USB HOST ชนิดพกพา

PORTABLE USB HOST

โดย

นายเผ่าพันธุ์ วีระกุล รหัส 47010466

นายพลกฤษณ์ ไตรเดช รหัส 47010491

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

USB HOST ชนิดพกพา

PORTABLE USB HOST

โดย

นายเผ่าพันธุ์ วีระกุล รหัส 47010466

นายพลกฤษณ์ ไตรเดช รหัส 47010491

อาจารย์ที่ปรึกษา

ดร. กสิน วิเชียรชม

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง USB Host ชนิดพกพา

(Portable USB Host)

ผู้จัดทำ

1. นายเผ่าพันธุ์ วีระกุล รหัส 47010466
2. นายพลกฤษณ์ ไตรเดช รหัส 47010491

..... อาจารย์ที่ปรึกษา

(ดร.กสิน วิเชียรชม)

USB Host แบบพกพา

นายเผ่าพันธุ์ วีระกุล รหัส 47010466
นายพลกฤษณ์ ไตรเดช รหัส 47010491
ดร.กสิน วิเชียรชม อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

USB HOST แบบพกพาจะทำหน้าที่แทนคอมพิวเตอร์ในการติดต่อระหว่างอุปกรณ์ ที่มีการเชื่อมต่อ แบบ USB ในเบื้องต้นนี้ได้ทำการออกแบบให้เชื่อมต่อกับอุปกรณ์ USB ที่เป็นอุปกรณ์ประเภทที่ใช้ในการเก็บข้อมูล ซึ่งจะมีส่วนประกอบอยู่ 3 ส่วน คือ 1) ส่วนรับข้อมูล โดยส่วนนี้จะเป็นช่อง USB สำหรับเสียบอุปกรณ์ USB ตัวที่เป็นต้นทางซึ่งมีข้อมูลที่ต้องการถ่ายโอนอยู่ 2) ส่วนเขียนข้อมูล โดยส่วนนี้จะเป็นช่อง USB สำหรับเสียบอุปกรณ์ USB ตัวที่ต้องการจะคัดลอกข้อมูลลงไป 3) ส่วนการควบคุม ใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมการทำงาน ซึ่งการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ USB นั้น จะใช้ตัววงจรรวมเบอร์ VNC1L ที่ทำหน้าที่แปลงสัญญาณระหว่างระบบ UART กับระบบ USB นอกจากการทำงานดังที่ได้กล่าวข้างต้นแล้ว USB HOST แบบพกพายังสามารถพัฒนาเพื่อทำการติดต่อกับอุปกรณ์อื่นๆได้อย่างหลากหลาย โดยการเพิ่มส่วนของ Hardware และพัฒนา Firmware ให้เหมาะสม

Portable USB Host

Mr.Powpan Werakul ID.47010466
Mr.Palagrit Tridech ID.47010491
Dr.Kasin Vichienchom Advisor
Academic Year 2007

Abstract

This report describes a design of the portable USB host. The host composes of a microcontroller and two USB bus interface chips. It allows two USB devices to communicate to each other without computer. In this prototype the data transfer between two USB thumb drives is demonstrated by IC VNC1L. This portable USB Host can be developed further to use with other USB devices such as mobile phone and digital camera by adding some hardware and firmware.

กิตติกรรมประกาศ

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

จึงขอขอบคุณมา ณ ที่นี้

นายเผ่าพันธุ์ วีระกุล 47010466

นายพลกฤษณ์ ไตรเดช 47010491

ผู้จัดทำ

สารบัญ

สารบัญรูป

เรื่อง **หน้า**

สารบัญตาราง

เรื่อง **หน้า**

บรรณานุกรม

1. ประจิน พลังสันติกุล, เรียนรู้และใช้งาน CCS C คอมไพเลอร์ เขียนโปรแกรมภาษา C ควบคุมไมโครคอนโทรลเลอร์ PIC, บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด, กรุงเทพฯ
2. ณัฏฐพล วงศ์สุนทรชัยและชัยวัฒน์ ลิ้มพรจิตรวิไล, เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ PIC 16F628, บริษัท อินโนเวทีฟ เอ็กเพอริเมนต์ จำกัด, กรุงเทพฯ
3. วารสารเซมิคอนดัคเตอร์อิเล็กทรอนิกส์
4. Hardware White Paper, Microsoft Extensible Firmware Initiative FAT32 File System Specification, Version 1.03, December 6, 2000, Microsoft Corporation
5. www.vinculum.com

บทที่ 1

บทนำ

1.1 ความเป็นมาของโครงงาน

ในปัจจุบันคอมพิวเตอร์ได้เข้ามามีอิทธิพลต่อชีวิตประจำวันของคนทั่วไป ทั้งในด้านการทำงาน การติดต่อสื่อสาร ความบันเทิง ล้วนมีการใช้งานกันอย่างแพร่หลาย ที่ผ่านมาการเชื่อมต่อของอุปกรณ์ทางคอมพิวเตอร์จะใช้ parallel port , serial port, PS/2 port เป็นต้น ซึ่งจะมีข้อเสียต่างๆมากมาย ทั้งด้านขนาดของ port ที่มีขนาดใหญ่ ความรวดเร็วในการถ่ายโอนข้อมูล การ detect อุปกรณ์ที่ส่วนมากจะต้องใช้ driver และการใช้งานที่ไม่มีความหลากหลาย ทำให้มีเทคโนโลยีที่ใช้กันเป็นมาตรฐานปัจจุบัน นั่นคือ USB port ที่มีข้อได้เปรียบต่างๆมากมาย อีกทั้งอุปกรณ์ที่เป็น USB มีราคาถูกลงอย่างเห็นได้ชัด เช่น USB thumb drive ,USB mouse ,USB keyboard เป็นต้น ทำให้อุปกรณ์ USB เข้ามาแทนที่ port แบบอื่นๆจนเรียกได้ว่าเป็นเทคโนโลยีหลักในการพัฒนาอุปกรณ์ ทางคอมพิวเตอร์

สำหรับโครงงานนี้จะเป็นการออกแบบอุปกรณ์ USB ที่ทำหน้าที่เป็น Host นั่นคือทำหน้าที่ติดต่อกับอุปกรณ์ USB แทนคอมพิวเตอร์นั่นเอง โดยจุดประสงค์หลักจะเป็นการโอนถ่ายข้อมูลระหว่างอุปกรณ์ที่เป็นdata storage เช่น thumb drive, โทรศัพท์มือถือ เป็นต้น เนื่องจากข้อมูลที่มีจำนวนมากและมีความสำคัญ USB Host จะทำให้การถ่ายโอนข้อมูลมีความสะดวกและรวดเร็วมากขึ้น โดยไม่จำเป็นต้องอาศัยคอมพิวเตอร์

1.2 วัตถุประสงค์

เพื่อศึกษาการเขียน/อ่านข้อมูลการ์ดหน่วยความจำของ SD Card, ศึกษาการติดต่อสื่อสารข้อมูลในลักษณะ SPI BUS, UART และแบบ USB, ศึกษาโครงสร้างของข้อมูลแบบ FAT, ตลอดจนเพิ่มทักษะในการใช้งานไมโครคอนโทรลเลอร์ตระกูล PIC

1.3 ขอบเขตโครงงาน

โครงงาน USB Host ชนิดพกพานี้ ได้ออกแบบให้สามารถอ่านข้อมูลจากอุปกรณ์ Data storage ตัวต้นทางเพื่อนำไปเขียนลงในอุปกรณ์ Data storage ตัวปลายทาง ซึ่งการติดต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ Data storage นี้จะติดต่อกันผ่านระบบ UART

1.4 โครงสร้างของรายงาน

รายงานฉบับนี้ได้อธิบายขั้นตอน วิธีในการออกแบบ รวมทั้งวงจรของ USB Host ชนิดพกพา ที่ดิดต่อกับ SDC/MMC และแบบที่ติดต่อกับ USB Device โดยมีเนื้อหาแบ่งเป็นบทต่างๆ ดังนี้

บทที่ 2 ทฤษฎี จะกล่าวถึงทฤษฎี และหลักการพื้นฐานต่างๆ ที่เกี่ยวข้องกับการออกแบบและสร้าง USB Host ชนิดพกพาทั้งแบบติดต่อกับ SDC/MMC และติดต่อกับ USB Device

บทที่ 3 การออกแบบ จะกล่าวถึงขั้นตอนในการออกแบบ ในการติดต่อกับหน่วยความจำแบบ SD Card และแบบ USB Device, การติดต่อกับ VNC1L โดยใช้ไมโครคอนโทรลเลอร์ PIC 18F8722

บทที่ 4 การทดลองและผลการทดลอง จะกล่าวถึงวิธีและขั้นตอนการทดลองต่างๆรวมถึงผลการทดลองด้วย

บทที่ 5 บทสรุป จะกล่าวถึงผลสรุปของการทดลอง ปัญหาที่เกิดขึ้น แนวทางการแก้ไขปัญหา และประโยชน์ที่ได้รับจากการทำโครงงานนี้

บทที่ 2

ทฤษฎี

2.1 ตารางการจัดเรียงไฟล์ (File Allocation Table : FAT)

ระบบปฏิบัติการของคอมพิวเตอร์แต่ละแพล็ตฟอร์ม (Platform) จะมีการจัดการระบบไฟล์ในฮาร์ดดิสก์ที่แตกต่างกัน บางระบบสามารถใช้ระบบไฟล์ได้หลายรูปแบบ โดยระบบไฟล์นั้นเป็นตารางที่ใช้บอกตำแหน่งของข้อมูลต่างๆที่อยู่บนฮาร์ดดิสก์ว่าอะไรอยู่ตรงไหน ปกติเมื่อซื้อฮาร์ดดิสก์มาใหม่ต้องทำการจัดข้อมูล (Format) ให้ฮาร์ดดิสก์ก่อนที่จะนำไปบรรจุข้อมูลการจัดข้อมูลในฮาร์ดดิสก์เป็นการแบ่งฮาร์ดดิสก์ออกเป็นส่วนๆ เพื่อให้คอมพิวเตอร์รู้ว่าตำแหน่งของข้อมูลอยู่ตรงไหน

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

ระบบจัดการไฟล์แบบ FAT เป็นระบบที่ไม่ยุ่งยากซับซ้อน ดังนั้นจึงถูกรองรับจากทุกระบบปฏิบัติการที่มีอยู่สำหรับคอมพิวเตอร์ส่วนบุคคล และยังสะดวกในการแลกเปลี่ยนข้อมูลระหว่างระบบปฏิบัติการที่แตกต่างกันซึ่งถูกติดตั้งบนคอมพิวเตอร์เครื่องเดียวกัน

ข้อด้อยของระบบจัดการไฟล์แบบ FAT คือ เมื่อไฟล์ถูกลบและไฟล์ใหม่ถูกเขียนลงไปแฟร็กเมนต์ (fragment) ของแต่ละไฟล์จะมีโอกาสกระจัดกระจายออกไปอยู่ทั่วทั้งหน่วยความจำส่งผลให้การอ่านและการเขียนไฟล์ทำได้ช้า การจัดเรียงข้อมูลเป็นหนึ่งในวิธีการทำให้การจัดเรียงไฟล์แบบ FAT เป็นระเบียบ แต่จะต้องทำการจัดเรียงข้อมูลอยู่บ่อยๆ และเป็นกระบวนการที่ใช้เวลามาก

ระบบไฟล์ FAT มีหลายรุ่นดังต่อไปนี้

2.1.1 FAT12

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

2.1.2 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 และมีเนื้อที่บางส่วนใช้เก็บข้อมูลเกี่ยวกับตัวFAT เอง) FAT16 นั้นถูกออกแบบมาเพื่อใช้งานกับไฟล์ต่างๆบนดิสก์ขนาดเล็กหรือขนาดกลาง ซึ่งมีเนื้อที่ประมาณ 2048 เมกะไบต์

ปัญหาของ FAT16 คือ

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

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

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

2.1.3 FAT32

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

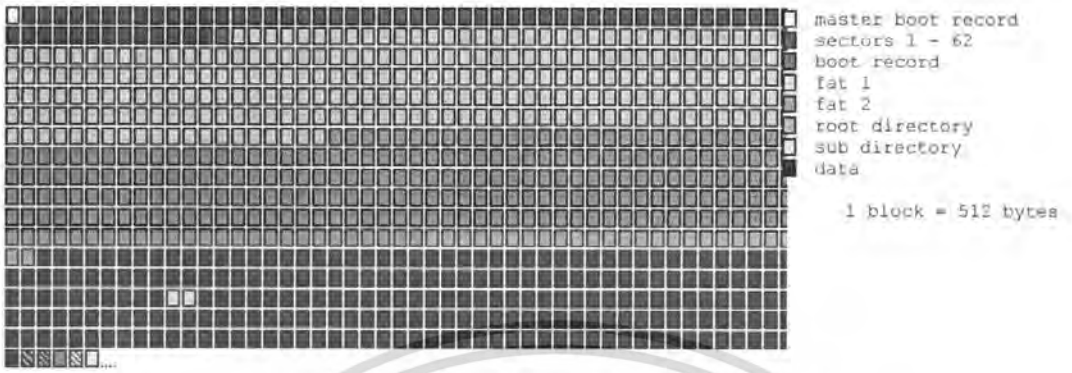
ตารางที่ 2.1 เปรียบเทียบระหว่าง FAT12, FAT16 และ FAT32

	FAT12	**FAT16**	**FAT32**
Developer	Microsoft		
Full Name	File Allocation Table		
	(12-bit version)	(16-bit version)	(32-bit version)
Introduce	1977 (Microsoft Disk BASIC)	July 1988 (MS-DOS 4.0)	August 1996 (Window 95 OSR2)
Structures			
Directory contents	Table		
File allocation	Linked List		
Bad blocks	Linked List		
Limits			

Max file size	32 MB	2 GB	4 GB
Max number of files	4,077	65,517	268,435,437
Max file name size	8.3 or 255 when using LFNs		
Max volume size	32 MB	2 GB 4 GB with some implementation	2 TB
Features			
Dates recorded	Creation, modified, access		
Date range	January 1, 1980 – December 31, 2107		
Forks	Not natively		
Attributes	Read-only, hidden, system, volume label, subdirectory, archive		
Permissions	No		
Transparent compression	Per-volume, stacker, DoubleSpace, DriveSpace		No
Transparent encryption	Per-volume only with DR-DOS		No

2.1.4 โครงสร้างของดิสก์หลัก (Main disk structures)

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



รูปที่ 2.1 โครงสร้างของดิสก์

1. เซ็คเตอร์สำรอง (Reserved sectors) อยู่ในส่วนต้นๆ โดยเซ็คเตอร์สำรองแรกสุดเป็นบูทเซ็คเตอร์ (Boot Sector) ซึ่งรวมพื้นที่ที่เรียกว่า BIOS Parameter Block โดยปกติจะประกอบด้วยโค้ดเริ่มต้นของระบบปฏิบัติการ ซึ่งจำนวนของเซ็คเตอร์สำรองทั้งหมดจะถูกระบุอยู่ในส่วนของนี้ ข้อมูลสำคัญต่างๆจากบูทเซ็คเตอร์สามารถเข้าถึงผ่านโครงสร้างระบบปฏิบัติการ ที่เรียกว่า Drive Parameter Blockใน DOS และ OS/2

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

3. บริเวณไดเร็คทอรี่ (Root Directory Region) นี่คือตารางไดเรคทอรี่ซึ่งเก็บข้อมูลเกี่ยวกับไฟล์และไดเร็คทอรี่ต่างๆ ในระบบจัดการไฟล์แบบ FAT12 และ FAT16 ตารางไดเร็คทอรี่นี้จะอยู่ในบริเวณไดเร็คทอรี่เท่านั้น และมีขนาดสูงสุดที่แน่นอน แต่ในระบบจัดการไฟล์แบบ FAT32 ตารางไดเร็คทอรี่จะอยู่รวมกับไฟล์ต่างๆในบริเวณข้อมูล ซึ่งสามารถขยายขนาดออกไปได้ไม่จำกัด

4. บริเวณข้อมูล (Data Region) เป็นส่วนที่มีไฟล์ข้อมูลอยู่จริง ขนาดของไฟล์และไดเร็คทอรี่ย่อยสามารถเพิ่มขึ้นได้เท่าที่มีคลัสเตอร์เหลืออยู่

2.1.4.1 บูทเซ็คเตอร์และโครงสร้างBPB(Bios Parameter Block)

ตารางที่ 2.2 โครงสร้างพื้นฐานของบูทเซ็คเตอร์และBPBซึ่งใช้ในทุกเวอร์ชั่นของFAT

ชื่อ	ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด
BS_jmpBoot	0x00	3	คำสั่งกระโดด(เพื่อที่จะข้ามส่วนหัวของการบูท)
BS_OEMName	0x03	8	ชื่อ OEM ค่าพื้นฐานคือ IBM 3.3 และ MS-DOS 5.0
BPB_BytsPerSec	0x0B	2	จำนวนไบต์ต่อหนึ่งเซ็คเตอร์
BPB_SecPerClus	0x0D	1	จำนวนเซ็คเตอร์ต่อหนึ่งคลัสเตอร์
BPB_RsvdSecCnt	0x0E	2	จำนวนของเซ็คเตอร์สำรองในบริเวณReserved
BPB_NumFATs	0x10	1	จำนวนของตารางการจัดเรียงข้อมูล
BPB_RootEntCnt	0x11	2	จำนวนของไดเร็คทอรี่ในบริเวณ Root directory สำหรับFAT32ค่านี้จะเป็นศูนย์เนื่องจากบริเวณ Root directoryของFAT32จะอยู่รวมกับไฟล์ต่างๆ ในบริเวณข้อมูล
BPB_TotSec16	0x13	2	จำนวนเซ็คเตอร์รวมทั้งหมดของบริเวณทั้งสี่ สำหรับ FAT32ค่านี้จะเป็นศูนย์
BPB_Media	0x15	1	ตัวบรรยายมีเดีย 0xF8 ด้านเดียว, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xF9 สองด้าน, 80 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xFA ด้านเดียว, 80 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFB สองด้าน, 80 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFC ด้านเดียว, 40 แทรกต่อด้าน, 9 เซคเตอร์ต่อ

			แทรก 0xFD สองด้าน, 40 แทรกต่อด้าน, 9 เซคเตอร์ต่อแทรก 0xFE ด้านเดียว, 40 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก 0xFF สองด้าน, 40 แทรกต่อด้าน, 8 เซคเตอร์ต่อแทรก
BPB_FATSz16	0x16	2	จำนวนฌซ็คเตอร์ต่อหนึ่งตารางการจัดเรียงข้อมูล
BPB_SecPerTrk	0x18	2	จำนวนเซ็คเตอร์ด่อหนึ่งTrack
BPB_NumHeads	0x1A	2	จำนวนของเฮดสำหรับการอินเทอรัป0x13
BPB_HiddSec	0x1C	4	จำนวนเซ็คเตอร์ที่ถูกซ่อน
BPB_TotSec32	0x20	4	จำนวนเซ็คเตอร์รวมทั้งหมดของบริเวณทั้งสี่ สำหรับFAT16ค่านี้จะเป็นศูนย์

ตารางที่ 2.3 โครงสร้างของบูทเซ็คเตอร์และBPBที่เพิ่มขึ้นมาในFAT32

ชื่อ	ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด
BPB_FATSz32	0x24	4	จำนวนเซ็คเตอร์ต่อหนึ่งตารางการจัดเรียงข้อมูล
BPB_ExtFlags	0x28	2	เครื่องหมายตารางการจัดเรียงข้อมูล
BPB_FSVer	0x2A	2	เวอร์ชั่น
BPB_RootClus	0x2C	4	หมายเลขของคลัสเตอร์แรกในRoot directory
BPB_FSInfo	0x30	2	หมายเลขเซ็คเตอร์ของเซ็คเตอร์ข้อมูล FS
BPB_BkBootSec	0x32	2	หมายเลขเซ็คเตอร์ของสำเนาบูทเซ็คเตอร์
BPB_Reserved	0x34	12	ส่วนสำรอง
BS_DrvNum	0x40	1	หมายเลขฟิสิคอลไดรฟ์

BS_Reserved1	0x41	1	ส่วนสำรอง
BS_BootSig	0x42	1	ซิกเนเจอร์ (Signature)
BS_VolID	0x43	4	หมายเลขซีเรียล
BS_VolLab	0x47	11	โวลุมลาเบล (Volume Label)
BS_FilSysType	0x52	8	ชนิดของระบบไฟล์ ต้องเป็นFAT32เสมอ
	0x5A	420	โค้ดบูทระบบปฏิบัติการ
	0x1FE	2	เซ็คเตอร์สุดท้าย (0x55 , 0xAA)

2.1.4.2 ตารางไดเร็คทอรี่

ตารางไดเร็คทอรี่เป็นไฟล์ชนิดพิเศษที่แสดงไดเร็คทอรี่ (ปัจจุบันรู้จักกันในนามของโฟลเดอร์) แต่ละไฟล์หรือไดเร็คทอรี่ประกอบไปด้วย 32 ไบต์ แต่ละไบต์จะบันทึกชื่อไฟล์, นามสกุลไฟล์,คุณลักษณะของไฟล์ (ชนิดเอกสารสำคัญ, ไดเร็คทอรี่, ถูกซ่อน, อ่านได้อย่างเดียว, ระบบ, และความจุ),วัน เวลาที่ไฟล์ถูกสร้าง, แอดเดรสของคลัสเตอร์แรกของไฟล์หรือไดเร็คทอรี่นั้น และสุดท้ายคือขนาดของไฟล์หรือไดเร็คทอรี่

ตัวอักษรที่สามารถใช้ตั้งชื่อไฟล์ได้สำหรับ DOS

- ตัวพิมพ์ใหญ่ A-Z
- ตัวเลข 0-9
- ช่องว่าง (ช่องว่างยาวๆจะไม่นับเป็นส่วนของชื่อไฟล์)
- ! # $ % & ( ) - @ ^ _ ‘ { } ~
- ค่า 128 ─ 255

จำนวนไดเร็คทอรี่ทั้งหมดทั้งในบริเวณไดเร็คทอรี่และในไดเร็คทอรี่ย่อย มีรูปแบบ ดังนี้

ตารางที่ 2.4 โครงสร้างของตารางไดเร็คทอรี่

ไบต์ออฟเซ็ต	ขนาด (ไบต์)	รายละเอียด
0x00	8	0x00
0x08	3	เครื่องหมายของตารางจัดเรียงข้อมูล

<table>
<tr><td rowspan="9">0x0B</td><td rowspan="9">1</td><td>บิต</td><td>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>โวลุมลาเบล</td></tr>
<tr><td>4</td><td>0x10</td><td>ไดเร็คทอรี่ย่อย</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 ถูกใช้โดยNT</td></tr>
<tr><td>0x0D</td><td>1</td><td colspan="3">เวลาสร้างไฟล์, ความละเอียด 10 มิลลิวินาที ค่าอยู่ที่ระหว่าง 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, 12=ธันวาคม)</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 ดูไบต์ออฟเซ็ต 0x01 ประกอบ</td></tr>
<tr><td>0x14</td><td>2</td><td colspan="3">ดัชนี EA (ถูกใช้โดย OS/2 และ NT) ในFAT12และFAT16 หรือ 2ไบต์สูงของคลัสเตอร์แรกใน FAT32</td></tr>
<tr><td>0x16</td><td>2</td><td colspan="3">เวลาที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซ็ต 0x0E ประกอบ</td></tr>
<tr><td>0x18</td><td>2</td><td colspan="3">วันที่แก้ไขไฟล์ล่าสุด ดูไบต์ออฟเซ็ต 0x10 ประกอบ</td></tr>
<tr><td>0x1A</td><td>2</td><td colspan="3">คลัสเตอร์แรกใน FAT12 และ FAT16, หรือ 2ไบต์ต่ำของคลัสเตอร์แรกในFAT32</td></tr>
</table>

0x1C	4	ขนาดไฟล์

2.1.4.3 ตาราง FAT

ขนาดของแต่ละคลัสเตอร์ขึ้นอยู่กับชนิดของระบบไฟล์แบบ FAT ที่ใช้และขนาดของพาร์ติชันโดยปกติขนาดของคลัสเตอร์จะอยู่ระหว่าง 2 กิโลไบต์ถึง 32 กิโลไบต์ แต่ละไฟล์อาจจะกินเนื้อที่มากกว่า 1คลัสเตอร์ขึ้นอยู่กับขนาดของไฟล์นั้น แต่ไม่จำเป็นต้องเป็นคลัสเตอร์ที่ติดกัน

ตารางการจัดเรียงข้อมูลเป็นสมุดรายชื่อของไฟล์ที่สัมพันธ์กับตำแหน่งคลัสเตอร์ของไฟล์ แต่ละคลัสเตอร์ใน FAT จะบันทึกข้อมูล 1 ใน 5 ดังนี้

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

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

ตารางที่ 2.5 ค่าต่างๆในตารางการจัดเรียงข้อมูล

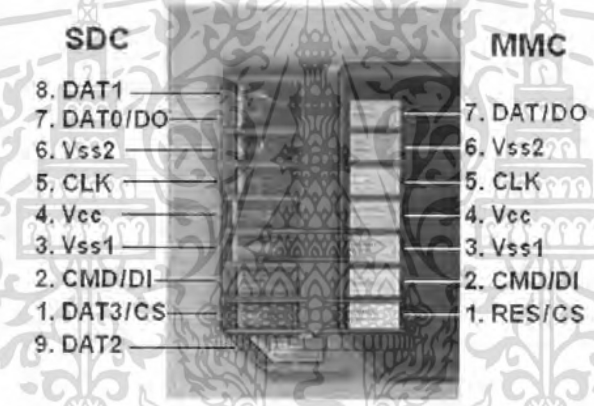
FAT12	FAT16	FAT32	รายละเอียด
0x000	0x0000	0x?0000000	คลัสเตอร์ว่าง
0x001	0x0001	0x?0000001	คลัสเตอร์สำรอง
0x002-0xFEF	0x0002-0xFFEF	0x?0000002-0x?FFFFFEF	คลัสเตอร์ที่ถูกใช้งาน, ค่าของคลัสเตอร์ถัดไป
0xFF0-0xFF6	0xFFF0-0xFFF6	0x?FFFFFF0-0x?FFFFFF6	ค่าสำรอง

0xFF7	0xFFF7	0x?FFFFFF7	คลัสเตอร์เสีย
0xFF8- 0xFFF	0xFFF8- 0xFFFF	0x?FFFFFF8- 0x?FFFFFFF	คลัสเตอร์สุดท้ายของไฟล์

สังเกตว่า FAT32 จะใช้เพียง 28 บิต จาก 32 บิตที่สามารถใช้ได้ โดยปกติ 4 บิตบนจะมีค่าเป็นศูนย์

2.2 SD Card และ MMC Card

2.2.1 หน้าสัมผัสสำหรับการเชื่อมต่อของการ์ด



รูปที่ 2.2 หน้าสัมผัสสำหรับการเชื่อมต่อของการ์ด

รูปที่ 2.2 แสดงหน้าสัมผัสสำหรับการเชื่อมต่อของ SDC (Secure Digital Memory Card) และ MMC (Multi Media Card) โดย MMC จะมี7ขา ส่วน SDC จะมี9ขา สามขาเป็นขาของ power supply คือ ขา4 และขา6 ขอบเขตของแรงดันที่ใช้ในการทำงานของการ์ดจะถูกกำหนดโดยรีจิสเตอร์ OCR ดังตารางที่ 2.6 แต่อย่างไรก็ตามคุณสมบัติโดยทั่วไปของ SDC และ MMC จะสามารถทำงานได้ที่แรงดัน 2.7 – 3.6 V.

ตารางที่2.6 OCR Register Definition

OCR Bit	VDD Voltage Window
0-7	Reserved
8	2.0-2.1

9	2.1-2.2
10	2.2-2.3
11	2.3-2.4
12	2.4-2.5
13	2.5-2.6
14	2.6-2.7
15	2.7-2.8
16	2.8-2.9
17	2.9-3.0
18	3.0-3.1
19	3.1-3.2
20	3.2-3.3
21	3.3-3.4
22	3.4-3.5
23	3.5-3.6
24-30	Reserved
31	Card power up status bit (Busy)

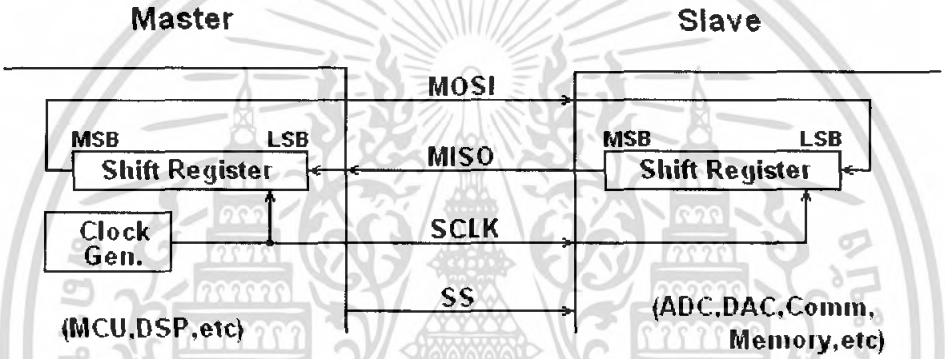
2.2.2 โหมดการติดต่อแบบ SPI (Serial Peripheral Interface)

SPI เป็นอีกหนึ่งโหมดการติดต่อที่ใช้สำหรับ SDC และ MMC นอกเหนือจาก Native mode ซึ่งโหมด SPI นี้จะใช้งานได้ง่ายกว่าเมื่อเทียบกับโหมด Native เนื่องจากในไมโครคอนโทรลเลอร์ส่วนใหญ่จะมี port SPI หรือ GPIO ที่ไว้ใช้สำหรับการเชื่อมต่อโหมดนี้อยู่ในตัวแล้ว ดังนั้นการเชื่อมต่อแบบ SPI จึงเหมาะกับงานที่มีราคาไม่สูง

2.2.2.1 โครงสร้างของ SPI

โครงสร้างของการเชื่อมต่อแบบ SPI แสดงดังรูปที่ 2.3 ตัว Master และตัว Slave จะติดต่อกันด้วยสายสัญญาณสามเส้น คือ SCLK (Serial Clock), MISO (Master-In Slave-Out) และ MOSI (Master-Out Slave-In) ส่วนสาย SS# (Slave Selected) ที่เพิ่มขึ้นมานั้นจะใช้เป็นสายสำหรับเลือกตัวSlaveที่จะมาติดต่อกับตัว Master

ในโหมด SPI นี้ การส่งข้อมูลจะทำการส่ง MSB (Most Significant Bit) ออกไปก่อน



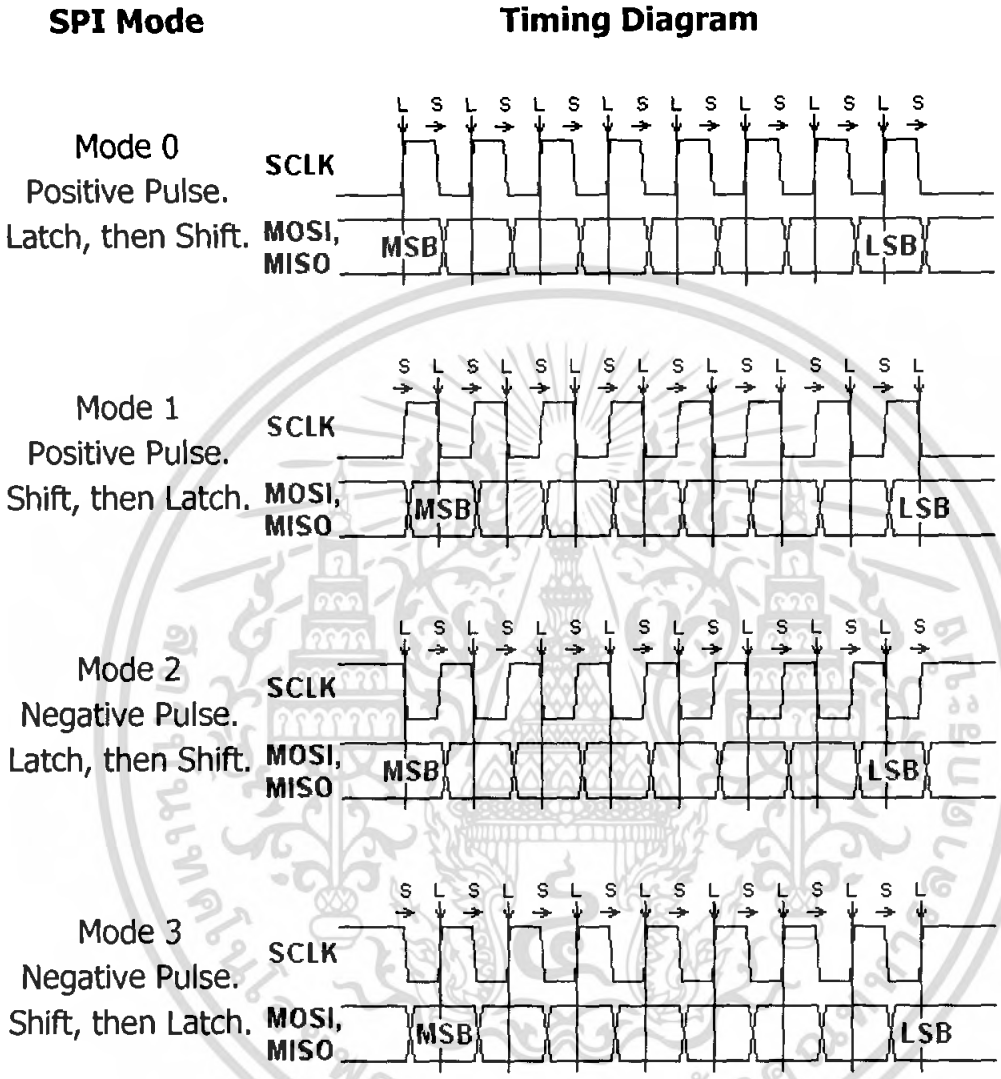
รูปที่ 2.3 โครงสร้างอย่างง่ายของการติดต่อในโหมด SPI

2.2.2.2 จังหวะเวลาในการถ่ายโอนข้อมูลของโหมด SPI

ในโหมด SPI การshift และ latch ของข้อมูลจะกระทำในช่วงที่ตรงกันข้ามของขอบสัญญาณ clock ซึ่งจะสามารถแบ่งออกได้เป็น 4 โหมดดังรูปที่ 2.4

สำหรับ SDC เมื่อมีการติดต่อแบบ SPI ควรใช้ 'SPI Mode 0' แต่สำหรับ MMC นั้นสามารถใช้โหมดใดก็ได้ ดังนั้นโดยทั่วไปจึงมักจะใช้ 'SPI Mode 0' (positive clock, front edge latch, back edge shift) ในการติดต่อกับ SDC/MMC อย่างไรก็ตาม 'SPI Mode 3' ก็สามารถใช้งานได้ดีเช่นเดียวกัน

SPI Transfer Timing

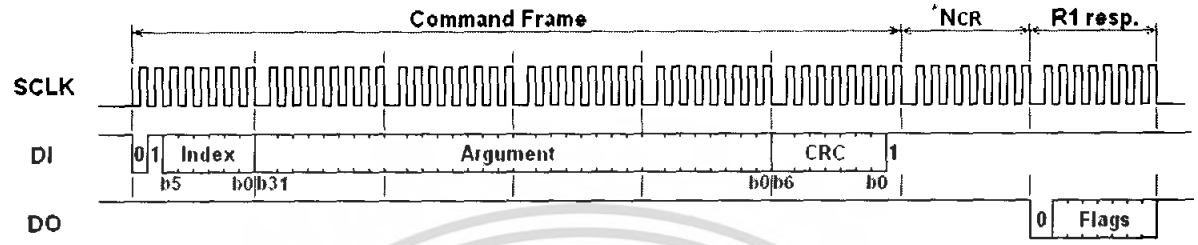


รูปที่ 2.4 Timing diagram ของการติดต่อแบบSPIทั้ง4โหมด

2.2.2.3 คำสั่งและการตอบกลับ

ในโหมด SPI นั้น โครงสร้างของคำสั่งจะถูกกำหนดไว้ให้มีความยาว6ไบต์ดังแสดงในรูปที่ 2.5 เมื่อชุดคำสั่งถูกส่งไปที่การ์ด จะมีการตอบกลับจากการ์ดไปที่โฮสท์ในรูปแบบ R1, R2 หรือ R3 เนื่องจากการถ่ายโอนข้อมูลนั้นจะถูกขับโดยclockแบบอนุกรมที่โฮสท์สร้างขึ้น ดังนั้นโฮสท์จะต้องสร้างclockต่อไปเรื่อยๆจนกว่าจะได้รับการตอบกลับจากการ์ด ช่วงระยะเวลาก่อนที่การ์ดจะตอบกลับหลังจากได้รับคำสั่งจากโฮสท์ (NCR) คือ 0 ถึง 8 ไบต์ สำหรับ SDC และ 1 ถึง 8

ไบต์ สำหรับ MMC ในช่วงที่มีการโอนถ่ายข้อมูลนี้ ขา SS จะต้องถูกกำหนดให้มีสถานะลอจิก เป็น '0' เสมอ



รูปที่ 2.5 command frame ที่ส่งจากโฮสท์ไปหาการ์ด

2.2.2.4 ชุดคำสั่งที่ใช้ในโหมด SPI

คำสั่งดังแสดงให้เห็นนี้เป็นเพียงคำสั่งโดยทั่วไปที่ใช้สำหรับการอ่าน เขียนและ initial การ์ด เท่านั้น

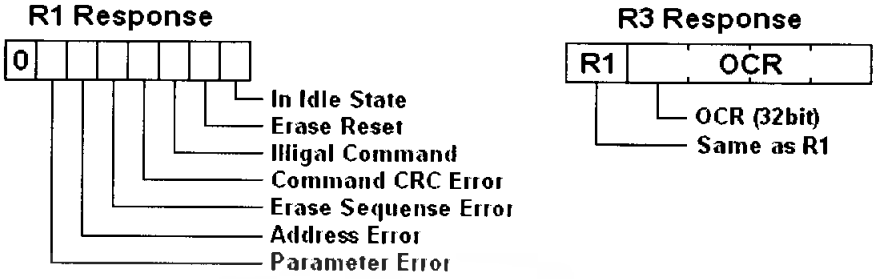
Command Index	Argument	Response Data	Abbreviation	Description
CMD0	None(0)	R1	No GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No SEND_OP_COND	Initiate initialization process.
CMD9	None(0)	R1	Yes SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes SEND_CID	Read CID register.
CMD12	None(0)	R1b	No STOP_TRANSMISSION	Stop to read data.
CMD17	Address[31:0]	R1	Yes READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of	R1	No SET_BLOCK_COUNT	For only MMC. Define

	blocks[15:0]				number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55	None(0)	R1	No	APP_CMD	Application specific command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.

*1:ACMD<n> means a command sequence of CMD55-CMD<n>.

2.2.2.5 Command Response ในโหมดSPI

การตอบกลับในโหมด SPI จะมีอยู่สามรูปแบบ คือ R1, R2 และ R3 ดังรูปที่ 2.6 ขึ้นอยู่กับคำสั่งที่โฮสท์ส่งมา แต่คำสั่งส่วนมากจะมีการตอบกลับในรูปแบบ R1 ถ้าหาก R1 มีค่าเป็น 0x00 แสดงว่าไม่มีข้อผิดพลาดเกิดขึ้น ส่วนการตอบสนองแบบ R3 นั้นจะตอบกลับเฉพาะ CMD58 เท่านั้น

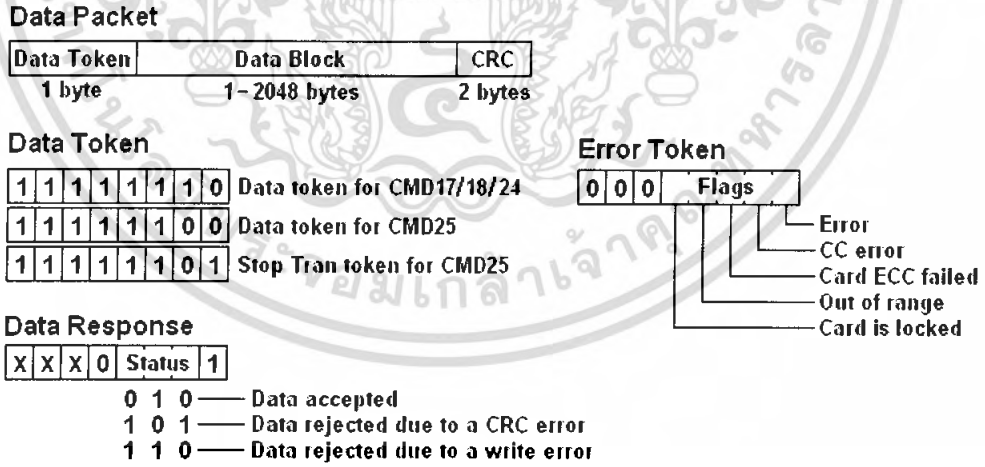


รูปที่ 2.6 การตอบกลับแบบต่างๆ

2.2.3 การถ่ายโอนข้อมูล

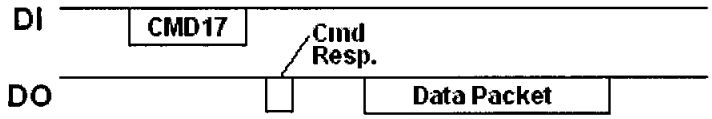
2.2.3.1 Data Packet และ Data response

ในการถ่ายโอนข้อมูล จะมีการถ่ายโอนหลังจากมีการส่งcommand responseแล้ว บล็อกข้อมูลจะถูกถ่ายโอนในรูปData Packetซึ่งประกอบไปด้วย Data Token, Data Block และCRC ดังแสดงในรูปที่ 2.7 จะเห็นได้ว่าData Token จะมีอยู่สามแบบ ซึ่งขึ้นอยู่กับแต่ละคำสั่งที่ส่ง



รูปที่ 2.7 โครงสร้างของData Packet

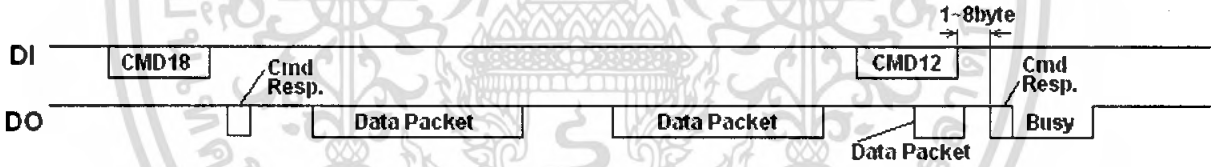
2.2.3.2 คำสั่ง Single Block Read



รูปที่ 2.8 timing diagram ของคำสั่ง Single Block Read

อาร์กิวเมนท์ที่ต่อท้ายคำสั่งนี้จะเป็นตัวกำหนดแอดเดรสเริ่มต้นของข้อมูลที่จะอ่าน เมื่อคำสั่งนี้ได้รับการตอบกลับ กระบวนการอ่านข้อมูลก็จะเริ่มขึ้นโดยชุดข้อมูลที่ต้องการอ่านนั้นก็จะถูกส่งไปยังโฮสท์ เมื่อโฮสท์ตรวจพบData tokenที่ส่งมา โฮสท์ก็จะทำการอ่านบล็อกข้อมูลที่ตามหลังTokenนั้น เมื่อมีความผิดพลาดเกิดขึ้นในการทำงานTokenที่แสดงความผิดพลาดจะถูกส่งมาแทนที่บล็อกข้อมูล

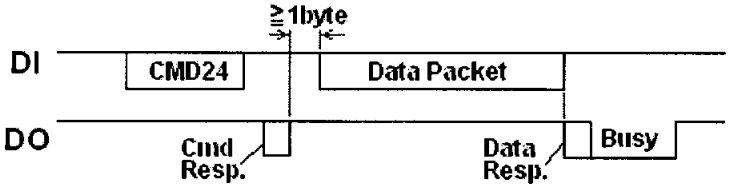
2.2.3.3 คำสั่ง Multiple Block Read



รูปที่ 2.9 timing diagram ของคำสั่ง Multiple Block Read

คำสั่งนี้เป็นการอ่านข้อมูลครั้งละหลายๆบล็อกจากตำแหน่งแอดเดรสที่กำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการอ่านต่อหนึ่งครั้งไว้ กระบวนการอ่านนี้จะดำเนินต่อเนื่องไปเรื่อยๆจนกว่าจะมีการส่งคำสั่ง CMD12 ไปที่การ์ด การอ่านข้อมูลนี้จึงจะหยุดลง

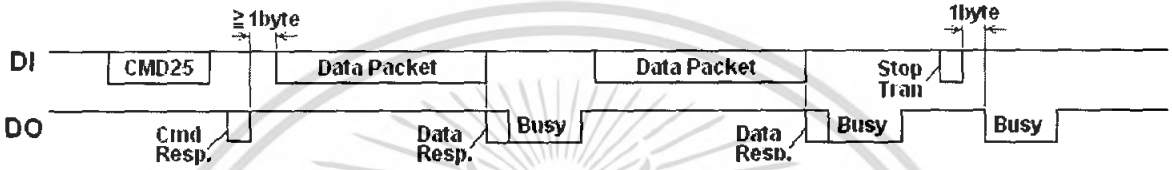
2.2.3.4 คำสั่ง Single Block Write



รูปที่ 2.10 timing diagram ของคำสั่ง Single Block Write

เมื่อคำสั่งนี้ได้รับการตอบรับ โฮสท์ก็จะทำการส่งแพ็คเกจข้อมูลไปที่การ์ดหลังจากนั้นเป็นช่วงประมาณหนึ่งไบต์ แพ็คเกจของข้อมูลนี้ก็จะมีลักษณะเหมือนกับแพ็คเกจของข้อมูลในคำสั่ง Read เมื่อข้อมูลถูกส่งไปที่การ์ดแล้ว การดก็จะตอบกลับมาด้วย Data Response ในทันที การ์ดโดยส่วนใหญ่จะสามารถเขียนข้อมูลได้บล็อกละ 512 ไบต์

2.2.3.5 คำสั่ง Multiple Block Write



รูปที่ 2.11 timing diagram ของคำสั่ง Multiple Block Write

คำสั่งนี้จะเขียนข้อมูลครั้งละหลายๆบล็อกลงไปในแอดเดรสที่เรากำหนด ถ้าหากไม่มีการกำหนดจำนวนบล็อกในการถ่ายโอน การถ่ายโอนนี้ก็จะดำเนินต่อไปเรื่อยๆจนกว่าจะถูกหยุดโดย Stop Tran Token ซึ่งก็จะทำให้เกิด Busy flag ขึ้นเป็นช่วงระยะ1ไบต์ต่อจาก Stop Tran Token

2.3 การสื่อสารแบบ UART

UART ย่อมาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัส (Asynchronous) นั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารแบบอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัสแล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าคอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังทำการแจ้งข้อมูลอื่นๆ ให้คอมพิวเตอร์ทราบด้วย เช่น อัตราความเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างอัตราการถ่ายทอดข้อมูลแบบโปรแกรมได้ (Programmable Baudrate Generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิตดังนั้นจึงกำหนดตัวหารให้อยู่ในช่วง 10 – 65,535

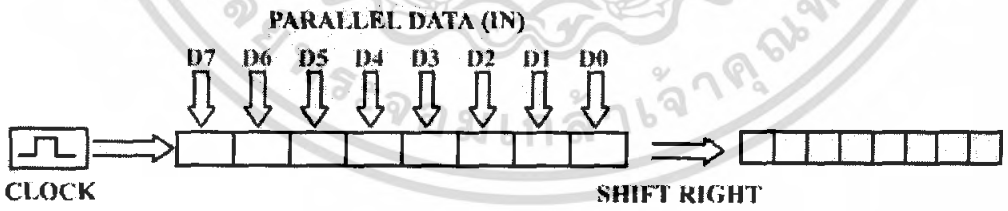
UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์ (Half Duplex) และฟูลดูเพล็ก (Full Doplex) โดยการส่งแบบ ฮาล์ฟดูเพล็กเป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กนั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

2.3.1 การส่งข้อมูลในระบบ UART

การส่งข้อมูลในระบบ UART ซึ่งเป็นแบบอะซิงโครนัสนั้นจะเริ่มส่งข้อมูลโดยส่งบิตเริ่มต้น (Start Bit) ไปก่อนแล้วตามด้วยบิตข้อมูล (Data Bits) จำนวน 5-8 บิต โดยจะส่ง LSB ออกไปก่อน จากนั้นจะส่งบิตเช็คข้อมูล (Parity Bit) แล้วสุดท้ายก็จะส่งบิตหยุด (Stop bits) จำนวน 1, 1.5 หรือ 2 บิต

บิตของ start bit นั้นจะมีสถานะตรงข้ามกับสถานะนิ่งเฉย แต่ stop bit จะมีสถานะเดียวกับสถานะนิ่งเฉย นอกจากนี้ stop bit ยังสร้างค่าหน่วงเวลาขึ้นก่อนการส่งในครั้งถัดไปจะเริ่มขึ้นด้วย

เนื่องจากข้อมูลในไมโครโปรเซสเซอร์เป็นข้อมูลแบบขนาน ถ้าต้องการโอนย้ายข้อมูลแบบอนุกรม สามารถทำได้โดยการนำข้อมูลแบบขนานมาเก็บในรีจิสเตอร์ที่เลื่อนค่าได้ การเลื่อนข้อมูลในรีจิสเตอร์จะใช้สัญญาณนาฬิกาเลื่อนข้อมูลออกมาทีละบิต ขบวนการเปลี่ยนข้อมูลแบบขนานเป็นแบบอนุกรม แสดงได้แสดงดังรูปที่ 2.12



รูปที่ 2.12 แสดงการเปลี่ยนข้อมูลแบบขนานเป็นแบบอนุกรม

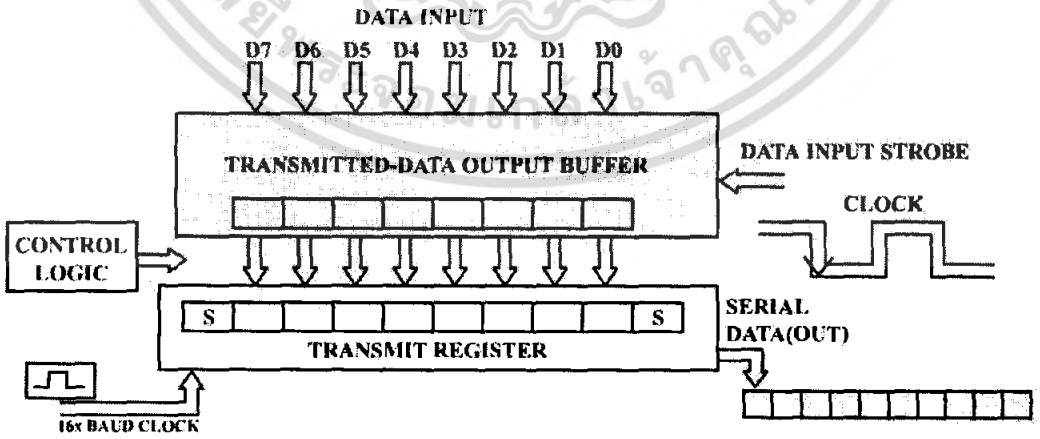
ความเร็วในการโอนย้ายข้อมูลของ UART เรียกว่าอัตราบอด (Baud rate) ซึ่งจะเป็นตัวบอกจำนวนบิตของข้อมูลที่จะส่งออกไปใน 1 วินาที ตัวอย่างเช่น การโอนย้ายข้อมูลด้วยอัตรา Baud 1200 หมายความว่า ส่งตัวอักษรได้ 120 ตัวต่อวินาที ถ้าตัวอักษรแต่ละตัวมีขนาด 10 บิต (ประกอบด้วย Start 1 บิต, ข้อมูลขนาด 8 บิต และ Stop 1 บิต) ดังแสดงในตารางที่ 2.7

ตารางที่ 2.7 แสดงบอดเรตที่ใช้ในการโอนถ่ายข้อมูลของระบบ UART

Baud Rate	Byte / Second
110	10
150	15
300	30
600	60
1200	120
2400	240
4800	480
9600	960
19200	1920
38400	3840

UART จะประกอบไปด้วยการทำงานส่วนหลัก ๆ อยู่ 4 หน่วย ได้แก่ หน่วยส่งข้อมูล, หน่วยรับข้อมูล, หน่วยสถานะ และหน่วยวงจรควบคุม

การส่งข้อมูลของ UART แสดงบล็อกไดอะแกรมให้เห็นได้ดังรูปที่ 2.13



รูปที่ 2.13 แสดงบล็อกไดอะแกรมการส่งข้อมูลของ UART

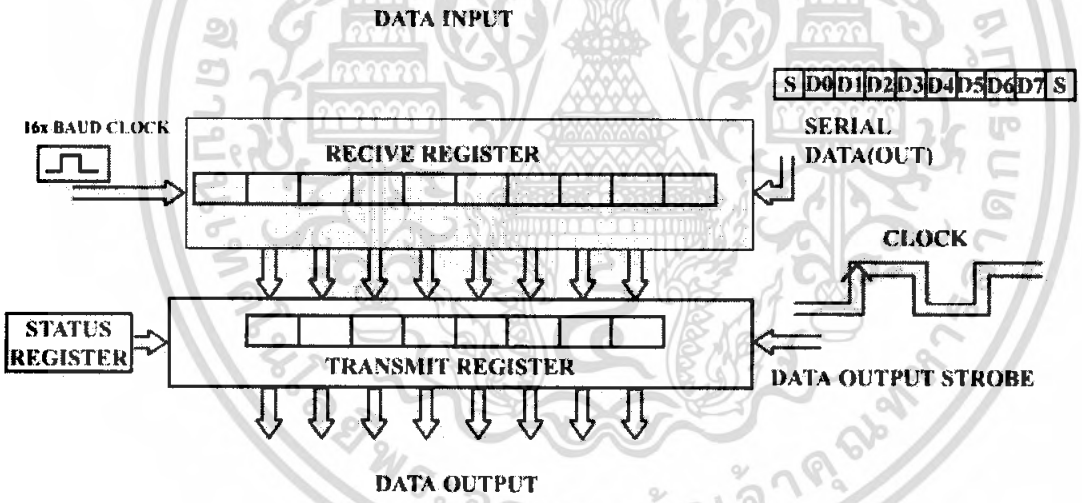
หน่วยส่งข้อมูลจะแบ่งออกได้ 2 ส่วน คือ บัฟเฟอร์ส่งข้อมูลออก และ รีจิสเตอร์ส่งข้อมูล

- บัฟเฟอร์ส่งข้อมูลออก (Transmitted-data output buffer) ทำหน้าที่เลื่อนข้อมูลออกไปทางสายส่งโดยข้อมูลจะเลื่อนแบบอนุกรม เริ่มจากบิต Start, บิตข้อมูล D0-D7 และบิต Stop โดยข้อมูล 8 บิตจะนำไปเก็บไว้ในบัฟเฟอร์ส่งข้อมูล เมื่อขอบขาลงสัญญาณควบคุมสโตรบ (Data input strobe) เข้ามา

- รีจิสเตอร์ส่งข้อมูล (Transmit register) ทำหน้าที่ส่งข้อมูลออกแบบอนุกรม เมื่อสัญญาณควบคุมสโตรบขอบขาขึ้นเข้ามา

- หน่วยควบคุมทางโลจิก (Control logic) ทำหน้าที่ค่อยควบคุมการทำงานของ UART

การรับข้อมูลของ UART แสดงบล็อคไดอะแกรมให้เห็นได้ดังรูปที่ 2.14



รูปที่ 2.14 แสดงบล็อกไดอะแกรมการรับข้อมูลของ UART

หน่วยรับข้อมูลจะแบ่งออกได้ 2 ส่วน คือรีจิสเตอร์รับข้อมูล และ บัฟเฟอร์รับข้อมูลเพื่อส่งออก

- รีจิสเตอร์รับข้อมูล (Receive register) ทำหน้าที่รับข้อมูลเข้ามาแบบอนุกรม เมื่อสัญญาณควบคุมสโตรบขอบขาขึ้นเข้ามา ข้อมูลจะเลื่อนเข้าไปในรีจิสเตอร์รับข้อมูล โดยจะเริ่มตั้งแต่มีบิต Start เข้ามา ข้อมูลทั้งหมดจะถูกนำไปเก็บในรีจิสเตอร์รับข้อมูลขนาด 8 บิต

- บัฟเฟอร์รับข้อมูลเพื่อส่งออก (Received -data output buffer) ทำหน้าที่รับข้อมูลขนาด 8 บิท เข้ามาเก็บในบัฟฟอร์รับข้อมูล เมื่อขอบขาลงสัญญาณควบคุมสโตรบ (Data output strobe) เข้ามา โดยนำข้อมูลออกมาในแบบขนานส่งให้ระบบไมโครโปรเซสเซอร์ต่อไป

- หน่วยรีจิสเตอร์สถานะ (Status register) ทำหน้าที่ค่อยตรวจสอบสถานะของ UART

2.4 โหมดการติดต่อแบบ USB (Universal Serial Bus)

2.4.1 คอนเน็คเตอร์ USB

คอนเน็คเตอร์ระบบUSBจะมีอยู่ 2 รูปแบบ ดังนี้

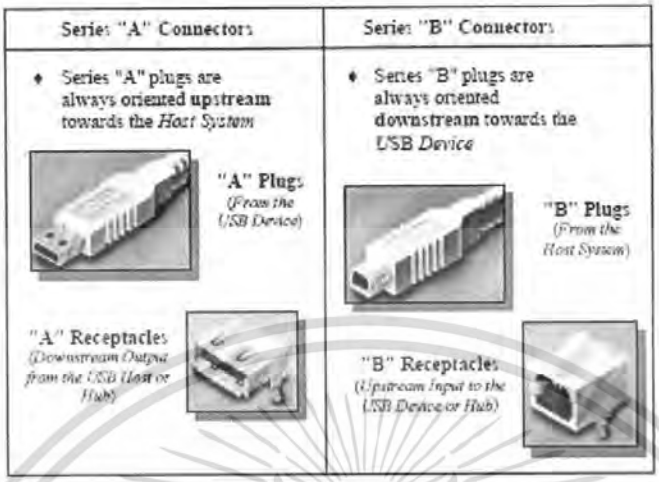
2.4.1.1 คอนเน็คเตอร์ซีรี่ส์ A

คอนเน็คเตอร์ชนิดนี้แสดงได้ดังรูปด้านซ้ายของรูปที่ 2.15 เป็นคอนเน็คเตอร์ด้านฮับที่เชื่อมต่อระหว่าง USB port ของฮับกับสายเชื่อมต่อจากอุปกรณ์ คือ คอนเน็คเตอร์ตัวเมียจะติดตั้งอยู่กับฮับ ส่วนคอนเน็คเตอร์ตัวผู้จะติดตั้งอยู่กับสายที่เชื่อมต่อจากอุปกรณ์

2.4.1.2 คอนเน็คเตอร์ซีรี่ส์ B

คอนเน็คเตอร์ชนิดนี้แสดงได้ดังรูปด้านขวาของรูปที่ 2.15 เป็นคอนเน็คเตอร์ด้านอุปกรณ์ที่เชื่อมต่อสายเข้ากับอุปกรณ์ USB คือ คอนเน็คเตอร์ตัวเมียจะติดตั้งอยู่ในอุปกรณ์ USB ส่วนคอนเน็คเตอร์ตัวผู้จะอยู่ที่สายซึ่งต่อมาจากหัว (ถ้าอุปกรณ์ใดที่มีสายต่อออกมาจากอุปกรณ์อย่างถาวรก็จะไม่มีการใช้คอนเน็คเตอร์รูปแบบนี้)

ถึงแม้จะมีการแบ่งคอนเน็คเตอร์ออกเป็นสองแบบ แต่ทั้งสองแบบนั้นก็มีสายสัญญาณเหมือนกัน โดยใช้สายสองเส้นในการส่งไฟเลี้ยงให้แก่อุปกรณ์และสายอีกสองเส้นสำหรับการรับและส่งข้อมูล แต่เนื่องจากระบบ USB นั้นออกแบบมาเพื่อให้สามารถเชื่อมต่อหรือปลดอุปกรณ์ออกจากระบบได้แม้อยู่ในระหว่างการใช้งาน ดังนั้นหน้าสัมผัสของคอนเน็คเตอร์จึงมีการออกแบบให้มีความพิเศษเล็กน้อย คือหน้าสัมผัสของสายสองเส้นที่ใช้ส่งไฟเลี้ยงจะยื่นออกมามากกว่าหน้าสัมผัสที่ใช้รับส่งข้อมูล



รูปที่ 2.15 ลักษณะของคอนเน็คเตอรUSBซีรี่ส์ A และซีรี่ส์ B

การออกแบบคอนเน็คเตอร์เช่นนี้จะทำให้อุปกรณ์ได้รับไฟเลี้ยงเข้าไปก่อนที่จะได้รับสัญญาณข้อมูลเมื่อมีการเชื่อมต่ออุปกรณ์ตัวใหม่เข้าไปในระบบ ทำให้อุปกรณ์ที่เชื่อมต่อเข้าไปใหม่สามารถทำงานได้ทันทีโดยไม่ได้รับความเสียหาย (หากอุปกรณ์ได้รับสัญญาณข้อมูลก่อนที่จะได้รับไฟเลี้ยงอาจเกิดความเสียหายแก่วงจรที่อยู่ภายในได้) ขาของคอนเน็คเตอร์แต่ละขามีหน้าที่ดังตารางที่ 2.8

ตารางที่ 2.8 หน้าที่ของขาคอนเน็คเตอร์ USB แต่ละขา

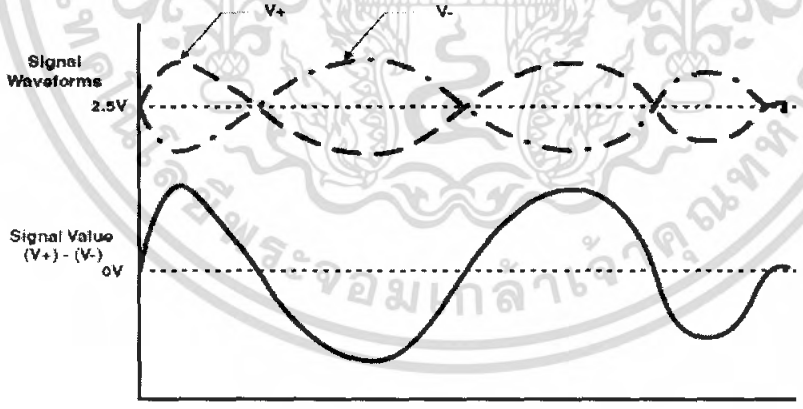
หมายเลขขา	ชื่อสัญญาณ	สีของสายสัญญาณ
1	ไฟเลี้ยง +5V	แดง
2	ข้อมูล – (D-)	ขาว
3	ข้อมูล + (D+)	เขียว
4	กราวนด์	ดำ

รูปที่ 2.16 ตำแหน่งของขาสัญญาณในหัวคอนเน็คเตอร์

2.4.2 โครงสร้างของ USB

USB เป็นมาตรฐานการรับส่งข้อมูลที่มีรูปแบบการเชื่อมต่อในระบบบัส คืออุปกรณ์ทุกๆ ตัวจะต้องส่งสัญญาณรวมกันไปในสายสัญญาณเพียงคู่เดียว ดังนั้นอุปกรณ์ทุกๆตัวที่เชื่อมต่อกับบัสจะต้องส่งข้อมูลเรียงลำดับกันไปเพื่อไม่ให้เกิดการชนกัน และเนื่องจากUSB เป็นระบบบัสที่ใช้สายส่งสัญญาณเพียงคู่เดียว และส่งสัญญาณแบบผลต่าง (Differential Signaling) ทำให้ในช่วงเวลาหนึ่งๆจะมีข้อมูลวิ่งไปได้เพียงทิศทางเดียวเท่านั้น ไม่สามารถเกิดการรับและส่งข้อมูลไปในเวลาเดียวกันได้ หรือที่เรียกกันว่าการส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ (Half Duplex)

สายสัญญาณ D+ และ D- จะมีขนาดสัญญาณเท่ากัน แต่มีเฟสต่างกัน180 องศา ซึ่งลักษณะนี้เรียกว่า Differential Signaling ซึ่ง สำหรับ USB ชนิดความเร็วสูง ค่าของข้อมูลจะเป็น 1 เมื่อ แรงดันในสาย D+ มากกว่า D- และจะเป็น 0 เมื่อแรงดันในสาย D+ น้อยกว่า D- ส่วนในโหมดความเร็วต่ำ ค่าของข้อมูลจะเป็น 1 เมื่อแรงดันนาย D+ น้อยกว่า D- และเป็น 0 เมื่อแรงดันในสาย D+ มากกว่า D- ดังรูป 2.17



รูปที่ 2.17 ค่าของสัญญาณที่ได้จากสายส่งชนิดความเร็วสูง

จังหวะการรับส่งข้อมูลของระบบบัส USB ทั้งหมดจะถูกควบคุมโดยโฮสท์ (Host) การรับส่งข้อมูลจะถูกกำหนดเป็นเฟรมโดยทุกๆ 1 มิลลิวินาที จะเกิดการรับส่งข้อมูลขึ้นหนึ่งเฟรม ในแต่ละเฟรมจะแบ่งย่อยออกเป็นแพ็คเกต (Packet) โดยเริ่มต้นแต่ละเฟรมด้วยแพ็คเกต SOF (Start

Of Frame) แล้วตามด้วยแพ็คเกตข้อมูลที่จะรับหรือส่งของอุปกรณ์แต่ละตัวเรียงต่อกันไปเรื่อยๆ ดังแสดงในรูปที่ 2.18

รูปที่ 2.18 เฟรมของข้อมูลในระบบUSB

แต่เนื่องจากแต่ละเฟรมข้อมูลจะต้องรับส่งให้เสร็จภายใน 1 มิลลิวินาที นั่นหมายความว่าข้อมูลของอุปกรณ์ทุกตัวที่เชื่อมต่อกับบัสจะต้องถูกกำหนดขนาดไม่ให้ใหญ่เกินกว่าที่จะสามารถส่งได้ภายใน 1 มิลลิวินาที และเล็กพอที่จะทำให้อุปกรณ์ทุกๆตัวสามารถใช้งานบัสไปพร้อมๆกันได้ ดังนั้นในระบบบัส USB จึงจำเป็นต้องอาศัยซอฟแวร์ (Software) เข้ามาคอยจัดการในเรื่องนี้ และต้องอาศัยฮาร์ดแวร์ (Hardware) ที่จะคอยกระจายการส่งและรวบรวมการรับข้อมูลจากอุปกรณ์ทุกตัวในระบบด้วย

2.5 ไอซีที่ใช้ในการเชื่อมต่อกับระบบ USB

ในการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ USB นั้น โดยที่ให้ไมโครคอนโทรลเลอร์ทำหน้าที่เป็นโฮสท์นั้น จำเป็นต้องมีICเข้ามาใช้เป็นตัวกลางของการติดต่อ เนื่องจากการสื่อสารระหว่างโฮสท์กับอุปกรณ์ USB นั้นมีความซับซ้อนมาก ดังนั้นจึงต้องมีการนำไอซีมาช่วยในการติดต่อซึ่งในโครงงานนี้ได้ทดลองใช้ไอซีสองเบอร์ คือ MAX3421E และ VNC1L-1A

2.5.1 ไอซีเบอร์ MAX 3421E

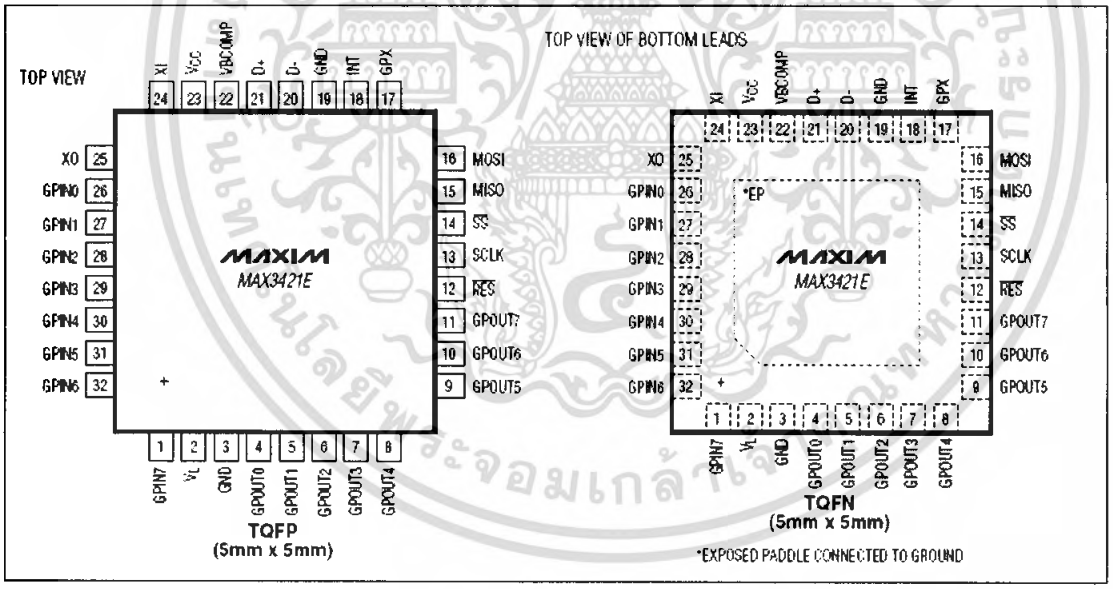
2.5.1.1 ลักษณะของ MAX 3421E

MAX3421E เป็น IC ที่ทำหน้าที่เป็น host/peripheral ในระบบ USB รองรับทั้ง full-speed และ low-speed สามารถใช้งานร่วมกับ microcontroller หรืออื่นๆเช่น DSP ASIC ควบคุมผ่านทาง SPI interface ลักษณะตัวถังของ MAX3421E จะเป็นแบบ surface mount

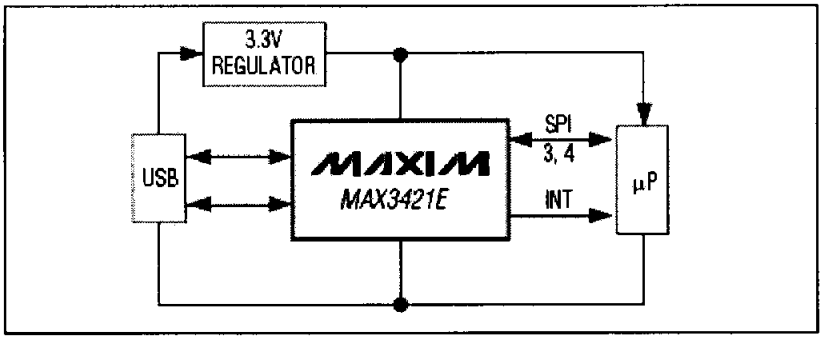
32 ขา มีทั้งแบบ TQFP และ TQFN ในส่วนของการประยุกต์ใช้งานก็สามารถนำมาใช้ได้อย่างหลากหลาย เช่น Instrumentation,MP3 player, PDAs หรืออื่นๆอีกมากมาย

สำหรับขาของไอซี VNC1L ที่ใช้ในโครงงานนี้ มีดังนี้

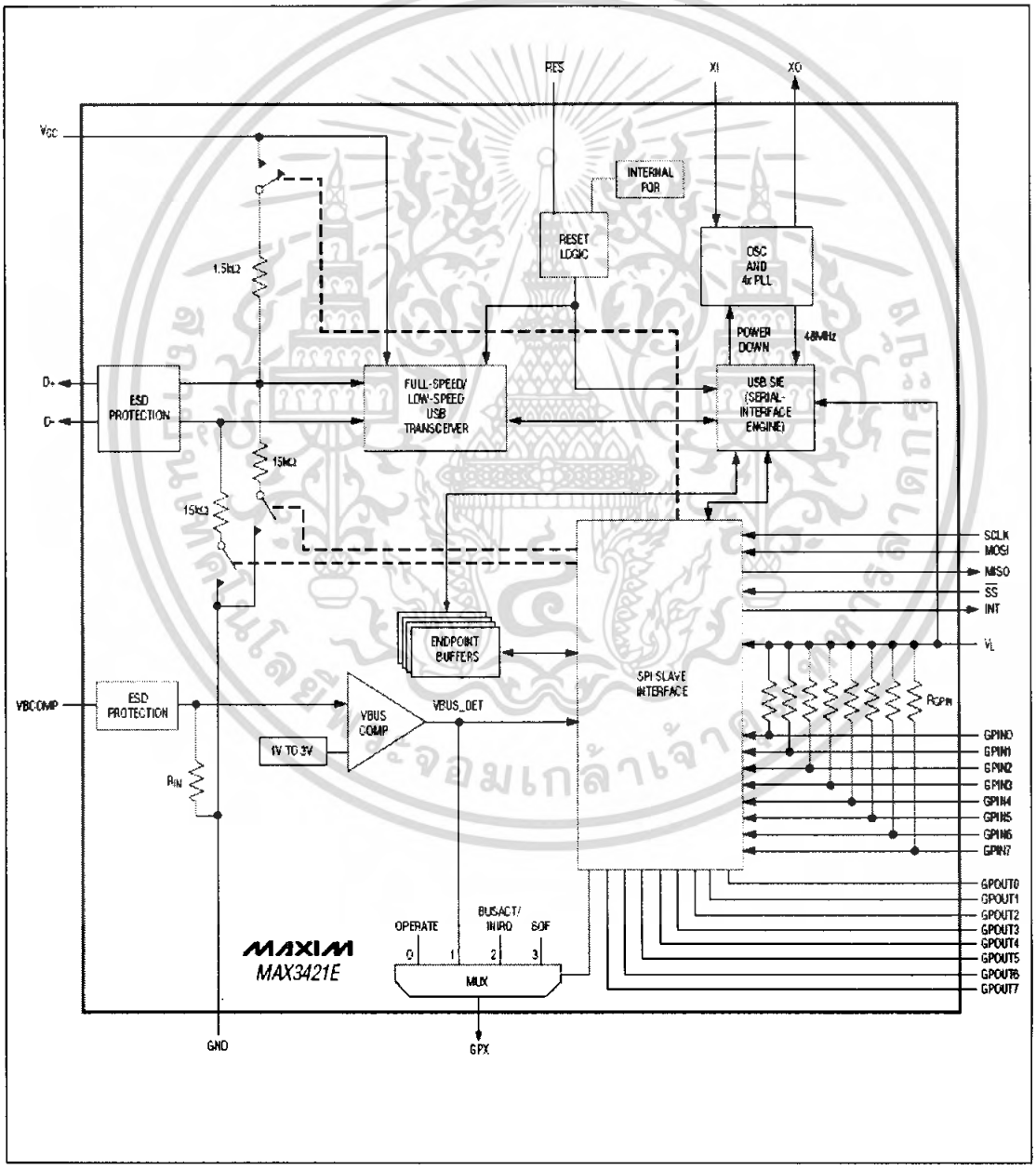
- SCK ทำหน้าที่ป้อนสัญญาณนาฬิกาจาก microcontroller ไปยัง MAX3421E
- SS ทำหน้าที่ set ค่าเป็น LOW ให้ IC ทำงาน
- MOSI ทำหน้าที่ส่งสัญญาณจาก SPI master ไปยัง SPI slave
- MISO ทำหน้าที่ส่งสัญญาณจาก SPI slave ไปยัง SPI master
- D+, D- ทำหน้าที่รับ/ส่งสัญญาณทางด้าน USB



รูปที่ 2.19 แพ็คเกจและขาต่างๆของ MAX 3421E



รูปที่ 2.20 การใช้งาน MAX 3421E อย่างง่าย



รูปที่ 2.21 โครงสร้างภายในของ MAX 3421E

2.5.1.2 การใช้งาน MAX 3421E

การใช้ microcontroller ติดต่อกับ MAX3421E นั้น จะใช้การป้อนคำสั่งผ่าน register ภายในตัว IC MAX3421E ที่ทำหน้าที่แตกต่างกันออกไป ซึ่งสามารถดูได้จากตารางที่ 2.9 ก่อนที่จะทำการส่งชุดคำสั่งให้ MAX3421E จะต้องทำให้ขา SS เป็นสถานะ LOW ก่อน แล้วจึงส่ง command byte ดังรูปแบบของ packet ในรูปที่ 2.22 ออกไป โดยบิตที่ 1-7 จะทำหน้าที่ระบุตำแหน่งของ register bit ที่ 1 จะทำหน้าที่ระบุว่าเป็นการอ่านหรือเขียน ส่วน ACKSTAT จะใช้เฉพาะใน peripheral mode เท่านั้น

ตารางที่ 2.9 รีจิสเตอร์ต่างๆที่ใช้ใน MAX 3421E

Reg	Name	b7	b6	b5	b4	b3	b2	b1	b0	acc
R0	—	0	0	0	0	0	0	0	0	—
R1	RCVFIFO	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R2	SNDFIFO	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R3	—	0	0	0	0	0	0	0	0	—
R4	SUDFIFO	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R5	—	0	0	0	0	0	0	0	0	RSC
R6	RCVBC	0	b6	b5	b4	b3	b2	b1	b0	RSC
R7	SNDBC	0	b6	b5	b4	b3	b2	b1	b0	—
R8	—	0	0	0	0	0	0	0	0	—
R9	—	0	0	0	0	0	0	0	0	—
R10	—	0	0	0	0	0	0	0	0	—
R11	—	0	0	0	0	0	0	0	0	—
R12	—	0	0	0	0	0	0	0	0	RSC
R13	USBIRQ	0	VBUSIRQ	NOVBUSIRQ	0	0	0	0	OSCOKIRQ	RC
R14	USBIEN	0	VBUSIE	NOVBUSIE	0	0	0	0	OSCOKIE	RSC
R15	USBCTL	0	0	CHIPRES	PWRDOWN	0	0	0	0	RSC
R16	CPUCTL	PULSEWID1	PULSEWID0	0	0	0	0	0	IE	RSC
R17	PINCTL	0	0	0	FDUPSPI	INTLEVEL	POSINT	GPXB	GPXA	RSC
R18	REVISION	b7	b6	b5	b4	b3	b2	b1	b0	R
R19	—	0	0	0	0	0	0	0	0	—
R20	IOPINS1	GPIN3	GPIN2	GPIN1	GPIN0	GPOUT3	GPOUT2	GPOUT1	GPOUT0	RSC
R21	IOPINS2	GPIN7	GPIN6	GPIN5	GPIN4	GPOUT7	GPOUT6	GPOUT5	GPOUT4	RSC
R22	GPINIRQ	GPINIRQ7	GPINIRQ6	GPINIRQ5	GPINIRQ4	GPINIRQ3	GPINIRQ2	GPINIRQ1	GPINIRQ0	RC
R23	GPINIEN	GPINIEN7	GPINIEN6	GPINIEN5	GPINIEN4	GPINIEN3	GPINIEN2	GPINIEN1	GPINIEN0	RSC
R24	GPINPOL	GPINPOL7	GPINPOL6	GPINPOL5	GPINPOL4	GPINPOL3	GPINPOL2	GPINPOL1	GPINPOL0	RSC
R25	HIRQ	HXFRDNIRQ	FRAMEIRQ	CONDETIRQ	SUSDNIRQ	SNDBAVIRQ	RCVDAVIRQ	RWUIRQ	BUSEVENTIRQ	RC
R26	HIEN	HXFRDNIE	FRAMEIE	CONDETIE	SUSDNIE	SNDBAVIE	RCVDAVIE	RWUIE	BUSEVENTIE	RSC
R27	MODE	DPPULLDN	DMPULLDN	DELAYISO	SEPIRQ	SOFKAENAB	HUBPRE	LOWSPEED	HOST	RSC
R28	PERADDR	0	b6	b5	b4	b3	b2	b1	b0	RSC
R29	HCTL	SNDTOG1	SNDTOG0	RCVTOG1	RCVTOG0	SIGRSM	SAMPLEBUS	FRMRST	BUSRST	LS
R30	HXFR	HS	ISO	OUTNIN	SETUP	EP3	EP2	EP1	EP0	LS
R31	HRSL	JSTATUS	KSTATUS	SNDTOGRD	RCVTOGRD	HRSLT3	HRSLT2	HRSLT1	HRSLT0	R

b7	b6	b5	b4	b3	b2	b1	b0
Reg4	Reg3	Reg2	Reg1	Reg0	0	DIR 1=wr 0=rd	ACKSTAT

รูปที่ 2.22 รูปแบบของรีจิสเตอร์บิต

ในส่วนของ response ที่ใช้บ่งบอกถึงสถานะของ IC เป็นดังตารางที่ 2.10

ตารางที่ 2.10 Response ของ MAX 3421E

HSRLT	Label	Meaning
0x00	hrSUCCESS	Successful Transfer
0x01	hrBUSY	SIE is busy, transfer pending
0x02	hrBADREQ	Bad value in HXFR reg
0x03	hrUNDEF	(reserved)
0x04	hrNAK	Peripheral returned NAK
0x05	hrSTALL	Perpheral returned STALL
0x06	hrTOGERR	Toggle error/ISO over-underrun
0x07	hrWRONGPID	Received the wrong PID
0x08	hrBADBC	Bad byte count
0x09	hrPIDERR	Receive PID is corrupted
0x0A	hrPKTERR	Packet error (stuff, EOP)
0x0B	hrCRCERR	CRC error
0x0C	hrKERR	K-state instead of response
0x0D	hrJERR	J-state instead of response
0x0E	hrTIMEOUT	Device did not respond in time
0x0F	hrBABBLE	Device talked too long

2.5.2 ไอซีเบอร์ VNC1L-1A

2.5.2.1 ลักษณะและขาที่ใช้งานของ VNC1L-1A

ไอซีเบอร์ VNC1L นี้มีลักษณะเป็นแพ็คเกจแบบเซอร์เฟซ 48 ขา ชนิด LQFP ใช้ไฟเลี้ยง 3.3 V มีพอร์ท USB 2.0 สองพอร์ท สามารถทำงานได้ทั้งแบบ low speed และ full speed นอกจากนี้ ยังสามารถเชื่อมต่อแบบ UART, SPI, PARALLEL ได้อีกด้วย มีหน่วยความจำภายในชนิด SRAM ขนาด 4kB

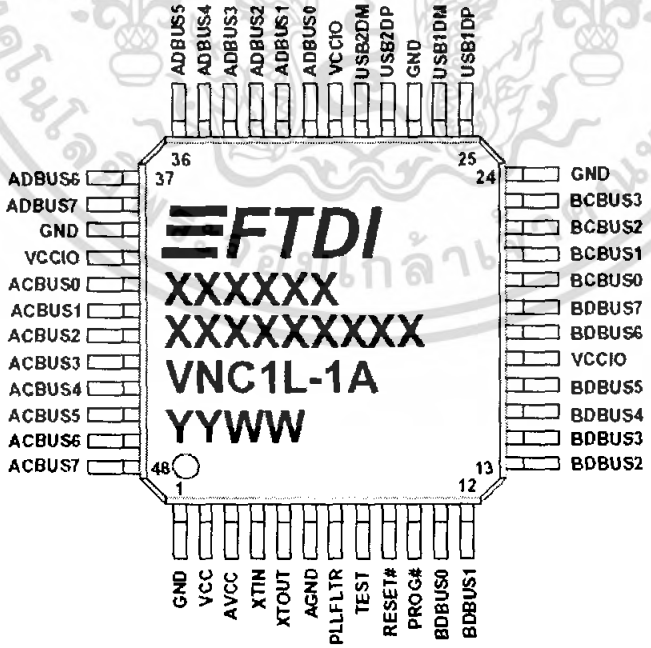
สำหรับขาของไอซี VNC1L ที่ใช้ในโครงงานนี้ มีดังนี้

- USB1DP และ USB1DM ใช้เป็นขาเชื่อมต่อกับอุปกรณ์ USB ที่ขา D+ และ D- ตามลำดับ
- USB2DP และ USB2DM ใช้งานเช่นเดียวกับ USB1DP และ USB1DM แต่ใช้เชื่อมต่อกับอุปกรณ์ USB อีกฝั่งหนึ่ง

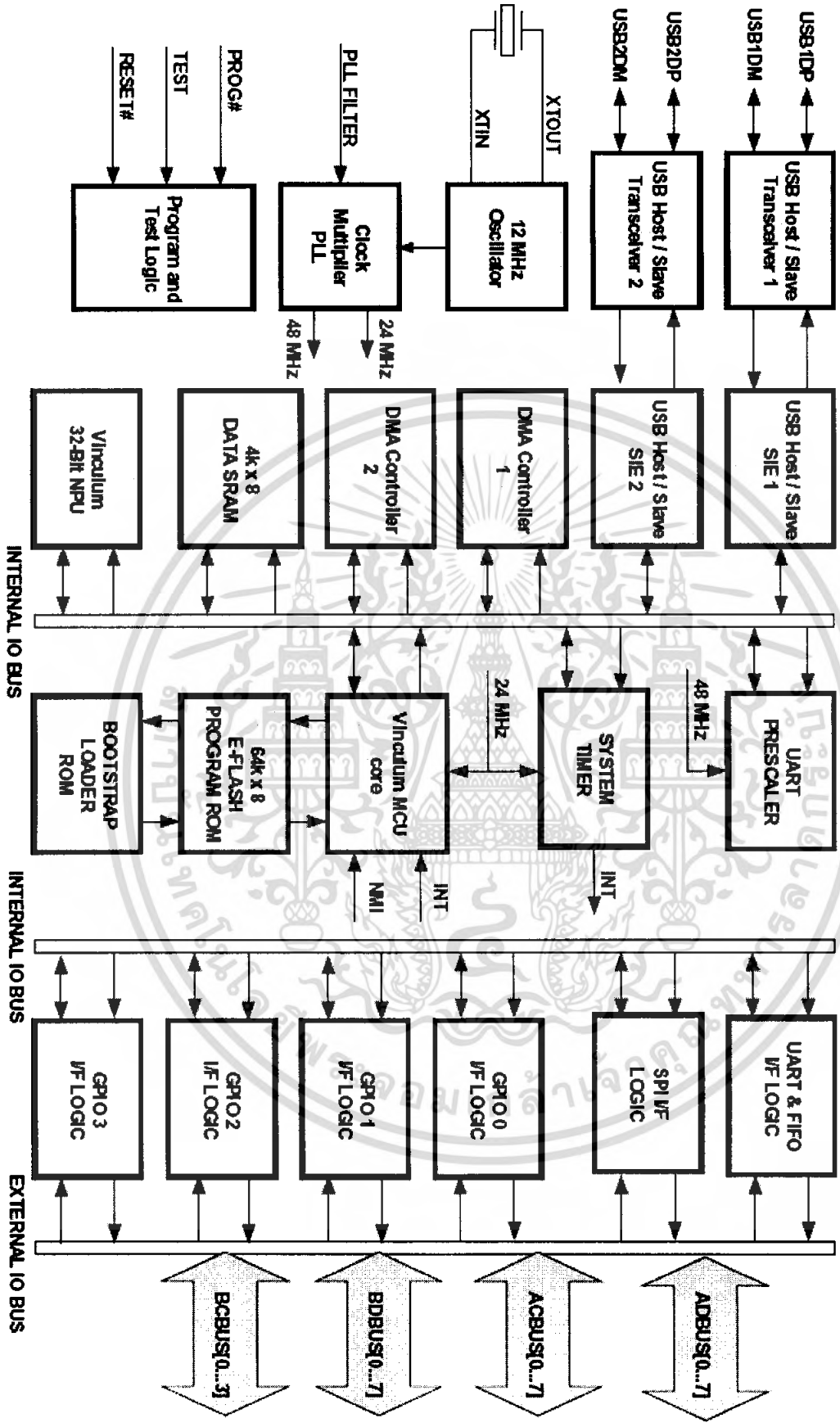
เนื่องจากในโครงงานนี้ใช้การเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับไอซี VNC1L เป็นแบบ UART ดังนั้น ขาที่ใช้ตรงส่วนนี้มีดังนี้

- ADBUS0 ใช้เป็นขา +5V
- ADBUS1 ใช้เป็นขา Tx
- ADBUS2 ใช้เป็นขา Rx
- ADBUS3 ใช้เป็นขา GND

นอกจากนี้ก็จะเป็นขาไฟเลี้ยงและกราวนด์



รูปที่ 2.23 ลักษณะและขาต่างๆของไอซี VNC1L



รูปที่ 2.24 บล็อกไดอะแกรมของไอซี VNC1L

2.5.2.2 การใช้งานไอซี VNC1L

ในการใช้งานไอซี VNC1L นี้ จะต้องมีการโปรแกรม firmware ลงไปก่อน โดยมีขั้นตอนดังนี้

1. จ่ายไฟเลี้ยงให้แก่ไอซี และเชื่อมต่อไอซีเข้ากับคอมพิวเตอร์ผ่านทางพอร์ทอนุกรม
2. ต่อขา RESET เข้ากับไฟเลี้ยง 3.3 V และต่อขา PROG ลงกราวนด์
3. ใช้โปรแกรม VPROG_COM ในการโปรแกรม firmware ซึ่งสามารถดาวน์โหลดได้จาก www.vinculum.com
4. เมื่อโปรแกรมเสร็จแล้วให้ต่อขา PROG เข้ากับไฟเลี้ยง 3.3 V แล้วจึงนำไปใช้งาน

2.5.2.3 คำสั่งที่ใช้ในการอ่านเขียนไฟล์สำหรับ VNC1L

คำสั่งโดยทั่วไปที่ใช้สำหรับการอ่านเขียนไฟล์ของ VNC1L มีดังนี้

DIR คำสั่งนี้ใช้เรียกดูชื่อไฟล์ในไดเร็คทอรี่ปัจจุบัน โดยชื่อไฟล์ที่ได้มาจะอยู่ในรูป 8.3 (ชื่อ 8 ตัว นามสกุล 3 ตัว)

DIR file คำสั่งนี้ใช้เรียกดูรายชื่อไฟล์ หรือไดเร็คทอรี่ ตามที่ชื่อที่ระบุตรงพารามิเตอร์ file โดยชื่อที่ระบุนี้หากเป็นไฟล์ต้องกำหนดนามสกุลด้วย แต่ถ้าเป็นไดเร็คทอรี่ก็ระบุเพียงแค่ชื่อ

หากเป็นการเรียกดูไฟล์ จะมีการบอกขนาดของไฟล์ด้วย ซึ่งขนาดของไฟล์นั้นจะมีหน่วยเป็นไบต์โดยจะส่งต่อจากชื่อเป็นจำนวน 4 ไบต์ ซึ่งจะส่งไบต์ LSB ออกมาก่อน

CD file คำสั่งนี้ใช้สำหรับการเข้าไปในไดเร็คทอรี่ย่อยตามชื่อที่ระบุตรงพารามิเตอร์ file โดยจะต้องเข้าจากชั้นนอกสุดไปทีละชั้น

CD .. คำสั่งนี้ใช้สำหรับออกจากโฟลเดอร์ย่อยทีละชั้น

RD file คำสั่งนี้ใช้อ่านไฟล์ตามชื่อที่ระบุตรงพารามิเตอร์ file ออกมาทั้งหมด ก่อนจะใช้คำสั่งนี้จะต้องปิดไฟล์ใดๆที่เปิดอยู่เสียก่อน การใช้คำสั่งนี้ไม่ต้องใช้คำสั่งเปิดไฟล์เพื่ออ่าน

MKD file, MKD fie date time ทั้งสองคำสั่งนี้จะใช้สำหรับสร้างไดเร็คทอรี่ย่อยขึ้นในไดเร็คทอรี่ปัจจุบัน ซึ่งจะถูกกำหนดชื่อพารามิเตอร์ file ส่วนคำสั่ง MKD file date time จะมรพารามิเตอร์ date time เพิ่มขึ้นมา ซึ่งผู้ใช้สามารถระบุวัน เวลาให้กับไดเร็คทอรี่นี้ได้

DLF file คำสั่งนี้ใช้ลบไฟล์ตามชื่อที่ระบุในพารามิเตอร์ไฟล์ โดยไฟล์ที่จะลบนี้ต้องไม่ถูกเปิดเพื่อเขียนอยู่

OPW file คำสั่งนี้ใช้เปิดไฟล์ขึ้นมาเพื่อเขียนข้อมูลหรือสร้างไฟล์ขึ้นมาใหม่ถ้าชื่อไฟล์ที่ระบุไม่มีอยู่ในไดเร็คทอรี่ที่ทำงานอยู่ ซึ่งตำแหน่งเริ่มต้นของไฟล์จะอยู่ต่อจากตำแหน่งตอนปิดไฟล์ และเราสามารถใช้คำสั่ง SEK ส่งต่อจากคำสั่งนี้เพื่อเลื่อนตำแหน่งที่จะเขียนได้

WRF dword data คำสั่งนี้ใช้เขียนข้อมูลตามจำนวนไบต์ที่ระบุในพารามิเตอร์ dword ไปยังไฟล์ที่ถูกเปิดอยู่ เมื่อเขียนข้อมูลจนครบตามจำนวนไบต์ที่กำหนดแล้วและหากยังไม่ทำการปิดไฟล์ ผู้ใช้จะสามารถส่งคำสั่ง WRF นี้เขียนข้อมูลต่อได้อีก โดยตำแหน่งของข้อมูลที่เขียนลงไปใหม่นั้นจะอยู่ถัดจากตำแหน่งสุดท้ายของข้อมูลเดิม จึงไม่เกิดการทับกันของข้อมูล

CLF file คำสั่งนี้ใช้ปิดไฟล์ที่เปิดอยู่ โดยชื่อไฟล์ที่ระบุในพารามิเตอร์ file จะต้องตรงกับชื่อไฟล์ที่ถูกเปิดอยู่ด้วย ส่วนไฟล์ที่ถูกเปิดเพื่ออ่านไม่จำเป็นต้องมีการปิดไฟล์

OPR file หรือ OPR file date คำสั่งนี้ใช้เปิดไฟล์ขึ้นมาเพื่ออ่าน ถ้าไฟล์ที่ระบุไม่มีอยู่ในไดเร็คทอรี่จะไม่มีการสร้างไฟล์ใหม่ ส่วนคำสั่ง SEK สามารถส่งต่อจากคำสั่งนี้ได้เพื่อกำหนดตำแหน่งเริ่มต้นในการอ่านข้อมูล การเปิดไฟล์เพื่ออ่านนี้ไม่จำเป็นต้องมีการปิดไฟล์หลังจากอ่านเสร็จโดยสามารถส่งคำสั่งอื่นต่อได้เลย

RDF dword คำสั่งนี้ใช้อ่านข้อมูลจากไฟล์ที่ถูกเปิดอยู่ออกมาตามจำนวนไบต์ที่ระบุในพารามิเตอร์ dword ก่อนใช้คำสั่งนี้ต้องทำการเปิดไฟล์ด้วยคำสั่ง OPR file ก่อน ผู้ใช้สามารถใช้คำสั่ง RDF อ่านข้อมูลออกมาได้เรื่อยๆจนกว่าจะสั่งปิดไฟล์โดยข้อมูลที่อ่านแต่ละครั้งจะถูกอ่านต่อจากตำแหน่งสุดท้ายของการอ่านครั้งก่อนหน้า จะไม่ไปอ่านจากตำแหน่งเริ่มต้นนอกเสียจากจะปิดไฟล์ที่อ่านอยู่นั้นแล้วเปิดไฟล์ตัวเดิมนี้ขึ้นมาอ่านใหม่

หากระบุจำนวนไบต์ที่จะอ่านในพารามิเตอร์ dword มากกว่าขนาดของข้อมูลที่จะอ่าน ข้อมูลในส่วนที่อ่านเกินก็จะอ่านออกมาได้เป็นค่า 0 หรืออาจแฮงค์ได้ ดังนั้นในการใช้คำสั่งนี้ผู้ใช้ควรตรวจสอบขนาดของไฟล์ก่อนด้วยคำสั่ง DIR เพื่อจะได้ไม่กำหนดจำนวนไบต์ที่จะอ่านเกินขนาดของไฟล์ที่เปิดอ่าน

SEK dword คำสั่งนี้ใช้ในการกำหนดตำแหน่งของ pointer ให้ชี้ไปยังตำแหน่งของข้อมูลที่อยู่ในไฟล์ที่ผู้ใช้เปิดไว้ ซึ่งจะกำหนดตำแหน่งผ่านพารามิเตอร์ dword(32 bit) ค่าเริ่มต้นของตำแหน่ง pointer จะเริ่มต้นจากค่า 0x00000000 ซึ่งจะเป็นการชี้ไปยังตำแหน่งไบต์แรกของข้อมูลที่เปิดอยู่ การจะใช้คำสั่งนี้ต้องใช้ต่อจากคำสั่ง OPR หรือ OPW

หากใช้คำสั่ง SEK ต่อจากคำสั่ง OPR โดยยังไม่มีการปิดไฟล์ ผู้ใช้สามารถใช้คำสั่ง SEK สลับกับคำสั่ง RDF ได้ตลอด แต่หากใช้คำสั่ง SEK ต่อจากคำสั่ง OPW จะสามารถใช้คำสั่ง SEK ได้ในครั้งแรกเพียงครั้งเดียว หลังจากนั้นการเขียนข้อมูลจะไม่สามารถกำหนดตำแหน่งได้โดยจะเขียนข้อมูลลงไปต่อจากตำแหน่งสุดท้ายของการเขียนครั้งก่อนหน้า การจะใช้คำสั่ง SEK ได้นั้นต้องทำการปิดแล้วเปิดไฟล์ขึ้นมาใหม่ก่อน

REN file file คำสั่งนี้ใช้สำหรับเปลี่ยนชื่อไฟล์หรือชื่อไดเร็คทอรี่ย่อย โดยพารามิเตอร์ file ตัวแรก คือชื่อเก่าของไฟล์ ส่วนตัวที่สองคือชื่อใหม่ที่จะเปลี่ยน

FS และ FSE คำสั่งนี้ใช้สำหรับตรวจสอบพื้นที่ว่างของ flash drive ว่าเหลือเท่าไร โดยถ้าหากมีพื้นที่ว่างน้อยกว่า 4GB ก็สามารถใช้คำสั่ง FS ได้ แต่ถ้ามากกว่า 4GB ต้องใช้คำสั่ง FSE คำสั่ง FS จะส่งค่าพื้นที่ว่างออกมาเป็นเลขฐาน16 จำนวน 4 ไบต์ ส่วนคำสั่ง FSE จะส่งมาเป็นเลขฐาน16 จำนวน 6 ไบต์ ค่าของเลขฐาน16 ที่ถูกส่งออกมานั้นจะส่งไบต์ LSB ออกมาก่อน

IDD และ IDDE ทั้งสองคำสั่งนี้ใช้สำหรับแสดงข้อมูลเกี่ยวกับแฟลชไดร์ฟที่นำมาต่อ โดยคำสั่ง IDDE สามารถสนับสนุนแฟลชไดร์ฟที่มีความจุ 2 TB ได้ด้วย

DVL คำสั่งนี้ใช้สำหรับเรียกดูชื่อของ volume ของดิสก์ (disk) โดยชื่อของ volume นี้จะไม่เกิน 11 ตัวอักษร ถูกเก็บอยู่ในส่วนของ Master Boot Record (MBR) ของดิสก์

DSN คำสั่งนี้ใช้เรียกดู Serial Number ของดิสก์ โดยค่าที่ได้มาจะมีขนาด 32 บิต

DIRT file คำสั่งนี้ใช้ดูค่าวันและเวลาของไฟล์ที่ระบุในพารามิเตอร์ file ซึ่งจะส่งค่าวันเวลาออกมาจำนวน 10 ไบต์ ในรูปของเลขฐาน16 โดยจะส่ง LSB ออกมาก่อน ข้อมูล 10 ไบต์นี้จะประกอบไปด้วย 4 ไบต์แรกซึ่งเป็นค่าของวันและเวลาที่ไฟล์ถูกสร้างขึ้น ส่วน 2 ไบต์ต่อมาคือค่าของวันที่เข้ามายังไฟล์ครั้งล่าสุดและ 4 ไบต์สุดท้ายคือค่าของวันและเวลาที่ไฟล์นั้นถูกแก้ไขปรับปรุง ซึ่งค่าวันและเวลาที่อ่านออกมาได้สามารถนำมาถอดเป็นค่าวันและเวลาจริงๆโดยให้อ้างอิงแต่ละ bit field

บทที่ 3

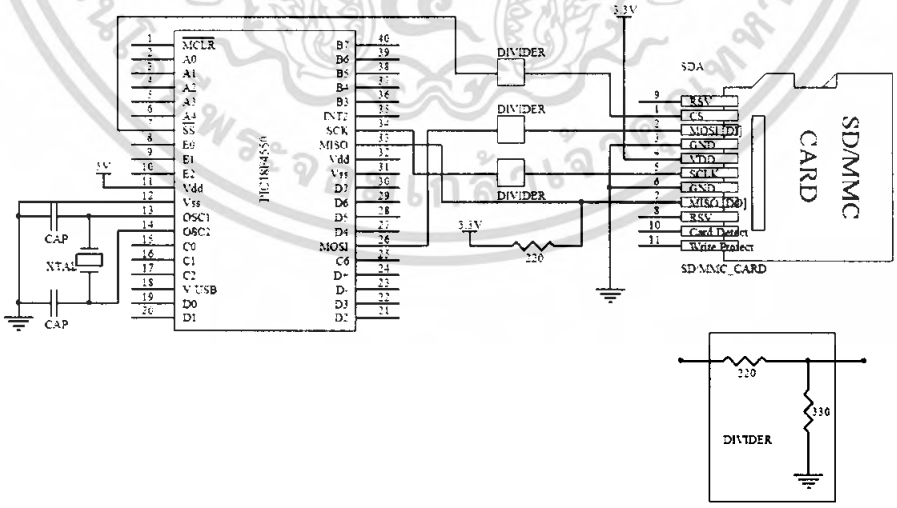
การออกแบบ

ในการออกแบบโครงงานนี้ ผู้ออกแบบได้แบ่งออกเป็น 2 ช่วง คือ ช่วงแรกเป็นการศึกษาเกี่ยวกับระบบโครงสร้างข้อมูลแบบ FAT เพื่อใช้ในการอ่านและเขียน ดังนั้นในช่วงแรกนี้ผู้ออกแบบจึงเลือกใช้ SDC/MMC มาเป็นตัวทดลองในการติดต่อกับไมโครคอนโทรลเลอร์ก่อน เนื่องจาก SDC/MMC มีรูปแบบการเชื่อมต่อที่ง่าย เหมาะสำหรับในช่วงเริ่มต้นเพื่อการอ่านและเขียนข้อมูลของโครงสร้างแบบ FAT

ในช่วงที่สองจะเป็นการศึกษาเกี่ยวกับระบบการเชื่อมต่อแบบ USB ซึ่งต้องมี IC เข้ามาช่วยเชื่อมต่อระหว่างไมโครคอนโทรเลอร์กับ USB Device

3.1 การติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SDC/MMC

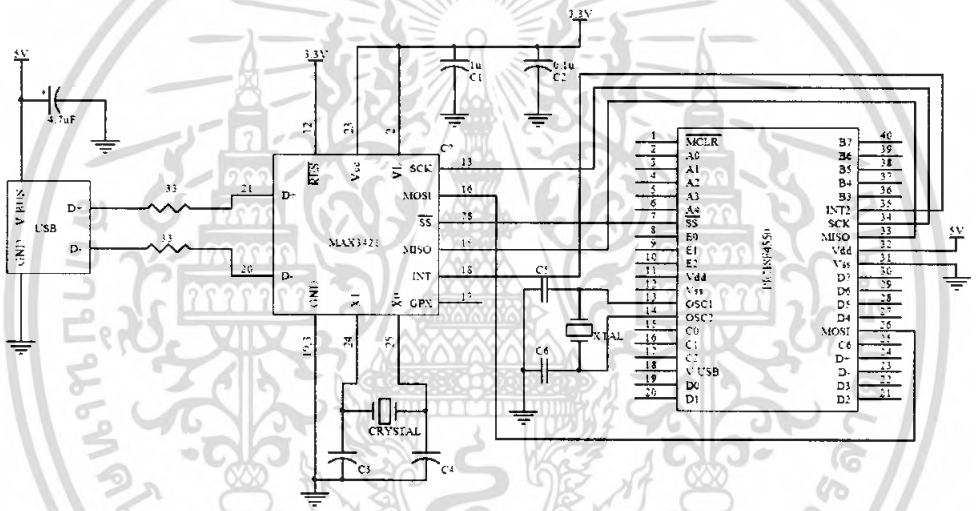
ในการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SDC/MMC นี้จะใช้ Port SPI ของไมโครคอนโทรลเลอร์ติดต่อกับการ์ดโดยตรงดังรูปที่ 3.1



รูปที่ 3.1 การติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SDC/MMC

3.2 การติดต่อระหว่าง MCU กับ USB Device โดยใช้ MAX 3421E

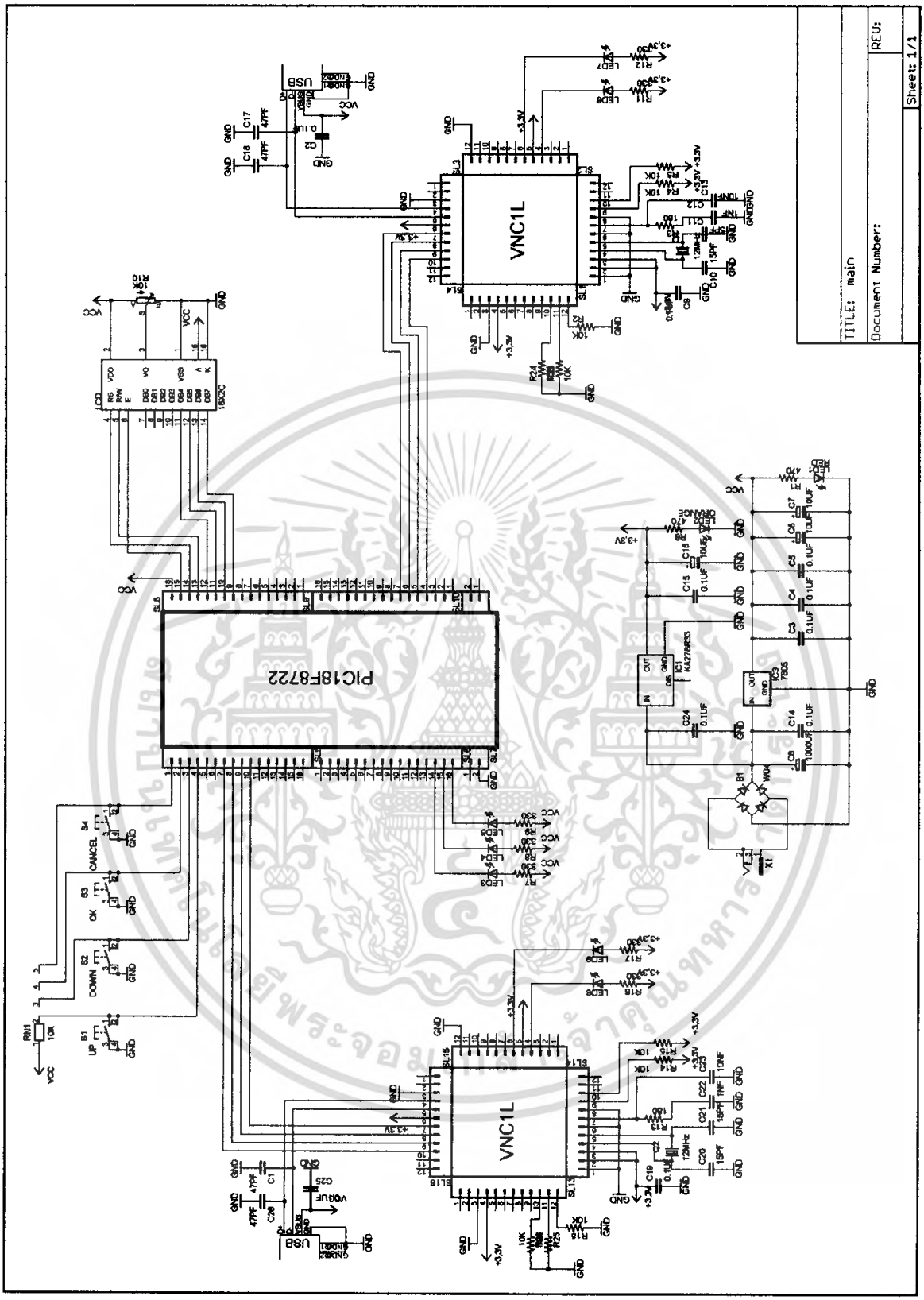
ในการติดต่อกับ USB Device นั้นก็จะใช้ Port SPI ของไมโครคอนโทรลเลอร์เช่นเดียวกันกับการติดต่อกับ SDC/MMC แต่ก็จำเป็นต้องมี IC ที่เข้ามาช่วยเป็นตัวกลางเพื่อแปลงการเชื่อมต่อระหว่างระบบ SPI กับระบบ USB ซึ่ง IC นั้นก็คือ MAX 3421E นั่นเอง ซึ่ง MAX 3421E นี้จะมีความสามารถเปลี่ยนไมโครคอนโทรลเลอร์ให้มีคุณสมบัติเป็นโฮสท์ได้ ซึ่งวงจรมีลักษณะดังรูปที่ 3.2



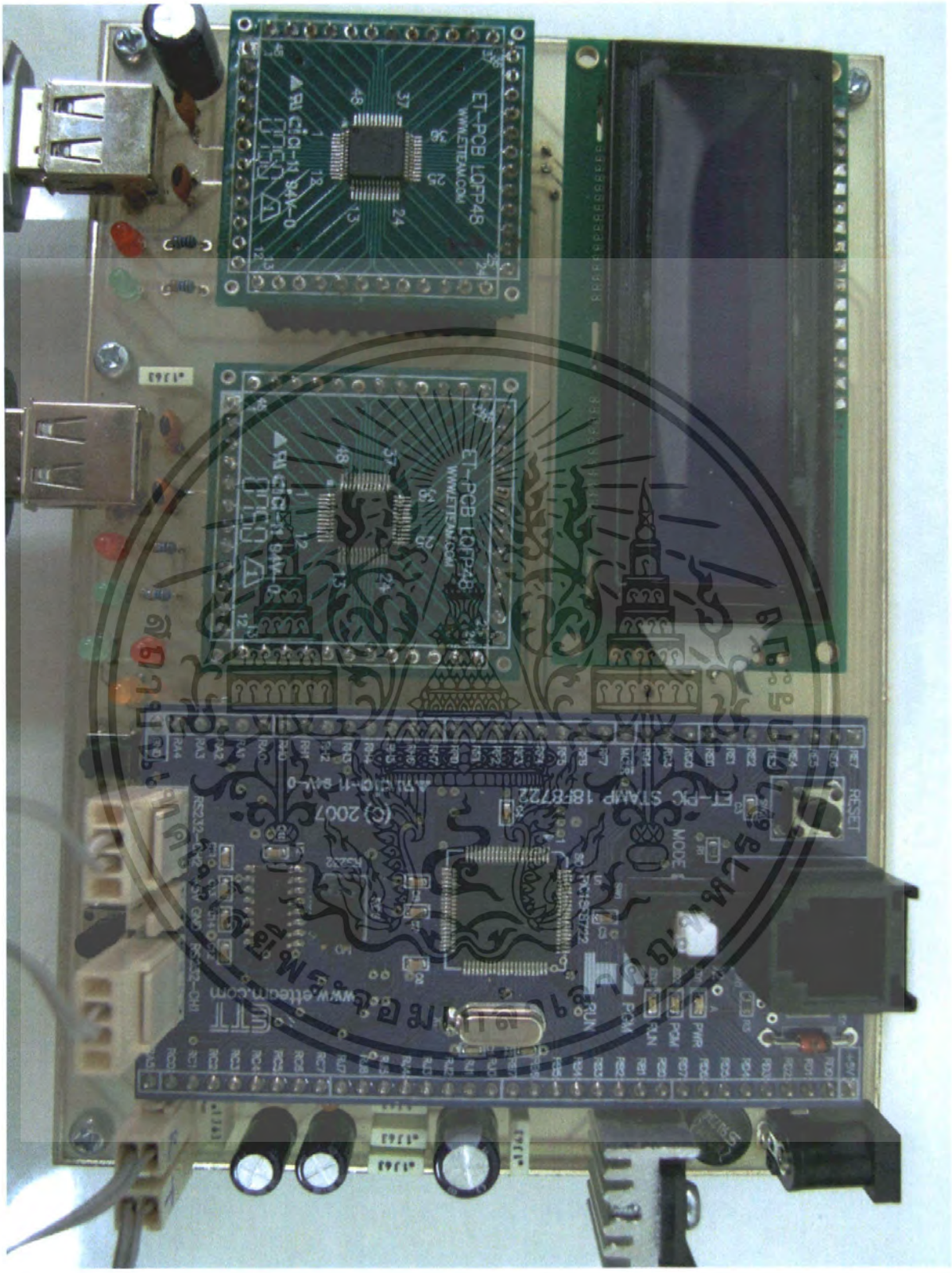
รูปที่ 3.2 การติดต่อระหว่าง MCU กับ USB Device โดยใช้ MAX3421E

3.3 การติดต่อระหว่าง MCU กับ USB Device โดยใช้ VNC1L

ในการติดต่อกับ USB Device โดยใช้ Port UART ของไมโครคอนโทรลเลอร์ซึ่งต้องมี IC ที่เข้ามาช่วยเป็นตัวกลางเพื่อแปลงการเชื่อมต่อระหว่างระบบ UART กับระบบ USB ซึ่ง IC นั้นก็คือ VNC1L นั่นเอง ซึ่ง VNC1L นี้จะมีความสามารถเปลี่ยนไมโครคอนโทรลเลอร์ให้มีคุณสมบัติเป็นโฮสท์ได้เช่นเดียวกับ MAX 3421E ซึ่งวงจรมีลักษณะดังรูปที่ 3.3



รูปที่ 3.3 การติดต่อระหว่าง MCU กับ USB Device โดยใช้ VNC1L



รูปที่ 3.4 รูปอุปกรณ์ USB Host แบบพกพา

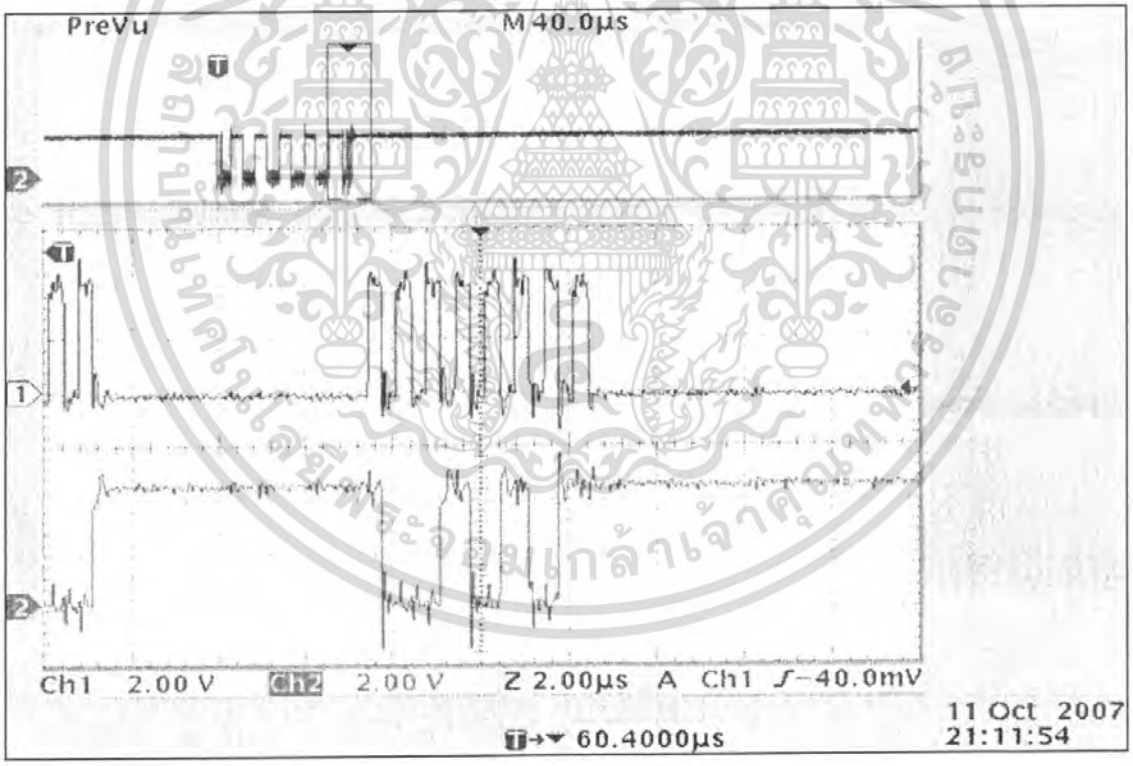
บทที่ 4

การทดลองและผลการทดลอง

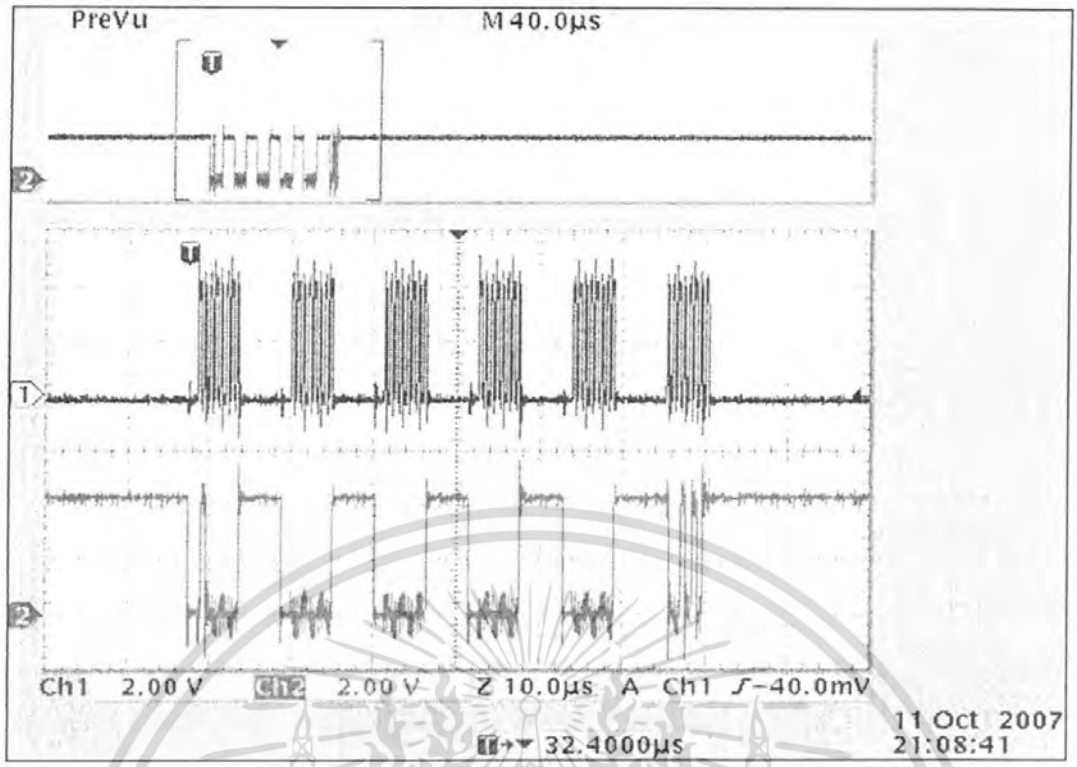
4.1 การทดลองระบบ FAT

<u>การทดลองที่ 1</u> เป็นการทดลองเรื่องระบบ FAT โดยการทดสอบกับ SD Card ทดลองโดยการส่งคำสั่งและรับการตอบกลับมาจาก SD Card เพื่อทดสอบการเข้าถึงข้อมูลในระบบ FAT

1.1 ทำการวัดขา CLK และขา MOSI ของไมโครคอนโทรลเลอร์โดยใช้ออสซิลโลสโคป

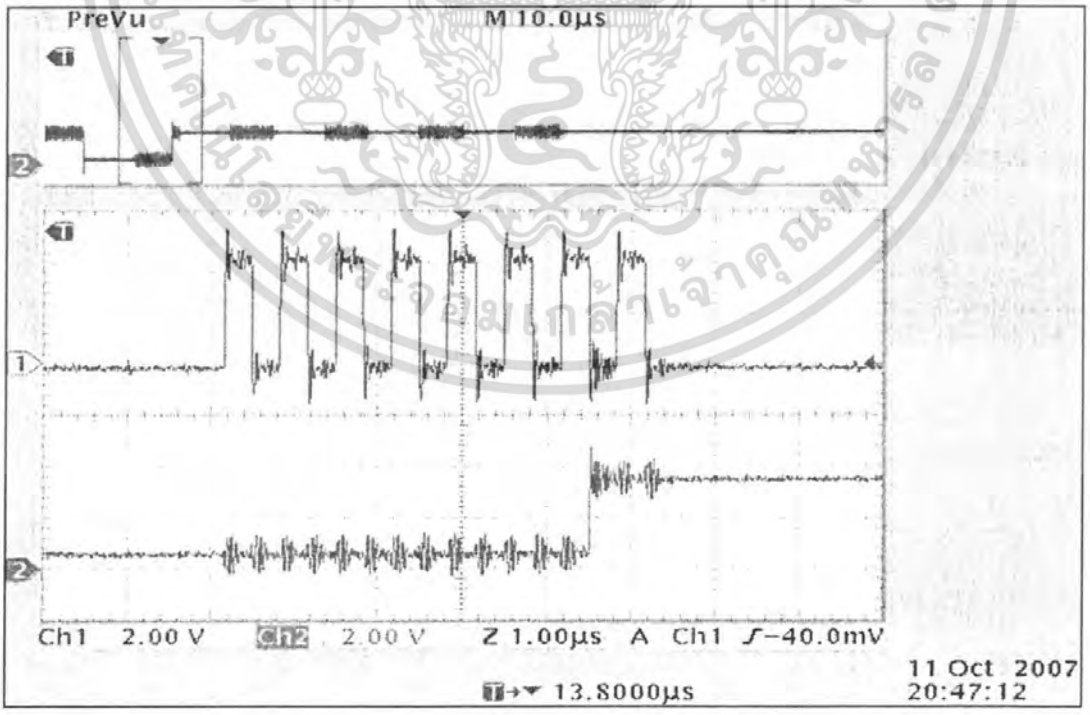


รูปที่ 4.1 ลักษณะของสัญญาณในการติดต่อแบบ SPI

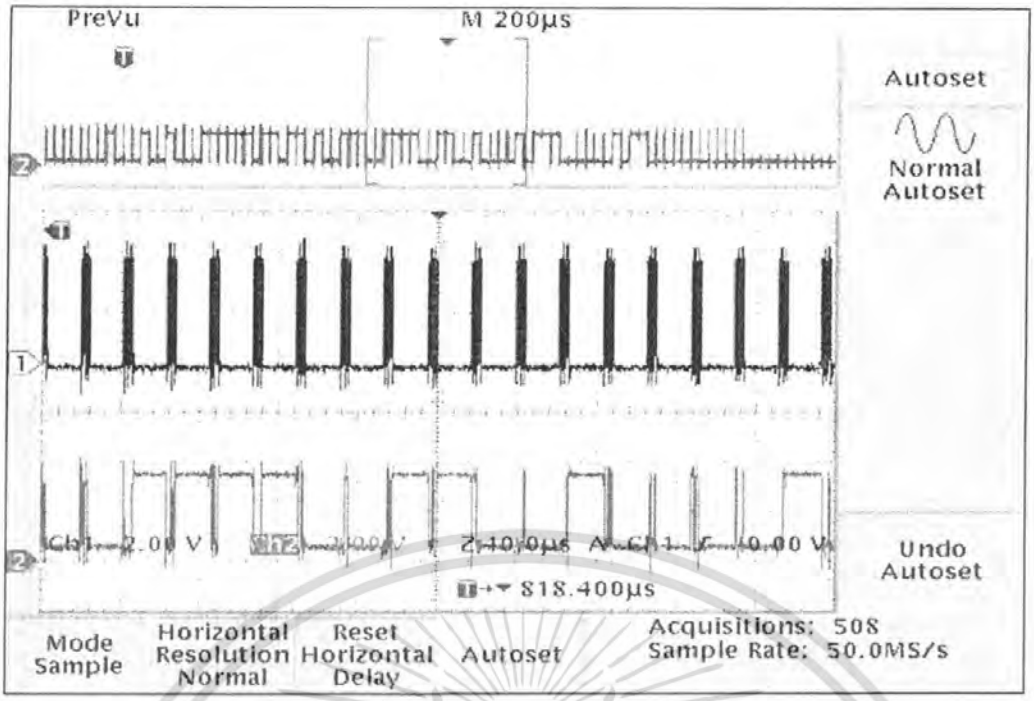


รูปที่ 4.2 สัญญาณของชุดคำสั่ง Reset ที่ส่งไปยัง SD Card

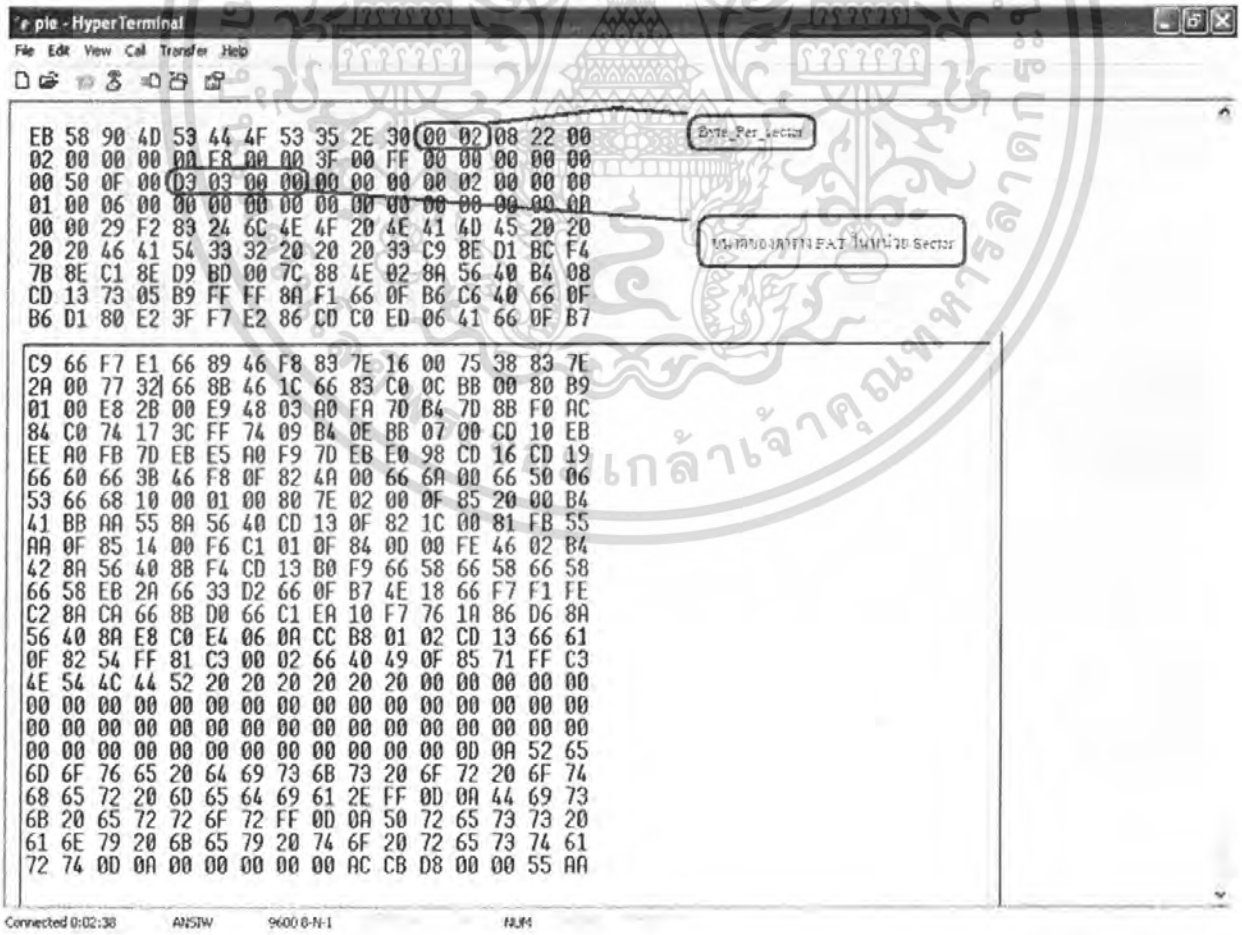
1.2 ทำการวัดขา CLK และขา MISO ของไมโครคอนโทรลเลอร์โดยใช้ออสซิลโลสโคป



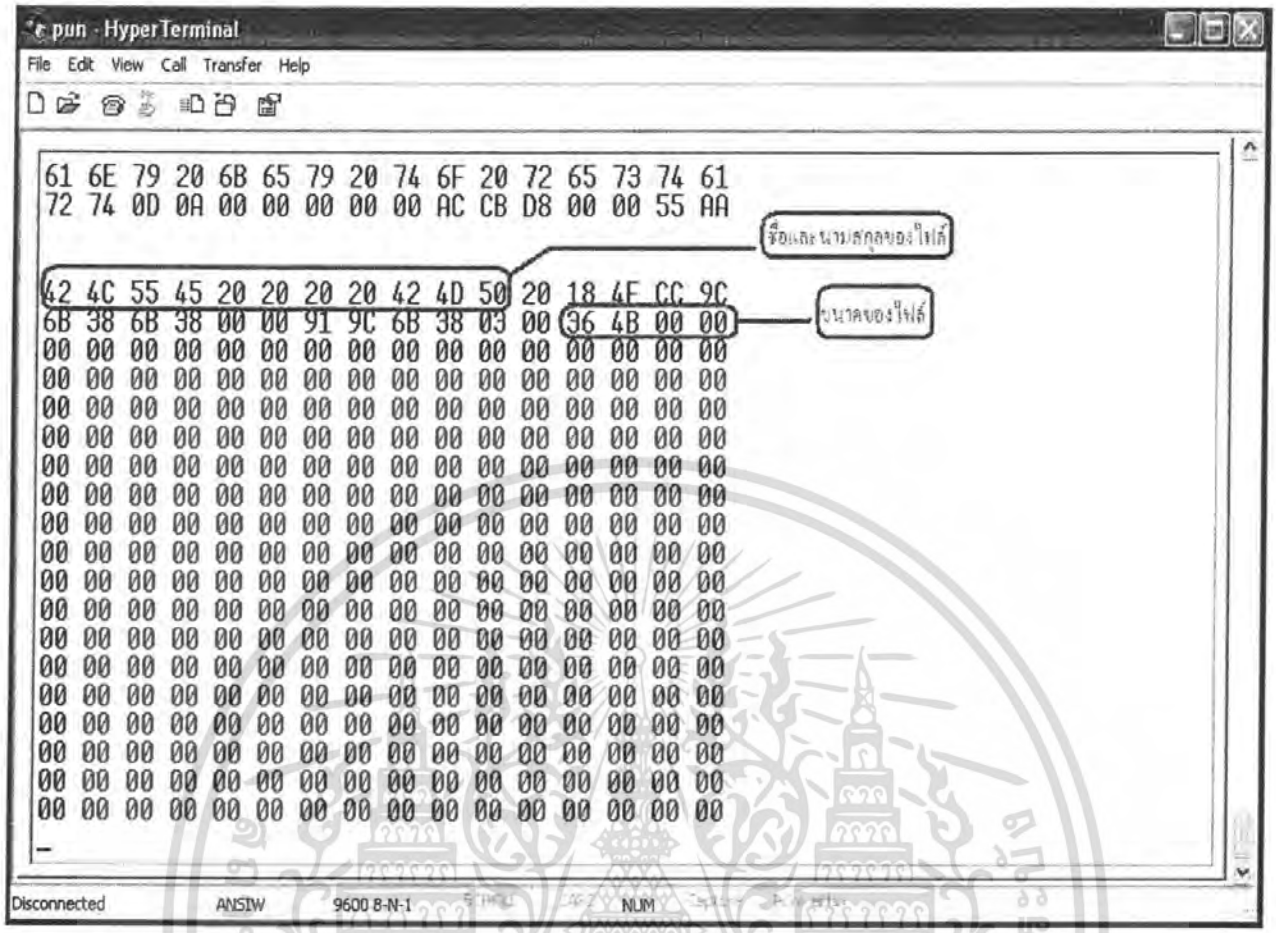
รูปที่ 4.3 Response ที่ตอบกลับมาจาก SD Card



รูปที่ 4.4 ชุดข้อมูลที่ได้รับจาก SD Card



รูปที่ 4.5 ข้อมูลใน Bios Parameter Block (BPB)

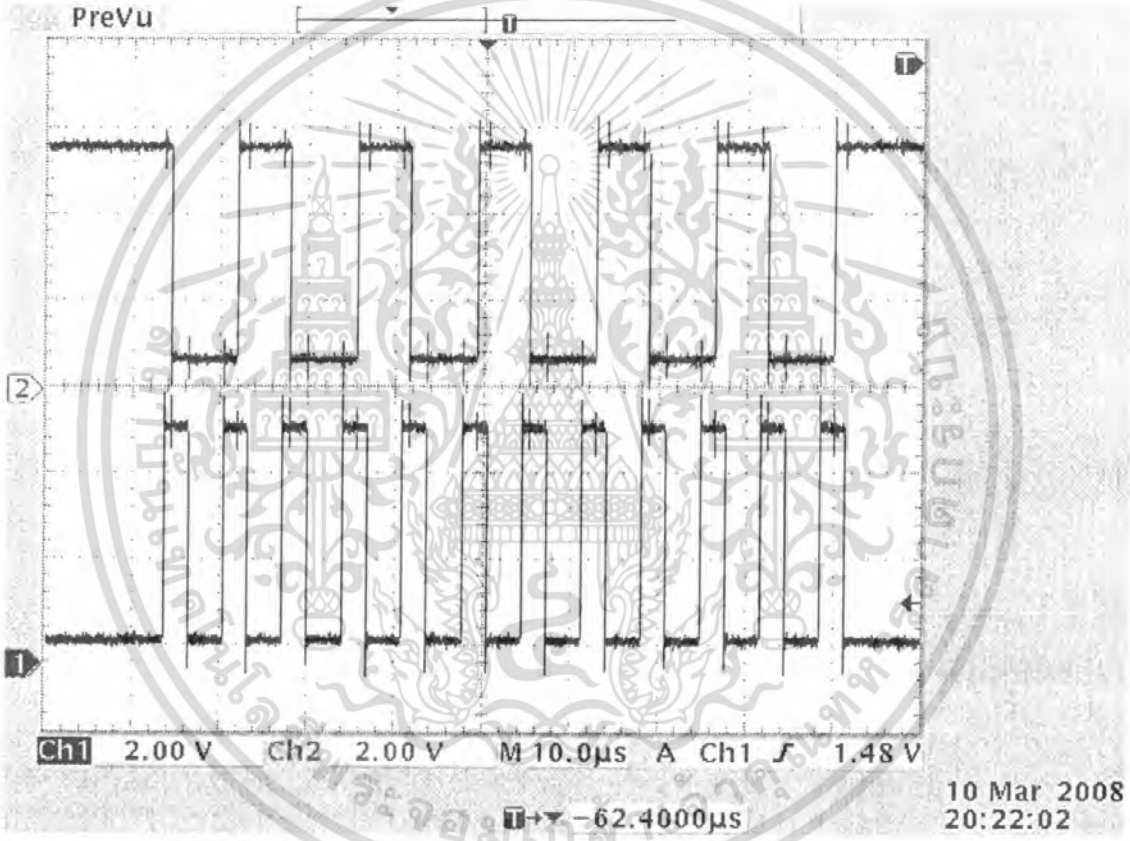


รูปที่ 4.6 ข้อมูลใน Root Directory

4.2 การทดลองติดต่อแบบ SPI ระหว่างไมโครคอนโทรลเลอร์กับ VNC1L

<u>การทดลองที่ 2</u> เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบ SPI ระหว่างไมโครคอนโทรลเลอร์กับVNC1L ซึ่งต้องใช้คล็อค (CLK) ในการรับส่ง 12 บิต

2.1 วัดสัญญาณของขา CLK และขา MOSI ของไมโครคอนโทรลเลอร์

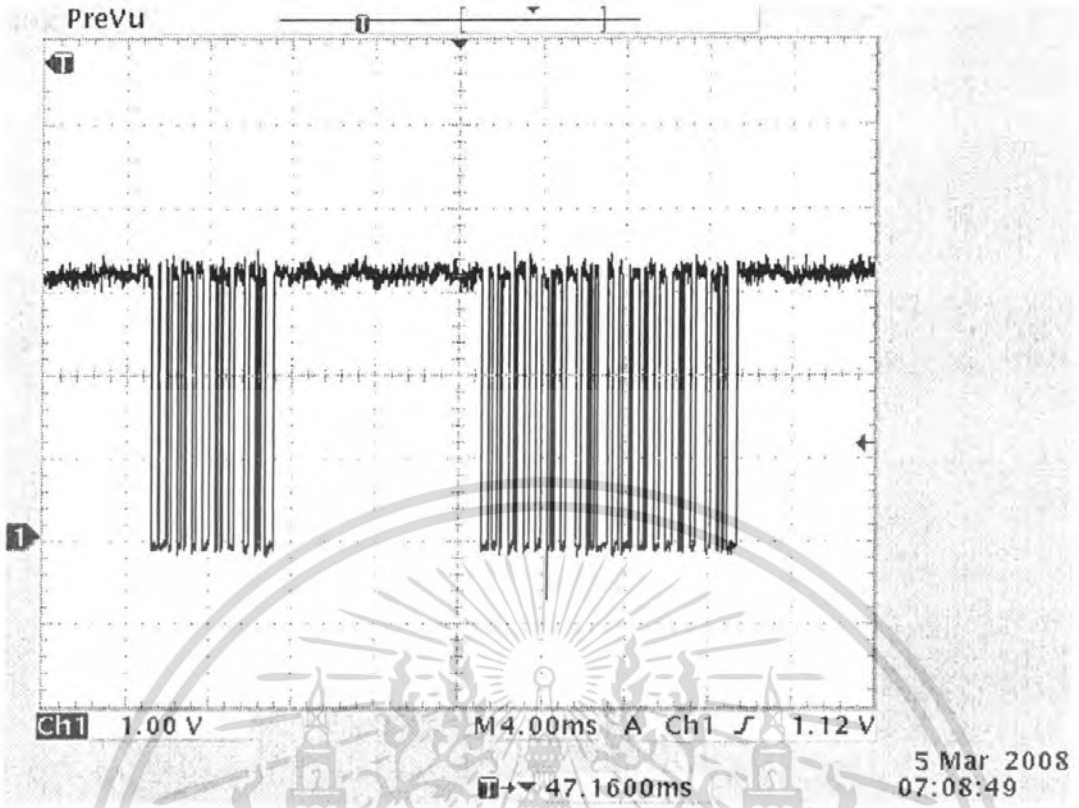


รูปที่ 4.7 สัญญาณ CLK 12 บิต และข้อมูลที่ส่งไปให้ VNC1L

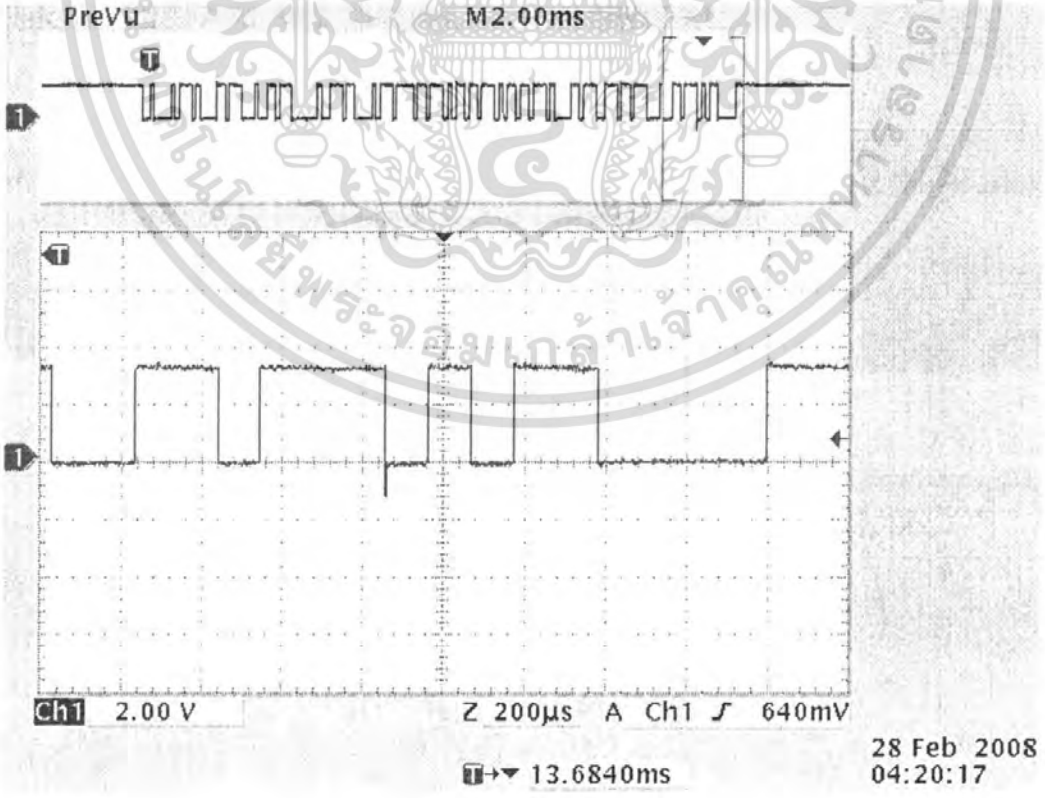
4.3 การทดลองติดต่อแบบ UART ระหว่างไมโครคอนโทรลเลอร์กับ VNC1L

<u>การทดลองที่ 3</u> เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบอนุกรม ระหว่างไมโครคอนโทรลเลอร์กับVNC1L

3.1 วัดสัญญาณที่ส่งไปให้ VNC1L และสัญญาณที่ VNC1L ส่งกลับมา



รูปที่ 4.8 สัญญาณชุดคำสั่งที่ส่งไป VNC1L

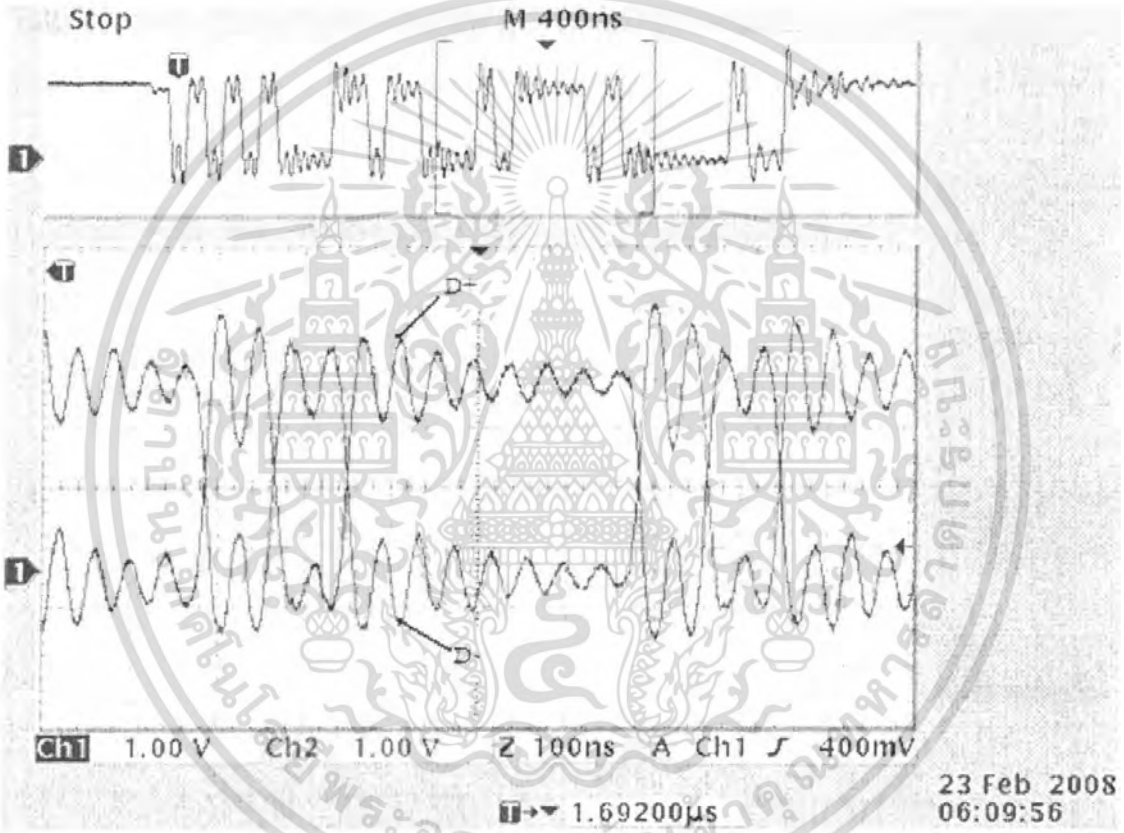


รูปที่ 4.9 สัญญาณข้อมูล 0x0D ที่ส่งไปให้ VNC1L

4.4 การทดลองวัดสัญญาณในการส่งระบบ USB

<u>**การทดลองที่ 4**</u> เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบ USB ระหว่าง VNC1L กับ USB Device

4.1 วัดสัญญาณขา D+ และขา D- ที่ VNC1L ใช้ติดต่อกับ USB Device



รูปที่ 4.10 สัญญาณขา D+ และขา D-

4.5 การทดสอบการใช้งานของอุปกรณ์ USB Host

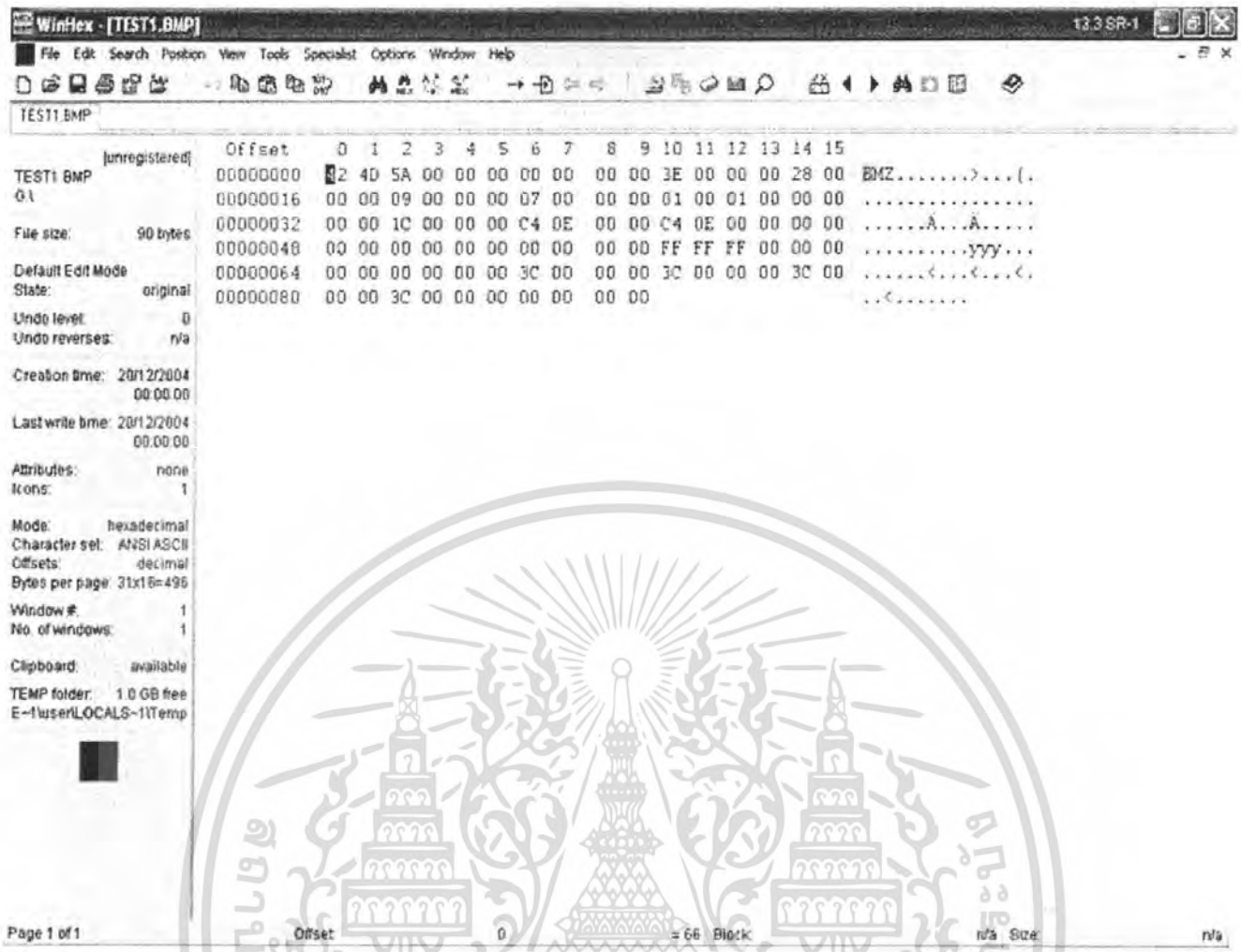
<u>การทดลองที่ 5</u> เป็นการทดสอบการคัดลอกข้อมูลระหว่าง USB Device สองตัว

5.1 ใช้โปรแกรม WINHEX เปิดดูข้อมูลในไฟล์ในอุปกรณ์ตัวต้นทาง

5.2 ทำการคัดลอกข้อมูลโดยใช้อุปกรณ์ USB Host

5.3 ใช้โปรแกรม WINHEX เปิดดูข้อมูลในไฟล์ในอุปกรณ์ตัวปลายทาง เพื่อตรวจสอบความถูกต้องของข้อมูล

รูปที่ 4.11 ข้อมูลของไฟล์ตัวต้นฉบับใน USB Device ตัวต้นทาง



รูปที่ 4.12 ข้อมูลของไฟล์ตัวที่คัดลอกใน USB Device ตัวปลายทาง

บทที่ 5

บทสรุป

5.1 สรุปผลการทดลอง

ในการทดลองที่ 1 ซึ่งทดลองการติดต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card โดยใช้ระบบการติดต่อแบบ SPI ซึ่งผลที่ได้คือ สามารถติดต่อกับ SD Card ได้ อีกทั้งยังสามารถอ่านรายละเอียดระบบ FAT ของการ์ดและอ่าน / เขียนข้อมูลลงการ์ดได้

ในการทดลองที่ 2 เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบ SPI ระหว่างไมโครคอนโทรลเลอร์กับVNC1L ซึ่งไม่สามารถติดต่อกันได้เนื่องจากต้องส่ง CLK 12 Bit ซึ่งต้องสร้างขึ้นมาเอง ซึ่ง Clock ที่สร้างขึ้นมานั้นอาจจะไม่ตรงกับ Clock ของ VNC1L

ในการทดลองที่ 3 เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบอนุกรม ระหว่างไมโครคอนโทรลเลอร์กับVNC1L ซึ่งผลที่ได้คือ สามารถรับและส่งข้อมูลกันระหว่างไมโครคอนโทรลเลอร์ กับ VNC1L ได้

ในการทดลองที่ 4 เป็นการทดลองในการวัดสัญญาณของการติดต่อแบบ USB ระหว่าง VNC1L กับ USB Device ซึ่งจะเห็นได้ว่าสัญญาณ D+ และ D- ที่วัดได้นั้นจะมีแรงดันที่เท่ากันแต่มีเฟสต่างกัน 180 องศา ซึ่งเป็นไปตามหลักการของการรับส่งข้อมูลแบบ Differential Signaling

ในการทดลองที่ 5 เป็นการทดสอบการคัดลอกข้อมูลระหว่าง USB Device สองตัว ซึ่งข้อมูลของไฟล์ที่คัดลอกมานั้นมีข้อมูลเหมือนกับข้อมูลของไฟล์ต้นฉบับ แต่เวลาซึ่งใช้ระบุว่าไฟล์ตัวคัดลอกถูกสร้างขึ้นนั้นจะไม่ได้ถูกกำหนดไว้ เนื่องจากไม่ได้มีการกำหนดเวลาไว้ในคำสั่งขณะที่สร้างไฟล์

โดยสรุป อุปกรณ์ USB Host ตัวนี้ สามารถที่จะคัดลอกข้อมูลจากทัมบ์ไดรฟ์ตัวต้นทางไปยังทัมบ์ไดรฟ์ตัวปลายทางได้โดยสมบูรณ์ อีกทั้งยังสามารถเลือกไฟล์ที่ต้องการหรือไม่ต้องที่จะคัดลอกได้อีกด้วย

5.2 ปัญหาและแนวทางแก้ไข

- เนื่องจากการติดต่อระหว่างไมโครคอนโทรลเลอร์และ VNC1L เป็นการติดต่อแบบอนุกรมซึ่งมีความเร็วต่ำ ทำให้การคัดลอกไฟล์ทำได้ช้า อาจแก้ไขโดยใช้การติดต่อที่มีความเร็วสูงกว่า หรือตั้งค่าบอดเรตให้สูงขึ้น แต่ก็จะมีโอกาสสื่อสารผิดพลาดได้

- ในการติดต่อกับ USB Device นั้นต้องใช้กระแสไฟมากเนื่องจาก USB Device บางชนิดต้องใช้กระแสในการทำงานมากพอสมควร

- สำหรับไฟล์ที่มีชื่อยาวเกิน 8 ตัวอักษร จะทำให้การคัดลอกเกินความผิดพลาดได้

- การทำงานของอุปกรณ์ USB Host จะถูกจำกัดด้วย Firmware ของ VNC1L ซึ่งการทำงานบางอย่างที่ต้องการนั้นก็ไม่มีรองรับใน Firmware

5.3 ประโยชน์ที่ได้รับ

- ได้รู้ถึงการจัดการข้อมูลในระบบ FAT ได้ในระดับหนึ่ง

- ได้ศึกษาการติดต่อในระบบต่างๆ เช่น SPI, UART, USB

- ได้ศึกษาการใช้ IC ที่ใช้ติดต่อกับ USB Device เช่น MAX3421E, VNC1L

- ได้รู้ถึงการใช้งาน SD Card

5.4 สรุปการทำงานและแนวทางการพัฒนา

การทำงานของ Hardware & Software ตัวปัจจุบันนั้นจะเริ่มด้วยการรอรับคำสั่งเพื่อดูจำนวนและชื่อของ File ใน USB Thumb drive ตัวต้นทาง โดยทำการกดปุ่ม SW1 ต่อจากนั้น LCD จะทำการแสดงชื่อ File ทีละFile โดยสามารถเลื่อนไปยัง File ต่อไปโดยการกดปุ่ม SW2

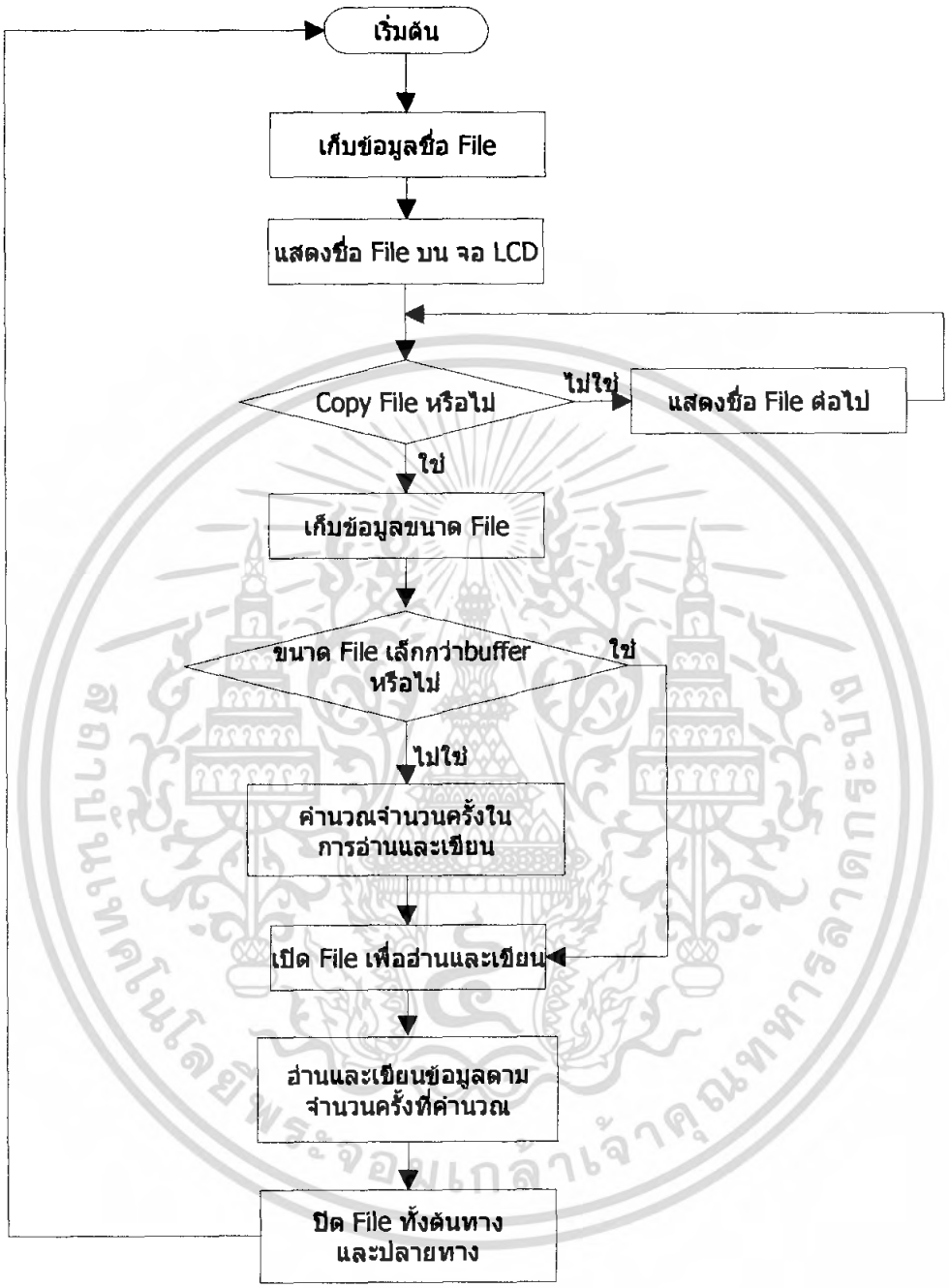
หากต้องการที่จะคัดลอก File ใด จะใช้การรับคำสั่งจาก SW3 เป็นการเริ่มการคัดลอก เมื่อเสร็จสิ้นการคัดลอก LCD จะแสดงข้อความ Complete หลังจากนั้นโปรแกรมจะเริ่มทำงานใหม่ตั้งแต่ต้น หากข้อมูลมีชื่อ File เกิน 8 ตัวอักษรจะเกิดการผิดพลาดของชื่อ File ที่คัดลอกไป โดยเกิดเครื่องหมาย ~ ขึ้นที่ชื่อ File

เนื่องด้วยระยะเวลาการพัฒนามีจำกัด Hardware & Software ตัวปัจจุบัน จึงยังมีข้อจำกัดและข้อควรปรับปรุงอยู่หลายประการด้วยกัน และมีแนวทางการพัฒนาต่อไปดังนี้

- การเพิ่ม Baud Rate ของการติดต่อแบบ UART จะทำให้การรับ-ส่งข้อมูลเร็วขึ้น
- เปลี่ยนรูปแบบการติดต่อเป็น SPI หรือ Parallel จะทำให้การรับ-ส่งข้อมูลเร็วขึ้น
- การตรวจสอบความถูกต้องของการคัดลอกโดยเปรียบเทียบขนาดของ File ที่คัดลอกไปกับต้นฉบับ
- การตรวจสอบความพร้อมของ USB Thumb drive ตัวปลายทาง โดยการตรวจสอบขนาดพื้นที่ว่างว่าพอสำหรับคัดลอก File ที่ต้องการหรือไม่ ตรวจสอบว่า USB Thumb drive ตัวปลายทางมี File ชื่อเดียวกับที่ต้องการคัดลอกอยู่หรือไม่
- การตรวจสอบว่ามี USB Thumb drive เสียบอยู่หรือไม่ ซึ่งอาจเกิดความผิดพลาดขึ้นขณะทำการคัดลอก File ได้
- เพิ่มคำสั่ง Delete File ใช้ลบ File ที่ไม่ต้องการออก เพื่อเพิ่มพื้นที่ในการรองรับข้อมูลให้มากขึ้น

ภาคผนวก

แผนผังการทำงานของโปรแกรม



```
//*****************************************************************************
//Developer    Blackswan & AntiMaew
//Institute    King Mongkut's Institute of Technology Ladkrabang
//Major        Electronics
//*****************************************************************************
//Declaration  PIN
//*****************************************************************************
#include <18F8722.h>
#define  CLOCK_SP    10000000

#fuses HS
#fuses NOLVP,NOWDT
#fuses NOPROTECT
#use delay (clock=CLOCK_SP)
#use
rs232(baud=9600,UART1,XMIT=PIN_C6,RCV=PIN_C7,DISABLE_INTS,stop=1,TIMEOUT=10,stream=vnc
_1)
#use rs232(baud=9600,UART2,XMIT=PIN_G1,RCV=PIN_G2,DISABLE_INTS,stop=1,stream=vnc_2)

#include "lcd.c"
#include "string.h"
#include "input.c"

#define resetvnc1l_1  PIN_E3
#define resetvnc1l_2  PIN_E2
#define RTS2        PIN_C5
#define CTS2        PIN_C4
#define RTS1        PIN_G0
```

```
#define CTS1        PIN_G3
#define sw1         PIN_F7
#define sw2         PIN_F6
#define sw3         PIN_F5
#define sw4         PIN_F4
#define led_red     PIN_A1
#define led_yellow  PIN_A2
#define led_green   PIN_A3
#define CTS_1       PIN_H6
#define RTS_1       PIN_H5
#define CTS_2       PIN_F0
#define RTS_2       PIN_H7

void blink_y ();
void blink_g ();
void blink_r ();
void q_file  ();
void name_file();
void calfilesize();

//*************************************************************************
//Declaration GLOBAL Variable
//*************************************************************************
unsigned char Data[2000],cname[1000];
unsigned int16 a,b,c,d,e,f,g,h,i,j,k,m,n,x,y,z,size[10];
int l;
boolean as;
```

```
int32 filesize;
//*****************************************************************************
//Interrupt Function
//*****************************************************************************
#INT_RDA
void serial_isr()
{
 Data[m]=fgetc(vnc_1);
 Data[m+1]=fgetc(vnc_1);
 m=m+2;
}

//*****************************************************************************
//Functions Prototype
//*****************************************************************************
void lcd_start(void)
{
 lcd_putc('\f');
 lcd_gotoxy(1,1);
 lcd_putc("@@@_USB HOST_@@@");
 lcd_gotoxy(1,2);
 lcd_putc("....Blackswan...");
 delay_ms(1000);
 lcd_gotoxy(1,2);
 lcd_putc("<<< PUSH SW1 >>>");
 lcd_gotoxy(1,1);
 lcd_putc(" VIEW FILES NAME");
```

```
}



/*****************************************************************************/
//############################## Main Program ##############################
/*****************************************************************************/
#zero_ram
void main(void)
{
 begin:
 disable_interrupts(GLOBAL);
 disable_interrupts(INT_RDA);
 output_low(CTS_1);
 output_low(CTS_2);
 delay_ms(10);

   lcd_init();
   lcd_start();

   i=0;
   m=0;
   output_high(led_yellow);
   output_high(led_green);
   output_high(led_red);

 while(true)
 {
```

```
    if(!input(sw1)){delay_ms(10);goto copy;}
    blink_y();
}

  copy:

  fPrintf(vnc_1,"ipa\r");
  fPrintf(vnc_1,"ipa\r");

  delay_ms(100);//*****very important in recieving 1st data bit//
          //*****Can not change//

  output_high(CTS_1);
  delay_ms(10);
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_RDA);
  fPrintf(vnc_1,"dir\r");
  output_low(CTS_1);
  delay_ms(500);

  //*************do not use but important*************************
  for(n=0;n<128;n++)
  {
   delay_ms(10);
  }

  disable_interrupts(GLOBAL);
```

```
disable_interrupts(INT_RDA);

j=0;
for(i=3;i<1000;i++)
{
  if(data[i]==0x2E){j=j+1;}
  if(data[i]==0x3E){Q_file();}//i stop count that can know how many DATA
}

name_file();

check:

lcd_putc('\f');
lcd_gotoxy(1,1);
lcd_putc(" WAIT FOR COPY  ");
lcd_gotoxy(1,2);
lcd_putc("Push SW1 to copy");
delay_ms(1000);

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  while(input(sw1)){blink_y();}
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    lcd_putc("/// Copying /// ");
```

```
    lcd_gotoxy(1,2);
    lcd_putc(" Please Wait....");

for(n=0;n<1000;n++) /***********Copy caname**************/
{
cname[n]=Data[n];
delay_us(1);
}

fPrintf(vnc_1,"ipa\r");
fPrintf(vnc_1,"ipa\r");

delay_ms(100);//*****very important in recieving 1st data bit//
        //*****Can not change//
m=0;
output_high(CTS_1);
delay_ms(10);
enable_interrupts(GLOBAL);
enable_interrupts(INT_RDA);
fPrintf(vnc_1,"dir ");

for(c=3;c<1000;c++)
{
  if(cname[c]==0x0d){goto enterfilename;}
}

enterfilename:
```

```
 for(b=3;b<c;b++)
 {
 fPrintf(vnc_1,"%c",cname[b]);
 }
 fPrintf(vnc_1,"\r");
 output_low(CTS_1);
 delay_ms(10);

 for(n=0;n<128;n++)
 {
  delay_ms(10);
 }
 disable_interrupts(GLOBAL);
 disable_interrupts(INT_RDA);

for(k=0;k<40;k++)
{
   if(data[k]==0x24){goto next;} /**** Scan for parameter k *****/
}
next:
 calfilesize(); /*************** Calculate file size ********************/

 fPrintf(vnc_1,"ipa\r");
 fPrintf(vnc_2,"ipa\r");
 delay_ms(1);

 fPrintf(vnc_1,"opr ");
```

```
 for(b=3;b<c;b++)
 {
  fPrintf(vnc_1,"%c",cname[b]);
 }
 fPrintf(vnc_1,"\r");
 fPrintf(vnc_1,"ipa\r");
 delay_ms(10);

 fPrintf(vnc_2,"opw ");
 for(b=3;b<c;b++)
 {
 fPrintf(vnc_2,"%c",cname[b]);
 }
 fPrintf(vnc_2,"\r");
 fPrintf(vnc_2,"ipa\r");
 delay_ms(10);

 output_high(CTS_1);
 delay_ms(10);
 m=0;
 enable_interrupts(GLOBAL);
 enable_interrupts(INT_RDA);

 if(filesize<1000)
{
 e=filesize;
 goto small;
```

```
}
 else if(filesize>=1000)
{
    l=filesize/1000;
    e=filesize%1000;
    goto large;
}

 large: /*************** Filesize > 1000 ******************/

  for(f=0;f<l;f++)
{
 m=0;
 fPrintf(vnc_1,"rdf 1000"); /*********Read Data**********/
 fPrintf(vnc_1,"\r");
 delay_ms(10);

 fPrintf(vnc_2,"wrf 1000"); /*********Write Data*********/
 fPrintf(vnc_2,"\r");
 delay_ms(10);

 output_low(CTS_1);  /***********/
 delay_ms(1200);   /****** delay for get data ******/

 for(n=0;n<m-6;n++)
 {
 fPrintf(vnc_2,"%c",Data[n]);
```

```
 delay_ms(2);
 }
}

small: /*************** Filesize < 1000 ********************/

output_high(CTS_1);
delay_ms(10);
m=0;

fPrintf(vnc_1,"rdf "); /*********Read Data**********/
fPrintf(vnc_1,"%ld",e);
fPrintf(vnc_1,"\r");
delay_ms(10);

fPrintf(vnc_2,"wrf "); /*********Write Data*********/
fPrintf(vnc_2,"%ld",e);
fPrintf(vnc_2,"\r");
delay_ms(10);

output_low(CTS_1); /***********/
delay_ms(1100);   /****** delay for get data ******/

for(n=0;n<m-6;n++)
{
 fPrintf(vnc_2,"%c",Data[n]);
 delay_ms(2);
```

```
 }
   blink_g();



disable_interrupts(GLOBAL);
disable_interrupts(INT_RDA);


fPrintf(vnc_2,"clf "); /************ Close File ***************/
for(b=3;b<c;b++)
{
 fPrintf(vnc_2,"%c",cname[b]);
}
fPrintf(vnc_2,"\r");


fPrintf(vnc_1,"clf "); /************ Close File ***************/
for(b=3;b<c;b++)
{
 fPrintf(vnc_1,"%c",cname[b]);
}
fPrintf(vnc_1,"\r");


 lcd_putc('\f');
 lcd_gotoxy(1,1);
 lcd_putc("@@@_Complete_@@@");
 lcd_gotoxy(1,2);
 lcd_putc("*** ANTIMAEW ***");
 delay_ms(2000);
```

```
    goto begin;
 }
 /*****************************************************************************/
 //################################  END Program  ################################
 /*****************************************************************************/
 void blink_y (void)
 {
  output_low(led_yellow);
  delay_ms(30);
  output_high(led_yellow);
  delay_ms(30);
 }

 void blink_g (void)
 {
  output_low(led_green);
  delay_ms(30);
  output_high(led_green);
  delay_ms(30);
 }

 void blink_r (void)
 {
  output_low(led_red);
  delay_ms(30);
  output_high(led_red);
  delay_ms(30);
```

```
}
void Q_file(void)
{
    lcd_putc('\f');
    lcd_gotoxy(1,1);
    lcd_putc("Files Detect= ");
    lcd_gotoxy(14,1);
    delay_ms(300);
if(j==0){lcd_putc("0");}else if(j==1){lcd_putc("1");}else if(j==2){lcd_putc("2");}
else if(j==3){lcd_putc("3");}else if(j==4){lcd_putc("4");}else if(j==5){lcd_putc("5");}
else if(j==6){lcd_putc("6");}else if(j==7){lcd_putc("7");}else if(j==8){lcd_putc("8");}
else if(j==9){lcd_putc("9");}else if(j==10){lcd_putc("10");}else if(j==11){lcd_putc("11");}
else if(j==12){lcd_putc("12");}else if(j==13){lcd_putc("13");}else if(j==14){lcd_putc("14");}
else if(j==15){lcd_putc("15");}else if(j==16){lcd_putc("16");}else if(j==17){lcd_putc("17");}
else if(j==18){lcd_putc("18");}else if(j==19){lcd_putc("19");}else if(j==20){lcd_putc("20");}
else if(j==21){lcd_putc("21");}else if(j==22){lcd_putc("22");}else if(j==23){lcd_putc("23");}
else if(j==24){lcd_putc("24");}else if(j==25){lcd_putc("25");}else if(j==26){lcd_putc("26");}
else if(j==27){lcd_putc("27");}else if(j==28){lcd_putc("28");}else if(j==29){lcd_putc("29");}
else if(j==30){lcd_putc("30");}
else {
    lcd_putc("!!");
    lcd_gotoxy(1,2);
    lcd_putc("  More than 30  ");
   }
   delay_ms(1500);
}
```

```
void name_file(void)
{
    a=3;
    y=1;
    lcd_putc('\f');
    as=1;
    while(a<1000 && as)
    {
      write:

      if(data[a]==0x3E){as=0;}
      if(data[a]==0x30){lcd_gotoxy(y,1);lcd_putc("0");}
      if(data[a]==0x31){lcd_gotoxy(y,1);lcd_putc("1");}
      if(data[a]==0x32){lcd_gotoxy(y,1);lcd_putc("2");}
      if(data[a]==0x33){lcd_gotoxy(y,1);lcd_putc("3");}
      if(data[a]==0x34){lcd_gotoxy(y,1);lcd_putc("4");}
      if(data[a]==0x35){lcd_gotoxy(y,1);lcd_putc("5");}
      if(data[a]==0x36){lcd_gotoxy(y,1);lcd_putc("6");}
      if(data[a]==0x37){lcd_gotoxy(y,1);lcd_putc("7");}
      if(data[a]==0x38){lcd_gotoxy(y,1);lcd_putc("8");}
      if(data[a]==0x39){lcd_gotoxy(y,1);lcd_putc("9");}
      if(data[a]==0x41){lcd_gotoxy(y,1);lcd_putc("A");}
      if(data[a]==0x42){lcd_gotoxy(y,1);lcd_putc("B");}
      if(data[a]==0x43){lcd_gotoxy(y,1);lcd_putc("C");}
      if(data[a]==0x44){lcd_gotoxy(y,1);lcd_putc("D");}
      if(data[a]==0x45){lcd_gotoxy(y,1);lcd_putc("E");}
      if(data[a]==0x46){lcd_gotoxy(y,1);lcd_putc("F");}
```

```
 if(data[a]==0x47){lcd_gotoxy(y,1);lcd_putc("G");}
 if(data[a]==0x48){lcd_gotoxy(y,1);lcd_putc("H");}
 if(data[a]==0x49){lcd_gotoxy(y,1);lcd_putc("I");}
 if(data[a]==0x4A){lcd_gotoxy(y,1);lcd_putc("J");}
 if(data[a]==0x4B){lcd_gotoxy(y,1);lcd_putc("K");}
 if(data[a]==0x4C){lcd_gotoxy(y,1);lcd_putc("L");}
 if(data[a]==0x4D){lcd_gotoxy(y,1);lcd_putc("M");}
 if(data[a]==0x4E){lcd_gotoxy(y,1);lcd_putc("N");}
 if(data[a]==0x4F){lcd_gotoxy(y,1);lcd_putc("O");}
 if(data[a]==0x50){lcd_gotoxy(y,1);lcd_putc("P");}
 if(data[a]==0x51){lcd_gotoxy(y,1);lcd_putc("Q");}
 if(data[a]==0x52){lcd_gotoxy(y,1);lcd_putc("R");}
 if(data[a]==0x53){lcd_gotoxy(y,1);lcd_putc("S");}
 if(data[a]==0x54){lcd_gotoxy(y,1);lcd_putc("T");}
 if(data[a]==0x55){lcd_gotoxy(y,1);lcd_putc("U");}
 if(data[a]==0x56){lcd_gotoxy(y,1);lcd_putc("V");}
 if(data[a]==0x57){lcd_gotoxy(y,1);lcd_putc("W");}
 if(data[a]==0x58){lcd_gotoxy(y,1);lcd_putc("X");}
 if(data[a]==0x59){lcd_gotoxy(y,1);lcd_putc("Y");}
 if(data[a]==0x5A){lcd_gotoxy(y,1);lcd_putc("Z");}
 if(data[a]==0x2E){lcd_gotoxy(y,1);lcd_putc(".");}
 if(data[a]==0x0D)
 {
  a++;y=1;delay_ms(1500);lcd_putc('\f');
  goto write;
 }
 a++;
```

```
    y++;
   }
}

void calfilesize(void)
{
 z=0;
 d=k+1;
 for(x=0;x<16;x++)
 {
  if(data[d]==0x30){size[z]=0; z++;}
  if(data[d]==0x31){size[z]=1; z++;}
  if(data[d]==0x32){size[z]=2; z++;}
  if(data[d]==0x33){size[z]=3; z++;}
  if(data[d]==0x34){size[z]=4; z++;}
  if(data[d]==0x35){size[z]=5; z++;}
  if(data[d]==0x36){size[z]=6; z++;}
  if(data[d]==0x37){size[z]=7; z++;}
  if(data[d]==0x38){size[z]=8; z++;}
  if(data[d]==0x39){size[z]=9; z++;}
  if(data[d]==0x41){size[z]=10; z++;}
  if(data[d]==0x42){size[z]=11; z++;}
  if(data[d]==0x43){size[z]=12; z++;}
  if(data[d]==0x44){size[z]=13; z++;}
  if(data[d]==0x45){size[z]=14; z++;}
  if(data[d]==0x46){size[z]=15; z++;}
  d++;
```

```
    }
        filesize=0;
        filesize=size[6];
        filesize=filesize<<4;
        filesize=filesize+size[7];
        filesize=filesize<<4;
        filesize=filesize+size[4];
        filesize=filesize<<4;
        filesize=filesize+size[5];
        filesize=filesize<<4;
        filesize=filesize+size[2];
        filesize=filesize<<4;
        filesize=filesize+size[3];
        filesize=filesize<<4;
        filesize=filesize+size[0];
        filesize=filesize<<4;
        filesize=filesize+size[1];
}
```

MICROCHIP

PIC18F8722 FAMILY

64/80-Pin, 1-Mbit, Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Peripheral Highlights:

- Two Master Synchronous Serial Port (MSSP) modules supporting 2/3/4-wire SPI™ (all 4 modes) and I^2C™ Master and Slave modes
- Two Capture/Compare/PWM (CCP) modules
- Three Enhanced Capture/Compare/PWM (ECCP) modules:
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
- Two Enhanced Addressable USART modules:
  - Supports RS-485, RS-232 and LIN 1.2
  - Auto-Wake-up on Start bit
  - Auto-Baud Detect
- 10-bit, up to 16-channel Analog-to-Digital Converter module (A/D)
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual analog comparators with input multiplexing
- High-current sink/source 25 mA/25 mA
- Four programmable external interrupts
- Four input change interrupts

External Memory Interface (PIC18F8527/8622/8627/8722 only):

- Address capability of up to 2 Mbytes
- 8-bit or 16-bit interface
- 8, 12, 16 and 20-bit Address modes

Power-Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 15 µA typical
- Sleep current down to 0.2 µA typical
- Timer1 Oscillator: 1.8 µA, 32 kHz, 2V
- Watchdog Timer: 2.1 µA

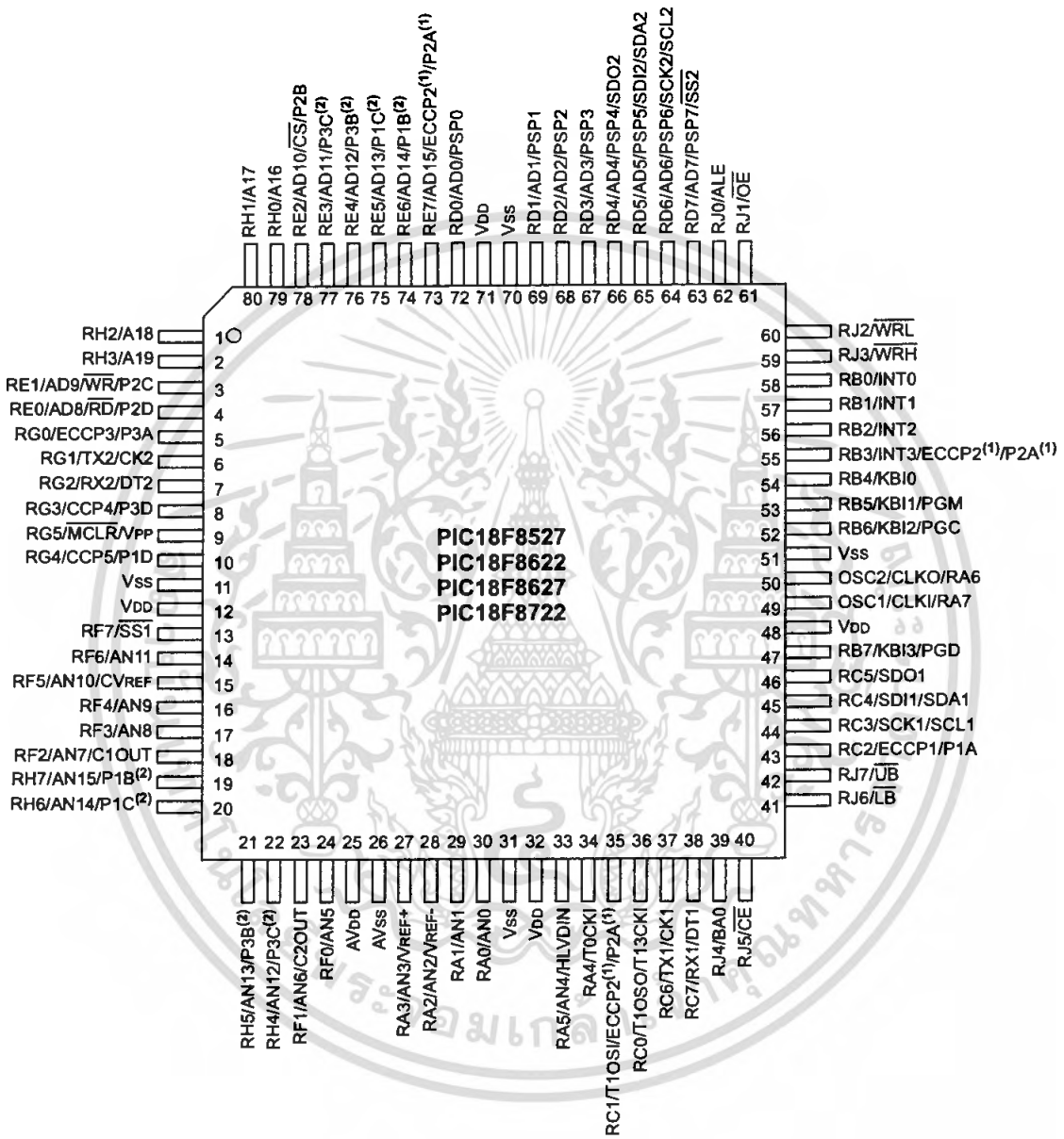
Special Microcontroller Features:

- C compiler optimized architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: 100 years typical
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-Supply In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Wide operating voltage range: 2.0V to 5.5V
- Fail-Safe Clock Monitor
- Two-Speed Oscillator Start-up
- nanoWatt Technology

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	MSSP			EUSART	Comparators	Timers 8/16-bit	External Bus
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI™	Master I^2C™				
PIC18F6527	48K	24576	3936	1024	54	12	2/3	2	Y	Y	2	2	2/3	N
PIC18F6622	64K	32768	3936	1024	54	12	2/3	2	Y	Y	2	2	2/3	N
PIC18F6627	96K	49152	3936	1024	54	12	2/3	2	Y	Y	2	2	2/3	N
PIC18F6722	128K	65536	3936	1024	54	12	2/3	2	Y	Y	2	2	2/3	N
PIC18F8527	48K	24576	3936	1024	70	16	2/3	2	Y	Y	2	2	2/3	Y
PIC18F8622	64K	32768	3936	1024	70	16	2/3	2	Y	Y	2	2	2/3	Y
PIC18F8627	96K	49152	3936	1024	70	16	2/3	2	Y	Y	2	2	2/3	Y
PIC18F8722	128K	65536	3936	1024	70	16	2/3	2	Y	Y	2	2	2/3	Y

Pin Diagrams (Continued)

80-Pin TQFP



Note 1: The ECCP2/P2A pin placement is determined by the CCP2MX configuration bit and Processor mode settings.
2: P1B, P1C, P3B and P3C pin placement is determined by the ECCPMX configuration bit.

FTDI Chip™

Future Technology
Devices International Ltd.

VINCULUM

BINDING USB TECHNOLOGIES

Vinculum VNC1L

Embedded USB Host Controller I.C.

The Vinculum VNC1L is the first of FTDI's Vinculum family of Embedded USB host controller integrated circuit devices. Not only is it able to handle the USB Host Interface, and data transfer functions but owing to the inbuilt MCU and embedded Flash memory, Vinculum can encapsulate the USB device classes as well. When interfacing to mass storage devices such as USB Flash drives, Vinculum also transparently handles the FAT File structure communicating via UART, SPI or parallel FIFO interfaces via a simple to implement command set. Vinculum provides a new cost effective solution for providing USB Host capability into products that previously did not have the hardware resources available.

The VNC1L is available in Pb-free (RoHS compliant) compact 48-Lead LQFP package.

1. Features

1.1 Hardware Features

- Single chip embedded USB host / slave controller I.C. device
- Entire USB protocol handled on the chip
- 8 / 32 bit V-MCU Core
- Twin DMA controllers for hardware acceleration
- Integrated 12 MHz to 48 MHz clock multiplier
- Integrated power-on-reset circuit with optional RESET# input pin
- 64k byte embedded Flash ROM program memory
- 4k byte internal data SRAM
- Standard USB firmware library supplied by FTDI
- Program or update firmware via USB Flash disk or UART interface
- Firmware easily upgradable in the field
- PROG# firmware programming control pin
- Two independent USB 2.0 Low speed / Full speed USB Host / Slave ports with integrated pull-up and pull-down resistors
- Four fully configurable data I/O and control Buses
- UART interface mode for data I/O, firmware programming, and command monitor interface
- FIFO interface mode with 8 bit bi-directional data bus and simple 4 wire handshake for data I/O and command monitor interface
- SPI slave interface mode for data I/O and command monitor interface
- Up to 28 GPIO interface pins for data I/O and command monitor interface
- Interface to MCU / PLD / FPGA via UART, FIFO, or SPI interface
- Legacy PS/2 keyboard and mouse interfaces
- Multi-processor configuration capable
- Support for USB suspend and resume
- Support for bus powered, self powered, and high-power bus powered USB device configurations
- 3.3V operation with 5V safe inputs
- Low operating and USB suspend current (25mA running / 2mA stnadby)
- Fully compliant with USB 2.0 specification - USB full speed (12 Mbps) and low speed (1.5 Mbps) USB host and slave device compatible
- 0°C to 70°C operating temperature range
- Full driver support for target / slave applications
- Available in compact Pb-free and green 48 Pin LQFP package (RoHS compliant)
- Full range of reference designs and evaluation kits available

1.2 Standard Firmware

- USB slave device and USB Flash disk interface with selectable UART / FIFO / SPI interface or USB slave device as the command monitor port (VDIF firmware)
- FTDI USB slave device and USB Flash disk interface with selectable UART / FIFO / SPI interface as the command monitor port (VDAP firmware)
- USB Flash disk to USB Flash disk with GPIO command monitor interface (VDFC firmware)
- FTDI USB slave device and USB Flash disk interface with selectable UART / FIFO / SPI interface as the command monitor port with audio playback command extensions (VMSC firmware)

1.3 Typical Applications

- Add USB host capability to embedded products
- Interface USB Flash drive to MCU / PLD / FPGA
- USB Flash drive to USB Flash drive file transfer interface
- Digital camera to USB Flash drive or other USB slave device interface
- PDA to USB Flash driver or other USB slave device interface
- MP3 Player to USB Flash drive or other USB slave device interface
- USB MP3 Player to USB MP3 Player
- Mobile phone to USB Flash drive or other USB slave device interface
- GPS to mobile phone interface
- Instrumentation USB Flash drive or other USB slave device interfacing
- Datalogger USB Flash drive or other USB slave device interface
- Set Top Box - USB device interface

2. Block Diagram

2.1 Simplified Block Diagram

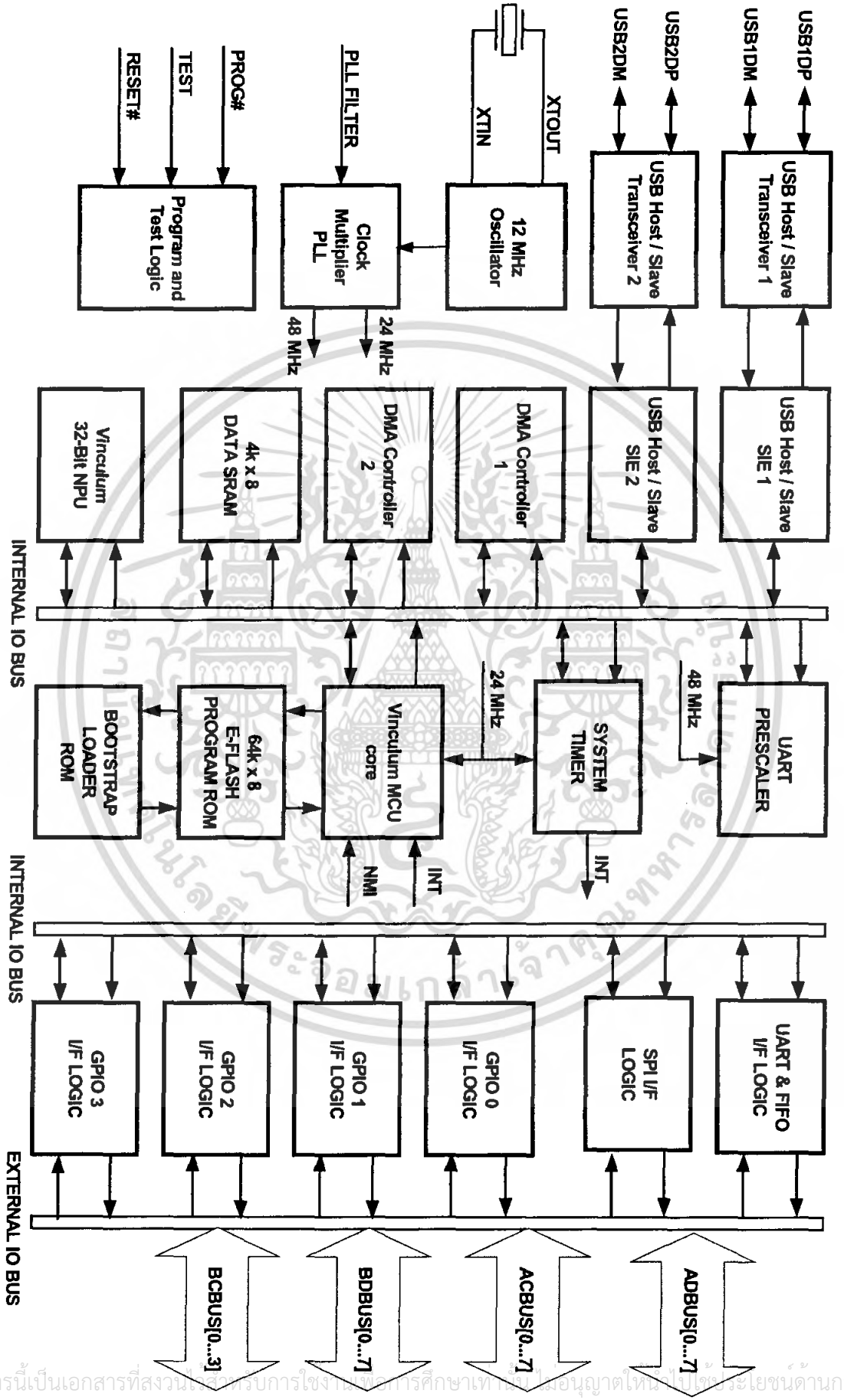


Figure 1 - Simplified Block Diagram

2.2 Functional Block Descriptions

USB Host / Slave Transceivers 1 and 2 - The two USB transceiver cells provide the USB host / slave physical USB 1.1 / USB 2.0 full-speed device interface. On each the output drivers provide 3.3V level slew rate control signalling, whilst a differential receiver and two single ended receivers provide USB data in, SEO and USB Reset condition detection. These cells also incorporate internal USB pull-up or pull down resistors as required for host or slave mode.

USB Host / Slave Serial Interface Engine (SIE) - These blocks handle the parallel to serial and serial to parallel conversion of the USB Physical layer including bit suffing / unstuffing, CRC generation / checking, USB frame generation and error checking.

12 MHz Oscillator - The 12MHz Oscillator cell generates a 12MHz reference clock input to the Clock Multiplier PLL from an exteral 12MHz crystal.

Clock Multiplier PLL - The Clock Multiplier PLL takes the 12MHz input from the Oscillator Cell and generates 24MHz and 48MHz reference clock signals, which is used by the USB SIE Blocks, the MCU core, System Timer and UART Prescaler blocks.

Program and Test Logic - this block provides a means of programming the onboard E-Flash memory. When PROG# is pulled low and the device is reset, the onboard E-Flash memory is bypassed by an internal hard coded BootStrap Loader ROM which contains code to allow the E-Flash memory to be programmed via commands to the UART interface. FTDI provides a software utility which allows the VNC1L to be programmed using this method. The TEST pin is used in manufacturing to enhance the testability of the various internal blocks and should be tied to GND.

DMA Controller 1 and 2 - The twin DMA controllers in the VNC1L greatly enhance performance by allowing data from the two USB SIE controllers, UART, FIFO and SPI to be transferred between each other via the data SRAM with minimal MCU intervention.

Data SRAM - This 4k x 8bit block acts as the data (variable) memory for the Vinculum MCU, though it can also be accessed transparently to the MCU by the twin DMA controllers.

NPU (Numeric CoProcessor) - Most Vinclum MCU operations are 8-bit, however there are some scenarios such as transversing disk FAT tables which involve extensive 32 bit arithmetic. In order to speed up these operations, the MCU has a dedicated 32 bit co-processor block.

UART Prescaler - this block provides the master transmit / receive clock for the UART block. By varying the prescalar value, the baud rate of the UART can be adjusted over a range of 300 baud to 1M baud.

SYSTEM TIMER - The system timer provides a regular interrupt to the Vinculum MCU, typically at 1mS intervals. This is used by the MCU to provide timeouts and other timing functions.

VINCULUM MCU CORE - The "heart" of the VNC1L is the VMCU core based on FTDI's proprietary 8-bit embedded MCU (EMCU) architectiure. VMCU has a Harvard architecture i.e. separate code and data space and supports 64k byes of program code, 64k byes of (paged) data space and 256 bytes of IO space. It uses "enhanced CISC" technology - typically VNCU instructions would replace several lines of code in conventional CISC or RISC processors giving RISC like performance in a CISC architecture with the advantage over both of excellent code compression in the program ROM space.

E-FLASH Program ROM - The VNCL1L has 64k bytes of embedded Flash (E-Flash) memory. No special programming voltages are necessary for programming the onboard E-FLASH as these are provided internally on-chip. Common methods of programming the E-FLASH (both under control of the VMCU) are via the UART by pulling the PROG# pin low and resetting the device OR by using the programming via a USB FLASH drive feature provided in many of the VNC1L firmware packages.

BOOTSTRAP LOADER ROM - This is a small block of hard encoded ROM (512 x 8 bits) whivh bypasses the main e_FLASH memory when PROG# is pulled low. This provides a means of programming the entire E-Flash memory via the UART interface.

UART and FIFO Logic - These provide optional serial and parallel interfaces to the VNC1L equivalent to the interfaces on FTDI's FT232 and FT245 USB UART and FIFO products.

GPIO Blocks - general purpose IO pins. See the tables below to determine which are available for any specific configuration.

2. Device Pin Out and Signal Descriptions

2.1 48 Lead LQFP Pin Out

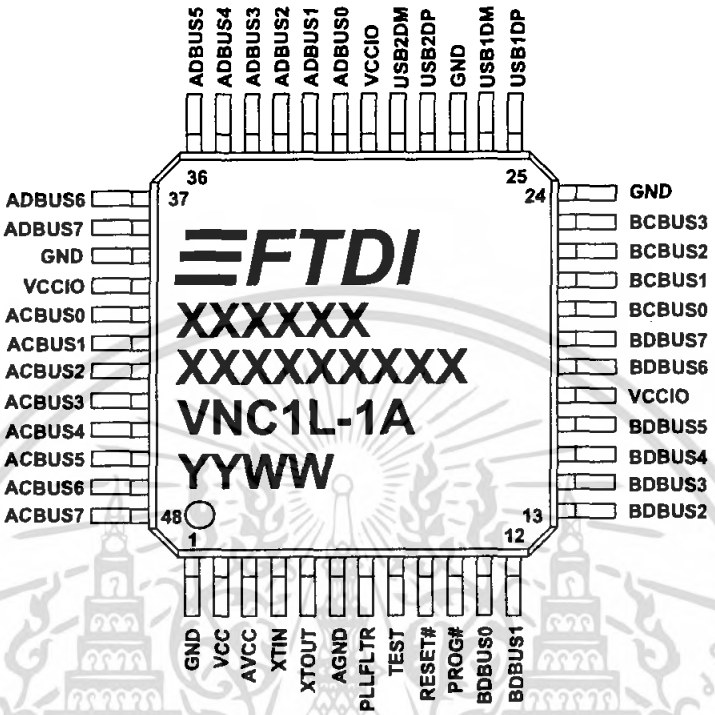


Figure 2 - 48 pin LQFP Package Pin Out

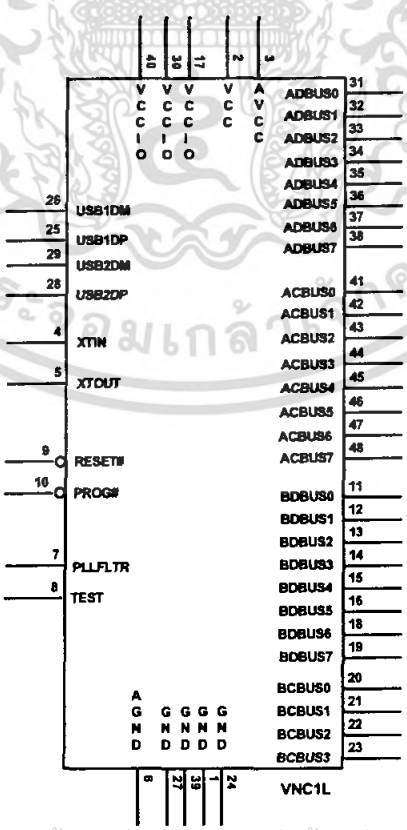


Figure 3 - VNC1L Pin Out - Schematic

2.2 48 Lead LQFP Package Signal Descriptions

Table 1 - Pin Out Description

Pin No.	Name	Type	Description
USB Interface Group			
25	USB1DP	I/O	USB host / slave port 1 - USB Data Signal Plus with integrated pull up / pull down resistor.
26	USB1DM	I/O	USB host / slave port 1 - USB Data Signal Minus with integrated pull up / pull down resistor.
28	USB2DP	I/O	USB host / slave port 2 - USB Data Signal Plus with integrated pull up / pull down resistor.
29	USB2DM	I/O	USB host / slave port 2 - USB Data Signal Minus with integrated pull up / pull down resistor.
Power and Ground Group			
1, 24, 27, 39	GND	PWR	Device ground supply pins
2	VCC	PWR	3.3V supply to the device core.
3	AVCC	PWR	+3.3V supply to the internal clock multiplier. This pin requires a 100 nF decoupling capacitor.
6	AGND	PWR	Device analog ground supply for internal clock multiplier
17, 30, 40	VCCIO	PWR	+3.3V supply to the ADBUS, ACBUS, BDBUS and BCBUS Interface pins (11...16, 18...23, 31...38, 41...48).
Miscellaneous Signal Group			
4	XTIN	Input	Input to 12MHz Oscillator Cell. Connect 12 MHz crystal across pins 4 and 5, with suitable loading capacitors to GND. This pin can also be driven by an external 12 MHz clock signal. Note that the switching threshold of this pin is VCC/2, so if driving from an external source, the source must be driving at 5V CMOS level, or a.c. coupled to centre around VCC/2.
5	XTOUT	Output	Output from 12MHz Oscillator Cell. Connect 12 MHz crystal across pins 4 and 5, with suitable loading capacitors to GND. XTOUT stops oscillating during USB suspend, so take care using this signal to clock external logic.
7	PLLFLTR	Input	External PLL filter circuit input. RC filter circuit must be fitted on this pin.
8	TEST	Input	Puts the device into I.C. test mode. Must be tied to GND for normal operation.
9	RESET#	Input	Can be used by an external device to reset the VNC1L. This pin can be used in combination with PROG# and the UART interface to program firmware into the VNC1L. If not required pull up to VCC via a 10 kΩ resistor.*
10	PROG#	Input	This pin is used in combination with the RESET# pin and the UART interface to program firmware into the VNC1L.*
Data and Control Bus Signals			
			Interface Mode
			UART Interface / Parallel FIFO Interface / SPI Slave Interface / I/O Port
11	BDBUS0	I/O	5V safe bidirectional data / control bus, BD bit 0 — PortBD0
12	BDBUS1	I/O	5V safe bidirectional data / control bus, BD bit 1 — PortBD1
13	BDBUS2	I/O	5V safe bidirectional data / control bus, BD bit 2 — PortBD2
14	BDBUS3	I/O	5V safe bidirectional data / control bus, BD bit 3 — PortBD3
15	BDBUS4	I/O	5V safe bidirectional data / control bus, BD bit 4 — PortBD4
16	BDBUS5	I/O	5V safe bidirectional data / control bus, BD bit 5 — PortBD5
18	BDBUS6	I/O	5V safe bidirectional data / control bus, BD bit 6 — PortBD6
19	BDBUS7	I/O	5V safe bidirectional data / control bus, BD bit 7 — PortBD7
20	BCBUS0	I/O	5V safe bidirectional data / control bus, BC bit 0 — PS2Clk1** / PS2Clk1** / PS2Clk1** / PortBC0
21	BCBUS1	I/O	5V safe bidirectional data / control bus, BC bit 1 — PS2Data1** / PS2Data1** / PS2Data1** / PortBC1
22	BCBUS2	I/O	5V safe bidirectional data / control bus, BC bit 2 — PS2Clk2** / PS2Clk2** / PS2Clk2** / PortBC2
23	BCBUS3	I/O	5V safe bidirectional data / control bus, BC bit 3 — PS2Data2** / PS2Data2** / PS2Data2** / PortBC3
31	ADBUS0	I/O	5V safe bidirectional data / control bus, AD bit 0 — TXD / D0 / SCLK / PortAD0
32	ADBUS1	I/O	5V safe bidirectional data / control bus, AD bit 1 — RXD / D1 / SDI / PortAD1
33	ADBUS2	I/O	5V safe bidirectional data / control bus, AD bit 2 — RTS# / D2 / SDO / PortAD2
34	ADBUS3	I/O	5V safe bidirectional data / control bus, AD bit 3 — CTS# / D3 / CS / PortAD3
35	ADBUS4	I/O	5V safe bidirectional data / control bus, AD bit 4 — DTR# / D4 / / PortAD4
36	ADBUS5	I/O	5V safe bidirectional data / control bus, AD bit 5 — DSR# / D5 / / PortAD5
37	ADBUS6	I/O	5V safe bidirectional data / control bus, AD bit 6 — DCD# / D6 / / PortAD6
38	ADBUS7	I/O	5V safe bidirectional data / control bus, AD bit 7 — RI# / D7 / / PortAD7
41	ACBUS0	I/O	5V safe bidirectional data / control bus, AC bit 0 — TXDEN# / RXF# / / PortAC0

Table 1 continued - Pin Out Description

42	ACBUS1	I/O	5V safe bidirectional data / control bus, AC bit 1		TXE#		PortAC1
43	ACBUS2	I/O	5V safe bidirectional data / control bus, AC bit 2		RD#		PortAC2
44	ACBUS3	I/O	5V safe bidirectional data / control bus, AC bit 3		WR		PortAC3
45	ACBUS4	I/O	5V safe bidirectional data / control bus, AC bit 4				PortAC4
46	ACBUS5	I/O	5V safe bidirectional data / control bus, AC bit 5				PortAC5
47	ACBUS6	I/O	5V safe bidirectional data / control bus, AC bit 6				PortAC6
48	ACBUS7	I/O	5V safe bidirectional data / control bus, AC bit 7. To use a 12 MHz crystal with the VNC1L fit a 47 kΩ pull-down resistor. Alternatively, fitting a 47 kΩ pull-up resistor on this pin will switch off the internal clock multiplier, allowing the device to be fed with an external 48Mz clock signal into XTIN.				PortAC7

* These pins are pulled to VCC via internal 200kΩ resistors.
** PS/2 Ports can be available while UART, FIFO, or SPI interface is enabled.

2.3 UART Interface Signal Descriptions

Table 4 - Data and Control Bus Signal Mode Options - UART Interface

Pin No.	Name	Type	Description
31	TXD	Output	Transmit asynchronous data output
32	RXD	Input	Receive asynchronous data input
33	RTS#	Output	Request To Send Control Output / Handshake signal
34	CTS#	Input	Clear To Send Control Input / Handshake signal
35	DTR#	Output	Data Terminal Ready Control Output / Handshake signal
36	DSR#	Input	Data Set Ready Control Input / Handshake signal
37	DCD#	Input	Data Carrier Detect Control Input
38	RI#	Input	Ring Indicator Control Input. When the Remote Wake up option is enabled in the EEPROM, taking RI# low can be used to resume the PC USB Host controller from suspend.
41	TXDEN	Output	Enable Transmit Data for RS485 designs

2.4 Parallel FIFO Interface Signal Descriptions and Timing Diagrams

Table 5 - Data and Control Bus Signal Mode Options - Parallel FIFO Interface

Pin No.	Name	Type	Description
31	D0	I/O	FIFO Data Bus Bit 0
32	D1	I/O	FIFO Data Bus Bit 1
33	D2	I/O	FIFO Data Bus Bit 2
34	D3	I/O	FIFO Data Bus Bit 3
35	D4	I/O	FIFO Data Bus Bit 4
36	D5	I/O	FIFO Data Bus Bit 5
37	D6	I/O	FIFO Data Bus Bit 6
38	D7	I/O	FIFO Data Bus Bit 7
41	RXF#	OUTPUT	When high, do not read data from the FIFO. When low, there is data available in the FIFO which can be read by strobing RD# low, then high again.
42	TXE#	OUTPUT	When high, do not write data into the FIFO. When low, data can be written into the FIFO by strobing WR high, then low.
43	RD#	INPUT	Writes the data byte on the D0...D7 pins into the transmit FIFO buffer when WR goes from high to low.
44	WR	INPUT	Enables the current FIFO data byte on D0...D7 when low. Fetched the next FIFO data byte (if available) from the receive FIFO buffer when RD# goes from high to low

Figure 4 - FIFO Read Cycle

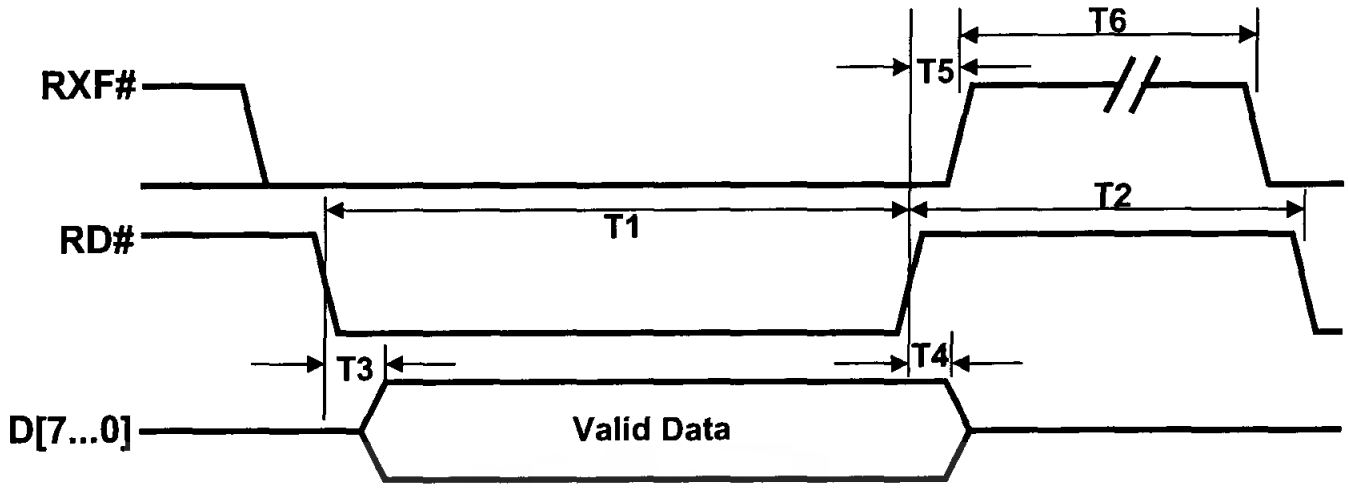


Table 6 - FIFO Read Cycle Timings

Time	Description	Min	Max	Unit
T1	RD Active Pulse Width	50	-	ns
T2	RD to RD Pre-Charge Time	50 + T6	-	ns
T3	RD Active to Valid Data*	20	50	ns
T4	Valid Data Hold Time from RD Inactive*	0	-	ns
T5	RD Inactive to RXF#	0	25	ns
T6	RXF Inactive After RD Cycle	80	-	ns

* Load = 30pF

Figure 5 - FIFO Write Cycle

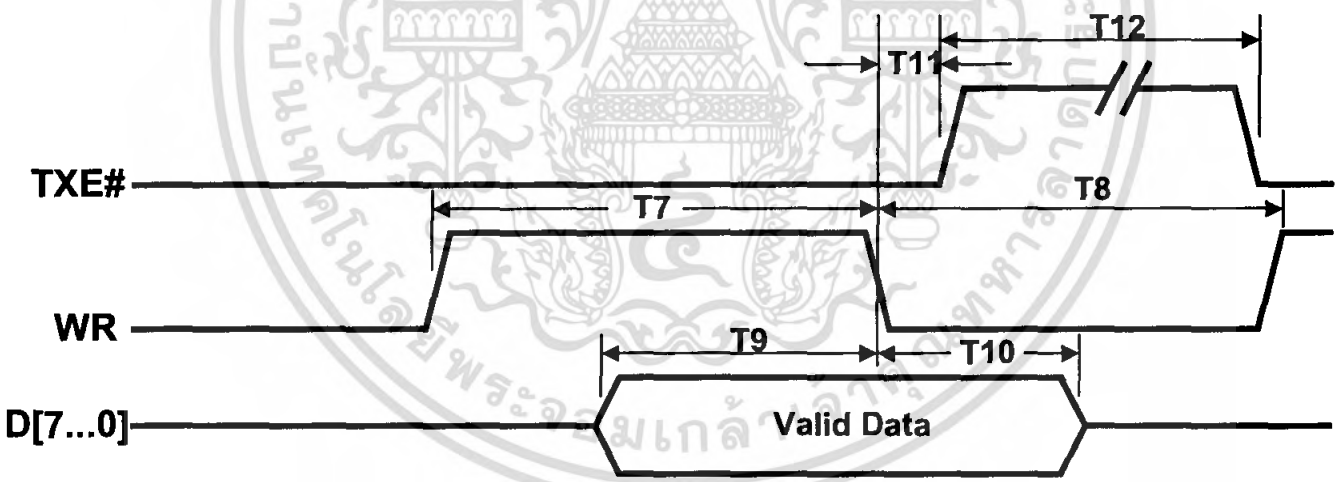


Table 7 - FIFO Write Cycle Timings

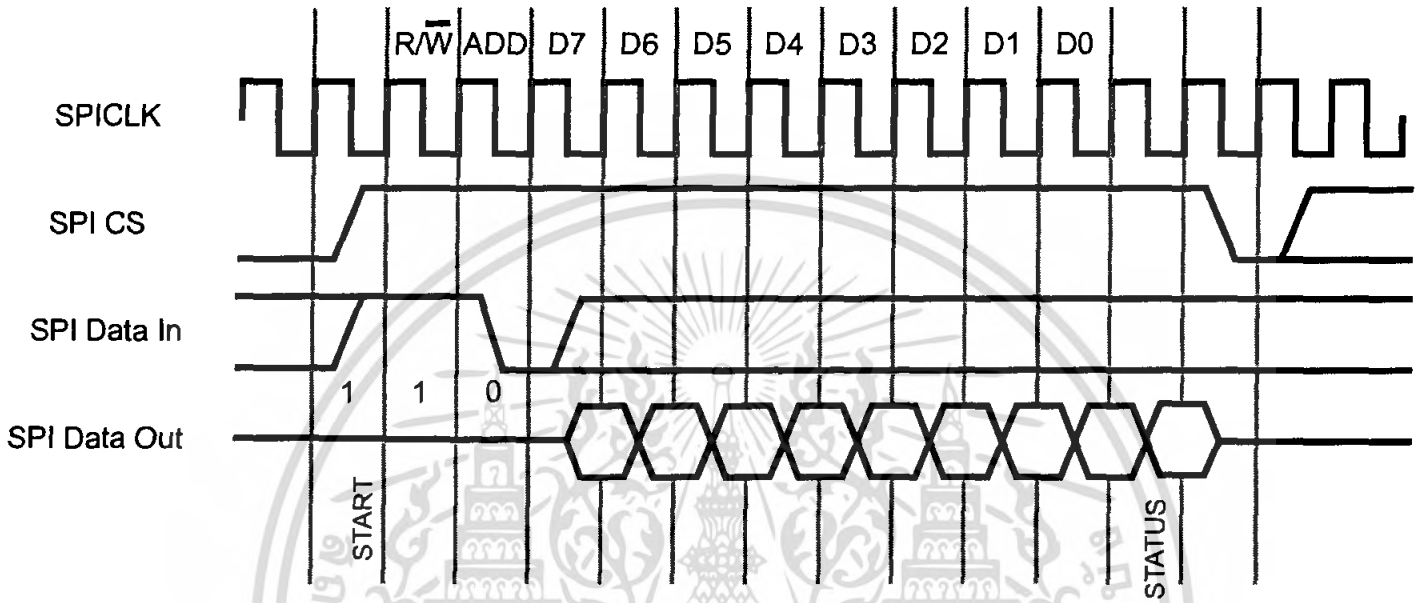
Time	Description	Min	Max	Unit
T7	WR Active Pulse Width	50	-	ns
T8	WR to RD Pre-Charge Time	50	-	ns
T9	Data Setup Time before WR Inactive	20	-	ns
T10	Data Hold Time from WR Inactive	0	-	ns
T11	WR Inactive to TXE#	5	25	ns
T12	TXE Inactive After WR Cycle	80	-	ns

2.5 SPI Interface Signal Descriptions and Timing Diagrams

Table 8 - Data and Control Bus Signal Mode Options - SPI Interface

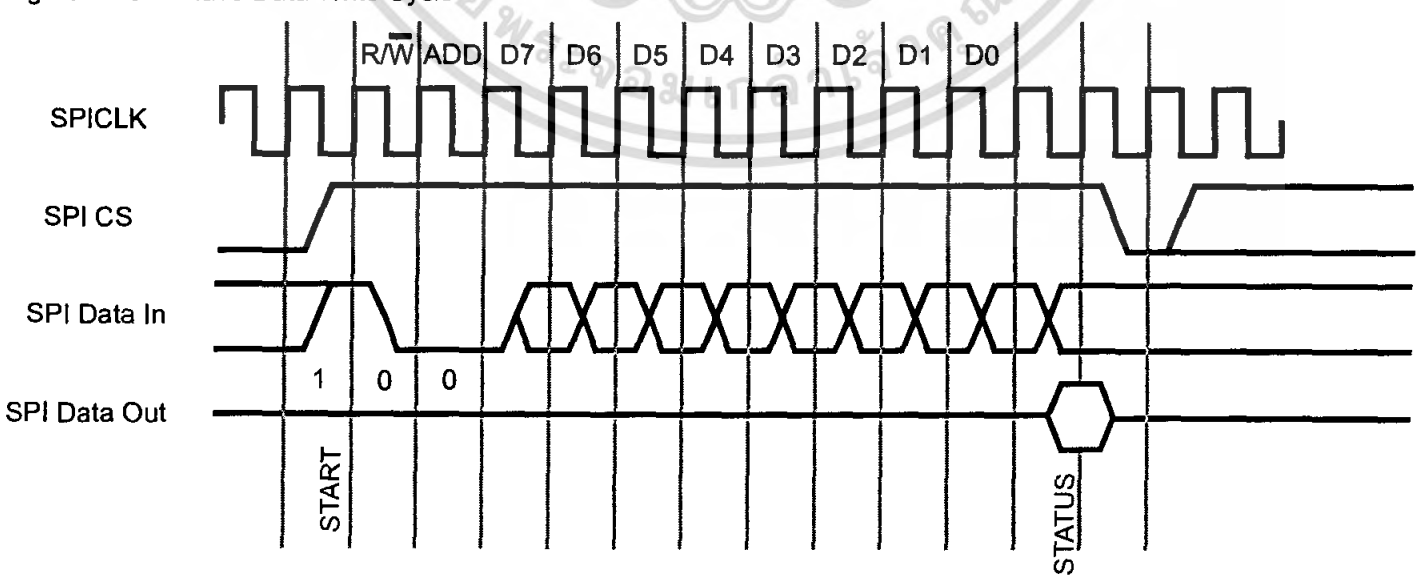
Pin No.	Name	Type	Description
31	SCLK	Input	SPI Clock input, 12MHz maximum.
32	SDI	Input	SPI Serial Data Input
33	SDO	Output	SPI Serial Data Output
34	CS	Input	SPI Chip Select Input

Figure 6 - SPI Slave Data Read Cycle



From Start - SPI CS must be held high for the entire read cycle, and must be taken low for at least one clock period after the read is completed. The first bit on SPI Data In is the R/W bit - inputting a '1' here allows data to be read from the chip. The next bit is the address bit, ADD, which is used to indicate whether the data register ('0') or the status register ('1') is read from. During the SPI read cycle a byte of data will start being output on SPI Data Out on the next clock cycle after the address bit, MSB first. After the data has been clocked out of the chip, the status of SPI Data Out should be checked to see if the data read is new data. A '0' level here on SPI Data Out means that the data read is new data. A '1' indicates that the data read is old data, and the read cycle should be repeated to get new data. Remember that CS must be held low for at least one clock period before being taken high again to continue with the next read or write cycle.

Figure 7 - SPI Slave Data Write Cycle



From Start - SPI CS must be held high for the entire write cycle, and must be taken low for at least one clock period after the write is completed. The first bit on SPI Data In is the R/W bit - inputting a '0' here allows data to be written to the chip. The next bit is the address bit, ADD, which is used to indicate whether the data register ('0') or the status

register ('1') is written to. During the SPI write cycle a byte of data can be input to SPI Data In on the next clock cycle after the address bit, MSB first. After the data has been clocked in to the chip, the status of SPI Data Out should be checked to see if the data read was accepted. A '0' level on SPI Data Out means that the data write was accepted. A '1' indicates that the internal buffer is full, and the write should be repeated. Remember that CS must be held low for at least one clock period before being taken high again to continue with the next read or write cycle.

Figure 8 - SPI Slave Data Timing Diagrams

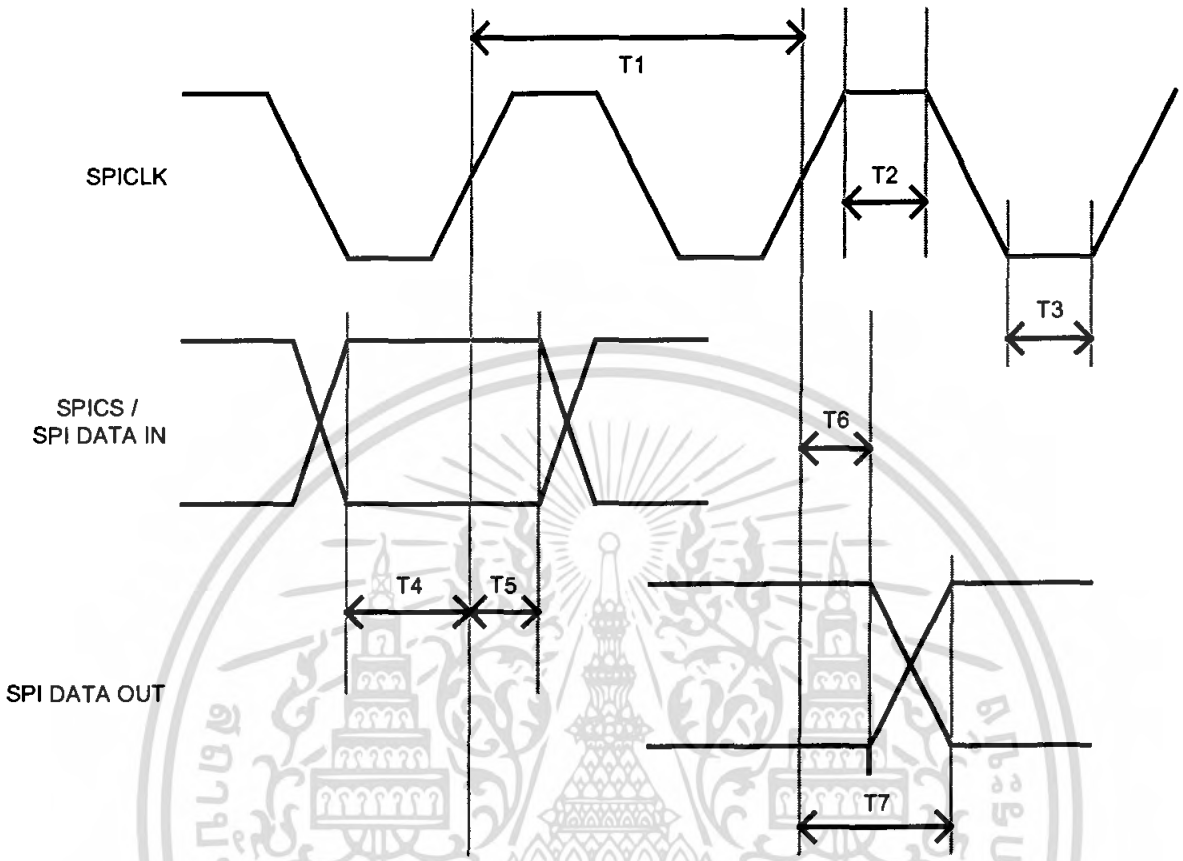


Table 9 - SPI Slave Data Timing

Time	Description	Min	Typical	Max	Unit
T1	SPICLK Period	83	-	-	ns
T2	SPICLK High	20	-	-	ns
T3	SPICLK Low	20	-	-	ns
T4	Input Setup Time	10	-	-	ns
T5	Input Hold TIme	10	-	-	ns
T6	Output Hold Time	2	-	-	ns
T7	Output Valid Time	-	-	20	ns

Table 10 - Status Register (ADD = '1')

Bit	Description
0	RXF#
1	TXE#
2	-
3	-
4	RXF IRQEn
5	TXE IRQEn
6	-
7	-

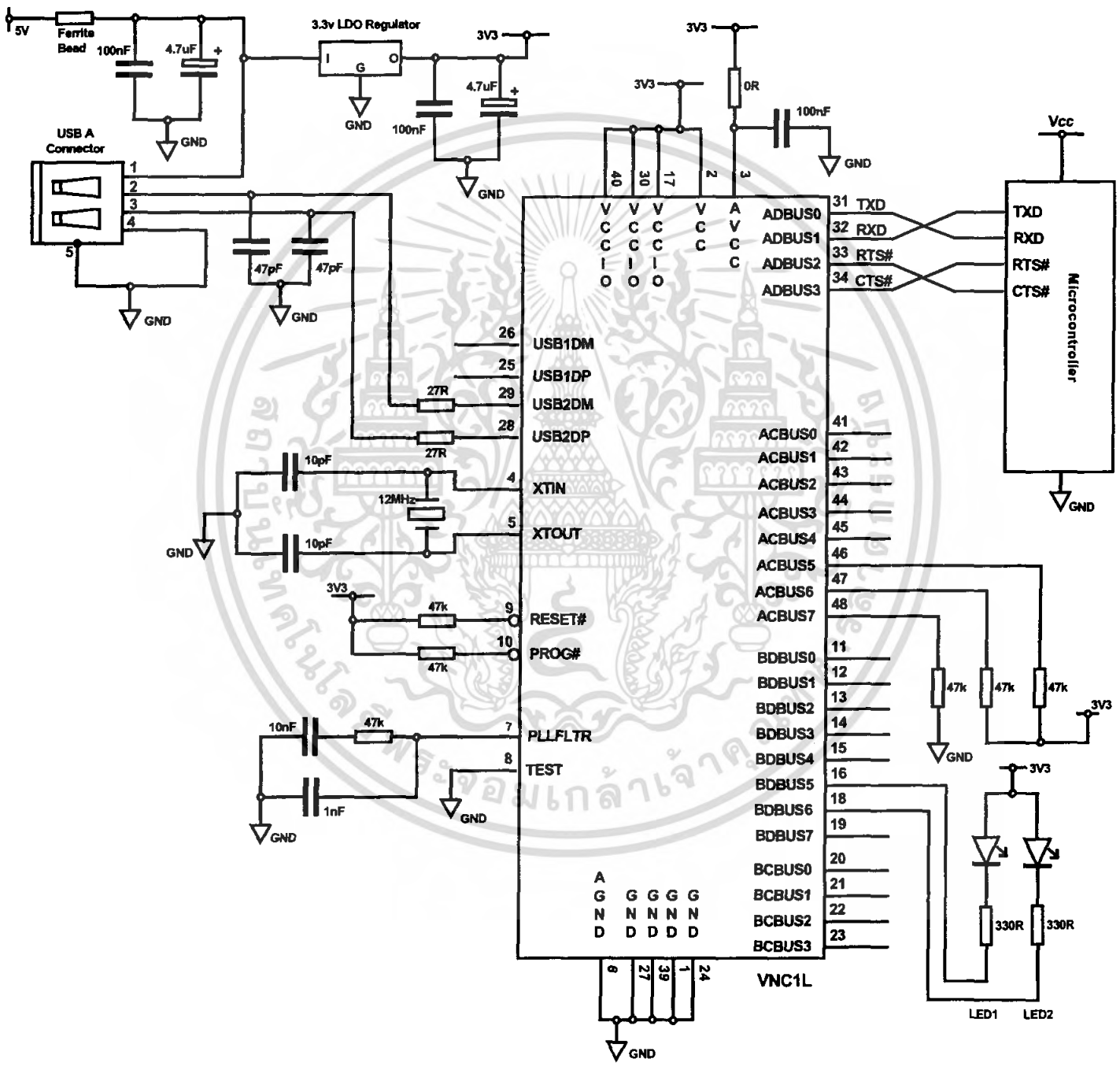
2.6 PS/2 Keyboard and Mouse Interface

Table 11 - Data and Control Bus Signal Mode Options - PS/2 Keyboard and Mouse Interface

Pin No.	Name	Type	Description
20	PS2Clk1	I/O	PS/2 Keyboard or Mouse interface 1 clock signal
21	PS2Data1	I/O	PS/2 Keyboard or Mouse interface 1 data signal
22	PS2Clk2	I/O	PS/2 Keyboard or Mouse interface 2 clock signal
23	PS2Data2	I/O	PS/2 Keyboard or Mouse interface 2 data signal

5. Device Configurations

5.1 Example VNC1L Schematic (MCU - UART interface)



19-3953; Rev 2; 2/07

EVALUATION KIT AVAILABLE

USB Peripheral/Host Controller with SPI Interface

General Description

The MAX3421E USB peripheral/host controller contains the digital logic and analog circuitry necessary to implement a full-speed USB peripheral or a full-/low-speed host compliant to USB specification rev 2.0. A built-in transceiver features ±15kV ESD protection and programmable USB connect and disconnect. An internal serial interface engine (SIE) handles low-level USB protocol details such as error checking and bus retries. The MAX3421E operates using a register set accessed by an SPI™ interface that operates up to 26MHz. Any SPI master (microprocessor, ASIC, DSP, etc.) can add USB peripheral or host functionality using the simple 3- or 4-wire SPI interface.

The MAX3421E makes the vast collection of USB peripherals available to any microprocessor, ASIC, or DSP when it operates as a USB host. For point-to-point solutions, for example, a USB keyboard or mouse interfaced to an embedded system, the firmware that operates the MAX3421E can be simple since only a targeted device is supported.

Internal level translators allow the SPI interface to run at a system voltage between 1.4V and 3.6V. USB-timed operations are done inside the MAX3421E with interrupts provided at completion so an SPI master does not need timers to meet USB timing requirements. The MAX3421E includes eight general-purpose inputs and outputs so any microprocessor that uses I/O pins to implement the SPI interface can reclaim the I/O pins and gain additional ones.

The MAX3421E operates over the extended -40°C to +85°C temperature range and is available in a 32-pin TQFP package (5mm x 5mm) and a 32-pin TQFN package (5mm x 5mm).

Applications

Embedded Systems
Medical Devices
Microprocessors and DSPs
Custom USB Devices
Cameras
Desktop Routers
PLCs
Set-Top Boxes
PDAs
MP3 Players
Instrumentation

Features

- **Microprocessor-Independent USB Solution**
- **Software Compatible with the MAX3420E USB Peripheral Controller with SPI Interface**
- **Complies with USB Specification Revision 2.0 (Full-Speed 12Mbps Peripheral, Full-/Low-Speed 12Mbps/1.5Mbps Host)**
- **Integrated USB Transceiver**
- **Firmware/Hardware Control of an Internal D+ Pullup Resistor (Peripheral Mode) and D+/D- Pulldown Resistors (Host Mode)**
- **Programmable 3- or 4-Wire, 26MHz SPI Interface**
- **Level Translators and V_L Input Allow Independent System Interface Voltage**
- **Internal Comparator Detects V_{BUS} for Self-Powered Peripheral Applications**
- **ESD Protection on D+, D-, and VBCOMP**
- **Interrupt Output Pin (Level- or Programmable-Edge) Allows Polled or Interrupt-Driven SPI Interface**
- **Eight General-Purpose Inputs and Eight General-Purpose Outputs**
- **Interrupt Signal for General-Purpose Input Pins, Programmable Edge Polarity**
- **Intelligent USB SIE**
- **Automatically Handles USB Flow Control and Double Buffering**
- **Handles Low-Level USB Signaling Details**
- **Contains Timers for USB Time-Sensitive Operations so SPI Master Does Not Need to Time Events**
- **Space-Saving Lead-Free TQFP and TQFN Packages (5mm x 5mm)**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE	PKG CODE
MAX3421EEHJ+	-40°C to +85°C	32 TQFP	H32-1
MAX3421EETJ+	-40°C to +85°C	32 TQFN-EP*	T3255-4

*EP = Exposed paddle, connected to ground.
+Denotes lead-free package.

SPI is a trademark of Motorola, Inc.

USB Peripheral/Host Controller with SPI Interface

Features in Host Operation

- Eleven Registers (R21–R31) are Added to the MAX3420E Register Set to Control Host Operation
- Host Controller Operates at Full Speed or Low Speed
- FIFOS
  - SNDFIFO: Send FIFO, Double-Buffered 64-Byte
  - RCVFIFO: Receive FIFO, Double-Buffered 64-Byte
- Handles DATA0/DATA1 Toggle Generation and Checking
- Performs Error Checking for All Transfers
- Automatically Generates SOF (Full-Speed)/EOP (Low-Speed) at 1ms Intervals
- Automatically Synchronizes Host Transfers with Beginning of Frame (SOF/EOP)
- Reports Results of Host Requests
- Supports USB Hubs
- Supports ISOCHRONOUS Transfers
- Simple Programming
  - SIE Automatically Generates Periodic SOF (Full-Speed) or EOP (Low-Speed) Frame Markers
  - SPI Master Loads Data, Sets Function Address, Endpoint, and Transfer Type, and Initiates the Transfer
  - MAX3421E Responds with an Interrupt and Result Code Indicating Peripheral Response
  - Transfer Request Can be Loaded Any Time
  - SIE Synchronizes with Frame Markers
  - For Multipacket Transfers, the SIE Automatically Maintains and Checks the Data Toggles

Features in Peripheral Operation

- Built-In Endpoint FIFOS
  - EP0: CONTROL (64 bytes)
  - EP1: OUT, Bulk or Interrupt, 2 x 64 Bytes (Double-Buffered)
  - EP2: IN, Bulk or Interrupt, 2 x 64 Bytes (Double-Buffered)
  - EP3: IN, Bulk or Interrupt (64 Bytes)
- Double-Buffered Data Endpoints Increase Throughput by Allowing the SPI Master to Transfer Data Concurrent with USB Transfers
- SETUP Data Has its Own 8-Byte FIFO, Simplifying Firmware

Typical Application Circuits

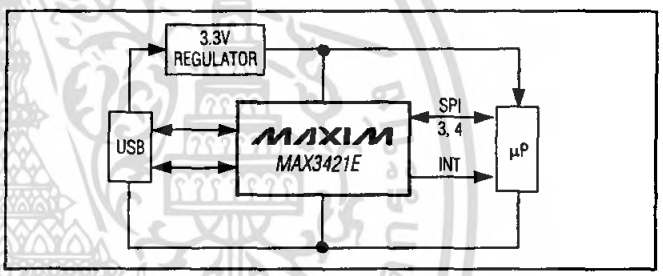
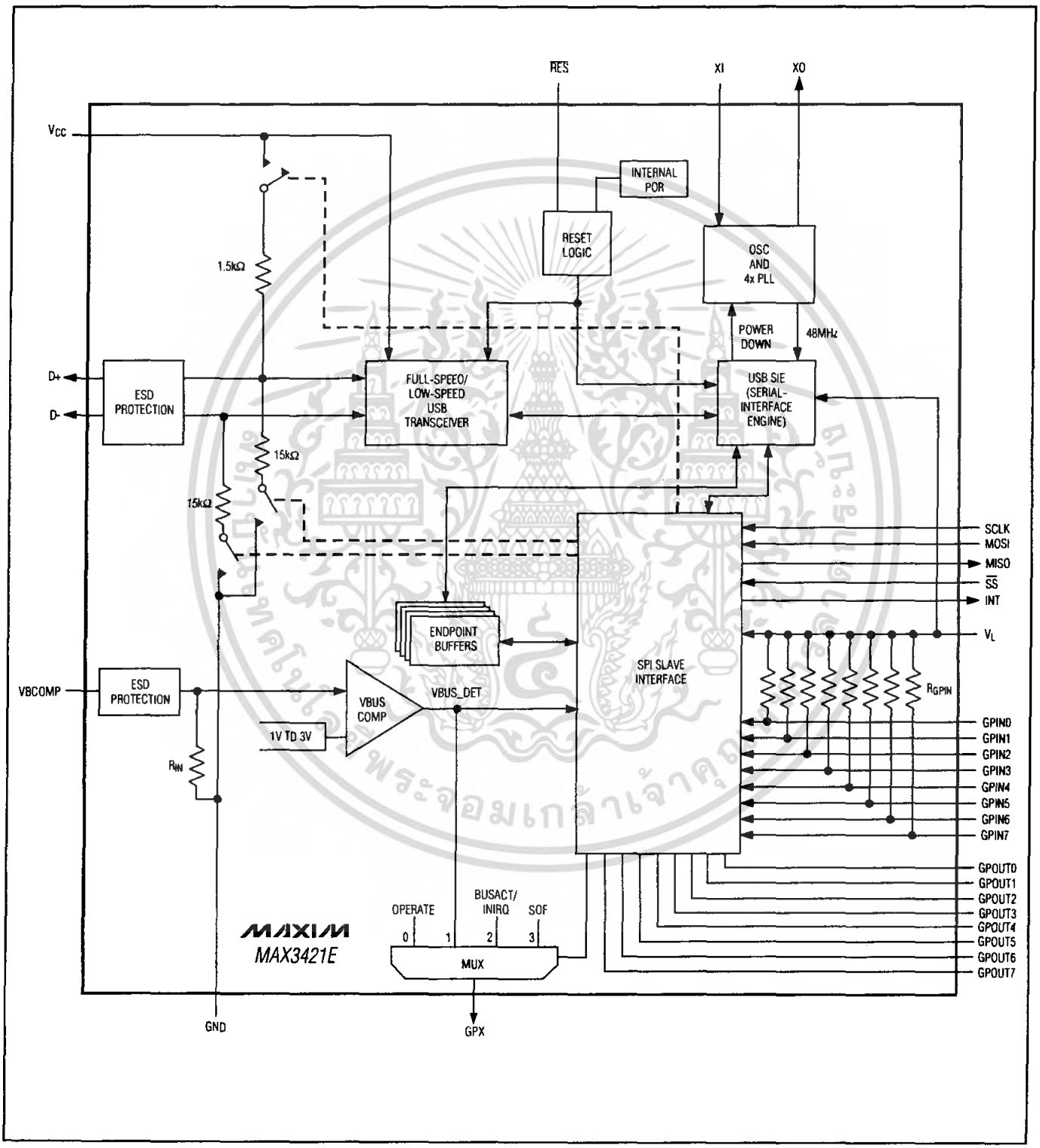


Figure 1. The MAX3421E Connects to Any Microprocessor Using 3 or 4 Interface Pins

The MAX3421E connects to any microprocessor (µP) using 3 or 4 interface pins (Figure 1). On a simple microprocessor without SPI hardware, these can be bit-banged general-purpose I/O pins. Eight GPIN and eight GPOUT pins on the MAX3421E more than replace the µP pins necessary to implement the interface. Although the MAX3421E SPI hardware includes separate data-in (MOSI, master-out, slave-in) and data-out (MISO, master-in, slave-out) pins, the SPI interface can also be configured for the MOSI pin to carry bidirectional data, saving an interface pin. This is referred to as half-duplex mode.

Functional Diagram



Pin Description

PIN	NAME	INPUT/ OUTPUT	FUNCTION
1	GPIN7	Input	General-Purpose Input. GPIN7–GPIN0 are connected to V_L with internal pullup resistors. GPIN7–GPIN0 logic levels are referenced to the voltage on V_L.
2	V_L	Input	Level-Translator Voltage Input. Connect V_L to the system's 1.4V to 3.6V logic-level power supply. Bypass V_L to ground with a 0.1µF capacitor as close to V_L as possible.
3, 19	GND	Input	Ground
4	GPOUT0	Output	General-Purpose Push-Pull Outputs. GPOUT7–GPOUT0 logic levels are referenced to the voltage on V_L.
5	GPOUT1		
6	GPOUT2		
7	GPOUT3		
8	GPOUT4		
9	GPOUT5		
10	GPOUT6		
11	GPOUT7		
12	$\overline{RES}$	Input	Device Reset. Drive $\overline{RES}$ low to clear all of the internal registers except for PINCTL (R17), USBCTL (R15), and SPI logic. The logic level is referenced to the voltage on V_L. (See the *Device Reset* section for a description of resets available on the MAX3421E.)
13	SCLK	Input	SPI Serial-Clock Input. An external SPI master supplies SCLK with frequencies up to 26MHz. The logic level is referenced to the voltage on V_L. Data is clocked into the SPI slave interface on the rising edge of SCLK. Data is clocked out of the SPI slave interface on the falling edge of SCLK.
14	$\overline{SS}$	Input	SPI Slave Select Input. The $\overline{SS}$ logic level is referenced to the voltage on V_L. When $\overline{SS}$ is driven high, the SPI slave interface is not selected, the MISO pin is high impedance, and SCLK transitions are ignored. An SPI transfer begins with a high-to-low $\overline{SS}$ transition and ends with a low-to-high $\overline{SS}$ transition.
15	MISO	Output	SPI Serial-Data Output (Master-In Slave-Out). MISO is a push-pull output. MISO is tri-stated in half-duplex mode or when $\overline{SS}$ = 1. The MISO logic level is referenced to the voltage on V_L.
16	MOSI	Input or Input/ Output	SPI Serial-Data Input (Master-Out Slave-In). The logic level on MOSI is referenced to the voltage on V_L. MOSI can also be configured as a bidirectional MOSI/MISO input and output. (See Figure 15.)
17	GPX	Output	General-Purpose Multiplexed Push-Pull Output. The internal MAX3421E signal that appears on GPX is programmable by writing to the GPXB and GPXA bits of the PINCTL (R17) register and the SEPIRQ bit of the MODE (R27) register. GPX indicates one of five signals (see the *GPX section*).
18	INT	Output	Interrupt Output. In edge mode, the logic level on INT is referenced to the voltage on V_L and is a push-pull output with programmable polarity. In level mode, INT is open-drain and active low. Set the IE bit in the CPUCTL (R16) register to enable INT.
20	D-	Input/ Output	USB D- Signal. Connect D- to a USB connector through a 33Ω ±1% series resistor. A switchable 15kΩ D- pulldown resistor is internal to the device.

Pin Description (continued)

PIN	NAME	INPUT/ OUTPUT	FUNCTION
21	D+	Input/ Output	USB D+ Signal. Connect D+ to a USB connector through a 33Ω ±1% series resistor. A switchable 1.5kΩ D+ pullup resistor and 15kΩ D+ pulldown resistor is internal to the device.
22	VBCOMP	Input	V_{BUS} Comparator Input. VBCOMP is internally connected to a voltage comparator to allow the SPI master to detect (through an interrupt or checking a register bit) the presence or loss of power on V_{BUS}. Bypass VBCOMP to ground with a 1.0µF ceramic capacitor. VBCOMP is pulled down to ground with R_{IN} (see *Electrical Characteristics*).
23	V_{CC}	Input	USB Transceiver and Logic Core Power-Supply Input. Connect V_{CC} to a positive 3.3V power supply. Bypass V_{CC} to ground with a 1.0µF ceramic capacitor as close to the V_{CC} pin as possible.
24	XI	Input	Crystal Oscillator Input. Connect XI to one side of a parallel resonant 12MHz ±0.25% crystal and a load capacitor to GND. XI can also be driven by an external clock referenced to V_{CC}.
25	XO	Output	Crystal Oscillator Output. Connect XO to the other side of a parallel resonant 12MHz ±0.25% crystal and a load capacitor to GND. Leave XO unconnected if XI is driven with an external source.
26	GPIN0	Input	General-Purpose Inputs. GPIN7–GPIN0 are connected to V_L with internal pullup resistors. GPIN7–GPIN0 logic levels are referenced to the voltage on V_L.
27	GPIN1		
28	GPIN2		
29	GPIN3		
30	GPIN4		
31	GPIN5		
32	GPIN6		
EP	GND	Input	Exposed Paddle, Connected to Ground. Connect EP to GND or leave unconnected. EP is located on the bottom of the TQFN package. The TQFP package does not have an exposed paddle.

Register Description

The SPI master controls the MAX3421E by reading and writing 26 registers in peripheral mode (see Table 1) or reading and writing 23 registers in host mode (see Table 2). Setting the HOST bit in the MODE (R27) register configures the MAX3421E for host operation. When operating as a USB peripheral, the MAX3421E is register-compatible with the MAX3420E with the additional features listed in Note 1b below Table 1. For a complete description of register contents, refer to the *MAX3421E Programming Guide* on the Maxim website.

A register access consists of the SPI master first writing an SPI command byte followed by reading or writing the contents of the addressed register. All SPI transfers are MSB first. The command byte contains the register address, a direction bit (read = 0, write = 1), and the ACKSTAT bit (Figure 5). The SPI master addresses the MAX3421E registers by writing the binary value of the register number in the Reg4 through Reg0 bits of the command byte. For example, to access the IOPINS1 (R20) register, the Reg4 through Reg0 bits would be as follows: Reg4 = 1, Reg3 = 0, Reg2 = 1, Reg1 = 0, Reg0 = 0. The DIR (direction) bit determines the direction for the data transfer. DIR = 1 means the data byte(s) are written to the register, and DIR = 0 means the data byte(s) are read from the register. The ACKSTAT bit sets the ACKSTAT bit in the EPSTALLS (R9) register (peripheral mode only). The SPI master sets this bit to indicate that it has finished servicing a CONTROL transfer. Since the bit is frequently used, having it in the SPI command byte improves firmware efficiency. The ACKSTAT bit is ignored in host mode. In SPI full-duplex mode, the MAX3421E clocks out eight USB status bits as the command byte is clocked in (Figures 6, 7). In half-duplex mode, these status bits are accessed as register bits.

The first five registers (R0–R4) address FIFOs in both peripheral and host modes. Repeated accesses to these registers freeze the internal register address so that multiple bytes may be written to or read from a FIFO in the same SPI access cycle (while $\overline{SS}$ is low). Accesses to registers R5–R19 increment the internal register address for every byte transferred during the SPI access cycle. Accessing R20 freezes access at that register, accessing R21–R31 increments the internal address, and repeated accesses to R31 freeze at R31.

The register maps in Table 1 and Table 2 show which register bits apply in peripheral and host modes. Register bits that do not apply to a particular mode are shown as zeros. These register bits read as zero values and should not be written to with a logic 1.

Register Map in Peripheral Mode

The MAX3421E maintains register compatibility with the MAX3420E when operating in USB peripheral mode (MAX3421E HOST bit is set to 0 (default)). Firmware written for the MAX3420E runs without modification on the MAX3421E. To support new MAX3421E features, the register set includes new bits, described in Note 1b at the bottom of Table 1.

Register Map in Host Mode

As Table 2 shows, in host mode (HOST = 1), some MAX3420E registers are renamed (for example R1 becomes RCVFIFO), some are not used (shown with zeros), and some still apply to host mode. In addition, 11 registers (R21–R31) add the USB host capability.

b7	b6	b5	b4	b3	b2	b1	b0
Reg4	Reg3	Reg2	Reg1	Reg0	0	DIR	ACKSTAT*

**The ACKSTAT bit is ignored in host mode.*

Figure 5. SPI Command Byte

STATUS BITS (PERIPHERAL MODE)							
b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
SUSPIRQ	URESIRQ	SUDAVIRQ	IN3BAVIRQ	IN2BAVIRQ	OUT1DAVIRQ	OUT0DAVIRQ	IN0BAVIRQ

Figure 6. USB Status Bits Clocked Out as First Byte of Every Transfer in Peripheral Mode (Full-Duplex Mode Only)

STATUS BITS (HOST MODE)							
b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
HXFRDNIRQ	FRAMEIRQ	CONNIRQ	SUSDNIRQ	SNDBAVIRQ	RCVDAVIRQ	RSMREQIRQ	BUSEVENTIRQ

Figure 7. USB Status Bits Clocked Out as First Byte of Every Transfer in Host Mode (Full-Duplex Mode Only)

Table 1. MAX3421E Register Map in Peripheral Mode (HOST = 0) (Notes 1a, 1b)

REG	NAME	b7	b6	b5	b4	b3	b2	b1	b0	acc
R0	**EP0FIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R1	**EP1OUTFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R2	**EP2INFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R3	**EP3INFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R4	**SUDFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R5	**EP0BC**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R6	**EP1OUTBC**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R7	**EP2INBC**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R8	**EP3INBC**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R9	**EPSTALLS**	0	ACKSTAT	STLSTAT	STLEP3IN	STLEP2IN	STLEP1OUT	STLEP0OUT	STLEP0IN	RSC
R10	**CLRTOGS**	EP3DISAB	EP2DISAB	EP1DISAB	CTGEP3IN	CTGEP2IN	CTGEP1OUT	0	0	RSC
R11	**EPIRQ**	0	0	SUDAVIRQ	IN3BAVIRQ	IN2BAVIRQ	OUT1DAVIRQ	OUT0DAVIRQ	IN0BAVIRQ	RC
R12	**EPIEN**	0	0	SUDAVIE	IN3BAVIE	IN2BAVIE	OUT1DAVIE	OUT0DAVIE	IN0BAVIE	RSC
R13	**USBIRQ**	URESDNIRQ	VBUSIRQ	NOVBUSIRQ	SUSPIRQ	URESIRQ	BUSACTIRQ	RWUDNIRQ	OSCOKIRQ	RC
R14	**USBIEN**	URESDNIE	VBUSIE	NOVBUSIE	SUSPIE	URESIE	BUSACTIE	RWUDNIE	OSCOKIE	RSC
R15	**USBCTL**	HOSCSTEN	VBGATE	CHIPRES	PWRDOWN	CONNECT	SIGRWU	0	0	RSC
R16	**CPUCTL**	PULSEWID1	PULSEWID0	0	0	0	0	0	IE	RSC
R17	**PINCTL**	EP3INAK	EP2INAK	EP0INAK	FDUPSPI	INTLEVEL	POSINT	GPXB	GPXA	RSC
R18	**REVISION**	0	0	0	1	0	0	1	0	R
R19	**FNADDR**	0	b6	b5	b4	b3	b2	b1	b0	R
R20	**IOPINS1**	GPIN3	GPIN2	GPIN1	GPIN0	GPOUT3	GPOUT2	GPOUT1	GPOUT0	RSC
R21	**IOPINS2**	GPIN7	GPIN6	GPIN5	GPIN4	GPOUT7	GPOUT6	GPOUT5	GPOUT4	RSC
R22	**GPINIRQ**	GPINIRQ7	GPINIRQ6	GPINIRQ5	GPINIRQ4	GPINIRQ3	GPINIRQ2	GPINIRQ1	GPINIRQ0	RSC
R23	**GPINIEN**	GPINIEN7	GPINIEN6	GPINIEN5	GPINIEN4	GPINIEN3	GPINIEN2	GPINIEN1	GPINIEN0	RSC
R24	**GPINPOL**	GPINPOL7	GPINPOL6	GPINPOL5	GPINPOL4	GPINPOL3	GPINPOL2	GPINPOL1	GPINPOL0	RSC
R25	—	0	0	0	0	0	0	0	0	—
R26	—	0	0	0	0	0	0	0	0	—
R27	**MODE**	0	0	0	SEPIRQ	0	0	0	HOST = 0	RSC
R28	—	0	0	0	0	0	0	0	0	—
R29	—	0	0	0	0	0	0	0	0	—
R30	—	0	0	0	0	0	0	0	0	—
R31	—	0	0	0	0	0	0	0	0	—

Note 1a: The acc (access) column indicates how the SPI master can access the register.
R = read, RC = read or clear, RSC = read, set, or clear.
Writing to an R register (read only) has no effect.
Writing a 1 to an RC bit (read or clear) clears the bit.
Writing a zero to an RC bit has no effect.

Table 1. MAX3421E Register Map in Peripheral Mode (HOST = 0) (Notes 1a, 1b) (Continued)

Note 1b: In peripheral mode, the MAX3421E performs identically to the MAX3420E with the following enhancements:

1) R16 adds the PULSEWID0 and PULSEWID1 bits to control the INT pulse width in edge interrupt mode (see Figure 12.) These bits default to the MAX3420E setting of 10.6µs.
2) R21 adds four more GPIO bits.
3) R22 and R23 add general-purpose input pins to the interrupt system. R24 controls the edge polarity.
4) R27 controls the peripheral/host mode and the SEPIRQ bit.
5) When [GPXB:GPXA] = [1:0] and the bit SEPIRQ = 1 (R27 bit 4), the GPX output replaces the BUSACT signal with a second IRQ pin dedicated to the GPIN pin interrupts.

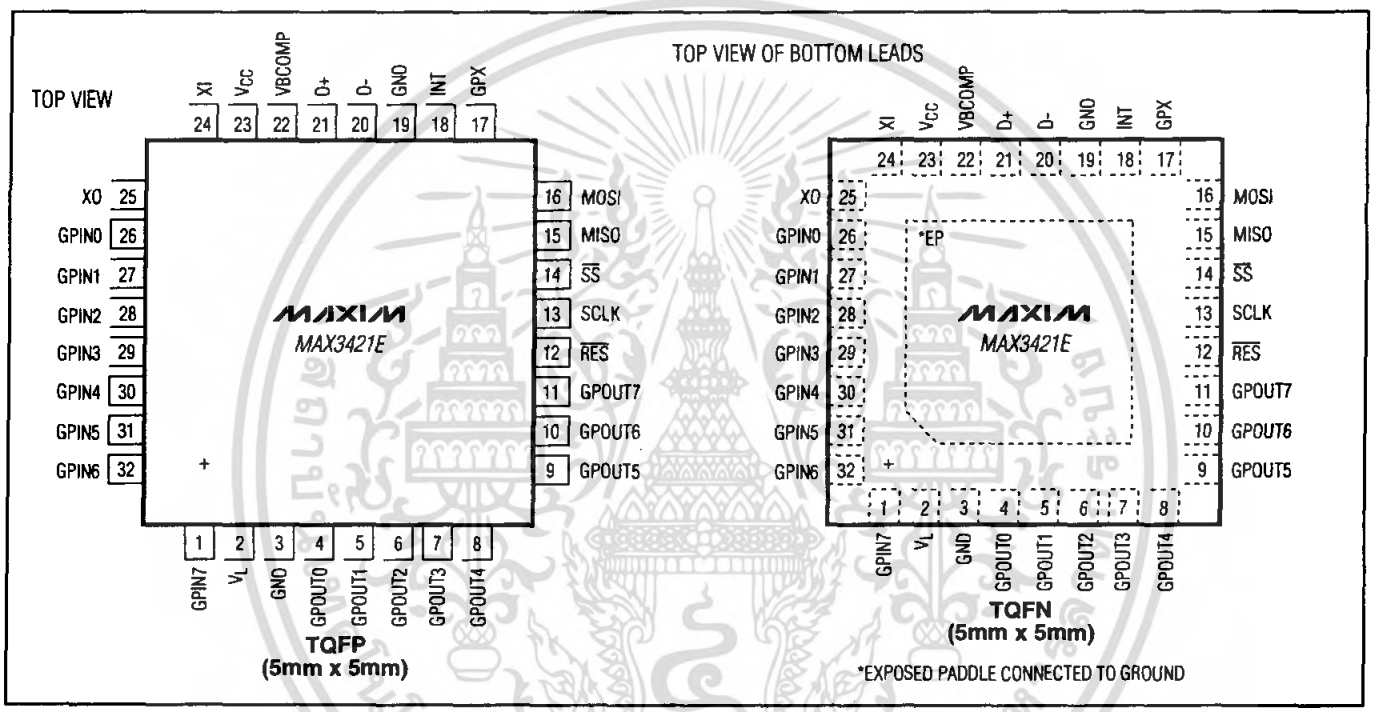
Table 2. MAX3421E Register Map in Host Mode (HOST = 1) (Note 2)

REG	NAME	b7	b6	b5	b4	b3	b2	b1	b0	acc
R0	—	0	0	0	0	0	0	0	0	—
R1	**RCVFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R2	**SNDFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R3	—	0	0	0	0	0	0	0	0	—
R4	**SUDFIFO**	b7	b6	b5	b4	b3	b2	b1	b0	RSC
R5	—	0	0	0	0	0	0	0	0	—
R6	**RCVBC**	0	BC6	BC5	BC4	BC3	BC2	BC1	BC0	RSC
R7	**SNDBC**	0	BC6	BC5	BC4	BC3	BC2	BC1	BC0	RSC
R8	—	0	0	0	0	0	0	0	0	—
R9	—	0	0	0	0	0	0	0	0	—
R10	—	0	0	0	0	0	0	0	0	—
R11	—	0	0	0	0	0	0	0	0	—
R12	—	0	0	0	0	0	0	0	0	—
R13	**USBIRQ**	0	VBUSIRQ	NOVBUSIRQ	0	0	0	0	OSCOKIRQ	RC
R14	**USBIEN**	0	VBUSIE	NOVBUSIE	0	0	0	0	OSCOKIE	RSC
R15	**USBCTL**	0	0	CHIPRES	PWRDOWN	0	0	0	0	RSC
R16	**CPUCTL**	PULSEWID1	PULSEWID0	0	0	0	0	0	IE	RSC
R17	**PINCTL**	EP3INAK	EP2INAK	EP0INAK	FDUPSPI	INTLEVEL	POSINT	GPXB	GPXA	RSC
R18	**REVISION**	0	0	0	1	0	0	1	0	R
R19	—	0	0	0	0	0	0	0	0	—
R20	**IOPINS1**	GPIN3	GPIN2	GPIN1	GPIN0	GPOUT3	GPOUT2	GPOUT1	GPOUT0	RSC
R21	**IOPINS2**	GPIN7	GPIN6	GPIN5	GPIN4	GPOUT7	GPOUT6	GPOUT5	GPOUT4	RSC
R22	**GPINIRQ**	GPINIRQ7	GPINIRQ6	GPINIRQ5	GPINIRQ4	GPINIRQ3	GPINIRQ2	GPINIRQ1	GPINIRQ0	RC
R23	**GPINIEN**	GPINIEN7	GPINIEN6	GPINIEN5	GPINIEN4	GPINIEN3	GPINIEN2	GPINIEN1	GPINIEN0	RSC
R24	**GPINPOL**	GPINPOL7	GPINPOL6	GPINPOL5	GPINPOL4	GPINPOL3	GPINPOL2	GPINPOL1	GPINPOL0	RSC
R25	**HIRQ**	HXFRDNIRQ	FRAMEIRQ	CONNIRQ	SUSDNIRQ	SNDBAVIRQ	RCVDAVIRQ	RSMREQIRQ	BUSEVENTIRQ	RC
R26	**HIEN**	HXFRDNIE	FRAMEIE	CONNIE	SUSDNIE	SNDBAVIE	RCVDAVIE	RSMREQIE	BUSEVENTIE	RSC
R27	**MODE**	DPPULLDN	DMPULLDN	DELAYISO	SEPIRQ	SOFKAENAB	HUBPRE	SPEED	HOST = 1	RSC
R28	**PERADDR**	0	b6	b5	b4	b3	b2	b1	b0	RSC
R29	**HCTL**	SNDTOG1	SNDTOG0	RCVTOG1	RCVTOG0	SIGRSM	BUSSAMPLE	FRMRST	BUSRST	LS
R30	**HXFR**	HS	ISO	OUTNIN	SETUP	EP3	EP2	EP1	EP0	LS
R31	**HRSL**	JSTATUS	KSTATUS	SNDTOGRD	RCVTOGRD	HRSLT3	HRSLT2	HRSLT1	HRSLT0	R

Table 2. MAX3421E Register Map in Host Mode (HOST = 1) (Note 2) (Continued)

Note 2: *The acc (access) column indicates how the SPI master can access the register.*
R = read; RC = read or clear; RSC = read, set, or clear; LS = load-sensitive.
Writing to an R register (read only) has no effect.
Writing a 1 to an RC bit (read or clear) clears the bit.
Writing a zero to an RC bit has no effect.
Writing to an LS register initiates a host operation based on the contents of the register.

Pin Configurations



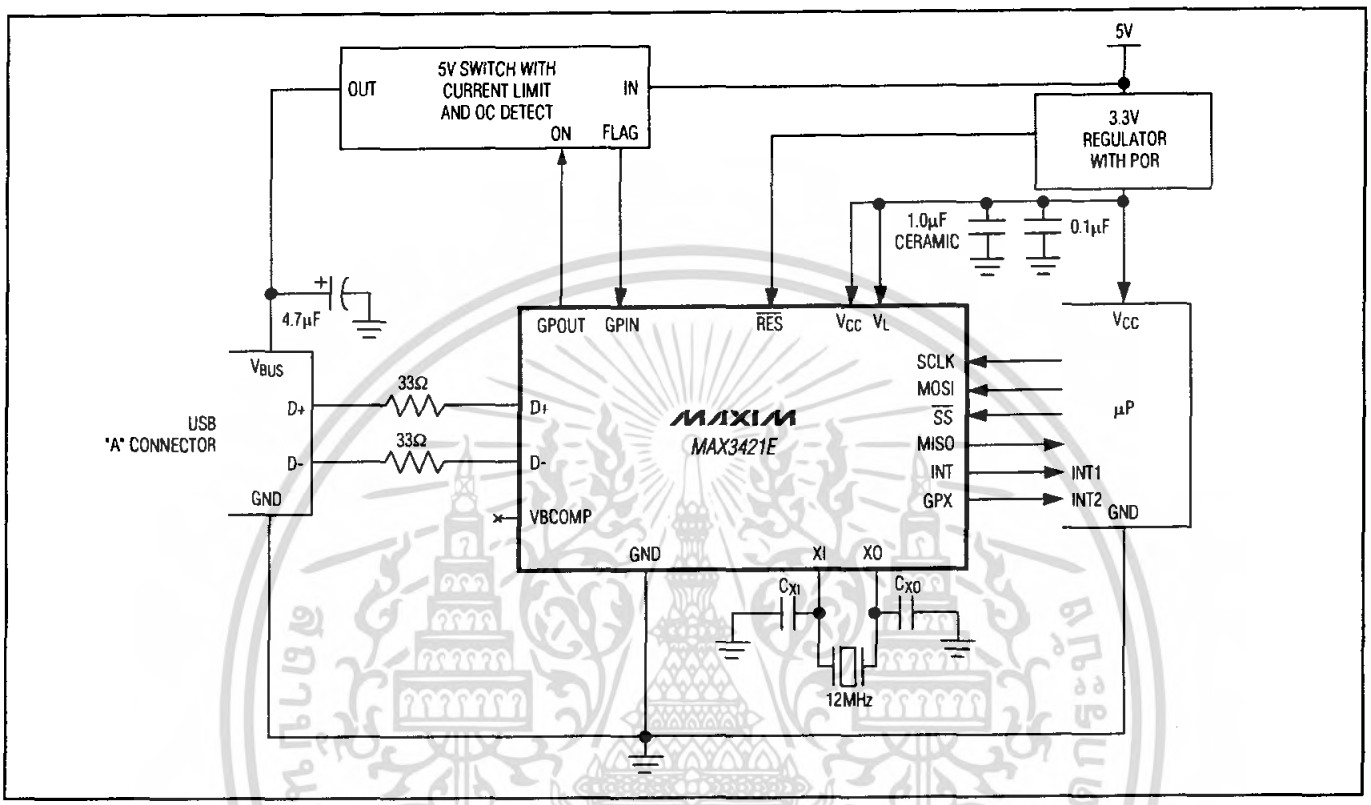


Figure 21. MAX3421E in a Host Application

eight GPOUT pins turns the V_{BUS} switch on and off. Seven MAX3421E GPIN and GPOUT pins are available to the system.

Short-Circuit Protection

The MAX3421E withstands V_{BUS} shorts to D+ and D- on the USB connector side of the 33Ω series resistors.

ESD Protection

D+, D-, and VBCOMP possess extra protection against static electricity to protect the devices up to ±15kV. The ESD structures withstand high ESD in all operating modes: normal operation, suspend mode, and powered down. VBCOMP and V_{CC} require 1µF ceramic capacitors connected to ground as close to the pins as possible. D+, D-, and VBCOMP provide protection to the following limits:

- ±15kV using the Human Body Model
- ±8kV using the Contact Discharge method specified in IEC 61000-4-2
- ±12kV using the IEC 61000-4-2 Air Gap Method

ESD Test Conditions

ESD performance depends on a variety of conditions. Contact Maxim for a reliability report that documents test setup, test methodology, and test results.

Human Body Model

Figure 22 shows the Human Body Model, and Figure 23 shows the current waveform generated when discharged into a low impedance. This model consists of a 100pF capacitor charged to the ESD voltage of interest, which then discharges into the test device through a 1.5kΩ resistor.

IEC 61000-4-2

The IEC 61000-4-2 standard covers ESD testing and performance of finished equipment. It does not specifically refer to integrated circuits. The major difference between tests done using the Human Body Model and IEC 61000-4-2 is a higher peak current in IEC 61000-4-2, due to lower series resistance. Hence, the ESD withstand voltage measured to IEC 61000-4-2 generally is lower than that measured using the Human Body