เครื่องเล่น MP3 ชนิดพกพา แบบใช้ SD Card

Portable MP3 Player with SD Card Interface

โดย

นายกฤษฎิ์ ด่านวิรุทัย

นายชจรพงษ์ ตันจ้อย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

เครื่องเล่น MP3 ขนิดพกพา แบบใช้ SD Card

Portable MP3 Player with SD Card Interface

โดย

นายกฤษฎิ์ ค่านวิรุทัย รหัส 46010016

นายขจรพงษ์ ตันจ้อย รหัส 46010071

อาจารย์ที่ปรึกษา

ดร.กสิน วิเชียรชม

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ ปีการศึกษา 2549

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องเล่น MP3 ชนิดพกพา แบบใช้ SD Card

(Portable MP3 Player with SD Card Interface)

ผู้จัดทำ 1.นายกฤษฎิ์ ต่านวิรุทัย รหัส 46010016 ชั้นปีที่ 4

2.นายขจรพงษ์ ตันจ้อย รหัส 46010071 ชั้นปีที่ 4

..... อาจารย์ที่ปรึกษา

(ดร.กสิน วิเชียรชม)

เครื่องเล่น MP3 ชนิดพกพา แบบใช้ SD Card

นายกฤษฎิ์ ด่านวิรุทัย รหัส 46010016
นายขจรพงษ์ ตันจ้อย รหัส 46010071
ดร.กสิน วิเชียรชม อาจารย์ที่ปรึกษา
ปีการศึกษาที่ 2549

บทคัดย่อ

เครื่องเล่น MP3 ชนิดพกพาที่ได้ออกแบบ แบ่งออกเป็น 4 ส่วน คือ SD Card, MP3 Decoder, LCD และไมโครคอนโทรลเลอร์ ส่วนที่หนึ่งคือ การเก็บข้อมูล ข้อมูลเพลงจะถูกเก็บไว้ใน SD Card โดยมีการจัดเก็บข้อมูลในรูปแบบ FAT32 ส่วนที่สองคือ การถอดรหัสข้อมูลโดยใช้ MP3 Decoder เป็นการนำข้อมูลดิจิตอลมาถอดรหัสเป็นอนาล็อก แล้วส่งข้อมูลอนาล็อกออกไปยังชุดหูฟัง ส่วนที่สามคือ การแสดงผล เป็นการนำชื่อเพลงออกมาแสดงผ่านทางจอ LCD และส่วนสุดท้ายคือ การควบคุม ส่วนนี้เป็นส่วนที่ควบคุมการทำงานทั้งหมด โดยใช้ไมโครคอนโทรลเลอร์ ตระกูล PIC เครื่องเล่น MP3 นี้สามารถเล่นไฟล์MP3 และไฟล์WAV อีกทั้งสามารถควบคุม การเล่นเพลง หยุดเล่นเพลง การเลือกเพลง และการปรับระดับเสียง

Portable MP3 Player with SD Card Interface

Mr.Grit Danvirutai ID.46010016

Mr.Khachompong Tunjoy ID.46010071

Dr.Kasin Vichienchom Advisor

Educational Year 2006

Abstract

This Portable MP3 player with built-in SD card interface includes four parts which are a SD card reader, a MP3 decoder, a LCD display and a microcontroller. The audio data is stored in SD card using FAT32 format. A PIC microcontroller is served as a controller unit. It transfers the audio data from the SD card to the MP3 decoder. The decoder decodes the data and then drives analog audio directly to the head-set speaker. This Portable MP3 can play MP3 file and WAV file. It has user interface buttons for control basic functions such as play, stop and volume control.

กิตติกรรมประกาศ

โครงงานเครื่องเล่น MP3 ชนิดพกพา ซึ่งประกอบด้วยชิ้นงานและเอกสารประกอบโครงงานนี้จะสำเร็จลุล่วงมาด้วยดีมิได้หากขาดอาจารย์กสิน วิเชียรชม อาจารย์ที่ปรึกษาผู้คอยให้คำแนะนำ และดูแลอย่างใกล้ชิดมาโดยตลอด พร้อมทั้งการให้ความช่วยเหลือจากเพื่อนในภาควิชา สุดท้ายบุคคลที่จะลืมมิได้เลยคือ ผู้ปกครองซึ่งคอยสนับสนุนและคอยให้กำลังใจเสมอมา

นายกฤษฎิ์ ด่านวิรุทัย

นายขจรพงษ์ ตันจ้อย

ผู้จัดทำ

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงาน

ปัจจุบันจะสังเกตได้ว่าเครื่องเล่นทั่วๆไปจะมีความสามารถในการเล่นไฟล์ข้อมูลได้หลายประเภท เช่น DVD, VCD, CD และที่ขาดไม่ได้คือ MP3 ซึ่งเป็นอีกหนึ่งมาตรฐานในการบีบอัดไฟล์เสียงและกำลังได้รับความนิยมอยู่ในขณะนี้

เนื่องจากผู้จัดทำได้มีโอกาสอ่านบทความทางอิเล็กทรอนิกส์ ตลอดจนข้อมูลข่าวสารทางอินเทอร์เน็ต ทำให้ทราบว่าปัจจุบันมีไอซีที่สามารถถอดรหัสไฟล์ MP3 ได้ และมีอยู่หลายบริษัทที่ผลิตออกมาให้ทดลองใช้กัน เช่น STA013 ของ ST Technology, AT89C51SDN2 ของ ATMEL, MICRONAS MAS3570 และ VS10xx ของ VLSI เป็นต้น สำหรับการใช้งานก็มีความยากง่ายต่างกัน เช่นการต่อวงจรร่วมภายนอกและเทคนิคในการติดต่อสื่อสาร สำหรับผู้จัดทำเลือกใช้ VS1011B เหตุผลคือ เป็นไอซีที่ใช้อุปกรณ์ภายนอกน้อยชิ้น การติดต่อเพื่อควบคุมการทำงานจะง่ายกว่า และมีราคาถูกกว่าไอซีเบอร์อื่นๆที่ได้กล่าวมาข้างต้น

เทคโนโลยีเกี่ยวกับการ์ดหน่วยความจำ (Memory Card) ก็ได้มีการพัฒนาออกสู่ตลาดมากมายหลายรูปแบบ เช่น Memory Stick, Smart Media, Compact Flash, SD/MMC Card เป็นต้น ซึ่งการใช้งานก็มีหลายลักษณะเช่นกัน ในส่วนของผู้จัดทำได้เลือกใช้ SD Card เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไม่ค่อยสูงมากและเป็นการ์ดที่หาซื้อได้ง่าย

สุดท้ายในส่วนของไมโครคอนโทรลเลอร์ เนื่องจากผู้จัดทำต้องการไมโครคอนโทรลเลอร์ที่ต้องการกำลังไฟฟ้าค่อนข้างต่ำ มีการประมวลผลที่รวดเร็ว ผู้จัดทำจึงเลือกใช้ไมโครคอนโทรลเลอร์ตระกูล PIC (PIC18F4539) ในการควบคุมการทำงานทั้งหมด และเลือกใช้ภาษาซีในการเขียนโค้ดโปรแกรม

1.2 วัตถุประสงค์

เพื่อศึกษาการเขียน/อ่านข้อมูลการ์ดหน่วยความจำแบบ SD Card, ศึกษาการบีบอัดไฟล์แบบ MP3, ศึกษาการติดต่อสื่อสารข้อมูลในลักษณะ SPI BUS , เรียนรู้และฝึกใช้งานแอลซีดี ตลอดจนเพิ่มทักษะในการใช้งานไมโครคอนโทรลเลอร์ตระกูล PIC

1.3 ขอบเขตของโครงงาน

โครงงานเครื่องเล่น MP3 ชนิดพกพา มี SD Card นี้ ได้ออกแบบและสร้างวงจรในการเขียน/อ่านการ์ดหน่วยความจำแบบ SD Card, การถอดรหัสไฟล์ MP3 โดยใช้ไอซี VS1011B ซึ่งการติดต่อกับ

อุปกรณ์ทั้งสองจะเป็นการติดต่อในลักษณะ SPI BUS, มีการแสดงผลโดยใช้จอแอลซีดี และใช้ไมโครคอนโทรลเลอร์ (PIC18F4539) ในการควบคุม

1.4 โครงสร้างของรายงาน

รายงานฉบับนี้ได้อธิบายขั้นตอน วิธีในการออกแบบ รวมทั้งวงจร และผลการทดลองทดสอบคุณสมบัติต่างๆของเครื่องเล่น MP3 ชนิดพกพา มี SD Card โดยมีเนื้อหาแบ่งเป็นบทต่างๆ ดังนี้

บทที่ 2 ทฤษฎี จะกล่าวถึงทฤษฎี และหลักการพื้นฐานต่างๆ ที่เกี่ยวข้องกับการออกแบบและสร้างเครื่องเล่น MP3 ชนิดพกพา มี SD Card

บทที่ 3 การออกแบบ จะกล่าวถึงขั้นตอนในการออกแบบและโค้ดโปรแกรม ในการติดต่อกับการ์ดหน่วยความจำแบบ SD Card, การติดต่อกับไอซีถอดรหัสไฟล์ MP3, และการติดต่อกับแอลซีดีแสดงผล โดยใช้ไมโครคอนโทรลเลอร์ PIC

บทที่ 4 การทดลอง และผลการทดลอง จะกล่าวถึงการทดลอง และผลการทดลอง เมื่อทำการทดสอบสั่งงานเครื่องเล่น MP3 ชนิดพกพา มี SD Card นี้ ด้วยอินพุทต่างๆ

บทที่ 5 สรุป และวิจารณ์

บทที่ 2
ทฤษฎี

2.1 MP3 (Moving Picture Expert Group-1 Layer 3)

2.1.1 ประวัติของ MP3

ประมาณปี 1992 มาตรฐานภาพเคลื่อนไหวและเสียงในแบบ MPEG (Moving Pictures Experts Group) ได้ถูกกำหนดขึ้นภายใต้ ISO และ IEC ซึ่งเป็นหน่วยงานที่รองรับมาตรฐานในการใช้งานเทคโนโลยีด้านดิจิตอล ซึ่ง MPEG-1 นี้ได้เป็นจุดเริ่มต้นของของการนำเอาระบบดิจิตอลวีดีโอ มาแทนระบบอนาล็อกได้ โดยเราจะเห็นได้จากผลิตภัณฑ์ประเภทวีดีโอซีดี (VCD) , SVCD และที่สำคัญยังให้คุณภาพที่ดีกว่า , ยืดหยุ่นกว่าด้วย และที่สำคัญราคาไม่สูงนัก นอกจาก MPEG-1 ต่อมาก็มีการพัฒนามาเป็นมาตรฐาน MPEG-2 สำหรับเทคโนโลยี DVD และ Digital TV ซึ่งในอนาคตต่อไปเราจะเห็นการนำไปใช้เทคโนโลยีนี้ได้มากขึ้น

จากที่ได้กล่าวมา คือความเป็นมาของมาตรฐาน MPEG ซึ่งมีการบันทึกข้อมูลทั้งภาพและเสียง เราสามารถหยิบเพียงส่วนของการบันทึกเสียงในแบบ MPEG มาบันทึกเพลงอย่างเดียวได้ เทคโนโลยีการบีบอัดเสียงแบบ MP3 จึงเกิดขึ้น

MP3 หรือ MPEG-1 Layer 3 (หรือ MPEG-2 Layer 3) เป็นมาตรฐานที่ถูกกำหนดขึ้นสำหรับงานเสียงโดยเฉพาะ เพราะเป็นวิธีการเข้ารหัสที่มีขนาดเล็ก และคุณภาพเสียงอยู่ในระดับชัดเจน เทคนิคการบีบอัดเสียงในแบบ MPEG-1 และ MPEG-2 เป็นวิธีการที่น่าศึกษา เพราะว่าใช้เทคนิคทางด้าน Digital Image Processing ในการตัดข้อมูลที่ซ้ำกัน และไม่จำเป็นออกไป ทำให้ไฟล์ข้อมูลที่ได้มีขนาดเล็กลง

2.1.2 การแปลงไฟล์เสียงจากแผ่นออดิโอเป็นไฟล์ MP3 (Ripping)

เราเรียกการแปลงไฟล์เสียงจากแผ่นออดิโอซีดีเป็นไฟล์ MP3 ว่ากระบวนการ "Ripping" ซึ่งปัจจุบันก็มีซอฟแวร์ในการแปลงไฟล์ในลักษณะดังกล่าวอยู่มาก ยกตัวอย่างเช่น Window Media Player, iTunes เป็นต้น

2.1.3 อัตราการสุ่ม (Sampling Rates)

เป็นตัวกำหนดความละเอียดของคลื่นความถี่เสียง รูปคลื่นเสียงใน 1 ลูก (Sine Wave) เราจะเห็นว่าในตำแหน่งที่ต่างกัน ก็จะมีความลาดชันที่ต่างกันและยิ่งไปกว่านั้น คลื่นเสียงแต่ละลูกก็จะมีรูปร่างต่างกันออกไป โดยสำหรับอัตราการสุ่มก็คือจุดที่บอกว่าเสียง ณ ตำแหน่งนั้นๆ อยู่ที่ความถี่เท่าใด ซึ่งอาจเกิดคำถามขึ้นว่าไฟล์เสียงที่มีอัตราการสุ่มต่างๆ กันเช่น 11, 22, 44 และ 48 กิโลเฮิรตซ์ หมายถึงอะไร และแตกต่างกันอย่างไร

สำหรับอัตราการสุ่มที่มีหน่วยเป็นกิโลเฮิรตซ์ ก็คือ จำนวนจุดที่บอกความถี่ของเสียงในหนึ่งคลื่นเสียง เช่น ไฟล์ที่มีอัตราการสุ่มเท่ากับ 22 กิโลเฮิรตซ์ หมายความว่าในหนึ่งคลื่นเสียงประกอบไปด้วยจุดที่บอกความถี่ประมาณ 2 หมื่นจุดด้วยกัน และในทำนองเดียวกันไฟล์ที่มีอัตราการสุ่มเท่ากับ 44 กิโลเฮิรตซ์ หมายความว่ามีจุดบอกตำแหน่งความถี่ จำนวน 4 หมื่นกว่าจุดในหนึ่งคลื่นเสียงนั่นเอง ซึ่งในกรณีนี้ยิ่งมีมากเท่าไหร่ก็หมายถึงคุณภาพเสียงก็มีมากขึ้นเท่านั้น

2.1.4 อัตราบิต (Bit Rates)

ไฟล์ MP3 สามารถถูก rip จากไฟล์ออดิโอในแผ่นซีดีได้ตั้งแต่ 8 กิโลบิตต่อวินาที ถึง 320 กิโลบิตต่อวินาที ค่าอัตราบิตยิ่งสูงไฟล์ที่ได้ยิ่งมีคุณภาพเสียงที่ดี แต่ไฟล์ก็มีขนาดใหญ่ตามไปด้วย ไฟล์เพลง MP3 ส่วนมากจะบีบอัดกันที่อัตราบิต 128 กิโลบิตต่อวินาที ถ้าลดอัตราบิตลงอยู่ที่ 96 กิโลบิตต่อวินาที ขนาดไฟล์จะเล็กลง และคุณภาพเสียงก็จะลดลงไปด้วย ถ้าเราฟังกันแบบธรรมดาๆ สำหรับบางท่านที่สนใจแค่เนื้อเพลง และความพอใจที่ได้ฟังมากกว่าคุณภาพเสียง คุณภาพเท่านี้ก็คงเพียงพอแล้ว แต่สำหรับบางท่านที่สนใจในเรื่องของคุณภาพเสียง คุณภาพเสียงก็คงสำคัญกว่าขนาดไฟล์ เพราะฉะนั้นเทคโนโลยี MP3 จึงนิยมที่จะบีบอัดกันที่อัตราบิต 128 กิโลบิตต่อวินาที

2.1.5 การบีบอัดข้อมูล (Compression)

จะทำการสุ่มเอาบางส่วนของข้อมูลมาในอัตราส่วนที่เหมาะสม เช่น 1:4 หมายถึง ข้อมูลเสียงจริง ๆ คือ 4 แต่อ่านเข้ามาเพียง 1 แม้จะสูญเสียบ้างแต่ก็น้อยมาก และเสียงที่ได้ยินนั้น ก็ยังแยกไม่ออก อัตราการสุ่มของ MP3 ในชั้น (Layer) ต่าง ๆ มีดังนี้

ชั้น 1 1:4

ชั้น 2 1:6 ถึง 1:8

ชั้น 3 1:10 ถึง 1:12

เมื่อไฟล์ที่ได้มีขนาดเล็ก การส่งผ่านบนอินเตอร์เน็ตหรือการนำไปใช้งานก็ทำได้ง่ายขึ้น สะดวกขึ้น และเมื่อมีการนำไฟล์ MP3 ไปเปิดฟัง โปรแกรมที่อ่านข้อมูลไฟล์ MP3 ก็จะขยายข้อมูลออก (Decompress) เพื่อให้ได้เนื้อความเดิม โดยใช้เทคนิคของ Digital Processing เหมือนกัน

2.1.6 การตัดความถี่

จากการวิจัยพบว่า คนทั่วไปมีความสามารถในการรับฟังเฉลี่ยแล้วอยู่ในช่วง 2 – 5 กิโลเฮิรตซ์ เพราะฉะนั้น ความถี่ในช่วงก่อนหน้า 2 กิโลเฮิรตซ์ และที่สูงกว่า 5 กิโลเฮิรตซ์ จึงไม่จำเป็นต้องนำไปรวมในข้อมูล MP3 ด้วย

นอกจากนี้ MP3 ได้นำการเข้ารหัสแบบ Variable Length มาใช้ ทำให้ผลของการเข้ารหัสสามารถลดขนาดของข้อมูลลงได้กว่า 20% โดยเมื่อรวมเรื่องของการบีบอัดข้อมูลและการตัดความถี่เข้าไปด้วย ทำให้การบันทึกข้อมูลแบบ MP3 ได้ขนาดไฟล์ที่เล็กมากกว่า 50% ถ้าเปรียบเทียบกับการบีบอัดในแบบ wav โดยใช้แหล่งข้อมูลเดียวกัน

From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Name	Size	Type	Date Modified
Track01.mp3	4,059 KB	MP3 audio file (mp3)	12/09/2003 11:20 AM
Track02.mp3	4,305 KB	MP3 audio file (mp3)	12/09/2003 11:22 AM
Track03.mp3	4,193 KB	MP3 audio file (mp3)	12/09/2003 11:23 AM
Track04.mp3	4,825 KB	MP3 audio file (mp3)	12/09/2003 12:45 PM
Track05.mp3	4,802 KB	MP3 audio file (mp3)	12/09/2003 12:47 PM
Track06.mp3	5,078 KB	MP3 audio file (mp3)	12/09/2003 1:01 PM
Track07.mp3	3,164 KB	MP3 audio file (mp3)	12/09/2003 4:10 PM
Track08.mp3	4,813 KB	MP3 audio file (mp3)	12/09/2003 4:13 PM
Track09.mp3	4,395 KB	MP3 audio file (mp3)	12/09/2003 4:14 PM
Track10.mp3	6,851 KB	MP3 audio file (mp3)	12/09/2003 4:16 PM
Track11.mp3	6,523 KB	MP3 audio file (mp3)	12/09/2003 4:22 PM
Track12.mp3	1,423 KB	MP3 audio file (mp3)	12/09/2003 4:23 PM
Track01.wav	44,755 KB	Wave Sound	12/09/2003 10:30 AM
Track02.wav	47,465 KB	Wave Sound	12/09/2003 10:31 AM
Track03.wav	46,237 KB	Wave Sound	12/09/2003 10:32 AM
Track04.wav	53,208 KB	Wave Sound	12/09/2003 10:33 AM
Track05.wav	52,944 KB	Wave Sound	12/09/2003 10:34 AM
Track06.wav	55,998 KB	Wave Sound	12/09/2003 10:35 AM
Track07.wav	34,890 KB	Wave Sound	12/09/2003 10:36 AM
Track08.wav	53,070 KB	Wave Sound	12/09/2003 10:37 AM
Track09.wav	48,465 KB	Wave Sound	12/09/2003 10:38 AM
Track10.wav	75,533 KB	Wave Sound	12/09/2003 11:13 AM
Track11.wav	71,916 KB	Wave Sound	12/09/2003 11:15 AM
Track12.wav	15,700 KB	Wave Sound	12/09/2003 11:17 AM

รูปที่ 2.1 เปรียบเทียบขนาดไฟล์ระหว่างไฟล์ MP3 กับ ไฟล์ WAVE

2.2 ตารางการจัดเรียงไฟล์ (File Allocation Table:FAT)

ระบบปฏิบัติการของคอมพิวเตอร์แต่ละแพล็ตฟอร์ม (Platform) จะมีการจัดการระบบไฟล์ในฮาร์ดดิสก์ที่แตกต่างกัน บางระบบสามารถใช้ระบบไฟล์ได้หลายรูปแบบ โดยระบบไฟล์นั้นเป็นตารางที่ใช้บอกตำแหน่งของข้อมูลต่างๆที่อยู่บนฮาร์ดดิสก์ว่าอะไรอยู่ตรงไหน ปกติเมื่อซื้อฮาร์ดดิสก์มาใหม่ต้องทำการจัดข้อมูล (Format) ให้ฮาร์ดดิสก์ ก่อนที่จะนำไปบรรจุข้อมูล การจัดข้อมูลในฮาร์ดดิสก์เป็นการแบ่งฮาร์ดดิสก์ออกเป็นส่วนๆ เพื่อให้คอมพิวเตอร์รู้ว่าตำแหน่งของข้อมูลอยู่ตรงไหน

FAT เป็นระบบไฟล์ที่ใช้ในระบบปฏิบัติการในตระกูล Microsoft และเป็นระบบไฟล์ที่มีพัฒนาการมาอย่างต่อเนื่อง ระบบไฟล์ในตระกูลนี้มีลักษณะคือ เป็นการกำหนดหมายเลขให้กับทุกๆ คลัสเตอร์ (Cluster) ในแต่ละส่วนแบ่งของฮาร์ดดิสก์ (Partition) แล้วทำการสร้างตารางที่มีจำนวนช่องตามจำนวนคลัสเตอร์ เพื่อเป็นการระบุสถานที่หรือคลัสเตอร์ที่ทำการเก็บข้อมูลของไฟล์แต่ละไฟล์ และมีตารางอีกตารางหนึ่งที่เรียกว่าไดเรคทอรี่ (Directory) สำหรับเก็บข้อมูลรายละเอียดของไฟล์ เช่น คุณลักษณะ (Attribute) ต่าง ๆ และ หมายเลข Cluster เริ่มต้นที่เก็บตัวข้อมูลจริง ๆ

ระบบจัดการไฟล์แบบ FAT เป็นระบบที่ไม่ยุ่งยากซับซ้อน ดังนั้นจึงถูกรองรับจากทุกระบบปฏิบัติการที่มีอยู่สำหรับคอมพิวเตอร์ส่วนบุคคล และยังสะดวกในการแลกเปลี่ยนข้อมูลระหว่างระบบปฏิบัติการที่แตกต่างกันซึ่งถูกติดตั้งบนคอมพิวเตอร์เครื่องเดียวกัน

ข้อด้อยของระบบจัดการไฟล์แบบ FAT คือ เมื่อไฟล์ถูกลบและไฟล์ใหม่ถูกเขียนลงไป แฟรกเมนต์ (fragment) ของแต่ละไฟล์จะมีโอกาสกระจัดกระจายออกไปอยู่ทั่วทั้งหน่วยความจำ ส่งผลให้การอ่านและการเขียนไฟล์ทำได้ช้า การจัดเรียงข้อมูลเป็นหนึ่งในวิธีการทำให้การจัดเรียงไฟล์แบบ FAT เป็นระเบียบ แต่จะต้องทำการจัดเรียงข้อมูลอยู่บ่อยๆ และเป็นกระบวนการที่ใช้เวลามาก

ระบบไฟล์ FAT มีหลายรุ่นดังต่อไปนี้

2.2.1 FAT12

ระบบ FAT12 เป็นรุ่นที่เก่าแก่ที่สุดของตระกูล FAT ใช้กับระบบปฏิบัติการ DOS ซึ่งใช้ 12 บิต ในการอ้างอิงถึงหมายเลขคลัสเตอร์ เพราะฉะนั้นสามารถอ้างถึงคลัสเตอร์ได้มากที่สุด 4086 คลัสเตอร์ (ประมาณ 2 ยกกำลัง 12 = 4096 และมีเนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัว FAT เอง) ระบบ FAT12 จึงเหมาะกับหน่วยความจำที่มีเนื้อที่ไม่มาก เช่น Floppy Disk หรือดิสก์ขนาดเล็ก ที่มีเนื้อที่ไม่เกิน 16 เมกะไบต์ และระบบ FAT12 นี้สามารถใช้งานกับไฟล์ที่มีชื่อยาวเพียง 8.3 ตัวอักษรเท่านั้น

2.2.2 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 และมีเนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัว FAT 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 MS-DOS ยุคแรก 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

2.2.3 FAT32

ระบบ FAT32 ใช้กับระบบปฏิบัติการ Windows 95/98, Windows ME, Windows 2000 โดยทำการแก้ไขเพิ่มเติมจาก FAT16 เพื่อที่จะได้มีจำนวนคลัสเตอร์ต่อพาร์ติชันมากขึ้น ดังนั้นเลยมีความสามารถที่จะรองรับหน่วยความจำที่มีขนาดใหญ่สูงสุดได้ถึง 2 เทระไบต์ (2000 กิกะไบต์) ซึ่งจะใช้ 28 บิต (อีก 4 บิต สำรองเอาไว้) ฉะนั้นสามารถอ้างถึงคลัสเตอร์ได้ทั้งหมด 286 ล้านคลัสเตอร์ (ประมาณ 2 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)
Introduced	1977 (Microsoft Disk BASIC)	July 1988 (MS-DOS 4.0)	August 1996 (Windows 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

2.2.4 โครงสร้างของดิสก์หลัก (Main disk structures)

Boot sector	More reserved sectors (optional)	File Allocation Table#1	File Allocation Table#2	Root Directory	Data Region (for files and directories)...(To end of partition or disk)

ระบบไฟล์แบบ FAT ประกอบไปด้วย 4 ส่วนที่แตกต่างกัน ดังนี้

1. เซคเตอร์สำรอง (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 โดยบริเวณตารางจัดเรียงไฟล์นี้จะสอดคล้องกับ บริเวณข้อมูล (Data Region) ทำหน้าที่ระบุตำแหน่งคลัสเตอร์ของไฟล์และไดเรคทอรี่ต่างๆ

3. บริเวณไดเรคทอรี่ (Root Directory Region) นี่คือตารางไดเรคทอรี่ซึ่งเก็บข้อมูลเกี่ยวกับไฟล์และไดเรคทอรี่ต่างๆ ในระบบจัดการไฟล์แบบ FAT12 และ FAT16 ตารางไดเรคทอรี่นี้จะอยู่ในบริเวณไดเรคทอรี่เท่านั้น และมีขนาดสูงสุดที่แน่นอน แต่ในระบบจัดการไฟล์แบบ FAT32 ตารางไดเรคทอรี่จะอยู่รวมกับไฟล์ต่างๆในบริเวณข้อมูล ซึ่งสามารถขยายขนาดออกไปได้ไม่จำกัด

4. บริเวณข้อมูล เป็นส่วนที่ไฟล์ข้อมูลอยู่จริง ขนาดของไฟล์และไดเรคทอรี่ย่อยสามารถเพิ่มขึ้นได้เท่าที่มีคลัสเตอร์เหลืออยู่

2.2.4.1 บูทเซคเตอร์

ตารางที่ 2.2 โครงสร้างพื้นฐาน 36 ไบต์แรกของบูทเซคเตอร์ ซึ่งใช้ในทุกเวอร์ชันของ FAT

ไบต์ออฟเซต (Byte Offset)	ความยาว (ไบต์)	รายละเอียด
0x00	3	คำสั่งกระโดด (เพื่อจะข้ามส่วนหัวของการบูท)
0x03	8	ชื่อ OEM ค่าพื้นฐานคือ ไอบีเอ็ม 3.3 และ MS-DOS 5.0
0x0b	2	จำนวนไบต์ต่อเซคเตอร์, บล็อกของพารามิเตอร์ BIOS เริ่มต้นที่นี่
0x0d	1	จำนวนเซคเตอร์ต่อคลัสเตอร์
0x0e	2	เซคเตอร์สำรอง
0x10	1	จำนวนของตารางการจัดเรียงข้อมูล
0x11	2	จำนวนสูงสุดของไดเรคทอรี่
0x13	2	จำนวนเซคเตอร์ทั้งหมด
0x15	1	ตัวบรรยายมีเดีย 0xF8 ด้านเดียว, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xF9 สองด้าน, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 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
0x16	2	จำนวนเซคเตอร์ต่อตารางการจัดเรียงข้อมูล
0x18	2	จำนวนเซคเตอร์ต่อแทรก (track)
0x1a	2	จำนวนของเฮด
0x1c	4	จำนวนของเซคเตอร์ที่โดนซ่อน
0x20	4	จำนวนเซคเตอร์ทั้งหมด (ใช้ในกรณีมากกว่า 65535 เซคเตอร์)

ตารางที่ 2.3 โครงสร้างของบูทเซคเตอร์ที่เพิ่มขึ้นมาใน FAT32

ไบต์ออฟเซต	ความยาว (ไบต์)	รายละเอียด
0x24	4	จำนวนเซคเตอร์ต่อตารางการจัดเรียงข้อมูล
0x28	2	เครื่องหมายของตารางจัดเรียงข้อมูล
0x2a	2	เวอร์ชัน
0x2c	4	หมายเลขคลัสเตอร์ของไดเรคทอรี่เริ่มต้น
0x30	2	หมายเลขเซคเตอร์ของเซคเตอร์ข้อมูล FS
0x32	2	หมายเลขเซคเตอร์ของสำเนาบูทเซคเตอร์
0x34	12	สำรอง
0x40	1	ฟิสิคอลไดรฟ์นัมเบอร์ (Physical Drive Number)
0x41	1	สำรอง
0x42	1	ซิกเนเจอร์ (Signature)
0x43	4	หมายเลขซีเรียล
0x47	11	โวลุ่มลาเบล (Volume Label)
0x52	8	ชนิดของระบบไฟล์แบบ FAT "FAT32"
0x5a	420	โค้ดบูทระบบปฏิบัติการ
0x1FE	2	เซคเตอร์สุดท้าย (0x55 0xAA)

2.2.4.2 ตารางไดเรคทอรี่

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

ตัวอักษรที่สามารถใช้ตั้งชื่อไฟล์ได้สำหรับ DOS

- ตัวพิมพ์ใหญ่ A-Z
- ตัวเลข 0-9
- ช่องว่าง (ช่องว่างยาวๆจะไม่นับเป็นส่วนของชื่อไฟล์)

- ! # $ % & ( ) - @ ^ _ ‘ { } ~
- ค่า 128 – 255

จำนวนไดเรคทอรี่ทั้งหมดทั้งในบริเวณไดเรคทอรี่และในไดเรคทอรี่ย่อย มีรูปแบบ ดังนี้

ตารางที่ 2.4 โครงสร้างของตารางไดเรคทอรี่

<table>
<tr><th>ไบต์ออฟเซต</th><th>ความยาว (ไบต์)</th><th colspan="3">รายละเอียด</th></tr>
<tr><td>0x00</td><td>8</td><td colspan="3">0x00</td></tr>
<tr><td>0x08</td><td>3</td><td colspan="3">เครื่องหมายของดารางจัดเรียงข้อมูล</td></tr>
<tr><td rowspan="9">0x0b</td><td rowspan="9">1</td><td>บิต</td><td>มาสก์ (Mask)</td><td>รายละเอียด</td></tr>
<tr><td>0</td><td>0x01</td><td>อ่านได้อย่างเดียว</td></tr>
<tr><td>1</td><td>0x02</td><td>ถูกซ่อน</td></tr>
<tr><td>2</td><td>0x04</td><td>ระบบ</td></tr>
<tr><td>3</td><td>0x08</td><td>โวลุ่มลาเบล</td></tr>
<tr><td>4</td><td>0x10</td><td>ไดเรคทอรี่ย่อย</td></tr>
<tr><td>5</td><td>0x20</td><td>เอกสารสำคัญ</td></tr>
<tr><td>6</td><td>0x40</td><td>อุปกรณ์</td></tr>
<tr><td>7</td><td>0x80</td><td>ไม่ใช้</td></tr>
<tr><td>0x0c</td><td>1</td><td colspan="3">สำรอง ถูกใช้โดย NT</td></tr>
<tr><td>0x0d</td><td>1</td><td colspan="3">เวลาสร้างไฟล์, ความละเอียด 10 มิลลิวินาที ค่าอยู่ระหว่าง 0 – 199</td></tr>
<tr><td rowspan="4">0x0e</td><td rowspan="4">2</td><td>บิต</td><td colspan="2">รายละเอียด</td></tr>
<tr><td>15-11</td><td colspan="2">ชั่วโมง (0-23)</td></tr>
<tr><td>10-5</td><td colspan="2">นาที (0-59)</td></tr>
<tr><td>4-0</td><td colspan="2">วินาที/2 (0-29)</td></tr>
<tr><td rowspan="4">0x10</td><td rowspan="4">2</td><td>บิต</td><td colspan="2">รายละเอียด</td></tr>
<tr><td>15-9</td><td colspan="2">ปี (0=1980, 127=2107)</td></tr>
<tr><td>8-5</td><td colspan="2">เดือน (1=มกราคม (January), 12=ธันวาคม (Dcccmber))</td></tr>
<tr><td>4-0</td><td colspan="2">วันที่ (1-31)</td></tr>
<tr><td>0x12</td><td>2</td><td colspan="3">วันที่ใช้งานไฟล์ล่าสุด ดูไบต์ออฟเซต 0x10 ประกอบ</td></tr>
</table>

0x14	2	ดัชนี EA (ถูกใช้โดย OS/2 และ NT) ใน FAT12 และ FAT16, หรือ 2 ไบต์สูงของคลัสเตอร์แรกใน FAT32
0x16	2	เวลาที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซต 0x0e ประกอบ
0x18	2	วันที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซต 0x10 ประกอบ
0x1a	2	คลัสเตอร์แรกใน FAT12 และ FAT16, หรือ 2ไบต์ต่ำของคลัสเตอร์แรกใน FAT32
0x1c	4	ขนาดไฟล์

2.2.4.3 ตาราง FAT

ขนาดของแต่ละคลัสเตอร์ขึ้นอยู่กับชนิดของระบบไฟล์แบบ FAT ที่ใช้และขนาดของพาร์ติชัน โดยปกติขนาดของคลัสเตอร์จะอยู่ระหว่าง 2 กิโลไบต์ถึง 32 กิโลไบต์ แต่ละไฟล์อาจจะกินเนื้อที่มากกว่า 1 คลัสเตอร์ขึ้นอยู่กับขนาดของไฟล์นั้น 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

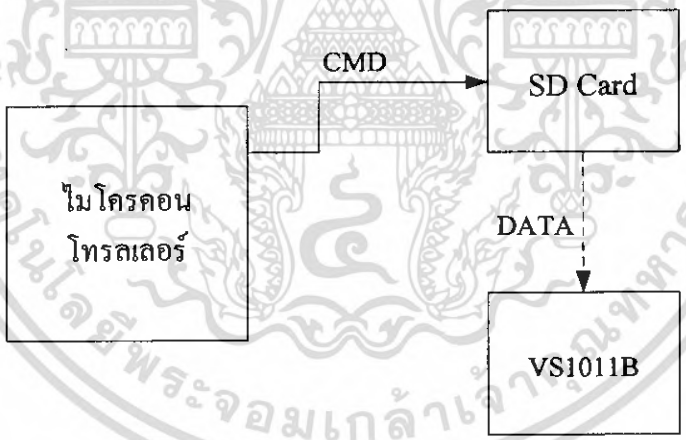
บทที่ 3
การออกแบบ

3.1 การติดต่อระหว่างไมโครคอนโทรลเลอร์, SD Card และ ตัวถอดรหัส MP3

ผู้จัดทำคิดหาวิธีการติดต่อระหว่างอุปกรณ์ทั้งสามที่น่าจะเป็นไปได้ และทำการทดสอบ 3 วิธี ด้วยกัน คือ

3.1.1 ให้ SD Card ส่งข้อมูลให้ตัวถอดรหัส MP3 โดยตรง

วิธีนี้เริ่มจากให้ไมโครคอนโทรลเลอร์ส่งคำสั่งอ่านข้อมูลแบบหลายบล็อก(หรือต่อเนื่อง)จาก SD Card ผ่านทางพอร์ต SPI จากนั้นข้อมูลที่ได้จาก SD Card จะถูกส่งไปถอดรหัสที่ตัวถอดรหัส MP3 โดยตรง ข้อดีของวิธีนี้คือ ไมโครคอนโทรลเลอร์ไม่จำเป็นต้องมีหน่วยความจำภายในคอยเก็บข้อมูลที่ได้จาก SD Card และทำให้การถอดรหัสเป็นไปอย่างต่อเนื่อง ส่วนข้อจำกัดของวิธีนี้คือ เป็นการละเมิดการติดต่อตามมาตรฐาน SPI ที่ต้องเป็นการติดต่อในลักษณะ Master/Slave แต่วิธีนี้เราให้ Slave ติดต่อกันเองซึ่งการทำงานในระบบจริงอาจทำให้เกิดความผิดพลาดได้

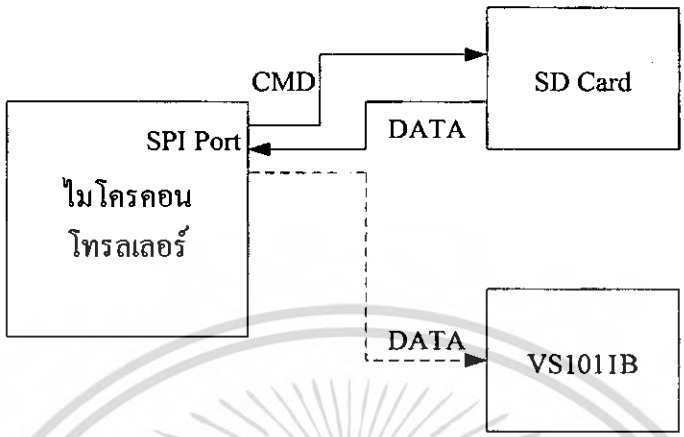


รูปที่ 3.1 วิธีการติดต่อระหว่างไมโครคอนโทรลเลอร์, SD Card, และตัวถอดรหัส MP3 วิธีที่ 1

3.1.2 ใช้พอร์ต SPI ร่วมกัน

วิธีนี้เริ่มจากให้ไมโครคอนโทรลเลอร์ส่งคำสั่งอ่านข้อมูล 1 บล็อก (512 ไบต์) จาก SD Card ผ่านทางพอร์ต SPI จากนั้นข้อมูลที่ได้จาก SD Card จะถูกส่งกลับมายังไมโครคอนโทรลเลอร์ จากนั้นไมโครคอนโทรลเลอร์จะส่งข้อมูลดังกล่าวผ่านทางพอร์ตเดียวกันนี้ไปให้ตัวถอดรหัส MP3 เพื่อทำการถอดรหัส และเมื่อส่งข้อมูลจนหมด 512 ไบต์ ไมโครคอนโทรลเลอร์ก็จะไปนำข้อมูลใหม่มาจาก SD Card เพื่อส่งให้ตัวถอดรหัส MP3 และจะทำงานเช่นนี้ไปเรื่อยๆจนจบเพลง ข้อดีของวิธีนี้คือ เรามีการใช้ทรัพยากรของไมโครคอนโทรลเลอร์ซึ่งมีพอร์ต SPI เพียงพอร์ตเดียวได้อย่างคุ้มค่า ส่วนข้อจำกัดของวิธี

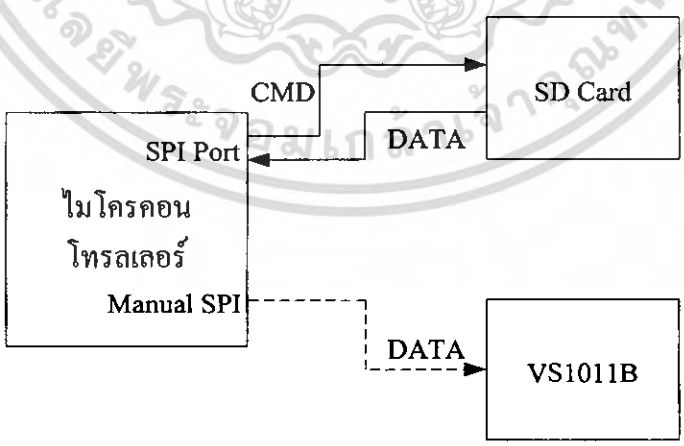
นี้คือ เราต้องหาอัตราเร็วในการรับ/ส่งข้อมูลที่เหมาะสมระหว่างอุปกรณ์ทั้งสาม เนื่องจากเราใช้พอร์ตร่วมกัน ดังนั้นอัตราเร็วในการรับ/ส่งข้อมูลจึงเป็นอัตราเดียวกัน



รูปที่ 3.2 วิธีการติดต่อระหว่างไมโครคอนโทรลเลอร์, SD Card, และตัวถอดรหัส MP3 วิธีที่ 2

3.1.3 ใช้ SPI 2 พอร์ต

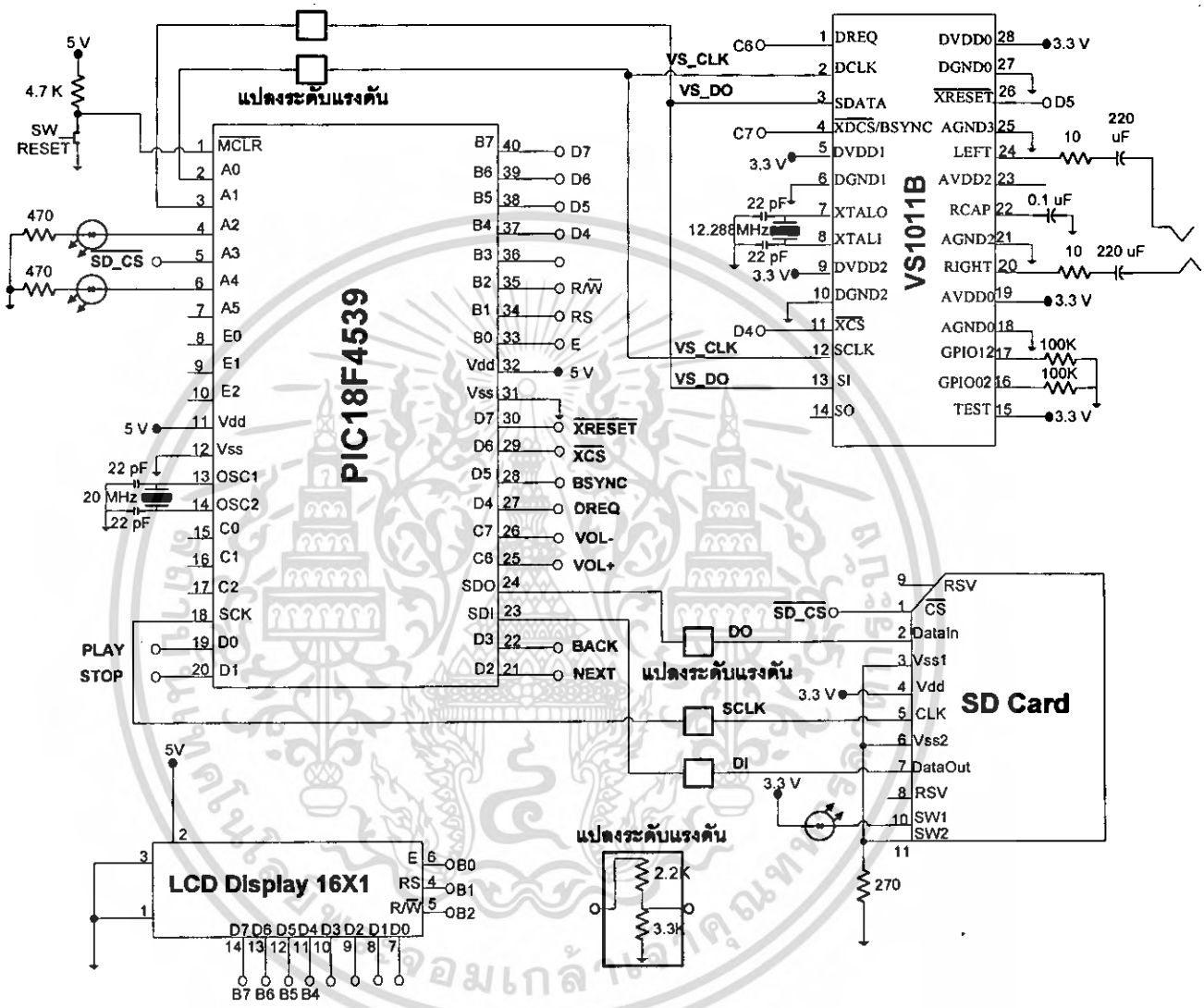
วิธีนี้จะคล้ายกับวิธีที่ 2 แต่ส่วนที่ต่างกันคือ การส่งข้อมูลจากไมโครคอนโทรลเลอร์ให้ตัวถอดรหัส MP3 นั้นจะไม่ใช้พอร์ตเดียวกันกับตอนที่ไปนำข้อมูลมาจาก SD Card แต่จะใช้พอร์ต SPI ที่เขียนฟังก์ชันขึ้นมาเองในการส่งข้อมูลแทน ข้อดีของวิธีนี้คือ เราสามารถกำหนดอัตราเร็วในการับ/ส่งข้อมูลระหว่างอุปกรณ์ทั้งสามได้เป็นอิสระต่อกัน และยังง่ายต่อการควบคุมในระบบจริงอีกด้วย ส่วนข้อจำกัดคือ เราต้องเปลืองพอร์ตสำหรับใช้กับฟังก์ชันที่เขียนขึ้นมาเอง และจำเป็นต้องสร้างสัญญาณลักษณะ SPI ขึ้นมาให้ตรงกับมาตรฐานของสัญญาณ SPI



รูปที่ 3.3 วิธีการติดต่อระหว่างไมโครคอนโทรลเลอร์, SD Card, และตัวถอดรหัส MP3 วิธีที่ 3

ซึ่งจากการพิจารณาข้อดี ข้อจำกัดและทำการทดสอบ 3 วิธีข้างต้น ผู้จัดทำเลือกใช้วิธีที่ 3 ในการติดต่อระหว่างอุปกรณ์ทั้งสามสำหรับเครื่องเล่น MP3 ที่ออกแบบ

3.2 วงจรรวมของเครื่องเล่น MP3 ชนิดพกพา

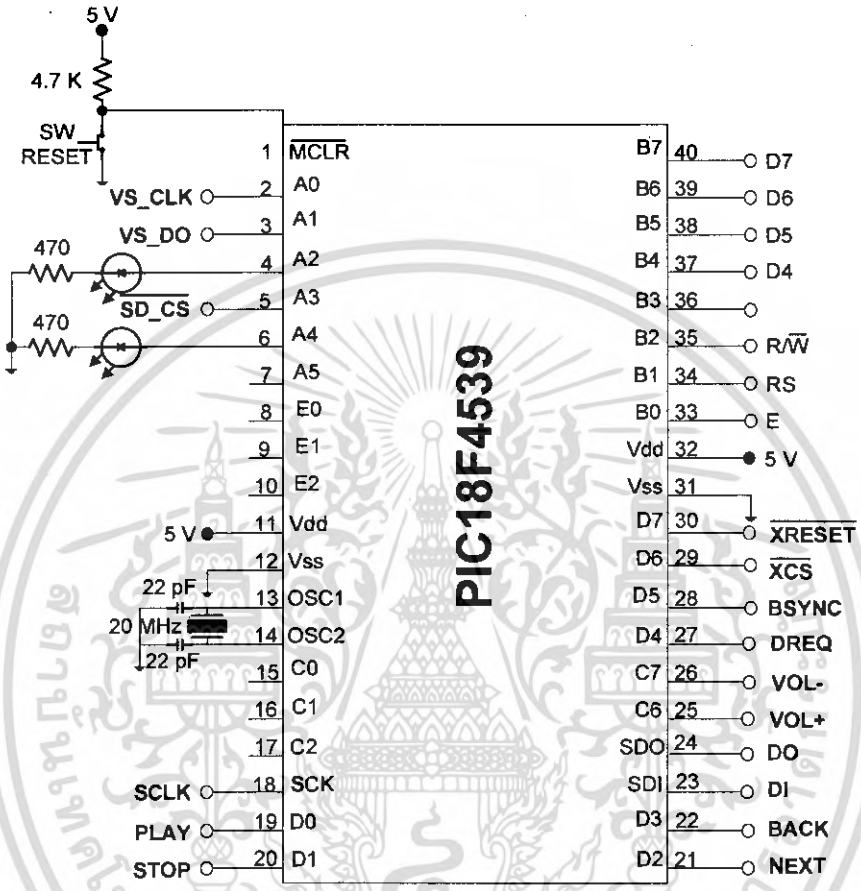


รูปที่ 3.4 วงจรรวมของเครื่องเล่น MP3 ชนิดพกพา

รูปที่ 3.5 เครื่องเล่น MP3 ที่ออกแบบ

เครื่องเล่น MP3 ชนิดพกพาแบ่งอุปกรณ์ออกเป็น 4 ส่วนหลัก ดังนี้

3.1.1 ไมโครคอนโทรลเลอร์



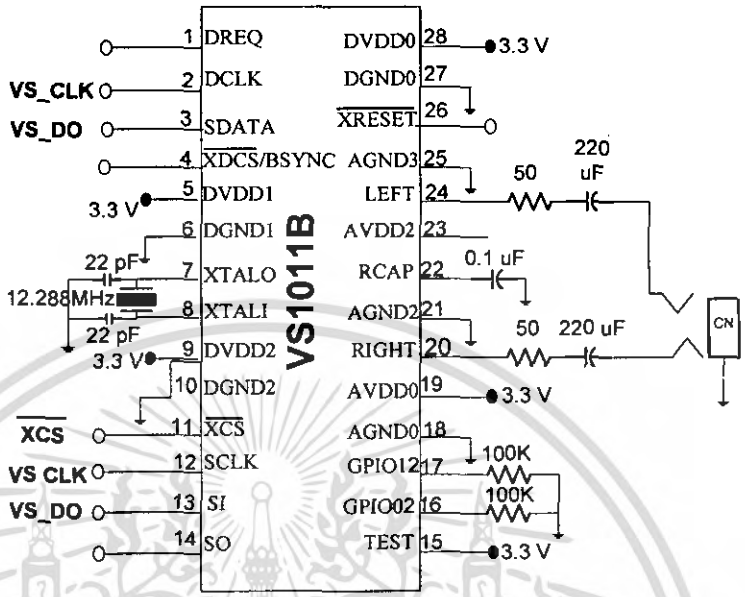
รูปที่ 3.6 วงจรส่วนของไมโครคอนโทรลเลอร์

เนื่องจากผู้จัดทำต้องการใช้ไมโครคอนโทรลเลอร์ที่มีการประมวลผลรวดเร็ว รองรับการติดต่อในระบบ SPI BUS มีขนาดของ RAM ที่เพียงพอ และมีจำนวนพอร์ตอินพุท/เอาท์พุทเหมาะสมจึงเลือกใช้ไมโครคอนโทรลเลอร์ตระกูล PIC เบอร์ 18F4539 และใช้คริสตอลความถี่ 20 เมกะเฮิรตซ์

ในส่วนของอินพุทเพื่อเล่นเพลงจะมีอยู่ 6 อินพุท คือ PLAY, STOP, NEXT, PREVIOUS, VOLUME UP, และ VOLUME DOWN

ในการติดต่อกับตัวถอดรหัส MP3 มี 6 ขา คือ $\overline{XRESET}$, $\overline{XCS}$, *BSYNC*, *DREQ*, *VS_CLK*, และ *VS_DO* ส่วนการติดต่อกับ SD/MMC Card มี 4 ขา คือ $\overline{SD\_CS}$, *SCK*, *SDO*, และ *SDI*

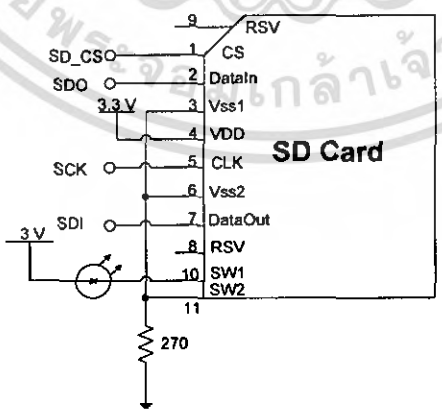
3.1.2 ตัวถอดรหัส MP3



รูปที่ 3.7 วงจรส่วนของตัวถอดรหัส MP3

สาเหตุที่ผู้จัดทำเลือกใช้ตัวถอดรหัส MP3 เบอร์ VS1011B คือ ในตัวถอดรหัส MP3 เบอร์นี้มีวงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อกและวงจรขับลำโพงอยู่ในตัว อีกทั้งยังมีตัวถัง (package) เป็นแบบ SOIC-28 ทำให้ง่ายต่อการใช้งาน

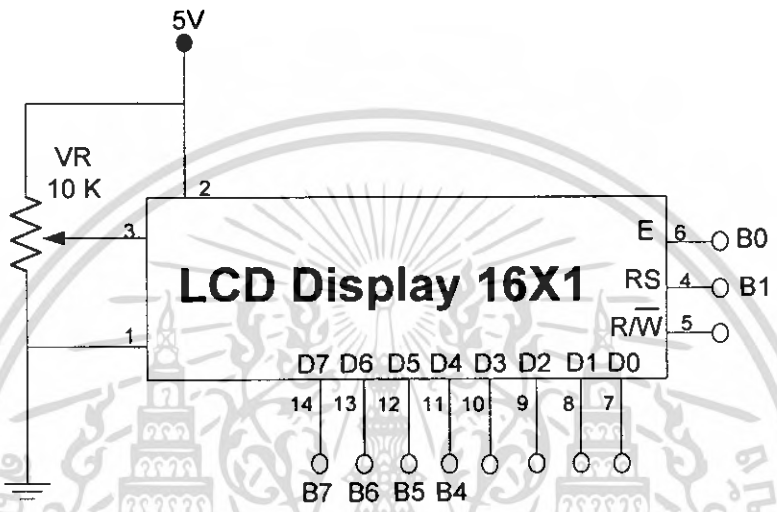
3.1.3 SD Card



รูปที่ 3.8 วงจรส่วนของ SD Card

ผู้จัดทำได้เลือกใช้ SD Card เนื่องจากเห็นว่ามีความเหมาะสมในเรื่องของราคาที่ไม่ค่อยสูงมาก และเป็นการ์ดที่หาซื้อได้ง่าย โดย SD Card ที่ใช้มีความจุ 512 เมกะไบต์

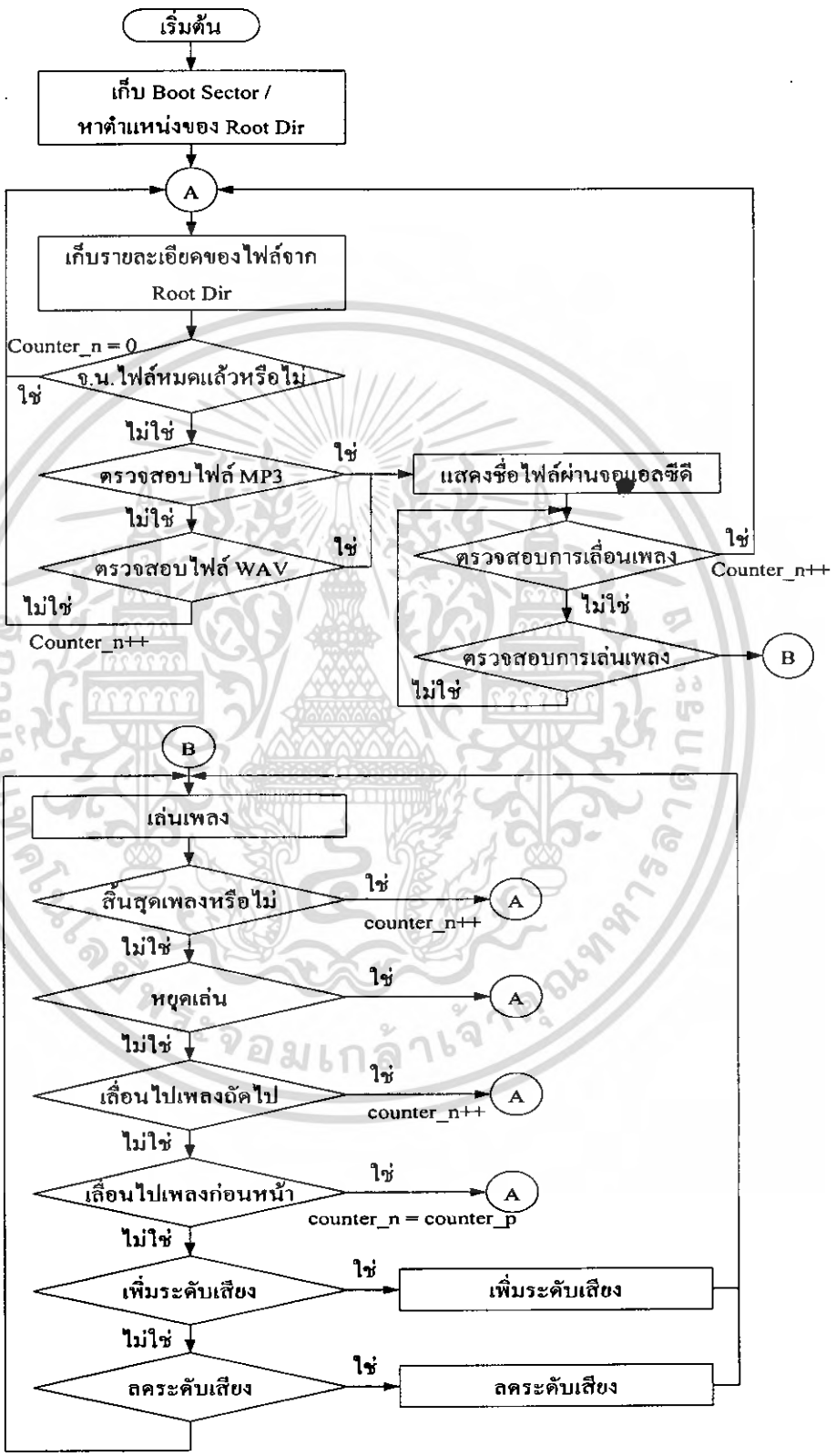
3.1.3 จอแสดงผล LCD



รูปที่ 3.9 วงจรส่วนของจอแสดงผล LCD

ผู้จัดทำเลือกใช้จอแสดงผล LCD ขนาด 16 ตัวอักษร 1 บรรทัด เพื่อแสดงชื่อเพลงที่กำลังเล่นอยู่

3.2 ผังการทำงานของเครื่องเล่น MP3 ชนิดพกพา



รูปที่ 3.10 ผังการทำงานของเครื่องเล่น MP3 ชนิดพกพา

บทที่ 4

การทดลองและผลการทดลอง

การทดลองแบ่งออกเป็น 5 การทดลอง ดังนี้

การทดลองที่ 1 วัดสัญญาณ Sine จากฟังก์ชันทดสอบเสียงสัญญาณ Sine (Sine Test) ที่ความถี่และระดับความดังต่างกัน

การทดลองที่ 2 วัดสัญญาณข้อมูลที่ SD Card ส่งกลับมา

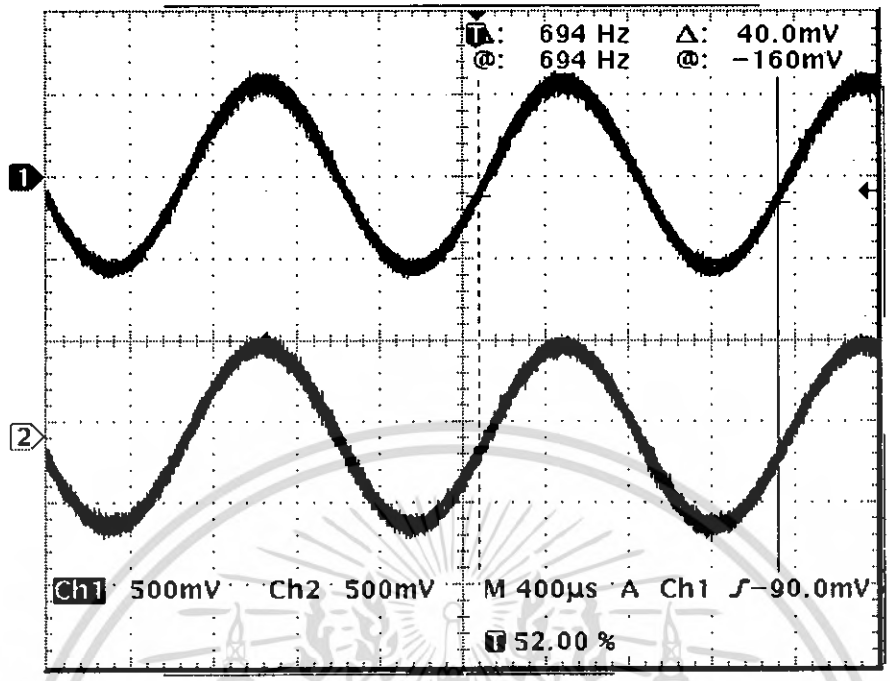
การทดลองที่ 3 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus

การทดลองที่ 4 ทดสอบหาอัตราเร็วที่เหมาะสมในการส่งข้อมูลให้กับตัวถอดรหัส MP3

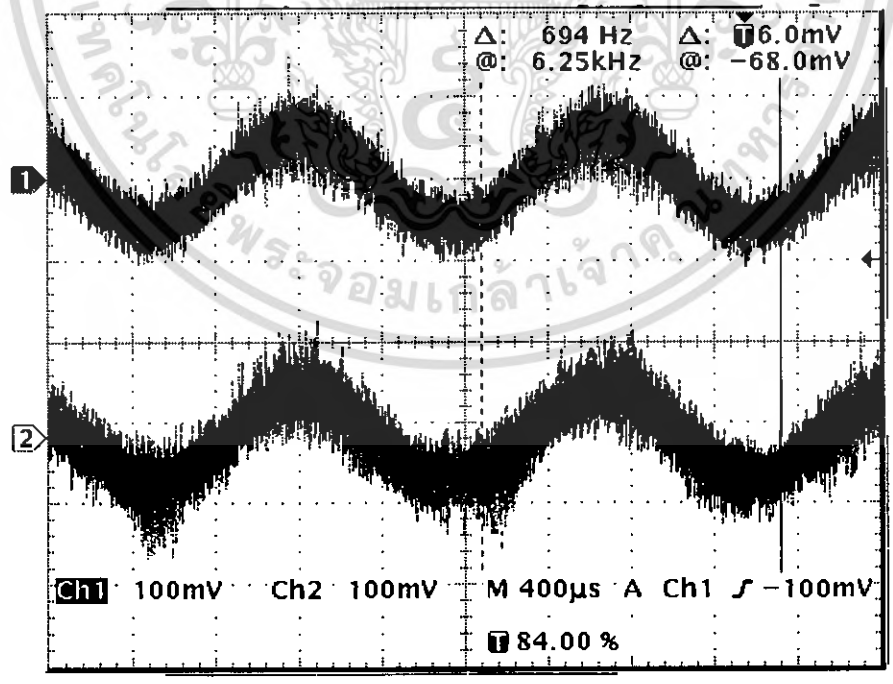
การทดลองที่ 1 วัดสัญญาณ Sine จากฟังก์ชันทดสอบเสียงสัญญาณ Sine (Sine Test) ที่ความถี่และระดับความดังต่างกัน

ตารางที่ 4.1 ผลการทดลองวัดสัญญาณ Sine จากฟังก์ชันทดสอบเสียงสัญญาณ Sine

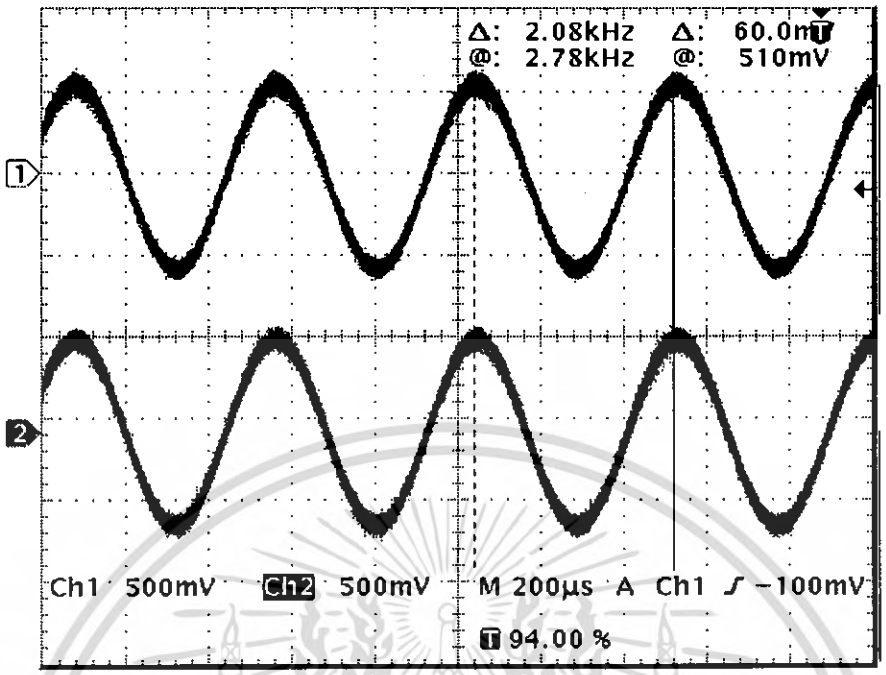
ครั้งที่	ความถี่ที่กำหนด (เฮิรตซ์)	ความถี่จากการทดลอง (เฮิรตซ์)	% คลาดเคลื่อน
1	689.1	694	0.71
2	2067.2	2080	0.62



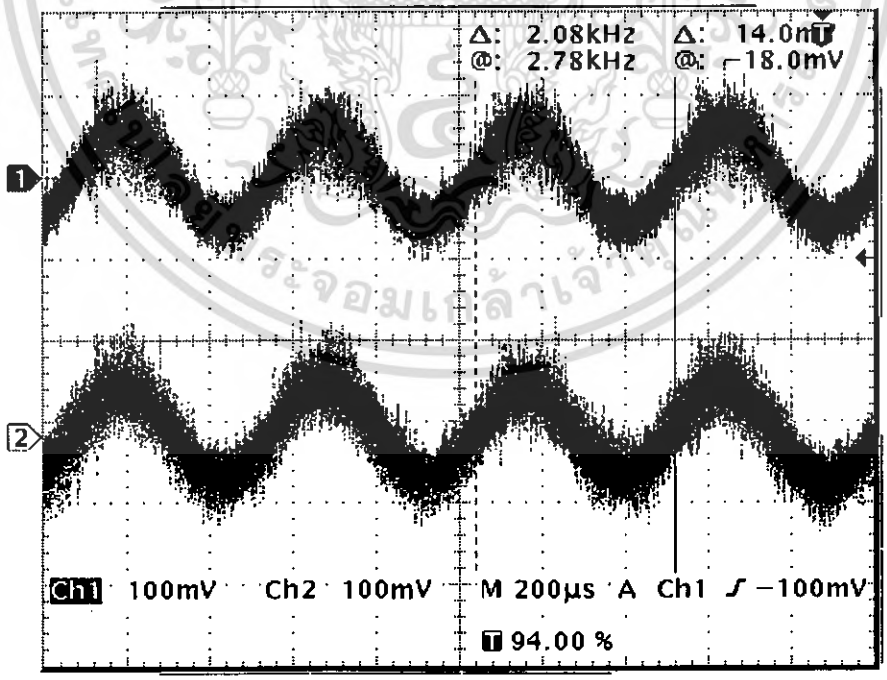
รูปที่ 4.1 สัญญาณ Sine ความถี่ 694 เฮิรตซ์ ระดับความดัง -0 เดซิเบล



รูปที่ 4.2 สัญญาณ Sine ความถี่ 694 เฮิรตซ์ ระดับความดัง -20 เดซิเบล



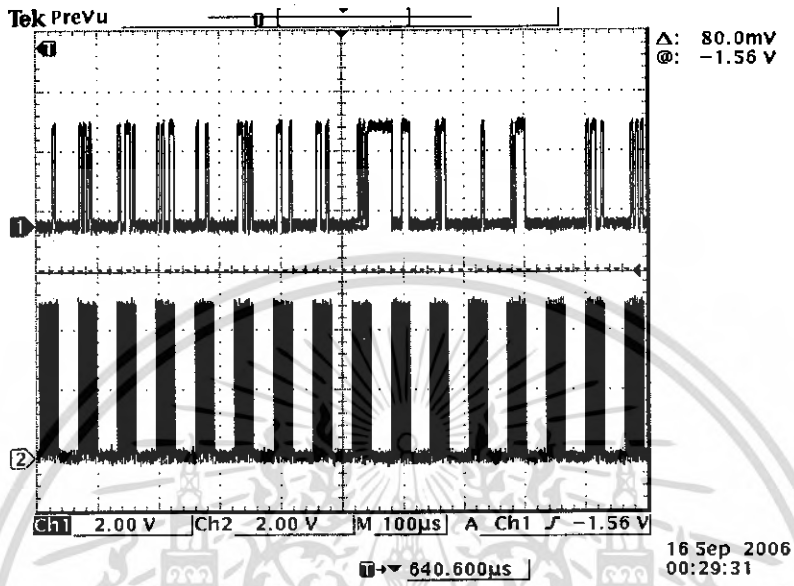
รูปที่ 4.3 สัญญาณ Sine ความถี่ 2.08 kHz ระดับความดัง -0 dB



รูปที่ 4.4 สัญญาณ Sine ความถี่ 2.08 kHz ระดับความดัง -20 dB

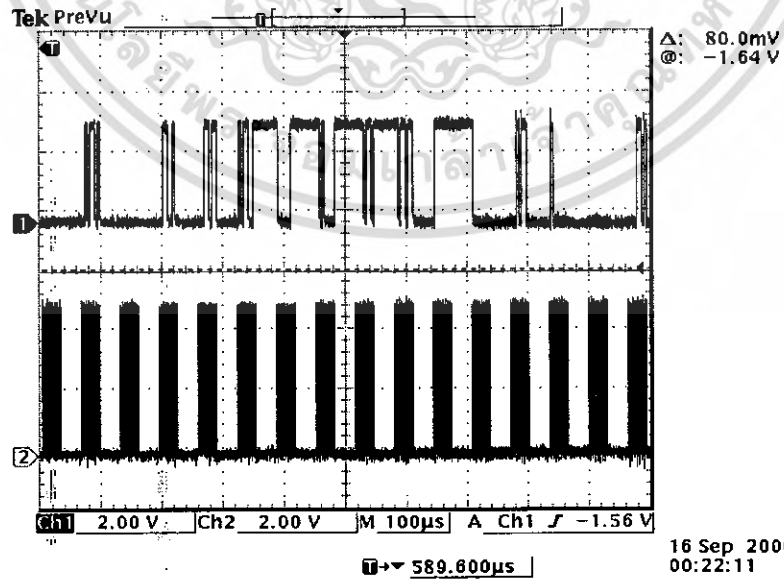
การทดลองที่ 2 วัดสัญญาณข้อมูลที่ SD Card ส่งกลับมา

การทดลองที่ 2.1 วัดสัญญาณข้อมูลที่ SD Card ส่งกลับมาหลังจากได้รับคำสั่งอ่านรีจิสเตอร์ CID



รูปที่ 4.5 สัญญาณข้อมูลของรีจิสเตอร์ CID (CH1) เทียบกับสัญญาณนาฬิกา (CH2)

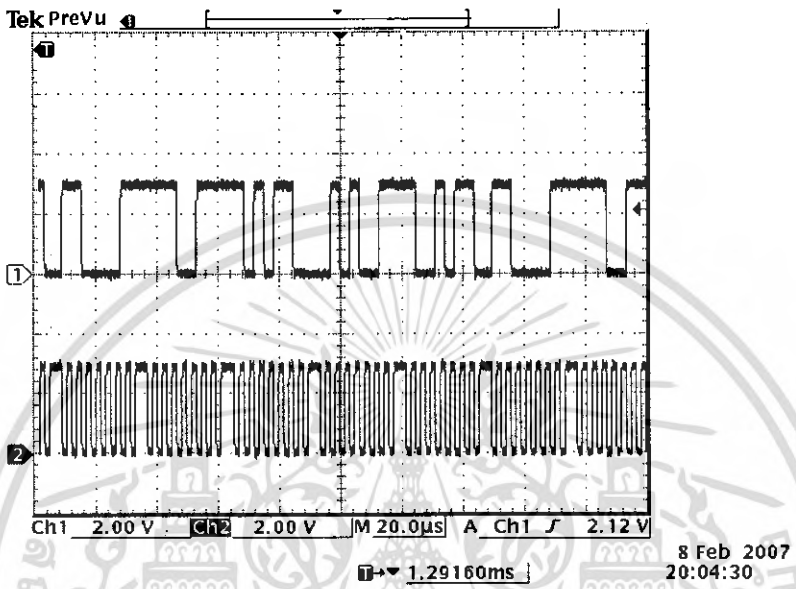
การทดลองที่ 2.2 วัดสัญญาณข้อมูลที่ SD Card ส่งกลับมาหลังจากได้รับคำสั่งอ่านรีจิสเตอร์ CSD



รูปที่ 4.6 สัญญาณข้อมูลของรีจิสเตอร์ CSD (CH1) เทียบกับสัญญาณนาฬิกา (CH2)

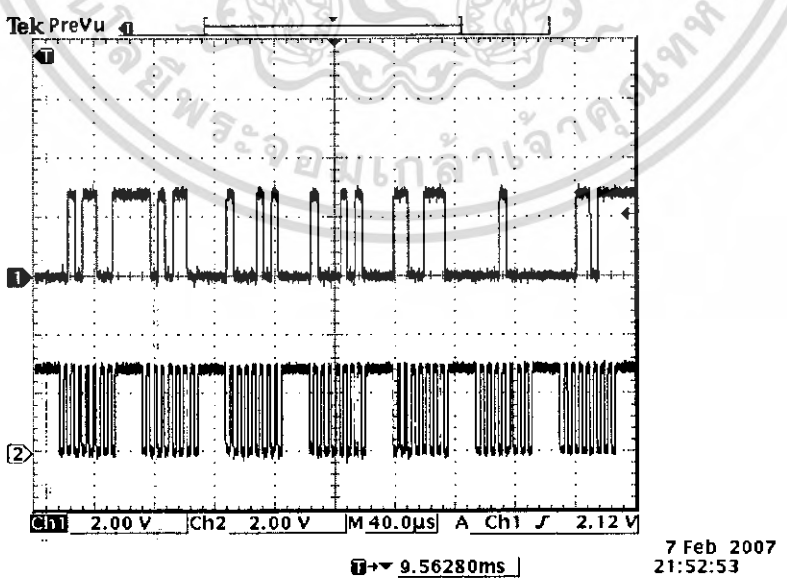
การทดลองที่ 3 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus

การทดลองที่ 3.1 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus โดยใช้พอร์ต SPI ของไมโครคอนโทรลเลอร์



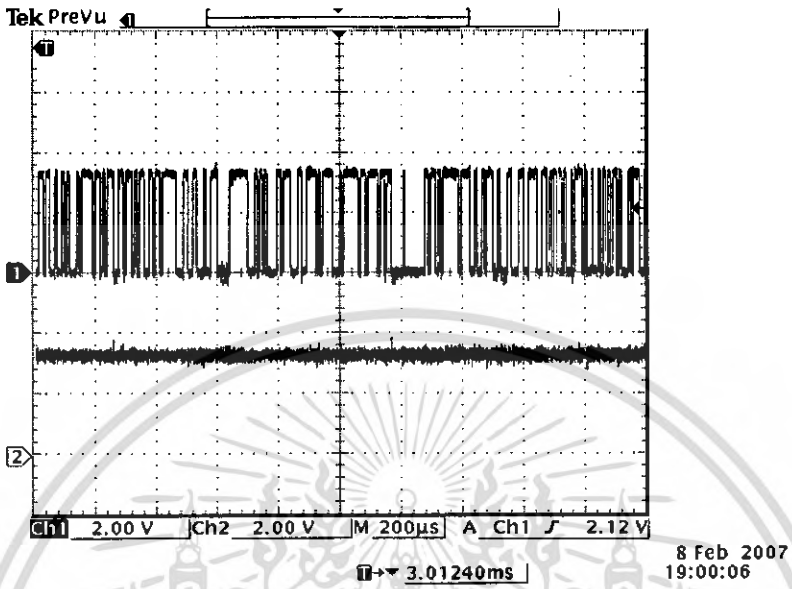
รูปที่ 4.7 สัญญาณข้อมูล(CH1) เทียบกับสัญญาณนาฬิกา (CH2) โดยอัตราบิตสูงสุดที่ทำได้ คือ 5Mbps

การทดลองที่ 3.2 วัดสัญญาณของการติดต่อแบบอนุกรมในลักษณะ SPI Bus จากฟังก์ชันที่เขียนเอง

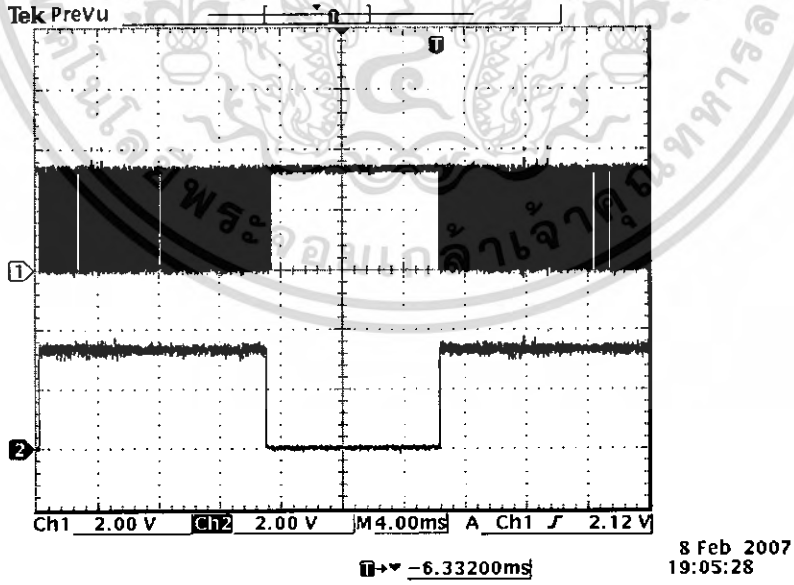


รูปที่ 4.8 สัญญาณข้อมูล(CH1) เทียบกับสัญญาณนาฬิกา (CH2) โดยอัตราบิตสูงสุดที่ทำได้ คือ 270 kbps

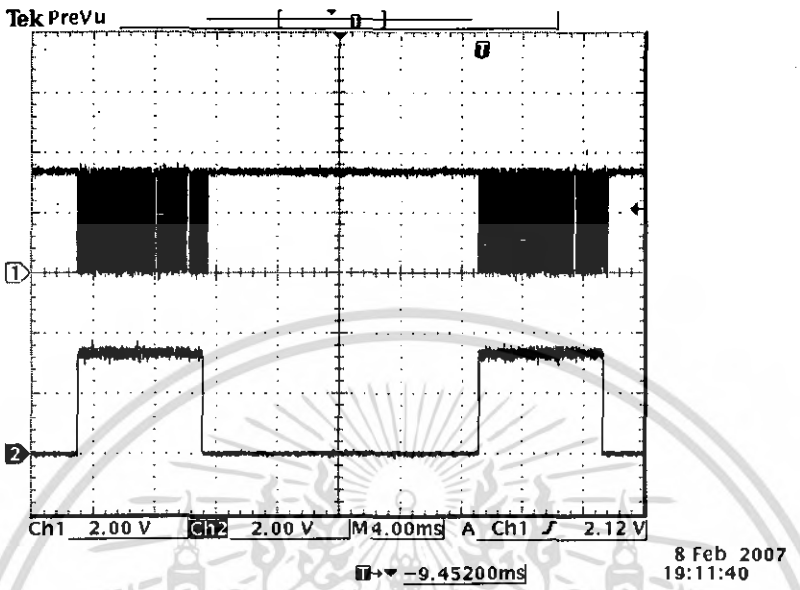
การทดลองที่ 4 ทดสอบหาอัตราเร็วที่เหมาะสมในการส่งข้อมูลให้กับตัวถอดรหัส MP3



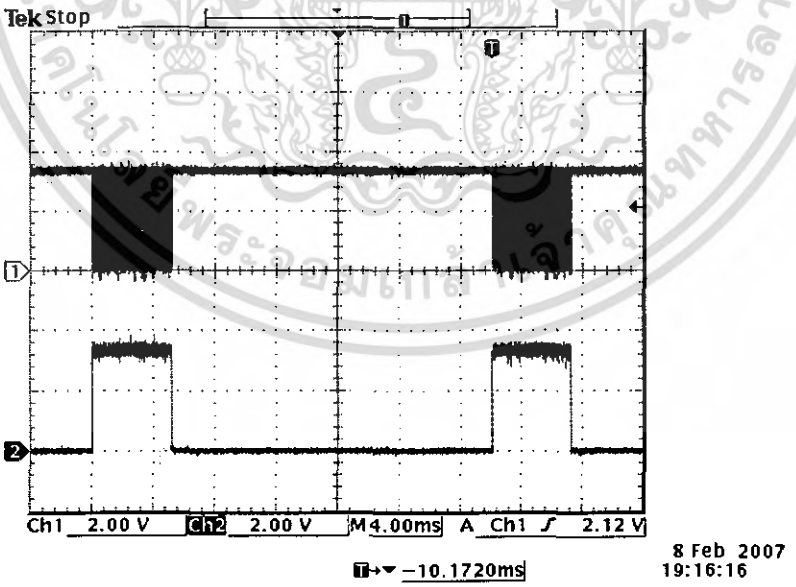
รูปที่ 4.9 สัญญาณข้อมูล (CH1) เทียบกับสัญญาณ Data Request (CH2) โดยอัตราบิตเท่ากับ 125kbps



รูปที่ 4.10 สัญญาณข้อมูล (CH1) เทียบกับสัญญาณ Data Request (CH2) โดยอัตราบิตเท่ากับ 250kbps



รูปที่ 4.11 สัญญาณข้อมูล (CH1) เทียบกับสัญญาณ Data Request (CH2) โดยอัตราบิตเท่ากับ 500kbps



รูปที่ 4.12 สัญญาณข้อมูล (CH1) เทียบกับสัญญาณ Data Request (CH2) โดยอัตราบิตเท่ากับ 1Mbps

บทที่ 5
บทสรุป

5.1 สรุป

ในรายงานได้กล่าวถึงความเป็นมาของโครงงาน แนวคิด ทฤษฎี และรายละเอียดการสร้างเครื่องเล่น MP3 ชนิดพกพามี SD Card รวมถึงการทดสอบการทำงานเบื้องต้น ในการทดสอบการติดต่อระหว่างไมโครคอนโทรลเลอร์ (PIC18F4539) กับตัวถอดรหัส MP3 (VS1011B) กระทำโดยให้ตัวถอดรหัส MP3 ถอดรหัสข้อมูลสัญญาณ Sine จากไมโครคอนโทรลเลอร์และสามารถปรับระดับความดังได้ และยังทำการทดสอบหาอัตราเร็วต่ำสุดในการส่งข้อมูลให้กับตัวถอดรหัส MP3 ที่ทำให้ตัวถอดรหัสยังสามารถถอดรหัสได้ถูกต้อง โดยสังเกตจากสัญญาณ Data Request จากการทดลองได้อัตราเร็วต่ำสุดประมาณ 250kbps ส่วนการทดสอบการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card กระทำโดยให้ไมโครคอนโทรลเลอร์อ่านค่ารีจิสเตอร์จาก SD Card 2 รีจิสเตอร์ คือ รีจิสเตอร์ CID และ CSD โดยการติดต่อทั้งหมดระหว่างไมโครคอนโทรลเลอร์กับตัวถอดรหัส MP3 และ SD Card เป็นการติดต่อในระบบ SPI BUS โดยเรายังทำการทดลองเปรียบเทียบสัญญาณลักษณะ SPI Bus จากพอร์ต SPI ของไมโครคอนโทรลเลอร์ PIC กับสัญญาณลักษณะ SPI จากฟังก์ชันที่เขียนขึ้นมาเองด้วย ซึ่งสัญญาณลักษณะ SPI ทั้งสองแบบมีลักษณะของสัญญาณเช่นเดียวกัน โดยอัตราบิตสูงสุดที่ทำได้จากฟังก์ชันที่เขียนขึ้นมาเองมีค่าเท่ากับ 270kbps

เครื่องเล่น MP3 ต้นแบบนี้แบ่งการติดต่อในลักษณะ SPI ออกเป็น 2 ส่วน คือ ส่วนแรกเป็นการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card โดยใช้พอร์ต SPI ที่มีอยู่แล้วในตัวไมโครคอนโทรลเลอร์ และส่วนที่สองเป็นการติดต่อระหว่างไมโครคอนโทรลเลอร์กับตัวถอดรหัส MP3 โดยใช้สัญญาณ SPI จากฟังก์ชันที่เขียนขึ้นมาเองซึ่งมีอัตราบิตสูงสุด 270kbps แต่ในการทำงานของเครื่องเล่นต้นแบบจริง ไม่ได้ส่งข้อมูลให้ตัวถอดรหัส MP3 อยู่ตลอดเวลา เนื่องจากต้องสับเปลี่ยนไปนำข้อมูลใหม่จาก SD Card มาทำการถอดรหัสด้วยรวมถึงต้องมีการตรวจสอบอินพุทต่างๆ ทำให้อัตราบิตจริงที่ส่งข้อมูลให้ตัวถอดรหัส MP3 ต่ำกว่า 270kbps มาก และเนื่องจากอัตราบิตต่ำไปนี่เอง เป็นสาเหตุทำให้เสียงที่ได้หลังจากทำการถอดรหัสแล้วขาดๆหายๆ ในการแก้ปัญหาส่วนนี้อาจทำได้โดยเปลี่ยนไมโครคอนโทรลเลอร์ให้มีความเร็วในการทำงานสูงขึ้น หรืออาจเลือกใช้ไมโครคอนโทรลเลอร์ที่มีพอร์ต SPI มากกว่า 1 พอร์ต ซึ่งสามารถกำหนดอัตราเร็วในการส่งข้อมูลได้อย่างอิสระ

ในส่วนของอินพุทสามารถใช้งานได้ตามที่ออกแบบคือ สามารถเล่นเพลง หยุดเพลง เลือกเพลง รวมถึงปรับระดับความดังของเสียงได้ โดยชื่อเพลงนั้นจะแสดงผ่านทางจอแอลซีดี

5.2 ปัญหาและแนวทางแก้ไข

- เนื่องจากอุปกรณ์หลัก (ตัวถอดรหัส MP3 และซ็อกเก็ตของ SD Card) เป็นเทคโนโลยีเซอร์เฟสเมาท์ (Surface mount) ทำให้ยากต่อการทดสอบ ผู้จัดทำจึงต้องออกแบบพีซีบี (PCB) เพื่อเปลี่ยนลักษณะขาให้สามารถทดลองในบอร์ดทดลองได้

- เนื่องจาก SD Card มีการจัดเรียงข้อมูลในระบบ FAT32 ในการตรวจสอบความถูกต้องจากการอ่านข้อมูลภายใน SD Card จึงทำได้ยาก ดังนั้นผู้จัดทำจึงใช้ซอฟต์แวร์ (Winhex) มาช่วยตรวจสอบข้อมูลและตำแหน่งแอดเดรสภายใน SD Card

- เนื่องจากผู้จัดทำไม่สามารถหาซื้อไมโครคอนโทรลเลอร์ตระกูล PIC ชนิดกินกำลังต่ำได้ ในการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับตัวถอดรหัส MP3 และ SD Card จึงต้องทำการลดระดับแรงดันโดยใช้ตัวต้านทาน หรือใช้ไอซีแปลงระดับแรงดัน

- เนื่องจากสัญญาณ SPI ที่สร้างขึ้นเองโดยใช้ไมโครคอนโทรลเลอร์ PIC มีอัตราเร็วในการส่งข้อมูลให้กับตัวถอดรหัส MP3 ไม่สูงพอ ทำให้เสียงที่ได้ขาดๆหายๆ ในการแก้ปัญหานั้นอาจทำได้โดยเปลี่ยนไมโครคอนโทรลเลอร์ให้มีความเร็วในการทำงานสูงขึ้น หรือเลือกใช้ไมโครคอนโทรลเลอร์ที่มีพอร์ต SPI 2 พอร์ต ซึ่งสามารถกำหนดอัตราเร็วในการรับ/ส่งข้อมูลได้ตามต้องการ

- ในการทำงานของเครื่องเล่น MP3 ที่ออกแบบนั้นสามารถเล่นเพลงได้จำนวนจำกัด เนื่องจากมี RAM ที่ใช้ในการเก็บรายละเอียดและตำแหน่งของข้อมูลเพลงไม่เพียงพอ โดยรายละเอียดของเพลงหนึ่งเพลงจะใช้เนื้อที่ 32 ไบต์ (สำหรับเพลงที่มีชื่อเพลงยาวไม่เกิน 8 ตัวอักษร) ถ้าเรามี RAM ขนาด 1024 ไบต์ เราจะสามารถเก็บรายละเอียดของเพลงได้สูงสุด $\frac{1024}{32} = 32$ เพลง แต่ในความเป็นจริงจะเก็บได้น้อยกว่านี้ เพราะต้องกันพื้นที่ RAM ส่วนหนึ่งคอยเก็บข้อมูลจาก SD Card รวมถึงเป็นหน่วยความจำคอยเก็บค่าของตัวแปรต่างๆขณะทำงานอีกด้วย สำหรับเครื่องเล่น MP3 ที่ออกแบบนั้นใช้ไมโครคอนโทรลเลอร์ที่มี RAM ขนาด 1408 ไบต์ แบ่งเป็นพื้นที่เก็บข้อมูลเพลงจาก SD Card 512 ไบต์ เก็บค่าของตัวแปรต่างๆขณะประมวลผล 74 ไบต์ และเหลือพื้นที่ไว้เก็บรายละเอียดและตำแหน่งเพลง 822 ไบต์ ดังนั้นจะเก็บรายละเอียดและตำแหน่งของเพลงได้ประมาณ 25 เพลง แนวทางแก้ปัญหาให้สามารถเก็บเพลงได้เยอะขึ้นทำได้โดยเลือกใช้ไมโครคอนโทรลเลอร์ที่มีขนาดของ RAM สูงขึ้น

5.3 ประโยชน์ที่ได้รับ

- ในปัจจุบันไฟล์ MP3 กำลังเป็นที่นิยมเนื่องด้วยขนาดของไฟล์มีขนาดเล็ก ทำให้เก็บข้อมูลได้เป็นจำนวนมาก จากการศึกษารูปแบบของไฟล์ MP3 ทำให้เข้าใจหลักการในการบีบอัดไฟล์และการแปลงไฟล์จาก MP3 เป็น PCM

- เนื่องจาก SD Card มีการจัดเรียงไฟล์แบบ FAT32 ทำให้ผู้จัดทำมีความเข้าใจในตัวระบบ FAT32 เพื่อใช้ในการติดต่อกับ SD Card

- เพิ่มพูนทักษะในการใช้งานไมโครคอนโทรลเลอร์ตระกูล PIC และเพิ่มทักษะในการเขียนโปรแกรม

บรรณานุกรม

1. ประจิน พลังสันติกุล, เรียนรู้และใช้งาน CCS C คอมไพเลอร์ เขียนโปรแกรมภาษา C ควบคุมไมโครคอนโทรลเลอร์ PIC, บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด, กรุงเทพฯ
2. ณัฏฐพล วงศ์สุนทรชัยและชัยวัฒน์ ลิ้มพรจิตรวิไล, เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ PIC16F628, บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด, กรุงเทพฯ
3. http://en.wikipedia.org/wiki/File_Allocation_Table, File Allocation Table
4. http://www.thaidev.com, OGG อีกหนึ่งเทคโนโลยีเพลงดิจิตอล
5. http://www.nsitez.net/nvillage, NTFS vs FAT
6. http://www.pcmag.com, Definition of MP3

ภาคผนวก

ภาคผนวก ก

โค้ดโปรแกรม (Source Code)

การเขียนโปรแกรมใช้โปรแกรม CCS C Compiler

โปรแกรมหลัก

```
#define _main_
#include <18F4539.h>
#include "Board.c"
#include "VS1011B.c"
#include "SD_Card.c"
#define use_portb_lcd
#include "LCD.c"

#fuses HS,NOWDT                 //High speed crystal, No watchdog timer
#fuses NOLVP                    //No low voltage protection
#use delay(clock=20000000)      //use built-in function delay with clock speed 20MHz

#define msg0 "**MP3** "

void main(){
#bit CKE=0xFC7.6
/***** Declare Variable *****/
 unsigned char volume_values[] = {254, 76, 72, 68, 64, 60, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20,
16, 12, 8, 4, 0};
 int32 sound_pointer, root_addr, start_addr, file_size, root_start;
 int16 i, sec_total, sec_num, rsv_size, fat_size, byte_per_sec;
 int counter_p, counter_n, file_ad[4], vol_pos, fat, root_ad[4];
 int1 play_en;
 unsigned char data[512];
 char str[8], Detail[512];  //file name byte 0-7, extension byte 8-10
                            //start cluster high 2 bytes : byte 21,20
                            //start cluster low  2 bytes : byte 27,26
                           //file size : byte 28-31
/***** Set Tristate *****/
 set_tris_c(0xd0);          //c7, c6, c4 = input , c5, c3, c2, c1, c0 = output
```

```
  set_tris_d(0x1f);          //d4, d3, d2, d1, d0 =input , d7, d6, d5 = output

 /***** Set up SPI *****/
  setup_spi(spi_master|spi_l_to_h|spi_clk_div_4); //idle state is low, bit rate = ext_clk/4 = 5Mbps
  CKE=1;    //detect data at clock rising edge

  delay_ms(2000);
 /****** Reset Device *****/
  lcd_init();
  LCD_Command(0x01); //clear LCD
  LCD_Command(0x80); //start 1st line
  strcpy(str,msg0);
  LCD_String(str,0);
  while(SDReboot());      //Reboot SD Card
  MP3Reset();             //Reset MP3 Decoder
  counter_n = 0;
  counter_p = 0;
  play_en = 0;

 /* Find start address of root directory */
  while (SDCommand(0x51,0x00,0x00,0x00,0x00));
  SDWaitForData();
  for(i=0;i<512;i++){
    Detail[i] = spi_read(0xff);
  }
  rsv_size = Detail[15];
  rsv_size = rsv_size<<8;
  rsv_size += Detail[14];
  fat = Detail[16];
  fat_size = Detail[37];
  fat_size = fat_size<<8;
  fat_size += Detail[36];
```

```
  root_start = rsv_size + (fat*fat_size);
  root_start = root_start*0x200;
  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;

/***** Find byte per sector *****/
byte_per_sec = Detail[12];
byte_per_sec = byte_per_sec<<8;
byte_per_sec += Detail[11];

  /* Get File Detail in root directory (start address, file name, file size)*/
  while (SDCommand(0x51,root_ad[0],root_ad[1],root_ad[2],root_ad[3]));
  SDWaitForData();
  for(i=0;i<512;i++){
    Detail[i] = spi_read(0xff);
  }

GetFileDetail:
  /***** cal start address of file *****/
  start_addr = Detail[21+(counter_n*32)];
  start_addr = start_addr<<8;
  start_addr += Detail[20+(counter_n*32)];
  start_addr = start_addr<<8;
  start_addr += Detail[27+(counter_n*32)];
  start_addr = start_addr<<8;
  start_addr += Detail[26+(counter_n*32)];
  start_addr = start_addr * 0x1000;

  /***** cal file size *****/
  file_size = Detail[31+(counter_n*32)];
```

```
  file_size = file_size<<8;
  file_size += Detail[30+(counter_n*32)];
  file_size = file_size<<8;
  file_size += Detail[29+(counter_n*32)];
  file_size = file_size<<8;
  file_size += Detail[28+(counter_n*32)];

  /****** cal how many sector per file ******/
  sec_total = file_size/byte_per_sec;

  /***** Check MP3 file or not? *****/
  if ( (Detail[8+(counter_n*32)] == 0x4D) && (Detail[9+(counter_n*32)] == 0x50) &&
(Detail[10+(counter_n*32)] == 0x33) )  //0x4d=M, 0x50=P, 0x33=3
  {
   /***** Show file name on LCD *****/
   LCD_Command(0x01);
   LCD_Command(0x80);
   for(i=0;i<8;i++){
    lcd_putc(Detail[i+(counter_n*32)]);
   }

check_play:
   while (true){
    delay_ms(200);
    if ( ( !(input(PLAY)) ) || (play_en == 1) ){       //check key input
     goto PlaySong;
    }
    if ( !(input(NEXT)) ){
     counter_n++;
     goto GetFileDetail;
    }
   }
```

```
}
/***** Check WAV file or not? *****/
else if( (Detail[8+(counter_n*32)] == 0x57) && (Detail[9+(counter_n*32)] == 0x41) &&
(Detail[10+(counter_n*32)] == 0x56) )  //0x57=W, 0x41=A, 0x56=V
{
 /***** Show file name on LCD *****/
 LCD_Command(0x01);
 LCD_Command(0x80);
 for(i=0;i<8;i++){
  lcd_putc(Detail[i+(counter_n*32)]);
 }
 while (true){
  if ( (!input(PLAY)) || (play_en == 1) )
    goto PlaySong;
  if (!input(NEXT)){
    counter_n++;
    goto GetFileDetail;
  }
 }
}

/* if play until end of last song -> play again at 1st song */
else if (Detail[0+(counter_n*32)] == 0x00)
{
 counter_n = 0;
 goto GetFileDetail;
}

/* if file is not MP3&WAV get new file detail */
else
{
 counter_n++;
```

```
  goto GetFileDetail;
 }

PlaySong:
 play_en = 1;
 sec_num = 0;
 vol_pos = 16;
 output_high(PIN_A2);
 delay_ms(500);
 output_low(PIN_A2);
 MP3SelectData();        //enable MP3 Decoder to receive data
 while(sec_num < sec_total){
 /***** find address of each sector *****/
  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;
  while (SDCommand(0x51,file_ad[0],file_ad[1],file_ad[2],file_ad[3]));
  SDWaitForData();
  /***** receive data from SD Card *****/
  for(i=0;i<512;i++){
   data[i] = SPIGetChar();
  }

  sound_pointer=0;
  while(sound_pointer < 512){
   while( !(input(MP3_DREQ)) );
   if (!input(VOL_UP)){          //check key input volume control
    MP3DeselectData();
    if(vol_pos == 21){
     vol_pos = 21;
```

```
  }
  else{
   vol_pos++;
  }
  MP3SetVolume(volume_values[vol_pos],volume_values[vol_pos]);
  MP3SelectData();
 }
 if (!input(VOL_DOWN)){
  MP3DeselectData();
  if(vol_pos == 0){
   vol_pos = 0;
  }
  else{
   vol_pos--;
  }
  MP3SetVolume(volume_values[vol_pos],volume_values[vol_pos]);
  MP3SelectData();
 }
 /***** transmit data to MP3 Decoder *****/
 for(i=0;i<8;i++){
  SPIPut(data[i]);
  sound_pointer++;
 }
 if (!input(NEXT)){    //check key input choose song
  counter_p = counter_n;
  counter_n++;
  goto GetFileDetail;
 }
 if (!input(PREV)){
  counter_n = counter_p;
  goto GetFileDetail;
 }
```

```
      if (!input(STOP)){
        play_en = 0;
        goto check_play;
      }
    }
    sec_num++;
  }
  counter_p = counter_n;
  counter_n++;
  goto GetFileDetail;
}
```

ไฟล์ Board.c

```
/*********************************************
File        : Board.c
Purpose   : define PIC pin, Select func.
*********************************************/
#ifndef _BOARD_
#define _BOARD_

/***** MP3 Decoder Connection *****/
#define  MP3_XRESET  PIN_D7
void MP3PutInReset(){output_low(MP3_XRESET);}
void MP3ReleaseFromReset(){output_high(MP3_XRESET);}

#define  MP3_XCS  PIN_D6
void MP3SelectControl(){output_low(MP3_XCS);}
void MP3DeselectControl(){output_high(MP3_XCS);}

#define  MP3_BSYNC  PIN_D5
void MP3SelectData(){output_high(MP3_BSYNC);}
void MP3DeselectData(){output_low(MP3_BSYNC);}

#define  MP3_DREQ  PIN_D4

/***** SD Card Connection *****/
#define  SD_CS  PIN_A3
void SDSelect(){output_low(SD_CS);}
void SDDeselect(){output_high(SD_CS);}

/***** Manual SPI *****/
#define SCK PIN_A0
#define SDO PIN_A1
#endif
```

ไฟล์ VS1011B.c

```
/*******************************************
File       : VS1011B.c
Purpose    : build func. about VS1011B
*******************************************/
#ifndef _VS1011B_
#define _VS1011B_

#use delay(clock=20000000)

#define VS_WRITE_COMMAND 0x02
#define SCI_MODE         0x0
#define SCI_STATUS       0x1
#define SCI_BASS         0x2
#define SCI_CLOCKF       0x3
#define SCI_DECODE_TIME  0x4
#define SCI_AUDATA       0x5
#define SCI_WRAM         0x6
#define SCI_WRAMADDR     0x7
#define SCI_HDAT0        0x8
#define SCI_HDAT1        0x9
#define SCI_AIADDR       0xa
#define SCI_VOL          0xb
#define SCI_AICTRL0      0xc
#define SCI_AICTRL1      0xd
#define SCI_AICTRL2      0xe
#define SCI_AICTRL3      0xf

/*******************************************
  Funetion      : SPIPut
  Description   : send SPI data 1 byte
  Parameters    : SPIData
```

```
 Returned        : nothing
*********************************************/
void SPIPut(out){
 char COUNT;
 #bit  OUT_BIT7=out.7;
        COUNT=0;
        for (COUNT=0;COUNT<8;COUNT++){
            output_low(SCK);              //falling edge of clock
            output_bit(SDO,OUT_BIT7); //prepare data
            out = out<<1;                 // shift output byte for next loop
            output_high(SCK);             // rising edge of clock
        }
}
/*********************************************
 Function        : MP3WriteRegister
 Description     : send VS1011B's SCI command
 Parameters      : addressbyte, highbyte, lowbyte
 Returned        : nothing
*********************************************/
void MP3WriteRegister(addressbyte, highbyte, lowbyte){
        MP3Selectcontrol();
        SPIPut(VS_WRITE_COMMAND);
        SPIPut(addressbyte);
        SPIPut(highbyte);
        SPIPut(lowbyte);
        delay_us(100);   //After sending SCI command, it's not allowed to
                         //send SCI or SDI data for 5 microseconds
        MP3DeselectControl();
}
/*********************************************
 Function        : MP3SetVolume
 Description     : Set MP3 volume
```

```
 Parameters      : leftchannel, rightchannel
 Returned        : nothing
*********************************************/
void Mp3SetVolume(leftchannel, rightchannel){
  Mp3WriteRegister(SCI_VOL, leftchannel, rightchannel);
}
/*********************************************
 Function        : MP3SendZero
 Description     : send 2048 zeros to VS1011B
 Parameters      : nothing
 Returned        : nothing
*********************************************/
void MP3SendZero(){
int16 i;
 MP3SelectData();
 while(c<2048){
  while( !(input(MP3_DREQ)) );
  for (i=0;i<8;i++){
   SPIPut(0x00);
   c++;
  }
 }
 MP3DeselectData();
}
/*********************************************
 Function        : MP3SoftReset
 Description     : VS1011B Soft Resct
 Parameters      : nothing
 Returned        : nothing
*********************************************/
 void MP3SoftReset(){
        MP3WriteRegister(SCI_MODE,0x00,0x04);       //SCI_MODE have 14bits
```

```
        delay_ms(1);                    //0004h = 00 0000 0000 0100b
                                        //1 : set SM_SoftReset
        while( !(input(MP3_DREQ)) );  //wait for start up (after software reset
                                        //DREQ stay low '0' at least 250us
        MP3WriteRegister(SCI_CLOCKF,0x80,0x00);     //double clock speed
/* Send null bytes to data interface */
        MP3SelectData();
        SPIPut(0);
        SPIPut(0);
        SPIPut(0);
        SPIPut(0);
        delay_ms(1);
        MP3DeselectData();
        MP3SendZero();
}
/*****************************************
 Function        : MP3Reset
 Description     : VS1011B Reset
 Parameters      : nothing
 Returned        : nothing
*****************************************/
void MP3Reset(){
/***** Reset MP3 Decoder *****/
        MP3PutInReset();
        delay_ms(100);
/***** Send dummy SPI byte to initialize SPI *****/
        SPIPut(0xff);
/***** Un-reset MP3 chip *****/
        MP3DeselectControl();
        MP3DeselectData();
        MP3ReleaseFromReset();
        while( !(input(MP3_DREQ)) );    //wait for DREQ
```

```
/***** Slow sample rate for slow analog part startup *****/
        MP3WriteRegister(SCI_AUDATA,0x00,0x0a);  //Set sample rate 10Hz
        delay_ms(100);
/***** Switch on the analog parts *****/
        MP3SetVolume(0xfe,0xfe); //set volume attenuate 254*(-0.5dB)= -127dB
        MP3WriteRegister(SCI_AUDATA,31,64);  //Set sample rate 8kHz
        MP3SetVolume(0,0);  //set volume maximum
        MP3SoftReset();
}
/*********************************************
 Function        : MP3SineTest
 Description     : test sine sound
 Parameters      : nothing
 Returned        : nothing
*********************************************/
void MP3SineTest(){
/***** Send SCI command for Sine test *****/
  MP3WriteRegister(SCI_MODE, 0x00, 0x20);  //SCI_MODE have 14bits
                        //0020h=00 0000 0010 0000
                        //1 : SM_TESTS
  delay_us(100);
  MP3DeselectControl();

  while( !(input(MP3_DREQ)) );
/***** Send a Sine Test Header to Data port *****/
        MP3SelectData();
        SPIPut(0x53);    //Sine test initialized with the 8-byte sequence
        SPIPut(0xef);    //0x53 0xef 0x6e n 0 0 0 0
        SPIPut(0x6e);    //n = 0x02 :sample rate=44.1kHz,sine skip speed=2
        SPIPut(0x02);    //freq = 691Hz
        SPIPut(0x00);
        SPIPut(0x00);
```

```
        SPIPut(0x00);
        SPIPut(0x00);
        delay_us(100);
        MP3DeselectData();
        delay_ms(3000);

/***** Stop the sine test sound *****/
        MP3SelectData();
        SPIPut(0x45);
        SPIPut(0x78);
        SPIPut(0x69);
        SPIPut(0x74);
        SPIPut(0x00);
        SPIPut(0x00);
        SPIPut(0x00);
        SPIPut(0x00);
        delay_us(100);
        MP3DeselectData();
        delay_ms(500);
}
#endif
```

ไฟล์ SD_Card.c

```
/*******************************************

File        : SD_Card.c

Purpose  : build functions about SD Card

Update    : Aug 30th, 2006

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

#ifndef _SD_Card_

#define _SD_Card_

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

 Function     : SPI8Clocks

 Description : send 8 clock cycles

 Parameters  : nclocks

 Returned     : nothing

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

void SPI8Clocks(unsigned char nClocks){

  while(nclocks--){

          spi_write(0xff);

        }

}

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

 Function          : SPIGetChar

 Description       : get response from SD Card

 Parameters        : nothing

 Returned          : data on data_out line

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

unsigned char SPIGetChar(){

  spi_write(0xff);

  return  spi_read();

}

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

 Function     : SDCommand

 Description : send command to SD Card
```

```
  Parameters  : c1, c2, c3, c4, c5
  Returned    : Response from SD Card (c2)
*********************************************/
unsigned char SDCommand(unsigned char c1,unsigned char c2,unsigned char c3,unsigned char
c4,unsigned char c5){
//c1 = command byte, c2 = argument byte 1, c3 = argument byte 2,
//c4 = argument byte 3, c5 = argument byte 4

  unsigned char c,dat_out;
  SDDeselect();
/* Provide clock edge before and after asserting SD CS */
  SPI8Clocks(8);  // NES = 8 Clock Cycles
  SDSelect();
  SPI8Clocks(8);  // NCS = 8 Clock Cycles

  c=0;
/* If card still seems to be busy, give it some time... */
  while( ((dat_out=SPIGetChar()) !=0xff) && (c<100) ){
    c++;
  }
/* The bus should be stable high now */
  if (dat_out != 0xff){
    SDDeselect();
    return 0x81;
  }
/* Send SD Command */
  spi_write(c1);
  spi_write(c2);
  spi_write(c3);
  spi_write(c4);
  spi_write(c5);
  spi_write(0x95);  //Valid CRC for init, then don't care
```

```
/* Now ok to use c1..c5 as temporaries */
/* Wait for R1 style response (bit 7 low) from SD Card */
  c1=100;  //try max 100 times  max NCR(time from end of command
           //to response)= 8 Clock Cycles = 64 clocks

  while( ((c2=SPIGetChar()) & 0x80) && (c1--) );
  return  c2;
}
/*****************************************
 Function     : SDWaitForData
 Description : wait for start block of data (0xfe)
 Parameters  : nothing
 Returned    : result of waiting. 0 : Ok, 5 : fail to recieve start block of data token
*****************************************/
unsigned char SDWaitForData(){
  unsigned char  result;
// Wait until something else than 0xff is read from the bus
  do{
    result=SPIGetChar();
  }while(result==0xff);
// Something was received from the bus
  if (result!=0xfe){  //Read fail, so I should flush any data that might be pending from SD Card
    SPI8Clocks(200); // Flush SD by sending lots of FF's to it
    SPI8Clocks(200);
    SPI8Clocks(200);
    SDDeselect();
    return 5;
  }
  return 0;
}
/*****************************************
 Function          : SDReboot
```

```
  Description      : reboot SD Card (software reset)
  Parameters       : nothing
  Returned         : result of reboot. 0 : Ok, 2 : Error
*********************************************/
unsigned char SDReboot(){
  unsigned char c;
  SDSelect();
  for (c=0;c<200;c++){
   spi_write(0xff);
  }

  while (SDCommand(0x40,0,0,0,0) != 0x01 );
  SDCommand(0x7b,0,0,0,0);

  c=255;
  while( (c--) && (SDCommand(0x41,0,0,0,0)) ){  //max.255times SD-wake up call.
                                                //If not Idle SD return 0x00
   delay_ms(10);
   if (c==1)  //Time-out
   return 2;  //Not able to power up SD Card
  }
  return 0;  //Reboot Ok
}
#endif
```

ไฟล์ LCD.c

```
////////////////////////////////////////////////////////////////////////
////                    LCDD.C                                      ////
////            Driver for common LCD modules                       ////
////                                                                ////
//// lcd_init()   Must be called before any other function.         ////
////                                                                ////
//// lcd_putc(c)  Will display c on the next position of the LCD.   ////
////                The following have special meaning:             ////
////                 \f  Clear display                              ////
////                 \n  Go to start of second line                 ////
////                 \b  Move back one position                     ////
////                                                                ////
//// lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)  ////
////                                                                ////
//// lcd_getc(x,y)   Returns character at position x,y on LCD       ////
////                                                                ////
////////////////////////////////////////////////////////////////////////
////      (C) Copyright 1996,2003 Custom Computer Services          ////
//// This source code may only be used by licensed users of the CCS C ////
//// compiler.  This source code may only be distributed to other     ////
//// licensed users of the CCS C compiler.  No other use, reproduction ////
//// or distribution is permitted without written permission.        ////
//// Derivative programs created using this software in object code  ////
//// form are not restricted in any way.                             ////
////////////////////////////////////////////////////////////////////////

// As defined in the following structure the pin connection is as follows:
//    D0  enable
//    D1  rs
//    D2  rw
//    D4  D4
```

```
//    D5  D5
//    D6  D6
//    D7  D7
//
//  LCD pins D0-D3 are not used and PIC D3 is not used.

// Un-comment the following define to use port B
// #define use_portb_lcd TRUE

struct lcd_pin_map {                // This structure is overlayed
        BOOLEAN enable;             // on to an I/O port to gain
        BOOLEAN rs;                 // access to the LCD pins.
        BOOLEAN rw;                 // The bits are allocated from
        BOOLEAN unused;             // low order up.  ENABLE will
        int     data : 4;           // be pin B0.
      } lcd;

#if defined(__PCH__)
#if defined use_portb_lcd
  #byte lcd = 0xF81                 // This puts the entire structure
#else
  #byte lcd = 0xF83                 // This puts the entire structure
#endif
#else
#if defined use_portb_lcd
  #byte lcd = 6                     // on to port B (at address 6)
#else
  #byte lcd = 8                     // on to port D (at address 8)
#endif
#endif

#if defined use_portb_lcd
```

```
  #define set_tris_lcd(x) set_tris_b(x)
#else
  #define set_tris_lcd(x) set_tris_d(x)
#endif

#define lcd_type 2          // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40   // LCD RAM address for the second line

BYTE const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
                    // These bytes need to be sent to the LCD
                    // to start it up.
                    // The following are used for setting
                    // the I/O port direction register.

struct lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are out
struct lcd_pin_map const LCD_READ = {0,0,0,0,15}; // For read mode data pins are in

BYTE lcd_read_byte() {
    BYTE low,high;
    set_tris_lcd(LCD_READ);
    lcd.rw = 1;
    delay_cycles(1);
    lcd.enable = 1;
    delay_cycles(1);
    high = lcd.data;
    lcd.enable = 0;
    delay_cycles(1);
    lcd.enable = 1;
    delay_us(1);
    low = lcd.data;
    lcd.enable = 0;
    set_tris_lcd(LCD_WRITE);
```

```
    return( (high<<4) | low);
}

void lcd_send_nibble( BYTE n ) {
    lcd.data = n;
    delay_cycles(1);
    lcd.enable = 1;
    delay_us(2);
    lcd.enable = 0;
}

void lcd_send_byte( BYTE address, BYTE n ) {
    lcd.rs = 0;
    while ( bit_test(lcd_read_byte(),7) ) ;
    lcd.rs = address;
    delay_cycles(1);
    lcd.rw = 0;
    delay_cycles(1);
    lcd.enable = 0;
    lcd_send_nibble(n >> 4);
    lcd_send_nibble(n & 0xf);
}

void lcd_init() {
   BYTE i;
   set_tris_lcd(LCD_WRITE);
   lcd.rs = 0;
   lcd.rw = 0;
   lcd.enable = 0;
   delay_ms(15);
   for(i=1;i<=3;++i) {
     lcd_send_nibble(3);
```

```
    delay_ms(5);
   }
   lcd_send_nibble(2);
   for(i=0;i<=3;++i)
    lcd_send_byte(0,LCD_INIT_STRING[i]);
}

void lcd_gotoxy( BYTE x, BYTE y) {
  BYTE address;
  if(y!=1)
   address=lcd_line_two;
  else
   address=0;
  address+=x-1;
  lcd_send_byte(0,0x80|address);
}

void lcd_putc( char c) {
  switch (c) {
    case '\f'   : lcd_send_byte(0,1);
                  delay_ms(2);
                                break;
    case '\n'   : lcd_gotoxy(1,2);      break;
    case '\b'   : lcd_send_byte(0,0x10);  break;
    default     : lcd_send_byte(1,c);     break;
  }
}

char lcd_getc( BYTE x, BYTE y) {
  char value;
   lcd_gotoxy(x,y);
   while ( bit_test(lcd_read_byte(),7) ); // wait until busy flag is low
```

```
  lcd.rs=1;
  value = lcd_read_byte();
  lcd.rs=0;
  return(value);
}

void LCD_Command(int cm){
 lcd_send_byte(0,cm);
}

void LCD_ShiftLeft(){
 lcd_send_byte(0,0x18);
}

void LCD_ShiftRight(){
 lcd_send_byte(0,0x1c);
}

void LCD_MoveLeft(char p){
 char i;
 for(i=0;i<p;i++){
  LCD_ShiftLeft();
  delay_ms(500);
 }
}

void LCD_MoveRight(char p){
 char i;
 for(i=0;i<p;i++){
  LCD_ShiftRight();
  delay_ms(500);
 }
```

```
}

void strcpy(char *s1, char *s2){
  while(*s1++ = *s2++);
}

void LCD_String(char *s, int dly){
  while(*s!=0){
    lcd_putc(*s++);
    delay_ms(dly);
  }
}
```

ภาคผนวก ข

Data Sheets

PIC18FXX39
Data Sheet

Enhanced FLASH Microcontrollers with Single Phase Induction Motor Control Kernel

MICROCHIP

PIC18FXX39

Enhanced FLASH Microcontrollers with Single Phase Induction Motor Control Kernel

High Performance RISC CPU:

- Linear program memory addressing to 24 Kbytes
- Linear data memory addressing to 1.4 Kbytes
- 20 MHz operation (5 MIPs):
  - 20 MHz oscillator/clock input
  - 5 MHz oscillator/clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- 8 x 8 Single Cycle Hardware Multiplier

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced FLASH program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory
- FLASH/Data EEPROM Retention: > 100 years
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Programmable code protection
- Power saving SLEEP mode
- Single supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins

Analog Features:

- Compatible 10-bit Analog-to-Digital Converter module (A/D) with:
  - Fast sampling rate
  - Conversion available during SLEEP
  - DNL = ±1 LSb, INL = ±1 LSb
- Programmable Low Voltage Detection (PLVD)
  - Supports interrupt on Low Voltage Detection
- Programmable Brown-out Reset (BOR)

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option - Timer1/Timer3
- Two PWM modules:
  - Resolution is 1- to 10-bit,
    Max. PWM freq. @ 8-bit resolution = 156 kHz
    10-bit resolution = 39 kHz
- Single Phase Induction Motor Control kernel
  - Programmable Motor Control Technology (ProMPT™) provides open loop Variable Frequency (VF) control
  - User programmable Voltage vs. Frequency curve
  - Most suitable for shaded pole and permanent split capacitor type motors
- Master Synchronous Serial Port (MSSP) module with two modes of operation:
  - 3-wire SPI™ (supports all 4 SPI modes)
  - I^2C™ Master and Slave mode
- Addressable USART module:
  - Supports RS-485 and RS-232
- Parallel Slave Port (PSP) module

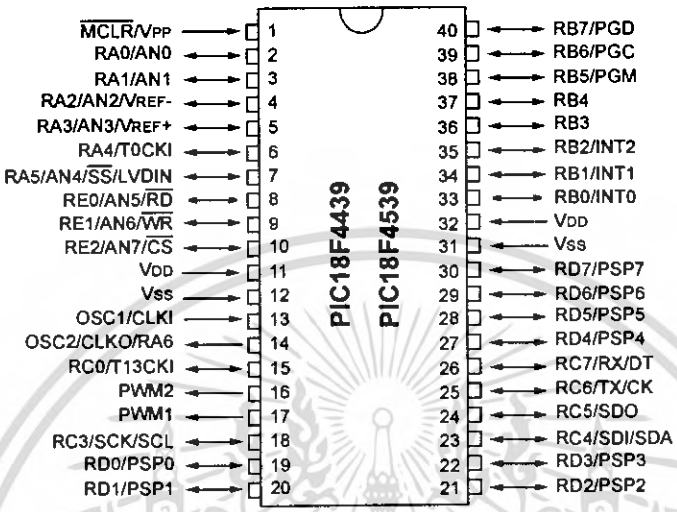
CMOS Technology:

- Low power, high speed FLASH/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges

Device	Program Memory		Data Memory		I/O Pins	10-bit A/D (ch)	PWM 10-bit	MSSP		AUSART	Timers 16-bit/WDT
	Bytes	Words	SRAM (Bytes)	EEPROM (Bytes)				SPI	Master I^2C		
PIC18F2439	12K	6144	640	256	21	5	2	Yes	Yes	Yes	3/1
PIC18F2539	24K	12288	1408	256	21	5	2	Yes	Yes	Yes	3/1
PIC18F4439	12K	6144	640	256	32	8	2	Yes	Yes	Yes	3/1
PIC18F4539	24K	12288	1408	256	32	8	2	Yes	Yes	Yes	3/1

Pin Diagrams (Cont.'d)

40-Pin DIP



28-Pin DIP, SOIC

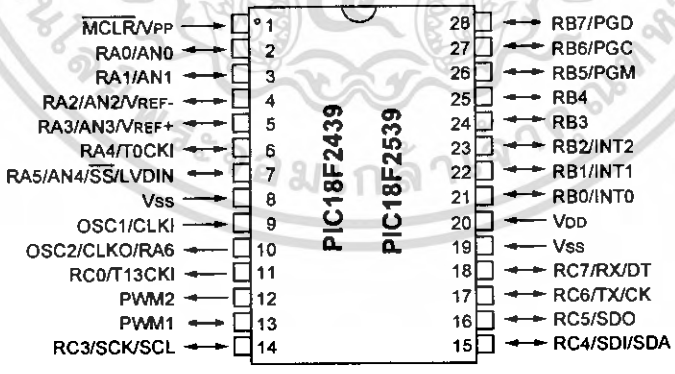


TABLE 1-1: PIC18FXX39 DEVICE FEATURES

Features	PIC18F2439	PIC18F2539	PIC18F4439	PIC18F4539
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	12K	24K	12K	24K
Program Memory (Instructions)	6144	12288	6144	12288
Data Memory (Bytes)	640	1408	640	1408
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	15	15	16	16
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	3	3	3	3
PWM Modules(1)	2	2	2	2
Single Phase Induction Motor Control	Yes	Yes	Yes	Yes
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin TQFP 44-pin QFN	40-pin DIP 44-pin TQFP 44-pin QFN

Note 1: PWM modules are used exclusively in conjunction with the motor control kernel, and are not available for other applications.

16.3.3 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, re-initialize 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. That is:

- 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 TRISC<4> bit set

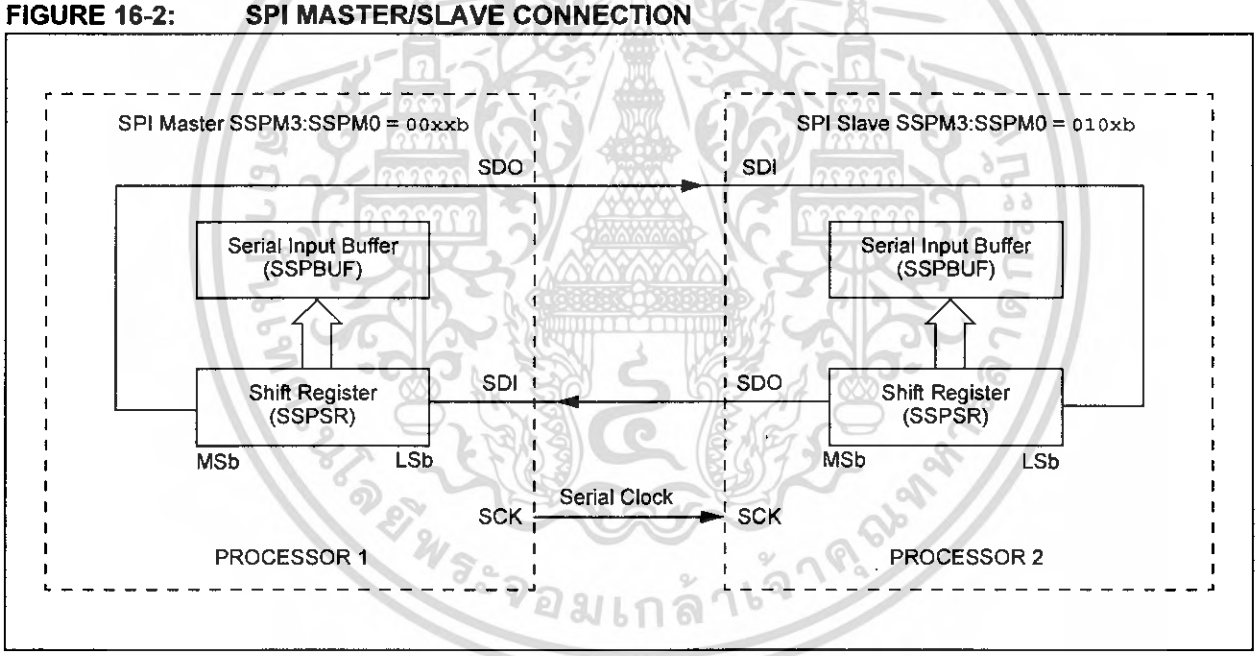
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

16.3.4 TYPICAL CONNECTION

Figure 16-2 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 16-2: SPI MASTER/SLAVE CONNECTION



16.3.5 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 16-2) 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 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.

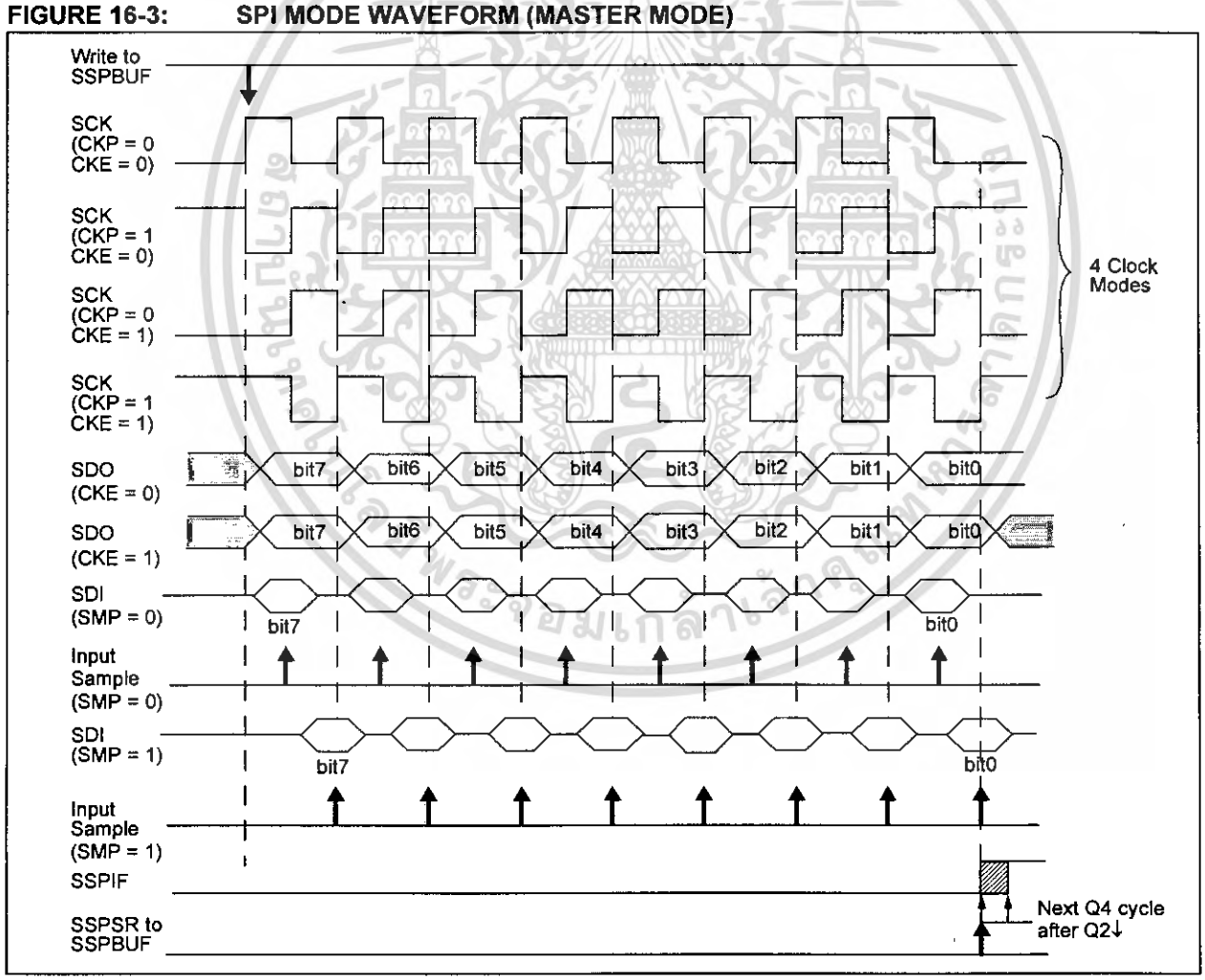
The clock polarity is selected by appropriately programming the CKP bit (SSPCON1<4>). This then, would give waveforms for SPI communication as shown in Figure 16-3, Figure 16-5, and Figure 16-6, where the MSB is transmitted first. In Master mode, the SPI clock rate (bit rate) is user-programmable to be one of the following:

- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)

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

Figure 16-3 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 16-3: SPI MODE WAVEFORM (MASTER MODE)



23.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings (†)

Ambient temperature under bias ..... -55°C to +125°C

Storage temperature ..... -65°C to +150°C

Voltage on any pin with respect to Vss (except VDD, $\overline{MCLR}$, and RA4) ..... -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

Voltage on RA4 with respect to Vss ..... 0V to +8.5V

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 PORTA, PORTB, and PORTE **(Note 3)** (combined) ..... 200 mA

Maximum current sourced by PORTA, PORTB, and PORTE **(Note 3)** (combined) ..... 200 mA

Maximum current sunk by PORTC and PORTD **(Note 3)** (combined) ..... 200 mA

Maximum current sourced by PORTC and PORTD **(Note 3)** (combined) ..... 200 mA

Note 1: Power dissipation is calculated as follows:
Pdis = VDD x {IDD - ∑ IOH} + ∑ {(VDD-VOH) x IOH} + ∑(Vol x IOL)

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

3: PORTD and PORTE not available on the PIC18F2X39 devices.

† 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 23-1: PIC18FXX39 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)

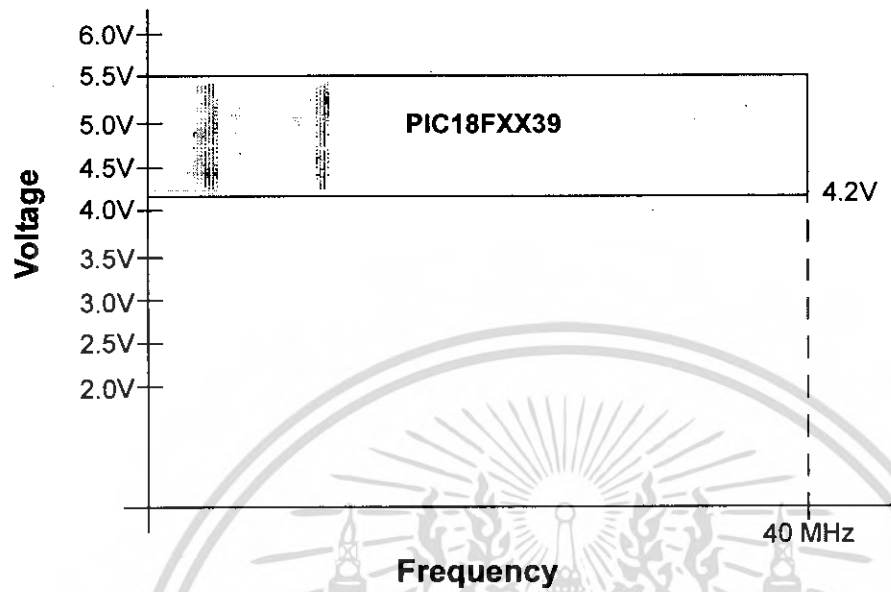
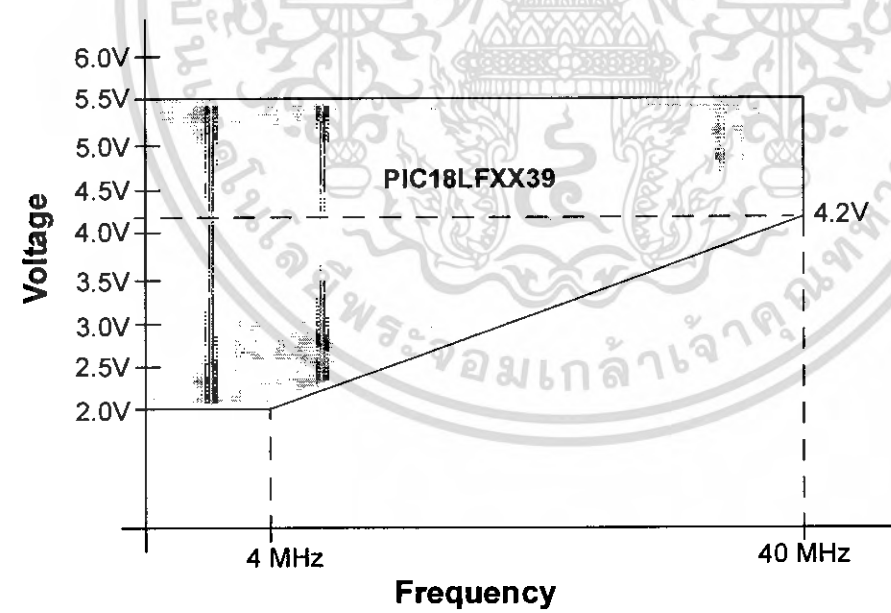


FIGURE 23-2: PIC18LFXX39 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)



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

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

VS1011b - MP3 AUDIO CODEC

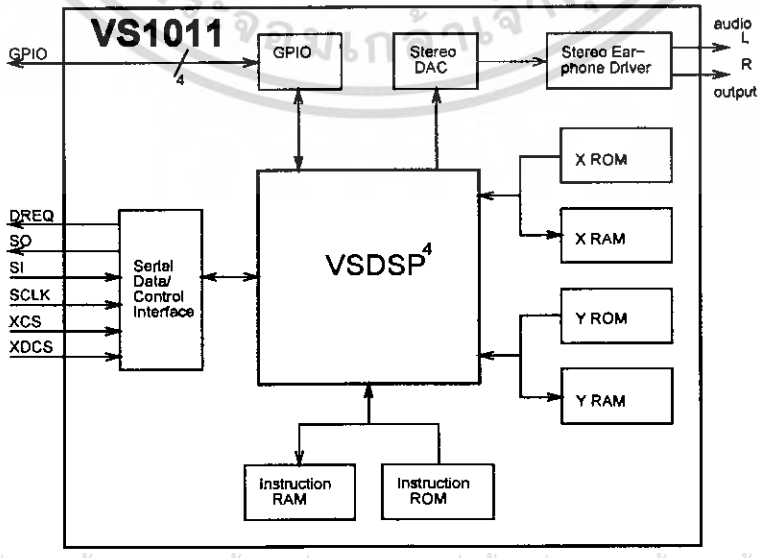
Features

- Decodes MPEG 1 & 2 audio layer 3 (ISO 11172-3), WAV and PCM files
- Supports VBR (variable bitrate) for MP3
- Stream support
- Can be used as a slave co-processor
- Operates with single clock 12.288..14 MHz or 24.576..28 MHz.
- Low-power operation
- High-quality stereo DAC with no phase error between channels
- Stereo earphone driver capable of driving a 30Ω load
- Separate 2.5 .. 3.6V operating voltages for analog and digital
- 5.5 KiB On-chip RAM for user code / data
- Serial control and data interfaces
- New functions may be added with software and 4 GPIO pins

Description

VS1011b is a single-chip MP3 audio decoder. The chip contains a high-performance, low-power DSP processor core VS_DSP^4, working memory, 5 KiB instruction RAM and 0.5 KiB data RAM for user applications, serial control and input data interfaces, 4 general purpose I/O pins, as well as a high-quality variable-sample-rate stereo DAC, followed by an earphone amplifier and a ground buffer.

VS1011b receives its input bitstream through a serial input bus, which it listens to as a system slave. The input stream is decoded and passed through a digital volume control to an 18-bit oversampling, multi-bit, sigma-delta DAC. The decoding is controlled via a serial control bus. In addition to the basic decoding, it is possible to add application specific features, like DSP effects, to the user RAM memory.



4 Characteristics & Specifications

Unless otherwise noted: AVDD=2.7..3.6V, DVDD=2.3..3.6V, TA=-30..+85°C, XTALI=26.000MHz, Full-Scale Output Sinewave at 1.526 kHz, measurement bandwidth 20..20000 Hz, analog output load 30Ω with ground buffer, bitstream 128 kbit/s, local components as shown in Figure 4.

Note, that some analog values are in practice better than in these tables if chips are used within a limited temperature range and not too close to lower voltage limits.

4.1 Analog Characteristics

Parameter	Symbol	Min	Typ	Max	Unit
DAC Resolution			16		bits
Total Harmonic Distortion	THD		0.1	0.2	%
Dynamic Range (DAC unmuted, A-weighted)	IDR		88		dB
S/N Ratio (full scale signal)	SNR	70	81		dB
Interchannel Isolation (Crosstalk)[1]			41		dB
Interchannel Isolation (Crosstalk)[2]			90		dB
Interchannel Isolation (Crosstalk)[3]			70		dB
Interchannel Gain Mismatch		-0.5		0.5	dB
Frequency Response, 20..15000 Hz		-0.2		0.2	dB
Full Scale Output Voltage (Peak-to-peak)		1.4	1.6[4]	2.1	Vpp
Deviation from Linear Phase				5	°
Out of Band Energy			-90		dB
Analog Output Load Resistance	AOLR		30[5]		Ω
Analog Output Load Capacitance[6]				10	pF

[1] Ground buffer, 30Ω load
[2] Ground buffer, no load
[3] AC coupled towards ground, 30Ω load
[4] Double voltage can be achieved with +-to-+ wiring for mono difference sound.
[5] AOLR may be much lower, but below *Typical* distortion performance may be compromised.
[6] Use small series resistor if load is capacitive.

4.2 Power Consumption

Following table measured with XTALI=12.288MHz, clock doubler on.

Parameter	Symbol	Min	Typ	Max	Unit
Power Supply Rejection			40		dB
Power Supply Consumption AVDD, Reset			1.4	30.0	µA
Power Supply Consumption AVDD, no load, no signal			6.2	8.0	mA
Power Supply Consumption AVDD, o. @ 30Ω.			6.5	40.0	mA
Power Supply Consumption DVDD, Reset			6.5	30.0	µA
Power Supply Consumption DVDD			16.0		mA

4.3 DAC Interpolation Filter Characteristics

Parameter	Symbol	Min	Typ	Max	Unit
Passband (to -3dB corner)		0		0.453	Fs
Passband (Ripple Spec)		0		0.340	Fs
Passband Ripple				±0.2	dB
Stop Band		0.560Fs			Hz
Stop Band Rejection		85			dB
Group Delay			15/Fs		s

Fs is conversion frequency

4.4 Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Analog Positive Supply	AVDD	-0.3	3.6	V
Digital Positive Supply	DVDD	-0.3	3.6	V
Current at Any Digital Output			±50	mA
Voltage at Any Digital Input		DGND-1.0	DVDD+1.0	V
Operating Temperature		-30	+85	°C
Functional Operating Temperature		-40	+95	°C
Storage Temperature		-65	+150	°C

4.5 Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Unit
Analog and Digital Ground	AGND DGND		0.0		V
Positive Analog	AVDD	2.5	2.7	3.6	V
Positive Digital	DVDD	2.3	2.5	3.6	V
Ambient Operating Temperature		-30		+85	°C

The following values are to be used when the clock doubler is active:

Parameter	Symbol	Min	Typ	Max	Unit
Input Clock Frequency	XTALI		12.288	15	MHz
Internal Clock Frequency[1]	CLKI		24.576	30	MHz

[1] The maximum sample rate that may be played with correct speed is CLKI/512.

The following values are to be used when the clock doubler is not active:

Parameter	Symbol	Min	Typ	Max	Unit
Input Clock Frequency	XTALI		24.576	30	MHz
Internal Clock Frequency[1]	CLKI		24.576	30	MHz

[1] The maximum sample rate that may be decoded with correct speed is CLKI/512.

Note: See Application notes for what clock speeds are required to play specific bit rates and sample rates.

4.6 Digital Characteristics

Parameter	Symbol	Min	Typ	Max	Unit
High-Level Input Voltage		0.7DVDD			V
Low-Level Input Voltage				0.3DVDD	V
High-Level Output Voltage at I_O = -1.0 mA		0.7DVDD			V
Low-Level Output Voltage at I_O = 1.0 mA				0.3DVDD	V
Input Leakage Current		-1.0		1.0	μA

4.7 Switching Characteristics - Clocks

Parameter	Symbol	Min	Typ	Max	Unit
Master Clock Frequency [1]	XTALI		12.288		MHz
Master Clock Frequency [2]	XTALI		24.576		MHz
Master Clock Duty Cycle		40	50	60	%
Clock Output [3]	XTALO		XTALI		MHz

[1] Clock doubler active.
[2] Clock doubler not active.
[3] Do not load XTALO by connecting other devices to it.

4.8 Switching Characteristics - DREQ Signal

Parameter	Symbol	Min	Typ	Max	Unit
Data Request Signal	DREQ			200	ns

4.9 Switching Characteristics - SPI Interface Output

Parameter	Symbol	Min	Typ	Max	Unit
SPI Input Clock Frequency				$\frac{CLKI}{6}$	MHz
Rise time for SO				25	ns

Note: Maximum load for SO is 100 pF.

4.10 Switching Characteristics - Boot Initialization

Parameter	Symbol	Min	Max	Unit
_RESET active time		2		XTALI
Power-up to software ready			2 ms + 30000 XTALI	
_RESET inactive to software ready			30000	XTALI

5.1.3 SOIC-28

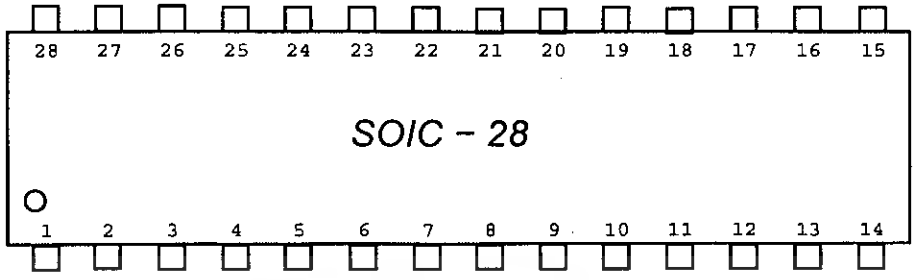


Figure 3: Pin Configuration, SOIC-28.

SOIC-28 package dimensions are at *http://www.vlsi.fi/vs1001/soic28.pdf* .

5.2.2 SOIC-28 Pin Descriptions

Pin Name	Pin	Pin Type	Function
DREQ	1	DO	data request, input bus
GPIO2[2] / DCLK[1]	2	DIO	serial input data bus clock
GPIO3[2] / SDATA[1]	3	DI	serial data input
XDCS / BSYNC[1]	4	DI	byte synchronization signal
DVDD1	5	PWR	digital power supply
DGND1	*6*	PWR	*digital ground*
XTALO	7	CLK	crystal output
XTALI	8	CLK	crystal input
DVDD2	9	PWR	digital power supply
DGND2	10	PWR	digital ground
XCS	11	DI	chip select input (active low)
SCLK	12	DI	clock for serial bus
SI	13	DI	serial input
SO	14	DO3	serial output
TEST	15	DI	reserved for test, connect to DVDD
GPIO0[2]	16	DIO	reserved for test, *do not connect!*
GPIO1[2]	17	DIO	reserved for test, *do not connect!*
AGND0	18	PWR	analog ground
AVDD0	19	PWR	analog power supply
RIGHT	20	AO	right channel output
AGND2	21	PWR	analog ground
RCAP	22	AIO	filtering capacitance for reference
AVDD2	23	PWR	*analog power supply*
LEFT	24	AO	left channel output
AGND3	25	PWR	analog ground
XRESET	26	DI	active low asynchronous reset
DGND0	27	PWR	digital ground
DVDD0	28	PWR	digital power supply

[1] First pin function is active in New Mode, latter in Compatibility Mode.

[2] If not used, use 100 kΩ pull-down resistor.

Pin types:

Type	Description
DI	Digital input, CMOS Input Pad
DO	Digital output, CMOS Input Pad
DIO	Digital input/output
DO3	Digital output, CMOS Tri-stated Output Pad

Type	Description
AI	Analog input
AO	Analog output
AIO	Analog input/output
PWR	Power supply pin

7 SPI Buses

7.1 General

The SPI Bus - that was originally used in some Motorola devices - has been used for both VS1011b's Serial Data Interface SDI (Chapters 7.3 and 8.4) and Serial Control Interface SCI (Chapters 7.4 and 8.5).

7.2 SPI Bus Pin Descriptions

7.2.1 VS1002 Native Modes (New Mode)

These modes are active on VS1011b when SM_SDINEW is set to 1. DCLK, SDATA and BSYNC are replaced with GPIO2, GPIO3 and XDCS, respectively.

SDI Pin	SCI Pin	Description
XDCS	XCS	Active low chip select input. A high level forces the serial interface into standby mode, ending the current operation. A high level also forces serial output (SO) to high impedance state. If SM_SDISHARE is 1, pin XDCS is not used, but the signal is generated internally by inverting XCS.
SCK		Serial clock input. The serial clock is also used internally as the master clock for the register interface. SCK can be gated or continuous. In either case, the first rising clock edge after XCS has gone low marks the first bit to be written.
SI		Serial input. If a chip select is active, SI is sampled on the rising CLK edge.
-	SO	Serial output. In reads, data is shifted out on the falling SCK edge. In writes SO is at a high impedance state.

7.2.2 VS1001 Compatibility Mode

This mode is active when SM_SDINEW is 0 (default). In this mode, DCLK, SDATA and BSYNC are active.

SDI Pin	SCI Pin	Description
-	XCS	Active low chip select input. A high level forces the serial interface into standby mode, ending the current operation. A high level also forces serial output (SO) to high impedance state. There is no chip select for SDI, which is always active.
DCLK	SCK	Serial clock input. The serial clock is also used internally as the master clock for the register interface. SCK can be gated or continuous. In either case, the first rising clock edge after XCS has gone low marks the first bit to be written.
SDATA	SI	Serial input. SI is sampled on the rising SCK edge, if XCS is low.
-	SO	Serial output. In reads, data is shifted out on the falling SCK edge. In writes SO is at a high impedance state.

7.3 Serial Protocol for Serial Data Interface (SDI)

7.3.1 General

The serial data interface operates in slave mode so the DCLK signal must be generated by an external circuit.

Data (SDATA signal) can be clocked in at either the rising or falling edge of DCLK (Chapter 8.6).

VS1011b assumes its data input to be byte-sychronized. SDI bytes may be transmitted either MSb or LSb first, depending of contents of SCI_MODE (Chapter 8.6).

7.3.2 SDI in VS1002 Native Modes (New Mode)

In VS1002 native modes (which are available also in VS1011b), byte synchronization is achieved by XDCS (or XCS if SM_SDISHARE is 1). The state of XDCS (or XCS) may not change while a data byte transfer is in progress. To always maintain data synchronization even if there may be glitches in the boards using VS1011b, it is recommended to turn XDCS (or XCS) every now and then, for instance once after every flash data block or a few kilobytes, just to keep sure the host and VS1011b are in sync.

For new designs, using VS1002 native modes are recommended, as they are easier to implement than BSYNC generation.

7.3.3 SDI in VS1001 Compatibility Mode

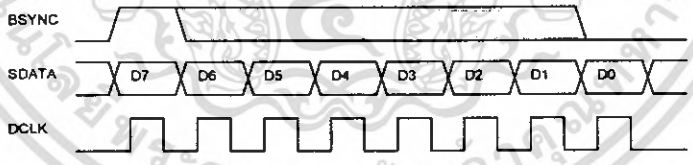


Figure 5: BSYNC Signal - one byte transfer.

When VS1011b is running in VS1001 compatibility mode, a BSYNC signal must be generated to ensure correct bit-alignment of the input bitstream. The first DCLK sampling edge (rising or falling, depending on selected polarity), during which the BSYNC is high, marks the first bit of a byte (LSB, if LSB-first order is used, MSB, if MSB-first order is used). If BSYNC is '1' when the last bit is received, the receiver stays active and next 8 bits are also received.

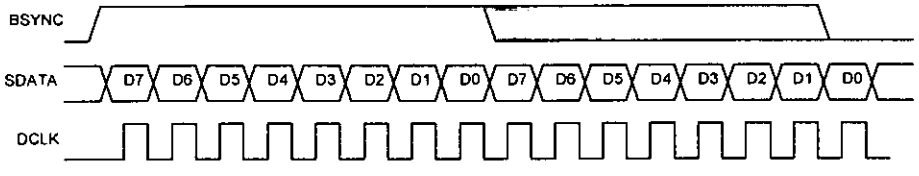


Figure 6: BSYNC Signal - two byte transfer.

Using VS1001 compatibility mode in new designs is strongly discouraged.

7.3.4 SDI and DREQ

The DREQ signal of the data interface is used in slave mode to signal if VS1011b's FIFO is capable of receiving more input data. If DREQ is high, VS1011b can take at least 32 bytes of data. When there is less than 32 bytes of free space, DREQ is turned low, and the sender should stop transferring new data. Because of the 32-byte safety area, the sender may send upto 32 bytes of data at a time without checking the status of DREQ, making controlling VS1011b easier for low-speed microcontrollers.

Note: DREQ may turn low or high at any time, even during a byte transmission. Thus, DREQ should only be used to decide whether to send more bytes. It should not abort a byte transmission that has already started.

7.4 Serial Protocol for Serial Command Interface (SCI)

7.4.1 General

The serial bus protocol for the Serial Command Interface SCI (Chapter 8.5) consists of an instruction byte, address byte and one 16-bit data word. Each read or write operation can read or write a single register. Data bits are read at the rising edge, so the user should update data at the falling edge. Bytes are always send MSb firrst.

The operation is specified by an 8-bit instruction opcode. The supported instructions are read and write. See table below.

Instruction		
Name	Opcode	Operation
READ	0000 0011	Read data
WRITE	0000 0010	Write data

Note: After sending an SCI command, it is not allowed to send SCI or SDI data for 5 microseconds.

7.4.2 SCI Read

VS1011b registers are read by the following sequence, as shown in Figure 7. First, XCS line is pulled SI line followed by an 8-bit word address. After the address has been read in, any further data on SI is ignored. The 16-bit data corresponding to the received address will be shifted out onto the SO line.

XCS should be driven high after data has been shifted out.

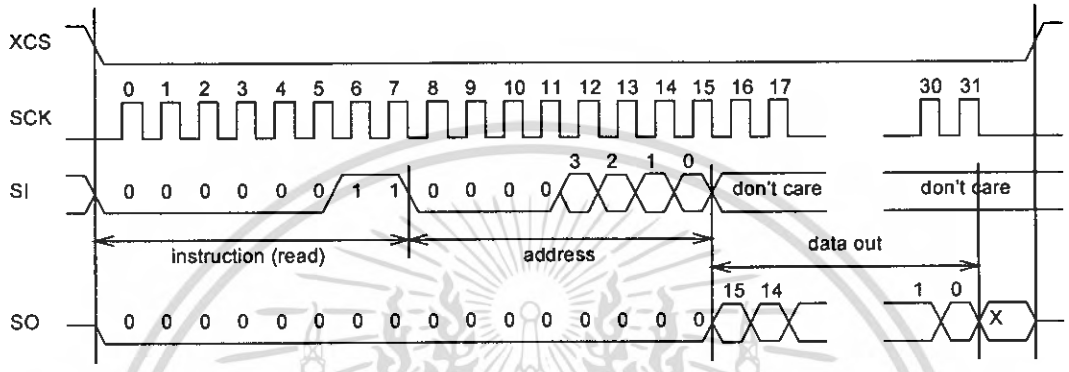


Figure 7: SCI Word Read

7.4.3 SCI Write

VS1011b registers are written to using the following sequence, as shown in Figure 8. First, XCS line is pulled low to select the device. Then the WRITE opcode (0x2) is transmitted via the SI line followed by an 8-bit word address.

After the word has been shifted in and the last clock has been sent, XCS should be pulled high to end the WRITE sequence.

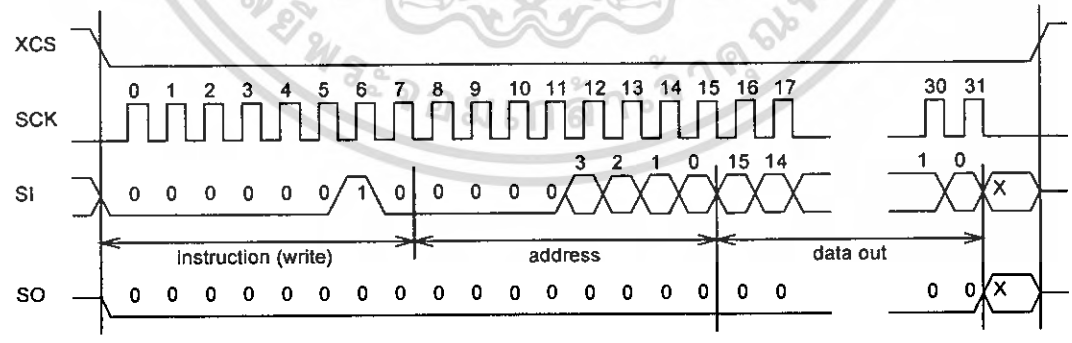


Figure 8: SCI Word Write

7.5 SPI Timing Diagram

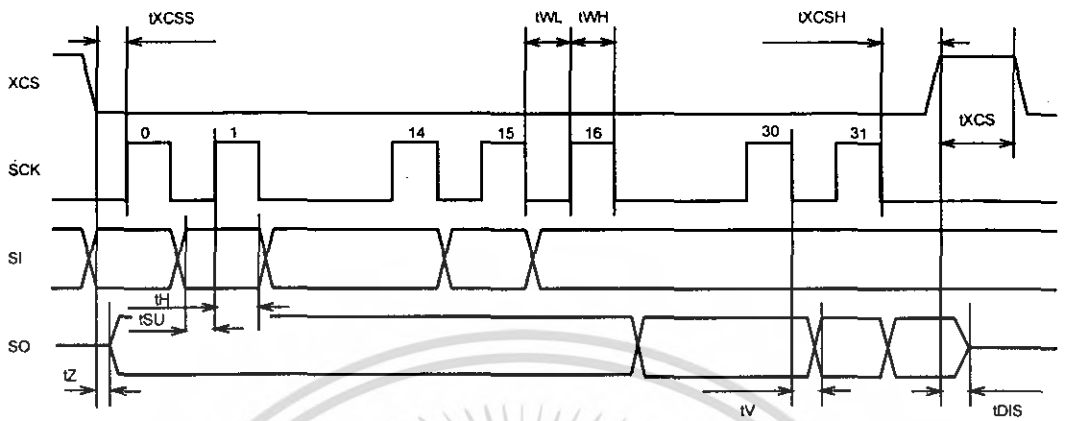


Figure 9: SPI Timing Diagram.

Symbol	Min	Max	Unit
tXCSS	5		ns
tSU	-26		ns
tH	2		XTALI cycles
tZ	0		ns
tWL	2		XTALI cycles
tWH	2		XTALI cycles
tV		2 (+ 25ns[1])	XTALI cycles
tXCSH	-26		ns
tXCS	2		XTALI cycles
tDIS		10	ns

[1] 25ns is when pin loaded with 100pF capacitance. The time is shorter with lower capacitance.

Note: As tWL and tWH, as well as tH require at least 2 clock cycles, the maximum speed for the SPI bus that can easily be used is 1/6 of VS1011b's external clock speed XTALI. Slightly higher speed can be achieved with very careful timing tuning. For details, see Application Notes for VS10XX.

Note: Negative numbers mean that the signal can change in different order from what is shown in the diagram.

8 Functional Description

8.1 Main Features

VS1011b is based on a proprietary digital signal processor, VS_DSP. It contains all the code and data memory needed for MPEG and WAV PCM audio decoding, together with serial interfaces, a multirate stereo audio DAC and analog output amplifiers and filters.

VS1011b can play all MPEG 1 and 2 layer III files, with all sample rates and bitrates, including variable bitrate (VBR).

8.2 Supported Audio Codecs

Conventions	
Mark	**Description**
+	Format is supported
-	Format exists but is not supported
	Format doesn't exist

8.2.1 Supported MP3 (MPEG layer 3) Formats

MPEG 1.0[1]:

Samplerate / Hz	Bitrate / kbit/s													
	32	40	48	56	64	80	96	112	128	160	192	224	256	320
48000	+	+	+	+	+	+	+	+	+	+	+	+	+[3]	+[3]
44100	+	+	+	+	+	+	+	+	+	+	+	+	+	+[3]
32000	+	+	+	+	+	+	+	+	+	+	+	+	+	+

MPEG 2.0[1]:

Samplerate / Hz	Bitrate / kbit/s													
	8	16	24	32	40	48	56	64	80	96	112	128	144	160
24000	+	+	+	+	+	+	+	+	+	+	+	+	+	+
22050	+	+	+	+	+	+	+	+	+	+	+	+	+	+
16000	+	+	+	+	+	+	+	+	+	+	+	+	+	+

MPEG 2.5[1] [2]:

Samplerate / Hz	Bitrate / kbit/s													
	8	16	24	32	40	48	56	64	80	96	112	128	144	160
12000	+	+	+	+	+	+	+	+	+	+	+	+	+	+
11025	+	+	+	+	+	+	+	+	+	+	+	+	+	+
8000	+	+	+	+	+	+	+	+	+	+	+	+	+	+

[1] Also all variable bitrate (VBR) formats are supported.

[2] Incompatibilities may occur because MPEG 2.5 is not a standard format.

[3] Nominal CLKI=24.576 MHz may be too little for glitchless playback.

8.2.2 Supported RIFF WAV Formats

The most common RIFF WAV subformats are supported.

Format	Name	Supported	Comments
0x01	PCM	+	16 and 8 bits, any sample rate ≤ 48kHz
0x02	ADPCM	-	
0x03	IEEE_FLOAT	-	
0x06	ALAW	-	
0x07	MULAW	-	
0x10	OKI_ADPCM	-	
0x11	IMA_ADPCM	-	
0x15	DIGISTD	-	
0x16	DIGIFIX	-	
0x30	DOLBY_AC2	-	
0x31	GSM610	-	
0x3b	ROCKWELL_ADPCM	-	
0x3c	ROCKWELL_DIGITALK	-	
0x40	G721_ADPCM	-	
0x41	G728_CELP	-	
0x50	MPEG	-	
0x55	MPEGLAYER3	+	For supported MP3 modes, see Chapter 8.2.1
0x64	G726_ADPCM	-	
0x65	G722_ADPCM	-	

8.3 Data Flow of VS1011b

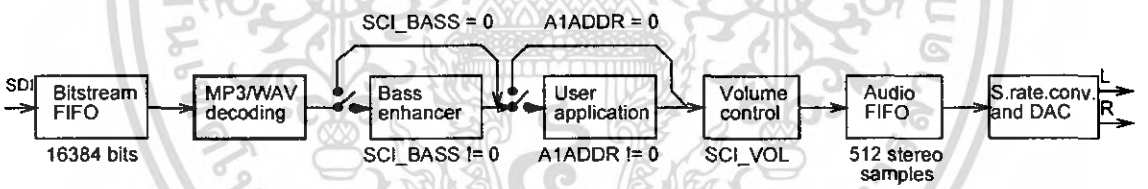


Figure 13: Data Flow of VS1011b.

First, depending on the audio data, MP3 or PCM WAV data is received and decoded from the SDI bus.

After decoding, data may be sent to the Bass Enhancer depending on SCI_BASS.

Then, if SCI_AIADDR is non-zero, application code is executed from the address pointed to by that register. For more details, see Application Notes for VS10XX.

After the optional user application, the signal is fed to the volume control unit, which also copies the data to the Audio FIFO.

The Audio FIFO holds the data, which is read by the Audio interrupt (Chapter 10.9.1) and fed to the sample rate converter and DACs. The size of the audio FIFO is 512 stereo (2×16-bit) samples.

The sample rate converter converts all different sample rates to CLKI/512 and feeds the data to the DAC, which in order creates a stereo in-phase analog signal. This signal is then forwarded to the earphone amplifier.

8.4 Serial Data Interface (SDI)

The serial data interface is meant for transferring compressed MP3 audio data as well as WAV PCM data.

Also several different tests may be activated through SDI as described in Chapter 9.

8.5 Serial Control Interface (SCI)

The serial control interface is compatible with the SPI bus specification. Data transfers are always 16 bits. VS1011b is controlled by writing and reading the registers of the interface.

The main controls of the control interface are:

- control of the operation mode
- uploading user programs
- access to header data
- status information
- feeding input data

8.6 SCI Registers

SCI registers, prefix SCI_, offset 0xC000				
Reg	**Type**	**Reset**	**Abbrev[bits]**	**Description**
0x0	rw	0	MODE	Mode control.
0x1	rw	0x1C[1]	STATUS	Status of VS1011b.
0x2	rw	0	BASS	Built-in bass enhancer.
0x3	rw	0	CLOCKF	Clock freq + doubler.
0x4	r	0	DECODE_TIME	Decode time in seconds.
0x5	rw	0	AUDATA	Misc. audio data.
0x6	rw	0	WRAM	RAM write.
0x7	rw	0	WRAMADDR	Base address for RAM write.
0x8	r	0	HDAT0	Stream header data 0.
0x9	r	0	HDAT1	Stream header data 1.
0xA	rw	0	AIADDR	Start address of application.
0xB	rw	0	VOL	Volume control.
0xC	rw	0	AICTRL0	Application control register 0.
0xD	rw	0	AICTRL1	Application control register 1.
0xE	rw	0	AICTRL2	Application control register 2.
0xF	rw	0	AICTRL3	Application control register 3.

[1] Firmware changes the value of this register immediately to 0x18, and in less than 100 ms to 0x10.

8.6.1 SCI_MODE (RW)

SCI_MODE is used to control operation of VS1011b.

Bit	Name	Function	Value	Description
0	SM_DIFF	Differential	0 1	normal in-phase audio left channel inverted
1	SM_SETTOZERO1	Set to zero	0 1	right wrong
2	SM_RESET	Soft reset	0 1	no reset reset
3	SM_OUTOFWAV	Jump out of WAV decoding	0 1	no yes
4	SM_SETTOZERO2	set to zero	0 1	right wrong
5	SM_TESTS	Allow SDI tests	0 1	not allowed allowed
6	SM_STREAM	Stream mode	0 1	no yes
7	SM_SETTOZERO3	set to zero	0 1	right wrong
8	SM_DACT	DCLK active edge	0 1	rising falling
9	SM_SDIORD	SDI bit order	0 1	MSb first MSb last
10	SM_SDISHARE	Share SPI chip select	0 1	no yes
11	SM_SDINEW	VS1002 native SPI modes	0 1	no yes
12	SM_SETTOZERO4	set to zero	0 1	right wrong
13	SM_SETTOZERO5	set to zero	0 1	right wrong

When SM_DIFF is set, the player inverts the left channel output. For a stereo input this creates a virtual surround, and for a mono input this effectively creates a differential left/right signal.

By setting SM_RESET to 1, the player is software reset. This bit clears automatically.

When the user decoding a WAV file wants to get out of the file without playing it to the end, set SM_OUTOFWAV, and send zeros to VS1002c until SM_OUTOFWAV is again zero. If the user doesn't want to check SM_OUTOFWAV, send 128 zeros.

If SM_TESTS is set, SDI tests are allowed. For more details on SDI tests, look at Chapter 9.6.

SM_STREAM activates VS1011b's stream mode. In this mode, data should be sent with as even intervals as possible (and preferable with data blocks of less than 512 bytes), and VS1011b makes every attempt to keep its input buffer half full by changing its playback speed upto 5%. For best quality sound, the

9 Operation

9.1 Clocking

VS1011b operates on a single, nominally 24.576 MHz fundamental frequency master clock. This clock can be generated by external circuitry (connected to pin XTALI) or by the internal clock chrystal interface (pins XTALI and XTALO). This clock is sufficient to support a high quality audio output for almost all standard sample rates and bit-rates (see Application Notes for VS10XX).

9.2 Hardware Reset

When the XRESET -signal is driven low, VS1011b is reset and all the control registers and internal states are set to the initial values. XRESET-signal is asynchronous to any external clock. The reset mode doubles as a full-powerdown mode, where both digital and analog parts of VS1011b are in minimum power consumption stage, and where clocks are stopped. Also XTALO and XTALI are grounded.

After a hardware reset (or at power-up), the user should set such basic software registers as SCI_VOL for volume (and SCI_CLOCKF if the input clock is anything else than 24.576 MHz) before starting decoding.

9.3 Software Reset

In some cases the decoder software has to be reset. This is done by activating bit 2 in SCI_MODE register (Chapter 8.6.1). Then wait for at least 2 μs, then look at DREQ. DREQ will stay down for at least 6000 clock cycles, which means an approximate 250 μs delay if VS1011b is run at 24.576 MHz. After DREQ is up, you may continue playback as usual.

If you want to make sure VS1011b doesn't cut the ending of low-bitrate data streams and you want to do a software reset, it is recommended to feed 2048 zeros to the SDI bus after the file and before the reset.

9.4 Play/Decode

This is the normal operation mode of VS1011b. SDI data is decoded. Decoded samples are converted to analog domain by the internal DAC. If there are bad problems in the decoding process, the error flags of SCI_HDAT0 and SCI_HDAT1 are set to 0 and analog outputs are muted.

When there is no input for decoding, VS1011b goes into idle mode (lower power consumption than during decoding) and actively monitors the serial data input for valid data.

9.5 Feeding PCM data

VS1011b can be used as a PCM decoder by sending to it a WAV file header. If the length sent in the WAV file is 0 or 0xFFFFFFF, VS1011b will stay in PCM mode indefinitely. 8-bit linear and 16-bit linear audio is supported in mono or stereo.

9.6 SDI Tests

There are several test modes in VS1011b, which allow the user to perform memory tests, SCI bus tests, and several different sine wave tests.

All tests are started in a similar way: VS1011b is hardware reset, SM_TESTS is set, and then a test command is sent to the SDI bus. Each test is started by sending a 4-byte special command sequence, followed by 4 zeros. The sequences are described below.

9.6.1 Sine Test

Sine test is initialized with the 8-byte sequence 0x53 0xEF 0x6E n 0 0 0 0, where n defines the sine test to use. n is defined as follows:

n bits		
Name	**Bits**	**Description**
F_sIdx	7:5	Sample rate index
S	4:0	Sine skip speed

F_sIdx	F_s
0	44100 Hz
1	48000 Hz
2	32000 Hz
3	22050 Hz
4	24000 Hz
5	16000 Hz
6	11025 Hz
7	12000 Hz

The frequency of the sine to be output can now be calculated from $F = F_s \times \frac{S}{128}$.

Example: Sine test is activated with value 126, which is 0b01111110. Breaking n to its components, $F_sIdx = 0b011 = 1$ and thus $F_s = 22050Hz$. $S = 0b11110 = 30$, and thus the final sine frequency $F = 22050Hz \times \frac{30}{128} \approx 5168Hz$.

To exit the sine test, send the sequence 0x45 0x78 0x69 0x74 0 0 0 0.

Note: Sine test signals go through the digital volume control, so it is possible to test channels separately.

SanDisk

SanDisk Secure Digital Card

Product Manual

Version 1.9

Document No. 80-13-00169

December 2003

SanDisk Corporation

Corporate Headquarters • 140 Caspian Court • Sunnyvale, CA 94089

Phone (408) 542-0500 • Fax (408) 542-0503

www.sandisk.com

1.1. Scope

This document describes the key features and specifications of the SD Card, as well as the information required to interface this product to a host system.

1.2. Product Models

The SD Card is available in the capacities shown in Table 1-1.

Table 1-1. SD Card Capacities

Model No.	Capacities
SDSDB-16	16 MB
SDSDB-32	32 MB
SDSDJ-64	64 MB
SDSDJ-128	128 MB
SDSDJ-256	256 MB
SDSDJ-512	512 MB
SDSDJ-1024	1024 MB

SDSDB = Binary NAND technology.
SDSDJ = Multi Level Cell (MLC) NAND technology.

1.3. System Features

The SD Card provides the following features:

- Up to 1-GB of data storage.
- SD Card protocol compatible.
- Supports SPI Mode.
- Targeted for portable and stationary applications for secured (copyrights protected) and non-secured data storage.
- Voltage range:
  - Basic communication (CMD0, CMD15, CMD55, ACMD41): 2.0—3.6V.
  - Other commands and memory access: 2.7—3.6V.
- Variable clock rate 0—25 MHZ.
- Up to 12.5 MB/sec data transfer rate (using 4 parallel data lines).
- Maximum data rate with up to 10 cards.
- Correction of memory field errors.
- Copyrights Protection Mechanism—Complies with highest security of SDMI standard.
- Password Protected of Cards (not on all models).
- Write Protect feature using mechanical switch.
- Built-in write protection features (permanent and temporary).
- Card Detection (Insertion/Removal).
- Application specific commands.

1.5.9.6. Read and Write Operations

The SD Card supports two read/write modes as shown in Figure 1-3.

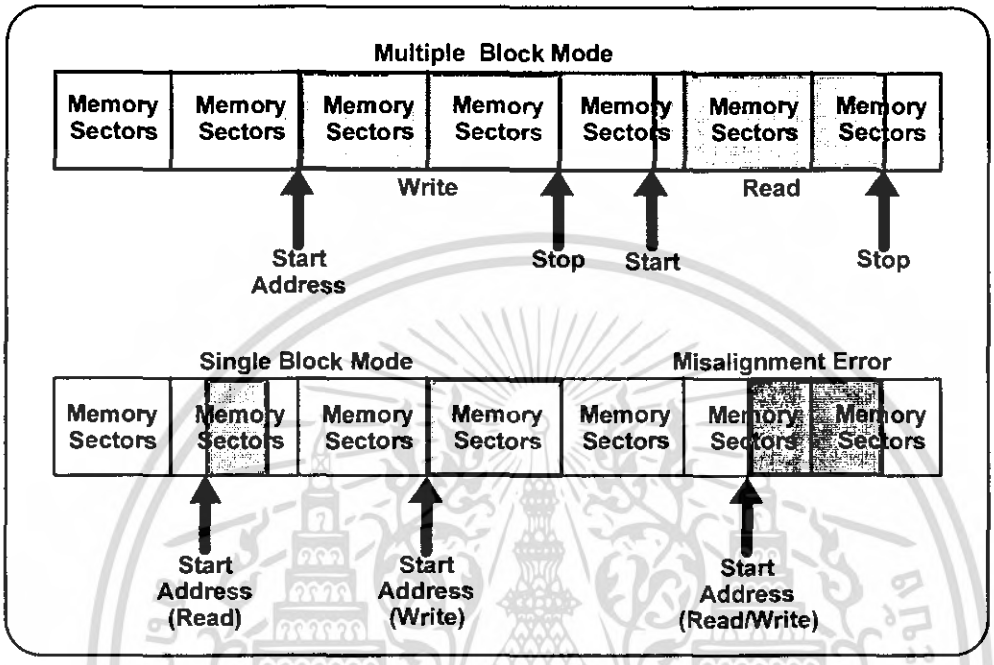


Figure 1-3. Data Transfer Formats

Single Block Mode

In this mode the host reads or writes one data block in a pre-specified length. The data block transmission is protected with 16-bit CRC that is generated by the sending unit and checked by the receiving unit.

The block length for read operations is limited by the device sector size (512 bytes) but can be as small as a single byte. Misalignment is not allowed. Every data block must be contained in a single physical sector. The block length for write operations must be identical to the sector size and the start address aligned to a sector boundary.

Multiple Block Mode

This mode is similar to the single block mode, but the host can read/write multiple data blocks (all have the same length) which will be stored or retrieved from contiguous memory addresses starting at the address specified in the command. The operation is terminated with a stop transmission command.

Misalignment and block length restrictions apply to multiple blocks as well and are identical to the single block read/write operations.

1.5.9.7. Data Transfer Rate

The SD Card can be operated using either a single data line (DAT0) or four data lines (DAT0-DAT3) for data transfer. The maximum data transfer rate for a single data line is 25 Mbit per second and for four data lines it is 100 Mbit (12 MB) per second.

2. Product Specifications

For all the following specifications, values are defined at ambient temperature and nominal supply voltage unless otherwise stated.

2.1. System Environmental Specifications

Table 2-1. System Environmental Specifications

Temperature	Operating: Non-Operating:	-25° C to 85° C -40° C to 85° C
Humidity	Operating: Non-Operating:	25% to 95%, non-condensing 25% to 95%, non-condensing
ESD Protection	Contact Pads: Non Contact Pad Area:	± 4kV, Human body model according to ANSI EOS/ESD-S5.1-1998 ± 8kV (coupling plane discharge) ± 15kV (air discharge) Human body model per IEC61000-4-2

2.2. Reliability and Durability

Table 2-2. Reliability and Durability Specifications

Durability	10,000 mating cycles
Bending	10N
Torque	0.15N.m or ±2.5 deg.
Drop Test	1.5m free fall
UV Light Exposure	UV: 254nm, 15Ws/cm^2 according to ISO 7816-1
Visual Inspection/Shape and Form	No warpage; no mold skin; complete form; no cavities; surface smoothness ≤ -0.1 mm/cm^2 within contour; no cracks; no pollution (oil, dust, etc.)
Minimum Moving Force of WP Switch	40 gf (ensures that the WP switch will not slide while it is inserted in the connector).
WP Switch Cycles	Minimum 1,000 Cycles @ slide force 0.4N to 5N

2.3. Typical Card Power Requirements

Table 2-3. Card Power Requirements

VDD (ripple: max, 60 mV peak to peak)	2.7 V – 3.6 V

(Ta = 25°C @3 V)

	Value	Measurement	Notes
Sleep	250	uA	Max
Read	65	mA	Max
Write	75	mA	Max

2.4. System Performance

Table 2-4. System Performance

	Typical	Maximum
Block Read Access Time Binary Products MLC Products	 1.5msec 10msec	 100msec 100msec
Block Write Access Time Binary Products MLC Products	 24msec 40msec	 250msec 250msec
CMD1 to Ready (after power up)	50msec	500msec
Sleep to Ready	1msec	2msec

NOTES: All values quoted are under the following conditions:
1) Voltage range: 2.7 V to 3.6 V.
2) Temperature range: -25° C to 85° C.
3) Are independent of the SD Card clock frequency.

2.5. System Reliability and Maintenance

Table 2-5. System Reliability and Maintenance Specifications

MTBF	> 1,000,000 hours
Preventive Maintenance	None
Data Reliability	< 1 non-recoverable error in 10^{14} bits read
Endurance	100,000 write/erase cycles (typical)

2.6. Physical Specifications

Refer to Table 2-6 and to Figures 2-1 through 2-3 for SD Card physical specifications and dimensions.

Table 2-6. Physical Specifications

Weight	2.0 g. maximum
Length:	32mm ± 0.1mm
Width:	24mm ± 0.1mm
Thickness:	2.1mm ± 0.15mm (in substrate area only, 2.25mm maximum)

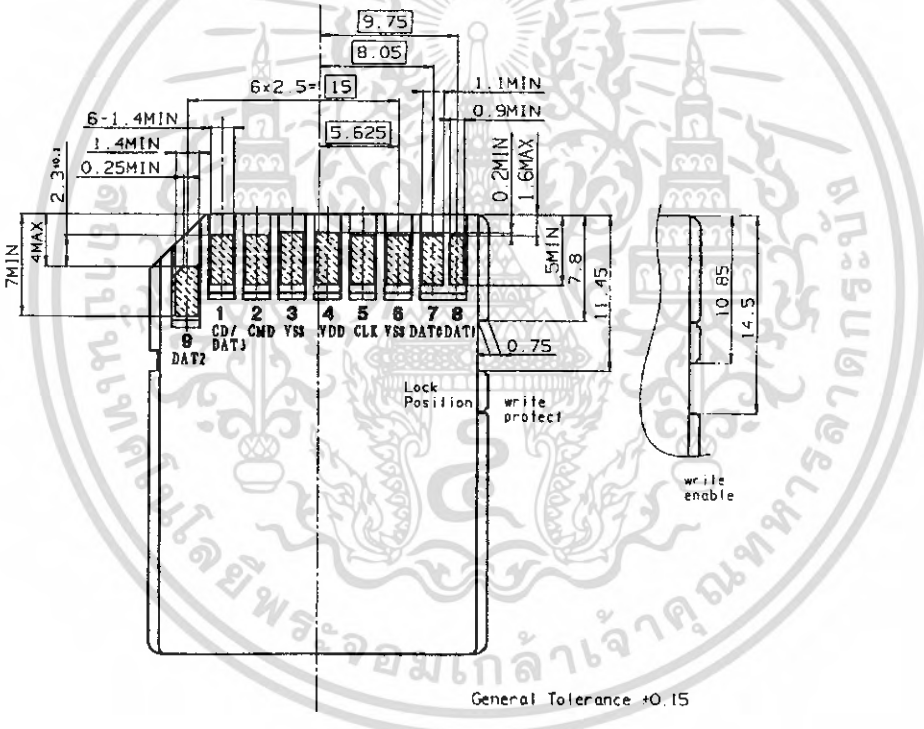


Figure 2-1. SD Card Dimensions

3. SD Card Interface Description

3.1. General Description of Pins and Registers

The SD Card has nine exposed contacts on one side (see Figure 3-1). The host is connected to the SD Card using a dedicated 9-pin connector.

3.1.1. Pin Assignments in SD Card Mode

Table 3-1 lists the pin assignments and definitions in SD Card Mode.

Table 3-1. SD Bus Mode Pad Definition

Pin #	Name	Type[1]	SD Description
1	CD/DAT3[2]	I/O[3]	Card Detect/Data Line [Bit 3]
2	CMD	I/O	Command/Response
3	V_{SS1}	S	Supply voltage ground
4	V_{DD}	S	Supply voltage
5	CLK	I	Clock
6	V_{SS2}	S	Supply voltage ground
7	DAT0	I/O	Data Line [Bit 0]
8	DAT1	I/O	Data Line [Bit 1]
9	DAT2	I/O	Data Line [Bit 2]

NOTES: 1) S=power supply; I=input; O=output using push-pull drivers.
2) The extended DAT lines (DAT1-DAT3) are input on power up. They start to operate as DAT lines after the SET_BUS_WIDTH command. It is the responsibility of the host designer to connect external pullup resistors to all data lines even if only DAT0 is to be used. Otherwise, non-expected high current consumption may occur due to the floating inputs of DAT1 & DAT2 (in case they are not used).
3) After power up, this line is input with 50Kohm(+/-20Kohm) pull-up (can be used for card detection or SPI mode selection). The pull-up may be disconnected by the user, during regular data transfer, with SET_CLR_CARD_DETECT (ACMD42) command.

3.1.2. Pin Assignments in SPI Mode

Table 3-2 lists the pin assignments and definitions in SPI Mode.

Table 3-2. SPI Bus Mode Pad Definition

Pin #	Name	Type[1]	SPI Description
1	CS	I	Chip Select (Active low)
2	DataIn	I	Host to Card Commands and Data
3	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground
7	DataOut	O	Card to Host Data and Status
8	RSV(2)	I	Reserved
9	RSV(2)	I	Reserved

NOTES: 1) S=power supply; I=input; O=output.
2) The 'RSV' pins are floating inputs. It is the responsibility of the host designer to connect external pullup resistors to those lines. Otherwise non-expected high current consumption may occur due to the floating inputs.

Each card has a set of information registers (refer to Table 3-3). Detailed descriptions are provided in Section 3.5.

Table 3-3. SD Card Registers

Name	Width	Description
CID	128	Card identification number: individual card number for identification.
RCA[1]	16	Relative card address: local system address of a card, dynamically suggested by the card and approved by the host during initialization.
CSD	128	Card specific data: information about the card operation conditions.
SCR	64	SD Configuration Register: information about the SD Card's special features capabilities.
OCR	32	Operation Condition Register

NOTE: 1) The RCA register is not available in SPI Mode.

The host may reset the cards by switching the power supply off and on again. The card has its own power-on detection circuitry which puts the card into an idle state after the power-on. The card can also be reset by sending the GO_IDLE (CMD0) command.

Parameter	Symbol	Min.	Max.	Unit	Remark
Inputs CMD, DAT (referenced to CLK)					
Input set-up time	t_{ISU}	5		ns	$C_L \leq 25$ pF (1 cards)
Input hold time	t_{IH}	5		ns	$C_L \leq 25$ pF (1 cards)
Outputs CMD, DAT (referenced to CLK)					
Output delay time during Data Transfer Mode	t_{ODLY}	0	14	ns	$C_L \leq 25$ pF (1 cards)
Output delay time during Identification Mode	t_{ODLY}	0	50	ns	$C_L \leq 25$ pF (1 cards)

NOTE: OHz stops the clock. The given minimum frequency range is for cases where a continuous clock is required.

3.5. SD Card Registers

There is a set of seven registers within the card interface. The OCR, CID, CSD and SCR registers carry the card configuration information. The RCA register holds the card relative communication address for the current session. The card status and SD status registers hold the communication protocol related status of the card.

3.5.1. Operating Conditions Register (OCR)

The 32-bit operation conditions register stores the V_{DD} voltage profile of the card. The SD Card is capable of executing the voltage recognition procedure (CMD1) with any standard SD Card host using operating voltages form 2 to 3.6 Volts.

Accessing the data in the memory array, however, requires 2.7 to 3.6 Volts. The OCR shows the voltage range in which the card data can be accessed. The structure of the OCR register is described in Table 3-8.

Table 3-8. OCR Register Definition

OCR Bit	VDD Voltage Window
0-3	Reserved
4	1.6-1.7
5	1.7-1.8
6	1.8-1.9
7	1.9-2.0
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

OCR Bit	VDD Voltage Window
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)

The level coding of the OCR register is as follows:

- Restricted voltage windows=LOW
- Card busy=LOW (bit 31)

The least significant 31 bits are constant and will be set as described in Figure 4-8. If bit 32 (the busy bit) is set, it informs the host that the card power up procedure is finished.

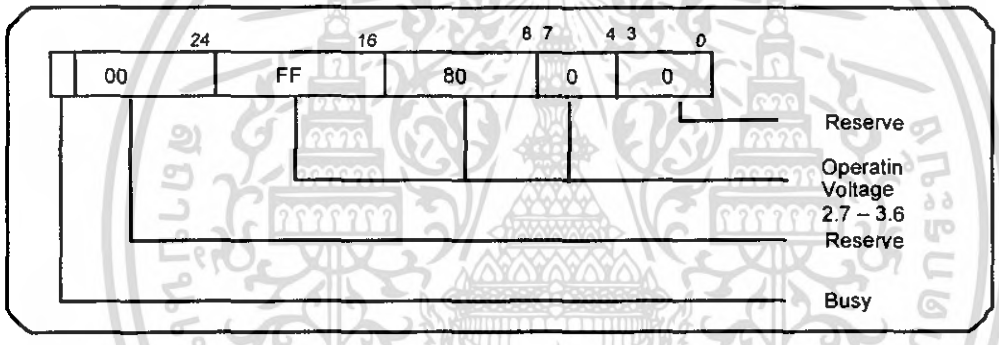


Figure 3-8. OCR Structure

3.5.2. Card Identification (CID) Register

The CID register is 16 bytes long and contains a unique card identification number as shown in Table 3-9. It is programmed during card manufacturing and cannot be changed by SD Card hosts. Note that the CID register in the SD Card has a different structure than the CID register in the MultiMediaCard.

Table 3-9. CID Fields

Name	Type	Width	CID—Slice	Comments	CID Value
Manufacturer ID (MID)	Binary	8	[127:120]	The manufacturer IDs are controlled and assigned by the SD Card Association.	0x03
OEM/Application ID (OID)	ASCII	16	[119:104]	Identifies the card OEM and/or the card contents. The OID is assigned by the 3C.*	SD ASCII Code 0x53, 0x44
Product Name (PNM)	ASCII	40	[103:64]	5 ASCII characters long	SD128, SD064, SD032, SD016, SD008
Product Revision** (PRV)	BCD	8	[63:56]	Two binary coded decimal digits	Product Revision (30)
Serial Number (PSN)	Binary	32	[55:24]	32 Bits unsigned integer	Product Serial Number
Reserved		4	[23:20]		
Manufacture Date Code (MDT)	BCD	12	[19:8]	Manufacture date–yym (offset from 2000)	Manufacture date(for example: Apr 2001 = 0x014)
CRC7 checksum*** (CRC)	Binary	7	[7:1]	Calculated	CRC7
Not used, always '1'		1	[0:0]		

* 3C = The 3 SDA founding companies: Toshiba, SanDisk, and MEI.

** The product revision is composed of two Binary Coded Decimal (BCD) digits, four bits each, representing an "n.m" revision number. The "n" is the most significant nibble and the "m" is the least significant nibble. Example: The PRV binary value filed for product revision "6.2" will be: 0110 0010.

*** The CRC Checksum is computed by the following formula:
CRC Calculation: G(x)=x7+3+1
M(x)=(MID-MSB)*x119+...+(CIN-LSB)*x0
CRC[6...0]=Remainder[(M(x)*x7)/G(x)]

3.5.3. CSD Register

The Card Specific Data (CSD) register contains configuration information required to access the card data. In Table 3-10, the cell type column defines the CSD field as Read only (R), One Time Programmable (R/W) or erasable (R/W/E). This table shows the value in "real world" units for each field and coded according to the CSD structure. The Model dependent column marks (with a check mark, √) the CSD fields that are model dependent. Note that the CSD register in the SD Card has a different structure than the CSD in the MultiMediaCard.

Table 3-10. CSD Register

Name	Field	Width	Cell Type	CSD-Slice	CSD Value	CSD Code
CSD structure	CSD_STRUCTURE	2	R	[127:126]	1.0	00b
Reserved	-	6	R	[125:120]	-	000000b
data read access-time-1	TAAC Binary MLC	 8 8	 R R	 [119:112] [119:112]	 1.5msec 10msec	 00100110b 00001111b
data read access-time-2 in CLK cycles (NSAC*100)	NSAC	8	R	[111:104]	0	00000000b
max. data transfer rate	TRAN_SPEED	8	R	[103:96]	25MHz	00110010b
card command classes	CCC	12	R	[95:84]	All (incl. WP, Lock/unlock)	1F5h

5. SPI Protocol Definition

5.1. SPI Bus Protocol

While the SD Card channel is based on command and data bit-streams, which are initiated by a start bit and terminated by a stop bit, the SPI channel is byte oriented. Every command or data block is built of eight bit bytes and is byte aligned (multiples of eight clocks) to the CS signal.

Similar to the SD Bus protocol, the SPI messages are built from command, response and data-block tokens. All communication between host and cards is controlled by the host (master). The host starts every bus transaction by asserting the CS signal low.

The response behavior in SPI Bus mode differs from the SD Bus mode in the following three ways:

- The selected card always responds to the command.
- An eight or 16-bit response structure is used.
- When the card encounters a data retrieval problem, it will respond with an error response (which replaces the expected data block) rather than time-out as in the SD Bus mode.

In addition to the command response, every data block sent to the card during write operations will be responded with a special data response token. A data block may be as big as one card write block (WRITE_BL_LEN) and as small as a single byte.[1]

5.1.1. Mode Selection

The SD Card wakes up in the SD Bus mode. It will enter SPI mode if the CS signal is asserted (negative) during the reception of the reset command (CMD0). If the card recognizes that the SD Bus mode is required it will not respond to the command and remain in the SD Bus mode. If SPI mode is required, the card will switch to SPI mode and respond with the SPI mode R1 response.

The only way to return to the SD Bus mode is by power cycling the card. In SPI mode, the SD Card protocol state machine is not observed. All the SD Card commands supported in SPI mode are always available.

The default command structure/protocol for SPI mode is that CRC checking is disabled. Since the card powers up in SD Bus mode, CMD0 must be followed by a valid CRC byte (even though the command is sent using the SPI structure). Once in SPI mode, CRCs are disabled by default.

CMD0 is a static command and always generates the same 7-bit CRC of 4Ah. Adding the "1," end bit (bit 0) to the CRC creates a CRC byte of 95h. The following hexadecimal sequence can be used to send CMD0 in all situations for SPI mode, since the CRC byte (although required) is ignored once in SPI mode. The entire CMD0 sequence appears as 40 00 00 00 00 95 (hexadecimal).

1) The default block length is as specified in the CSD (512 bytes). A set block length of less than 512 bytes will cause a write error. The only valid write set block length is 512 bytes. CMD16 is not mandatory if the default is accepted.

5.1.2. Bus Transfer Protection

Every SD Card token transferred on the bus is protected by CRC bits. In SPI mode, the SD Card offers a non protected mode which enables systems built with reliable data links to exclude the hardware or firmware required for implementing the CRC generation and verification functions.

In the non-protected mode the CRC bits of the command, response and data tokens are still required in the tokens however, they are defined as "don't care" for the transmitters and ignored by the receivers.

The SPI interface is initialized in the non-protected mode. The host can turn this option on and off using CRC_ON_OFF command (CMD59).

The CRC7/CRC16 polynomials are identical to that used in SD Bus mode. Refer to this section in the SD Bus mode chapter.

5.1.3. Data Read

SPI mode supports single block and multiple block read operations (SD Card CMD17 or CMD18). Upon reception of a valid read command the card will respond with a response token followed by a data token in the length defined in a previous SET_BLOCK_LENGTH (CMD16) command (see Figure 5-I).

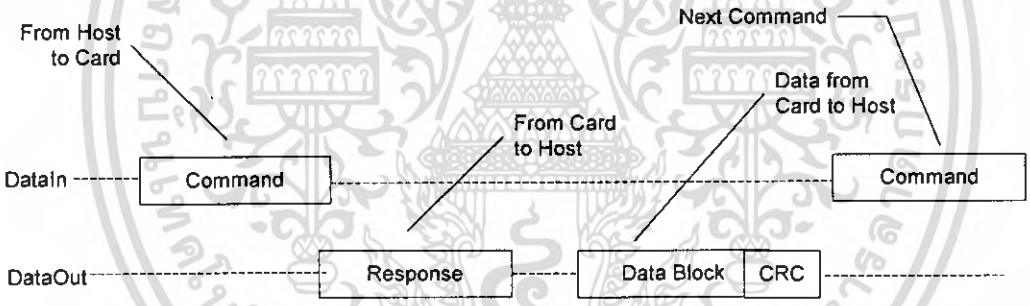


Figure 5-1. Single Block Read Operation

A valid data block is suffixed with a 16-bit CRC generated by the standard CCITT polynomial:

$$x^{16}+x^{12}+x^{5}+1.$$

The maximum block length is 512 bytes as defined by READ_BL_LEN (CSD parameter). Block lengths can be any number between 1 and READ_BL_LEN.

The start address can be any byte address in the valid address range of the card. Every block, however, must be contained in a single physical card sector.

In case of data retrieval error, the card will not transmit any data. Instead, a special data error token will be sent to the host. Figure 5-2 shows a data read operation, which terminated with an error token rather than a data block.

Figure 5-2. Read Operation—Data Error

In the case of a Multiple Block Read operation, every transferred block has a 16-bit CRC suffix. The Stop Transmission command (CMD12) will actually stop the data transfer operation (the same as in SD Bus mode).

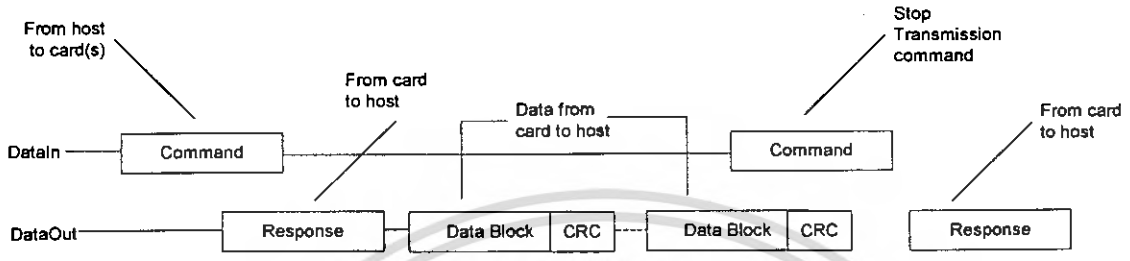


Figure 5-3. Multiple Block Read Operation

5.1.4. Data Write

In SPI mode, the SD Card supports single block or multiple block write operations. Upon reception of a valid write command (SD Card CMD24 or CMD25), the card will respond with a response token and will wait for a data block to be sent from the host. CRC suffix and start address restrictions are identical to the read operation (see Figure 5-4). The only valid block length, however, is 512 bytes. Setting a smaller block length will cause a write error on the next write command.

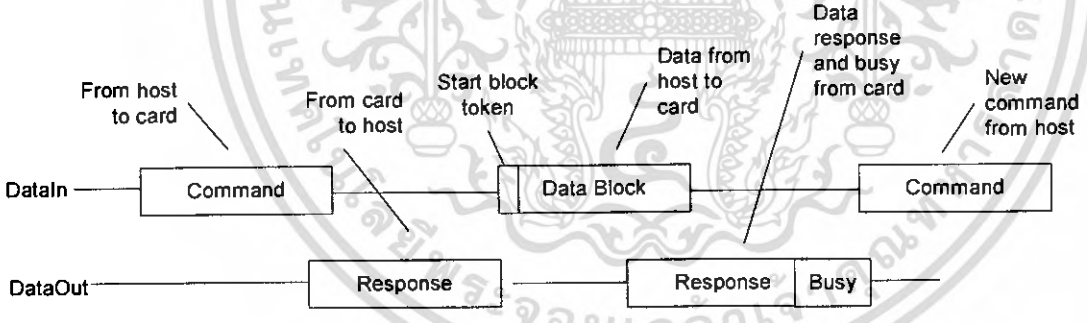


Figure 5-4. Single Block Write Operation

Every data block has a prefix or 'start block' token (one byte). After a data block is received the card will respond with a data-response token, and if the data block is received with no errors, it will be programmed. As long as the card is busy programming, a continuous stream of busy tokens will be sent to the host (effectively holding the dataOut line low).

Once the programming operation is completed, the host must check the results of the programming using the SEND_STATUS command (CMD13). Some errors (e.g., address out of range, write protect violation, etc.) are detected during programming only. The only validation check performed on the data block and communicated to the host via the data-response token is CRC and general Write Error indication.

Table 5-2. Description of SPI Bus Commands

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets the SD Card
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No				
CMD3	No				
CMD4	No				
CMD5	Reserved				
CMD6	Reserved				
CMD7	No				
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its card-specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its card identification (CID).
CMD11	No				
CMD12	Yes	None	R1b	STOP _TRANSMISSION	Forces the card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	No				
CMD15	No				
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).[1]
CMD17	Yes	[31:0] data address	R1	READ_SINGLE _BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.[2]
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE _BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_ TRANSMISSION command.
CMD19	Reserved				
CMD20	No				
CMD21 ... CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1[3]	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.[4]
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop *transmission token is sent (instead of 'start block').*

1) The only valid block length for write is 512 bytes. The valid block length for read is 1 to 512 bytes. A set block length of less than 512 bytes will cause a write error. The card has a default block length of 512 bytes. CMD16 is not mandatory if the default is accepted.
2) The start address and block length must be set so that the data transferred will not cross a physical block boundary.
3) Data followed by data response plus busy.
4) The start address must be aligned on a sector boundary. The block length is always 512 bytes.

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD26	No				
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28[1]	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WP_GRP_SIZE).
CMD29[4]	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_ PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.[2]
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	ERASE_WR_BLK_START _ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WR_BLK_END _ADDR	Sets the address of the last write block in a continuous range to be erased.
CMD34 .... CMD37	Reserved				
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.
CMD39	No				
CMD40	No				
CMD41 ... CMD54	Reserved				
CMD55	Yes	[31:0] stuff bits	R1	APP_CMD	Notifies the card that the next command is an application specific command rather than a standard command.
CMD56	Yes	[31:0] stuff bits [0]: RD/WR.[3]	R1	GEN_CMD	Used either to transfer a Data Block to the card or to get a Data Block from the card for general purpose/application specific commands. The size of the Data Block is defined with SET_BLOCK_LEN command.
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR register of a card.
CMD59	Yes	[31:1] don't care* [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on, a '0' will turn it off.
CMD60-63	No				

* The bit places must be filled but the values are irrelevant.

1) These features are not currently supported in the SanDisk SD Card.
2) 32 write protection bits (representing 32 write protect groups starting at the specified address) followed by 16 CRC bits are transferred in a payload format via the data line.
3) RD/WR_: "1"=the host will get a block of data from the card. "0"=the host sends a block of data to the card.

Table 5-3 describes all the application specific commands supported or reserved by the SD Card. All the following commands should be preceded with APP_CMD (CMD55).

Table 5-3. Application Specific Commands Used or Reserved by the SD Card–SPI Mode

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
ACMD6	No				
ACMD13	Yes	[31:0] stuff bits	R2	SD_STATUS	Send the SD Card status. The status fields are given in Table 4-21
ACMD17	Reserved				
ACMD18	Yes	–	–	–	Reserved for SD security applications1
ACMD19 to ACMD21	Reserved				
ACMD22	Yes	[31:0] stuff bits	R1	SEND_NUM_WR_BLOCKS	Send the numbers of the well-written (without errors) blocks. Responds with 32bit+CRC data block.
ACMD23	Yes	[31:23] stuff bits [22:0]Number of blocks	R1	SET_WR_BLK_ ERASE_COUNT	Set the number of write blocks to be pre-erased before writing (to be used for faster Multiple Block WR command). "1"=default (one wr block)(2).
ACMD24	Reserved				
ACMD25	Yes	–	–	–	Reserved for SD security applications1
ACMD26	Yes	–	–	–	Reserved for SD security applications1
ACMD38	Yes	–	–	–	Reserved for SD security applications1
ACMD39 to ACMD40	Reserved				
ACMD41	Yes	None	R1	SEND_OP_ COND	Activates the card's initialization process.
ACMD42	Yes	[31:1] stuff bits [0]set_cd	R1	SET_CLR_CARD_ DETECT	Connect[1]/Disconnect[0] the 50KOhm pull-up resistor on CD/DAT3 (pin 1) of the card. The pull-up may be used for card detection.
ACMD43 ... ACMD49	Yes	–	–	–	Reserved for SD security applications.[1]
ACMD51	Yes	[31:0] staff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR).

NOTES: (1) Refer to "SD Card Security Specification" for detailed explanation about the SD Security Features
(2) Command STOP_TRAN (CMD12) shall be used to stop the transmission in Write Multiple Block whether the pre-erase (ACMD23) feature is used or not.

5.2.3. Responses

There are several types of response tokens. As in the SD Card mode, all are transmitted MSB first.

5.2.3.1. Format R1

This response token is sent by the card after every command with the exception of SEND_STATUS commands. It is 1 byte long, the MSB is always set to zero and the other bits are error indications. A '1' signals error.

- In idle state—The card is in idle state and running initializing process.
- Erase reset—An erase sequence was cleared before executing because an out of erase sequence command was received.
- Illegal command—An illegal command code was detected.
- Communication CRC error—The CRC check of the last command failed.
- Erase sequence error—An error in the sequence of erase commands occurred.
- Address error—A misaligned address, which did not match the block length was used in the command.
- Parameter error—The command's argument (e.g., address, block length) was out of the allowed range for this card.

The structure of the R1 format is shown in Figure 5-7.

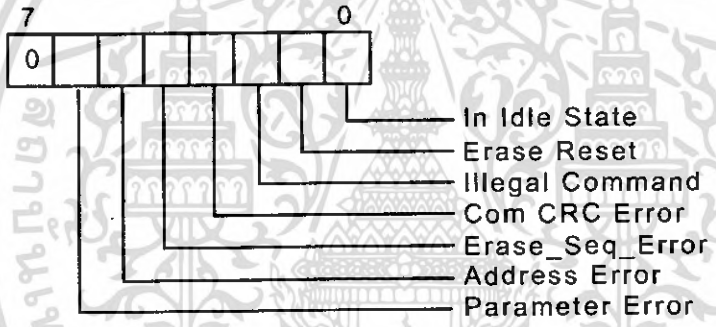


Figure 5-7. R1 Response Format

5.2.3.2. Format R1b

This response token is identical to R1 format with the optional addition of the busy signal. The busy signal token can be any number of bytes. A zero value indicates card is busy. A non-zero value indicates card is ready for the next command.

5.2.3.3. Format R2

This 2-bytes long response token is sent by the card as a response to the SEND_STATUS command. The format of the R2 status is shown in Figure 5-8.

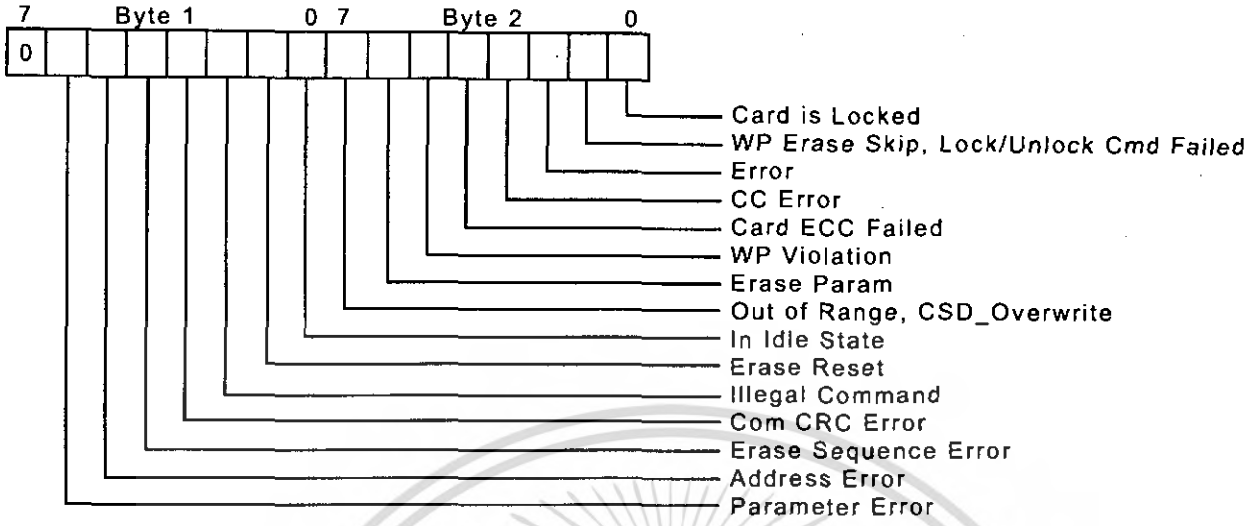


Figure 5-8. R2 Response Format

The first byte is identical to response R1. The content of the second byte is described below:

- **Erase param**—An invalid selection, sectors for erase.
- **Write protect violation**—The command tried to write a write-protected block.
- **Card ECC failed**—Card internal ECC was applied but failed to correct the data.
- **CC error**—Internal card controller error.
- **Error**—A general or an unknown error occurred during the operation.
- ***Write protect erase skip***—*Only partial address space was erased due to existing WP blocks.*
- **Card is locked**—Supported by the SanDisk SD Card.

5.2.3.4. Format R3

This response token is sent by the card when a READ_OCR command is received. The response length is 5 bytes. The structure of the first (MSB) byte is identical to response type R1. The other four bytes contain the OCR register.

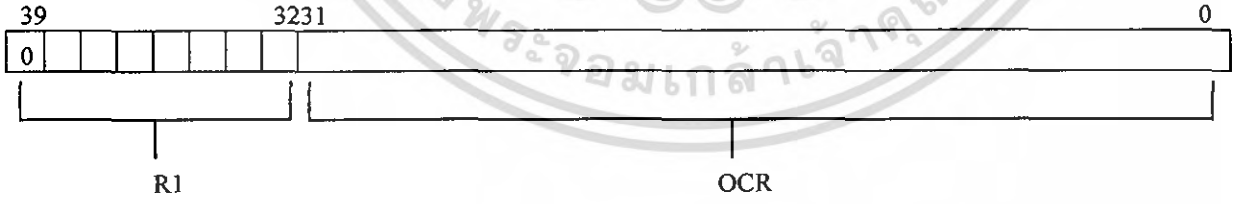
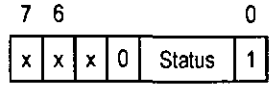


Figure 5-9. R3 Response Format

5.2.3.5. Data Response

Every data block written to the card is acknowledged by a data response token. It is one byte long and has the following format:



The meaning of the status bits is defined as follows:

- '010'—Data accepted.
- '101'—Data rejected due to a CRC error.
- '110'—Data Rejected due to a Write Error

In case of any error (CRC or Write Error) during Write Multiple Block operation, the host shall stop the data transmission using CMD12. In case of Write Error (response '110') the host may send CMD13 (SEND_STATUS) in order to get the cause of the write problem. ACMD22 can be used to find the number of well written write blocks.

5.2.4. Data Tokens

Read and write commands have data transfers associated with them. Data is being transmitted or received via data tokens. All data bytes are transmitted MSB.

Data tokens are 4 to 515 bytes long and have the following format:

For Single Block Read, Single Block Write and Multiple Block Read:

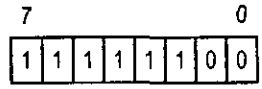
- First byte: Start Block.

- Bytes 2-513 (depends on the data block length): User data.
- Last two bytes: 16-bit CRC.

For Multiple Block Write operation:

- First byte of each block.

If data is to be transferred then—Start Block



If Stop transmission is requested—Stop Tran

1	1	1	1	1	1	0	1

Note that this format is used only for Multiple Block Write. In case of Multiple Block Read the stop transmission is done using STOP_TRAN Command (CMD12).

5.2.5. Data Error Token

If a read operation fails and the card cannot provide the required data it will send a data error token, instead. This token is one byte long and has the format shown in Figure 5-10.

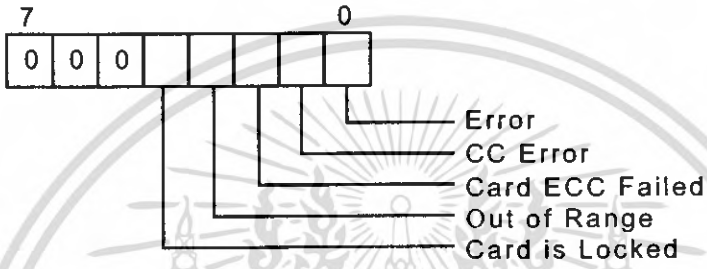


Figure 5-10. Data Error Token

The four least significant bits (LSB) are the same error bits as in response format R2.

5.2.6. Clearing Status Bits

As described in the previous paragraphs, in SPI mode, status bits are reported to the host in three different formats: response R1, response R2 and data error token (the same bits may exist in multiple response types—e.g., Card ECC failed). As in the SD mode, error bits are cleared when read by the host, regardless of the response format.

5.3. Card Registers

In SPI Mode, only the OCR, CSD and CID registers are accessible. Their format is identical to their format in the SD Card mode. However, a few fields are irrelevant in SPI mode.

5.4. SPI Bus Timing Diagrams

All timing diagrams use the schematics and abbreviations listed in Table 5-5.

Table 5-4. SPI Bus Timing Abbreviations

H	Signal is high (logical '1')
L	Signal is low (logical '0')
X	Don't care
Z	High impedance state (-> = 1)
*	Repeater
Busy	Busy Token
Command	Command token
Response	Response token
Data block	Data token

All timing values are defined in Table 5-5. The host must keep the clock running for at least N_{CR} clock cycles after the card response is received. This restriction applied to command and data response tokens.

5.4.1. Command/Response

Host Command to Card Response–Card is Ready

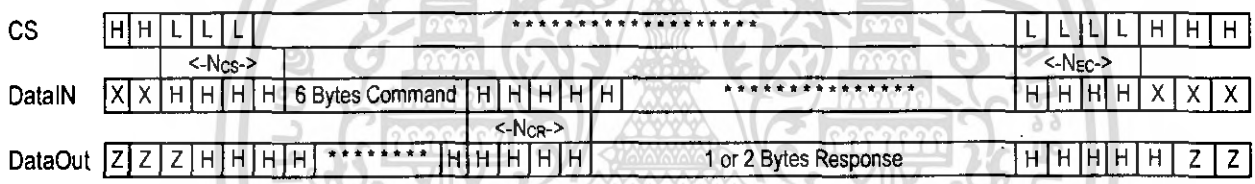


Figure 5-11. Host Command to Card Response–Card is Ready

Host Command to Card Response–Card is Busy

The following timing diagram describes the command response transaction for commands when the card responses which the R1b response type (e.g., SET_WRITE_PROT and ERASE). When the card is signaling busy, the host may deselect it (by raising the CS) at any time. The card will release the DataOut line one clock after the CS going high. To check if the card is still busy it needs to be reselected by asserting (set to low) the CS signal. The card will resume busy signal (pulling DataOut low) one clock after the falling edge of CS.

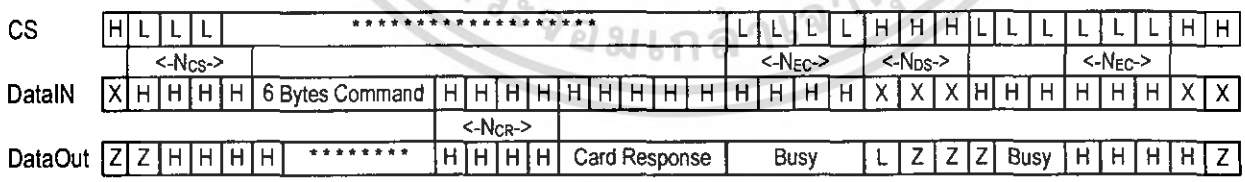


Figure 5-12. Host Command to Card Response–Card is Busy

Card Response to Host Command

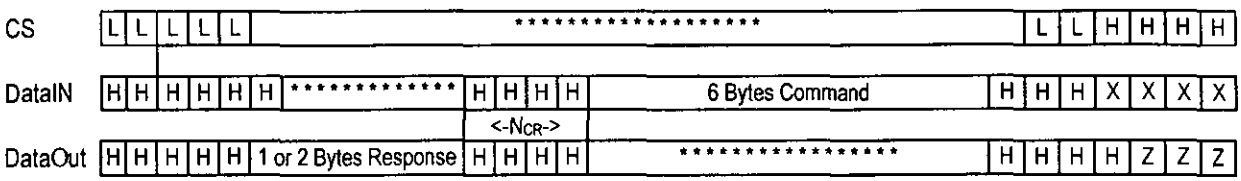


Figure 5-13. Card Response to Host Command

5.4.2. Data Read

The following timing diagram describes all single block read operations with the exception of SEND_CSD command.

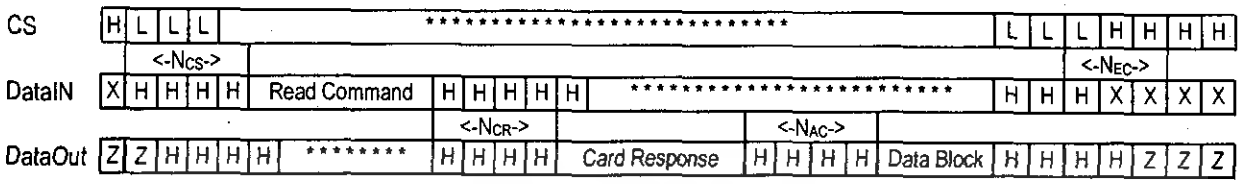


Figure 5-14. Single Block Read Timing

The following table describes Stop transmission operation in case of Multiple Block Read.

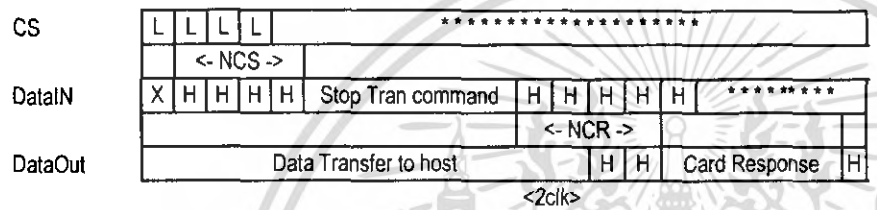


Figure 5-15. Multiple Block Read Timing

Reading the CSD Register

The following timing diagram describes the SEND_CSD command bus transaction. The timeout values for the response and the data block are N_{CR} (Since the N_{AC} is still unknown).

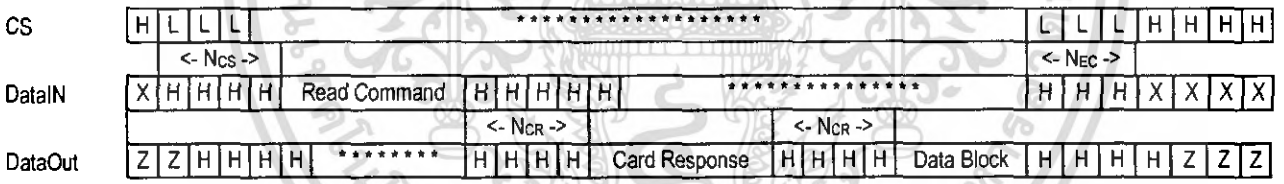


Figure 5-16. Reading the CSD Register

5.4.3. Data Write

The host may deselect a card (by raising the CS) at any time during the card busy period (refer to the given timing diagram). The card will release the DataOut line one clock after the CS going high. To check if the card is still busy it needs to be re-selected by asserting (set to low) the CS signal.

The card will resume busy signal (pulling DataOut low) one clock after the falling edge of CS.

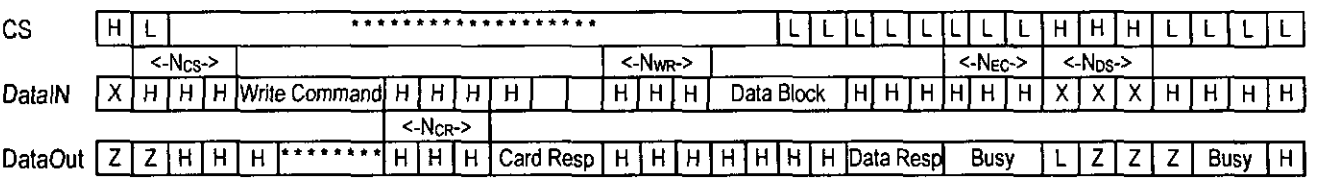
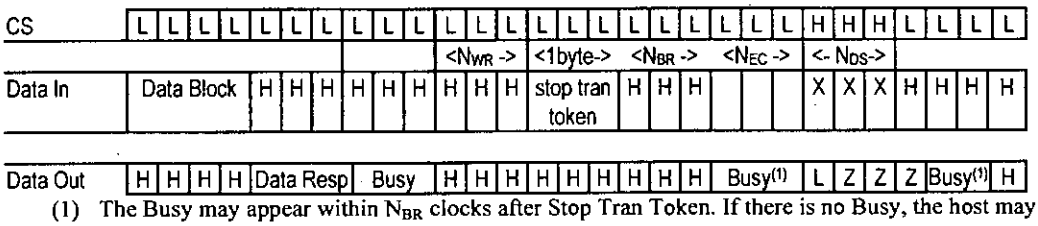


Figure 5-17. Device Write Timing

The following figure describes stop transmission operation in Multiple Block Write transfer.



(1) The Busy may appear within N_{BR} clocks after Stop Tran Token. If there is no Busy, the host may continue to the next command.

Figure 5-18. Stop Transmission Timing—Multiple Block Write

5.4.4. Timing Values

Table 5-5 shows the timing values and definitions. For more information, refer to Table 4-17 in Section 4.0, Section 5.1.9.2, and the applications note in Appendix A, "Host Design Considerations: NAND MMC and SD-based Products."

Table 5-5. Timing Constants Definitions

	Min	Max	Unit
N_{CS}	0	-	8 Clock Cycles
N_{CR}	0	8	8 Clock Cycles
N_{RC}	1	-	8 Clock Cycles
N_{AC}	1	See Note	8 Clock Cycles
N_{WR}	1	-	8 Clock Cycles
N_{EC}	0	-	8 Clock Cycles
N_{DS}	0	-	8 Clock Cycles
N_{BR}	0	1	8 Clock Cycles

NOTE: min [{{(TAAC * f) + (NSAC * 100)} * 1/8}, {(100ms * f) * 1/8}] where units = (8 clocks) and "f" is the clock frequency.

5.5. SPI Electrical Interface

The SPI Mode electrical interface is identical to that of the SD Card mode.

5.6. SPI Bus Operating Conditions

Identical to SD Card mode.

5.7. Bus Timing

Identical to SD Card mode. The timing of the CS signal is the same as any other card input.

Bolymin, Inc.

◆◆◆

LCD MODULE SPECIFICATION

MODEL NO.

BC1601A series

FOR MESSRS:

_____

ON DATE OF:

_____

APPROVED BY:

_____

◆◆◆

1. Numbering System

	B	C	2004	A	G	P	L	E	B	xxx
	0	1	2	3	4	5	6	7	8	9

0	Brand	Bolymin	
1	Module Type	C= character type G= graphic type P= TAB/TCP type	O= COG type F= COF type
2	Format	2002 20 characters, 4 lines 12232 122 x 32 dots	
3	Version No.	A type	
4	LCD Color	G STN/gray Y STN/yellow-green C color STN	B=STN/blue F=FSTN T=TN
5	LCD Type	R=positive reflective P=positive transflective	M=positive/transmissive N=negative/transmissive
6	Backlight type/color	L LED array yellow-green H LED edge white R LED array red G LED edge yellow-green	D=LED edge/blue E EL/white B=EL/blue C=CCFL/white
7	CGRAM Font	J=English/Japanese Font E=English European Font	C=English/Cyrillic Font H=English/Hebrew Font
8	View Angle/ Operating Temperature	B=Bottom Normal Temperature H=Bottom Wide Temperature U=Bottom Ultra wide Temperature	T=Top/Normal Temperature W=Top/Wide Temperature C=9H/Normal Temperature
9	Special Code	3=3 volt logic power supply n=negative voltage for LCD c=cable/connector xxx=to be assigned on data sheet	t=temperature compensation for LCD p=touch panel

◆◆◆

4. Absolute Maximum Ratings

4.1 Electrical Absolute Maximum Ratings

(Vss=0V, Ta=25℃)

Item	Symbol	Min	Max	Unit
Supply Voltage (Logic)	Vdd-Vss	-0.3	7	V
Supply Voltage (LCD Driver)	Vdd-Vo	-0.3	13	V
Input Voltage	VI	Vss	Vdd	V
Normal Type	TOP	0	+50	℃
	TSTG	-10	+60	℃
Wide Temperature Type	Top	-20	+70	℃
	Tstg	-30	+80	℃

4.2 Environmental Absolute Maximum Ratings

Item	Operating		Storage		Comment
	(Min.)	(Max.)	(Min.)	(Max.)	
Humidity	Note (2)		Note (2)		Without condensation
Vibration	--	4.9M/S^2	--	19.6M/S^2	XYZ Direction
Shock	--	29.4M/S^2	--	490M/S^2	XYZ Direction

Note (1) Ta = 0℃ : 50Hr Max.

Note (2) Ta ≦40℃ : 90% RH MAX

Ta > 40℃ : Absolute humidity must be lower than the humidity of 90% at 40℃.

◆◆◆

5. Electrical Characteristics

Item	Symbol	Condition	Min	Typ	Max	Unit
Supply Voltage For Logic	Vdd-Vss	—	3.0	—	5.5	V
Supply Voltage For LCD * Wide Temp、Type	Vdd-Vo	* Ta=-20℃	—	5.5	—	V
		Ta=0℃	—	—	—	V
		Ta=25℃	—	**4.5**	—	V
		Ta=50℃	—	—	—	V
		* Ta=+70℃	—	3.8	—	V
Input High Volt.	V_{IH}	—	2.2	—	Vdd	V
Input Low Volt.	V_{IL}	—	—	—	0.6	V
Output High Volt.	V_{OH}	—	2.4	—	—	V
Output Low Volt.	V_{OL}	—	—	—	0.4	V
Supply Current	Idd	Vdd=5V	—	1.2	—	mA

◆◆◆

7. Interface Pin Function

Pin No.	Symbol	Level	Description
1	Vss	0V	Ground
2	Vdd	5.0V	Supply Voltage for logic **(option +3V)**
3	Vo	(Variable)	Operating voltage for LCD
4	RS	H/L	H:DATA, L:Instruction code
5	R/W	H/L	H:Read(MPU→Module)L:Write(MPU→Module)
6	E	H,H→L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	A / Vee	—	Power supply for LED backlight (+) / Negative voltage output
16	K	—	Power supply for LED backlight (-)