc_deselect(){output_high(mmc_cs);}

void mmc_clock(unsigned char cycle)
{
  do{
    spi_write(0xff);
    cycle--;
  }while(cycle>0);
}
```

```
void mmc_send_cmd(unsigned char cmd,unsigned char arg1,unsigned char
                  arg2,unsigned char arg3,unsigned char arg4
                  ,unsigned char crc)
{

  spi_write(cmd);
  spi_write(arg1);
  spi_write(arg2);
  spi_write(arg3);
  spi_write(arg4);
  spi_write(crc);


}

void MMC_init()
{
   unsigned char resp[2],count,j;

   mmc_deselect();
   mmc_clock(10);
   mmc_select();
   mmc_send_cmd(0x40,0x00,0x00,0x00,0x00,0x95);
   count=10000;

   do{
     resp[0]=spi_read(0xff);
    }while(resp[0]!=0x01 && --count>0);
     if(resp[0]==0x01){
     //printf("CMD0 response = %X\n\r" ,resp[0]);
     goto reset;
     }
     //else{printf("ERROR response CMD0 = %X\n\r" ,resp[0]);}

     reset:
   j=0;
   for(j=0;j<10;j++)
   {
     mmc_send_cmd(0x41,0x00,0x00,0x00,0x00,0xff);
     count=10000;
     do{
       resp[1]=spi_read(0xff);
      }while(resp[1]!=0x00 && --count>0);
      /*if(resp[1]==0x00){
      printf("CMD1 response = %X\n\r" ,resp[1]);
      }
      else{printf("ERROR response CMD1 = %X\n\r" ,resp[1]);} */
   }
}
```

```
void CheckFile()
{
  FileName = detail[0];
  if(FileName == 0)
  {
    printf("No File in card, Check Card!!!\n");
    while(1);
  }
  start_addr = detail[21+(numfile*32)];
  start_addr = start_addr<<8;
  start_addr += detail[20+(numfile*32)];
  start_addr = start_addr<<8;
  start_addr += detail[27+(numfile*32)];
  start_addr = start_addr<<8;
  start_addr += detail[26+(numfile*32)];
  if((start_addr & 0xFFFF) == 0)
  {
    //printf("No data in file, Check Card!!!\n");
    numfile++;
  }
}
/*********************************************************************/
/*********************************************************************/
/*                    MAIN PROGRAM                                   */
/*********************************************************************/
/*********************************************************************/

void main(void)
{
   setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_4);
   *0x94 |= 0x40;   // set CKE  = 1 - clock idle low
   *0x14 &= 0xEF;   // set CKP  = 0

   //pic_init();

/*********Initial MMC Card*****************************/
   resp[0]=0xFF;
   resp[1]=0xFF;
   MMC_init();

/*********Check Response for Header of DATA ****************************/

  delay_ms(50);

  mmc_deselect();
  mmc_clock(10);
  mmc_select();
  mmc_send_cmd(0x51,0x00,0x00,0x00,0x00,0xff); /******Address******/
  delay_ms(50);
```

```
count=50000;
resp[4]=0x55;
   do{
     resp[4]=spi_read(0xff);
   }while(resp[4]!=0x00 && --count>0);
   if(resp[4]==0x00){
   //blink1();
   //printf("Read BPB CMD response = %X\n\r" ,resp[4]);
   }
count=100000;
   do{
     resp[2]=spi_read(0xff);
   }while(resp[2]!=0xfe && --count>0);
   /*if(resp[2]==0xfe){
   //output_high(LED1);
   printf("Data token = %X\n\r" ,resp[2]);
   }
   else{
   //output_high(LED2);
   printf("ERROR Data token = %X\n\r" ,resp[2]);
   }*/

   i=0;
   for(i=0;i<512;i++)
   {
    detail[i]=spi_read(0xff);
   }
   spi_read(0xff);
   spi_read(0xff);
   spi_read(0xff);


  /*** show on Hyper Terminal ***/
   printf("\n\r");
   m=0;
   l=0;
   for(m=0;m<512;m++)
    {
    printf("%X " ,detail[m]);
    if(l==15){
    printf("\n\r");
    l=0;
    }
    else l++;
    }
    printf("\n\r");      /*****/

    mmc_deselect();
    spi_read(0xff);
    spi_read(0xff);
```

```
/*** Find start address of root directory ***/
      rsv_size=detail[15];
      rsv_size=rsv_size<<8;
      rsv_size=rsv_size+detail[14];

      fat_size=detail[37];
      fat_size=fat_size<<8;
      fat_size=fat_size+detail[36];

      fat=detail[16];

      byte_per_sec=detail[12];
      byte_per_sec=byte_per_sec<<8;
      byte_per_sec=byte_per_sec+detail[11];

      sec_per_clus=detail[13];

      root_start=rsv_size+(fat_size*fat);
      root_start=root_start*byte_per_sec;
      root_ad[0]=(root_start>>24)&0xFF;
      root_ad[1]=(root_start>>16)&0xFF;
      root_ad[2]=(root_start>>8)&0xFF;
      root_ad[3]=root_start&0xFF;

/******** Get file detail in root directory ********/

 delay_ms(50);
 mmc_deselect();
 mmc_clock(10);
 mmc_select();
 mmc_send_cmd(0x51,root_ad[0],root_ad[1],root_ad[2],root_ad[3],0xff);
 delay_ms(50);

  count=100000;
  resp[4]=0x55;
     do{
       resp[4]=spi_read(0xff);
     }while(resp[4]!=0x00 && --count>0);
     if(resp[4]==0x00){
     //blink1();
     //printf("Read root CMD response = %X\n\r" ,resp[4]);
     }

  count=100000;
     do{
       resp[2]=spi_read(0xff);
     }while(resp[2]!=0xfe && --count>0);
    /* if(resp[2]==0xfe){
     //output_high(LED1);
```

```
        printf("Data token = %X\n\r" ,resp[2]);
        }
        else{
        //output_high(LED2);
        printf("ERROR Data token = %X\n\r" ,resp[2]);
        }*/

        i=0;
        for(i=0;i<512;i++)
        {
         detail[i]=spi_read(0xff);
        }
        spi_read(0xff);
        spi_read(0xff);
        spi_read(0xff);

      /*** show on Hyper Terminal ***/

        printf("\n\r");
        m=0;
        l=0;
        for(m=0;m<512;m++)
         {
         printf("%X " ,detail[m]);
         if(l==15){
         printf("\n\r");
         l=0;
         }
         else l++;
         }
        printf("\n\r");      /*****/

        mmc_deselect();
        spi_read(0xff);
        spi_read(0xff);

numfile=0;

Getfiledetail:
  /***** find start of file *****/
  CheckFile();
  start_addr=detail[21+(numfile*32)];
  start_addr=start_addr<<8;
  start_addr=start_addr+detail[20+(numfile*32)];
  start_addr=start_addr<<8;
  start_addr=start_addr+detail[27+(numfile*32)];
  start_addr=start_addr<<8;
  start_addr=start_addr+detail[26+(numfile*32)];
  start_addr=(start_addr-0x02)*sec_per_clus*byte_per_sec;
  start_addr=start_addr+root_start;
```

```
/***** cal file size *****/
file_size=detail[63+(numfile*64)];
file_size=file_size<<8;
file_size=file_size+detail[62+(numfile*64)];
file_size=file_size<<8;
file_size=file_size+detail[61+(numfile*64)];
file_size=file_size<<8;
file_size=file_size+detail[60+(numfile*64)];

/***** sectors per file ******/
sec_total=file_size/byte_per_sec;

sec_num=0;

while(sec_num<sec_total+1){
//start_addr=start_addr+(sec_num*512);
file_ad[0]=(start_addr>>24)&0xFF;
file_ad[1]=(start_addr>>16)&0xFF;
file_ad[2]=(start_addr>>8)&0xFF;
file_ad[3]=start_addr&0xFF;

delay_ms(50);
mmc_deselect();
mmc_clock(10);
mmc_select();
mmc_send_cmd(0x51,file_ad[0],file_ad[1],file_ad[2],file_ad[3],0xff);
delay_ms(50);

count=80000;
resp[4]=0x55;
   do{
     resp[4]=spi_read(0xff);
   }while(resp[4]!=0x00 && --count>0);
   if(resp[4]==0x00){
   //blink1();
   //printf("Read file CMD response = %X\n\r" ,resp[4]);
   }
count=120000;
   do{
     resp[2]=spi_read(0xff);
   }while(resp[2]!=0xfe && --count>0);
   /*if(resp[2]==0xfe){
   blink1();
   printf("Data token = %X\n\r" ,resp[2]);
   }
   else{
   blink2();
   printf("ERROR Data token = %X\n\r" ,resp[2]);
   }*/
```

```
        i=0;
        for(i=0;i<512;i++)
        {
         data[i]=spi_read(0xff);
        }
        spi_read(0xff);
        spi_read(0xff);
        spi_read(0xff);

        /*** show on Hyper Terminal ***/
        printf("\n\r");
        m=0;
        l=0;
        for(m=0;m<512;m++)
         {
         printf("%X " ,data[m]);
         if(l==15){
         printf("\n\r");
         l=0;
         }
         else l++;
         }
        printf("\n\r");      /*****/
        //start_addr = start_addr+0x200;
        sec_num++;

        mmc_deselect();
        spi_read(0xff);
        spi_read(0xff);


    //numfile++;
    //goto Getfiledetail;

} //End Main Program
}
/*********************************************************************/
/*   Functions                                                       */
/*********************************************************************/

/*void blink1(void)
{
  output_high(led1);delay_ms(100);
  output_low(led1) ;delay_ms(100);
  output_high(led1);delay_ms(100);
  output_low(led1) ;delay_ms(100);
}
```

```
void blink2(void)
{
  output_high(led2);delay_ms(100);
  output_low(led2); delay_ms(100);
  output_high(led2);delay_ms(100);
  output_low(led2); delay_ms(100);
}*/
/*void pic_init(void)
{
  for(i=0;i<3;i++)
  {
  output_high(PIN_A0);delay_ms(100);
  output_low(PIN_A0);delay_ms(100);
  }
  //output_high(PIN_A0);
}*/

unsigned char spiget()
{
  spi_write(0xff);
  return spi_read();
}
```

File : LCD.C

```
#include <18F458.h>
#fuses HS,NOLVP,NOWDT,NOBROWNOUT,NOPROTECT
#use delay(clock=10000000)
#use RS232(BAUD=9600, BITS=8, PARITY=N, XMIT=PIN_C6, RCV=PIN_C7)
//#use fast_io(c)
#include "Arial9.c"
#include "ground.c"
#include "gclcd.h"
#include "stdlib.h"

#define COMMAND_SIZE 10
#define NUM_COMMANDS 11

#define checkcard   PIN_B1
#define ledred      PIN_E0
#define ledgreen    PIN_E1

#define MMCSD_PIN_SCL    PIN_B2 //o
#define MMCSD_PIN_SDI    PIN_C4 //i
#define MMCSD_PIN_SDO    PIN_C5 //o
#define MMCSD_PIN_SELECT  PIN_B4 //o
#define FAT32
#include <mmcsd.c>
#include "fat.c"

void nocard(void)
{
   gclcd_font();
   gclcd_color(BLACK);
   gclcd_gotoxy(10,50);  // X=start  y=line
   printf(gclcd_putc,"NO SD/MMC");
   gclcd_color(BLACK);
   gclcd_gotoxy(10,70);  // X=start  y=line
   printf(gclcd_putc,"PUT SD/MMC");
   gclcd_color(BLACK);
}
void main(void)
{
 //unsigned int8 contraste = 60;
  set_tris_a(0xFF);
  set_tris_b(0xFF);
  set_tris_c(0xFF);
  set_tris_d(0xFF);
  set_tris_e(0xFF);
  output_low(ledred);
  delay_ms(500);
  output_high(ledred);
  setup_adc_ports(NO_ANALOGS);
```

```
   gclcd_init();
   gclcd_font();

 while(1)
 {
   /*  if(input(checkcard))
   {
     nocard();
   }
   if(!input(checkcard))
   {
   i = fat_init();
   if(i)
   {
   gclcd_font();
   gclcd_color(BLACK);
   gclcd_gotoxy(10,50);  // X=start  y=line
   printf(gclcd_putc,"ERROR FAT");
   }

   gclcd_font();
   //printf(gclcd_putc,"\f");
   gclcd_color(BLACK);
   gclcd_gotoxy(10,50);  // X=start  y=line
   printf(gclcd_putc,"NO SD/MMC");
   gclcd_color(BLACK);
   //gclcd_color(GOLD);
   gclcd_gotoxy(10,70);  // X=start  y=line
   printf(gclcd_putc,"PUT SD/MMC");
   gclcd_color(BLACK);
   }*/

  printf(gclcd_putc,"\f");
  gclcd_gotoxy(5,0);  // X=start  y=line
  printf(gclcd_putc,"TEST GLCD");
  gclcd_color(RED);
  gclcd_gotoxy(5,15);
  printf(gclcd_putc,"NOKIA 6100");
  gclcd_color(BLUE);
  delay_ms(500);
  printf(gclcd_putc,"\f");
  IMGENDX = large[0];
  IMGENDY = large[1];
  gclcd_img(30,0);
  delay_ms(500);
 // printf(gclcd_putc,"\f");
}
}
```

File : LCD.h

```
////////////////////////////////////////////////////////////
///  Libreria para PCF8833 (Nokia 6100 Color Display)     ///
///              Version: 0.5                             ///
/// Basado en: http://www.apetech.de/nokia6100.php        ///
/// y librera GLC.C de CCS.                               ///
/// Usar GTP GCLCD de Lager para generar fuentes e imagenes ///
/// Ajustado para trabajar con PICs y Compilador CCS      ///
/// Por: Jaime Fernndez-Caro Belmonte jim2k2@hotmail.com  ///
/// Prohibida su utilizacin tanto parcial como total      ///
/// en diseos comerciales                                 ///
////////////////////////////////////////////////////////////

#define HIGH_COLOR    // Por defecto 64k colores
#define SCREEN_COLOR  WHITE
#define USE_GRFCN

#define GCLCDX      132
#define GCLCDY      132
#define X_START     1
#define Y_START     1
#define X_END       (131)
#define Y_END       (131)

#define SOFT_RESET    0x01
#define COLOR_8_BIT   0x02
#define COLOR_16_BIT  0x05
#define BOOSTER_ON    0x03
#define SLEEP_OUT     0x11
#define DISPLAY_ON    0x29
#define MEM_CONTROL   0x36
#define COLMOD        0x3A
#define SETCON        0x25
#define ADDRX         0x2A
#define ADDRY         0x2B
#define MEMWRITE      0x2C
#define RGBSET        0x2D


#ifndef HIGH_COLOR
  static unsigned int8 COLORRED [8] =
  {0x00, 0x03, 0x05, 0x07, 0x09, 0x0B, 0x0D, 0x0F}; // red and green
 //{0x00, 0x02, 0x04, 0x06, 0x09, 0x0B, 0x0D, 0x0F}; // red and green, original
values
  static unsigned int8 COLOR_BLUE [] =
  {0x00, 0x08, 0x0B, 0x0F}; // blue
 //{0x00, 0x04, 0x0B, 0x0F}; // blue, original values
#endif
```

```
   static unsigned int8 COLORRED [8] =
   {0x00, 0x03, 0x05, 0x07, 0x09, 0x0B, 0x0D, 0x0F};

   static unsigned int8 COLORBLUE [4] =
   {0x00, 0x08, 0x0B, 0x0F};


//#ifdef  HIGH_COLOR
#define  RGB(r,g,b)    (((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >> 3))
 // #define  RGB(r,g,b)         RGB(0x00, 0x40, 0x00)  // == tansparent, one green
color reduced !!
//#else
 // #define  RGB(r,g,b)    ((r & 0xE0) | ((g & 0xE0) >> 3) | (b >> 6))
  // #define  NONE         RGB(0x00, 0x20, 0x00)  // == tansparent, one green color
reduced !!
//#endif

#define  BLACK        RGB(0x00, 0x00, 0x00)
#define  WHITE        RGB(0xFF, 0xFF, 0xFF)
#define  RED          RGB(0xFF, 0x00, 0x00)
#define  GREEN        RGB(0x00, 0xFF, 0x00)
#define  BLUE         RGB(0x00, 0x00, 0xFF)
#define  YELLOW       RGB(0xFF, 0xFF, 0x00)
#define  MAGENTA      RGB(0xFF, 0x00, 0xFF)
#define  CYAN         RGB(0x00, 0xFF, 0xFF)
#define  GRAY         RGB(0x80, 0x80, 0x40)
#define  SILVER       RGB(0xA0, 0xA0, 0x80)
#define  GOLD         RGB(0xA0, 0xA0, 0x40)

#define  CS          PIN_D1
#define  RST         PIN_D0
#define  SCK         PIN_C3
#define  SDO         PIN_C5

//#byte   SSPSTAT  = 0b01000000
#byte   SSPCON1 =    0xFC6
#bit    SSPEN   =    SSPCON1.5

void gclcd_init(void);
void gclcd_cmd(int8 cmd);
void gclcd_data(int8 data);
void gclcd_const(unsigned int8 *buffer, unsigned int8 count);
void gclcd_cls(void);
void gclcd_setarea(int8 startx, int8 endx, int8 starty, int8 endy);
void gclcd_contrast(int8 contrast);
void gclcd_img(unsigned int8 x, unsigned int8 y);
void gclcd_font(void);
int gclcd_putc(int16 c);
void gclcd_newline(void);
```

```
void gclcd_gotoxy(unsigned int8 x, unsigned int8 y);
void gclcd_pixel(unsigned int8 x, unsigned int8 y, int16 color);
void gclcd_line(int x1, int y1, int x2, int y2, int16 color);
void gclcd_bar(int x1, int y1, int x2, int y2, int width, int16 color);
void gclcd_rect(int x1, int y1, int x2, int y2, int width, int1 fill, int16 color);
void gclcd_circle(int x, int y, int radius, int width, int1 fill, int16 color);

struct _font{
  unsigned int8 width;
  unsigned int8 height;
}font;

unsigned int32 i;
unsigned int16 color;
unsigned int8 xCoord=0;
unsigned int8 yCoord=0;
unsigned int8 fontOffset=0;

unsigned int16 IMGENDX,IMGENDY,ii;

// Secuencia de inicializacion
void gclcd_init(void)
{
  int g;
  set_tris_a(0x00);
  set_tris_c(0x00);

  delay_ms(65);

  output_bit(CS,FALSE);  // GCLCD ENABLED
  output_bit(RST,FALSE);
  delay_ms(10);
  output_bit(RST,TRUE);
  output_bit(SCK,TRUE);
  output_bit(SDO,TRUE);
  output_bit(CS,TRUE);

   gclcd_cmd(SOFT_RESET); // Software Reset
   gclcd_cmd(SLEEP_OUT);  // Sleep Out
   gclcd_cmd(DISPLAY_ON); // Display ON
   gclcd_cmd(BOOSTER_ON); // Booster ON
   gclcd_cmd(COLMOD);

#ifdef HIGH_COLOR
  gclcd_data(COLOR_16_BIT);
#else
  gclcd_data(COLOR_8_BIT);
  gclcd_cmd(RGBSET);
  gclcd_const(COLOR_RED, sizeof COLOR_RED);    // red
  gclcd_const(COLOR_RED, sizeof COLOR_RED);    // green
```

```
   gclcd_const(COLOR_BLUE, sizeof COLOR_BLUE);   // blue
 #endif

   gclcd_cmd(MEM_CONTROL);
   gclcd_data(0b01001000);
   gclcd_contrast(65);
   gclcd_cls();
 }


 // Envia comando a la gclcd
 void gclcd_cmd(int8 cmd)
 {
   output_bit(CS,FALSE);
   output_bit(SCK,FALSE);
   output_bit(SDO,FALSE);
   output_bit(SCK,TRUE);

   setup_spi(spi_master|spi_h_to_l);
   //setup_spi(SPI_MASTER|SPI_h_TO_l|SPI_XMIT_L_TO_H);
   spi_write(cmd);
   while(!spi_data_is_in());
   output_bit(CS,TRUE);
   SSPEN = 0;
 }


 // Envia dato a la gclcd
 void gclcd_data(int8 data)
 {
   output_bit(CS,FALSE);
   output_bit(SCK,FALSE);
   output_bit(SDO,TRUE);
   output_bit(SCK,TRUE);

   //setup_spi(spi_master | spi_h_to_l);
   setup_spi(SPI_MASTER|SPI_H_TO_L|SPI_XMIT_L_TO_H);
   spi_write(data);
   while(!spi_data_is_in());

   output_bit(CS,TRUE);
   SSPEN = 0;
 }


 // Envia configuracion color 8bits
 void gclcd_const(unsigned int8 *buffer, unsigned int8 count)
 {
   for (i=0; i<count; i++)
   {
```

```
      gclcd_data(buffer[i]);
    }
  }


 // Borra la gclcd con el color de fondo definido
 void gclcd_cls(void)
 {
   unsigned int16 i;

   gclcd_setarea(X_START, X_END, Y_START, Y_END);
   gclcd_cmd(MEMWRITE);
   for(i=0; i<(X_END*Y_END); i++)
   {
     #ifdef HIGH_COLOR
       gclcd_data(SCREEN_COLOR >> 8);
       gclcd_data(SCREEN_COLOR);
     #else
       gclcd_data(SCREEN_COLOR);
     #endif
   }
 }


 // Delimita el area de escritura
 void gclcd_setarea(int8 startx, int8 endx, int8 starty, int8 endy)
 {
   gclcd_cmd(ADDRX);
   gclcd_data(startx);
   gclcd_data(endx);
   gclcd_cmd(ADDRY);
   gclcd_data(starty);
   gclcd_data(endy);
   xCoord=startx;
   yCoord=starty;
 }


 // Cambia el contraste de la gclcd
 void gclcd_contrast(int8 contrast)
 {
   output_bit(CS,FALSE);
   gclcd_cmd(SETCON);
   gclcd_data(contrast);
   output_bit(CS,TRUE);
 }


 // Envia imagen a la gclcd
 void gclcd_img(unsigned int8 x, unsigned int8 y)
```

```
{
  gclcd_setarea(x,IMGENDX-1+x,y,IMGENDY-1+y);

  gclcd_cmd(MEMWRITE);
  for(ii=0; ii<(IMGENDX*IMGENDY); ii++)
  {
    #ifdef HIGH_COLOR
      gclcd_data(large[ii+2] >> 8);
      gclcd_data(large[ii+2]);
    #else
      gclcd_data(img[ii+2]);
    #endif
  }
}


// Seleccion fuente
void gclcd_font(void)
{
  font.width  = Arial9[0];
  font.height = Arial9[1];
}

// Envia caracter
int gclcd_putc(int16 c)
{
  unsigned int16 index;
  unsigned int8 i, j, w, data, skipped=1;
  unsigned int8 startY;

  startY=yCoord;

  gclcd_cmd(MEM_CONTROL);
  gclcd_data(0b01101000);

  if(c == '\n') gclcd_newline();
  if(c == '\f') gclcd_cls();
  if(c < 32) return 0;

  c -= 32;
  index = c*(font.width)*(font.height/8)+2;//+font.width;

  for(w=0; w<font.width; w++)
  {
    for(i=0; i<font.height/8; i++)
    {
      data = Arial9[index++];          // tipo de letra usado
      for(j=0; j<8; j++)
      {
        if(data&0x01)
```

```
        {
          if(skipped)
          {
            gclcd_gotoxy(xCoord, yCoord);
            gclcd_cmd(MEMWRITE);
           //  skipped=0;
          }
          #ifdef HIGH_COLOR
            gclcd_data(color >> 8);
            gclcd_data(color);
          #else
            gclcd_data(color);
          #endif
          yCoord++;
        }
        else
        {
          skipped=1;
          gclcd_gotoxy(xCoord,++yCoord);
        }
        data >>= 1;
      }
    }
    gclcd_gotoxy(++xCoord, startY);
  }
  gclcd_cmd(MEM_CONTROL);
  gclcd_data(0b01001000);
  return 0;
}


// Salto de linea
void gclcd_newline(void)
{
  if(yCoord+font.height < 128)
    gclcd_gotoxy(fontOffset, yCoord+font.height);
  else
    gclcd_gotoxy(fontOffset, 0);
}


// Posiciona el cursor en la posicion X,Y
void gclcd_gotoxy(unsigned int8 x, unsigned int8 y)
{
  gclcd_setarea(x, X_END, y, Y_END);
}


// Cambia el color a usar
void gclcd_color(unsigned int16 newColor)
```

```
{
  color = newColor;
}


// Pinta pixel de color en posicion X,Y dada
void gclcd_pixel(unsigned int8 x, unsigned int8 y, int16 color)
{
  gclcd_setarea(x, X_END, y, Y_END);
  gclcd_cmd(MEMWRITE);
  #ifdef HIGH_COLOR
    gclcd_data(color >> 8);
    gclcd_data(color);
  #else
    gclcd_data(color);
  #endif
}


#ifdef USE_GRFCN
// Pinta una linea de 1 pixel de x1,y1 a x2,y2 del color indicado
void gclcd_line(int x1, int y1, int x2, int y2, int16 color)
{
  signed int  x, y, addx, addy, dx, dy;
  signed long P;
  int i;
  dx = abs((signed int)(x2 - x1));
  dy = abs((signed int)(y2 - y1));
  x = x1;
  y = y1;

  if(x1 > x2)
    addx = -1;
  else
    addx = 1;
  if(y1 > y2)
    addy = -1;
  else
    addy = 1;

  if(dx >= dy)
  {
    P = 2*dy - dx;

    for(i=0; i<=dx; ++i)
    {
      gclcd_pixel(x,y,color);
      if(P < 0)
      {
        P += 2*dy;
```

```
        x += addx;
      }
      else
      {
        P += 2*dy - 2*dx;
        x += addx;
        y += addy;
      }
    }
  }
  else
  {
    P = 2*dx - dy;

    for(i=0; i<=dy; ++i)
    {
      gclcd_pixel(x,y,color);
      if(P < 0)
      {
        P += 2*dx;
        y += addy;
      }
      else
      {
        P += 2*dx - 2*dy;
        x += addx;
        y += addy;
      }
    }
  }
}


// Pinta una linea de los pixeles indicados de x1,y1 a x2,y2 del color dado
void gclcd_bar(int x1, int y1, int x2, int y2, int width, int16 color)
{
  signed int  x, y, addx, addy, j;
  signed long P, dx, dy, c1, c2;
  int i;

  dx = abs((signed int)(x2 - x1));
  dy = abs((signed int)(y2 - y1));
  x = x1;
  y = y1;
  c1 = -dx*x1 - dy*y1;
  c2 = -dx*x2 - dy*y2;

  if(x1 > x2)
  {
    addx = -1;
```

```
  c1 = -dx*x2 - dy*y2;
  c2 = -dx*x1 - dy*y1;
}
else
  addx = 1;
if(y1 > y2)
{
  addy = -1;
  c1 = -dx*x2 - dy*y2;
  c2 = -dx*x1 - dy*y1;
}
else
  addy = 1;

if(dx >= dy)
{
  P = 2*dy - dx;

  for(i=0; i<=dx; ++i)
  {
    for(j=-(width/2); j<width/2+width%2; ++j)
    {
      if(dx*x+dy*(y+j)+c1 >= 0 && dx*x+dy*(y+j)+c2 <=0)
        gclcd_pixel(x, y+j, color);
    }
    if(P < 0)
    {
      P += 2*dy;
      x += addx;
    }
    else
    {
      P += 2*dy - 2*dx;
      x += addx;
      y += addy;
    }
  }
}
else
{
  P = 2*dx - dy;

  for(i=0; i<=dy; ++i)
  {
    if(P < 0)
    {
      P += 2*dx;
      y += addy;
    }
    else
```

```
      {
        P += 2*dx - 2*dy;
        x += addx;
        y += addy;
      }
      for(j=-(width/2); j<width/2+width%2; ++j)
      {
        if(dx*x+dy*(y+j)+c1 >= 0 && dx*x+dy*(y+j)+c2 <=0)
          gclcd_pixel(x+j, y, color);
      }
    }
  }
}


// Pinta rectangulo con las coordenadas y color dados, si fill=0
// se pintan los bordes del rectangulo del numero de pixeles dados (width)
// si fill=1, el rectangulo es relleno, width no se tendru en cuenta.
void gclcd_rect(int x1, int y1, int x2, int y2, int width, int1 fill, int16 color)
{
  if(fill)
  {
    int y, ymax;                    // Find the y min and max
    if(y1 < y2)
    {
      y = y1;
      ymax = y2;
    }
    else
    {
      y = y2;
      ymax = y1;
    }

    for(; y<=ymax; ++y)             // Draw lines to fill the rectangle
      gclcd_line(x1, y, x2, y, color);
  }
  else
  {
    if(width)
    {
      gclcd_bar(x1, y1, x2, y1, width, color);     // Draw the 4 sides
      gclcd_bar(x1, y2, x2, y2, width, color);
      gclcd_bar(x1, y1, x1, y2, width, color);
      gclcd_bar(x2, y1, x2, y2, width, color);
    }
    else
    {
      gclcd_line(x1, y1, x2, y1, color);
      gclcd_line(x1, y2, x2, y2, color);
```

```
        gclcd_line(x1, y1, x1, y2, color);
        gclcd_line(x2, y1, x2, y2, color);
      }
    }
  }

// Pinta circunferencia cuyo centro viene dado por X,Y de radio y color dados
// si fill=0 se pinta un circulo (aun no implementado el numero de pixeles)
// contacta con jim2k2@hotmail.com si sabes la forma. gracias
void gclcd_circle(int x, int y, int radius, int width, int1 fill, int16 color)
{
  signed int a, b, P;
  a = 0;
  b = radius;
  P = 1 - radius;

  do
  {
    if(fill)
    {
      gclcd_line(x-a, y+b, x+a, y+b, color);
      gclcd_line(x-a, y-b, x+a, y-b, color);
      gclcd_line(x-b, y+a, x+b, y+a, color);
      gclcd_line(x-b, y-a, x+b, y-a, color);
    }
    else
    {
      gclcd_pixel(a+x, b+y, color);
      gclcd_pixel(b+x, a+y, color);
      gclcd_pixel(x-a, b+y, color);
      gclcd_pixel(x-b, a+y, color);
      gclcd_pixel(b+x, y-a, color);
      gclcd_pixel(a+x, y-b, color);
      gclcd_pixel(x-a, y-b, color);
      gclcd_pixel(x-b, y-a, color);
    }

    if(P < 0)
      P+= 3 + 2*a++;
    else
      P+= 5 + 2*(a++ - b--);
  } while(a <= b);
}
#endif
```

PIC18F2525/2620/4525/4620
Data Sheet

28/40/44-Pin
Enhanced Flash Microcontrollers
with 10-Bit A/D and nanoWatt Technology

MICROCHIP

PIC18F2525/2620/4525/4620

28/40/44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Power Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 2.5 µA typical
- Sleep mode current down to 100 nA typical
- Timer1 Oscillator: 1.8 µA, 32 kHz, 2V
- Watchdog Timer: 1.4 µA, 2V typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes, up to 40 MHz
- 4x Phase Lock Loop (PLL) – available for crystal and internal oscillators)
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 40 MHz
- Internal oscillator block:
  - 8 user selectable frequencies, from 31 kHz to 8 MHz
  - Provides a complete range of clock speeds from 31 kHz to 32 MHz when used with PLL
  - User tunable to compensate for frequency drift
- Secondary oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor
  - Allows for safe shutdown if peripheral clock stops

Peripheral Highlights:

- High-current sink/source 25 mA/25 mA
- Three programmable external interrupts
- Four input change interrupts
- Up to 2 Capture/Compare/PWM (CCP) modules, one with Auto-Shutdown (28-pin devices)
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart

Peripheral Highlights (Continued):

- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI™ (all 4 modes) and I^2C™ Master and Slave modes
- Enhanced Addressable USART module:
  - Supports RS-485, RS-232 and LIN 1.2
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-Wake-up on Start bit
  - Auto-Baud Detect
- 10-bit, up to 13-channel Analog-to-Digital Converter module (A/D):
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual analog comparators with input multiplexing
- Programmable 16-level High/Low-Voltage Detection (HLVD) module:
  - Supports interrupt on High/Low-Voltage Detection

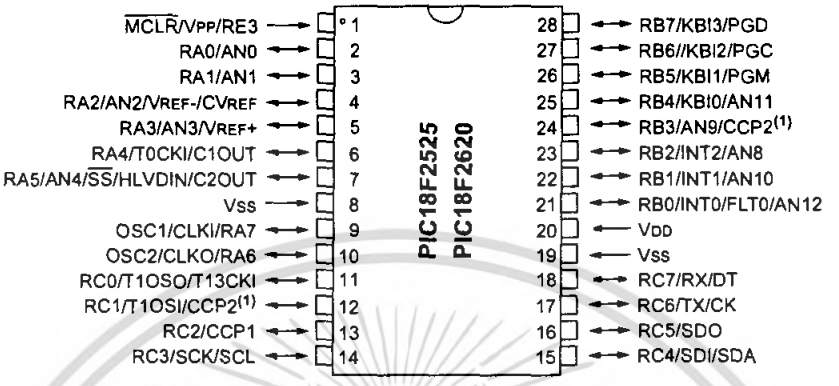
Special Microcontroller Features:

- C compiler optimized architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: 100 years typical
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Wide operating voltage range: 2.0V to 5.5V
- Programmable Brown-out Reset (BOR) with software enable option

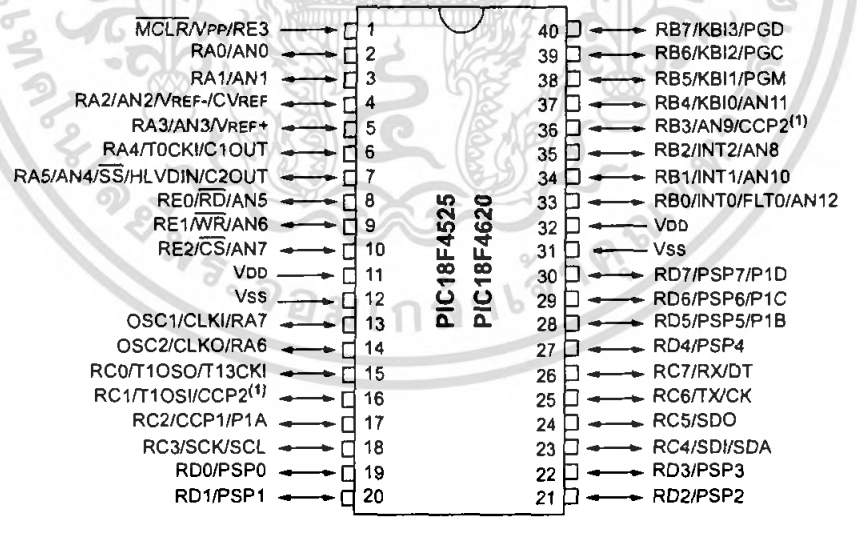
Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI™	Master I^2C™			
PIC18F2525	48K	24576	3986	1024	25	10	2/0	Y	Y	1	2	1/3
PIC18F2620	64K	32768	3986	1024	25	10	2/0	Y	Y	1	2	1/3
PIC18F4525	48K	24576	3986	1024	36	13	1/1	Y	Y	1	2	1/3
PIC18F4620	64K	32768	3986	1024	36	13	1/1	Y	Y	1	2	1/3

Pin Diagrams

28-Pin SPDIP, SOIC



40-Pin PDIP



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

TABLE 1-1: DEVICE FEATURES

Features	PIC18F2525	PIC18F2620	PIC18F4525	PIC18F4620
Operating Frequency	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz
Program Memory (Bytes)	49152	65536	49152	65536
Program Memory (Instructions)	24576	32768	24576	32768
Data Memory (Bytes)	3968	3968	3968	3968
Data EEPROM Memory (Bytes)	1024	1024	1024	1024
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	1	1
Enhanced Capture/Compare/ PWM Modules	0	0	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Parallel Communications (PSP)	No	No	Yes	Yes
10-bit Analog-to-Digital Module	10 Input Channels	10 Input Channels	13 Input Channels	13 Input Channels
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled
Packages	28-pin SPDIP 28-pin SOIC	28-pin SPDIP 28-pin SOIC	40-pin PDIP 44-pin QFN 44-pin TQFP	40-pin PDIP 44-pin QFN 44-pin TQFP

16.4.9 SETUP FOR PWM OPERATION

The following steps should be taken when configuring the ECCP module for PWM operation:

1. Configure the PWM pins, P1A and P1B (and P1C and P1D, if used), as inputs by setting the corresponding TRIS bits.
2. Set the PWM period by loading the PR2 register.
3. If Auto-Shutdown is required do the following:
   - Disable Auto-Shutdown (ECCP1AS = 0)
   - Configure source (FLT0, Comparator 1 or Comparator 2)
   - Wait for non-shutdown condition
4. Configure the ECCP module for the desired PWM mode and configuration by loading the CCP1CON register with the appropriate values:
   - Select one of the available output configurations and direction with the P1M1:P1M0 bits.
   - Select the polarities of the PWM output signals with the CCP1M3:CCP1M0 bits.
5. Set the PWM duty cycle by loading the CCPR1L register and CCP1CON<5:4> bits.
6. For Half-Bridge Output mode, set the dead-band delay by loading PWM1CON<6:0> with the appropriate value.
7. If auto-shutdown operation is required, load the ECCP1AS register:
   - Select the auto-shutdown sources using the ECCPAS2:ECCPAS0 bits.
   - Select the shutdown states of the PWM output pins using the PSSAC1:PSSAC0 and PSSBD1:PSSBD0 bits.
   - Set the ECCPASE bit (ECCP1AS<7>).
   - Configure the comparators using the CMCON register.
   - Configure the comparator inputs as analog inputs.
8. If auto-restart operation is required, set the PRSEN bit (PWM1CON<7>).
9. Configure and start TMR2:
   - Clear the TMR2 interrupt flag bit by clearing the TMR2IF bit (PIR1<1>).
   - Set the TMR2 prescale value by loading the T2CKPS bits (T2CON<1:0>).
   - Enable Timer2 by setting the TMR2ON bit (T2CON<2>).
10. Enable PWM outputs after a new PWM cycle has started:
    - Wait until TMRn overflows (TMRnIF bit is set).
    - Enable the CCP1/P1A, P1B, P1C and/or P1D pin outputs by clearing the respective TRIS bits.
    - Clear the ECCPASE bit (ECCP1AS<7>).

16.4.10 OPERATION IN POWER MANAGED MODES

In Sleep mode, all clock sources are disabled. Timer2 will not increment and the state of the module will not change. If the ECCP pin is driving a value, it will continue to drive that value. When the device wakes up, it will continue from this state. If Two-Speed Start-ups are enabled, the initial start-up frequency from INTOSC and the postscaler may not be stable immediately.

In PRI_IDLE mode, the primary clock will continue to clock the ECCP module without change. In all other power managed modes, the selected power managed mode clock will clock Timer2. Other power managed mode clocks will most likely be different than the primary clock frequency.

16.4.10.1 Operation with Fail-Safe Clock Monitor

If the Fail-Safe Clock Monitor is enabled, a clock failure will force the device into the Power Managed RC_RUN mode and the OSCFIF bit (PIR2<7>) will be set. The ECCP will then be clocked from the internal oscillator clock source, which may have a different clock frequency than the primary clock.

See the previous section for additional details.

16.4.11 EFFECTS OF A RESET

Both Power-on Reset and subsequent Resets will force all ports to Input mode and the CCP registers to their Reset states.

This forces the Enhanced CCP module to reset to a state compatible with the standard CCP module.

TABLE 16-3: REGISTERS ASSOCIATED WITH ECCP1 MODULE AND TIMER1 TO TIMER3

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	49
RCON	IPEN	SBOREN(1)	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	48
PIR1	PSPIF(2)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	52
PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	52
IPR1	PSPIP(2)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	52
PIR2	OSCFIF	CMIF	—	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF	52
PIE2	OSCFIE	CMIE	—	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE	52
IPR2	OSCFIP	CMIP	—	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP	52
TRISB	PORTB Data Direction Control Register								52
TRISC	PORTC Data Direction Control Register								52
TRISD	PORTD Data Direction Control Register								52
TMR1L	Timer1 Register Low Byte								50
TMR1H	Timer1 Register High Byte								50
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	50
TMR2	Timer2 Register								50
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	50
PR2	Timer2 Period Register								50
TMR3L	Timer3 Register Low Byte								51
TMR3H	Timer3 Register High Byte								51
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{T3SYNC}$	TMR3CS	TMR3ON	51
CCPR1L	Capture/Compare/PWM Register 1 Low Byte								51
CCPR1H	Capture/Compare/PWM Register 1 High Byte								51
CCP1CON	P1M1(2)	P1M0(2)	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	51
ECCP1AS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1(2)	PSSBD0(2)	51
PWM1CON	PRSEN	PDC6(2)	PDC5(2)	PDC4(2)	PDC3(2)	PDC2(2)	PDC1(2)	PDC0(2)	51

Legend: — = unimplemented, read as '0'. Shaded cells are not used during ECCP operation.

Note 1: The SBOREN bit is only available when the BOREN1:BOREN0 configuration bits = 01; otherwise, it is disabled and reads as '0'. See **Section 4.4 "Brown-out Reset (BOR)"**.

2: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

17.0 MASTER SYNCHRONOUS SERIAL PORT (MSSP) MODULE

17.1 Master SSP (MSSP) Module Overview

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)
  - Full Master mode
  - Slave mode (with general address call)

The I^2C interface supports the following modes in hardware:

- Master mode
- Multi-Master mode
- Slave mode

17.2 Control Registers

The MSSP module has three associated registers. These include a status register (SSPSTAT) and two control registers (SSPCON1 and SSPCON2). The use of these registers and their individual configuration bits differ significantly depending on whether the MSSP module is operated in SPI or I^2C mode.

Additional details are provided under the individual sections.

17.3 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four SPI modes are supported. To accomplish communication, typically three pins are used:

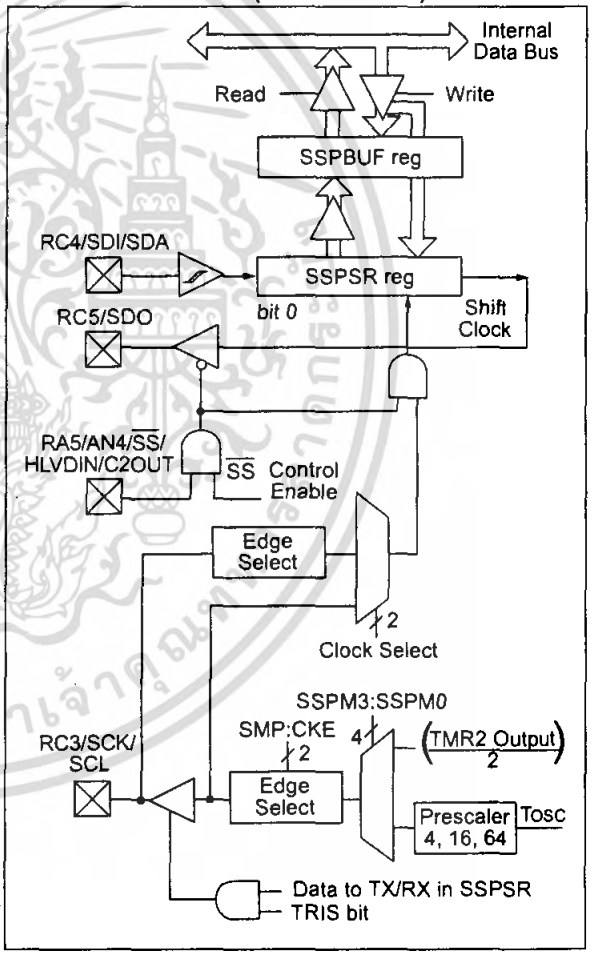
- Serial Data Out (SDO) – RC5/SDO
- Serial Data In (SDI) – RC4/SDI/SDA
- Serial Clock (SCK) – RC3/SCK/SCL

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select ($\overline{SS}$) – RA5/AN4/$\overline{SS}$/HLVDIN/C2OUT

Figure 17-1 shows the block diagram of the MSSP module when operating in SPI mode.

FIGURE 17-1: MSSP BLOCK DIAGRAM (SPI™ MODE)



17.3.1 REGISTERS

The MSSP module has four registers for SPI mode operation. These are:

- MSSP Control Register 1 (SSPCON1)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR) – Not directly accessible

SSPCON1 and SSPSTAT are the control and status registers in SPI mode operation. The SSPCON1 register is readable and writable. The lower 6 bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

REGISTER 17-1: SSPSTAT: MSSP STATUS REGISTER (SPI MODE)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in Slave mode.

bit 6 **CKE:** SPI Clock Select bit
1 = Transmit occurs on transition from active to Idle clock state
0 = Transmit occurs on transition from Idle to active clock state
Note: Polarity of clock state is set by the CKP bit (SSPCON1<4>).

bit 5 **D/$\overline{A}$:** Data/$\overline{Address}$ bit
Used in I^2C mode only.

bit 4 **P:** Stop bit
Used in I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.

bit 3 **S:** Start bit
Used in I^2C mode only.

bit 2 **R/$\overline{W}$:** Read/$\overline{Write}$ Information bit
Used in I^2C mode only.

bit 1 **UA:** Update Address bit
Used in I^2C mode only.

bit 0 **BF:** Buffer Full Status bit (Receive mode only)
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 17-2: SSPCON1: MSSP CONTROL REGISTER 1 (SPI MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit (Transmit mode only)
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit
SPI Slave mode:
1 = A new byte is received while the SSPBUF register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in Slave mode. The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow (must be cleared in software).
0 = No overflow

Note: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
1 = Enables serial port and configures SCK, SDO, SDI and $\overline{SS}$ as serial port pins
0 = Disables serial port and configures these pins as I/O port pins

Note: When enabled, these pins must be properly configured as input or output.

bit 4 **CKP:** Clock Polarity Select bit
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
0101 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin
0100 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = Fosc/64
0001 = SPI Master mode, clock = Fosc/16
0000 = SPI Master mode, clock = Fosc/4

Note: Bit combinations not specifically listed here are either reserved or implemented in I^2C mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

17.3.2 OPERATION

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPCON1<5:0> and SSPSTAT<7:6>). These control bits allow the following to be specified:

- Master mode (SCK is the clock output)
- Slave mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Data Input Sample Phase (middle or end of data output time)
- Clock Edge (output data on rising/falling edge of SCK)
- Clock Rate (Master mode only)
- Slave Select mode (Slave mode only)

The MSSP consists of a transmit/receive shift register (SSPSR) and a buffer register (SSPBUF). The SSPSR shifts the data in and out of the device, MSb first. The SSPBUF holds the data that was written to the SSPSR until the received data is ready. Once the 8 bits of data have been received, that byte is moved to the SSPBUF register. Then, the Buffer Full detect bit, BF (SSPSTAT<0>) and the interrupt flag bit, SSPIF, are set. This double-buffering of the received data (SSPBUF) allows the next byte to start reception before reading the data that was just received. Any write to the SSPBUF register during transmission/reception of data will be ignored and the write collision detect bit, WCOL (SSPCON1<7>), will be set. User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPBUF register completed successfully.

When the application software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF. The Buffer Full bit, BF (SSPSTAT<0>), indicates when SSPBUF has been loaded with the received data (transmission is complete). When the SSPBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter. Generally, the MSSP interrupt is used to determine when the transmission/reception has completed. The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur. Example 17-1 shows the loading of the SSPBUF (SSPSR) for data transmission.

The SSPSR is not directly readable or writable and can only be accessed by addressing the SSPBUF register. Additionally, the MSSP status register (SSPSTAT) indicates the various status conditions.

EXAMPLE 17-1: LOADING THE SSPBUF (SSPSR) REGISTER

```
LOOP    BTFSS   SSPSTAT, BF     ;Has data been received (transmit complete)?
        BRA     LOOP            ;No
        MOVF    SSPBUF, W       ;WREG reg = contents of SSPBUF
        MOVWF   RXDATA          ;Save in user RAM, if data is meaningful
        MOVF    TXDATA, W       ;W reg = contents of TXDATA
        MOVWF   SSPBUF          ;New data to xmit
```

17.3.3 ENABLING SPI I/O

To enable the serial port, SSP Enable bit, SSPEN (SSPCON1<5>), must be set. To reset or reconfigure SPI mode, clear the SSPEN bit, reinitialize the SSPCON registers and then set the SSPEN bit. This configures the SDI, SDO, SCK and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed as follows:

- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> bit cleared
- SCK (Master mode) must have TRISC<3> bit cleared
- SCK (Slave mode) must have TRISC<3> bit set
- $\overline{SS}$ must have TRISA<5> bit set

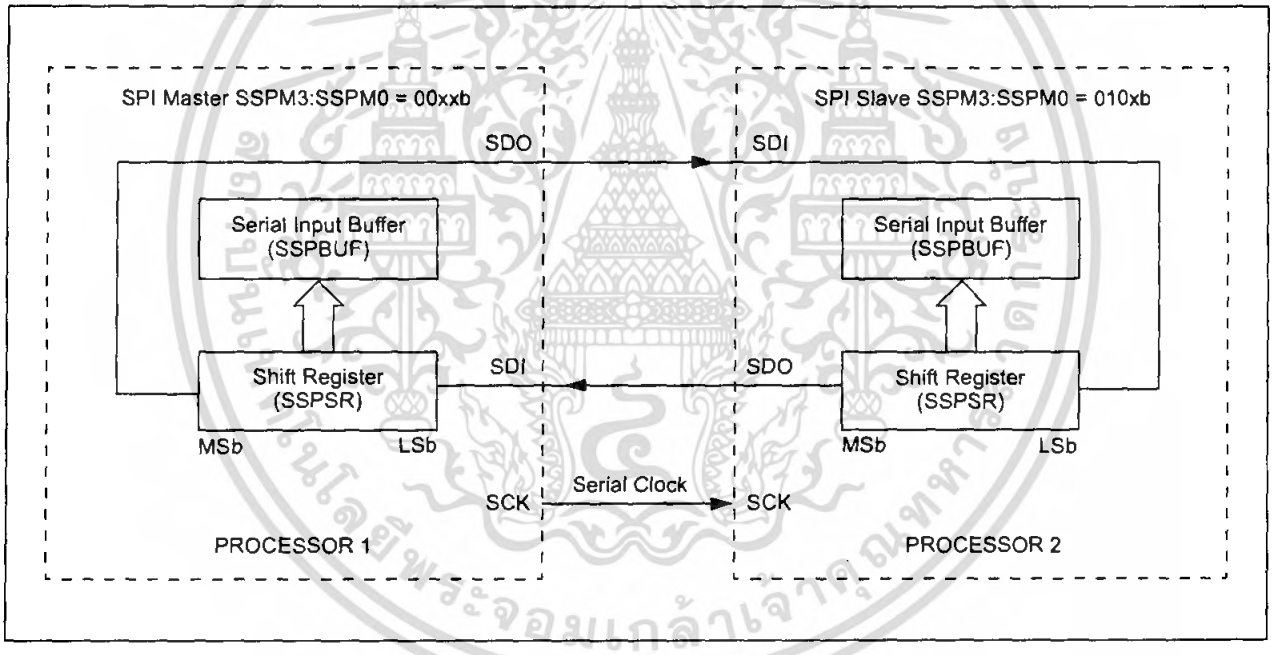
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

17.3.4 TYPICAL CONNECTION

Figure 17-2 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. Data is shifted out of both shift registers on their programmed clock edge and latched on the opposite edge of the clock. Both processors should be programmed to the same Clock Polarity (CKP), then both controllers would send and receive data at the same time. Whether the data is meaningful (or dummy data) depends on the application software. This leads to three scenarios for data transmission:

- Master sends data – Slave sends dummy data
- Master sends data – Slave sends data
- Master sends dummy data – Slave sends data

FIGURE 17-2: SPI™ MASTER/SLAVE CONNECTION



17.3.5 MASTER MODE

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2, Figure 17-2) is to broadcast data by the software protocol.

In Master mode, the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI operation is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a "Line Activity Monitor" mode.

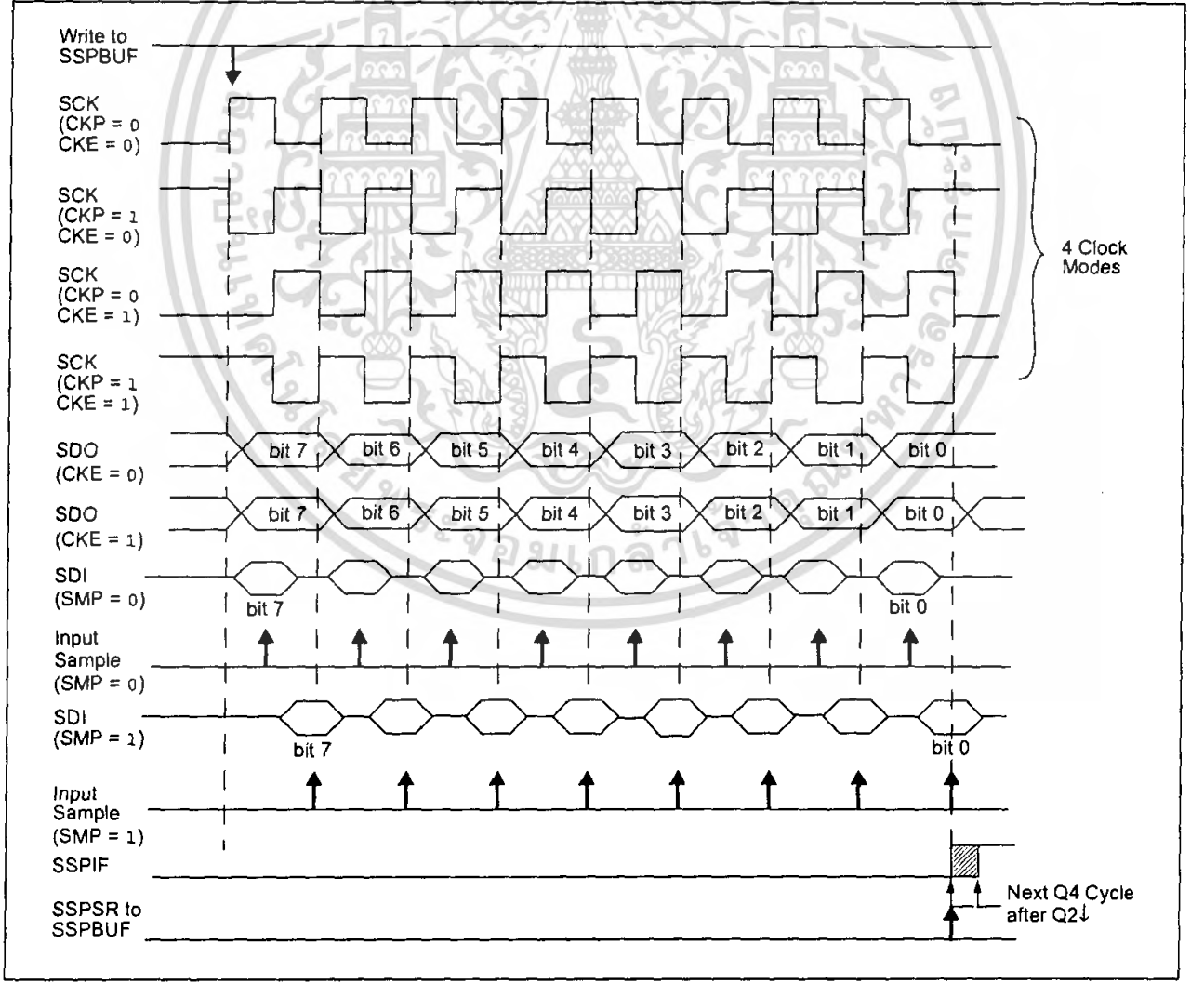
The clock polarity is selected by appropriately programming the CKP bit (SSPCON1<4>). This then, would give waveforms for SPI communication as shown in Figure 17-3, Figure 17-5 and Figure 17-6, where the MSB is transmitted first. In Master mode, the SPI clock rate (bit rate) is user programmable to be one of the following:

- Fosc/4 (or Tcy)
- Fosc/16 (or 4 • Tcy)
- Fosc/64 (or 16 • Tcy)
- Timer2 output/2

This allows a maximum data rate (at 40 MHz) of 10.00 Mbps.

Figure 17-3 shows the waveforms for Master mode. When the CKE bit is set, the SDO data is valid before there is a clock edge on SCK. The change of the input sample is shown based on the state of the SMP bit. The time when the SSPBUF is loaded with the received data is shown.

FIGURE 17-3: SPI™ MODE WAVEFORM (MASTER MODE)



26.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings(†)

Ambient temperature under bias ..... -40°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on any pin with respect to VSS (except VDD and $\overline{MCLR}$) ..... -0.3V to (VDD + 0.3V)
Voltage on VDD with respect to VSS ..... -0.3V to +7.5V
Voltage on $\overline{MCLR}$ with respect to VSS **(Note 2)** ..... 0V to +13.25V
Total power dissipation **(Note 1)** 1.0W
Maximum current out of VSS pin ..... 300 mA
Maximum current into VDD pin ..... 250 mA
Input clamp current, IIK (VI < 0 or VI > VDD) ..... ±20 mA
Output clamp current, IOK (VO < 0 or VO > VDD) ..... ±20 mA
Maximum output current sunk by any I/O pin ..... 25 mA
Maximum output current sourced by any I/O pin ..... 25 mA
Maximum current sunk by all ports ..... 200 mA
Maximum current sourced by all ports ..... 200 mA

Note 1: Power dissipation is calculated as follows:
$Pdis = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD} - V_{OH}) \times I_{OH}\} + \sum(V_{OL} \times I_{OL})$

2: Voltage spikes below VSS at the $\overline{MCLR}$/VPP/RE3 pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{MCLR}$/VPP/RE3 pin, rather than pulling this pin directly to VSS.

† **NOTICE:** Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

FIGURE 26-1: PIC18F2525/2620/4525/4620 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)

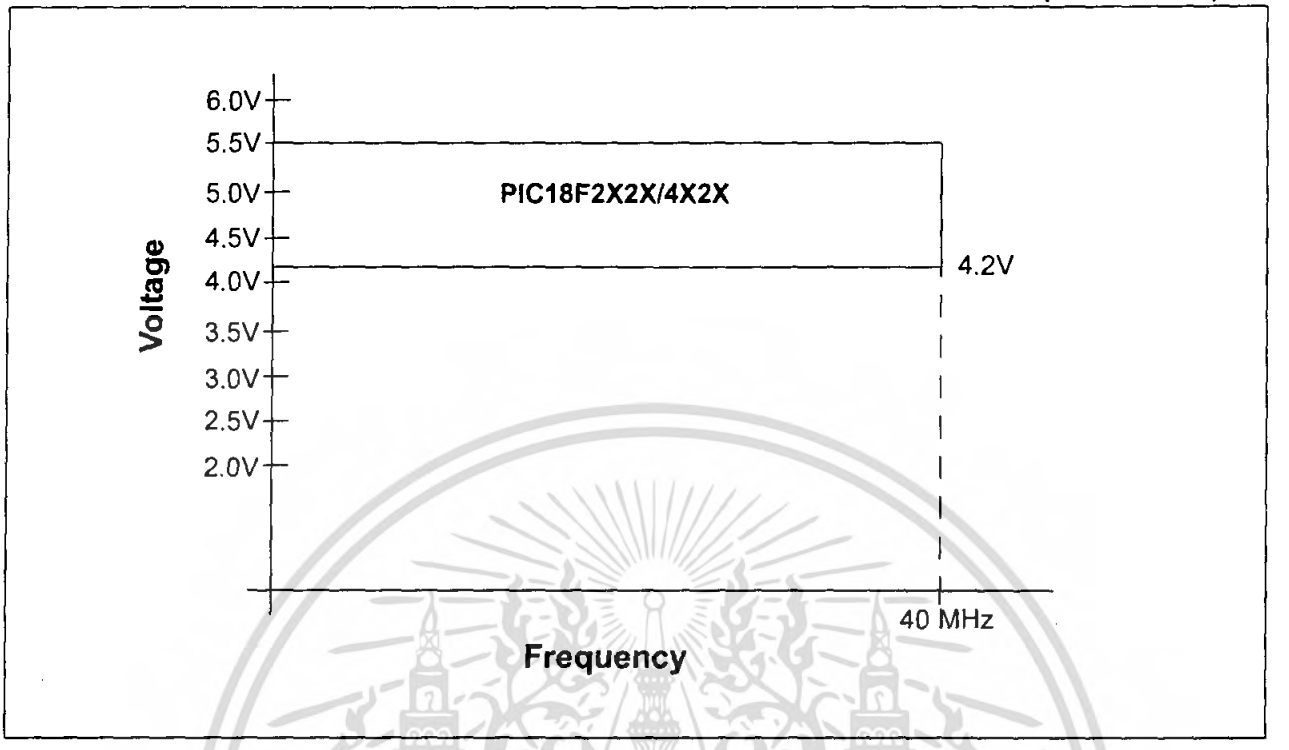


FIGURE 26-2: PIC18F2220/2320/4220/4320 VOLTAGE-FREQUENCY GRAPH (EXTENDED)

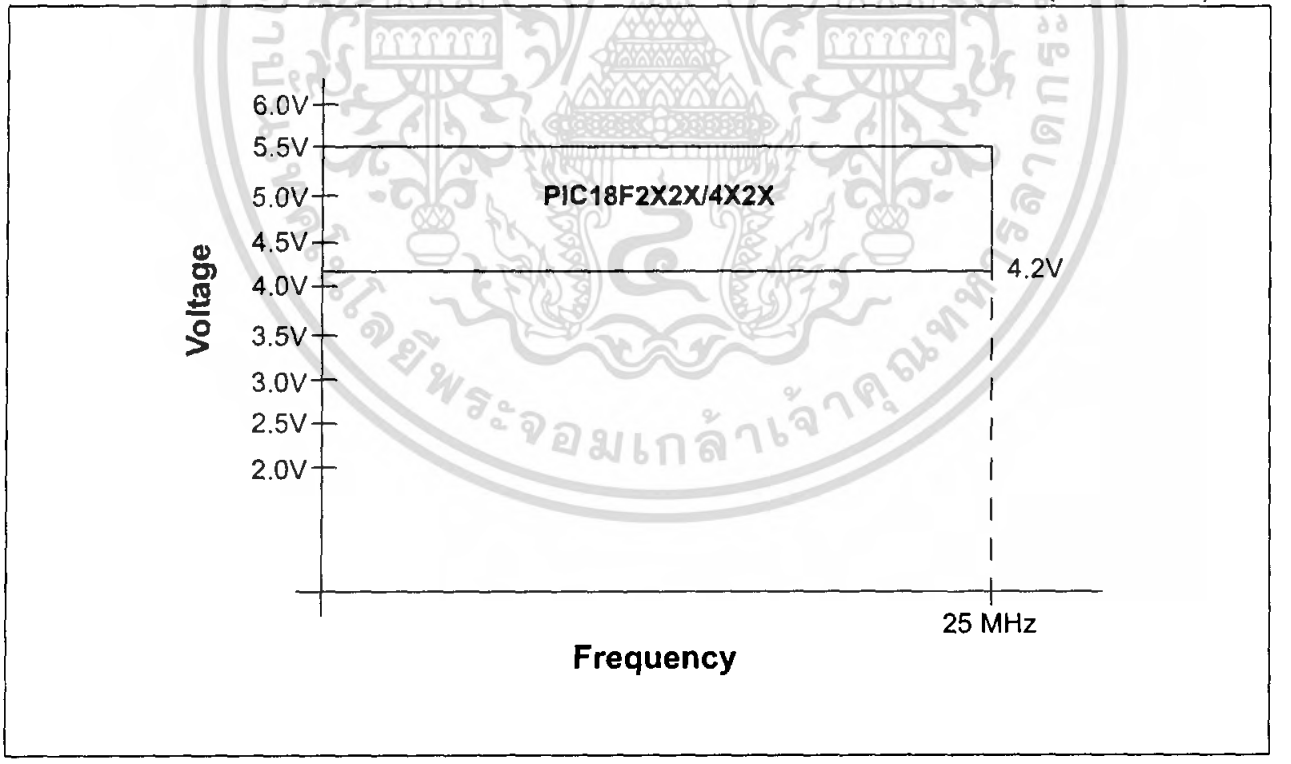
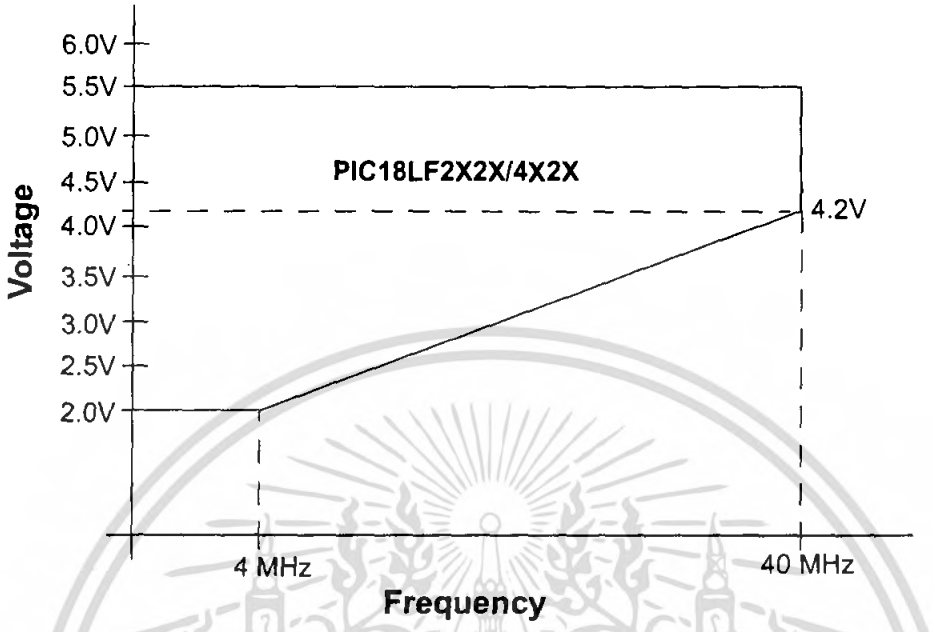


FIGURE 26-3: PIC18LF2525/2620/4525/4620 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)



FMAX = (16.36 MHz/V) (VDDAPPMIN – 2.0V) + 4 MHz

Note: VDDAPPMIN is the minimum voltage of the PICmicro® device in the application.

DATA SHEET

PCF8833
STN RGB - 132 × 132 × 3 driver

Objective specification

2003 Feb 14

Philips Semiconductors

PHILIPS

CONTENTS

STN RGB - 132 × 132 × 3 driver PCF8833

1 FEATURES

- Single chip LCD controller and driver
- 132 rows and 396 column outputs (132 × RGB)
- Low cross talk by Frame Rate Control (FRC)
- 4 kbyte colours (RGB) = 4 : 4 : 4 mode
- 256 colours (RGB) = 3 : 3 : 2 mode using the 209 kbit RAM and a Look-Up Table (LUT)
- 65 kbyte colours (RGB) = 5 : 6 : 5 mode using the 209 kbit RAM with dithering
- 8 colours Power-save mode
- Display data RAM 132 × 132 (RGB) (4 kbyte colour)
- Interfaces:
  - 3-line serial interface
  - 8-bit 8080 Intel CPU interface.
- Display features:
  - Area scrolling
  - 32-line partial Display mode
  - Software programmable colour depth mode
  - N-line inversion for low cross talk.
- On-chip:
  - Oscillator for display system, requires no external components (external clock also possible)
  - Generation of V_{LCD}
  - Segmented temperature compensation of V_{LCD} and frame frequency.
- Logic supply voltage range V_{DD1} to V_{SS1}:
  - 1.5 to 3.3 V.
- Analog supply voltage range for V_{LCD} generation V_{DD2} to V_{SS2}:
  - 2.4 to 4.5 V.
- Analog supply voltage range for reference voltage generation V_{DD3} to V_{SS1}:
  - 2.4 to 3.5 V.
- Display supply voltage range V_{LCD} to V_{SS1}:
  - 3.8 to 20 V.
- Low power consumption; suitable for battery operated systems
- CMOS compatible inputs
- Manufactured in silicon gate CMOS process
- Optimized layout for COF, Chip On Glass (COG) and Transformer Coupled Plasma (TCP) assembly.

2 GENERAL DESCRIPTION

The PCF8833 is a single chip low power CMOS LCD controller driver, designed to drive colour Super-Twisted Nematic (STN) displays of 132 rows and 132 RGB columns. All necessary functions for the display are provided in a single chip, including display RAM which has a capacity of 209 kbit (132 × 12-bit × 132). The PCF8833 uses the Multiple Row Addressing (MRA) driving technique in order to achieve the best optical performance at the lowest power consumption. The PCF8833 offers 2 types of microcontroller interfaces namely the 8080 system interface and the 3-line serial interface.

3 ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
PCF8833U/2DA/1	–	chip with bumps in tray	–

4 BLOCK DIAGRAM

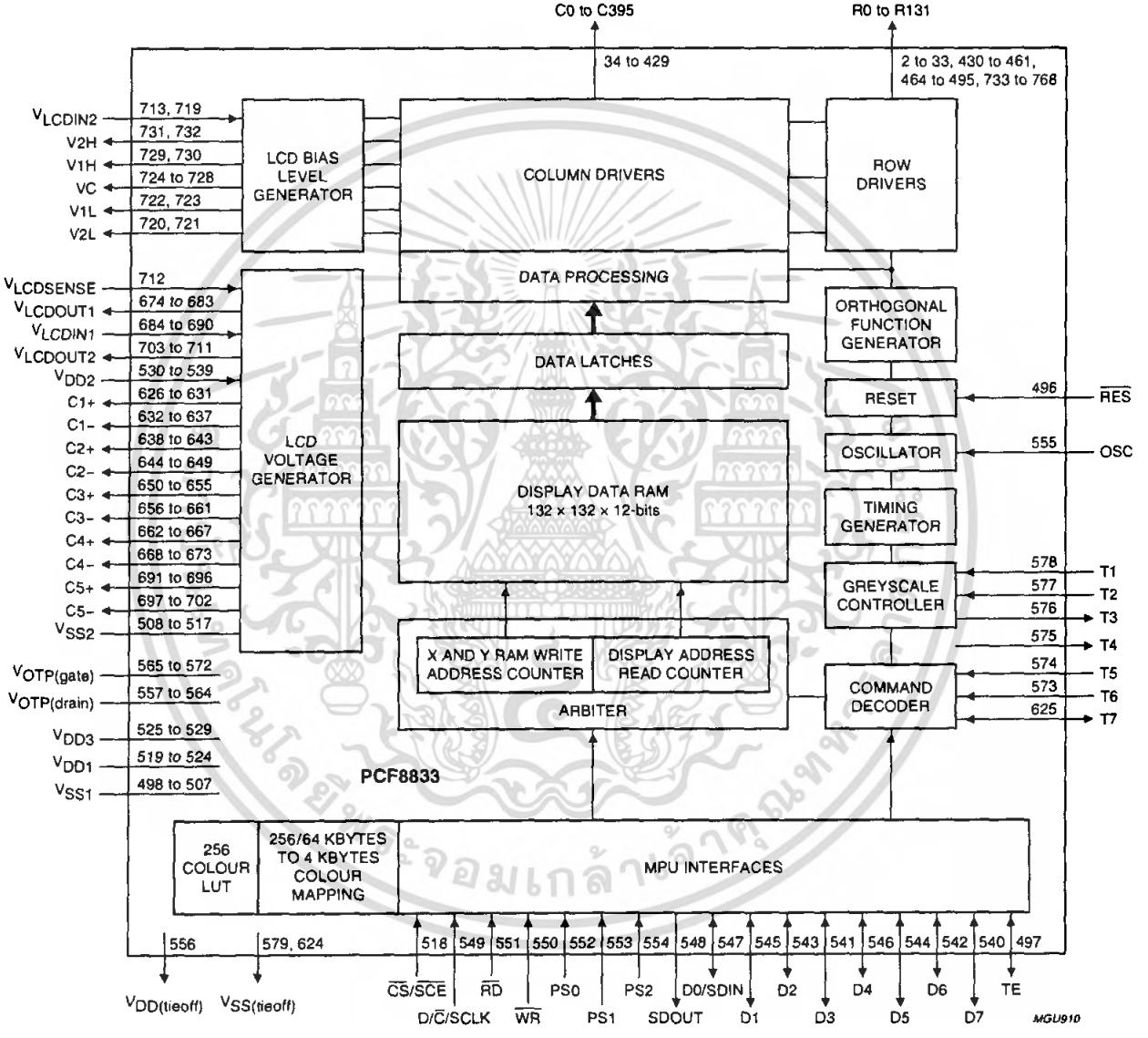


Fig.1 Block diagram.

5 PINNING

SYMBOL	PAD	TYPE	DESCRIPTION
R95 to R64	2 to 33	O	LCD row driver outputs
C0 to C395	34 to 429	O	LCD column driver outputs
R0 to R31	430 to 461	O	LCD row driver outputs
R63 to R32	464 to 495	O	LCD row driver outputs
$\overline{RES}$	496	I	external reset; this signal will reset the device and must be applied to properly initialize the chip (active LOW)
TE	497	O/I	tearing line (in Normal mode it is always an output)
V_{SS1}	498 to 507	PS	system ground
V_{SS2}	508 to 517	PS	system ground
$\overline{CS}/\overline{SCE}$	518	I	chip select parallel interface or serial chip enable (active LOW)
V_{DD1}	519 to 524	PS	logic supply voltage
V_{DD3}	525 to 529	PS	V_{DD2} and V_{DD3} are the supply voltage pins for the internal voltage generator including the temperature compensation circuits; V_{DD2} and V_{DD3} can be connected together but in this case care must be taken to respect the supply voltage range (see Chapter 13); V_{DD1} is used as the supply for the rest of the chip. V_{DD1} can be connected together with V_{DD2} and V_{DD3} but in this case care must also be taken to respect the supply voltage range; see Chapter 13. V_{DD2} and V_{DD3} must not be applied before V_{DD1}. If the internal voltage generator is not used, pins V_{DD2} and V_{DD3} must be connected to V_{DD1}.
V_{DD2}	530 to 539	PS	
D7	540	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D3	541	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D6	542	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D2	543	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D5	544	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D1	545	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D4	546	I/O	8-bit parallel data; in Serial mode tie to V_{SS1} or V_{DD1}
D0/SDIN	547	I/O	8-bit parallel data or serial data input
SDOUT	548	O	serial data output; in Parallel mode tie to V_{DD1}, V_{SS1} or D0
D/$\overline{C}$/SCLK	549	I	data/$\overline{command}$ indicator parallel interface or serial clock
$\overline{WR}$	550	I	$\overline{write\ clock}$ parallel interface; in Serial mode tie to V_{DD1} (active LOW)
$\overline{RD}$	551	I	$\overline{read\ clock}$ parallel interface; in Serial mode tie to V_{DD1} (active LOW)
PS0	552	I	set serial or parallel interface mode PS1 and PS2 must tied to either V_{SS1} or V_{DD1}
PS1	553	I	set serial or parallel interface mode PS1 and PS2 must tied to either V_{SS1} or V_{DD1}
PS2	554	I	set serial or parallel interface mode PS1 and PS2 must tied to either V_{SS1} or V_{DD1}

SYMBOL	PAD	TYPE	DESCRIPTION
OSC	555	I	oscillator input or external oscillator resistor connection; when the on-chip oscillator is used this input must be connected to V_{DD1}; an external clock signal, if used, is connected to this input and the internal oscillator must be switched off with a software command; if the oscillator and external clock are all inhibited by connecting pin OSC to V_{SS1}, the display is not clocked and may be left in a DC state; to avoid this the chip should always be put into Power-down mode before stopping the clock.
$V_{DD(tieoff)}$	556	O	can be used to tie inputs to V_{DD1}
$V_{OTP(drain)}$	557 to 564	PS	supply voltage for OTP programming (write voltage), in Application mode must be tied to V_{SS1} or left open-circuit
$V_{OTP(gate)}$	565 to 572	PS	supply voltage for OTP programming, in Application mode must be tied to V_{SS1} or left open-circuit
T6	573	I	test pin; not accessible to user; must be connected to V_{SS1}
T5	574	I	test pin; not accessible to user; must be connected to V_{SS1}
T4	575	O	test pin; not accessible to user; must be left open-circuit
T3	576	O	test pin; not accessible to user; must be left open-circuit
T2	577	I/O	test pin; not accessible to user; must be also connected to V_{SS1}
T1	578	I/O	test pin; not accessible to user; must be also connected to V_{SS1}
$V_{SS(tieoff)}$	579	O	can be used to tie inputs to V_{SS1}
$V_{SS(tieoff)}$	624	O	can be used to tie inputs to V_{SS1}
T7	625	I/O	test pin; not accessible to user; must be connected to V_{SS1}
C1+	626 to 631	I	positive input pump capacitor voltage multiplier 1
C1–	632 to 637	I	negative input pump capacitor voltage multiplier 1
C2+	638 to 643	I	positive input pump capacitor voltage multiplier 1
C2–	644 to 649	I	negative input pump capacitor voltage multiplier 1
C3+	650 to 655	I	positive input pump capacitor voltage multiplier 1
C3–	656 to 661	I	negative input pump capacitor voltage multiplier 1
C4+	662 to 667	I	positive input pump capacitor voltage multiplier 1
C4–	668 to 673	I	negative input pump capacitor voltage multiplier 1
$V_{LCDOUT1}$	674 to 683	O	output voltage multiplier 1
V_{LCDIN1}	684 to 690	PS	LCD supply input voltage 1
C5+	691 to 696	I	positive input pump capacitor voltage multiplier 2
C5–	697 to 702	I	negative input pump capacitor voltage multiplier 2
$V_{LCDOUT2}$	703 to 711	O	output voltage multiplier 2
$V_{LCDSENSE}$	712	I	voltage multiplier regulation input; must be connected to $V_{LCDOUT2}$
V_{LCDIN2}	713 to 719	PS	LCD supply input voltage 2
V2L	720, 721	O	LCD bias level
V1L	722, 723	O	LCD bias level
VC	724 to 728	O	LCD bias level
V1H	729, 730	O	LCD bias level

SYMBOL	PAD	TYPE	DESCRIPTION
V2H	731, 732	O	LCD bias level
R96 to R131	733 to 768	O	LCD row driver outputs
Dummy	1, 462, 463, 580 to 623, 769		

6 INSTRUCTIONS

The PCF8833 communicates with the host using an 8-bit parallel interface or a 3-line serial interface. Processing of instructions and data sent to the interface do not require the display clock. The display clock and interface clock are independent from each other. The display clock is derived from the built-in oscillator.

The PCF8833 has 2 types of accesses; those defining the operating mode of the device (instructions) and those filling the display RAM. Since writing to the RAM occurs more frequently, efficient data transfer is achieved by autoincrementing the RAM address pointers.

There are 3 types of instructions:

1. For defining display configuration
2. For setting X and Y addresses
3. Miscellaneous.

Commands in the range of 00H to AFH not defined in Table 1 and command DDH have the same effect as no operation (NOP).

All commands in range B0H to B9H and DEH to FFH are forbidden.

Table 1 Command table; note 1

D/$\overline{C}$	7	6	5	4	3	2	1	0	DEFAULT	OTP	DESCI
0	0	0	0	0	0	0	0	0	00H	–	no operation (N
0	0	0	0	0	0	0	0	1	01H	–	software reset (
0	0	0	0	0	0	0	1	0	02H	–	booster voltage
0	0	0	0	0	0	0	1	1	03H	–	booster voltage
0	0	0	0	0	0	1	0	0	04H	–	read display ide (RDDIDIF)
0	0	0	0	0	1	0	0	1	09H	–	read display sta
0	0	0	0	1	0	0	0	0	10H	–	Sleep_IN
0	0	0	0	1	0	0	0	1	11H	–	Sleep_OUT
0	0	0	0	1	0	0	1	0	12H	–	Partial mode or
0	0	0	0	1	0	0	1	1	13H	–	normal Display (NORON)
0	0	0	1	0	0	0	0	0	20H	–	display inversio
0	0	0	1	0	0	0	0	1	21H	–	display inversio
0	0	0	1	0	0	0	1	0	22H	–	all pixel off (DA
0	0	0	1	0	0	0	1	1	23H	–	all pixel on (DA
0	0	0	1	0	0	1	0	1	25H	–	set contrast (SE
1	X	$VCON_6$	$VCON_5$	$VCON_4$	$VCON_3$	$VCON_2$	$VCON_1$	$VCON_0$	00H	–	set contrast
0	0	0	1	0	1	0	0	0	28H	–	display off (DIS
0	0	0	1	0	1	0	0	1	29H	–	display on (DIS
0	0	0	1	0	1	0	1	0	2AH	–	column addres
1	xs[7]	xs[6]	xs[5]	xs[4]	xs[3]	xs[2]	xs[1]	xs[0]	02H	–	X address star
1	xe[7]	xe[6]	xe[5]	xe[4]	xe[3]	xe[2]	xe[1]	xe[0]	81H	–	X address end
0	0	0	1	0	1	0	1	1	2BH	–	page address :
1	ys[7]	ys[6]	ys[5]	ys[4]	ys[3]	ys[2]	ys[1]	ys[0]	02H	–	Y address star
1	ye[7]	ye[6]	ye[5]	ye[4]	ye[3]	ye[2]	ye[1]	ye[0]	81H	–	Y address end
0	0	0	1	0	1	1	0	0	2CH	–	memory write
1	D7	D6	D5	D4	D3	D2	D1	D0	XXH	–	write data
0	0	0	1	0	1	1	0	1	2DH	–	colour set (RG
1	X	X	X	X	R3	R2	R1	R0	00H	–	red tone 000
1	6 bytes for 6 red tones									–	6 red tones

D/$\overline{C}$	7	6	5	4	3	2	1	0	DEFAULT	OTP	DESCI
1	X	X	X	X	R3	R2	R1	R0	0FH	–	red tone 111
1	X	X	X	X	G3	G2	G1	G0	00H	–	green tone 000
1	6 bytes for 6 green tones									–	6 green tones
1	X	X	X	X	G3	G2	G1	G0	0FH	–	green tone 111
1	X	X	X	X	B3	B2	B1	B0	00H	–	blue tone 00
1	2 bytes for 2 blue tones									–	2 blue tones
1	X	X	X	X	B3	B2	B1	B0	0FH	–	blue tone 11
0	0	0	1	1	0	0	0	0	30H	–	partial area (PT
1	AA1S7	AA1S6	AA1S5	AA1S4	AA1S3	AA1S2	AA1S1	AA1S0	00H	–	PTLAR active a
1	AA1E7	AA1E6	AA1E5	AA1E4	AA1E3	AA1E2	AA1E1	AA1E1	1FH	–	PTLAR active a
0	0	0	1	1	0	0	1	1	33H	–	vertical scroll d (VSCRDEF)
1	TF_7	TF_6	TF_5	TF_4	TF_3	TF_2	TF_1	TF_0	00H	–	top fixed area
1	SA_7	SA_6	SA_5	SA_4	SA_3	SA_2	SA_1	SA_0	82H	–	scroll area
1	BF_7	BF_6	BF_5	BF_4	BF_3	BF_2	BF_1	BF_0	00H	–	bottom fixed ar
0	0	0	1	1	0	1	0	0	34H	–	tearing line off
0	0	0	1	1	0	1	0	1	35H	–	tearing line on
1	X	X	X	X	X	X	X	X	00H	–	
0	0	0	1	1	0	1	1	0	36H	–	memory data a (MADCTL)
1	MY	MX	V	LAO	RGB	X	X	X	00H	–	RAM data addr control
0	0	0	1	1	0	1	1	1	37H	–	set Scroll Entry
1	SEP7	SEP6	SEP5	SEP4	SEP3	SEP2	SEP1	SEP0	00H	–	scroll entry poi
0	0	0	1	1	1	0	0	0	38H	–	Idle mode off (I
0	0	0	1	1	1	0	0	1	39H	–	Idle mode on (I
0	0	0	1	1	1	0	1	0	3AH	–	interface pixel f
1	X	X	X	X	X	P2	P1	P0	03H	–	colour interface
0	1	0	1	1	0	0	0	0	B0H	x (2)	set V_{OP} (SETV
1	X	X	X	X	VPR_8	VPR_7	VPR_6	VPR_5	08H	x	V_{OP}
1	X	X	X	VPR_4	VPR_3	VPR_2	VPR_1	VPR_0	01H	x	V_{OP}
0	1	0	1	1	0	1	0	BRS	B4H	x	Bottom Row S

D/$\overline{C}$	7	6	5	4	3	2	1	0	DEFAULT	OTP	DESCI
0	1	0	1	1	0	1	1	TRS	B6H	x	Top Row Swap
0	1	0	1	1	1	0	0	FINV	B9H	–	super Frame IN
0	1	0	1	1	1	0	1	DOR	BAH	–	Data ORder (D
0	1	0	1	1	1	1	0	TCDFE	BDH	–	enable/disable (TCDFE)
0	1	0	1	1	1	1	1	TCVOPE	BFH	–	enable or disab comp (TCVOPE
0	1	1	0	0	0	0	0	EC	C0H	–	Internal or exte (EC)
0	1	1	0	0	0	0	1	0	C2H	x	set multiplicatio (SETMUL)
1	X	X	X	X	X	X	S1	S0	03H	x	multiplication fa
0	1	1	0	0	0	0	1	1	C3H	x	set TCVOP slo (TCVOPAB)
1	X	SLB_2	SLB_1	SLB_0	X	SLA_2	SLA_1	SLA_0	34H	x	
0	1	1	0	0	0	1	0	0	C4H	x	set TCVOP slo (TCVOPCD)
1	X	SLD_2	SLD_1	SLD_0	X	SLC_2	SLC_1	SLC_0	75H	x	
0	1	1	0	0	0	1	0	1	C5H	x	set divider frequ
1	X	DFA_6	DFA_5	DFA_4	DFA_3	DFA_2	DFA_1	DFA_0	56H	x	set divider factc
1	X	DFB_6	DFB_5	DFB_4	DFB_3	DFB_2	DFB_1	DFB_0	35H	x	set divider factc
1	X	DFC_6	DFC_5	DFC_4	DFC_3	DFC_2	DFC_1	DFC_0	30H	x	set divider factc
1	X	DFD_6	DFD_5	DFD_4	DFD_3	DFD_2	DFD_1	DFD_0	25H	x	set divider factc
0	1	1	0	0	0	1	1	0	C6H	x	set divider frequ mode (DF8colo
1	X	$DF8_6$	$DF8_5$	$DF8_4$	$DF8_3$	$DF8_2$	$DF8_1$	$DF8_0$	35H	x	set divider factc mode
0	1	1	0	0	0	1	1	1	C7H	x	set bias system
1	X	X	X	X	VB_3	VB_2	VB_1	VB_0	0BH	x	bias systems
0	1	1	0	0	1	0	0	0	C8H	–	temperature re (RDTEMP)
0	1	1	0	0	1	0	0	1	C9H	–	N-Line Inversic
1	NLI_7	NLI_6	NLI_5	NLI_4	NLI_3	NLI_2	NLI_1	NLI_0	13H	x	after NLI time s
0	1	1	0	1	1	0	1	0	DAH	x	read ID1 (RDID

D/$\overline{C}$	7	6	5	4	3	2	1	0	DEFAULT	OTP	DESCI
0	1	1	0	1	1	0	1	1	DBH	x	read ID2 (RDID
0	1	1	0	1	1	1	0	0	DCH	x	read ID3 (RDID
0	1	1	1	0	1	1	1	SFD	EFH	x	select factory d
0	1	1	1	1	0	0	0	0	F0H	–	enter Calibratio
1	X	X	ORA_2	ORA_1	ORA_0	X	OPE	CALMM	00H	–	set calibration c
0	1	1	1	1	0	0	0	1	F1H	–	shift data in OT (OTPSHTIN)
1	OS7	OS6	OS5	OS4	OS3	OS2	OS1	OS0	XX	–	multiple data by of bytes allowe

Notes

1. X = don't care.
2. This function can be set by OTP.
3. If the OTP bit Enable Factory Defaults (EFD) has been programmed to logic 1 (default value is logic 0), then the Se instruction is ignored and the device will always use the OTP default data.

9 SERIAL INTERFACE

Communication with the microcontroller can also occur via a clock-synchronized serial peripheral interface. The selection of this interface is achieved with pin PS0; see Section 7.1.1.

The serial interface is a 3-line bidirectional interface for communication between the microcontroller and the LCD driver chip. The 3 lines are chip enable ($\overline{SCE}$), Serial Clock (SCLK) and Serial Data (SD). The PCF8833 is connected to the SD pin of the microcontroller by two pins SDIN (data input) and SDOUT (data output) which are connected together.

9.1 Write mode

The Write mode of the interface means that the microcontroller writes commands and data to the PCF8833. Each data packet contains a control bit D/$\overline{C}$ and a transmission byte. If bit D/$\overline{C}$ is logic 0, the following byte is interpreted as a command byte. The command set is given in Table 1. If bit D/$\overline{C}$ is logic 1, the following bytes are stored in the display data RAM or registers. After every RAM data byte the address counter increments automatically. Figure 42 shows the general format of the Write mode and the definition of the transmission byte.

Any instruction can be sent in any order to the PCF8833; the MSB is transmitted first. The serial interface is initialized when $\overline{SCE}$ is HIGH. In this state, SCLK pulses have no effect and no power is consumed by the serial interface. A falling edge on pin $\overline{SCE}$ enables the serial interface and indicates the start of data transmission.

Figure 42 shows the protocol of the Write mode:

- When $\overline{SCE}$ is HIGH, SCLKs are ignored. The serial interface is initialized during the HIGH time of $\overline{SCE}$.
- At the falling edge of $\overline{SCE}$ SCLK must be LOW (see Fig.51)
- SDIN is sampled at the rising edge of SCLK
- D/$\overline{C}$ indicates, whether the byte is a command (D/$\overline{C}$ = 0) or data (D/$\overline{C}$ = 1). It is sampled with the first rising SCLK edge.
- If $\overline{SCE}$ stays LOW after the last bit of a data/$\overline{command}$ byte, the serial interface will receive the D/$\overline{C}$ bit of the next byte at the next rising edge of SCLK (see Fig.43).
- A reset pulse at pin $\overline{RES}$ interrupts the transmission. The data being written into the RAM may be corrupted. The registers are cleared. If $\overline{SCE}$ is LOW after the rising edge of $\overline{RES}$, the serial interface is ready to receive the D/$\overline{C}$ bit of a data/$\overline{command}$ byte; see Figs 44 and 50.

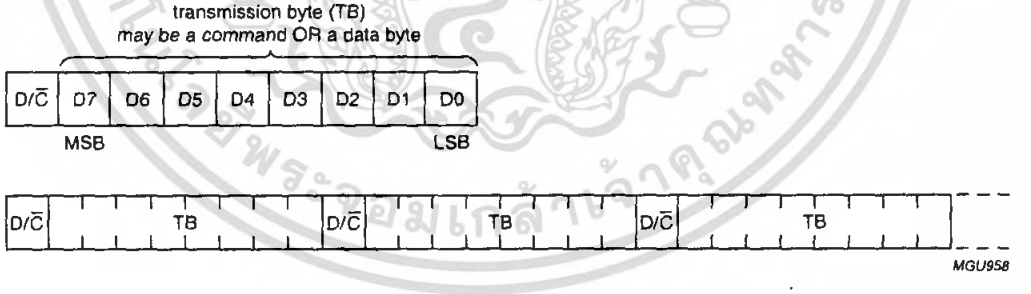


Fig.42 Serial data stream, Write mode.

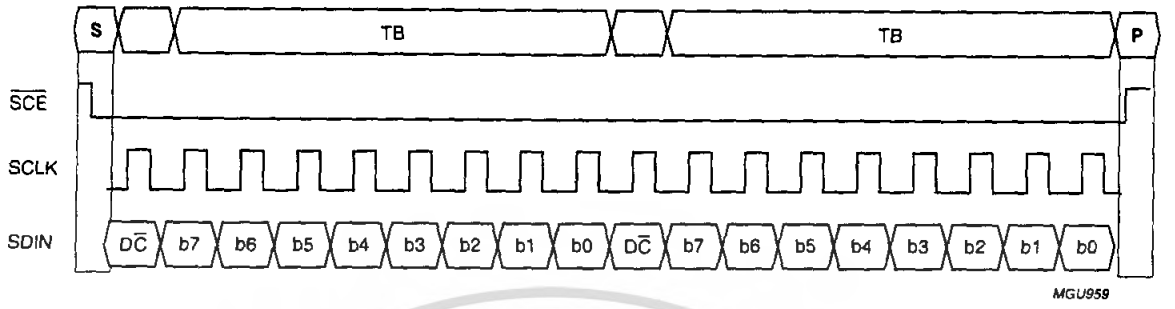


Fig.43 Serial bus protocol, write to register with control bit in transmission (PS[2:0] = XX0).

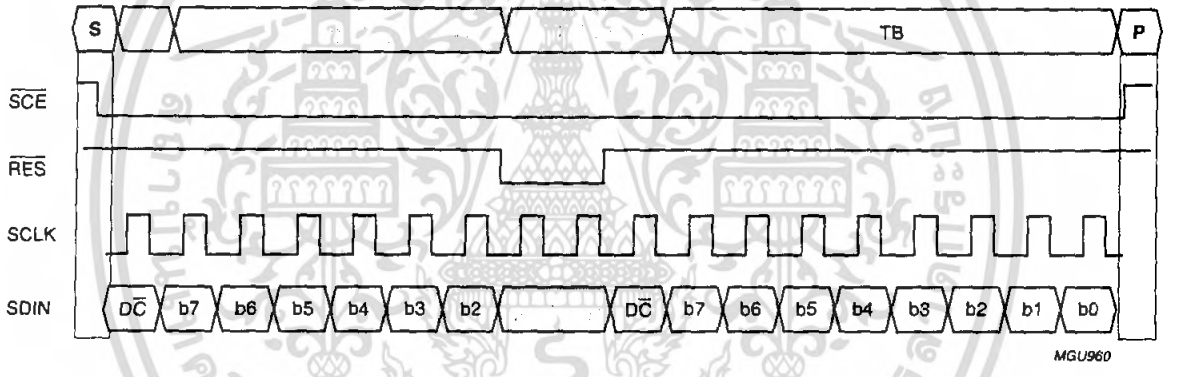


Fig.44 Serial bus protocol, Write mode, interrupted by reset ($\overline{RES}$); (PS[2:0] = XX0).

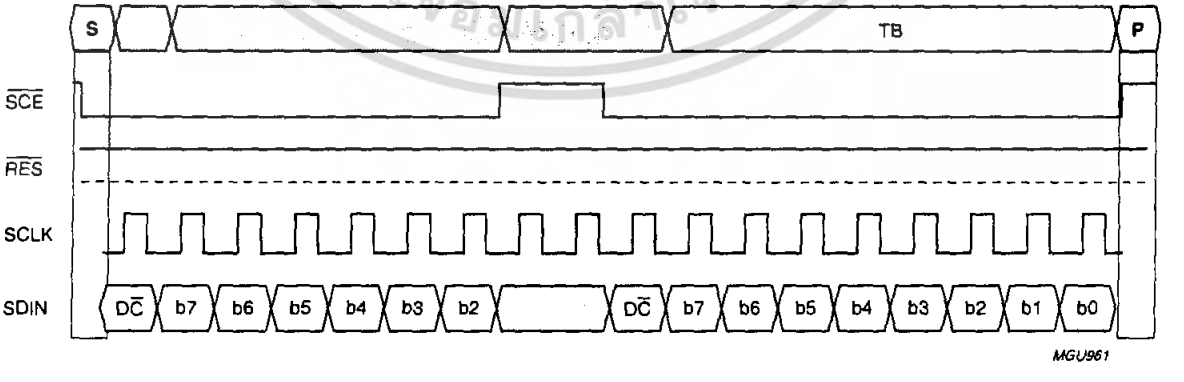


Fig.45 Serial bus protocol, Write mode, interrupted by chip enable ($\overline{SCE}$); (PS[2:0] = XX0).

9.2 Read mode

The Read mode of the serial interface means that the microcontroller reads data from the PCF8833. The PCF8833 can microcontroller in two different ways. The serial bus protocol for the RDID1, RDID2, RDID3 and RDTEMP commands is illu of these commands are given in Section 6.2. After a command has been issued, a byte is transmitted in the opposite dire to reach the timing characteristics as given in Chapter 13 data bit b7 must be handled as a don't care. When the speed at least half of maximum speed, at least for reading b7, the reading of data bit b7 is valid.

The PCF8833 samples the SDIN data at rising SCLK edges, but shifts SDOUT data at falling SCLK edges. Thus the mi read SDOUT data at rising SCLK edges.

After the read command has been sent, the SDIN line must be set to 3-state not later than the falling SCLK edge of the

When using the RDDIDIF (see Section 6.2.6) or RDDST (see Section 6.2.7) commands the PCF8833 sends 24 or 32 da microcontroller. The serial bus protocols for the RDDIDIF and RDDST commands are illustrated in Figs. 47 and 48. Afte been sent 3 or 4 bytes respectively are transmitted in the opposite direction (using SDOUT) after one dummy clock cycl

The 8th read bit is shorter than the others because it is terminated by the rising SCLK edge; see Figs 46, 47 and 48. Th SDOUT to 3-state.

The serial interface timing diagram is illustrated in Fig.51. For the dummy read cycle the time t_{ACC} is referenced to the ri

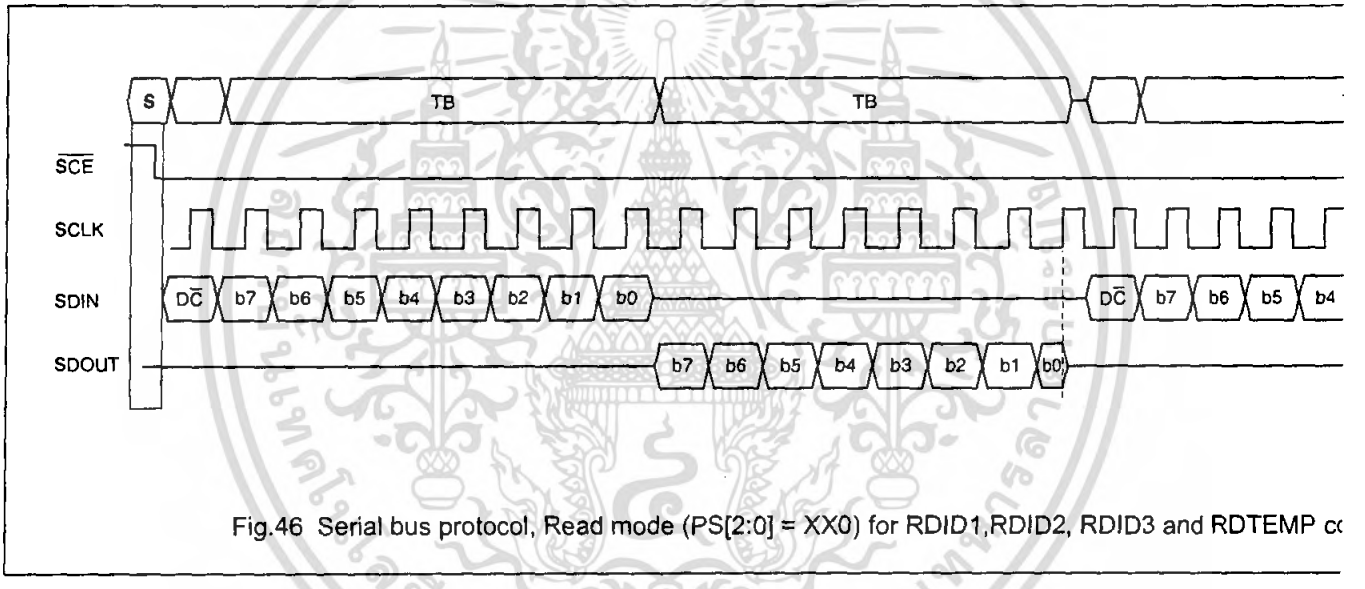


Fig.46 Serial bus protocol, Read mode (PS[2:0] = XX0) for RDID1,RDID2, RDID3 and RDTEMP c

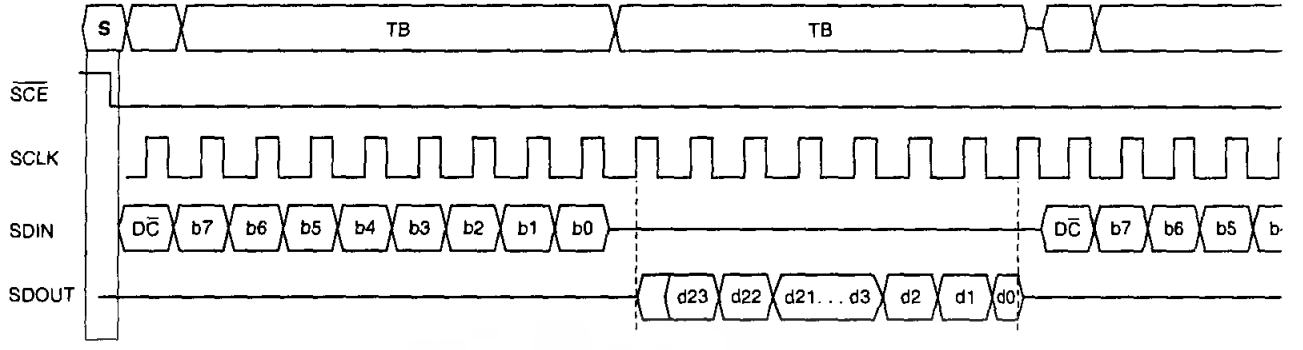


Fig.47 Serial bus protocol, Read mode (PS[2:0] = XX0) for the RDDIDIF command.

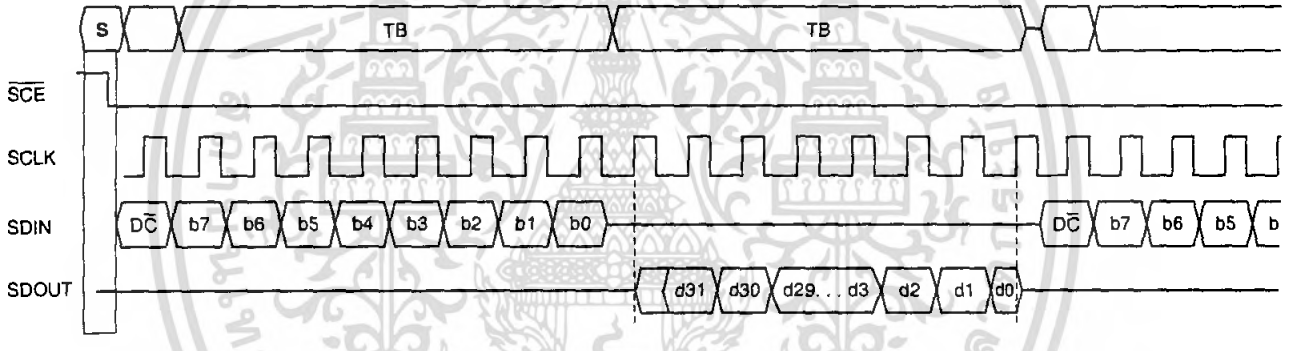


Fig.48 Serial bus protocol, Read mode (PS[2:0] = XX0) for the RDDST command.

12 DC CHARACTERISTICS

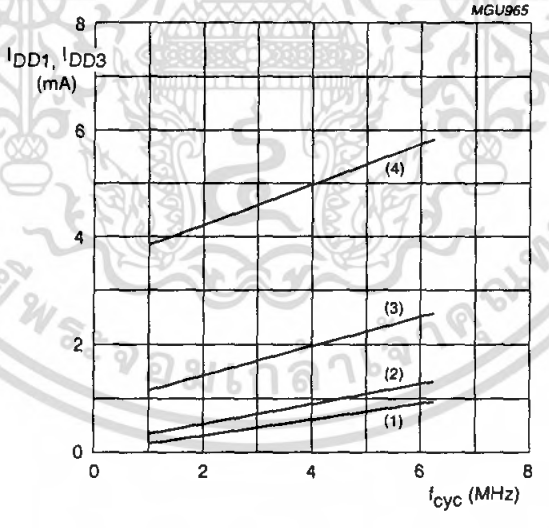
V_{DD1} = 1.5 to 3.3 V; $V_{DD2} = V_{DD3}$ = 2.4 to 3.5 V; V_{SS} = 0 V; V_{LCD} = 3.8 to 20.0 V; T_{amb} = −40 to +85 °C; unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Supplies						
V_{DD1}	logic supply voltage 1		1.5	–	3.3	V
V_{DD2}	supply voltage 2 for the internal voltage generator	note 1	2.4	–	4.5	V
V_{DD3}	supply voltage 3 for the internal voltage generator	note 1	2.4	–	3.5	V
V_{LCDIN1}	LCD supply voltage input 1	LCD input voltage 1 externally supplied (both voltage multipliers are disabled)	–	–	16.0	V
V_{LCDIN2}	LCD supply voltage input 2	LCD input voltage 2 externally supplied (both voltage multipliers are disabled)	–	–	20.0	V
$V_{LCDOUT1}$	LCD supply voltage output 1	LCD voltage internally generated with voltage multiplier 1 (voltage generator enabled); note 2	3.8	–	10.0	V
$V_{LCDOUT2}$	LCD supply voltage output 2	LCD voltage internally generated with voltage multiplier 2 (voltage generator enabled); note 2	3.8	–	20.0	V
$V_{LCD(tol)}$	tolerance of generated V_{LCD}	with calibration; note 3	−70	–	+70	mV
Static current consumption						
I_{DD1}	logic supply current	notes 5 and 6	–	1.5	5	µA
I_{DD2}, I_{DD3}	supply current for the internal voltage generator	notes 5 and 6	–	0.5	1	µA
Dynamic current consumption						
I_{DD1}	logic supply current	Normal mode; note 5	–	100	–	µA
I_{DD1}	logic supply current during RAM access	Normal mode; notes 5 and 7; see Fig.49	–	1000	–	µA
I_{DD2}, I_{DD3}	supply current for the internal voltage generator	Normal mode; note 5	–	tbf	–	µA
$I_{DD(tot)}$	total supply current (V_{DD1} + V_{DD2}, V_{DD3})	Normal mode; note 5	–	tbf	–	µA
Logic inputs and outputs						
V_{OL}	LOW-level output voltage	I_{OL} = 0.5 mA	V_{SS1}	–	$0.2V_{DD1}$	V
V_{OH}	HIGH-level output voltage	I_{OH} = −0.5 mA	$0.8V_{DD1}$	–	V_{DD1}	V
V_{IL}	LOW-level input voltage		V_{SS1}	–	$0.3V_{DD1}$	V
V_{IH}	HIGH-level input voltage		$0.7V_{DD1}$	–	V_{DD1}	V
I_L	leakage current	$V_I = V_{DD1}$ or V_{SS1}	−1	–	+1	µA

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Column and row outputs						
$R_{o(col)}$	column output resistance C0 to C395	V_{LCD2} = 10 V	–	–	5	kΩ
$R_{o(row)}$	row output resistance R0 to R131	V_{LCD2} = 10 V	–	–	5	kΩ
$V_{bias(col)}$	bias tolerance C0 to C395		−100	0	100	mV
$V_{bias(row)}$	bias tolerance R0 to R131		−100	0	100	mV

Notes

1. V_{DD2} and V_{DD3} always have to be higher than or equal to V_{DD1}.
2. The maximum possible V_{LCD} voltage that may be generated is dependent on supply voltage V_{DD2}, temperature and (display) load.
3. Valid for values of temperature, V_{PR} and TC used at the calibration and with temperature calibration disabled.
4. Power-save mode.
5. Conditions are: V_{DD1} = 2.75 V, V_{DD2} = 2.75 V, V_{LCD2} = 13.9 V, voltage multiplier 1 at 5 × V_{DD2}, inputs at V_{DD1} or V_{SS1}, interface inactive, internal V_{LCD} generation, V_{LCD2} output is loaded by 400 μA and V_{LCD1} output is loaded by 0 μA and T_{amb} = 25 °C.
6. During power-down all static currents are switched off.
7. V_{DD1} = 1.8 V and interface cycle time T_{cyc} = 333 ns.



(1) V_{DD1} = 1.5 V
(2) V_{DD1} = 1.8 V
(3) V_{DD1} = 2.5 V
(4) V_{DD1} = 3.3 V

Fig.49 Dynamic current consumption I_{DD1}, I_{DD3} for different V_{DD1} supplies when writing data from interface to display RAM at ambient temperature.

13 AC CHARACTERISTICS

V_{DD1} = 1.5 to 3.3 V; V_{DD2} = V_{DD3} = 2.4 to 3.5 V; V_{SS1} = V_{SS2} = 0 V; T_{amb} = −40 to +85 °C; note 1; unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
f_{frame}	LCD frame frequency (internal clock)	V_{DD1} = 3.0 V	–	tbf	–	Hz
f_{osc}	oscillator frequency	notes 2 and 3	–	600	–	kHz
$f_{clk(ext)}$	external clock frequency		–	tbf	–	kHz
Reset; see Fig.50						
$t_{W(RESL)}$	reset LOW pulse width	note 4	500	–	–	ns
t_{RSS}	reset spike suppression		–	–	100	ns
$t_{SU;RESL}$	reset LOW pulse set-up time after power-on		0	–	1	µs
t_{RT}	initialization	note 5	0	–	5	ms
t_{RI}	interface ready after reset pulse		0	–	1	µs
Serial interface; $V_{DD1(min)}$ = 1.65 V; note 6; see Fig.51						
T_{SCYC}	serial clock SCLK period (SCLK)		150	–	–	ns
t_{SHW}	SCLK pulse width HIGH		60	–	–	ns
t_{SLW}	SCLK pulse width LOW		60	–	–	ns
t_{SDS}	SDIN data set-up time		60	–	–	ns
t_{SDH}	SDIN data hold time		60	–	–	ns
t_{ACC}	SDOUT access time	C_L = 30 pF	10	–	50	ns
t_{OH}	SDOUT output disable time	C_L = 5 pF; R = 3 kΩ	25	–	50	ns
t_{SCC}	SCLK to $\overline{SCE}$ time		20	–	–	ns
t_{CHW}	$\overline{SCE}$ pulse width HIGH		40	–	–	ns
t_{CSS}	$\overline{SCE}$ to SCLK set-up time		60	–	–	ns
t_{CSH}	$\overline{SCE}$ to SCLK hold time		65	–	–	ns
8-bit parallel (8080-type) interface; $V_{DD1(min)}$ = 1.65 V; note 6; see Fig.52						
t_{CS}	$\overline{CS}$-WR and $\overline{CS}$-RD time	note 7	10	–	–	ns
t_{AH}	D/$\overline{C}$ address hold time		10	–	–	ns
t_{AS}	address set-up time		10	–	–	ns
T_{CYC}	system cycle time		160	–	–	ns
t_{CCLW}	$\overline{WR}$ control pulse width LOW	Write mode	38	–	–	ns
t_{CCLR}	$\overline{RD}$ control pulse width LOW	Read mode	38	–	–	ns
t_{CCHW}	$\overline{WR}$ control pulse width HIGH	Write mode	90	–	–	ns
t_{CCHR}	$\overline{RD}$ control pulse width HIGH	Read mode	90	–	–	ns
t_{DS}	D0 to D7 data set-up time		10	–	–	ns
t_{DH}	D0 to D7 data hold time		10	–	–	ns
t_{ACC}	read access time	note 8; C_L = 30 pF	–	–	30	ns
t_{OH}	output disable time	note 8; C_L = 5 pF; R = 3 kΩ; note 9	30	–	160	ns

Notes

1. V_{DD2} and V_{DD3} always have to be larger than or equal to V_{DD1}.
2. Not directly observable at any pin.
3. After calibration the following f_{osc} can be expected at 25 °C: 600 kHz ±4%; at different temperatures an additional variation of +0.12%/°C will not be exceeded.
4. All timing values are valid within the operating supply voltage and ambient temperature range and are referenced to V_{IL} and V_{IH} with an input voltage swing of V_{SS1} to V_{DD1}.
5. The initialization incorporates the start-up of the internal circuitry including the readout of the OTP cells. The start-up time for the internal voltage generation is not included.
6. The input signal rise time and fall time (t_r and t_f) are specified at 15 ns or less. When the cycle time is used at high speed, the specification is $t_r + t_f \leq (t_{CYC} - t_{CCLW} - t_{CCHW})$ or $t_r + t_f \leq (t_{CYC} - t_{CCLR} - t_{CCHR})$.
7. $\overline{CS}$ can be permanently tied LOW.
8. The output disable time and read access time is applicable after the second read cycle (see Fig.40).
9. For V_{DD1} = 1.8 V possible variation of t_{OH} is between 40 and 80 ns for a temperature range of –40 to +85 °C.

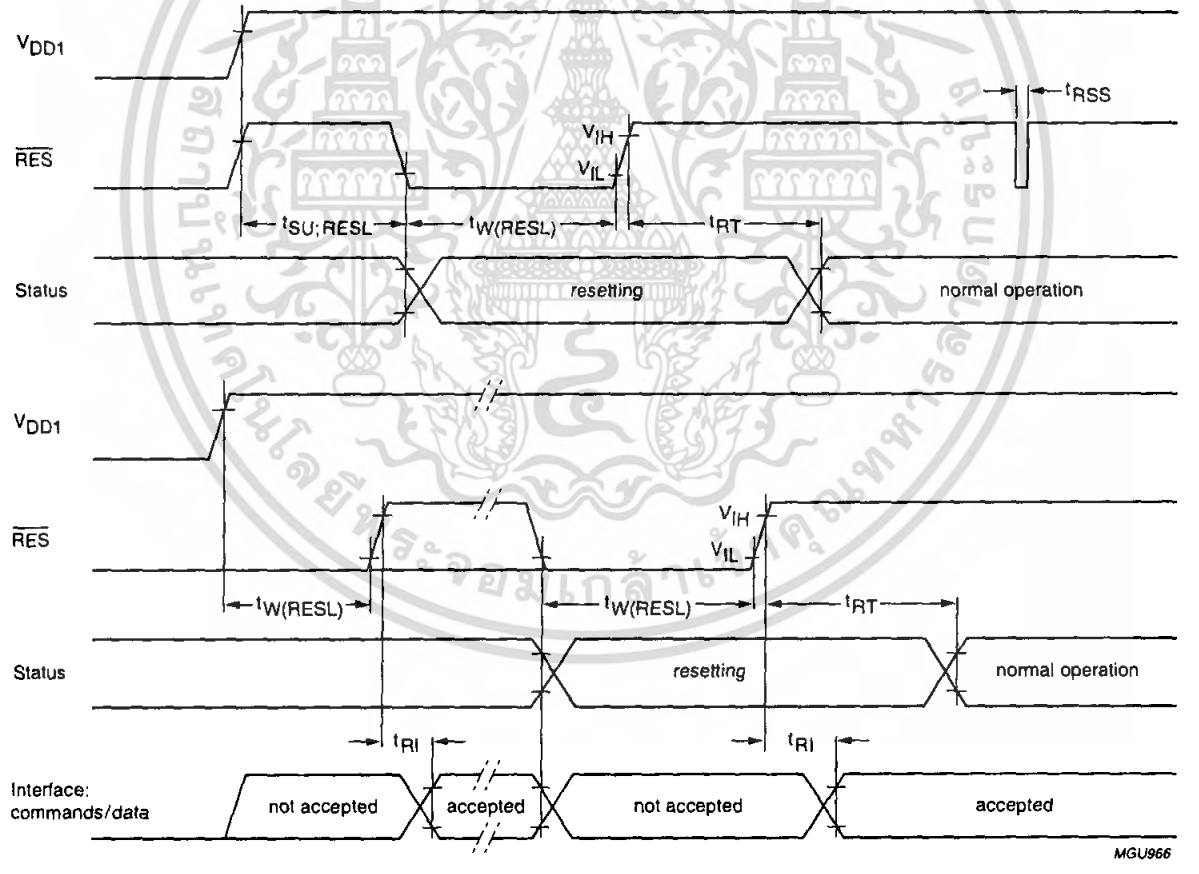


Fig.50 Reset timing.

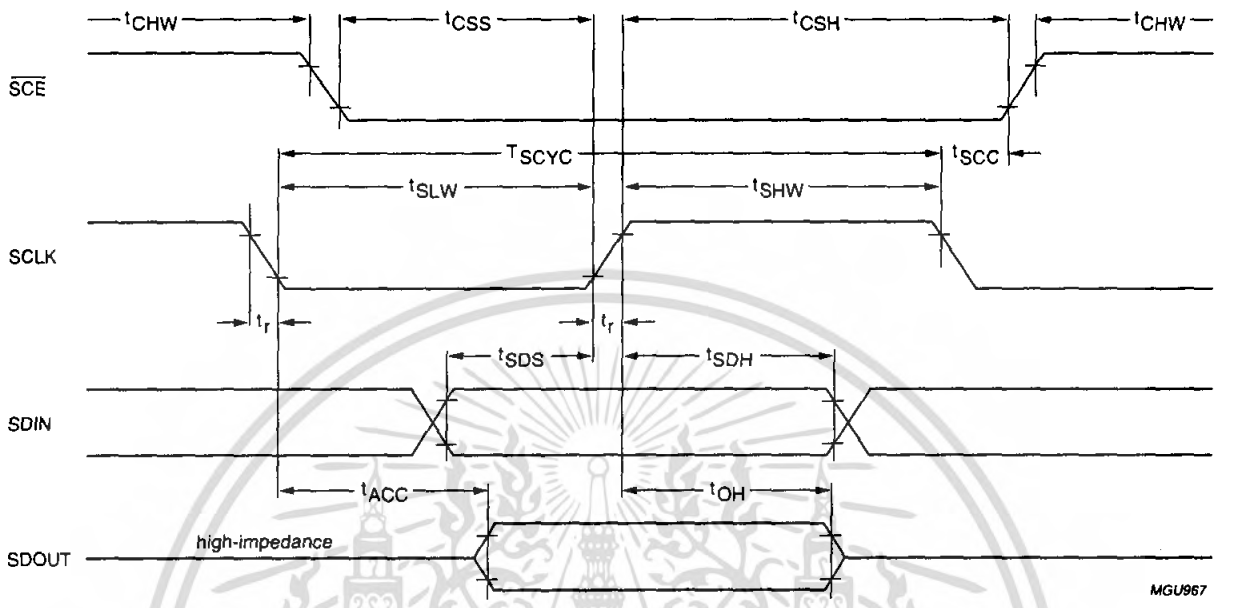


Fig.51 Serial interface timing.

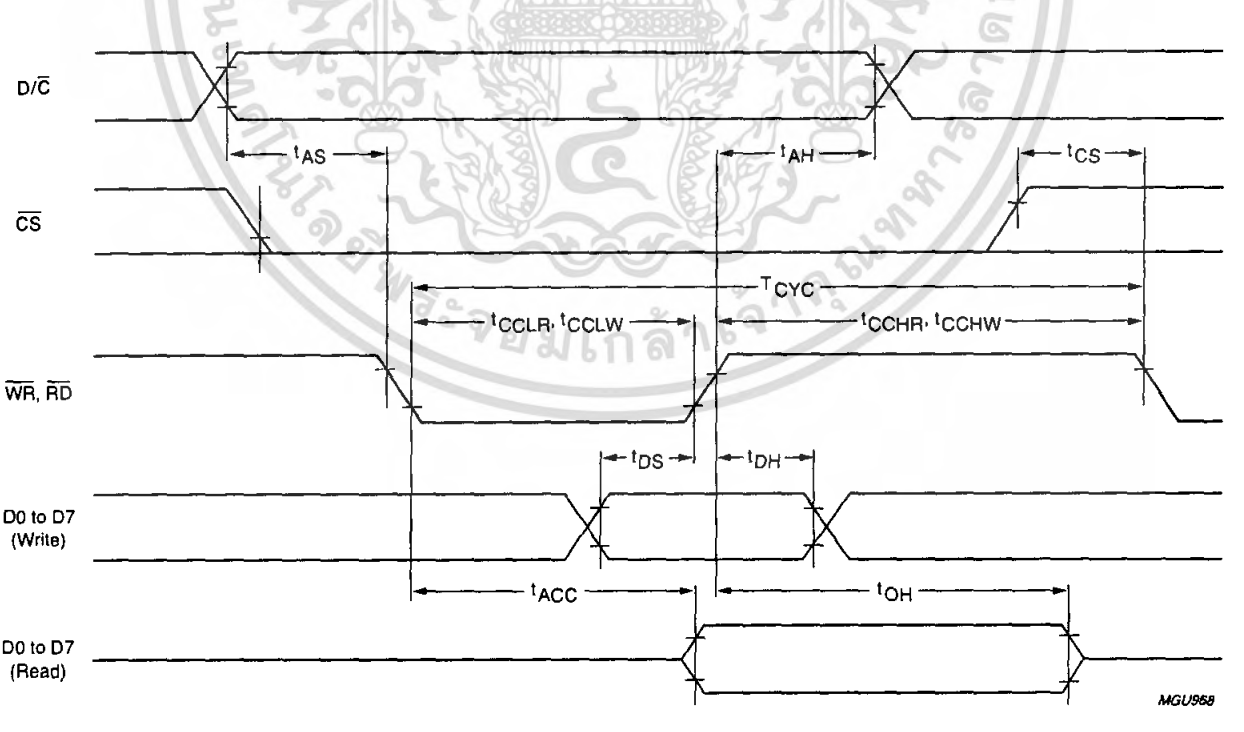


Fig.52 Parallel interface timing (8080 type).