เครื่องเล่น MP 3

MP 3 Player

โดย

นาย ภาณุวัฒน์ เพชรศรี

นาย ยมนา อินทมา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

เครื่องเล่น MP 3

MP 3 Player

โดย

นาย ภาณุวัฒน์ เพชรศรี รหัส 47015255

นาย ยมนา อินทมา รหัส 47015256

อาจารย์ที่ปรึกษา

ดร. กิติพล ชิดสกุล

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องเล่น MP 3

ผู้จัดทำ

1. นาย ภาณุวัฒน์ เพชรศรี
2. นาย ยมนา อินทมา

.....อาจารย์ที่ปรึกษา

(ดร. กิติพล ชิตสกุล)

เครื่องเล่น MP 3

นาย ภาณุวัฒน์ เพชรศรี รหัส 47015255
นาย ยมนา อินทมา รหัส 47015256
ดร. กิติพล ชิตสกุล อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

โครงงานนี้เป็นการออกแบบและสร้าง เครื่องเล่น MP 3 กับไฟล์ที่บันทึกบน SD Card ซึ่งมีขนาดเล็กและสามารถเก็บข้อมูลได้เป็นจำนวนมาก โดยใช้ไมโครคอนโทรลเลอร์ตระกูล AVR ดึงข้อมูลจาก SD Card และมีอุปกรณ์ถอดรหัสเป็นตัวแปลงไฟล์ MP 3 ให้เป็นเสียง ซึ่งการทำงานของเครื่องเล่น MP 3 นั้นจะใช้ไมโครคอนโทรลเลอร์นำข้อมูลที่เป็นไฟล์เสียงออกมาจาก SD Card มาพักไว้ในหน่วยความจำในตัว แล้วส่งข้อมูลที่ได้ไปยังอุปกรณ์ที่ใช้ในการถอดรหัสไฟล์จากข้อมูลเสียงให้เป็นเสียง

MP3 PLAYER

Mr. Panuwaat Phetsri ID. 47015255
Mr. Yommana Intama ID. 47015256
Dr. Kitiphol Chitsakul Advisor
Educational Year 2006

Abstract

This project is a design and construction of a MP3 player by using SD card which is a popular device at present. Since a SD card has small size and can store a lot of data, it is a good idea to use this device with the player. We use an AVR Microcontroller to take data from the SD card and pass to a decoder for transforming MP3 files to voice.

กิตติกรรมประกาศ

อันดับแรกต้องขอขอบคุณ ดร. กิติพล ชิตสกุล ซึ่งท่านเป็นอาจารย์ที่ปรึกษาของโครงงานนี้ซึ่งได้ให้คำแนะนำและคอยชี้แนะแนวทางในการดำเนินงานตลอดระยะเวลาที่ผ่านมา

ขอบพระคุณคุณพ่อ-แม่ซึ่งท่านทั้งสองนั้นได้คอยอบรมสั่งสอนดูแลและเป็นกำลังใจให้ตลอดมาจนข้าพเจ้ามีวันนี้

สุดท้ายขอขอบใจเพื่อนๆทุกคนที่คอยช่วยเหลือทุกๆด้าน

(ภาณุวัฒน์ เพชรศรี)

นาย ภาณุวัฒน์ เพชรศรี

(ยมนา อินทมา)

นาย ยมนา อินทมา

สารบัญ

สารบัญ(ต่อ)

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่1
บทนำ

เนื่องจากเทคโนโลยีในปัจจุบันได้พัฒนาไปอย่างรวดเร็วและ อุปกรณ์ SD CARD เป็นอีกชิ้นหนึ่งที่ได้ถูกสร้างขึ้นมาและเป็นอุปกรณ์ที่ได้รับความนิยมเป็นอย่างมาก ซึ่งลักษณะพิเศษของอุปกรณ์ชนิดนี้คือ มีขนาดเล็กสามารถนำติดตัวไปไหนมาไหนได้สะดวกและสามารถเก็บข้อมูลได้เป็นจำนวนมาก

1.1 วัตถุประสงค์

1. เพื่อศึกษาหลักการทำงานและใช้งาน SD CARD
2. เพื่อศึกษาโครงสร้างและหลักการทำงานของไมโครคอนโทรลเลอร์ ชนิด AVR เบอร์ ATMEGA 32
3. เพื่อศึกษาการติดต่อกันระหว่าง SD CARD กับ ไมโครคอนโทรลเลอร์ ชนิด AVR เบอร์ ATMEGA 32
4. ศึกษาหลักการเขียนโปรแกรมลงในตัว ไมโครคอนโทรลเลอร์ ชนิด AVR เบอร์ ATMEGA 32 เพื่อดึงข้อมูลจากอุปกรณ์ SD CARD

1.2 ขอบเขตของโครงงาน

สร้างเครื่องเล่นไฟล์เสียง MP3 ที่บันทึกในอุปกรณ์ SD CARD ได้

1.3 ประโยชน์ที่คาดว่าจะได้รับคือ

1. ทำให้สามารถใช้โปรแกรม AVR studio 4
2. เข้าใจหลักการทำงานของ SD CARD

1.4 โครงสร้างของรายงาน

รายงานนี้เป็นจะนำเสนอผลงานที่ได้ปฏิบัติเพื่อออกแบบเครื่องเล่น MP3 โดยแบ่งนำเสนอเป็นบทดอนดังนี้

บทที่ 1 เป็นการกล่าวโดยทั่วไปเกี่ยวกับโครงงานถึงวัตถุประสงค์ของโครงงาน ขอบเขตของโครงงานตลอดจนประโยชน์ที่จะได้รับ

บทที่ 2 เป็นหลักการทำงานและโครงสร้างของอุปกรณ์ SD CARD

บทที่ 3 เป็นหลักการทำงานและโครงสร้างของไมโครคอนโทรเลอร์

บทที่ 4 เป็นการออกแบบวงจรและผลการทดลอง

บทที่ 5 บทสรุป

บทที่2

โครงสร้างและหลักการทำงานของ SD CARD

ในปัจจุบันหน่วยความจำแบบ SD Card (Secure Digital Memory Card) นี้เป็นหน่วยความจำที่ได้รับความนิยมมากเพราะเป็นอุปกรณ์ที่สามารถพกพาได้สะดวก ซึ่งหน่วยความจำแบบ SD Card นี้จะมีฟังก์ชั่นการทำงานคล้ายกับหน่วยความจำชนิด MMC (Multi Media Card) ซึ่งภายในหน่วยความจำแบบ SD Card นี้จะมีไมโครคอนโทรลเลอร์, แฟลชเมมโมรี่คอนโทรล (erase, read, write and error control) ซึ่งการโอนย้ายข้อมูลโดยปกติระหว่างเมมโมรี่กับโฮสคอนโทรลนี้จะโอนย้ายข้อมูล 512 ไบต์ต่อบล็อกโดยอัตโนมัติ ดังนั้นมันจะดูเหมือนฮาร์ดดิสก์ทั่วๆไปนั่นเองซึ่งในจุดนี้เราสามารถนำไปประยุกต์ต่างๆได้ โดยทั่วไปไฟล์ระบบนั้นจะเป็นแบบ FAT 16/32 เท่านั้นสำหรับการทำการแบ่งส่วน โดย FAT 32 นั้นความจุของหน่วยความจำของการ์ดจะต้องมีค่ามากกว่า 2 Gbyte

รูปที่ 2 ลักษณะคอนแทคของหน่วยความจำแบบ SD Card และ MMC Card

จากรูปที่ 2 ด้านล่างแสดงช่องคอนแทคซึ่งใช้ในการติดต่อสื่อสารของหน่วยความจำแบบ SD Card และ MMC Card ซึ่งหน่วยความจำแบบ MMC Card จะมีช่องติดต่อ 7 คอนแทคส่วนหน่วยความจำ SD Card นั้นจำเพิ่มมาอีก 2 คอนแทค ในการโอนย้ายอนุกรมข้อมูลระหว่างโฮสกับการ์ดนั้นจะกระทำในสัญญาณนาฬิกา

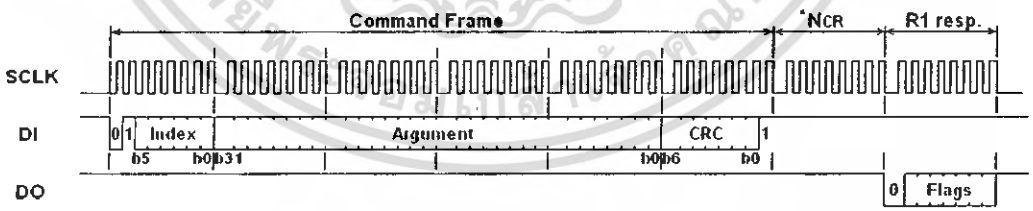
การติดต่อระหว่างโฮสกับหน่วยความจำแบบ SD Card นั้นจำมีอยู่ 2 Mode ด้วยกันคือ Mode SD MODE และ SPI MODE ซึ่งในการติดต่อนี้จะทำการติดต่อหน่วยความจำเป็นแบบ SPI Mode

2.1 SPI MODE

SPI Mode เป็นการกระทำทางเลือกหนึ่งในการติดต่อจากโฮสภายนอก ซึ่งหน่วยความจำจะสามารถติดต่อโดยช่องทางของ SPI Port หรือ GPIO Port ที่มีภายในไมโครคอนโทรลเลอร์ทั่วไป เพราะฉะนั้น SPI Mode จะเหมาะสมกับการประยุกต์ใช้งานในที่นี้ ซึ่งการติดต่อหน่วยความจำนี้จะทำการเลือกโหมดการทำงานของ SPI Mode เป็นโหมด 0 (positive clock, front edge latch, back edge shift) ดังนั้น SPI MODE 0 จะเหมาะสมสำหรับการตั้งค่าสำหรับติดต่อกับหน่วยความจำแบบ SD Card และ SPI Mode 3 ก็ทำงานได้ดีเช่นเดียวกัน

2.1.1 Command and Response

ใน SPI Mode ทิศทางข้อมูลจะอยู่บนsignal line ซึ่งการโอนย้ายอนุกรมข้อมูลจะกำหนดเป็นไบต์ ส่วนเฟรมของ command เป็นการส่งผ่านจากโฮสไปยังการ์ดซึ่งมีความยาวของขนาดเท่ากับ 6 ไบต์ต่อ 1 คำสั่ง command เมื่อเฟรมของ command เป็นการส่งผ่านไปยังการ์ดแล้ว ค่า Response (R1,R2,R3)ของ command ที่ส่งไปนั้นจะถูกส่งตอบกลับมายังโฮส ซึ่งโฮสนั้นจะต้องอ่านค่าResponse มาตลอดจนกว่าจะส่งค่า Response ที่ถูกต้องของ command กลับมาเพราะการส่งผ่านข้อมูลของการ์ดมายังโฮสนั้นจะต้องถูกขับจากสัญญาณนาฬิกาที่โฮสกำเนิดมานั่นเอง command response time (NCR) นั้นจะมีโครงสร้าง 0 ถึง 8 ไบต์ สัญญาณ CS นี้จะต้องควบคุมให้เป็น 0 ในระหว่างการติดต่อ Command , Response และ Data transfer ยังคงกระทำอยู่ ส่วนไบต์ของ CRC นั้นจะเป็นส่วนประกอบของชุดเฟรมของ Command ซึ่งเป็นทางเลือกใน SPI mode สำหรับการตรวจสอบความถูกต้องของข้อมูลที่ส่งออกไป



รูปที่ 2.1 ลักษณะของ Command Frame

2.1.2 SPI Command Set

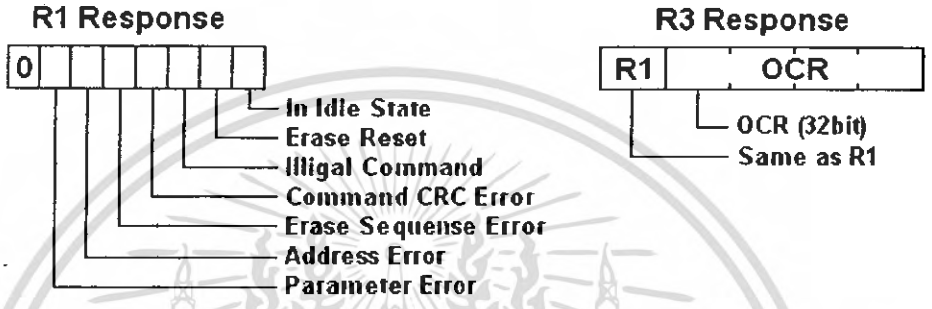
แต่ละ Command เป็นการแสดงเป็นเครื่องหมายในการย่อของ GO_IDLE_STATE หรือ CMD<n>, โดยที่ <n> นั้นเป็นตัวเลขใดๆของ command index และค่าของ n นี้จะสามารถเป็นได้ 0 ถึง 63 ซึ่งรูปแบบของตาราง command จะอธิบาย command ที่ใช้ทั่วๆไปในการ อ่าน เขียนและการกำหนดค่าเริ่มต้นของการ์ด สำหรับ command อื่นๆสามารถที่จะดูได้ที่ Data Sheet ของ MMCA and SDCA.

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.
ACMD41(*1)	None(0)	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Application specific command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.
*1:ACMD<n> means a command sequense of CMD55-CMD<n>.					

ตารางที่ 2.1 แสดง Command ที่ใช้ทั่วไปในการอ่านเขียนและกำหนดค่าเริ่มต้นของการ์ด

2.1.3 SPI Response

รูปแบบของ Response นั้นจะมีอยู่ 3 รูปแบบ คือ R1, R2 และ R3 ซึ่งจะขึ้นอยู่กับแต่ละ command ที่ใช้นั้นๆ โดยไบต์ของResponse R1 จะเป็นการส่งค่ากลับมาของ command ส่วนใหญ่



รูปที่ 2.2 ไบต์ของ R1 Response และ R3 Response

ซึ่งในแต่ละบิทสำหรับ R1 Response ที่แสดงนั้นถ้าค่าของไบต์ R1 = 0x00 จะหมายถึงว่าติดต่อกับหน่วยความจำแบบ SD Card นี้สำเร็จ เมื่อไหร่ก็ตามที่เกิดข้อผิดพลาดเกิดขึ้นบิทของ R1 นั้นจะเซตขึ้นดังที่สอดคล้องกับรูป ค่าของ R3 Response นั้นจะสำหรับ CMD 58 เท่านั้น มันจะมีไบต์แรกเหมือนกับ R1 และจะต่อท้ายเป็นขบวนที่บรรจุของ OCR

บาง Command จะใช้เวลานานมากกว่าจำนวนของ NCR ที่กำหนดดังนั้นโฮสคอนโทรลเลอร์นั้นจะต้องรอค่า response กลับมาด้วยโดยการส่งค่า 0xff ตลอดจนกว่าจะรับ response กลับมาได้แล้ว

2.2 ขั้นตอนการเซตค่าเริ่มต้นสำหรับ SPI Mode

หลังจาก Power on reset หน่วยความจำแบบ SD Card นั้นจะเข้าสู่โหมดการทำงาน native mode ซึ่งมีรูปแบบของขั้นตอนดังนี้

2.2.1 Power On

หลังจากแหล่งจ่ายมาถึงค่าแรงดันที่ 2.2 โวลท์ เราจะต้องเซตค่าของ DI และ CS ให้เป็น 1 และป้อนพัลส์ให้มากกว่า 74 พัลส์ไปยัง SCLK ซึ่งจะทำให้การ์ดนั้นสามารถที่จะรับ command ต่างๆได้

2.2.2 Software Reset

ส่ง CMD0 ร่วมกับ CS เป็น0 เพื่อที่จะไปรีเซตการ์ดซึ่งการ์ดนั้นจะตรวจสอบสัญญาณ CS เมื่อCMD0 ถูกตรวจพบจะทำการตรวจสอบ CS ว่าเป็น 0 หรือไม่ถ้าเป็น 0 การ์ดจะเข้าสู่ SPI Mode ดังนั้น CMD0 จะต้องส่ง native command ด้วยและในส่วนของ CRC นั้นจะต้องมีค่าของ CRC ที่ถูกต้องด้วย ทันทีที่การ์ดเข้าสู่ SPI Mode การตรวจสอบค่าของ CRC จะไม่ถูกนำมาพิจารณาอีกต่อไป ดังนั้นในการส่ง CMD0 จะกำหนดค่าของ CRC ให้เท่ากับ 0x95 ได้เลย เมื่อ CMD0 นั้นถูกตรวจสอบเรียบร้อยแล้วการ์ดนั้นจะเข้าสถานะว่าง (idle state) และผลของค่า response R1 นั้นจะมีค่าเท่ากับ 0x01 นั่นคือบิทของ idle state จะถูกเซตขึ้นซึ่งแสดงว่า SD Card นี้อยู่ในสถานะว่าง

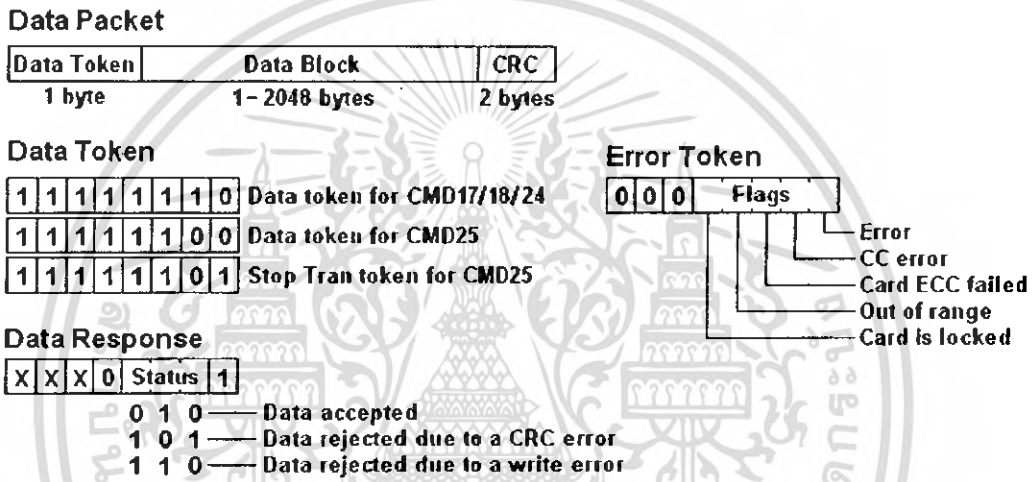
2.2.3 Initialization

ในสถานะว่าง SD Card นั้นจะยอมรับเฉพาะ CMD0 , CMD1 และ CMD58 เท่านั้น commandอื่นๆจะถูกปฏิเสธ แต่เมื่อการ์ดนั้นตรวจพบ CMD1 การ์ดจะเริ่มพร้อมที่จะทำงาน ซึ่งโฮสนั้นจะต้องส่ง CMD1 ซ้ำไปเรื่อยๆและคอยตรวจสอบค่า response นี้ด้วยไปจนกว่าค่าของ response R1 จะตอบกลับมาเป็น 0x00 ซึ่งนั้นก็คือ CMD1 นั้นเป็นการเคลียร์ค่าของ R1 นั้นเอง เพื่อที่จะทำการส่งค่า command ต่อๆไป ในขบวนการ initial การ์ดนี้จะใช้เวลาหลายๆร้อยมิลลิเซคอนด์ขึ้นอยู่กับขนาดของการ์ดนั้นๆ ดังนั้นเราจะต้องพิจารณาหาค่าเวลานี้เองหรือใช้ค่า data sheet ที่ผู้ผลิตให้มา หลังจากทำการ initial เสร็จก็จะสามารถอ่านและเขียนข้อมูลทั่วไปได้แล้ว ในช่วงเวลานี้จะสามารถอ่านค่ารีจิสเตอร์ OCR และ CID ได้เพื่อที่จะหาย่านของแรงดันที่กระทำและขนาดของการ์ดหรือคุณสมบัติอื่นๆถ้าต้องการได้

2.3 การโอนย้ายของข้อมูล

2.3.1 Data Packet and Data Response

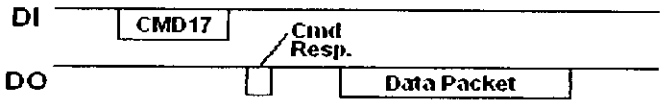
ในการติดต่อเกี่ยวกับการ โอนย้ายของข้อมูล 1 block หรือมากกว่า 1 block นั้น จะต้องทำการส่งหรือรับข้อมูลหลังจาก command response ซึ่งการ โอนย้ายข้อมูลนั้นจะประกอบไปด้วยกลุ่มของข้อมูลคือ data Token 1 ไบต์ Data Block 1-2048 ไบต์และ CRC อีก 2 ไบต์



รูปที่ 2.3 แสดงกลุ่มข้อมูลที่จะทำการโอนย้าย

2.3.2 การอ่านแบบ Single Block

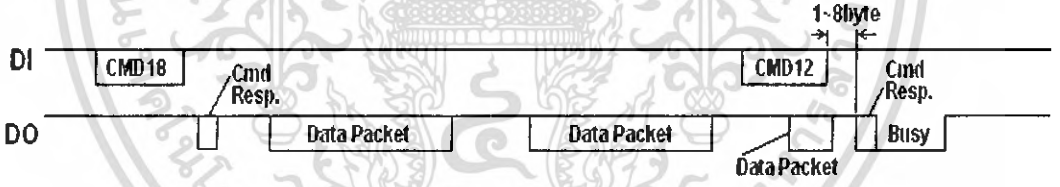
เป็นการระบุตำแหน่งที่จะเริ่มทำการอ่านในหน่วยของไบต์ เมื่อ CMD17 ถูกยอมรับแล้วการกระทำการอ่านข้อมูล data blockจะส่งไปยังโฮส หลังจาก data token ถูกตรวจพบ โดยโฮสจะได้รับตามมาคือ data block และ CRC 2 ไบต์ โดยปกติขนาดของบล็อกจะเท่ากับ 512 ไบต์ซึ่งเราสามารถเปลี่ยนแปลงได้โดย CMD16 ซึ่งถ้าเกิดข้อผิดพลาดในระหว่างการอ่านข้อมูล error token จะถูกส่งมาแทน



รูปที่ 2.4 แสดงการอ่านแบบ Single block

2.3.3 การอ่านแบบ Multiple Block

การอ่านข้อมูลแบบ Multiple Block เป็นคำสั่งในการอ่าน multiple block เป็นลำดับ จากการระบุที่อยู่ เมื่อจำนวนของการโอนย้าย block จะไม่เจาะจงก่อน command นี้ การติดต่อจะเริ่มต้นที่การเปิดและจบการอ่าน multi block การอ่านนั้นจะต่อเนื่องจนกระทั่งหยุดโดย CMD12 ไบต์ที่ได้รับโดยทันทีทันใดนั้นจะตามหลัง CMD12 มันจะถูกตัดทิ้งก่อนได้รับ response ของ CMD12



รูปที่ 2.5 แสดงการอ่านข้อมูลแบบ Multiple Block

2.3.4 การอ่านค่ารีจิสเตอร์ CSD และ CID

การอ่านค่ารีจิสเตอร์ทั้งสองนี้จะเหมือนกับการอ่านแบบ single block ยกเว้นขนาดของความยาวของ block CID และ CSD จะถูกส่งไปยังโฮสเท่ากับ 16 ไบต์ของ data block

2.3.5 รายละเอียดขาสัญญาณของ SD CARD

การเชื่อมต่อกับ SD CARD สามารถทำได้สองแบบ ก็คือ การเชื่อมต่อแบบ SD MODE และ การเชื่อมต่อแบบ SPI MODE ดังรายละเอียดตามตารางต่อไปนี้

- รายละเอียดขาสัญญาณต่างๆ เมื่อใช้การเชื่อมต่อในโหมด SD MODE

Pin No.	Name	Type[1]	Description
SD Mode			
1	CD/DAT3[2]	I/O[3], PP	Card detect/Data line [Bit 3]
2	CMD	I/O, PP	Command/Response
3	V_{SS1}	S	Supply voltage ground
4	V_{DD}	S	Supply voltage
5	CLK	I	Clock
6	V_{SS2}	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
9	DAT2	I/O, PP	Data line [Bit 2]

ตารางที่ 2.2 รายละเอียดขาสัญญาณต่างๆ เมื่อใช้การเชื่อมต่อในโหมด SD MODE

- รายละเอียดขาสัญญาณต่างๆ เมื่อใช้การเชื่อมต่อในโหมด SPI MODE

SPI Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	V_{SS1}	S	Supply voltage ground
4	V_{DD}	S	Supply voltage
5	CLK	I	Clock
6	V_{SS2}	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV[4]	---	Reserved
9	RSV[5]	---	Reserved

ตารางที่ 2.3 รายละเอียดขาสัญญาณต่าง ๆ เมื่อใช้การเชื่อมต่อในโหมด SPI MODE

บทที่ 3

ทฤษฎีไมโครคอลโทรเลอร์ AVR เบอร์ ATMEGA 32 และทฤษฎีที่เกี่ยวข้อง

AVR เป็นไมโครคอนโทรลเลอร์(MCU)ที่ได้รวบรวมอุปกรณ์สนับสนุนการทำงานของ CPU ไว้มากมายอาทิเช่น Analog to Digital , SPI , UART , Timer , Counter , PWM ซึ่งอุปกรณ์สนับสนุนการทำงานเหล่านี้ทำให้ MCU สามารถทำงานได้กว้างและใช้อุปกรณ์ต่อร่วมจากภายนอกน้อยมาก และสามารถประมวลคำสั่งได้ภายใน 1 clock ในบทนี้จะนำเสนอข้อมูลบางส่วนที่เป็นการทำงานภายในของ AVR - MCU แนะนำคุณสมบัติและขาต่อใช้งานของไมโครคอนโทรลเลอร์ สถาปัตยกรรมภายในและรีจีสเตอร์ใช้งานทั่วไป ตำแหน่ง I/O รีจีสเตอร์สถานะและการใช้งาน EEPROM การรีเซ็ตและการอินเตอร์รัพท์ การสื่อสารอนุกรม การเปรียบเทียบสัญญาณอนาลอกและการแปลงสัญญาณอนาลอกเป็นดิจิตอล การทำงานของพอร์ตอินพุต/เอาท์พุตการทำงานของTimer / Counter & Watch dog และการใช้กลุ่มคำสั่งต่าง ๆ

3.1 คุณสมบัติและขาต่อใช้งานของไมโครคอนโทรลเลอร์

1. สถาปัตยกรรมภายในถูกออกแบบให้ใช้สถาปัตยกรรมแบบ RISE (Reduce Instruction Set Computer) RISE คือ ทำให้การประมวลผลมีความเร็ว 1 คำสั่ง / 1 Clock หรือ CPU สามารถประมวลคำสั่งได้ I MIPS / MHz
2. มีคำสั่งในการควบคุมการทำงานของไมโครคอนโทรลเลอร์จำนวน 118 คำสั่ง
3. หน่วยความจำ 3 หน่วยความจำบันทึก PROGRAM MEMORY ขนาด 32 Kbyte
4. หน่วยความจำแบบ EEPROMสำหรับบันทึก DATA MEMORY ขนาด 1024 Byte
5. หน่วยความจำแบบ RAM ขนาด 2K Byte
6. ระบบการเปลี่ยนสัญญาณ ANALOG TO DIGITAL ขนาด 10 บิท จำนวน 8 CHANNEL
7. กลุ่มรีจีสเตอร์ใช้งานทั่วไปขนาด 8 บิท จำนวน 32 ตัว
8. พอร์ตอินพุตและเอาท์พุตขนาด 8 บิท จำนวน 4พอร์ต
9. ระบบการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัส (UART) 1 CHANEL
10. ระบบการสื่อสารข้อมูลดิจิตอลแบบซิงโครนัส (SPI) 1 CHANEL
11. ความถี่สัญญาณนาฬิกา 0 - 16 MHz (ATMEGA 32)
12. ระบบการรีเซ็ตแบบอัตโนมัติเมื่อเริ่มจ่ายกระแสไฟฟ้าเข้าไมโครคอนโทรลเลอร์ (Power on reset)
13. ระบบการกำเนิดความถี่สัญญาณแบบ PWM จำนวน 4 CHANNEL (ATMEGA 32)
14. ระบบการตรวจจับระดับสัญญาณอนาลอก (Analog Comparator)

15. 6 SLEEP MOD: IDEL, POWER SAVE, POWER DOWN, ADC Noise, Reduction, Standby and Extended standby

16. ระบบการป้องกันการ COPY ข้อมูลภายในหน่วยความจำ (LOCK FOR SOLFWARE SECURITY)

17. ระบบตรวจจับการทำงานผิดพลาดของ CPU (WATCHDOG TIMER WITH ON-CHIP OSCILATOR)

18. ระบบการอินเตอร์รัพท์จากภายนอก (EXTERNAL INTERRUPT)

19. TIMER/COUNTER ขนาด 16 บิท 1 CHANNEL

20. TIMER/COUNTER ขนาด 8 บิท 2 CHANNEL

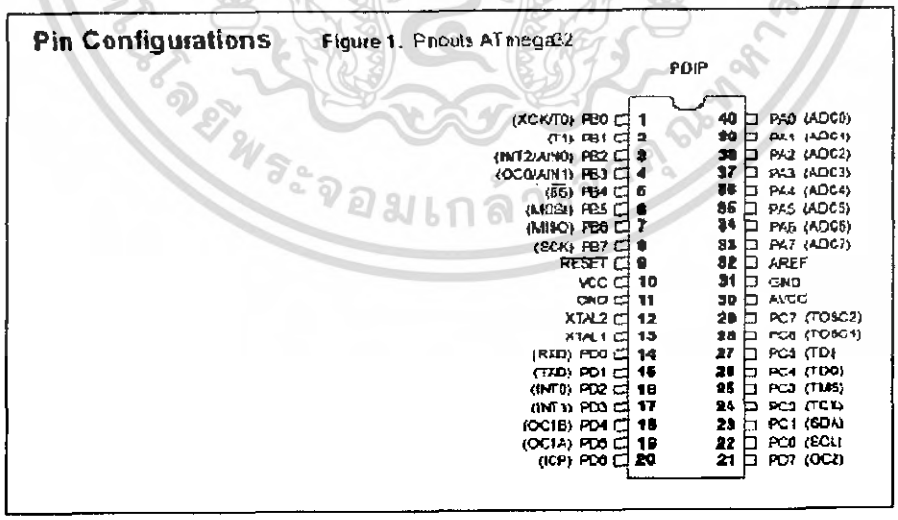
21. Vcc: 4.5 - 5.5 for ATMEGA 32

3.2 รายละเอียด

AT MEGA 32 เป็นไมโครคอนโทรลเลอร์ขนาด 10 บิทที่มีสถาปัตยกรรมแบบ RISE (reduce instruction set computer) ซึ่งทำให้การประมวลผลมีความเร็ว 1 คำสั่ง/ 1 clock หรือ CPU สามารถประมวลคำสั่งได้ 1 MIPS / MHz

3.2.1 โครงสร้างภายนอกและตำแหน่งขา

ภายในประกอบด้วยรีจิสเตอร์ใช้งานทั่วไปขนาด 8 บิท จำนวน 32 ตัวซึ่งแต่ละตัวจะต่อเข้ากับALU โดยตรง ทำให้การประมวลผล ต่อ 1 คำสั่งมีความเร็วกว่า CPU ที่มีสถาปัตยกรรมแบบCISC



รูป 3.1 แสดงโครงสร้างภายนอกและตำแหน่งขา

3.2.2 รายละเอียดของขาสัญญาณและการใช้งาน

VCC คือ ขาจ่ายไฟให้กับ CPU และ GND คือ กราวด์

Port A (PA7..PA0)

เป็นพอร์ต 2 ทิศทางขนาด 8 บิท โดยสามารถกำหนดให้แต่ละขาของพอร์ต สามารถ PULL UP ภายในแยกจากกันซึ่งสามารถรับกระแส SINK 20mA โดยพอร์ต A ยังใช้เป็นขาอินพุตเพื่อรับสัญญาณอนาลอกในส่วนของการแปลงสัญญาณ ANALOG TO DIGITAL

Port B (PB7..PB0)

เป็นพอร์ต 2 ทิศทางขนาด 8 บิท โดยสามารถกำหนดให้แต่ละขาของพอร์ต สามารถ PULL UP ภายในอิสระแยกจากกันซึ่งแต่ละขาสามารถรับกระแส SINK20mA และยังถูกนำไปใช้งานอื่นๆ อีก

Port C (PC7..PC0)

เป็นพอร์ต 2 ทิศทางขนาด 8 บิท โดยสามารถกำหนดให้แต่ละขาของพอร์ต สามารถ PULL UP ภายในอิสระแยกจากกันซึ่งแต่ละขาสามารถรับกระแส SINK20mA และยังถูกนำไปใช้งานอื่นๆอีก

Port D (PD7..PD0)

เป็นพอร์ต 2 ทิศทางขนาด 8 บิท โดยสามารถกำหนดให้แต่ละขาของพอร์ต .สามารถ PULL UP ภายในอิสระแยกจากกันซึ่งแต่ละขาสามารถรับกระแส SINK20mA และยังถูกนำไปใช้งานอื่นๆอีก

Reset คือ ขารีเซ็ต

XTAL 1 เป็นขาอินพุตของ OSE

XTAL 2 เป็นขาเอาท์พุตของ OSE

AVcc ใช้จ่ายไฟให้กับวงจร Analog to Digital

AREF เป็นขาแรงดันอ้างอิงที่ใช้งานในส่วนของวงจร Analog to Digital

AGND เป็นขากราวด์ของวงจร Analog to Digital

3.3 ฟังก์ชั่น ADC

การแปลงสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอลนั้นมีความจำเป็นมากเพราะว่าในตัวไมโครคอนโทรลเลอร์ไม่สามารถประมวลผลแบบอนาล็อกได้มันจะประมวลผลแบบดิจิตอลเท่านั้นดังนั้นเราจึงจำเป็นต้องมีการแปลงสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอล

ปกติใน CPU ของ AVR - ATMEGA32 นั้นจะมีฟังชั่น ADC อยู่ภายในตัวไอซี ดังนั้นไม่จำเป็นต้องใช้ไอซี ADC ต่อภายนอก สำหรับฟังก์ชั่น ADC นี้สามารถรับสัญญาณอนาล็อกได้สูงสุด 8 Channel โดยรับสัญญาณเข้ามาทาง พอร์ต A เราสามารถเลือกใช้ฟังก์ชั่นนี้ทำการแปลงสัญญาณอนาล็อกที่ละ Channel อย่างต่อเนื่อง หรือจะให้ทำการแปลงสัญญาณเฉพาะ channel ที่เราต้องการก็ได้เช่นกัน โดยสัญญาณดิจิตอลที่แปลงได้จะมีความละเอียด 10 บิต โดยการรับสัญญาณแต่ละขาของพอร์ท A โดยจะมีวงจร SAMPLE AND HOLD เพื่อช่วยให้สัญญาณอนาลอกที่รับเข้ามาเพื่อแปลงเป็นสัญญาณดิจิตอลที่มีระดับสัญญาณคงที่โดยปกติการใช้งานฟังก์ชั่นนี้เราจำเป็นต้องจัดแรงดัน AVCC AREF และ AGND ให้กับฟังก์ชั่นด้วย

คุณสมบัติ

1. 10bit resolution
2. 0.5LSB integral non-linearity
3. ±2 LSB Absolute Accuracy
4. 65-260 μσ Conversion Time
5. Up to 15KSPS at Maximum Resolution
6. 8 Multiplexed Single Ended Input Channels
7. 7 Differential Input Channels
8. 2 Differential Input Channels with Gain of 10xand 200x (1)
9. Optional left adjustment for ADC result readout
10. 0-Vcc ADC Input Voltage Range
11. Selectable 2.56V ADC Reference Voltage
12. Free Running or Single conversion Complete
13. Sleep mode noise Canceller

การทำงาน

ในส่วนของการแปลงสัญญาณ อนาลอก เป็นดิจิตอล สามารถทำได้ 2 mode คือ

1. Single Conversion Mode
2. Free Running Mode

การทำงาน Single Conversion Mode ผู้ใช้ต้องเป็นผู้กำหนดการใช้งานขึ้นเอง แต่ในส่วนของ Free Running Mode วงจร Analog to digital จะเป็นตัวจัดการอ่านข้อมูลและเก็บใน ADC Data Register ซึ่งบิต ADFR ใน Register ADCSR จะเป็น บิตที่ใช้เลือก Mode การใช้งานของวงจร Analog to digital สำหรับการกำหนดให้วงจร Analog to digital ทำงานนั้น สามารถทำได้โดยการ

เซ็ตบิต ADEN ในรีจีสเตอร์ ADCHRAให้เป็น 1 โดยบิตนี้จะเป็น 1 ไปตลอดจนกระทั่ง Conversion ของสัญญาณจะเรียบร้อยแล้วจึงทำให้บิตนี้เป็น 0โดยอัตโนมัติ แต่ถ้าเป็นการเปลี่ยน Channel ของการแปลงสัญญาณขณะที่ Channel เดิมยัง Conversion อยู่วงจร Analog to digital จะ Conversion สัญญาณ Channel เดิมให้เสร็จก่อนแล้วจึง Conversion สัญญาณChannel ถัดไป โดยข้อมูลที่ได้จากการแปลงสัญญาณอนาล็อกเป็นดิจิตอลจะเก็บไว้ในรีจีสเตอร์ ADCH และADCL

3.4 ฟังก์ชั่น PWM

การมอดูเลตความกว้างของพัลส์ (PWM) เป็นเทคนิคสำคัญที่ใช้ในการปรับปรุงสมรรถนะของอินเวอร์เตอร์ ดังนั้นการศึกษาเกี่ยวกับ PWM จึงมีความจำเป็นอย่างยิ่งสำหรับอินเวอร์เตอร์ เพื่อที่อินเวอร์เตอร์จะได้มีสมรรถนะและประสิทธิภาพในการทำงานที่ดีขึ้น เนื่องจากว่าPWMเป็นฟังก์ชั่นการทำงานหนึ่งในโหมด PWM ของ Timer/Counter ที่อยู่ภายใน AVR – ATMEGA32 ดังนั้นในหัวข้อต่อไปจะแนะนำเกี่ยวกับการทำงานของ Timer/Counter ของ AVR – ATMEGA32

Timer /Counter

ภายใน AVR - ATMEGA32 จัดให้มี Timer/Counter 3 ชุดโดยจัดเป็น Timer/Counter ขนาด 8บิท 2 ชุดและ Timer/Counter ขนาด 16 บิท 1 ชุดดังนี้คือ Timer/Counter2 และ Timer/Counter0 และTimer/Counter1 ซึ่ง Timer/Counter2 สามารถรับสัญญาณ Clock จากภายนอกซึ่งเป็น Option ที่จะนำ Timer/Counter2 มาทำเป็น RTC โดยใช้ XTAL ที่มีค่าความถี่เท่ากับ 32.768KHz มาเป็นฐานเวลาและ Timer/Counter0 และ Timer/Counter1 ใช้วงจร Rescaling ขนาด 10 บิทร่วมกัน ส่วน Timer/Counter2ใช้วงจร Rescaling แยกออกต่างหาก

แนะนำการใช้งาน Timer/Counter แต่ละประเภท

Timer/Counter0

โครงสร้างของ Timer/Counter 0 ขนาด 8 บิท ซึ่งสามารถเลือกสัญญาณ Clock ได้จาก CLK (Clock ของระบบ) หรือสัญญาณ Clock ของระบบที่ถูกหาร (Rescaling) หรือ สัญญาณจากภายนอก โดยการใช้งานจะอธิบายในรีจีสเตอร์ TCCRO และ TIFR ส่วนสัญญาณควบคุมสามารถทราบรายละเอียดได้จากรีจีสเตอร์ TCCRO ซึ่งการควบคุมการอินเตอร์รับจะควบคุมได้จาก รีจีสเตอร์ TIMSK เมื่อ Timer/Counter0 ได้รับสัญญาณจากภายนอก ซึ่งสัญญาณดังกล่าวจะ ซิงโครไน (Synchronized) กับสัญญาณนาฬิกาภายในCPUโดย Timer/Counter 0 จะเป็นวงจรนับขึ้นที่สามารถเขียนและอ่านข้อมูลได้ตลอดเวลา โดยเมื่อทำการเขียนข้อมูลลงTIMER/COUNTER 0 ในขณะที่มีสัญญาณ Clock จะทำให้ TIMER/COUNTER 0 นับค่าต่อเนื่องจากค่าที่ถูกเขียนลงไป

Timer/Counter 1

จะมีขนาด 16 บิท โดยสามารถเลือกสัญญาณนาฬิกาได้จาก CLK หรือสัญญาณที่ได้รับการหารจาก CLK (Rescaling) ซึ่งการหยุด Timer/Counter 1 จะอธิบายไว้ในรีจิสเตอร์ TCCR1A (Timer/Counter1ControlRegister)และTCCR1Bโดยแฟร์กที่แสดงสถานะต่าง (Overflow,Comparemath,Captureeven) ส่วนสัญญาณควบคุมจะอธิบายไว้ในรีจิสเตอร์ TCCR1A และTCCR1Bการควบคุมสัญญาณอินเตอร์รัพต์จะควบคุมได้จากรีจิสเตอร์TIMSK (TIMER/COUNTER INTERRUPT MASK REGISTER)

เมื่อ TIMER1/COUNTER1 จะประกอบด้วยส่วนของการเปรียบเทียบเอาท์พุต (Output CompearFunction) 2 ฟังก์ชั่นโดยจะใช้รีจิสเตอร์ OCR1A (Output Compare Register 1 A) และ OCR1B (Output Compare Register 1B) เป็นส่วนของการเก็บค่าข้อมูลของการเปรียบเทียบ TIMER1/COUNTER1 จะสามารถเลือกใช้ฟังก์ชั่น PWM ได้ทั้ง 8,9 และ 10 บิท

The Timer/Counter Control Register

Bits 7, 6-COM1A1, COM1A0: Compare Output Mode 1 A, bit 1 and 0 บิท COM1A1 และCOM1A0เป็นบิทที่ใช้ในการกำหนดลักษณะของสัญญาณที่เกิดขึ้นที่ขาOC1Aเมื่อ Timer/Counter1 เกิด Compare Match ซึ่งเมื่อใช้ฟังก์ชั่น Output Compare Match ของ Timer/Counter1 จะต้องควบคุมให้ขา OC1A มีสถานะเป็นเอาท์พุต

Bit 5, 4-COM1B1, COM1B0: Compare Output Mode 1 B, bit 1 and 0 บิท Com1A1 และ COM1A0เป็นบิทที่ใช้ในการกำหนดลักษณะของสัญญาณที่เกิดขึ้นที่ขา OC1A เมื่อTimer/Counter1 เกิด Compare Match ซึ่งเมื่อใช้ฟังชั่น Output Compare Match ของ Timer/Counter1 จะต้องควบคุมให้ขา OC1A มีสถานะเป็นเอาท์พุต

Bit 3...2-Res: Reserved bits ในส่วนของ AT mega32 จะสงวนบิทในกลุ่มนี้ไว้

Bit 1...0 - PWM11, PWM10: Pulse Width Modulator Select Bit เป็นบิทที่ใช้ในการกำหนดการทำงานของ PWM

The Timer/Counter1 Control Register B-TCCR11B

Bit 7-ICN1: Input Capture 1 Noise Canceller (4 CKs) บิทนี้เป็นบิทที่กำหนดให้ Input Capture 1 Noise Canceller ทำงานหรือไม่ทำงาน โดยเมื่อบิทนี้เป็น 1 จะเป็นการกำหนดให้ Input Capture 1 Noise Canceller ทำงาน แต่เมื่อบิทนี้เป็น 0 จะเป็นการกำหนดไม่ให้ Input Capture 1 Noise Canceller ทำงานชุด Noise Canceller จะถูกกำหนดให้ ทำงานโดยการ Sampling สัญญาณที่เข้ามาที่ ชุด Input Capture 1 โดยสัญญาณ Sampling แรกจะเริ่มที่ขอบแรกของสัญญาณขาขึ้นหรือขาลงขึ้นอยู่กับการกำหนดในบิท ICES1 โดยชุด Noise Canceller จะ Sampling ด้วยความถี่เท่ากับ

ความถี่ของ XTAL ซึ่งจะ Sampling ทั้งหมด 4 ครั้ง โดยลอจิกที่ได้จากการ Sampling จะต้องมีลอจิกเดียวกันกับลอจิกที่กำหนดในบิท ICES1

Bit 6-ICES1: Input Capture 1 Edge Select เป็นบิทที่ใช้กำหนดให้ชุด Input Capture 1 จะต้องDetect ถ้าบิท ICES1 เซ็ต เป็น 1 จะเป็นการกำหนดให้ชุด Input Capture1 ทำหน้าที่ Detect สัญญาณที่ขอบขาขึ้น แต่ถ้าบิท ICES 1 ถูกเคลียร์เป็น 0 จะเป็นการกำหนดให้ชุด Input Capture 1 ทำหน้าที่ Detect สัญญาณที่ขอบขาลง

Bit 5, 4-RES: Reserved bits บิทนี้ถูกสงวนไว้

Bit 3: CTC1: Clear Timer1/Counter1 on Compare Match บิทนี้เป็นที่ใช้ในการกำหนดว่าเมื่อเกิด Output Compare แล้วจะให้เกิดการนับต่อไปหรือจะให้มีการรีเซ็ตค่าให้เป็น 00000 แล้วจึงทำการนับต่อไป โดยถ้าเป็นบิทนี้เป็น 1 จะเป็นการกำหนดให้มีการรีเซ็ตค่าให้เป็น 0 เมื่อเกิดการ Output Compare แต่ถ้าบิทนี้เคลียร์เป็น 0 จะเป็นการกำหนดให้มีการนับค่าต่อเมื่อเกิด Output Compare

Bit 2, 1, 0-CS12, CS11, CS10: Clock Select1 bit 2, 1 and 0 เป็นบิทที่ใช้ในการเลือกสัญญาณ clock

The Timer/Counter In Capture Register – ICR1H AND ICR1L

เป็นรีจีสเตอร์ขนาด 16 ที่ใช้เก็บค่า Timer/Couner1 ที่อยู่ในรีจีสเตอร์ TCNT1 เมื่อ Input Capture สามารถ Detect ได้ เมื่อ Input Capture สามารถ Detect สัญญาณได้ตามที่กำหนดในบิท ICES1 จะทำให้CPU โหลดค่าในรีจีสเตอร์ TCNT1 ลงในรีจีสเตอร์ และในเวลาเดียวกับบิท ICF1 จะเซ็ตเป็น 1 โดยการอ่านค่าจากรีจิสเตอร์ ICR1 ของ CPU จะใช้รีจีสเตอร์ TEMP เป็นรีจีสเตอร์พักข้อมูล ซึ่งการใช้รีจีสเตอร์ TEMP ช่วยในการอ่านข้อมูลเพื่อให้ค่าที่อยู่ในรีจีสเตอร์ ICR1H และ ICR1L เสมือนถูกอ่านออกมาพร้อมกัน การอ่านค่าจากรีจีสเตอร์ ICRI จะต้องอ่านค่าจากรีจีสเตอร์ ICR1L ก่อนโดยเมื่อ CPU อ่านค่าจาก ICR1L จะทำให้ค่าในรีจีสเตอร์ ICR1H ถูกโหลดลงในรีจีสเตอร์ TEMPเมื่อ CPU อ่านค่าจาก ICRIH จะทำให้ค่าในรีจีสเตอร์TEMP ถูกส่งให้ CPU

3.5 การใช้งาน Timer/Counter1 ในโหมด PWM

การทำงานในโหมด PWM ของ Timer/Counter1 จะสามารถเลือกใช้งานได้ 8, 9 หรือ 10 บิทโดยเอาท์พุตที่ได้จะออกที่ขา PD5 (OC1A) และขา PD (OC1B) ในการทำงาน Timer/Counter1 จะนับขึ้นและนับลงซึ่งจะนับขึ้นจาก 0000 ถึงค่าสูงสุดและจะนับจากค่าสูงสุดลงมาที่ 0000แล้วจึงนับขึ้นอีกครั้ง

เมื่อค่าใน Timer/Counter1 เท่ากับค่าในรีจีสเตอร์ OCR1A หรือ OCR1B จะทำให้ขา PD5 (OC1A)/PD1 (OC1B) เปลี่ยนแปลงตามที่กำหนดในบิท COM1A/CoM1A0 หรือCoM1B/COM1B0

เมื่อ OCR1 มีค่าเท่ากับ 0000 หรือค่าสูงสุดจะทำให้เอาท์พุตขา OC1A/OCA1B มีลอจิกเป็น LOW หรือ HIGH ตามที่กำหนดในบิท COM1A1/COM1A0 หรือ COM1B1/COM1B0และเมื่อ Timer/Counter1เกิด Overflow และค่าการนับเป็น 0000 จะทำให้บิท TOV1 เซ็ตเป็น 1

Timer2&Counter

เป็น Timer / Counter ขนาด 8 บิทต่อไปจะกล่าวถึงรายละเอียดของรีจิสเตอร์ใช้งานใน Timer/Counter 2

The Timer/Counter 2 Control Register – TCCR2

Bit 7 - Res:Reserved Bit ใน AT90S4434/8535 บิทนี้สงวนไว้

Bit 6 - PWM2: Pulse Width Modulator Enable เป็นบิทที่ใช้ Enable ให้โหมด PWM ใน Timer/Counter2 ให้ทำงาน โดยถ้าบิทนี้เซ็ตเป็น 1 จะเป็นการกำหนดให้โหมด PWM ถูก Enable ให้ทำงานแต่ถ้าบิทนี้ถูกเคลียร์เป็น 0 จะเป็นการ Disable ไม่ให้โหมด PWM ใน Timer/Counter2 ทำงาน

Bit 5, 4 - Com21, Com20: Compare Output Mode, bit 1 and 0 เป็นบิทที่ใช้กำหนดลักษณะสัญญาณที่ขา PD7 (OC2) เมื่อ Timer/Counter2 ทำงานในโหมด Compare โดยเมื่อ Compare Output Match จะทำให้ขา PD7 (OC2) เป็นไปตามที่กำหนดในบิท Com21 และ Com20

Bit 3-CTC2: Clear Timer/Counter on Compare Match เป็นบิทที่ใช้กำหนดให้ Timer2/Counter2 ทำการ RESET ค่าเป็น 00 หลังจากที่ค่าในรีจิสเตอร์ TCNT2มีค่าเท่ากับค่าที่ตั้งไว้ในรีจิสเตอร์ OCR หรือ Compare Output Match ถ้าบิทนี้เซ็ตเป็น 1 จะทำให้ Timer/Counter2 รีเซ็ต

Bits 2, 1, 0-CS22, CS21, CS20: Clock Select bit 2, 1 and 0 เป็นบิทใช้ในการกำหนด ค่า Rescaling

3.6 การบีบอัดข้อมูลแบบเอมเป็ก (MPEG)

เอมเป็ก(MPEG) นั้นย่อมาจาก Moving Picture Expert Group ซึ่งเป็นชื่อของกลุ่มคนที่ร่วมมือกันสร้างมาตรฐานสากล(International Standard) เพื่อที่จะใช้ในการเข้ารหัสข้อมูลภาพและเสียงที่อยู่ในรูปแบบของสัญญาณดิจิตอล ต่อจากนั้นได้มีกลุ่มนักวิศวกรชาวยุโรปได้มีการพัฒนาการบีบอัดไฟล์ในรูปแบบของเอ็มเป็กต่อจนกลายมาเป็นการบีบอัดไฟล์แบบ "เอ็มพี3" (MP3 : MPEG-1 Audio Layer 3) ซึ่งใช้กันอย่างแพร่หลายในปัจจุบันภายใต้มาตรฐาน ISO/IEC ซึ่งเป็นมาตรฐานที่นำมาใช้ในการเข้ารหัสหรือถอดรหัสข้อมูลของตนเองได้โดยไม่ต้องขออนุญาติหรือจ่ายค่าลิขสิทธิ์ให้กับผู้ใด โดยมาตรฐานของเอ็มพี 3 คือ ISO/IEC 11172-3

มาตรฐานการบีบอัดไฟล์แบบเอ็มเป็กนั้นแบ่งออกได้เป็นกลุ่มๆ ตามชนิดข้อมูลที่ถูกเข้ารหัสและการนำไปใช้งานซึ่งแบ่งออกได้เป็น 5 กลุ่มดังนี้

1. MPEG-1 ข้อมูลภาพและเสียง ใช้ในระบบวิดีโอซีดี และเสียงเพลง
2. MPEG-2 เข้ารหัสข้อมูลภาพและเสียง ใช้ในระบบโทรทัศน์ดิจิตอลและดีวีดี
3. MPEG-4 เป็นส่วนขยายของ MPEG-1 เพื่อรับรูปแบบมัลติมีเดียต่างๆ เช่น 3D หรือการเข้ารหัสที่มีประสิทธิภาพมากขึ้น MPEG-4 แบ่งออกเป็นหลายส่วนตามหน้าที่แต่ละส่วน และทาง MPEG จะปล่อยให้ผู้ผลิตซอฟท์แวร์เป็นผู้พัฒนาโปรแกรมที่ใช้จริงๆ เอง ไม่จำเป็นต้องตาม MPEG-4 เต็มชุดก็ได้ พัฒนาได้เป็นบางส่วนก็พอ (แบบเดียวกับเอ็มพี 3 ที่หยิบแต่ส่วนออดิโอไปทำ)
4. MPEG-7 ไม่ใช่มาตรฐานเกี่ยวกับภาพและเสียงเหมือนอันอื่นๆ แต่เป็นมาตรฐานที่ใช้เก็บข้อมูลเกี่ยวกับตัวมีเดีย (metadata) เช่น หนังแผ่นนี้ชื่ออะไร หรือถ้าหนังเล่นมาถึงตอนนี้ ให้เล่นเพลงนี้ พร้อมขึ้นซับไทเทิลไฟล์นี้ เป็นต้น เก็บข้อมูลเป็น XML
5. MPEG-21 เป็นมาตรฐานที่กำหนดขึ้นว่าด้วยเรื่องเกี่ยวกับ Multimedia Framework

นอกจากทั้ง 5 กลุ่มนี้ยังมีกลุ่มแบบอื่นๆอีกแต่ไม่ได้ใช้กันในปัจจุบันคือ

MPEG-3 เป็นมาตรฐานที่เตรียมใช้กับ HDTV (High Definition Television หรือโทรทัศน์ความละเอียดสูง) แต่สุดท้ายไม่ได้ใช้ เพราะพบว่าแค่เทคโนโลยี MPEG-2 ที่มีอยู่เดิมเพียงพอสำหรับ HDTV แล้ว

3.6.1 หลักการและพื้นฐานการใช้งาน เอ็มพี 3

เอ็มพี 3 คือรูปแบบไฟล์ที่ใช้กันอย่างแพร่หลายในปัจจุบัน โดยมีรูปแบบการบีบอัดไฟล์อยู่ในรูปแบบมาตรฐานแบบ MPEG-1 ซึ่งเป็นรูปแบบของการเข้ารหัสข้อมูลแบบข้อมูลภาพและเสียง โดยตัวของ MPEG-1 นี้จะแบ่งออกเป็น 3 เลเยอร์ (layer) ตามความสามารถและความซับซ้อนในการเข้ารหัสข้อมูลโดยเลเยอร์ที่ 1 จะมีความซับซ้อนในการเข้ารหัสน้อยทำให้สามารถบีบอัดข้อมูลได้น้อย และในทางกลับกันเลเยอร์ที่ 3 ก็มีความซับซ้อนในการเข้ารหัสข้อมูลมากที่สุดนั่นคือเอ็มพี 3 จะมีรูปแบบการเข้ารหัสที่มีความซับซ้อนสูงและสามารถที่จะบีบอัดข้อมูลได้มากจึงสามารถจัดเก็บข้อมูลได้เป็นจำนวนมาก แต่ถึงอย่างไรรูปแบบในการบีบอัดของทั้ง 3 เลเยอร์ก็มีรูปแบบมาตรฐานเหมือนกันทั้งหมด โดยความเร็วในการบีบอัดและลักษณะที่แตกต่างกันของทั้ง 3 เลเยอร์แสดงได้ดังตาราง

มาตรฐานการบีบอัด	อัตราส่วน	ความเร็วในการส่งข้อมูล
MPEG-1 Layer 1	1:4	384 กิโลบิต/วินาที
MPEG-1 Layer 2	1:6 ถึง 1:8	256-192 กิโลบิต/วินาที
MPEG-1 Layer 3	1: 10 ถึง 1:12	128-115 กิโลบิต/วินาที

ตารางที่ 3.1 แสดงอัตราการบีบอัดข้อมูลและความเร็วในการส่งข้อมูลจากเครื่องอ่านของข้อมูลที่ถูกบีบอัดตามมาตรฐาน MPEG-1

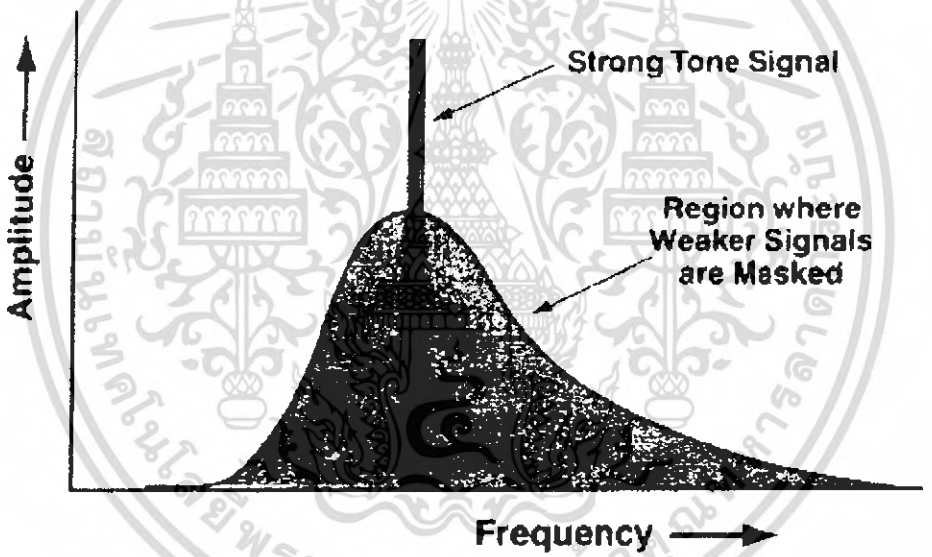
มาตรฐานการบีบอัด	อัตราส่วนการบีบอัด	เวลาที่ใช้ในการถอดรหัส
MPEG-1 Layer 1	4:1	19 ns
MPEG-1 Layer 2	6:1	35 ns
MPEG-1 Layer 3	12:1	59 ns

ตารางที่ 3.2 แสดงเวลาที่ใช้ในการแปลงข้อมูล

3.6.2 การเข้ารหัสแบบ MPEG

การเข้ารหัสแบบ MPEG นั้นโดยทั่วไปแล้วจะทำการตัดข้อมูลเสียงที่จัดเก็บบางส่วนออก แต่สามารถคงรายละเอียดของเสียงที่ได้ยินไว้เท่าเดิม ทั้งนี้เนื่องมาจากการลดขนาดข้อมูลตามมาตรฐาน MPEG นั้นใช้พฤติกรรมการได้ยินเสียงของมนุษย์มาเป็นเครื่องมือในการลดขนาดข้อมูล อย่างที่รู้กันโดยทั่วไปว่าหูของมนุษย์นั้นมีขีดจำกัดในด้านการรับฟัง โดยความถี่ที่มนุษย์สามารถรับฟังได้คือความถี่ระหว่าง 20-20,000 เฮิรตซ์ ซึ่งถ้าอยู่เกินช่วงนี้ไปหูของมนุษย์จะไม่สามารถได้ยินเสียงนั้นได้ แต่ถึงอย่างไรหูของมนุษย์ก็ใช่ว่าจะมีความไวต่อทุกๆความถี่ที่ได้ยินเท่ากันทุกคน ความถี่ 2-4 กิโลเฮิรตซ์ หูของมนุษย์มีความไวต่อเสียงมากที่สุดและเมื่อสังเกตที่ความถี่ต่างๆ จะพบว่าที่ความถี่สูงมากหรือต่ำมากมีความจำเป็นที่จะต้องใช้ความดังมากๆเพื่อที่จะทำให้มนุษย์ได้ยิน จากคุณสมบัติการรับฟังทั้งหมดที่กล่าวมานั้นเรียกรวมกันว่า "ไซโคอคูสติกโมเดล" (Psychoacoustic Model) ซึ่งถือเป็นเครื่องมือที่สำคัญมากในการนำมาเป็นตัวกรองเพื่อลดขนาดในการบีบอัดข้อมูลเสียงตามมาตรฐาน MPEG ซึ่งมีลำดับกระบวนการในการบีบอัดดังนี้

1. นำข้อมูลเสียงดิจิตอลป้อนเข้าฟิลเตอร์ (Filter) เพื่อแยกเสียงออกเป็นช่วงความถี่ย่อยๆ (Sub-bands) ซึ่งมีความกว้างเท่ากับย่านความถี่วิกฤต จำนวน 32 ช่องความถี่เรียกขั้นตอนนี้ว่า Sub-band Filtering
2. ใช้ไซโคอคูสติกโมเดลเป็นเครื่องมือในการวิเคราะห์ข้อมูลส่วนที่ไม่มีผลต่อการได้ยินของมนุษย์ออกไป โดยพิจารณาระหว่างช่วงความถี่ 2 ช่วงที่ติดกันและพิจารณาย่อยลงไปในแต่ละช่วงความถี่ด้วย
3. ถ้าวิเคราะห์แล้วพบว่าเสียงช่วงใดไม่มีผลกระทบต่อการได้ยิน ให้ตัดข้อมูลส่วนนั้นออกไปและไม่นำไปเข้ารหัสในส่วนถัดไป
4. นำข้อมูลที่เหลือมาเข้ารหัสซึ่งจะมีวิธีที่แตกต่างกันขึ้นอยู่กับแต่ละเลเยอร์



รูปที่ 3.2 ขอบเขตของช่วงความถี่ที่ถูกบังในการใช้ไซโคอคูสติก (Psychoacoustic)

3.6.3 โครงสร้างของข้อมูล เอ็มพี 3

ข้อมูลที่ถูกบีบอัดตามมาตรฐาน เอ็มพี 3 นั้นจะอยู่ในลักษณะของเฟรมข้อมูล โดยในแต่ละเฟรมข้อมูลจะมีส่วนประกอบภายในอยู่ 4 ส่วนคือ

- หัวข้อมูล (Header) เป็นข้อมูลขนาด 32 บิตแสดงลักษณะทั่วไปของไฟล์นั้นๆ
- ส่วนตรวจสอบความผิดพลาด (CRC) เป็นข้อมูลขนาด 16 บิต ใช้ตรวจสอบข้อมูลภายในเฟรมว่าถูกต้องหรือไม่จะมีหรือไม่ก็ได้

- ข้อมูลข้างเคียง (Side Information) มีขนาด 17 หรือ 32 ไบต์ (17 ไบต์สำหรับระบบโมโน 32 ไบต์สำหรับระบบอื่นๆ) เป็นส่วนที่เก็บองค์ประกอบที่ใช้ในการถอดรหัส
- ข้อมูลหลัก (Main Data) มีความยาวขึ้นอยู่กับอัตราการส่งข้อมูล (Baud rate) และอัตราการสุ่มข้อมูลในการแปลงกลับเป็นสัญญาณอะนาลอก (Sampling Frequency)

หัวข้อมูล (Header)	ตรวจสอบความผิดพลาด (CRC)	ข้อมูลข้างเคียง (Side Information)	ข้อมูลหลัก (Main Data)

รูปที่ 3.3 แสดงรูปแบบข้อมูลเอ็มเป็กเลเยอร์ 3

3.7 FAT: File Allocation Table

เป็นระบบไฟล์ที่ใช้ในระบบปฏิบัติการในตระกูล Microsoft และเป็นระบบไฟล์ที่มีพัฒนาการมาอย่างต่อเนื่อง ระบบไฟล์ในตระกูลนี้มีลักษณะคือ เป็นการกำหนดหมายเลขให้กับทุก ๆ คลัสเตอร์ (Cluster) ในแต่ละ พาร์ทิชั่น(Partition) แล้วทำการสร้างตารางที่มีจำนวนช่องตามจำนวน คลัสเตอร์ นั้น เพื่อเป็นการระบุสถานที่หรือ คลัสเตอร์ ที่ทำการเก็บข้อมูลของไฟล์แต่ละไฟล์ และมีตารางอีกตารางหนึ่งที่เรียกว่า "ไดเร็คทอรี่"(Directory) สำหรับเก็บข้อมูลรายละเอียดของไฟล์ เช่น คุณสมบัติต่าง ๆ และ หมายเลข คลัสเตอร์ เริ่มต้นที่เก็บตัวข้อมูลจริง ๆ ระบบไฟล์ FAT มีหลายรุ่นดังต่อไปนี้

1. FAT 12 เป็นระบบไฟล์ที่ใช้ใน Floppy Disk และ Hard disk ที่มีขนาดไม่เกิน 16 MBs หมายเลข คลัสเตอร์ มีขนาด 12 บิต จึงสามารถอ้างถึง คลัสเตอร์ ได้เพียง 4096 คลัสเตอร์ เท่านั้น
2. FAT 16 ใช้ตัวเลขขนาด 16 บิต ในการกำหนดหมายเลข คลัสเตอร์ จึงกำหนดหมายเลขได้ 65536 หมายเลข ระบบไฟล์นี้ มีใช้ในระบบปฏิบัติการของ Microsoft ทุกรุ่น พาร์ทิชั่น ที่จะใช้ระบบไฟล์นี้ได้ ต้องมีขนาดไม่เกิน 2GBs. FAT 16 ได้รับการปรับปรุงให้มีความสามารถมากขึ้นใน Windows 95 เพื่อให้สามารถใช้งานกับไฟล์ที่มีชื่อยาวได้ไม่เกิน 256 ตัว เรียก FAT 16 รุ่นนี้ว่า Virtual FAT หรือ VFAT
3. FAT 32 ระบบไฟล์ระบบนี้จะใช้หมายเลขขนาด 28 บิต ซึ่งตามทฤษฎีจะสามารถกำหนด คลัสเตอร์ ได้มากถึง 268,435,456 คลัสเตอร์ และสามารถใช้กับ พาร์ทิชั่น ที่

มีขนาดใหญ่ได้ถึง 2 Terabytes ระบบไฟล์แบบ FAT32 นี้มีใช้ใน Windows 95 OSR2 ขึ้นไป แต่ใช้ไม่ได้ใน Windows NT

คุณลักษณะ	FAT12	FAT16	FAT32
ใช้สำหรับ	Floppies and very small hard disk volumes	Small to moderate-sized hard disk	Medium-sized to very large hard disk volumes
ขนาดของการเข้ารหัสข้อมูล	12 bits	16 bits	28 bits
จำนวนมากสุดของคลัสเตอร์	4,086	65,526	~268,435,456
ขนาดของคลัสเตอร์ที่ใช้งาน	0.5 KB to 4 KB	2 KB to 32 KB	4 KB to 32 KB
ขนาดความจุสูงสุด	16,736,256	2,147,123,200	about 2^41

ตารางที่ 3.3 แสดงค่าข้อจำกัดต่าง ๆ ของ ระบบไฟล์แบบ FAT

FAT ที่นิยมใช้กันอยู่ใน ของระบบปฏิบัติการดอส และวินโดวส์ คือ FAT16 โดย FAT จะทำหน้าที่จัดการข้อมูลหลายๆ เซ็กเตอร์โดยในแต่ละเซ็กเตอร์จะแบ่งย่อยออกเป็นอีกหลายๆ คลัสเตอร์ (Cluster) ซึ่งในระบบ FAT16 (16 บิต) นั้นสามารถอ้างอิงหรือชี้ตำแหน่งคลัสเตอร์ได้สูงสุด 65,536 คลัสเตอร์ (ข้อมูลทางดิจิทัลจำนวน 1 bit สามารถเป็นได้เพียง 2 สถานะคือ 0 และ 1 ดังนั้น ถ้าเป็น 16 บิต สามารถเป็นได้เท่ากับ 2 ยกกำลัง 16 ซึ่งเท่ากับ 65,536) แต่ในระบบ FAT16 นั้นสามารถมีขนาดของคลัสเตอร์ใหญ่ที่สุด 32 KB (Kilobyte) ดังนั้นในระบบ FAT16 จึงสามารถอ้างข้อมูลในหนึ่งพาร์ทิชัน (partition) ได้สูงที่สุดที่ 2 GB (gigabyte) (32 KB คูณ 65,536 คลัสเตอร์เท่ากับ 2,097,152 KB หรือ 2,048 MB หรือ 2 GB)

ดังนั้นถ้าต้องการใช้ฮาร์ดดิสก์ขนาด 2 GB หรือฮาร์ดดิสก์ที่มีพาร์ทิชันเท่ากับ 2GB หมายความว่าขนาดของคลัสเตอร์ที่เล็กที่สุดเท่ากับ 32 KB ซึ่งหมายความว่าไม่ว่าไฟล์ที่ต้องการเก็บในฮาร์ดิสก์จะมีขนาดเล็กแค่ไหนก็ตาม ฮาร์ดิสก์ก็จะต้องจองพื้นที่ให้ไฟล์นี้ไม่ต่ำกว่า 32 KB ตัวอย่างเช่น ถ้าต้องการเก็บไฟล์ขนาด 1 KB ลงในฮาร์ดดิสก์ที่เป็น FAT 16 ฮาร์ดดิสก์จะต้องจองพื้นที่เพื่อเก็บไฟล์นี้ 32 KB ซึ่งหมายความว่าจะต้องสูญเสียพื้นที่ในฮาร์ดดิสก์ไปโดยไม่สามารถใช้ได้ถึง 31 KB ดังนั้นยิ่งมีไฟล์ขนาดเล็กกว่า 32 KB มากเท่าไร หมายความว่า จะสูญเสียเนื้อที่ ว่าง

บนฮาร์ดดิสก์ไปโดยไม่ได้ใช้ประโยชน์มากขึ้นเท่านั้น วิธีแก้ไขปัญหาการสูญเสียเนื้อที่นี้อาจจะทำได้ 2 กรณีคือ

1. ถ้ายังต้องการใช้ฮาร์ดดิสก์ที่เป็น FAT16 อยู่จะต้องใช้วิธีการแบ่งพาร์ทิชันฮาร์ดดิสก์ให้มีขนาดเล็กลง เพื่อให้ฮาร์ดดิสก์มีขนาดคลัสเตอร์เล็กลง จะได้ทำให้เนื้อที่ว่างที่ไม่สามารถใช้งานเหลือน้อยลง ดังตารางข้างล่างนี้คือถ้าใช้แบ่งพาร์ทิชันฮาร์ดดิสก์ไว้ที่ขนาด 512 MB ต่อพาร์ทิชัน ขนาดของคลัสเตอร์จะลดลงเหลือแค่ 8 KB ซึ่งจะทำให้ความสูญเสียเนื้อที่บนฮาร์ดดิสก์โดยเปล่าประโยชน์ลดน้อยลงถึง 3 เท่าเมื่อเทียบกับการแบ่งพาร์ทิชันไว้ที่ขนาด 2 GB แต่วิธีการแบ่งฮาร์ดดิสก์ออกเป็นหลายๆ พาร์ทิชันนี้อาจทำให้เกิดความยุ่งยากเช่น มีไดร์ฟฮาร์ดดิสก์หลายๆ ไดร์ฟ อาจทำให้สับสนเวลาใช้งาน

2. ให้ใช้ฮาร์ดดิสก์ที่เป็น FAT32 ซึ่งเป็นระบบ FAT แบบใหม่ 32 บิต ซึ่งมีในระบบปฏิบัติการวินโดวส์ 95 OSR2 (Windows 95 OEM Service Release 2) หรือในวินโดวส์ 98 หรือใช้โปรแกรมที่ช่วยแปลง FAT16 ให้เป็น FAT32 อย่างเช่น Partition-It, Partition Magic เป็นต้น สำหรับ FAT32 นี้ จำนวนคลัสเตอร์ที่จะอ้างอิงถึงได้เท่ากับ 2 ยกกำลัง 28 หรือเท่ากับ 268,436,456 คลัสเตอร์ ดังนั้นเมื่อใช้ขนาดของคลัสเตอร์ 4 KB ขนาดของพาร์ทิชันสูงสุดที่จะมีได้จะเท่ากับ 8 GB และถ้าขนาดของคลัสเตอร์สูงสุดที่ 32 KB จะทำให้ฮาร์ดดิสก์สามารถมีพาร์ทิชันได้สูงที่สุดที่ 2 TB (1 Tetra Byte เท่ากับ 1,024 GB)

ขนาดของพาร์ทิชัน	ขนาดของคลัสเตอร์ FAT32	ขนาดของคลัสเตอร์ FAT16
น้อยกว่า 260 Megabyte	512 byte	4 kilobyte
260-511 Megabyte	4 kilobyte	8 kilobyte
512-1023 Megabyte	4 kilobyte	8 kilobyte
1024-2048 Megabyte	4 kilobyte	16 kilobyte
2-8 Gigabyte	4 kilobyte	32 kilobyte
8-16 Gigabyte	8 kilobyte	32 kilobyte
16-32 Gigabyte	16 kilobyte	32 kilobyte
มากกว่า 32 Gigabyte	32 kilobyte	32 kilobyte

ตารางที่ 3.4 เปรียบเทียบขนาดระหว่างคลัสเตอร์ FAT32 และ FAT16

ในเครื่องคอมพิวเตอร์ที่ใช้ระบบปฏิบัติการวินโดวส์ 95 นั้น ถ้าสำรวจดูในโฟล์เดอร์ (Folder) ต่างๆ จะพบว่ามีไฟล์ขนาดเล็กๆ เป็นจำนวนหลายร้อยไฟล์ ตัวอย่างเช่น ไฟล์ที่มี

นามสกุล .dll ซึ่งไฟล์เหล่านี้จะเป็นต้นต่อของการทำให้เกิดการสูญเสียเนื้อที่ในฮาร์ดดิสก์อย่างไม่มีประโยชน์ โดยไม่ว่าฮาร์ดดิสก์จะมีคลัสเตอร์เป็นแบบ FAT32 หรือ FAT 16 ก็ตาม จากตารางข้างล่างต่อไปนี้จะแสดงถึงไฟล์ต่างๆ ในโฟล์เดอร์ของวินโดวส์ 95 โดยทั่วไป เมื่อใช้ฮาร์ดิสก์มีคลัสเตอร์เป็น 4 Kilobyte (FAT32) เนื้อที่ในฮาร์ดิสก์ 83.5 เปอร์เซ็นต์เป็นเนื้อที่ที่เสียไปโดยไม่มีประโยชน์ และเมื่อใช้ฮาร์ดิสก์มีคลัสเตอร์เป็น 32 Kilobyte (FAT16) เนื้อที่ในฮาร์ดิสก์ 97.9 เปอร์เซ็นต์เป็นเนื้อที่ที่เสียไปโดยไม่มีประโยชน์

3.8 ชิพถอดรหัสข้อมูล VS1011b

3.8.1 คุณสมบัติของ VS1011b

1. สามารถถอดรหัสข้อมูล Mpeg Audio ได้ทั้ง Layer 1, 2 และ 3

2. สนับสนุนการถอดรหัสข้อมูล Mpeg 1 และ 2 ทั้งทุก Layer ใน Mpeg 3 สามารถถอดรหัสได้ถึง Layer 2.5 โดยสนับสนุนข้อมูลทั้งแบบโมโนและสเตริโอ

3. สามารถส่งข้อมูลด้วยความเร็วได้หลายระดับ

4. ทำงานที่สัญญาณนาฬิกา 12.288 – 16 เมกะเฮิรตซ์ หรือ 24.576 – 26 เมกะเฮิรตซ์ (สำหรับความเร็วในการส่งต่ำ)

5. ประหยัดพลังงาน

6. ในชิพประกอบด้วย Digital analog converter (DAC) คุณภาพสูงโดยปราศจากเพสเอเรอระหว่างแชนแนล

7. สำหรับสัญญาณอนาลอกทำงานด้วยระดับแรงดัน 2.6 – 3.6 V

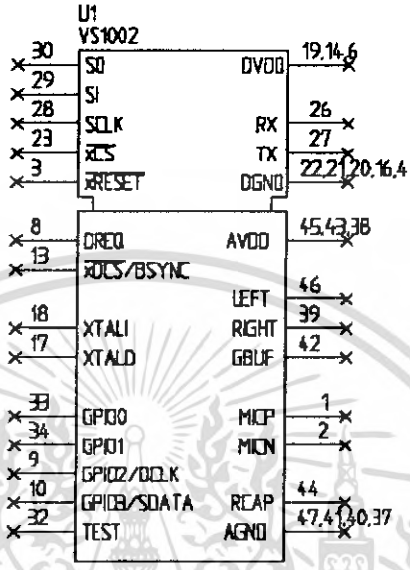
8. สำหรับสัญญาณดิจิตอลทำงานด้วยระดับแรงดัน 2.1 – 3.6 V

9. มี Ram บนชิพถึง 4 กิโลบิต สำหรับผู้ใช้

10. มีฟังก์ชั่นใหม่เพิ่มเติม เช่น ข้อมูลเข้าเป็น PCM ข้อมูลเข้าเป็นสตรีม (Streaming)

11. ประมวลผลใน 16 บิตเวิร์สของข้อมูล

3.8.2 รายละเอียดของขาใน VS1011b



รูปที่ 3.4 รายละเอียดขาต่าง ๆ ของชิพ VS1011b

ชื่อ	ขาที่	ชนิดของขา	หน้าที่
DREQ	1	DO	ขาแสดงร้องขอข้อมูล
DCIK	2	DIO	ขาสัญญาณนาฬิกาสำหรับข้อมูลอินพุตแบบอนุกรม
SDATA	3	DI	ขาข้อมูลอินพุตแบบอนุกรม
BSYNC	4	DI	ขาสัญญาณแสดงข้อมูลซิงโครไนด์
DVDD1	5	PWR	ขาไฟเลี้ยงดิจิตอล
DGND1	6	PWR	ขากราวด์ดิจิตอล
XTALO	7	CLK	ขาคริสตอลเอาท์พุต
XTALI	8	CLK	ขาคริสตอลอินพุต
DVDD2	9	PWR	ขาไฟเลี้ยงดิจิตอล
DGND2	10	PWR	ขากราวด์ดิจิตอล
XCS	11	DI	ขาเลือกอินพุตข้อมูลว่าเป็นข้อมูลหรือคอนโทรล (active low)
SCLK	12	DI	ขาสัญญาณนาฬิกาสำหรับข้อมูลอินพุตแบบอนุกรม
SI	13	DI	ขาอินพุตอนุกรม

ชื่อ	ขาที่	ชนิดของขา	หน้าที่ของขา
SO	14	DO3	ขาเอาท์พุตอนุกรม
TEST0	15	DI	ขาสำรองไว้สำหรับทดสอบ ต่อเข้ากับ DVDD
TEST1	16	DIO	ขาสำรองไว้สำหรับทดสอบ ไม่ต้องต่อ
TEST2	17	DIO	ขาสำรองไว้สำหรับทดสอบ ไม่ต้องต่อ
AGND1	18	PWR	ขากราวด์อนาลอก
AVDD1	19	PWR	ขาไฟเลี้ยงอนาลอก
RIGHT	20	AO	ขาเอาท์พุตแชนแนลขวา
AGND2	21	PWR	ขากราวด์อนาลอก
RCAP	22	AIO	ขาเปรียบเทียบความจุไฟฟ้า
AVDD2	23	PWR	ขาไฟเลี้ยงอนาลอก
LEFT	24	AO	ขาเอาท์พุตแชนแนลซ้าย
AGND3	25	PWR	ขากราวด์อนาลอก
XRESET	26	DI	ขารีเซตแบบอะซิงโครไนส์ ทำงานที่ระดับสัญญาณ “0”
DGND3	27	PWR	ขากราวด์ดิจิตอล
DVDD3	28	PWR	ขาไฟเลี้ยงดิจิตอล

ตารางที่ 3.5 หน้าที่ของขาในชิพ VS1011b

ความหมายของขาแต่ละชนิด

- DI คือ ขาดิจิตอลอินพุต
- DO คือ ขาดิจิตอลเอาท์พุต
- DIO คือ ขาดิจิตอลอินพุต/เอาท์พุต
- AO คือ ขาอนาล็อกเอาท์พุต
- CLK คือ ขาสัญญาณนาฬิกา/ขาแสดงการต่อกับคริสตอล
- PWR คือ ขาไฟเลี้ยงหรือกราวด์

3.8.3 ลักษณะการทำงาน

VS1011b ใช้คุณสมบัติพื้นฐานการประมวลผลสัญญาณดิจิตอล VS_DPS ภายในบรรจุรหัสและหน่วยความจำที่จำเป็นสำหรับการถอดรหัส Mpeg ทั้งหมด พร้อมด้วยพอร์ตอนุกรม กับตัวขยายและควบคุมเอาท์พุตที่เป็นสัญญาณอนาลอก สเตริโอ DAC หลายระบบ

ข้อมูลและบิตเรตในส่วนเพิ่มเติมสามารถส่งข้อมูลด้วยความเร็วหลายระดับ(VBR) เป็นตัวสนับสนุนระดับ VBR ที่ใช้สำหรับเพลงจากใกล้เคียง VS1011b สามารถเล่นได้ทั้งไฟล์ Mpeg 1และ 2 Layer 1,2และ 3 ส่วนประกอบไฟล์ทั้ง อัตราสุ่ม CD นั้นประมาณ 100 kbs/s สำหรับตัวอย่างเพลงระบบสเตริโอที่ความถี่ 44100Hz ด้วยเหตุที่การเข้ารหัสแบบเก่าต้องใช้ 128kbs/sสำหรับการทำงานเต็มที่ VBR การเข้ารหัสคุณภาพสูงได้ถูกนำมาใช้ประโยชน์อย่างกว้างขวาง

บทที่ 4

การทดลองและผลการทดลอง

ในการออกแบบวงจรนั้นได้นำเอาอุปกรณ์ SD CARD มาเชื่อมต่อกับตัวไมโครคอนโทรลเลอร์ชนิด AVR ระหว่างอุปกรณ์เราได้แสดงวงจรไว้ที่ภาคผนวก

4.1 การอ่านข้อมูลจาก SD CARD โดยใช้โปรแกรม WinHex โดยตัวโปรแกรมนี้จะแสดงข้อมูลที่อยู่ในอุปกรณ์ SD CARD ทั้งหมดโดยจะแสดงข้อมูลคือ Address ของข้อมูลว่าอยู่ที่ Address ที่เท่าไหร่ รูปที่ 4.1 เป็นการแสดงข้อมูลทั้งหมดที่ โปรแกรม Win Hex อ่านออกมา

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0003E000	41	42	00	61	00	64	00	20	00	44	00	0F	00	7A	61	00
0003E010	79	00	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF
0003E020	42	41	44	44	41	59	7E	31	4D	50	33	20	00	94	1B	A3
0003E030	7A	36	7A	36	00	00	EC	A2	7A	36	02	00	3D	14	39	00
0003E040	41	50	00	6F	00	74	00	61	00	74	00	0F	00	F1	6F	00
0003E050	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF	FF	FF
0003E060	50	4F	54	41	54	4F	20	20	4D	50	33	20	00	28	1D	A3
0003E070	7A	36	7A	36	00	00	4B	9B	70	36	25	07	0F	8D	3E	00
0003E080	41	4B	00	6F	00	6E	00	4C	00	61	00	0F	00	DC	46	00
0003E090	61	00	6E	00	2E	00	6D	00	70	00	00	00	33	00	00	00
0003E0A0	4B	4F	4E	4C	41	46	41	4E	4D	50	33	20	00	9D	20	A3
0003E0B0	7A	36	7A	36	00	00	21	9E	7A	36	F7	0E	E0	25	47	00
0003E0C0	43	4F	4E	43	45	52	54	4F	4D	50	33	20	18	8C	22	A3
0003E0D0	7A	36	7A	36	00	00	6E	9E	7A	36	DC	17	C0	7B	2C	00
0003E0E0	42	33	00	00	00	FF	FF	FF	FF	FF	FF	0F	00	88	FF	FF
0003E0F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF
0003E100	01	50	00	6C	00	61	00	79	00	67	00	0F	00	88	72	00
0003E110	6F	00	75	00	6E	00	64	00	2E	00	00	00	6D	00	70	00
0003E120	50	4C	41	59	47	52	7E	31	4D	50	33	20	00	AA	23	A3
0003E130	7A	36	7A	36	00	00	D8	A2	7A	36	6C	1D	0A	1E	3E	00
0003E140	53	54	41	52	20	20	20	20	4D	50	33	20	18	57	25	A3
0003E150	7A	36	7A	36	00	00	DC	A2	7A	36	30	25	5E	6E	35	00
0003E160	57	41	49	54	20	20	20	20	4D	50	33	20	18	9D	26	A3
0003E170	7A	36	7A	36	00	00	E2	A2	7A	36	DE	2B	0A	3E	3C	00
0003E180	41	4B	00	6F	00	72	00	64	00	20	00	0F	00	41	6D	00
0003E190	6F	00	72	00	6E	00	2E	00	6D	00	00	00	70	00	33	00
0003E1A0	4B	4F	52	44	4D	4F	7E	31	4D	50	33	20	00	C4	27	A3
0003E1B0	7A	36	7A	36	00	00	E5	A2	7A	36	66	33	B6	0D	3E	00

รูปที่ 4.1 แสดงข้อมูลทั้งหมดที่อยู่ใน SD CARD โดยใช้โปรแกรม Win Hex

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0003E000	41	42	00	61	00	64	00	20	00	44	00	0F	00	7A	61	00
0003E010	79	00	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF
0003E020	42	41	44	44	41	59	7E	31	4D	50	33	20	00	94	1B	A3
0003E030	7A	36	7A	36	00	00	EC	A2	7A	36	02	00	3D	14	39	00
0003E040	41	50	00	6F	00	74	00	61	00	74	00	0F	00	F1	6F	00
0003E050	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF	FF	FF
0003E060	50	4F	54	41	54	4F	20	20	4D	50	33	20	00	28	1D	A3
0003E070	7A	36	7A	36	00	00	4B	9B	70	36	25	07	0F	8D	3E	00
0003E080	41	4B	00	6F	00	6E	00	4C	00	61	00	0F	00	DC	46	00
0003E090	61	00	6E	00	2E	00	6D	00	70	00	00	00	33	00	00	00
0003E0A0	4B	4F	4E	4C	41	46	41	4E	4D	50	33	20	00	9D	20	A3
0003E0B0	7A	36	7A	36	00	00	21	9E	7A	36	F7	0E	E0	25	47	00
0003E0C0	43	4F	4E	43	45	52	54	4F	4D	50	33	20	18	8C	22	A3
0003E0D0	7A	36	7A	36	00	00	6E	9E	7A	36	DC	17	C0	7B	2C	00
0003E0E0	42	33	00	00	00	FF	FF	FF	FF	FF	FF	0F	00	88	FF	FF
0003E0F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF
0003E100	01	50	00	6C	00	61	00	79	00	67	00	0F	00	88	72	00
0003E110	6F	00	75	00	6E	00	64	00	2E	00	00	00	6D	00	70	00
0003E120	50	4C	41	59	47	52	7E	31	4D	50	33	20	00	AA	23	A3
0003E130	7A	36	7A	36	00	00	D8	A2	7A	36	6C	10	0A	1E	3E	00
0003E140	53	54	41	52	20	20	20	20	4D	50	33	20	18	57	25	A3
0003E150	7A	36	7A	36	00	00	DC	A2	7A	36	30	25	5E	6E	35	00
0003E160	57	41	49	54	20	20	20	20	4D	50	33	20	18	9D	26	A3
0003E170	7A	36	7A	36	00	00	E2	A2	7A	36	DE	2B	0A	3E	3C	00
0003E180	41	4B	00	6F	00	72	00	64	00	20	00	0F	00	41	6D	00
0003E190	6F	00	72	00	6E	00	2E	00	6D	00	00	00	70	00	33	00
0003E1A0	4B	4F	52	44	4D	4F	7E	31	4D	50	33	20	00	C4	27	A3
0003E1B0	7A	36	7A	36	00	00	E5	A2	7A	36	66	33	B6	0D	3E	00

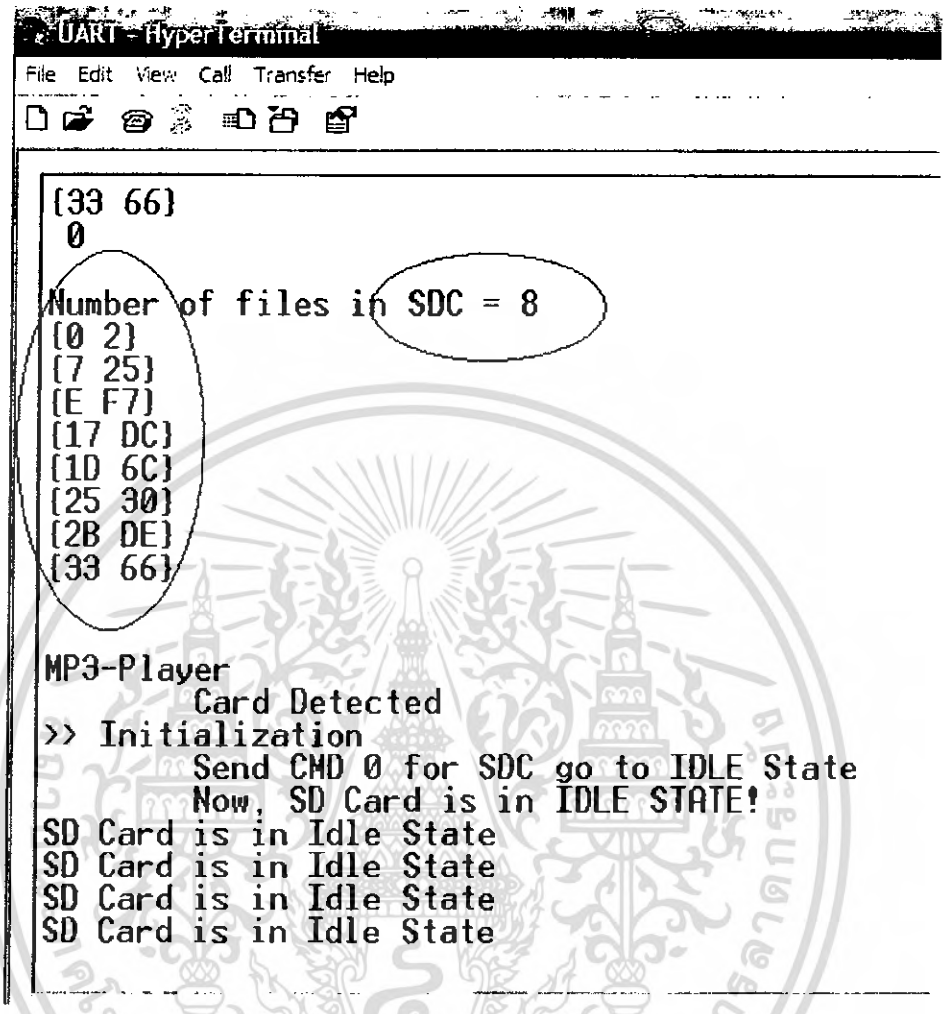
รูปที่ 4.2 แสดงรายละเอียดของข้อมูลที่เราต้องการอ่าน

ในรูปที่ 4.2 นั้นเราจะดูที่แถว B เราจะเห็นว่ามีค่า 0F อยู่ซึ่งค่านี้คือมันจะไม่อ่านแล้วข้ามไปจนกว่าจะจะเจอ 20 ค่านี้จะเป็นตัวบ่งบอกว่าเราจะเริ่มอ่านไฟล์จากตรงนี้

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0003E000	41	42	00	61	00	64	00	20	00	44	00	0F	00	7A	61	00
0003E010	79	00	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF
0003E020	42	41	44	44	41	59	7E	31	4D	50	33	20	00	94	1B	A3
0003E030	7A	36	7A	36	00	00	EC	A2	7A	36	02	00	3D	14	39	00
0003E040	41	50	00	6F	00	74	00	61	00	74	00	0F	00	F1	6F	00
0003E050	2E	00	6D	00	70	00	33	00	00	00	00	00	FF	FF	FF	FF
0003E060	50	4F	54	41	54	4F	20	20	4D	50	33	20	00	28	1D	A3
0003E070	7A	36	7A	36	00	00	4B	9B	70	36	25	07	0F	8D	3E	00
0003E080	41	4B	00	6F	00	6E	00	4C	00	61	00	0F	00	DC	46	00
0003E090	61	00	6E	00	2E	00	6D	00	70	00	00	00	33	00	00	00
0003E0A0	4B	4F	4E	4C	41	46	41	4E	4D	50	33	20	00	9D	20	A3
0003E0B0	7A	36	7A	36	00	00	21	9E	7A	36	F7	0E	E0	25	47	00
0003E0C0	43	4F	4E	43	45	52	54	4F	4D	50	33	20	18	8C	22	A3
0003E0D0	7A	36	7A	36	00	00	6E	9E	7A	36	DC	17	C0	7B	2C	00
0003E0E0	42	33	00	00	00	FF	FF	FF	FF	FF	FF	0F	00	88	FF	FF
0003E0F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF
0003E100	01	50	00	6C	00	61	00	79	00	67	00	0F	00	88	72	00
0003E110	6F	00	75	00	6E	00	64	00	2E	00	00	00	6D	00	70	00
0003E120	50	4C	41	59	47	52	7E	31	4D	50	33	20	00	AA	23	A3
0003E130	7A	36	7A	36	00	00	D8	A2	7A	36	6C	1D	DA	1E	3E	00
0003E140	53	54	41	52	20	20	20	20	4D	50	33	20	18	57	25	A3
0003E150	7A	36	7A	36	00	00	DC	A2	7A	36	30	25	5E	6E	35	00
0003E160	57	41	49	54	20	20	20	20	4D	50	33	20	18	9D	26	A3
0003E170	7A	36	7A	36	00	00	E2	A2	7A	36	DE	2B	0A	3E	3C	00
0003E180	41	4B	00	6F	00	72	00	64	00	20	00	0F	00	41	6D	00
0003E190	6F	00	72	00	6E	00	2E	00	6D	00	00	00	70	00	33	00
0003E1A0	4B	4F	52	44	4D	4F	7E	31	4D	50	33	20	00	C4	27	A3
0003E1B0	7A	36	7A	36	00	00	E5	A2	7A	36	66	33	B6	0D	3E	00

รูปที่ 4.3 เป็นตำแหน่งที่เก็บข้อมูลคลัสเตอร์แรกของไฟล์

ในตำแหน่งที่เราทำสัญลักษณ์เป็นตำแหน่งที่เก็บข้อมูลเริ่มต้นของไฟล์ซึ่งภายในจะเก็บข้อมูลเอาไว้ซึ่งเราต้องเอา 512 คูณเข้าไป



รูปที่ 4.4 เป็นรูปที่แสดงจำนวนไฟล์และตำแหน่งที่ไฟล์อยู่

บทที่ 5
บทสรุป

ในปริญญานิพนธ์ฉบับนี้ ได้นำเสนอรายละเอียดโครงงานสร้างเครื่องเล่นไฟล์เสียง MP3 ใช้ SD Card เป็นอุปกรณ์ที่เป็นตัวเก็บข้อมูลไฟล์เสียง โดยได้ศึกษารูปแบบมาตรฐานการบันทึกข้อมูลซึ่งรายงานไว้ในบทที่ 2 จากนั้นได้ศึกษาการนำไมโครคอนโทลเลอร์ตระกูล AVR มาใช้อ่านไฟล์เสียง MP3 ตามมาตรฐานจาก SD Card เพื่อส่งไปถอดรหัสให้เป็นเสียงที่รับฟังได้ โดยชิปถอดรหัส VS1011b และแสดง รูปวงจร รูปเครื่องต้นแบบที่พัฒนาขึ้นมารวมทั้งโปรแกรมไว้ที่ภาคผนวกของ ปริญญานิพนธ์ฉบับนี้

ตามที่ได้ออกแบบไว้เครื่องเล่นไฟล์เสียง MP3มีฟังก์ชันดังนี้

1. สามารถที่จะกดเลื่อนเพลงได้ คือ Next และ Previous
2 . สามารถที่จะกด Stop และ Play ได้
3. สามารถที่จะ เพิ่มและลดระดับความดังของเสียงได้

จากการทดลองเล่นกับไฟล์เสียง MP3 มาตรฐาน อุปกรณ์ที่ได้พัฒนาขึ้นมาสามารถเล่นกลับได้ และใช้งานตามฟังก์ชันที่ออกแบบไว้ได้ทุกประการ

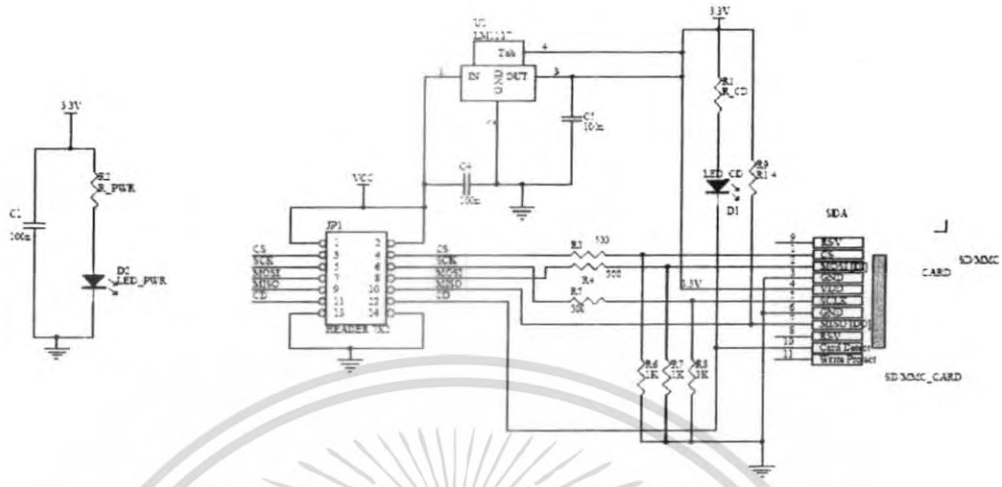
จุดด้อยของเครื่องเล่นไฟล์เสียง MP3นี้อยู่ที่ยังไม่สามารถเล่นไฟล์เสียง MP3ที่อยู่ในโฟลเดอร์ ได้ ซึ่งสามารถแก้ไขจุดด้อยนี้ได้เมื่อได้รับการพัฒนาต่อไปในอนาคต

บรรณานุกรม

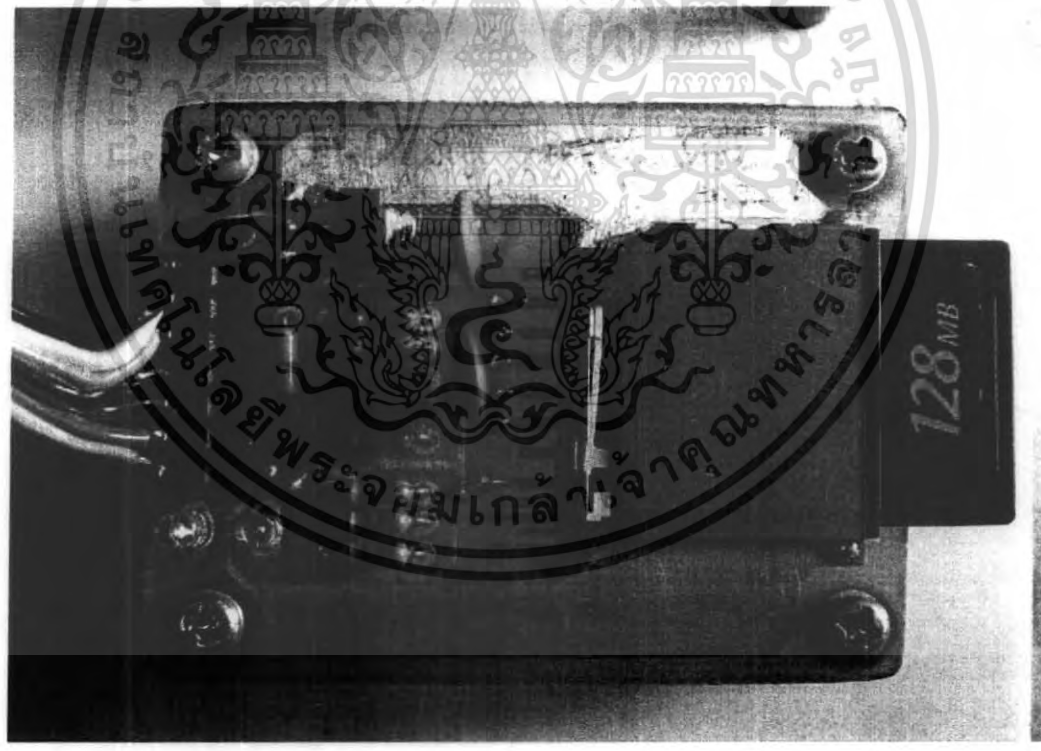
1. กองบรรณาธิการ, รายงานล่าสุด Windows 95 OSR2.1 Thai ในงาน Computer Thai 97, นิตยสาร Windows Magazine ฉบับ 53 เดือนธันวาคม 2540, หน้า 187-192
2. ธีระทัศน์ เกิดช่วย, ถอดเกราะ FAT32, นิตยสาร Windows Magazine ฉบับ 53 เดือนธันวาคม 2540, หน้า 198-202
3. กองบรรณาธิการ, คำถามที่น่าสนใจกับ Windows 95 OSR 2.X Thai Edition, นิตยสาร Quick PC ฉบับ 21 ปักษ์แรกเดือนมีนาคม 2541, หน้า 63-69
4. http://elm-chan.org/docs/mmc/mmc_e.html
5. http://www.ett.co.th
6. http://cs.gmu.edu/~menasce/osbook/distfs/sld014.html
7. http://library.thinkquest.org/C006208/data/device-harddrives3.php
8. http://pioneer.netserv.chula.ac.th/~wchairat/news/fat32-16.htm
9. http://library.thinkquest.org/C006208/data/device-harddrives3.php
10. http://203.146.17.98/suphat/unit3-1.htm

ภาคผนวก

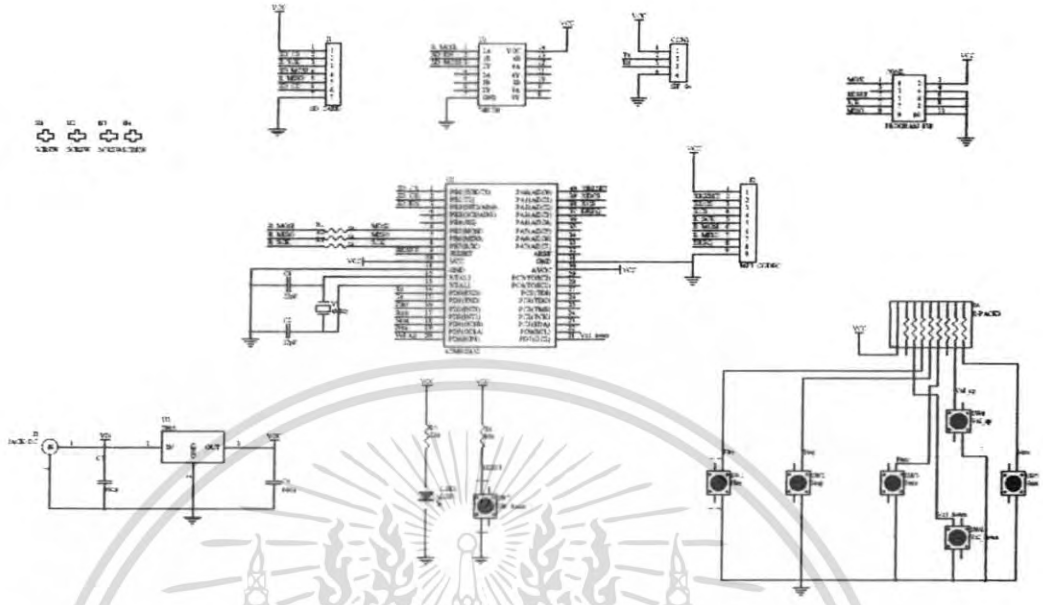
ส่วนประกอบทั้งหมดของเครื่องเล่น mp3



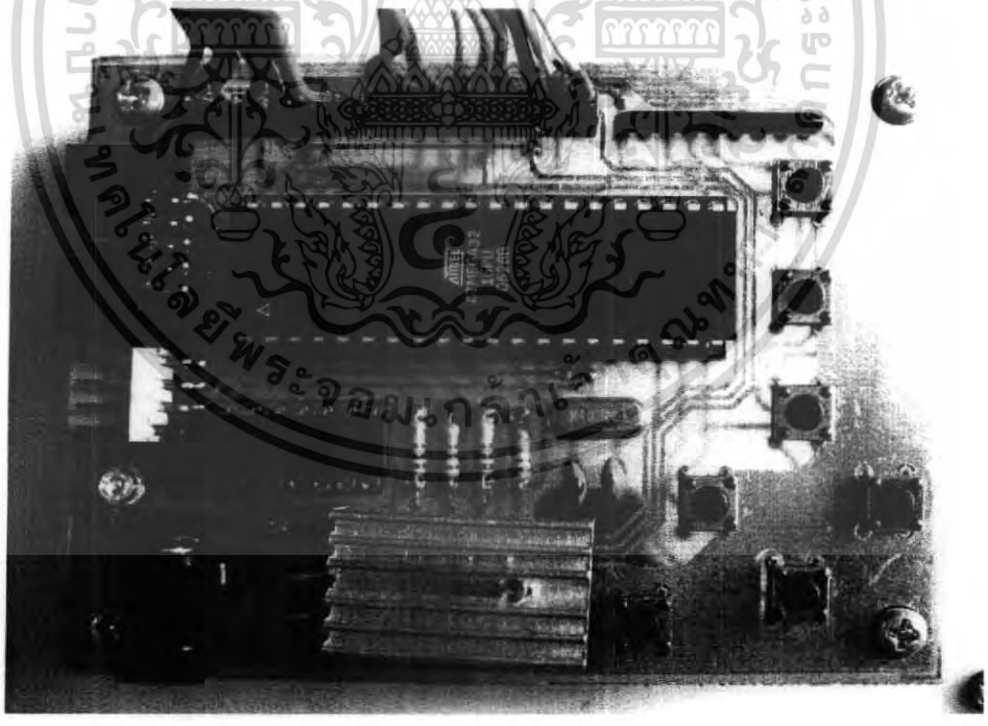
รูปวงจรในส่วนของ Sd card



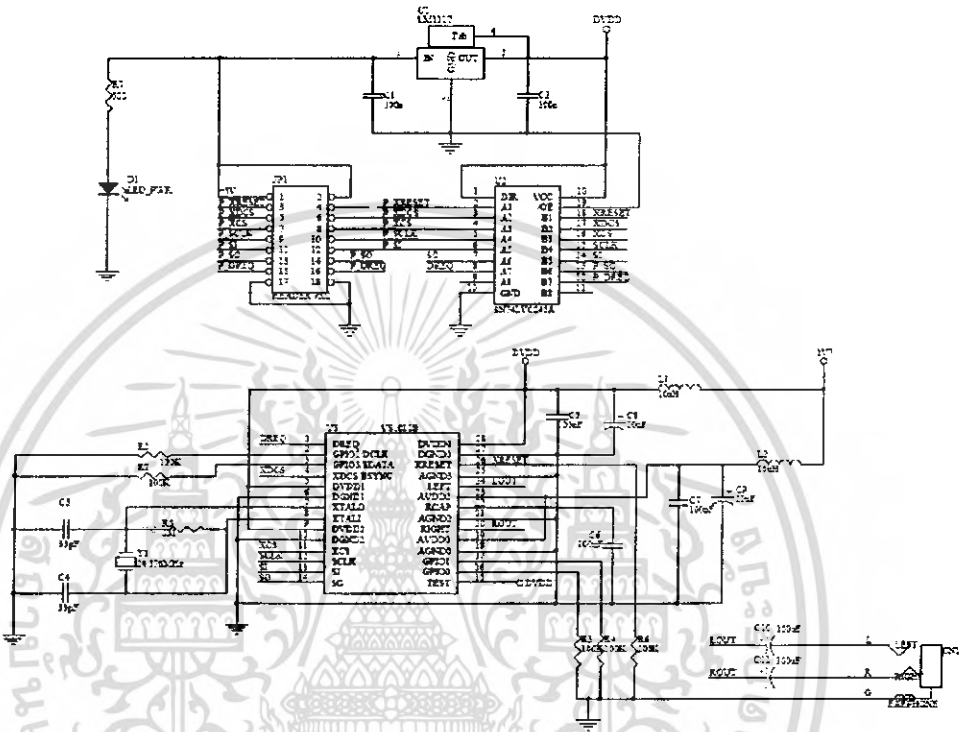
รูปถ่ายของชิ้นงานในส่วนของ Sd card



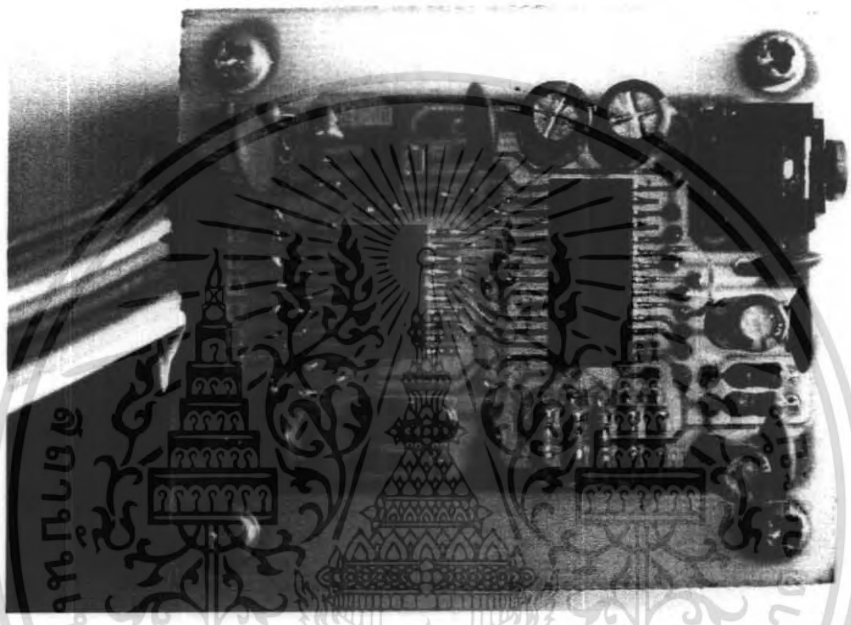
รูปวงจรในส่วนของไมโครคอนโทรลเลอร์



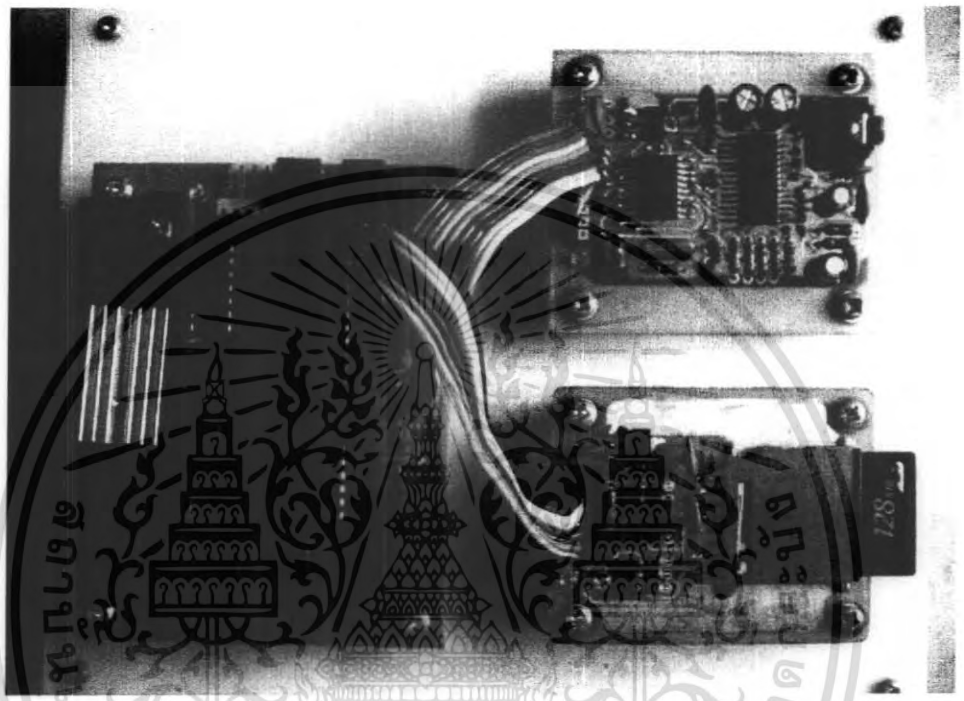
รูปถ่ายของชิ้นงานในส่วนของไมโครคอนโทรลเลอร์



รูปวงจรในส่วนของการ Decode



รูปถ่ายของชิ้นงานในส่วนของภาค Decode



รูปถ่ายแสดงชิ้นงานที่สมบูรณ์

**** Main Program ****

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.8d Professional
Automatic Program Generator
© Copyright 1998-2006 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Chip type           : ATmega32
Program type        : Application
Clock frequency     : 4.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 512
*****************************************************/

#include <atmega32.h>
#include <global.h>
#include <string.h>
#include <delay.h>
#include <stdio.h>
#include <uart9600.c>
#include <spi_io.c>
#include <SDC_media.c>
#include <fat_process.c>
#include <vs1011b.c>
#include <player.c>
#include <keypad.c>
```

```
void main(void)
{
FILEOBJ      fo;
word         index;
byte         fat_buf[512];
byte         resp;
dword        sector;

signed char    result;

UART9600_init();

printf("\n\n\rMP3-Player\n\r");
keypad_init();
spi_init();
SDC_hw_init();
VS1011B_init();
     while(!Detect_SDC()){};

SDC_media_init();
SD_get_volume_info();

printf("\n\r********** Main Program **********\n\r");

VS1011B_volume(volume[3],volume[3]);
FAT_init(fat_buf);
FAT_cache_file(fat_buf);

while(KEY_PLAY == 1){};
UART_printbyte(num_of_files);
```

```
while(1)
{
fplay(first_cluster[track],fat_buf);

track++;
if(track > num_of_files) track = 0;
}

printf("\n\r********************* END ******************\n\r");

}

                    **** #includes <uart9600.c> ****

#define UART_newline()  printf("\n\r");
void UART9600_init(void)
{
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;
}
void UART_SendByte(unsigned char data)
{
      while( !(UCSRA & (1 << UDRE)))
      UDR = data;
}
```

```
void UART_printbyte(byte data)
{
    printf(" %X",data);
}
void UART_printword(word data)
{
    byte tmp;
    tmp = (data & 0xff00)>>8;
    printf("{%X",tmp);
    tmp = (data & 0x00ff);
    printf(" %X}",tmp);
}
void UART_printdword(dword data)
{
    byte tmp;
    tmp = (data & 0xff000000) >> 24;
    printf("{%X",tmp);
    tmp = (data & 0x00ff0000) >> 16;
    printf(" %X",tmp);
    tmp = (data & 0x0000ff00) >> 8;
    printf(" %X",tmp);
    tmp = (data & 0x000000ff);
    printf(" %X}",tmp);
}
```

****** #include <spi_io.c>******

```
#include<ATMEGA32.h>

// For Master Mode
/*
PB7 --> SCK    As  OUTPUT
PB6 --> MISO   As  INPUT
PB5 --> MOSI   As  OUTPUT
PB4 --> /SS    As  OUTPUT
*/
#define SCK     PB7
#define MISO    PB6
#define MOSI    PB5
#define SS      PB4

#define MSTR 4
#define SPE  6
#define SPIF 7

void spi_init(void)
{
        DDRB |= (1<<PB5) | (1<<PB4) | (1<<PB7);
        PORTB &= ~(1<<PB7);
        SPCR = ((1<<MSTR)|(1<<SPE) );
}
unsigned char spi_io(unsigned char data)
{
        SPDR = data;
        while((SPSR&(1<<SPIF)) == 0x00){};
```

```
        return SPDR;
}
```

******* #include <SDC_media.c>*******

```
//---------------------------------- SDC_media----------------------------------
#define SDC_select     SD_PORT.PIN_SD_CS = 0;
#define SDC_deselect   SD_PORT.PIN_SD_CS = 1;

#define Send8Clk()          spi_io(0xff);
#define ReadCRC()           spi_io(0xff);spi_io(0xff);
#define SDC_FLOATING_BUS    0xFF
#define SDC_BAD_RESPONSE    SDC_FLOATING_BUS
#define SDC_SECTOR_SIZE     512

#define DATA_START_TOKEN    0xFE

static unsigned int current_blocklen = 0;
//************************** Command & Response Structure*********************

//RESPONSE_1
typedef union
{
    byte _byte;
    //struct
    //{
    unsigned IN_IDLE_STATE:1;
    unsigned ERASE_RESET:1;
    unsigned ILLEGAL_CMD:1;
    unsigned CRC_ERR:1;
    unsigned ERASE_SEQ_ERR:1;
```

```
        unsigned ADDRESS_ERR:1;
        unsigned PARAM_ERR:1;
        unsigned B7:1;
        //};
}RESPONSE_1;

//RESPONSE_2
typedef union
{
        word _word;
        //struct
        //{
        byte _byte0;
        byte _byte1;
        //};
        //struct
        //{
        unsigned IN_IDLE_STATE:1;
        unsigned ERASE_RESET:1;
        unsigned ILLEGAL_CMD:1;
        unsigned CRC_ERR:1;
        unsigned ERASE_SEQ_ERR:1;
        unsigned ADDRESS_ERR:1;
        unsigned PARAM_ERR:1;
        unsigned B7:1;
        unsigned CARD_IS_LOCKED:1;
        unsigned WP_ERASE_SKIP_LK_FAIL:1;
        unsigned ERROR:1;
        unsigned CC_ERROR:1;
        unsigned CARD_ECC_FAIL:1;
```

```
    unsigned WP_VIOLATION:1;
    unsigned ERASE_PARAM:1;
    unsigned OUTRANGE_CSD_OVERWRITE:1;
    //};
}RESPONSE_2;

//SDC_RESPONSE
typedef union
{
    RESPONSE_1 r1;
    RESPONSE_2 r2;
}SDC_RESPONSE;

//************************** Error Codes*******************************
typedef enum
{
    sdcValid = 0,        //No error
    sdcCardInitCommFailure, //Communication hasn't been established with the card.
    sdcCardNotInitFailure,  //Card did not initialize.
    sdcCardInitTimeout,     //Card initialization timed out.
    sdcCardTypeInvalid,     //Card type was not able to be defined.
    sdcCardBadCmd,          //Card did not recognize the command.
    sdcCardTimeout,         //Card timed out during a read, write or erase sequence.
    sdcCardCRCError,        //A CRC error occurred during a read.
    sdcCardDataReject,      //CRC did not match.
    sdcCardEraseTimeOut     //Erase timed out.
}SDC_Error;

void Check_error(SDC_Error error)
{
```

```
    switch(error)
    {
        case sdcValid: printf("sdcValid\n\r"); break;
        case sdcCardInitCommFailure: printf("sdcCardInitCommFailure\n\r"); break;
        case sdcCardNotInitFailure: printf("sdcCardNotInitFailure\n\r"); break;
        case sdcCardInitTimeout: printf("sdcCardInitTimeout\n\r"); break;
        case sdcCardTypeInvalid: printf("sdcCardTypeInvalid\n\r"); break;
        case sdcCardBadCmd: printf("sdcCardBadCmd\n\r"); break;
        case sdcCardTimeout: printf("sdcCardTimeout\n\r"); break;
        case sdcCardCRCError: printf("sdcCardCRCError\n\r"); break;
        case sdcCardDataReject: printf("sdcCardDataReject\n\r"); break;
        case sdcCardEraseTimeOut: printf("sdcCardEraseTimeOut\n\r"); break;
    }
}
/*************************SD_init*************************************/
//init pin which used for SD_Card Interface
void SDC_hw_init(void)
{
SD_PORT |= (1<<PIN_SD_CS)|(1 << PIN_SD_CD);
// SD_CS       PB0 --> Output
// SD_CD       PB1 --> INPUT with R pull up
SD_DDR |= (1<<PIN_SD_CS);
SD_DDR |= (1<<PIN_SD_ALLOW_SCI);
ALLOW_SCI;
}

unsigned char Detect_SDC(void)
{
    if(SD_PIN.PIN_SD_CD) return 0; //Card not presented
    else {printf("\tCard Detected\n\r");return 1;} //Card presented
```

```
}

/*****************************spi_io_sd*************************************/
void spi_io_sd(unsigned char* data, unsigned int length)
{
// transmit 'length' bytes via spi
while(length)
      {
      spi_io(*data);
      data++;
      length--;
      };
}
void SDC_cleanup(void)
{
SDC_deselect;
//pulse the SCK 8 times
Send8Clk();
}
//******************** A Function for Sending Command******************

/*******************************SD_send_cmd*******************************/
void SD_send_cmd(unsigned cmd, unsigned long data)
{
unsigned char buffer[6];//array 6 block for stores command sequence
buffer[0] = 0x40 + cmd;
buffer[1] = (data>>24)&0xff;
buffer[2] = (data>>16)&0xff;
buffer[3] = (data>>8)&0xff;
buffer[4] = (data&0xff);
```

```
buffer[5] = 0x95;//CRC Value
spi_io_sd(buffer,6);//send_data
}

/*****************************SD_get_R1*********************************/
unsigned char SD_get_R1(void)
{
unsigned char retval;
unsigned char max_errors = 255;
    //wait for first valid response byte
    do
    {
    retval = spi_io(0xff);
    max_errors--;
    }while( (retval & 0x80) && (max_errors>0) );
return retval;
}

/*****************************SD_get_R1b*********************************/
//get a byte long R1b and then waits for the card to be available again
unsigned char SD_get_R1b(unsigned char max_busy)
{
unsigned char retval;
unsigned char max_errors = 64;
    //wait for first valid response byte
    do
    {
    retval = spi_io(0xff);
    max_errors--;
    } while( (retval == 0xff) && max_errors );
```

```
    //loop while card sends the busy byte
    do
    {
    retval = spi_io(0xff);
    max_busy--;
    }while( (retval == SD_R1B_BUSY_BYTE) && max_busy );
return retval;
}

/*****************************SD_get_R2*******************************/
unsigned int SD_get_R2(void)
{
unsigned int retval;
unsigned char max_errors = 64;
    //wait for first valid response byte
    do
    {
    retval = spi_io(0xff);
    max_errors--;
    }while( (retval & 0x80) && (max_errors>0) );
//move data to upper byte
retval = (retval <<8);
//get second byte
max_errors = spi_io(0xff);
retval += max_errors;
return retval;
}
/************************SD_send_start_data_token*****************/
//Send the start data block token to the SD
//Start Block is  7[11111110]0 = 0xfe
```

```
void SD_send_start_data_token(void)
{
spi_io(SD_START_TOKEN_SINGLE);
}
/************************SD_wait_for_start_token*****************/
//waits for the card to send the start data block token
void SD_wait_for_start_token(unsigned char max_errors)
{
unsigned char retval;
      do
      {
      //get a byte from the SPI Bus
      retval = spi_io(0xff);
      //keep track of the trys
      max_errors--;
      //UART_printbyte(retval);
      }while( (retval != SD_START_TOKEN_SINGLE) );
}
/*****************************SD_GET_DATA************************/
//gets n bytes + crc from spi bus
void SD_get_data(unsigned char *ptr_data, unsigned int length)
{
SD_wait_for_start_token(128);
      while(length)
      {
      *ptr_data = spi_io(0xff);
      length--;
      ptr_data++;
      };
//get the 2 CRC bytes
```

```
ReadCRC();
}
/************************ SD_set_blocklength************************/
//set the blocklength for transmission
void SD_set_blocklen(unsigned int blocklen)
{
//make sure this block length is not already set
    if(current_blocklen != blocklen)
    {
    current_blocklen = blocklen;
    SDC_select;
    //tell the SD card that we want to know its status
    SD_send_cmd(SD_CMD_16_BLOCKLEN, blocklen);
    //get the response
    SD_get_R1();
    SDC_cleanup();
    };
}
/************************ Reading CSD Registor ************************/
void SD_get_CSD(unsigned char *ptr_data)
{
//select card
SDC_select;
//tell the SD card that we want to know its status
SD_send_cmd(SD_CMD_9_SEND_CSD,0x0);
//get the response
SD_get_R1();
//get the register data
SD_get_data(ptr_data, 16);
//return CS to 1 and send 8 clk
```

```
SDC_cleanup();
}
/*****************************SD_get_CID*****************************/
//read the CID register from the card
void SD_get_CID(unsigned char *ptr_data)
{
//select card
SDC_select;
//tell the SD card that we want to know its status
SD_send_cmd(SD_CMD_10_SEND_CID,0x0);
//get the response
SD_get_R1();
//get the register data
SD_get_data(ptr_data, 16);
//cleanup
SDC_cleanup();
}
/*********************************SD_get_volume_info*******************/
void SD_get_volume_info(void)
{
/*
Memory capacity = BLOCKNR*BLOCK_LEN
BLOCDNR = (C_SIZE + 1)*MULT
MULT = 2^(C_SIZE_MULT+2)
BLOCK_LEN = 2^(READ_BL_LEN)
*/
unsigned char data[16];
unsigned char name[6];
unsigned char i;
unsigned char mem_cap;
```

```
unsigned char   C_size_mult;
unsigned int    C_size;
unsigned char   MULT = 1;
printf("Volume Info:\n\r\t\t");
//read the CSD register
SD_get_CSD(data);
//printf("Get CSD Already!");

//Get C_Size(12 bits) from the CSD register(width 128 bit) that located in bit [73:62] of data
C_size = data[6] & 03; // data[6]>>>>[80:72]
C_size <<= 8;
C_size += data[7]; //data[7]>>>>[71:64]
C_size <<= 2;
C_size += (data[8] & 0xC0) >> 6; //data[8]>>>>[63:56]
//printf("C_size = %i ",C_size);

//Get C_Size_Mult(3 bits) from the CSD register that located in bit [49:47]
C_size_mult = data[9] & 03; //data[9]>>>>[55:48]
C_size_mult <<= 1;
C_size_mult += (data[10] & 80) >> 7; //data[10]>>>>[47:40]
//printf("C_size_mult = %u ",C_size_mult);

//MULT = 2^(9-C_size_mult)
     for (i = 0; i < (9-C_size_mult); i++)
          MULT *= 2; //2^(9-C_size_mult)

//compute MEMORY CAPACITY
mem_cap = (C_size+1)/MULT;

printf("Memory Capacity = %u  MB\n\r\t\t",mem_cap);
```

```
SD_get_CID(data);
name[0] = data[3];
name[1] = data[4];
name[2] = data[5];
name[3] = data[6];
name[4] = data[7];
name[5] = '\0';

printf("Card name: %s\n\r",name);

}
/**********************SD_get_sec_start*****************************/
//starts the read process of a sector
byte SD_get_sec_start(unsigned long sector)
{
byte response;
SD_set_blocklen(512);
//printf("set blocklenght");
//turn sectors into byte addr
//sector = sector <<9;
     /* printf("\n\rOffset>>");
      UART_printdword(sector);
      UART_newline();*/
//select card
SDC_select;
//tell the SD card that we want to know its status
SD_send_cmd(SD_CMD_17_READ_SINGLE, sector); //printf("send cmd17");
//get the response
response = SD_get_R1();
```

```
//wait untill Card starts sending data
SD_wait_for_start_token(255);
//printf("wait start token");
return response;
}


/*******************************SD_get_sec_stop*****************************/
//stop read process of a sector
void SD_get_sec_stop(void)
{
//get 2 CRC bytes
spi_io(0xff);
spi_io(0xff);
SDC_cleanup();
}


/*******************************SD_get_sector*******************************/
//get a whole sector and put it in the data buffer
void SD_get_sector(unsigned long sector, unsigned char* data)
{
    byte response;
SD_set_blocklen(512);
/*printf("\n\rOffset>>");
    UART_printdword(sector);
    UART_newline();*/
//select card
SDC_select;
```

```
//tell the SD card that we want to know its status
SD_send_cmd(SD_CMD_17_READ_SINGLE, sector);
//get the response
response = SD_get_R1();
//wait untill Card starts sending data
SD_wait_for_start_token(255);
global_tmp = 0;
      while (global_tmp++ < 512)
      {
      *data = spi_io(0xff);
      data++;
      };
//cleanup behind us
SD_get_sec_stop();
}


/*************************Check_resp*********************************/
void Check_resp(RESPONSE_1 resp)
{
delay_ms(100);
     if(resp.IN_IDLE_STATE == 1) printf("SD Card is in Idle State\n\r");
     else if(resp.ERASE_RESET == 1) printf("Erase Reset\n\r");
     else if(resp.ILLEGAL_CMD == 1) printf("Illegal Command\n\r");
     else if(resp.CRC_ERR == 1) printf("CRC_ERR\n\r");
     else if(resp.ERASE_SEQ_ERR == 1) printf("ERASE_SEQ_ERR\n\r");
     else if(resp.ADDRESS_ERR == 1) printf("ADDRESS_ERR\n\r");
     else if(resp.PARAM_ERR == 1) printf("PARAM_ERR\n\r");
     else if(resp.B7 == 1) printf("B7\n\r");
}
```

```
/*****************************SD_reset*********************************/
//Reset card and used SPI Interface
void SDC_media_init(void)
{

unsigned char i;
RESPONSE_1 resp;

//make sure card is ready to comm.
      for(i = 0; i < 10; i++)
      {
      spi_io(0xff);
      };

printf(">> Initialization\n\r");
//select card by pull down CS_PIN
SDC_select;
//put SD to Idle
SD_send_cmd(SD_CMD_0_GO_IDLE,0x0);
printf("\tSend CMD 0 for SDC go to IDLE State\n\r");
//get the response
resp._byte = SD_get_R1();
Send8Clk();

      if (resp.IN_IDLE_STATE == 1)
      {
      //lcd_clear();
      //lcd_putsf("Now, SD Card is\n in IDLE STATE!");
      //delay_ms(2000);
```

```
        printf("\tNow, SD Card is in IDLE STATE!\n\r");
        }

SDC_cleanup();
        //SD_SEND_OP_COND
        while(resp.IN_IDLE_STATE == 1)
        {
        SDC_select;
        //Tell Card that next command is APP_CMD
        SD_send_cmd(SD_CMD_55_APP_CMD,0x0);
        Send8Clk();
        resp._byte = SD_get_R1();
        Send8Clk();
        SDC_cleanup();

        //Send APP_CMD to active card
        SDC_select;
        SD_send_cmd(SD_ACMD_41_SEND_OP_COND,0x0);
        Send8Clk();
        resp._byte = SD_get_R1();
        Send8Clk();
        Check_resp(resp);
        //Cleanup
        SDC_cleanup();
        };

printf("\tSD Card is Activated\n\r");
}
```

```
/*********************************SD_write_sector*****************************/
unsigned char SD_write_sector(unsigned long sector, unsigned char *data)
{
unsigned int tmp = 512;
SD_set_blocklen(512);
//turn sectors into byte addr
sector = sector << 9;
//select card
SDC_select;
//tell the SD card that we want to write a sector
SD_send_cmd(SD_CMD_24_WRITE_SINGLE, sector);
//get the response
SD_get_R1();
//send the start token
spi_io(SD_START_TOKEN_SINGLE);
      while(tmp--)
      {
      spi_io(*data);
      data++;
      };
//send2crcs
spi_io(0xff);
spi_io(0xff);
//get the data response token
/*
can be one of the following:
     SD_DATA_ACCEPT
     SD_DATA_CRC
     SD_DATA_WRITE_ERROR
*/
```

```
tmp = (spi_io(0xff) & 0xf) >>1;
      if(tmp != SD_DATA_ACCEPT)
      {
      printf("ERROR!!!");
      };

      //all ok, wait while busy
      while(spi_io(0xff) == SD_R1B_BUSY_BYTE){};

return tmp;
}
```

*******#include <fat_process.c>*******

```
//Volume Information
#define FAT16   2
#define FAT32   3

      dword   lba_first_sector;       //LBA of the volume's first sector
      dword   lba_fat;                //LBA of the volume's FAT
      dword   lba_root;               //LBA of the volume's root directory
      dword   lba_data;               //LBA of the volume's data area
      word    rsv_sector;
      word    maxroot;              //maximum number of entries in the root directory
      dword   maxcls;               //maximum number of data clusters in the volume
      word    fatsize;              //number of sectors in the FAT
      byte    fatcopy;              //number of copies of the FAT
      dword   sec_per_clus;         //number of sectors per cluster

//File Information
#define FILE_NAME_SIZE  11
```

```
typedef struct
{
    DISK   *dsk;            //a DISK structure for the volume containing the file
    //word   cluster;       //number of the first file's cluster
    word   ccls;            //current cluster
    word   sec;             //current sector in the current cluster
    word   pos;             //current byte location in the current sector
    dword  seek;            //current byte location in the file
    dword  size;            //file size
    word   time;
    word   date;
    char   name[FILE_NAME_SIZE];  //file name
    word   entry;           //Position of the file's entry in its directory
    word   chk;             //FILE structure checksum = ~(entry+name[0])
    word   attributes;      //file's attributes
    word   dirclus;         //first cluster of the file's directory
    word   dirccls;         //current cluster of the file's directory
}FILE;

typedef FILE   *FILEOBJ;    //FILEOBJ is a pointer to a FILE structure

//Using Directories
//Storing an Entry

#define DIR_NAMESIZE   8
#define DIR_EXTENSION  3
#define NULL           0
#define FALSE          0
#define TRUE           !FALSE
```

```
typedef struct
{
    char   DIR_Name[DIR_NAMESIZE];      //name
    char   DIR_Extension[DIR_EXTENSION]; //extension
    byte   DIR_Attr;                    //Attributes
    byte   DIR_NTRes;                   //reserved by NT
    byte   DIR_CrtTimeTenth;
    word   DIR_CrtTime;
    word   DIR_CrtDate;
    word   DIR_LstAccDate;
    word   DIR_FstClusHI;
    word   DIR_WrtTime;
    word   DIR_WrtDate;
    word   DIR_FstClusLO;
    dword  DIR_FileSize;                //filesize
}_DIRENTRY;

typedef _DIRENTRY* DIRENTRY;


#define FAT_SecPerClus  13
#define FAT_RSV_SECTOR  14
#define FAT_COPY        16
#define FAT_MAX_ENT     17
#define FAT_SIZE        22
void FAT_init(unsigned char *data)
{
    word index;
    word tmp;
printf("\n\r********* FAT init **********\n\r");
```

```
SD_get_sector(0x00,data);
//UART_printbyte(*(data+14));
rsv_sector = *(data + 14);
rsv_sector += (word)*(data+15)<<8;
lba_fat = 0x00 + rsv_sector;
lba_fat <<= 9;
printf("LBA_fat = "); UART_printdword(lba_fat);UART_newline();
fatcopy = *(data + 16);
fatsize = *(data + 22);
fatsize += *(data + 23)<<8;
lba_root = rsv_sector + (fatcopy*fatsize);
lba_root <<= 9;
printf("LBA_root = "); UART_printdword(lba_root);UART_newline();

sec_per_clus = *(data + 13);
printf("dsk->SecPerClus = "); UART_printdword(sec_per_clus);UART_newline();


lba_data = (lba_root + (0x20<<9));
printf("LBA_data = "); UART_printdword(lba_data);UART_newline();

}

#define LFN_ENTRY      0x0F
void FAT_cache_file(unsigned char *data)
{
word index,temp,temp2;
unsigned char *backup;
DIRENTRY dir;
backup = data;
```

```
SD_get_sector(lba_root,data);
num_of_files = 0;
    while(1)
    {
        UART_printbyte(*data); UART_newline();
        if(*data == 0) break;
        else
        {
        data += 11;
        }

        if(*data == 0x20)
        {

        data += 15;
        temp = *data;
        printf("temp ="); UART_printword(temp); UART_newline();
        data++;
        temp2 = *data;
        temp2 <<= 8;
        printf("temp2 ="); UART_printword(temp2); UART_newline();
        temp |= temp2;
        first_cluster[num_of_files] = temp;

            UART_printword(first_cluster[num_of_files]);UART_newline();
        num_of_files++;
        data += 5;
        }
        else data += 21;
```

```
};
printf("\n\rNumber of files in SDC = %u\n\r",num_of_files);
for(index = 0; index < num_of_files; index++)
{
    UART_printword(first_cluster[index]);UART_newline();
}

}

#define CLUSTER_FAIL        0xffff
#define LAST_CLUSTER        0xfff8
#define LAST_CLUSTER_FAT16  0xfff8

//Obtaining a Cluster's Logical Block Adress
dword Cluster2Sector(word cluster)
{
    dword sector;
    //UART_printword(cluster);
    //Data clusters 0 and 1 don't exist.
    if(cluster == 0 || cluster ==1) sector = lba_root + cluster;

    else
    {
        //sec_per_clus <<= 9;//0x0800
        //UART_printdword(sec_per_clus);UART_newline();
        sector = ((dword)(cluster - 2) * sec_per_clus)<<9;
        sector += lba_data;
        //UART_printdword(sector);UART_newline();
    }
```

```
    return(sector);
}

//Reading From FAT
#define RAMreadW(a,f)   *(word *)(a+f)

word FAT_read(word cluster, byte *data)
{
    word next_cluster;
    dword sector;
    dword offset,cnt = 0;
    word index;

    offset = 0x04 + (2*(cluster-2));
    while(offset >= 512)
    {
    cnt++;
    offset -= 0x200;
    }
    //UART_printdword(offset);UART_newline();
    sector = lba_fat + (cnt << 9);
    //UART_printdword(sector);UART_newline();
    SD_get_sector(sector,data);

    next_cluster = *(data + offset);
    next_cluster += (word)*(data+1+offset)<<8;
    //UART_printword(next_cluster);UART_newline();
    return next_cluster;
}
```

```
/*
void FAT_clear_sector(unsigned long sector){
        // make surre the fatbuffer is invalidated
        fat_buf_sec = 0xffffffff;
        for(fat_u16 = 0; fat_u16 < 512; fat_u16++){
                fat_buf[fat_u16] = 0;
        };
        SD_write_sector(sector, fat_buf);
}
*/
                        *****#include <vs1011b.c>*****
void VS1011B_send_SCI(unsigned char reg, unsigned int data)
{
    unsigned int temp;
        // pull the CS line low
        VS_XCS_LO;
        delay_ms(1);
        // do the pseudo i2c start
        VS1011B_WRITE;
        spi_io(reg);
        // send the data
        temp = (data & 0xff00)>>8;
        spi_io(temp);
        temp = (data & 0x00ff);
        spi_io(temp);
        // pull the cs line back up
        VS_XCS_HI;
}

unsigned int VS1011B_read_SCI(unsigned char reg)
```

```
{
        unsigned int data;
        // pull the CS line low
        VS_XCS_LO;
        // do the pseudo i2c start
        VS1011B_READ;
        spi_io(reg);
        // get the reg data
        data = 0;
        data = spi_io(0x00);
        data <<= 8;
        data += spi_io(0x00);
        // pull the cs line back up
        VS_XCS_HI;
        return data;
}

void VS1011B_send_SDI(unsigned char data)
{
        VS_XDCS_LO;
        spi_io(data);
        VS_XDCS_HI;
}

void VS1011B_send_zeros(unsigned char count)
{
        do{
                VS1011B_send_SDI(0x0);
                count--;
        }while(count);
```

```
}

void VS1011B_SW_reset(void)
{
        unsigned int regval;
        regval |= (1 << SM_RESET);
        // set bit 2 of mode reg to 1 for reset
        VS1011B_send_SCI(REG_MODE, regval);
        while(VS1011B_NOT_ALLOW_DATA){};
        // for sanity
        delay_ms(50);
        regval = 0x00;
        regval |= (1 << SM_SDINEW); // set bit 11 of mode reg to 1 for work in native
mode(new mode)
    VS1011B_send_SCI(REG_MODE, regval);
    delay_ms(1);
        VS1011B_send_zeros(32);
}

void VS1011B_HW_reset(void)
{
    delay_ms(500);
        // pull down the reset pin for 1 second
        VS_XRESET_LO;
        // delay
        //DELAY_1SEC;
        delay_ms(500);
        // pull up again
        VS_XRESET_HI;
}
```

```
void VS1011B_init(void)
{
        // set XRESET output high;
        VS1011B_PORT |= (1 << PIN_VS1011B_XRESET);
        VS1011B_DDR  |= (1 << PIN_VS1011B_XRESET);

        // set XCS output HIGH
        VS1011B_PORT |= (1 << PIN_VS1011B_XCS);
        VS1011B_DDR  |= (1 << PIN_VS1011B_XCS);

        // set XDCS output HIGH
        VS1011B_PORT |= (1 << PIN_VS1011B_XDCS);
        VS1011B_DDR  |= (1 << PIN_VS1011B_XDCS);

        // set DREQ input with pull up
        VS1011B_PORT |= (1 << PIN_VS1011B_DREQ);
        VS1011B_DDR  &= ~(1 << PIN_VS1011B_DREQ);
        //printf("Init port to interface VS1011B\n\r");
        //reset
        VS1011B_HW_reset();
        //printf("Hardware Reset Already\n\r");
        VS1011B_SW_reset();
        //printf("Software Reset Already\n\r");
}

void VS1011B_sine(unsigned char state, unsigned char freq) // state is START or STOP, freq is
1-255
{
    unsigned int regval;
        regval |= (1 << SM_RESET);
```

```
// set bit 2 of mode reg to 1 for reset
        VS1011B_send_SCI(REG_MODE, regval);
        while(VS1011B_NOT_ALLOW_DATA){};
        // for sanity
        delay_ms(50);
        regval = 0x00;
        regval |= (1 << SM_SDINEW)|(1 << SM_TESTS); // set bit 11 of mode reg to 1 for work
in native mode(new mode)
    VS1011B_send_SCI(REG_MODE, regval);
    delay_ms(1);
        VS1011B_send_zeros(32);

        if(state == 0x01){
                VS1011B_send_SDI(0x53);
                VS1011B_send_SDI(0xEF);
                VS1011B_send_SDI(0x6E);
                VS1011B_send_SDI(freq);
                VS1011B_send_zeros(0x04);
        } else {
                VS1011B_send_SDI(0x45);
                VS1011B_send_SDI(0x78);
                VS1011B_send_SDI(0x69);
                VS1011B_send_SDI(0x74);
                VS1011B_send_zeros(0x04);

                VS1011B_SW_reset();
        };
}
```

```
// sets the Volume register for VS1001
void VS1011B_volume(unsigned char left, unsigned char right)
{

        unsigned int regval;
        regval = left;
        regval <<= 8;
        regval += right;
        VS1011B_send_SCI(REG_VOL, regval);
}


/*
// read the decode time in seconds from the VS1001
u16 VS1001_get_decode_time(void){
        return VS1001_read_SCI(REG_DECODETIME);
};*/

/*
// used to decode the bitrate val read from the AUDATA register
const unsigned int sample_rate_values[15] PROGMEM = {0, 44100, 48000, 32000, 22050,
24000, 16000, 11025, 12000, 8000};


// read the AUDATA register from VS1011B and fill structure
void VS1011B_get_audio_data(AUDIO_DATA* audio){
        unsigned int audata = VS1011B_read_SCI(REG_AUDATA);
        audio->sample_rate = pgm_read_word(sample_rate_values+((audata&0x1E00)>>9));
        audio->bitrate = audata&0x1FF;
        audio->is_stereo = (audata&0x8000)>>15;
```

```
} */
```

******* #include <player.c> *******

```
char scan_key(void);

void volume_up(void)
{
      vol_left--;
      vol_right--;
      if((vol_left <= 0) || (vol_right <= 0))
      {
      vol_left = 0;
      vol_right = 0;
      }

      VS1011B_volume(volume[vol_left],volume[vol_right]);
}

void volume_down(void)
{
      vol_left++;
      vol_right++;
      if((vol_left > 19) || (vol_right > 19))
      {
      vol_left = 0;
      vol_right = 0;
      }

      VS1011B_volume(volume[vol_left],volume[vol_right]);
}
```

```
byte play_sector(dword sector, byte *data)
{
    word index;

    SD_get_sector(sector,data);

      NO_ALLOW_SCI;

      for(index = 0; index < 512; index++)
      {
            if((512%32) == 0) WAIT_VS1011B_ALLOW_DATA;

                VS_XDCS_LO;

                spi_io(*data);

            delay_us(3);
            VS_XDCS_HI;
            //UART_printbyte(*data);
            data++;
      }

      // trigger the 74hc08 so that the MMC can receive data
      ALLOW_SCI;
      // get the 2nd CRC bytes
      ReadCRC();
      // give enough time
```

```
        SDC_cleanup();

}
void fplay(word cluster, byte *data)
{
    FILEOBJ fo;
    dword sector;
    word index,next_cluster,ccls;
    byte *backup;
    //byte backup_cluster;
    backup = data;
    //parameter cluster is The first cluster of file

    ccls = cluster;
    UART_printword(ccls);UART_newline();
    do
    {
    delay_us(10);
    scan_key();

    if((key_press == 'n')||(key_press == 'b'))
    {
    ccls = first_cluster[track];
    //backup_cluster = ccls;
    key_press = 'p';
    };
    if(key_press == 's')
    {
    ccls = first_cluster[track];
    };
```

```
        UART_SendByte(key_press);
        sector = Cluster2Sector(ccls);

            for(index = 0; index < sec_per_clus; index++)
            {
            play_sector(sector,data);
            sector += 0x200;
            }

        next_cluster = FAT_read(ccls,data);

        //UART_printword(next_cluster);UART_newline();
        if(next_cluster == 0xffff) break;
        else ccls++;

        }while(1);
}
void fplay2(dword addr, byte *data)
{
        FILEOBJ fo;
        dword sector;
        word index,next_cluster;
        byte *backup;
        backup = data;

        sector = addr<<9;
```

```
        UART_printdword(sector);UART_newline();
        do
        {
        //key_press = scan_key();

            for(index = 0; index < sec_per_clus; index++)
            {
            play_sector(sector,data);
            sector += 0x200;
            }

        }while(1);
}
                        ***** #include <keypad.c> *****
#include <ATMEGA32.h>



void keypad_init(void)
{


    DDRD = 0x00;
    PORTD = 0x00;
}

#define KEY_PLAY      PIND.2
#define KEY_STOP      PIND.3
#define KEY_NEXT      PIND.4
#define KEY_PREV      PIND.5
```

```
#define Vol_up        PIND.6
#define Vol_down      PIND.7
unsigned char Detect_play(void)
{
     if(KEY_PLAY == 1) return 0; //Card not presented
     else {printf("\tStart Play songs\n\r");return 1;} //Card presented
}
char scan_key(void)
{

     delay_ms(10);
     if(KEY_PLAY == 0)
     {
     delay_ms(10);
     //while(KEY_PLAY == 0){};
     //delay_ms(1);
     //while(KEY_PLAY == 1){};
     key_press = 'p';
     //UART_SendByte(press);
     }

     delay_ms(1);
     if(KEY_STOP == 0)
     {
     delay_ms(10);
     while(KEY_STOP == 0){};
     key_press = 's';
     //UART_SendByte(press);
     }
```

```
delay_ms(1);
if(KEY_NEXT == 0)
{
delay_ms(10);
while(KEY_NEXT == 0){};
key_press = 'n';
track++;
if(track > num_of_files) track = 0;
//UART_SendByte(press);
}

delay_ms(1);
if(KEY_PREV == 0)
{
delay_ms(10);
while(KEY_PREV == 0){};
key_press = 'b';
track--;
if(track < 0) track = num_of_files - 1;
//UART_SendByte(press);
}

delay_ms(1);
if(Vol_up == 0)
{
//while(ROW1 == 0){};
delay_ms(10);
//key_press = 'u';
//UART_SendByte(press);
```

```
volume_up();
}

delay_ms(1);
if(Vol_down == 0)
{
//while(ROW2 == 0){};
delay_ms(10);
//key_press = 'd';
//UART_SendByte(press);
volume_down();
}
}
```

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 32K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 1024 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 2K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega32L
  - 4.5 - 5.5V for ATmega32
- Speed Grades
  - 0 - 8 MHz for ATmega32L
  - 0 - 16 MHz for ATmega32
- Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

ATMEL®

8-bit AVR® Microcontroller with 32K Bytes In-System Programmable Flash

ATmega32
ATmega32L

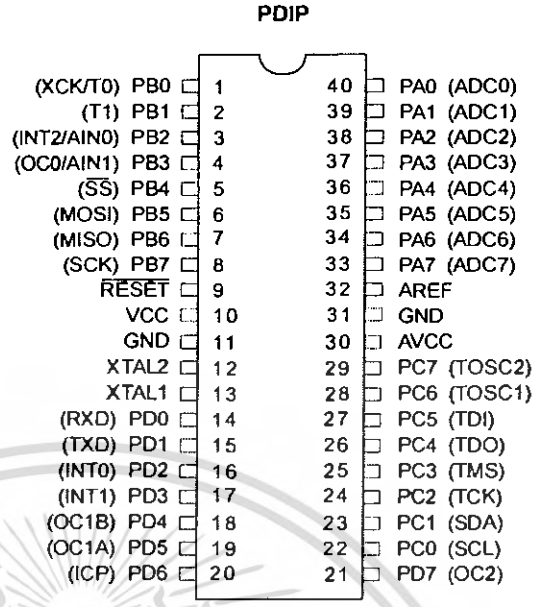
Preliminary

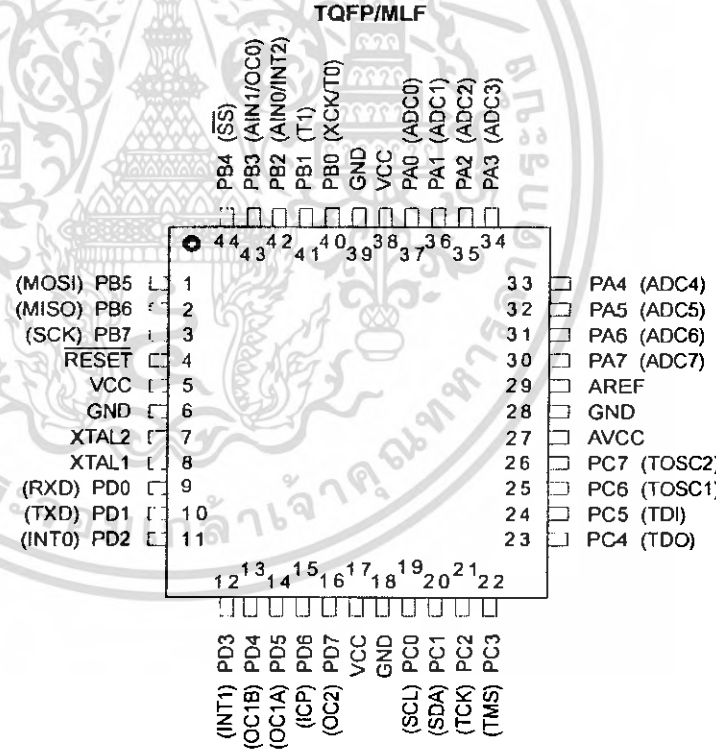
2503F–AVR–12/03

ATMEL

Pin Configurations

Figure 1. Pinouts ATmega32





Disclaimer

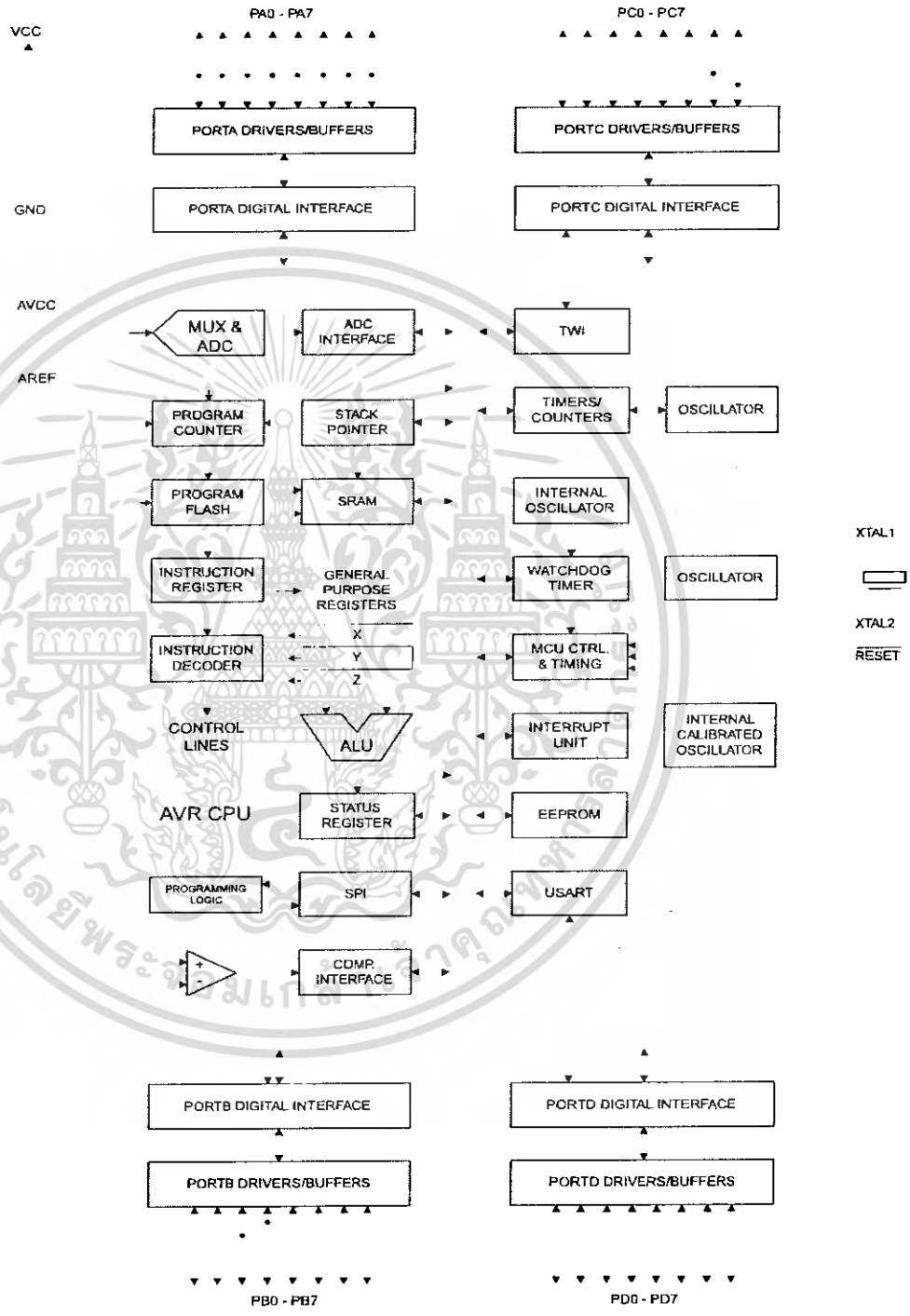
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Overview

The ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 2. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega32 provides the following features: 32K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024 bytes EEPROM, 2K byte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega32 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega32 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

VCC

Digital supply voltage.

GND

Ground.

Port A (PA7..PA0)

Port A serves as the analog inputs to the A/D Converter.

Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega32 as listed on page 55.

Port C (PC7..PC0)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

The TD0 pin is tri-stated unless TAP states that shift out data are entered.

Port C also serves the functions of the JTAG interface and other special features of the ATmega32 as listed on page 58.

Port D (PD7..PD0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega32 as listed on page 60.

$\overline{\text{RESET}}$

Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 35. Shorter pulses are not guaranteed to generate a reset.

XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting Oscillator amplifier.

AVCC

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.

AREF

AREF is the analog reference pin for the A/D Converter.

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	10
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								80
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	45, 65
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	66
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	80, 110, 128
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	81, 111, 128
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	246
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	175
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	30, 64
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	38, 65, 226
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	78
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								80
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								28
	OCDR	On-Chip Debug Register								222
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	54,83,129,196,216
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	105
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	108
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								109
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								109
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								109
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								109
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								109
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								109
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								110
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								110
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	123
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								125
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								125
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	126
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	40
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				162
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	160
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR8	EEAR8	17
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								17
$1D ($3D)	EEDR	EEPROM Data Register								17
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	17
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	62
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	62
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	62
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	62
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	62
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	63
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	63
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	63
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	63
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	63
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
$0F ($2F)	SPDR	SPI Data Register								136
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	136
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	134
$0C ($2C)	UDR	USART I/O Data Register								157
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	158
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	159
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								162
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	197
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	212
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	214
$05 ($25)	ADCH	ADC Data Register High Byte								215
$04 ($24)	ADCL	ADC Data Register Low Byte								215
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								177
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	177

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	176
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								175

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← Stack	None	4
RETI		Interrupt Return	PC ← Stack	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	Stack ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← Stack	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A

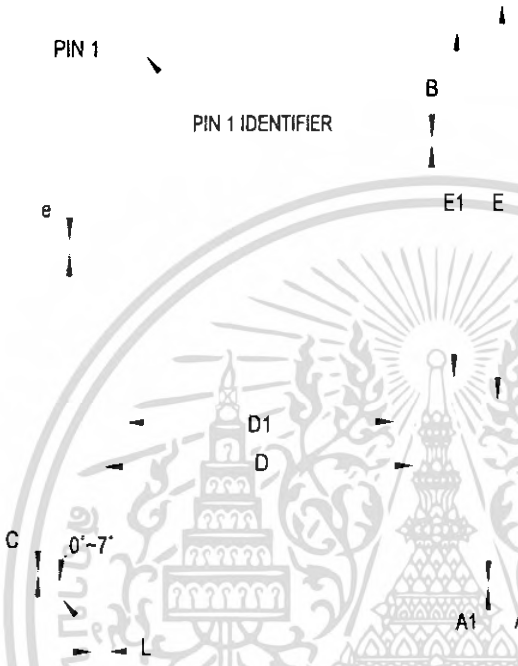
Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
8	2.7 - 5.5V	ATmega32L-8AC ATmega32L-8PC ATmega32L-8MC	44A 40P6 44M1	Commercial (0°C to 70°C)
		ATmega32L-8AI ATmega32L-8PI ATmega32L-8MI	44A 40P6 44M1	Industrial (-40°C to 85°C)
16	4.5 - 5.5V	ATmega32-16AC ATmega32-16PC ATmega32-16MI	44A 40P6 44M1	Commercial (0°C to 70°C)
		ATmega32-16AI ATmega32-16PI ATmega32-16MC	44A 40P6 44M1	Industrial (-40°C to 85°C)

Package Type	
44A	44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flat Package (TQFP)
40P6	40-pin, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44M1	44-pad, 7 x 7 x 1.0 mm body, lead pitch 0.50 mm, Micro Lead Frame Package (MLF)

Packaging Information

44A



COMMON DIMENSIONS
(Unit of Measure = mm)

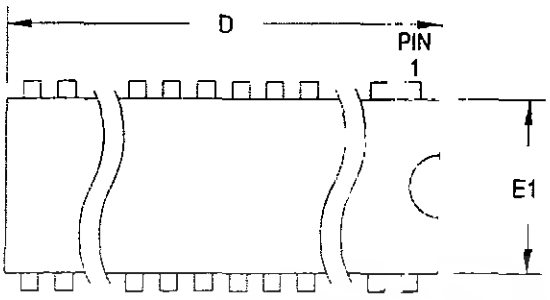
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	11.75	12.00	12.25	
D1	9.90	10.00	10.10	Note 2
E	11.75	12.00	12.25	
E1	9.90	10.00	10.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

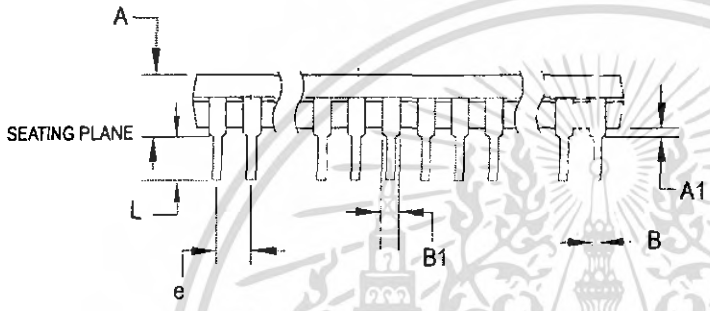
Notes:
1. This package conforms to JEDEC reference MS-026, Variation ACB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

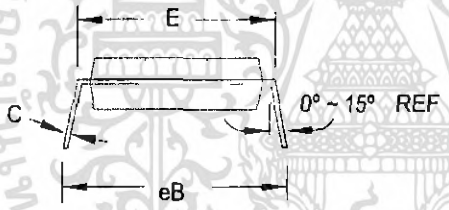
10/5/2001

	TITLE	DRAWING NO.	REV.
ATMEL 2325 Orchard Parkway San Jose, CA 95131	44A, 44-lead, 10 x 10 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	44A	B

40P6







COMMON DIMENSIONS
(Unit of Measure = mm)

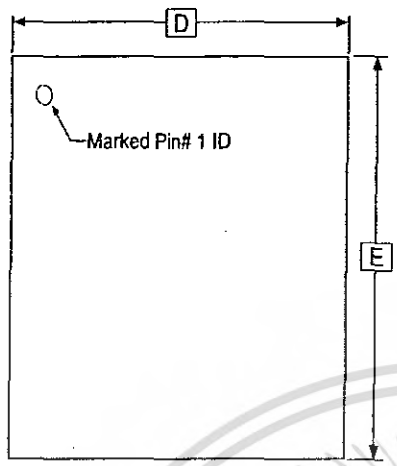
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.826	
A1	0.381	–	–	
D	52.070	–	52.578	Note 2
E	15.240	–	15.875	
E1	13.462	–	13.970	Note 2
B	0.356	–	0.559	
B1	1.041	–	1.651	
L	3.048	–	3.556	
C	0.203	–	0.381	
eB	15.494	–	17.526	
e	2.540 TYP			

Notes:
1. This package conforms to JEDEC reference MS-011, Variation AC.
2. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

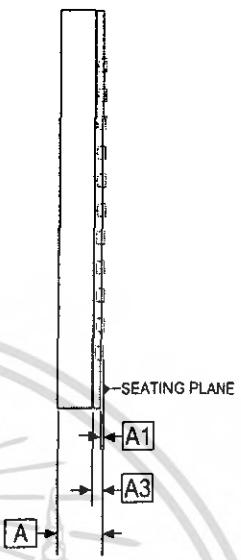
09/28/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**40P6**, 40-lead (0.600"/15.24 mm Wide) Plastic Dual *Inline Package (PDIP)*	40P6	B

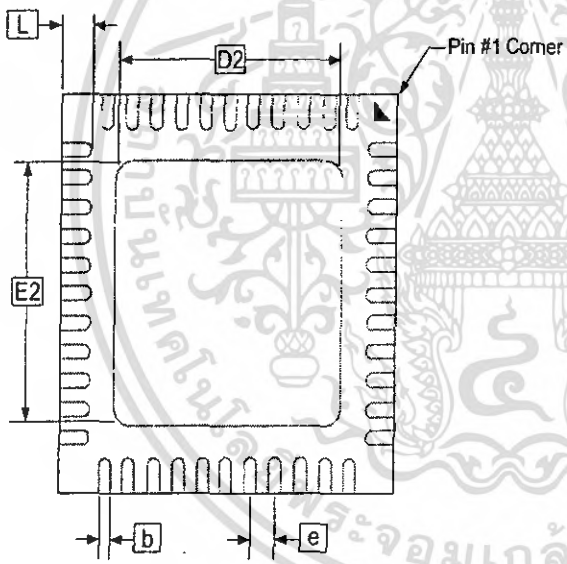
44M1



TOP VIEW



SIDE VIEW



BOTTOM VIEW

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A3		0.25 REF		
b	0.18	0.23	0.30	
D		7.00 BSC		
D2	5.00	5.20	5.40	
E		7.00 BSC		
E2	5.00	5.20	5.40	
e		0.50 BSC		
L	0.35	0.55	0.75	

Notes: 1. JEDEC Standard MO-220, Fig. 1 (SAW Singulation) VKKD-1.

01/15/03

	TITLE	DRAWING NO.	REV.
ATMEL 2325 Orchard Parkway San Jose, CA 95131	**44M1**, 44-pad, 7 x 7 x 1.0 mm Body, Lead Pitch 0.50 mm Micro Lead Frame Package (MLF)	44M1	C

Errata

ATmega32 Rev. A

There are no errata for this revision of ATmega32.However, a proposal for solving problems regarding the JTAG instruction IDCODE is presented below.

IDCODE masks data from TDI input

The public but optional JTAG instruction IDCODE is not implemented correctly according to IEEE1149.1; a logic one is scanned into the shift register instead of the TDI input while shifting the Device ID Register. Hence, captured data from the preceding devices in the boundary scan chain are lost and replaced by all-ones, and data to succeeding devices are replaced by all-ones during Update-DR.

If ATmega32 is the only device in the scan chain, the problem is not visible.

Problem Fix / Workaround

Select the Device ID Register of the ATmega32 (Either by issuing the IDCODE instruction or by entering the Test-Logic-Reset state of the TAP controller) to read out the contents of its Device ID Register and possibly data from succeeding devices of the scan chain. Note that data to succeeding devices cannot be entered during this scan, but data to preceding devices can. Issue the BYPASS instruction to the ATmega32 to select its Bypass Register while reading the Device ID Registers of preceding devices of the boundary scan chain. Never read data from succeeding devices in the boundary scan chain or upload data to the succeeding devices while the Device ID Register is selected for the ATmega32. Note that the IDCODE instruction is the default instruction selected by the Test-Logic-Reset state of the TAP-controller.

Alternative Problem Fix / Workaround

If the Device IDs of all devices in the boundary scan chain must be captured simultaneously (for instance if blind interrogation is used), the boundary scan chain can be connected in such way that the ATmega32 is the fist device in the chain. Update-DR will still not work for the succeeding devices in the boundary scan chain as long as IDCODE is present in the JTAG Instruction Register, but the Device ID registered cannot be uploaded in any case.

Datasheet Change Log for ATmega32

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

Changes from Rev. 2503E-09/03 to Rev. 2503F-12/03

1. Updated "Calibrated Internal RC Oscillator" on page 27.

Changes from Rev. 2503D-02/03 to Rev. 2503E-09/03

1. Updated and changed "On-chip Debug System" to "JTAG Interface and On-chip Debug System" on page 33.
2. Updated Table 15 on page 35.
3. Updated "Test Access Port – TAP" on page 217 regarding the JTAGEN fuse.
4. Updated description for Bit 7 – JTD: JTAG Interface Disable on page 226.
5. Added a note regarding JTAGEN fuse to Table 105 on page 255.
6. Updated Absolute Maximum Ratings* , DC Characteristics and ADC Characteristics in "Electrical Characteristics" on page 285.
7. Added a proposal for solving problems regarding the JTAG instruction IDCODE in "Errata" on page 15.

Changes from Rev. 2503C-10/02 to Rev. 2503D-02/03

1. Added EEAR9 in EEARH in "Register Summary" on page 6.
2. Added Chip Erase as a first step in"Programming the Flash" on page 282 and "Programming the EEPROM" on page 283.
3. Removed reference to "Multi-purpose Oscillator" application note and "32 kHz Crystal Oscillator" application note, which do not exist.
4. Added information about PWM symmetry for Timer0 and Timer2.
5. Added note in "Filling the Temporary Buffer (Page Loading)" on page 249 about writing to the EEPROM during an SPM Page Load.
6. Added "Power Consumption" data in "Features" on page 1.
7. Added section "EEPROM Write During Power-down Sleep Mode" on page 20.
8. Added note about Differential Mode with Auto Triggering in "Prescaling and Conversion Timing" on page 202.
9. Updated Table 90 on page 230.
10. Added updated "Packaging Information" on page 12.

Changes from Rev. 2503B-10/02 to Rev. 2503C-10/02

1. Updated the "DC Characteristics" on page 285.

Changes from Rev. 2503A-03/02 to Rev. 2503B-10/02

1. Canged the endurance on the Flash to 10,000 Write/Erase Cycles.
2. Bit nr.4 – ADHSM – in SFIOR Register removed.
3. Added the section "Default Clock Source" on page 23.
4. When using External Clock there are some limitations regards to change of frequency. This is described in "External Clock" on page 29 and Table 118 on page 287.
5. Added a sub section regarding OCD-system and power consumption in the section "Minimizing Power Consumption" on page 32.
6. Corrected typo (WGM-bit setting) for:
   - "Fast PWM Mode" on page 73 (Timer/Counter0)
   - "Phase Correct PWM Mode" on page 74 (Timer/Counter0)
   - "Fast PWM Mode" on page 118 (Timer/Counter2)
   - "Phase Correct PWM Mode" on page 119 (Timer/Counter2)
7. Corrected Table 67 on page 162 (USART).
8. Updated V_{IL}, I_{IL}, and I_{IH} parameter in "DC Characteristics" on page 285.
9. Updated Description of OSCCAL Calibration Byte.
   In the datasheet, it was not explained how to take advantage of the calibration bytes for 2, 4, and 8 MHz Oscillator selections. This is now added in the following sections:
   Improved description of "Oscillator Calibration Register – OSCCAL" on page 28 and "Calibration Byte" on page 256.
10. Corrected typo in Table 42.
11. Corrected description in Table 45 and Table 46.
12. Updated Table 119, Table 121, and Table 122.
13. Added "Errata" on page 15.

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2503F–AVR–12/03