คีย์บอร์ดเปล่งเสียง

SPEAKING KEYBOARD

โดย

นายวีรวัฒน์ ศิริวงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2550

คีย์บอร์ดเปล่งเสียง

SPEAKING KEYBOARD

โดย

นายวีรวัฒน์ ศิริวงษ์

อาจารย์ที่ปรึกษา

อ.เฉลิมพันธ์ หวังวิวัฒนา

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง คีย์บอร์ดเปล่งเสียง (SPEAKING KEYBOARD)

ผู้จัดทำ นายวีรวัฒน์ ศิริวงษ์

_____อาจารย์ที่ปรึกษา

(อ.เฉลิมพันธ์ หวังวิวัฒนา)

คีย์บอร์ดเปล่งเสียง

นายวีรวัฒน์ ศิริวงษ์ รหัส 47010726
อ.เฉลิมพันธ์ หวังวิวัฒนา อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

บทคัดย่อ

โครงงานนี้ได้ทำการออกแบบวงจรเพื่อแสดงผลข้อมูลตัวอักษร ที่พิมพ์ลงบนคีย์บอร์ดของคอมพิวเตอร์ ให้ออกมาเป็นเสียง โดยใช้ microcontroller ตรวจจับตำแหน่งตัวอักษรของคีย์บอร์ด โดยใช้รหัส Scan code ของคีย์บอร์ด จากนั้นส่งข้อมูลของตำแหน่งที่ได้ ไปให้ชิพเสียงเพื่อเปล่งเสียงหรือแสดงผลการเปรียบเทียบออกทางลำโพง

SPEAKING KEYBOARD

Mr. Veerawat Sirivong ID.47010726

Mr. Chaleomphan Wangwiwattana Advisor

Educational Year 2007

Abstract

This project is a design of speaking keyboard which sends sounds when a keyswitch is presses. We use a microcontroller to detect the alphabets on the keyboard by scan code of keyboard. Then it transfers the information of its position to the sound chip which plays the sound on a speaker.

กิตติกรรมประกาศ

โครงงานนี้จะไม่สามารถสำเร็จลุล่วงไปได้ด้วยดี หากขาด อาจารย์ จิรวัฒน์ ปานกลาง อาจารย์ เฉลิมพันธ์ หวังวิวัฒนา ที่ให้คำปรึกษาและห้องปฏิบัติการที่มีอุปกรณ์ครบสมบูรณ์ อาจารย์และพี่ๆที่ให้คำปรึกษา ชุมนุมอิเล็กทรอนิกส์คลับ ที่ช่วยเหลือทางด้านสถานที่และอุปกรณ์ ซึ่งเป็นประโยชน์อย่างมาก เพื่อนๆทุกคนที่ช่วยซึ่งกันและกัน และที่สำคัญคือการสนันสนุนทางด้านเงินทุนและกำลังใจ จากบิดามารดา

นายวีรวัฒน์ ศิริวงษ์
ผู้จัดทำ

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

เนื่องจากในปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์ได้เข้ามามีบทบาทต่อชีวิตประจำวันเป็นอย่างมาก ทั้งในด้านการเรียนรู้ การทำงาน การสื่อสาร และความบันเทิง

โครงงานนี้จึงได้มีแนวคิดการทำคีย์บอร์ดที่แสดงผลออกมาเป็นเสียง เพื่อเพิ่มประสิทธิภาพในการใช้งาน นอกจากนี้ยังสามารถใช้ในการเรียนรู้การใช้คีย์บอร์ดในเบื้องต้น และยังเป็นแนวทางในการพัฒนาอุปกรณ์สำหรับผู้พิการทางสายตา

วัตถุประสงค์

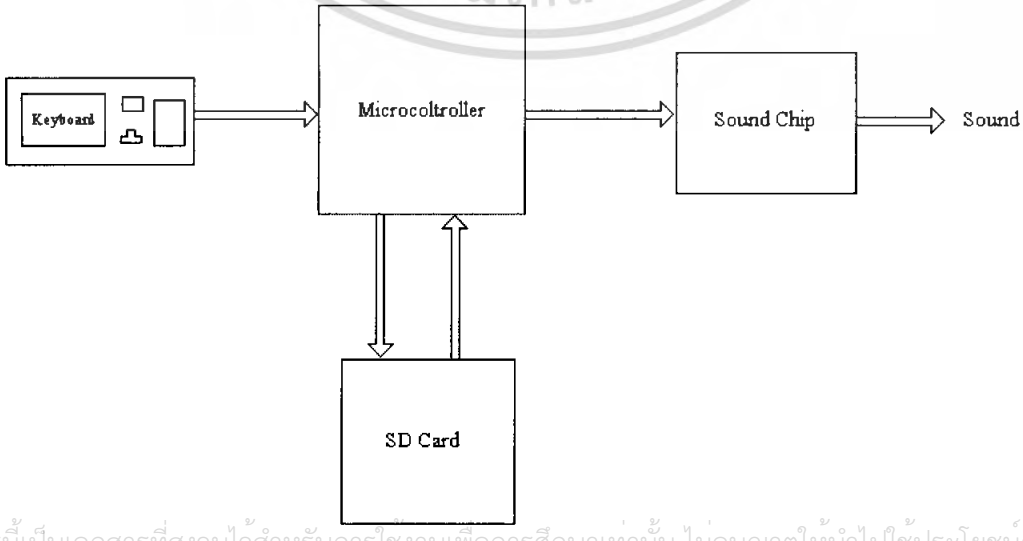
1. เพื่อศึกษาการทำงานของไมโครคอนโทรลเลอร์ในการทำงานเพื่อควบคุมอุปกรณ์ต่างๆ
2. สามารถใช้ภาษา C ในการควบคุมการทำงานของไมโครคอนโทรลเลอร์ได้
3. เพื่อเป็นแนวทางในการพัฒนาอุปกรณ์เพื่อผู้พิการทางสายตา

ขอบเขตโครงงาน

1.ใช้ไมโครคอนโทรลเลอร์ในการควบคุมอุปกรณ์ต่างๆ

2.ใช้คีย์บอร์ดเป็นตัวรับข้อมูลตัวอักษรภาษาอังกฤษแล้วส่งข้อมูลไปให้กับไมโครคอนโทรลเลอร์

3.ใช้ชิพเสียงแปลงข้อมูลเสียงแล้วส่งออกมาทางลำโพง



บทที่ 2
หลักการและทฤษฎีที่เกี่ยวข้อง

2.1 ไมโครคอนโทรลเลอร์

2.1.1 ไมโครคอนโทรลเลอร์ คืออะไร

ไมโครคอนโทรลเลอร์ เป็นอุปกรณ์อิเล็กทรอนิกส์ที่สามารถทำงานตามเงื่อนไขต่างๆตามที่เราเขียนหรือตั้งโปรแกรมไว้โดยที่ตัวไมโครคอนโทรลเลอร์ เองสามารถเชื่อมต่อกับอุปกรณ์ภายนอกได้ทันที(แล้วแต่เบอร์และคุณสมบัติของเบอร์นั้นๆ)เราจึงสามารถนำไมโครคอนโทรลเลอร์ไปประยุกต์ใช้ในงานการควบคุมต่างๆ มากมาย เช่น การควบคุมมอเตอร์ การควบคุมหลอดไฟ หรือการควบคุมการทำงานของหุ่นยนต์เป็นต้น

แต่หลายคนคงเคยได้ยินคำว่า ไมโครโปรเซสเซอร์ หรือ โปรเซสเซอร์มาแล้ว แล้ว แต่ไมโครโปรเซสเซอร์โดยทั่วไปจะทำหน้าที่ ประมวลผล และทำงานเร็วมาก แต่ไม่เหมาะนำมาทำงานในลักษณะการเชื่อมต่อกับอุปกรณ์ภายนอกได้ (สามารถทำได้แต่ต้องใช้อุปกรณ์รอบข้างเสริมมาก เช่น เบอร์ Z80 เป็นต้น)

ดังนั้นในงานควบคุมขนาดเล็กเรานิยมใช้ไมโครคอนโทรลเลอร์เข้ามาใช้งานมากกว่าด้วยสาเหตุ

-ไมโครคอนโทรลเลอร์มีหลายค่าย หลายเบอร์ให้เลือกใช้งาน

-ไมโครคอนโทรลเลอร์มีขนาดเล็ก และราคาถูก

-ไมโครคอนโทรลเลอร์ปัจจุบันสามารถเขียนโปรแกรมได้หลายภาษา เช่น C เป็นต้น ทำให้เรียนรู้ได้เร็ว มีเครื่องมือสนับสนุนในการทำงานมากมาย

ซึ่งในปัจจุบันมีหนังสือให้ความรู้ทางด้านไมโครคอนโทรลเลอร์มากมายทั้งภาษาไทย และต่างประเทศ

2.1.2 ส่วนประกอบภายในของไมโครคอนโทรลเลอร์

1. ส่วนประมวลผล (Processing unit) ทำหน้าที่คำนวณทางคณิตศาสตร์และการตัดสินใจแบบเงื่อนไข(Logic)

2. ส่วนเก็บข้อมูล (Memory) ทำหน้าที่เก็บข้อมูลต่างๆซึ่งสามารถแบ่งออกได้เป็น

-เก็บแบบชั่วคราว (RAM) จะเก็บได้เมื่อมีไฟเลี้ยงอยู่ และเมื่อไม่มีไฟเลี้ยงข้อมูลจะสูญหาย

-เก็บแบบถาวร(EPROM) จะใช้ในการเก็บ code เป็นส่วนใหญ่ ข้อมูลไม่หายเมื่อไม่มีไฟเลี้ยง

3. ส่วนเชื่อมต่อ หรือ port ต่างๆ ซึ่งทำหน้าที่รับ-ส่งสัญญาณให้กับอุปกรณ์ภายนอกได้

4. ส่วนกำเนิดสัญญาณนาฬิกา โดยที่ตัวไมโครคอนโทรลเลอร์นั้นจะสามารถทำงานได้เมื่อมีสัญญาณนาฬิกา ส่วนมากเราจะใช้ คริสตอล(X-TAL) มาเป็นตัวกำเนิดสัญญาณจากภายนอกก่อนส่งไปภายในตัวไมโครคอนโทรลเลอร์

2.1.3การเรียนรู้การใช้งานไมโครคอนโทรลเลอร์

การใช้งานของไมโครคอนโทลเลอร์นั้นเราไม่อาจจะนำเอาตัวไอซีชนิดนี้ไปใช้งานเดี่ยวๆได้ แต่จะมีวงจรอิเล็กทรอนิกส์ภายนอกต่อร่วมด้วยเสมอ ดังนั้นการที่จะใช้งานไมโครคอนโทรลเลอร์ได้ดีนั้นจำเป็นต้องมีพื้นฐานของการใช้งานวงจรอิเล็กทรอนิกส์ด้วยเช่นเดียวกัน นอกจากความรู้ทางด้านอิเล็กทรอนิกส์ แล้วยังต้องมีความเข้าใจลักษณะสถาปัตยกรรมของไมโครคอนโทลเลอร์ตัวนั้นๆรวมทั้งการมีเครื่องมือในการทดลองและทดสอบการเขียนโปรแกรม

ภาษาที่ใช้ในการเขียนโปรแกรมของไมโครคอนโทรลเลอร์ปัจจุบันมีมีตัวคอมไพเลอร์ภาษาต่างๆให้เลือกใช้ เช่นภาษาแอสเซมบลี ภาษา C ภาษาเบสิก เป็นต้น ส่วนจะเลือกใช้ภาษาไหนก็แล้วแต่ตามใจชอบครับภาษาแอสเซมบลี เป็นภาษาที่มีความสามารถสูง แต่ใช้เวลาศึกษานาน เป็นภาษาแรกๆ ในการใช้งานสมัยก่อนหน้านี้ภาษาเบสิก เป็นภาษาที่เขียนได้ง่าย ใช้เวลาเรียนรู้ได้เร็ว ภาษา C เช่นเดียวกับภาษาเบสิก คือ เขียนง่าย มีความสามารถใกล้เคียงกับ แอสเซมบลี เรียนรู้ได้เร็ว เนื่องจากตัวไมโครคอนโทรลเลอร์มีหลายมีหลายตระกูลและหลายเบอร์แต่ละเบอร์ก็มีความสามารถแตกต่างกันไปดังนั้น หากต้องการศึกษาเบอร์ไมโครคอนโทรลเลอร์ เบอร์ใด สามารถศึกษาได้จาก datasheet ของเบอร์นั้นๆ

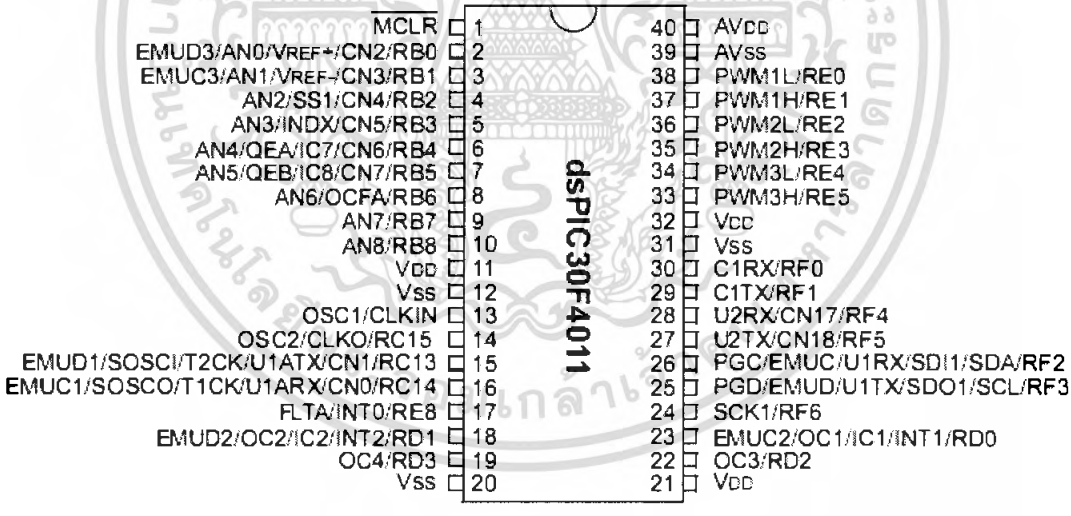
2.1.4 PIC คือ อะไร

PIC คือไมโครคอนโทรลเลอร์อีกตระกูลหนึ่ง ย่อมาจากคำว่า Peripheral Interface Controller ซึ่ง concept ของเจ้าไมโครคอนโทรลเลอร์ตระกูลนี้ก็คือ พยายามรวมเอาทุกอย่างเอาไว้ในตัวของมันไม่ว่าจะเป็น PROGRAM MEMROY, RAM, EEPROM, SERIAL, I2C, PWM, A/D ฯลฯ โดยไม่จำเป็นต้องต่ออุปกรณ์เสริมจากภายนอก ในตัวของ PIC จะมีฟังก์ชันที่ใช้ในการประมวลผล รวมทั้งหน่วยความจำ ซึ่งทำให้มันเหมือนกัน CPU ตัวหนึ่งเลยทีเดียว ความเร็วของ PIC ภาคของความถี่สัญญาณนาฬิกา ปัจจุบันสามารถทำสัญญาณนาฬิกาได้ที่ 20 เมกะเฮิรตซ์ ซึ่งทำให้หนึ่งคำสั่งของ PIC ใช้เวลาเพียง 0.25 ไมโครวินาที แต่อย่างไรก็ตามได้มีบริษัทอื่นได้ซื้อลิขสิทธิ์ PIC จาก Microchip และได้สร้างชิพที่มีความเร็วได้มากกว่าเดิมขึ้นไปอีก

ในอดีตหน่วยความจำของ PIC จะค่อนข้างน้อย คืออยู่ระหว่าง 512 words ถึง 4K words แต่ในปัจจุบัน บริษัท Microchip ซึ่งเป็นเจ้าของ PIC ได้พัฒนาจนทำให้หน่วยความจำของ PIC มีขนาดเป็นหลายสิบกิโลไบต์ และมีทีท่าว่าจะขยายได้ใหญ่ขึ้นเรื่อยๆ ในเรื่องของการนับขนาดของหน่วยความจำของ PIC จะนับไม่เหมือนปกติ โดยที่ หนึ่งคำสั่งของ PIC จะมีขนาด 14 bits ดังนั้นเราจะเรียกว่า 1 word ของ PIC จะมีขนาด 14 bits เช่น PIC16F84A ระบุว่ามีหน่วยความจำ 1 K (ซึ่งหมายถึง 1 Kword ถ้าคำนวณให้เป็นแบบ 1 byte = 8 bit จะได้ว่า 1 x 1,024 x 14 = 14,336 bits ดังนั้นก็คือ 14,336/(8 x 1,024) = 1.75K bytes นั่นเอง

สถาปัตยกรรมของ PIC ตอนนี้มี 3 สายหลักๆ สมัยก่อนมีแค่สอง คือขึ้นต้นด้วย 16xxx,17xxx และ 18xxx ถ้าพูดถึง คุณสมบัติที่เหนือกว่าเรียงจากน้อยสุดไปมากสุดก็คือ 16 -> 17 -> 18 คำสั่งภาษาแอสเซมบลี ของ 17 และ 18 จะมีมากกว่า 16 ทำให้เขียนโปรแกรมได้ง่ายกว่า ราคาก็จะสูงกว่าด้วย แต่ที่เป็นที่นิยมก็คือตระกูล 16xxx

dsPIC เป็น mcu รุ่นที่สามต่อจาก รุ่น 16,17 -> 18 -> dsPIC โดยที่ขนาด ของ Architecture จะใหญ่ขึ้นเรื่อยๆ ราคา และ ประสิทธิภาพ ก็ใหญ่ตาม เช่นกัน

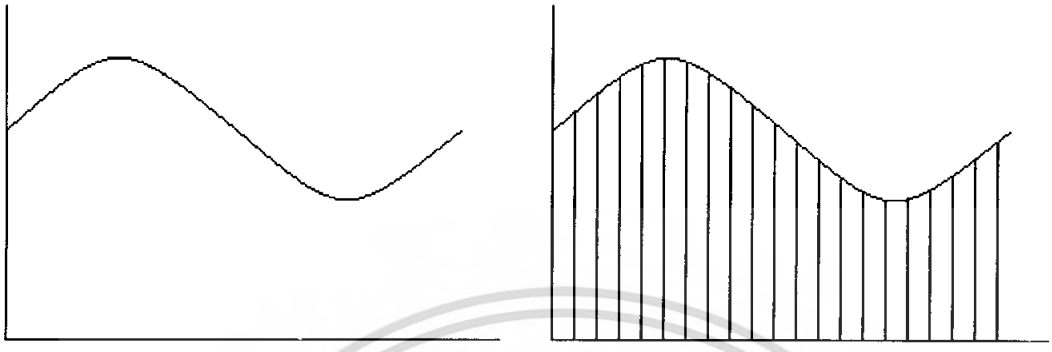


รูปที่ 2.1 dsPIC 30F4011

2.2 เสียงแบบดิจิตอล

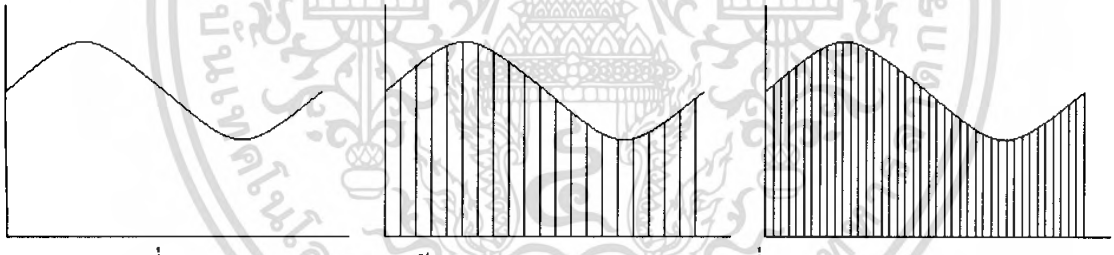
โดยปกติเสียงมีรูปแบบเป็นคลื่น และเป็นสัญญาณแบบอนาลอก การนำเสียงมาใช้งานและจัดเก็บลงในระบบคอมพิวเตอร์ จำเป็นจะต้องทำการแปลงเสียงให้อยู่ในรูปแบบอนาลอกให้เป็นรูปแบบดิจิตอล เรียกว่า การสุ่มสัญญาณ (Sampling) วิธีการดังกล่าว คือ ทำการสุ่มสัญญาณเสียง

ด้วยความถี่สม่ำเสมอค่าหนึ่ง เพื่อวัดค่าของสัญญาณ ณ ตำแหน่งนั้น ๆ และแปลงสัญญาณดังกล่าวให้เป็นดิจิตอล ดังรูป



รูปที่ 2.2 การ Sampling สัญญาณเสียง

จำนวนครั้งของการสุ่มสัญญาณเสียงในหนึ่งวินาที เรียกว่า อัตราการสุ่ม (Sampling Rate) หากอัตราการสุ่มมีค่าสูง สัญญาณดิจิตอลที่ได้จะมีความใกล้เคียงกับสัญญาณอนาลอกต้นฉบับ หากอัตราการสุ่มมีค่าต่ำสัญญาณดิจิตอลที่ได้ จะมีความผิดเพี้ยนจากสัญญาณอนาลอกต้นฉบับมาก ดังรูป



รูปที่ 2.3 สัญญาณอนาลอกต้นฉบับ และสัญญาณดิจิตอลที่อัตราการสุ่มค่าต่าง ๆ

สัญญาณเสียงที่บันทึกลงในคอมแพคดิสก์มีอัตราการสุ่มที่ 44100 ครั้งต่อวินาที หรือ 44.1 kHz ในการสุ่มสัญญาณแต่ละครั้ง สัญญาณอนาลอกต้นฉบับจะถูกจัดระดับเพื่อแปลงเป็นสัญญาณดิจิตอล จำนวนระดับมีผลต่อคุณภาพของเสียงที่ได้รับฟัง หากจำนวนของระดับมีมากเสียงที่ได้รับฟังจะมีคุณภาพดี หากจำนวนของระดับมีน้อย เสียงที่ได้รับฟังจะมีคุณภาพไม่ดี หากแบ่งระดับออกเป็นสองระดับ จะต้องใช้จำนวนเต็มฐานสองจำนวนหนึ่งบิต ในการแทนค่าสัญญาณสองระดับ คือ “0” และ “1” หากแบ่งระดับออกเป็นสี่ระดับ จะต้องใช้จำนวนเต็มฐานสองจำนวนสองบิต ในการแทนค่าสัญญาณสี่ระดับ คือ “00” “01” “10” และ “11” สัญญาณเสียงที่บันทึกลงในคอมแพคดิสก์มีการแบ่งระดับสูงถึง 65536 ระดับ ซึ่งจะต้องใช้จำนวนเต็มฐานสองในการแทนค่าสัญญาณจำนวน 16 บิต ดังนั้นหากต้องการบันทึกเสียงที่มีความยาวหนึ่งวินาที จะต้องใช้พื้นที่ในการจัดเก็บดังนี้

อัตราการสุ่ม x จำนวนบิตที่ใช้ในการแบ่งระดับ x จำนวนลำโพง

ในกรณีของคอมแพคดิสก์ที่บันทึกเสียงแบบสองลำโพง (Stereo) จะต้องใช้พื้นที่ในการจัดเก็บเสียงความยาวหนึ่งวินาทีเท่ากับ

44100 x 16 x 2 = 1411200 บิต หรือ 176400 ไบต์

หากต้องการบันทึกเสียงความยาว 45 นาทีลงในคอมแพคดิสก์จะต้องใช้พื้นที่ในการจัดเก็บมากถึง 475 MB

2.2.1 การบีบอัดข้อมูลเสียง

เนื่องจากปริมาณของข้อมูลเสียงมีมาก พื้นที่ในการจัดเก็บข้อมูลของสื่อแบบเคลื่อนที่ได้อาจมีไม่เพียงพอ ดังนั้นเทคนิคการบีบอัดข้อมูลจึงเข้ามามีบทบาทในการจัดเก็บข้อมูล รูปแบบการบีบอัดข้อมูลที่ได้รับความนิยมมีดังนี้

1. MIDI (Musical Instrument Digital Interface : 31.25 Kbs data rate) รูปแบบของเสียงเพลงดิจิตอลที่สามารถสร้างได้จากโปรแกรมคอมพิวเตอร์หรือเครื่องดนตรีต่างๆที่สามารถพ่วงต่อใช้งานกับคอมพิวเตอร์ได้ เนื่องจากไฟล์ตระกูล MIDI หรือ MID นี้มีขนาดเล็กจึงสามารถนำไปประยุกต์ใช้งานได้หลายอย่าง เช่น ใช้เปิดเพลงบนเว็บไซต์ สร้างคาราโอเกะด้วยตัวเอง เป็นต้น

2. WAV (Wave File) ไฟล์เสียงที่มักจะคุ้นเคยกันมากที่สุด เป็นรูปแบบของไฟล์เสียง ซึ่งกลายเป็นมาตรฐานเสียงเพลง เกมส์ของเครื่องคอมพิวเตอร์ PC มีนามสกุล *.wav ใช้งานได้ทั้งเครื่องคอมพิวเตอร์ PC และ Macintosh แต่มีขนาดใหญ่มาก

3. Ripper เป็นวิธีการหนึ่งของการแปลงไฟล์เสียงต่างๆ จากคอมแพ็คดิสก์ไปเก็บไว้ที่ฮาร์ดดิสก์ของคอมพิวเตอร์ แผ่นดิสก์หรือสื่อบันทึกอื่นๆ ได้ในรูปแบบของไฟล์เสียง *.wav โดยปกติแล้วมักจะนำ โปรแกรมประเภทนี้มาใช้ในขั้นตอนการแปลงเสียงเพลงไปสร้างเป็นเพลง MP3 (Encoder) เพราะคุณภาพของเสียงเพลงที่ใกล้เคียงกันแต่ขนาดของไฟล์เพลงที่น้อยลง ประหยัดพื้นที่ที่ใช้เก็บไฟล์เพลงได้มากขึ้น

4. MP3 หรือ MPEG-1 Layer-3 คือ การเข้ารหัสไฟล์เสียงแบบหนึ่ง ซึ่งใช้ประโยชน์จากคุณสมบัติการรับฟังของหูมนุษย์ โดยพยายามคงคุณภาพเสียงไว้ให้ได้มากที่สุด หรือที่เรียกว่าวิธีการบีบอัดข้อมูล ซึ่งจะทำให้มีไฟล์มีขนาดเล็กลง ส่วนใหญ่จะเป็นพวกเพลง เสียงประกอบที่ใช้ในเว็บ เนื่องจากมันมีขนาดเล็กมากกว่าไฟล์ *.wav หรือไฟล์เสียงทั่วไป ประมาณ 5-10 เท่า โดยใช้มาตรฐาน MPEG (Moving Picture Expert Group)

2.2.2 มาตรฐาน MPEG (Moving Picture Expert Group)

MP3 ไม่ได้เป็น MPEG-3 อย่างที่ใครหลายคนเข้าใจ ใน MPEG-1 จะแบ่งเป็นหลายส่วน เช่น ส่วนของภาพ ส่วนของเสียง ส่วนของมีเดีย สามารถหยิบเฉพาะบางส่วนไปใช้งานจริงได้ และส่วนของเสียงใน MPEG-1 คือส่วนที่เรียกว่า Layer 2 และ Layer 3 ซึ่ง Layer 2 นั้นตกสมัยไปแล้ว ส่วน MPEG-1 Layer 3 ก็คือ MP3 นั่นเอง

2.2.3 ความเหมือนของ MPEG Layer 1 ,Layer 2 และ Layer 3

เลเยอร์ข้อมูลทั้งหมดจะใช้โครงสร้างพื้นฐานเช่นเดียวกัน โดยกลยุทธ์การเข้ารหัส สามารถอธิบายได้ว่าเป็น "การกำหนดรูปของเสียงรบกวนอย่างฉลาด" ตัวเข้ารหัสจะทำการวิเคราะห์ Spectral Components ของสัญญาณเสียงด้วยการคำนวณ Filterbank และทำการนำเอา Psychoacoustic Model ใส่เข้าไปเพื่อคาดการณ์ Noise-level ที่สามารถสังเกตได้ ในการเข้ารหัสและ Quantization นั้น ตัวเข้ารหัสจะพยายามกำหนดจำนวนของบิตข้อมูลในทิศทางที่เข้ากันได้ กับทั้งข้อกำหนดบิตเรตและมาสค์กิ้ง ส่วนตัวถอดรหัสนั้นจะซับซ้อนน้อยกว่ามาก เพราะงานเพียงอย่างเดียวก็คือการประกอบสัญญาณเสียงกลับคืนมาจาก Spectral Components ที่ได้เข้ารหัสเอาไว้ เลเยอร์ทั้งหมดนั้นจะใช้ Analysis Filterbank เหมือนกันทั้งหมด Layer-3 จะเพิ่ม MDCT Transform เพื่อเพิ่มความละเอียดของความถี่ เลเยอร์ทั้งหมดจะใช้ "ข้อมูลเฮดเดอร์" เดียวกันในบิตสตรีม เพื่อสนับสนุนรูปแบบมาตรฐานที่เข้ากันได้ในแบบลำดับชั้นที่สูงขึ้น เลเยอร์ทั้งหมดจะมีความอ่อนไหวในบิตเออเรอร์เหมือนกัน ทั้งหมดจะใช้โครงสร้างบิตสตรีมซึ่งบรรจุส่วนต่างๆ ซึ่งมีความอ่อนไหวต่อบิตเออเรอร์เหมือนกัน เลเยอร์ทั้งหมดจะสนับสนุนการเพิ่มเติมข้อมูลเข้าไปในบิตสตรีมข้อมูลเสียงเหมือนกัน เลเยอร์ทั้งหมดสามารถใช้ความถี่ Sampling 32 ,44.1 หรือ 48 kHz ได้เหมือนกัน ช่วงความถี่ของการบีบย่อในรูปแบบของ Layer-1 ,2 และ 3 แจกแจงได้ดังนี้

Layer-1 : จาก 32 kbps ถึง 448 kbps

Layer-2 : จาก 32 kbps ถึง 384 kbps

Layer-3 : จาก 32 kbps ถึง 320 kbps

2.2.4 ความแตกต่างกันของ MPEG Layer 1 ,Layer 2 และ Layer 3

จาก Layer-1 ถึง Layer-3 นั้นมีการเพิ่มความซับซ้อนขึ้นเรื่อยๆ (ส่วนใหญ่จะจริงในส่วน ของตัวเข้ารหัส) การล่าช้าของ Codec โดยรวมมีมากขึ้นและประสิทธิภาพเพิ่มสูงขึ้น (คุณภาพเสียงต่อบิตเรต) จุดเด่นของ MP3 ที่เหนือกว่า Layer-1 และ Layer-2 ก็อยู่ที่ Layer-1 นั้น จะได้รับการออกแบบหลักๆ เพื่อใช้สำ หรับ DCC (Digital Compact Cassette) ซึ่งใช้อัตราส่วนอยู่ที่ 384 kbps

(เรียกว่า "PASC") ส่วน Layer-2 นั้นได้รับการออกแบบให้แลกระหว่างความซับซ้อนกับประสิทธิภาพโดยมันจะมีคุณภาพเสียงดีที่บิตเรตได้ต่ำถึง 192 kbps แต่ถ้าต่ำกว่านั้นคุณภาพเสียงก็จะแย่ลง ส่วน Layer-3 นั้นได้รับการออกแบบสำหรับอัตราบิตเรตต่ำๆ มาตั้งแต่ต้น โดยได้มีการเพิ่มคุณสมบัติชั้นสูงเข้าไปใน Layer-2 ได้แก่ความละเอียดอัตราความถี่ที่สูงขึ้นถึง 18 เท่า ซึ่งช่วยให้ตัวเข้ารหัส Layer-3 สามารถแก้ไขเสียงรบกวนได้ดีขึ้นมาก และเฉพาะ Layer-3 เท่านั้นที่ใช้การเข้ารหัส Entropy (เหมือนกับวีดีโอ MPEG) เพื่อลดการซ้ำซ้อน และก็เฉพาะ Layer-3 ที่ใช้ Bit Reservoir (เหมือนวีดีโอ MPEG) เพื่อกำ จัด Artefacts ในช่วงเวลาวิกฤต และ Layer-3 ยังสามารถใช้กรรมวิธีเข้ารหัส Joint-stereo ชั้นสูงได้อีกด้วย

ตารางแสดงคุณภาพเสียงของ MPEG Layer-3

คุณภาพเสียง	แบนด์วิธ	โหมด	บิตเรต	อัตราการลดทอน
เสียงโทรศัพท์	2.5 kHz	Mono	8 kbps	96:1
ดีกว่าคลื่นสั้น	4.5 kHz	Mono	16 kbps	48:1
ดีกว่าวิทยุ AM	7.5 kHz	Mono	32 kbps	24:1
เทียบเท่าวิทยุ FM	11 kHz	Stereo	56-64 kbps	24-26:1
ใกล้เคียง CD	15 kHz	Stereo	96 kbps	16:1
CD	>15 kHz	Stereo	112-128 kbps	12-14:1

ตารางที่ 2.1 ตารางเปรียบเทียบคุณภาพเสียงของ MPEG Layer-3

2.3 VS1002d - MPEG AUDIO CODEC

2.3.1 คุณสมบัติของ VS1002d

1.สามารถถอดรหัสข้อมูลเอ็มเป็กออดิโอได้ทั้งเลเยอร์ 1,2 และ3

2.สนับสนุนการถอดรหัสข้อมูลเอ็มเป็ก 1 และ 2 ทุกเลเยอร์ในเอ็มเป็ก 3สามารถถอดรหัสได้ถึงเลเยอร์ 2.5 โดยสนับสนุนข้อมูลทั้งแบบโมโนและสเตริโอ

3.สามารถส่งข้อมูลด้วยความเร็วได้หลายระดับ

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

5.ประหยัดพลังงาน

6.ในชิพประกอบด้วย ดิจิตอลอะนาลอกคอนเวสเตอร์(DAC)คุณภาพสูงโดยปราศจากเพสเอเรอระหว่างแชลแนล

7.สำหรับสัญญาณอนาลอกทำงานด้วยระดับแรงดัน 2.6 - 3.6 V

8.สำหรับสัญญารดิจิตอลทำงานด้วยระดับแรงดัน 2.1 – 3.6 V

9.มีแรมบนชิพถึง 4 กิโลบิต สำหรับผู้ใช้

10.มีฟังก์ชั่นใหม่เพิ่มเติม เช่น ข้อมูลเข้าเป็น PCM ข้อมูลเข้าเป็นสตรีม(Streaming)

11.ประมวลผลใน16บิตเวิรดส์ของข้อมูล

2.3.2 ข้อมูลของขาใน VS1002d

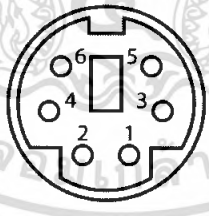
ชื่อขา	หน้าที่ของขา
DREQ	ขาแสดงร้องขอข้อมูล
DCIK	ขาสัญญาณนาฬิกาสำหรับข้อมูลอินพุตแบบอนุกรม
SDATA	ขาข้อมูลอินพุตแบบอนุกรม
BSYNC	ขาสัญญาณแสดงข้อมูลซิงโครไนส์
DVDD1	ขาไฟเลี้ยงดิจิตอล
DGND1	ขากราวด์ดิจิตอล
XTALO	ขาคริสตอลเอาท์พุต
XTALI	ขาคริสตอลอินพุต
DVDD2	ขาไฟเลี้ยงดิจิตอล
DGND2	ขากราวด์ดิจิตอล
XCS	ขาเลือกอินพุทข้อมูลว่าเป็นข้อมูลหรือคอนโทรล
SCLK	ขาสัญญาณนาฬิกาสำหรับข้อมูลอินพุตแบบอนุกรม
SI	ขาอินพุตอนุกรม
SO	ขาเอาท์พุตอนุกรม
TEST0	ขาสำรองไว้สำหรับทดสอบ ต่อเข้ากับ DVDD
TEST1	ขาสำรองไว้สำหรับทดสอบ ไม่ต้องต่อ
TEST2	ขาสำรองไว้สำหรับทดสอบ ไม่ต้องต่อ
AGND1	ขากราวด์อนาลอก
AVDD1	ขาไฟเลี้ยงอนาลอก

ชื่อขา	หน้าที่ของขา
RIGHT	ขาเอาท์พุตแชนแนลขวา
AGND2	ขากราวด์อนาลอก
RCAP	ขาเปรียบเทียบความจุไฟฟ้า
AVDD2	ขาไฟเลี้ยงอนาลอก
LEFT	ขาเอาท์พุตแชนแนลซ้าย
AGND3	ขากราวด์อนาลอก
XRESET	ขาเอาท์พุตแบบอะซิงโครไนส์ ทำงานที่ระดับสัญญาณ “0”
DGND3	ขากราวด์ดิจิตอล
DVDD3	ขาไฟเลี้ยงดิจิตอล

ตารางที่ 2.2 หน้าที่ของขาในชิพ VS1002d

2.4 พอร์ต PS/2 และ คีย์บอร์ด

พอร์ต PS/2 ถูกพัฒนาขึ้นโดยบริษัท IBM เป็นที่รู้จักกันในฐานะของช่องเชื่อมต่อสำหรับเมาส์และคีย์บอร์ดเข้ากับคอมพิวเตอร์ พอร์ตชนิดนี้จะทำงานร่วมกับปลั๊ก DIN ขนาดเล็กที่ภายในมีเข็มสัญญาณ 6 เข็ม เรียงตัวกันเป็นวง ดังรูป



รูปที่ 2.4 พอร์ต PS/2 ตัวเมีย

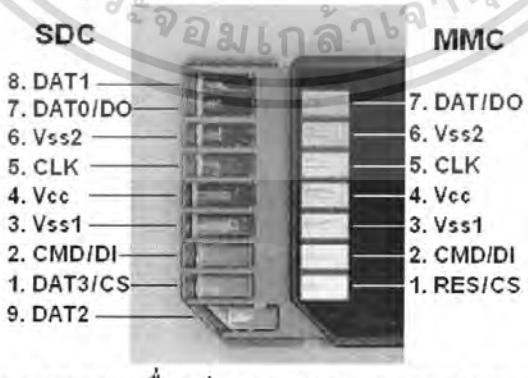
คียบอร์ด หรือ แป้นพิมพ์ เป็นอุปกรณ์คอมพิวเตอร์ที่ทุกเครื่องจำเป็นต้องมี โดยปกติมักจะมีลักษณะเป็นสี่เหลี่ยมผืนผ้าหรือใกล้เคียง มีแป้นต่างๆ ประมาณร้อยแป้นอยู่บนคีย์บอร์ต (ขึ้นอยู่กับผังแป้นพิมพ์) ซึ่งถอดแบบมาจากเครื่องพิมพ์ดีด ออกแบบมาเพื่อใช้สำหรับรับข้อมูลที่เป็นตัวอักขระ แล้วทำการเปลี่ยนเป็นรหัส จากนั้นจึงส่งให้คอมพิวเตอร์ประมวลผล หรือใช้ควบคุมฟังก์ชันการทำงานบางอย่างของคอมพิวเตอร์ และเพื่อให้การป้อนข้อมูลที่เป็นอักขระและตัวเลขทำได้ง่ายและสะดวกขึ้น คีย์บอร์ดจึงแยกแผงที่เป็นแป้นอักขระกับแป้นตัวเลขแยกไว้ต่างหาก ดังรูป

รูปที่ 2.6 รูปแสดงแผงอักขระของคีย์บอร์ด

2.5 เอสดีการ์ด(SD Card)

2.5.1คุณลักษณะรูปร่างและโครงสร้าง

ลักษณะภายนอกของเอสดีการ์ด จะมีขนาดที่มีการกำหนดไว้เป็นมาตรฐาน คือ มีความกว้าง 24 มิลลิเมตร,ความยาว 32 มิลลิเมตร และความหนา 2.1 มิลลิเมตร มีน้ำหนักโดยประมาณ 2 กรัม และจะมีมุมตัดที่มุมบนขวามือ ขาที่ใช้ในการติดต่อกับอุปกรณ์ภายนอกของ SD Card มีทั้งหมด 9 ขาซึ่งมีมากกว่า MMC ที่มี 7 ขา แสดงการเปรียบเทียบของขาสัมผัสระหว่าง SD Card กับ MMC ได้ดังรูปที่ 2.7 SD Card นั้นที่ตำแหน่งด้านข้างทางซ้ายจะมีสวิตช์เลื่อน หากเลื่อนมายังตำแหน่ง Lock จะเป็นการไม่อนุญาตให้ทำการเขียนหรือลบข้อมูลบนตัวการ์ดได้ (Write Protect) แต่ถ้าหากเลื่อนสวิตช์มายังตำแหน่ง Un-Lock จะอนุญาตให้ทำการเขียนหรือลบข้อมูลบนตัวการ์ดได้ แต่ทั้งสองกรณีนี้ยังสามารถที่จะอ่านข้อมูลได้ตลอดเวลา



รูปที่ 2.5 แสดงขาเชื่อมต่อของ SD Card และ MMC Card

2.5.2การทำงานในโหมด SD และโหมด SPI

การติดต่อสื่อสารของ SD Card นั้นมีวิธีการติดต่ออยู่ 2 โหมด คือ SD และโหมด SPI ซึ่งหากดูรายละเอียดจากตารางจะทราบถึงข้อเปรียบเทียบการทำงานของ SD Card คือ จะสามารถกำหนดโหมดการทำงานได้ 2 โหมด ซึ่งแต่ละโหมดจะมีการอ้างอิงถึงการใช้คำสั่ง กล่าวคือหากต้องการติดต่อกับการ์ดในโหมด SPI ที่ขาสัญญาณ CS (CS หรือ Chip Select ในบางแหล่งจะใช้ชื่อเป็น SS หรือ Slave Select ซึ่งเป็นอันที่เข้าใจว่าเป็นขาสัญญาณเดียวกัน) จะต้องกำหนดให้มีสถานะลอจิกต่ำ (Logic Low)ไว้ตลอดเวลาและขาสัญญาณที่ต้องกำหนดให้ใช้งานด้วย คือขา SCLK(Serial Clock),MISO(Master-In-Slave-Out) และ MOSI(Master-Out-Slave-In)การส่งข้อมูลนั้นจะมีการรับและส่งข้อมูลระหว่างตัวอุปกรณ์(Host)กับการ์ดทีละ 1 บิต ข้อมูลในแต่ละบิตนั้นจะต้องมีการอ้างอิงสัญญาณนาฬิกาที่ขา SCLK ขึ้นมา 1 ลูกต่อข้อมูล 1 บิต

สำหรับการติดต่อกันแบบ SD โหมดนั้น ขาที่เคยทำหน้าที่เป็นขา CS ใน SPI โหมด จะทำหน้าที่เป็นขา Card Detect และเมื่อมีการกำหนดการติดต่อในโหมด SD แล้ว จะมีขาที่ทำหน้าที่เป็นขาข้อมูล(Data)ทั้งหมด 4 ขา เพราะฉะนั้นจะได้ความเร็วในการสื่อสารมากเป็น 4 เท่าเมื่อเปรียบเทียบกับโหมด SPI

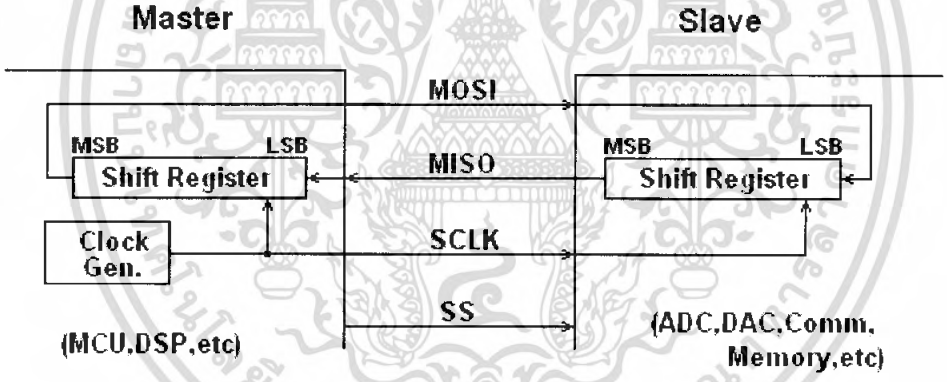
Pin	SD Mode		SPI Mode	
	Name	Description	Name	Description
1	CD/DAT3	Card Detect/Data Line[Bit3]	CS	Chip Select
2	CMD	Command/Response	DI	Data In
3	VSS	Supply Voltage Ground	VSS	Supply Voltage Ground
4	VDD	Supply Voltage	VDD	Supply Voltage
5	CLK	Clock	SCLK	Clock
6	VSS2	Supply Voltage Ground	VSS2	Supply Voltage Ground
7	DAT0	Data Line[Bit0]	DO	Data Out
8	DATI	Data Line[Bit1]	RSV	-
9	DAT2	Data Line[Bit2]	RSV	-

ตารางที่ 2.3 รายละเอียดขาต่างๆของ SD Card

2.6การเชื่อมต่ออุปกรณ์ต่อพ่วงแบบอนุกรม(Serial Peripheral Interface:SPI)

การเชื่อต่ออุปกรณ์ต่อพ่วงแบบอนุกรมเป็นวิธีหนึ่งที่ใช้ในการเชื่อมต่อติดต่อสื่อสารระหว่างไอซีภายในบอร์ด การเชื่อมต่ออุปกรณ์ต่อพ่วงแบบอนุกรมนี้ถูกเผยแพร่โดย Motorola,inc. (Freeacale Semiconductor) เนื่องจากวิธีการเชื่อมต่อแบบนี้สามารถทำได้ง่าย จึงเป็นอีกทางเลือกหนึ่งในการเชื่อมต่อกันระหว่างไอซีนอกจากการเชื่อมต่อแบบ IIC การเชื่อมต่ออุปกรณ์ต่อพ่วงแบบอนุกรมนี้จะใช้สายสัญญาณในการสื่อสารถึง 4 เส้น ซึ่งจะมากกว่าแบบ IIC ซึ่งใช้เพียง 2 เส้น แต่การเชื่อมต่อแบบ SPI จะสามารถส่งผ่านข้อมูลได้มากกว่า ซึ่งสามารถส่งข้อมูลได้สูง 20 ล้านบิตต่อวินาที(ทั้งนี้ขึ้นอยู่กับข้อจำกัดของอุปกรณ์)ปัจจุบันได้มีการนำวิธีการเชื่อมต่ออุปกรณ์แบบ SPI ไปใช้ในไอซีแปลงอนาลอกเป็นดิจิตอล, ไอซีแปลดิจิตอลเป็นอนาลอก หรือไอซีด้านการสื่อสารต่างๆ ที่ต้องการความเร็วในการส่งข้อมูลสูงๆ

2.6.1โครงสร้างของSPI



รูปที่ 2.6 แสดงโครงสร้างของ SPI

จากรูปด้านบนแสดงโครงสร้างของการเชื่อมต่ออุปกรณ์ภายนอกแบบอนุกรมซึ่งไอซีที่เป็นมาสเตอร์ถูกเชื่อมโยงกับไอซีที่เป็นสเลฟ ด้วยสายสัญญาณ 3 สาย คือ

1.SCLK(Serial Clock)

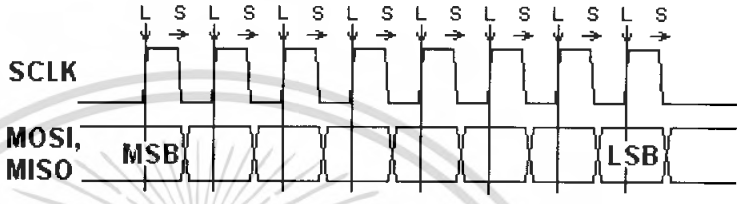
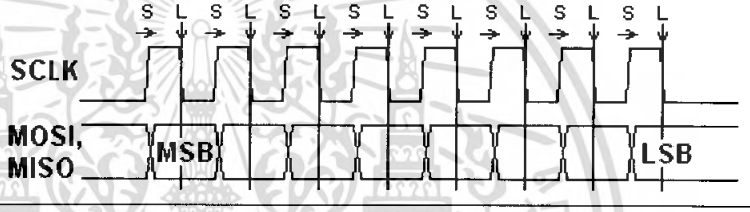
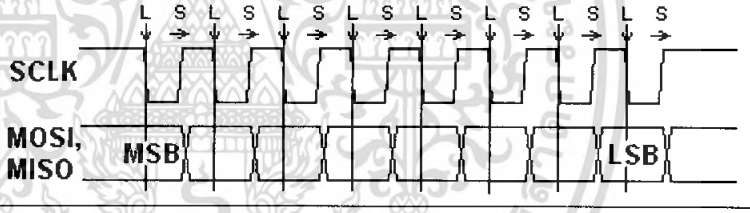
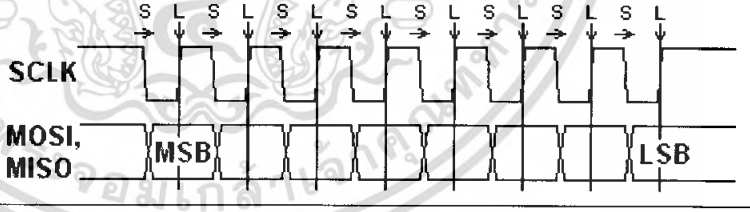
2.MISO(Master-In-Slave-Out)

3.MOSI(Master-Out-Slave-In)

และภายนในไอซีทั้ง 2 ประกอบไปด้วยชิฟท์รีจิสเตอร์ 8 บิต ซึ่งใช้ในการแลกเปลี่ยนข้อมูลโดยการเลื่อนข้อมูลเข้าไปยังชิฟท์รีจิสเตอร์ตามจังหวะของสัญญาณนาฬิกาซึ่งสร้างจากไอซีที่เป็นมาสเตอร์ นอกจากนี้ ยังมีอีกหนึ่งขาสัญญาณนอกจากขาทั้ง 3 ข้างต้น คือ ขา SS (Slave Select) ซึ่งใช้ในการบอกไอซีว่าให้เริ่มรับชุดข้อมูล ซึ่งจะใช้ในการกำหนดว่าจะให้ไอซีที่เป็นสเลฟตัวไหนรับข้อมูลใน

กรณีที่มีการต่อไอซีสเลฟหลายตัว การกำหนดชื่อของขาสัญญาณในไอซีต่างๆกัน อาจจะกำหนดชื่อต่างกันออกไป เช่น DI,DO,CS เป็นต้น

ในการชิพข้อมูลและการแลทช์ข้อมูลของการสื่อสารแบบ SPI จะทำงานที่ขอบของสัญญาณนาฬิกาตรงข้ามกัน นอกจากนี้ยังแบ่งการเชื่อมต่อกับอุปกรณ์ภายนอกแบบอนุกรมออกได้เป็น 4 แบบ เนื่องจากขั้ว และเฟสของสัญญาณนาฬิกา ดังแสดงในตาราง

SPI Mode	Timing Diagram
Mode 0 Positive Pulse. Latch, then Shift.	
Mode 1 Positive Pulse. Shift, then Latch.	
Mode 2 Negative Pulse. Latch, then Shift.	
Mode 3 Negative Pulse. Shift, then Latch.	

ตารางที่ 2.4 แสดงแผนผังเวลาการถ่ายโอนข้อมูลของการสื่อสารแบบ SPI

2.7 การเก็บข้อมูลแบบ FAT

FAT ย่อมาจากคำว่า File Allocation Table ซึ่งคือตารางที่ใช้เก็บตำแหน่งของข้อมูลต่างๆ ที่อยู่บนฮาร์ดดิสก์ ว่าอยู่ตรงตำแหน่งไหนของฮาร์ดดิสก์

ปกติเมื่อผู้ใช้ซื้อฮาร์ดดิสก์มาใหม่ 1 ตัว ผู้ใช้ต้องทำการฟอร์แมต (Format) ฮาร์ดดิสก์ก่อนที่จะนำไปบรรจุข้อมูล การฟอร์แมตฮาร์ดดิสก์เป็นการแบ่งฮาร์ดดิสก์ออกเป็นส่วนๆ เพื่อให้คอมพิวเตอร์รู้ว่าตำแหน่งของข้อมูลอยู่ตรงไหนของฮาร์ดดิสก์ การแบ่งนี้จะแบ่งออกเป็น เซ็กเตอร์

(Sector) และแทร็ก (Track) โดยเซ็กเตอร์นั้นจะเป็นการแบ่งตามแนวตั้งฉากกับศูนย์กลางของแผ่นดิสก์ และแทร็กจะเป็นการแบ่งเป็นเส้นขวางรอบศูนย์กลางของแผ่นดิสก์ FAT เป็นระบบไฟล์ชนิดหนึ่งที่ถูกกำหนดโดยซอฟต์แวร์ระบบปฏิบัติการ (operating system) ซึ่งระบบไฟล์ที่นิยมใช้กันอยู่ในปัจจุบันมี

-FAT ของระบบปฏิบัติการดอส (DOS) และวินโดวส์ (Windows)

-NTFS ของระบบปฏิบัติการวินโดวส์เอ็นที (Windows NT)

-HPFS ของระบบปฏิบัติการโอเอสทู (OS2)

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 ดังนั้นถ้าเป็น 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 คลัสเตอร์ เท่ากับ 2,097,152 KB หรือ 2,048 MB หรือ 2 GB)

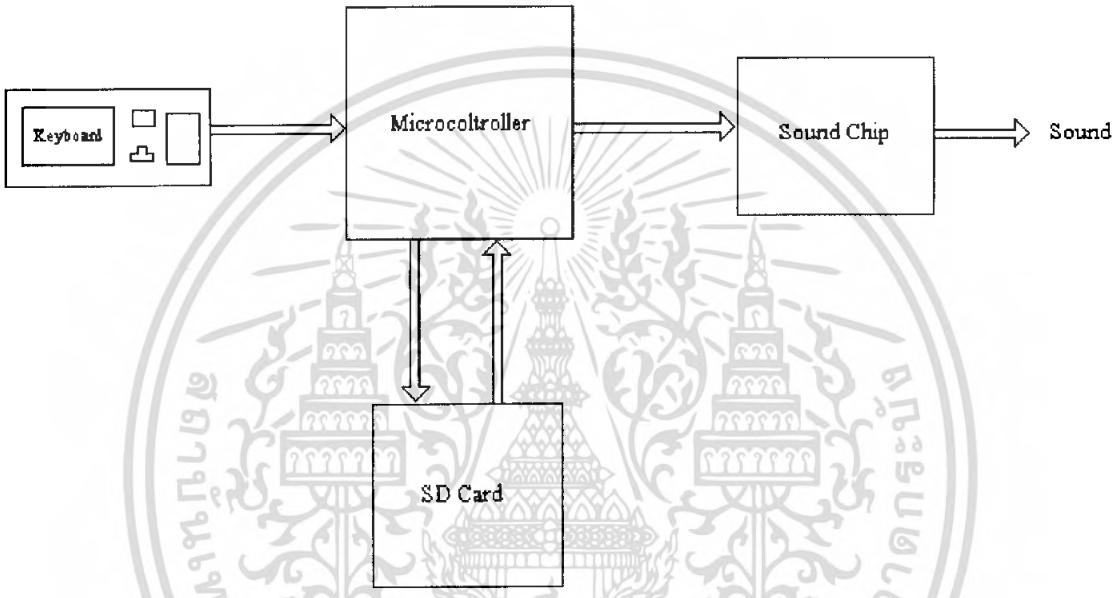
ดังนั้นถ้าผู้ใช้ต้องการใช้ฮาร์ดดิสก์ขนาด 2 GB หรือฮาร์ดดิสก์ที่มีพาร์ทิชันเท่ากับ 2GB หมายความว่าขนาดของคลัสเตอร์ที่เล็กที่สุดเท่ากับ 32 KB ซึ่งหมายความว่าไม่ว่าไฟล์ที่ต้องการเก็บในฮาร์ดิสก์จะมีขนาดเล็กแค่ไหนก็ตาม ฮาร์ดิสก์ก็จะต้องจองพื้นที่ให้ไฟล์นี้ไม่ต่ำกว่า 32 KB ตัวอย่างเช่น ถ้าต้องการเก็บไฟล์ขนาด 1 KB ลงในฮาร์ดดิสก์ที่เป็น FAT 16 ฮาร์ดดิสก์จะต้องจองพื้นที่เพื่อเก็บไฟล์นี้ 32 KB ซึ่งหมายความว่าผู้ใช้จะต้องสูญเสียพื้นที่ในฮาร์ดดิสก์ไปโดยไม่สามารถใช้ได้ถึง 31 KB ดังนั้นยิ่งมีไฟล์ขนาดเล็กกว่า 32 KB มากเท่าไร หมายความว่า จะสูญเสียเนื้อที่ว่างบนฮาร์ดดิสก์ไปโดยไม่ได้ใช้ประโยชน์มากขึ้นเท่านั้น วิธีแก้ไขปัญหาการสูญเสียเนื้อที่นี้อาจจะทำได้ 2 กรณีคือ

1. ถ้าผู้ใช้ยังต้องการใช้ฮาร์ดดิสก์ที่เป็น FAT16 อยู่ ผู้ใช้ต้องแบ่งพาร์ทิชันฮาร์ดดิสก์ให้มีขนาดเล็กลง เพื่อให้ฮาร์ดดิสก์มีขนาดคลัสเตอร์เล็กลง จะได้ทำให้เนื้อที่ว่างที่ไม่สามารถใช้งานเหลือน้อยลง ถ้าผู้ใช้แบ่งพาร์ทิชันฮาร์ดดิสก์ไว้ที่ขนาด 512 MB ต่อพาร์ทิชัน ขนาดของคลัสเตอร์จะลดลงเหลือแค่ 8 KB ซึ่งจะทำให้ความสูญเสียเนื้อที่บนฮาร์ดดิสก์โดยเปล่าประโยชน์ลดน้อยลงถึง 3 เท่าเมื่อเทียบกับการแบ่งพาร์ทิชันไว้ที่ขนาด 2 GB แต่วิธีการแบ่งฮาร์ดดิสก์ออกเป็นหลายๆ พาร์ทิชันนี้อาจทำให้เกิดความยุ่งยากเช่น มีไดร์ฟฮาร์ดดิสก์หลายๆ ไดร์ฟ อาจทำให้สบสนเวลาใช้งาน

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 นี้จำนวนคลัสเตอร์ที่จะอ้างอิงถึงได้เท่ากับ 2 ยกกำลัง 32 หรือเท่ากับ 268,436,456 คลัสเตอร์ ดังนั้นเมื่อใช้ขนาดของคลัสเตอร์ 4 KB ขนาดของพาร์ทิชันสูงสุดที่จะมีได้จะเท่ากับ 8 GB และถ้าขนาดของคลัสเตอร์สูงสุดที่ 32 KB จะทำให้ฮาร์ดดิสก์สามารถมีพาร์ทิชันได้สูงที่สุดที่ 2 TB (1 Teta Byte เท่ากับ 1,024 GB)

บทที่ 3
การออกแบบวงจร

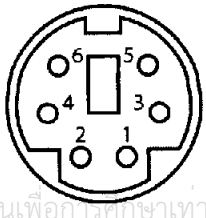
3.1 บลอคไดอะแกรมของวงจร



รูปที่ 3.1 บลอคไดอะแกรมของวงจร

จากรูป เมื่อเรากดปุ่มบนคีย์บอร์ด คีย์บอร์ดก็จะทำการส่งข้อมูลรหัสscan codeผ่าน พอร์ต PS/2 ไปยังไมโครคอนโทรลเลอร์แล้วทำการเปรียบเทียบข้อมูลที่ได้เพื่อหาข้อมูลเสียงที่ตรงกับตัวอักษรที่ได้ทำการกดปุ่มไว้ หลังจากนั้นจะส่งข้อมูลของเสียงไปยังชิพเสียงเพื่อทำการถอดรหัสแล้วส่งสัญญาณเสียงที่ได้ออกมาทางลำโพง

3.2 การเชื่อมต่อคีย์บอร์ดกับไมโครคอนโทรลเลอร์ผ่าน PS/2



รูปที่ 3.2 พอร์ต PS/2 ตัวเมีย

Pin 1	+DATA	Data
Pin 2	Not connected	Not connected
Pin 3	GND	Ground
Pin 4	Vcc	+5 V DC at 100 mA
Pin 5	+CLK	Clock
Pin 6	Not connected	Not connected

ตารางที่ 3.1 ตารางแสดงหน้าที่ขาต่างๆของพอร์ต PS/2

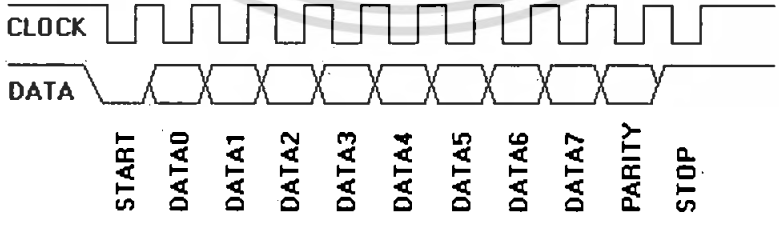
เมื่อเรากดปุ่มบนคีย์บอร์ดจะมีการส่งข้อมูลรหัสScan codeออกมาเป็นข้อมูลอนุกรม 11 บิต

1 start bit มีค่าเป็น 0

8 data bits

1 parity bit (odd parity)

1 stop bit มีค่าเป็น 1



รูปที่ 3.3 สัญญาณข้อมูลของคีย์บอร์ด

3.3 การเชื่อมต่อไมโครคอนโทรลเลอร์กับ SD Card

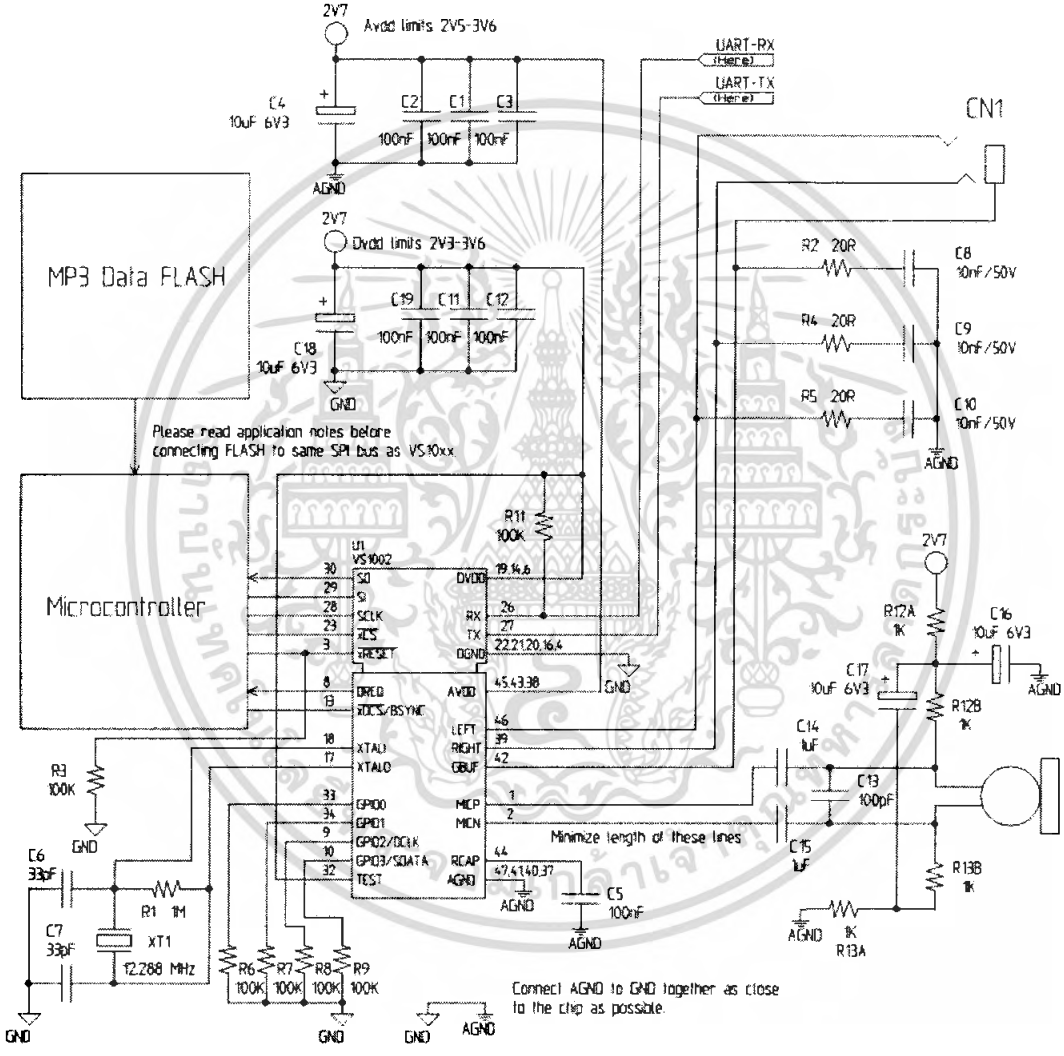
เมื่อไมโครคอนโทรลเลอร์ได้รับข้อมูลตัวอักษรแล้ว ก็จะทำการเรียกไฟล์เสียงที่ตรงกันจาก SD Card ซึ่งในการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับ SD Card จะใช้การเชื่อมต่อแบบ SPI มีขาสัญญาณอนุกรม 4 เส้น ดังรูป

Pins	SPI Mode		
	Name	IO Type	Description
1	CS	I	Chip Select (Negative True) (CS0)
2	DI	I	Data In (MOSI)
3	V_{SS}	S	Ground
4	V_{dd}	S	Supply Voltage
5	SCLK	I	Clock (SCK)
6	V_{SS2}	S	Ground
7	DO	O/PP	Data Out (MISO)
8	RSV	-	Reserved (*)
9	RSV	-	Reserved (*)

ตารางที่ 3.2 แสดงหน้าที่ขาต่างๆของ SD Card

3.4 การเชื่อมต่อไมโครคอนโทรลเลอร์กับ VS1002d

เมื่อได้รับข้อมูลไฟล์เสียงมาแล้วไมโครคอนโทรลเลอร์ก็จะทำการส่งไฟล์เสียงไปยัง VS1002d เพื่อทำการแปลงไฟล์เสียงให้ออกมาเป็นเสียงตัวอักษร ซึ่งในการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์ กับ VS1002d จะใช้การเชื่อมต่อแบบ SPI มีขาสัญญาณอนุกรม 4 เส้น



รูปที่ 3.4 วงจรและการใช้งาน VS1002d

การติดต่อกับ Sound chip นั้นแบ่งเป็น 2 กรณีคือ การส่งคำสั่ง(SCI) และการส่งข้อมูล(SDI)

1.SCI(Serial Protocol for Serial Command Interface)

1.1 ทำให้ขาสัญญาณ xCS เป็น 0 เพื่อเข้าสู่โหมด SCI

1.2 ถ้าต้องการอ่านข้อมูล ให้ส่ง Opcode 03H ตามด้วยแอดเดรส 8 บิต หลังจากนั้นVS1002d จะส่งข้อมูล 16 บิต กลับมา

1.3 ถ้าต้องการเขียนข้อมูล ให้ส่ง Opcode 02H ตามด้วยแอดเดรส 8 บิต ตามด้วยข้อมูลที่เขียน 16 บิต

1.4 ทำให้ขาสัญญาณ xCS เป็น 1 เพื่อจบโหมด SCI

2. SDI(Serial Protocol for Serial Data Interface)

2.1 ตรวจสอบขา DREQ ถ้าเป็น 1 ก็จะเริ่มการส่งข้อมูล

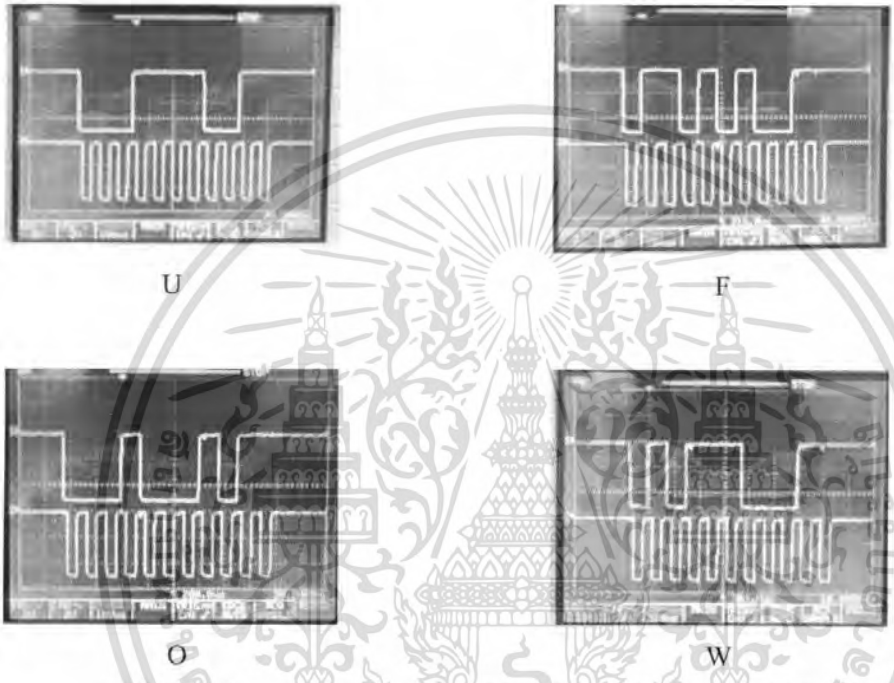
2.2 ในการส่งข้อมูลแต่ละไบต์ ต้องให้ขา BSYNC เป็น 1 ก่อนที่จะส่งข้อมูลบิตแรก และให้กลับเป็น 0 ก่อนที่จะส่งข้อมูลบิตสุดท้ายออกไป

2.3 ส่งข้อมูลออกไปจนจบ

บทที่ 4

ผลการทดลอง

4.1การทดลองจับสัญญาณคีย์บอร์ดจากออสซิลโลสโคป



รูปที่ 4.1 แสดงสัญญาณข้อมูลของคีย์บอร์ดที่จับได้จากออสซิลโลสโคป

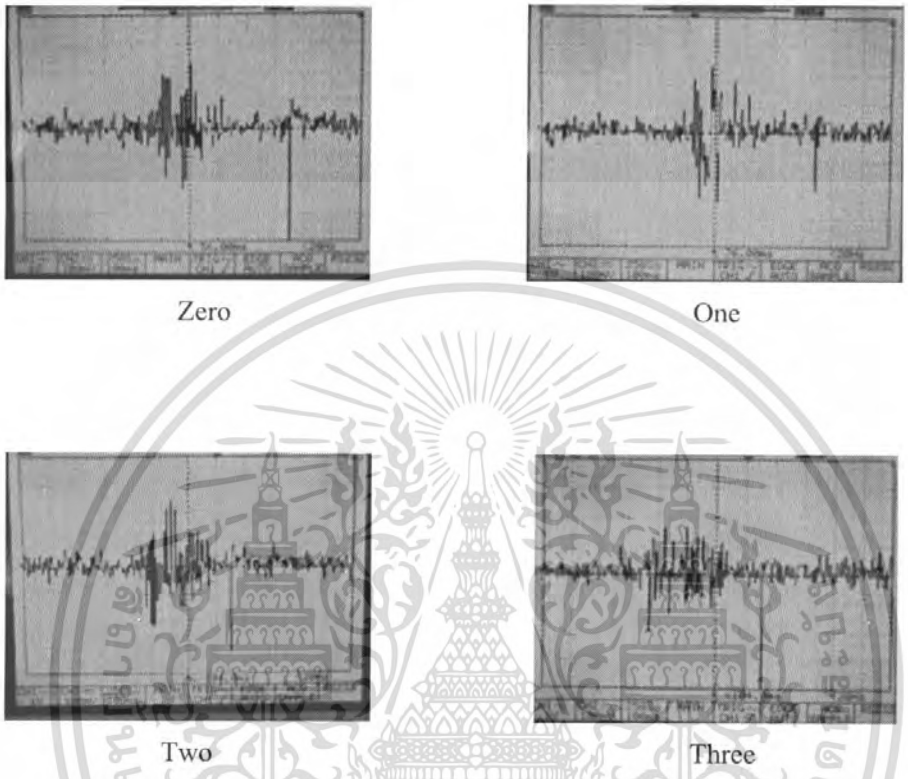
4.2การทดลองจับสัญญาณคีย์บอร์ดให้มาแสดงผลยังLCD

ในการจับสัญญาณข้อมูลจากคีย์บอร์ดจะได้ข้อมูลทั้งหมด 11 บิต เมื่อตัดStart Bit(บิต1),Parity Bit(บิต10) และStop Bit(บิต11) จะได้ข้อมูล 8 บิต เนื่องจากการส่งจะส่งจาก LSB ก่อนจึงต้องนำไปกลับบิตข้อมูลก่อน แล้วแปลงเป็นเลขฐาน 16 ก็จะได้ค่า Scan Code

ปุ่มบนคีย์บอร์ด	สัญญาณที่จับได้(11บิต)	ข้อมูลScan Code(8 บิต)	ข้อมูลScan Code
R	01011010011	0010 1101	2D
V	00101010001	0010 1010	2A
K	00100001011	0100 0010	42

ตารางที่ 4.1 แสดงสัญญาณข้อมูลจากคีย์บอร์ดที่แสดงผลที่LCD

4.3การทดลองจับสัญญาณเสียง



รูปที่ 4.2 แสดงสัญญาณเสียงที่จับได้

บทที่ 5
สรุปและวิจารณ์ผลการทดลอง

5.1สรุป

ในรายงานได้กล่าวถึงความเป็นมาของโครงการ แนวคิด ทฤษฎี และรายละเอียดการสร้างคีย์บอร์ดเปล่งเสียง รวมถึงการทดสอบการทำงานเบื้องต้น โดยในโครงงานนี้เป็นการใช้ไมโครคอนโทรลเลอร์ ในการเชื่อมต่อระหว่าง คีย์บอร์ด,SD Card และ ไอซีถอดรหัสMP3 ซึ่งเป็นการเชื่อมต่ออุปกรณ์ภายนอกแบบอนุกรณ์(SPI) นอกจากนี้สามารถเข้าถึงไฟล์ซึ่งถูกจัดเก็บใน SD Card ด้วยระบบไฟล์แบบ FAT สามารถที่จะอ่านข้อมูลที่อยู่ในการ์ดที่ตำแหน่งแอดเดรสต่างๆเพื่อส่งไปให้กับไอซีถอดรหัสMP3เพื่อแปลงเป็นเสียงออกหูฟังต่อไป โดยให้เสียงที่ออกมามีความสัมพันธ์กับที่ผู้ใช้ได้กดปุ่มบนคีย์บอร์ด

5.2ปัญหาและแนวทางแก้ไข

1.เนื่องจากไม่มีอุปกรณ์ที่ใช้สำหรับการดีบั๊กโปรแกรม ซึ่งโปรแกรมมีความซับซ้อนทำให้ยากต่อการตรวจสอบการทำงานของโปรแกรม แล้วจึงทำให้การแก้ไขทำได้ยากเนื่องจากไม่ทราบว่าตัวโปรแกรมผิดพลาดที่จุดใด

2.ไมโครคอนโทรลเลอร์เบอร์dsPIC30F4011ต้องใช้แรงดันไฟเลี้ยง 5 โวลท์ ในการทำงานทำให้ลอจิกสูงที่ได้จากไมโครคอนโทรลเลอร์คือ 5 โวลท์แต่ SD Card และ ไอซีถอดรหัสMP3 นั้นใช้งานแรงดันลอจิกสูงที่ 3.3 โวลท์ ดังนั้นจึงใช้วงจรเปลี่ยนแรงดันทำให้สามรถเชื่อมต่อกับ SD Card และ ไอซีถอดรหัสMP3 ได้

3.ข้อมูลไฟล์เสียงที่ออกมาไม่เหมือนกับที่บันทึกไว้ แต่ก็มีลักษณะใกล้เคียงกับที่บันทึกไว้ ซึ่งอาจเกิดจากความผิดพลาดในการเขียนโปรแกรม

5.3ประโยชน์ที่ได้รับ

ในการทำโครงงานนี้ได้รับความรู้ใหม่ๆ เช่นการเขียนโปรแกรมไมโครคอนโทรลเลอร์dsPICเพื่อเชื่อมต่อสื่อสารกับคีย์บอร์ด, SD Card และการเขียนโปรแกรมเพื่อเชื่อมต่อกับไอซีถอดรหัสMP3 ซึ่งในปัจจุบัน SD Card มีความนิยมมากในการใช้เป็นอุปกรณ์เก็บข้อมูลอีกทั้งเครื่องเล่นMP3 ก็มีใช้กันอย่างแพร่หลาย ดังนั้นในโครงงานนี้จึงทำให้มีความรู้ความเข้าใจในการอ่านข้อมูลใน SD Card และการเชื่อมต่อกับไอซีถอดรหัสMP3 อีกทั้งในการทำโครงงานนี้ยังช่วยเพิ่มประสบการณ์

ในการทำงาน ทำให้ได้ความรู้จากการศึกษาวงจร พัฒนาทักษะทางด้านการเขียนโปรแกรมภาษาC และตัวอุปกรณ์ต่างๆ ซึ่งจะเป็นประโยชน์ในอนาคตต่อไป

นอกจากนี้สิ่งที่สำคัญที่สุดที่ได้รับจากการทำโครงงานนี้คือการได้ทำอะไรเพื่อช่วยเหลือคนที่ด้อยโอกาสในสังคมอย่างคนพิการทางสายตา

บรรณานุกรม

1.นคร ภักดีชาติ,ชัยวัฒน์ ลิ้มพรจิตรวิไล , “คู่มือการทดลอง dsPIC Microcontroller เบื้องต้น ด้วยโปรแกรมภาษา C กับ MPLAB C30” , 376 หน้า

ภาคผนวก

ตาราง scan code

ปุ่มคีย์บอร์ด	Scan Code	ปุ่มคีย์บอร์ด	Scan Code
Esc	76	Back Space	66
F1	05	TAB	0D
F2	06	Q	15
F3	04	W	1D
F4	0C	E	24
F5	03	R	2D
F6	0B	T	2C
F7	83	Y	35
F8	0A	U	3C
F9	01	I	43
F10	09	O	44
F11	78	P	4D
F12	07	[{	54
~ `	0E	] }	5B
1 !	16	Caps	58
2 @	1E	A	1C
3 #	26	S	1B
4 $	25	D	23
5 %	2E	F	2B
6 ^	36	G	34
7 &	3D	H	33
8 *	3E	J	3B
9 (	46	K	42
0)	45	L	4B
- _	4E	: ;	4C
+ =	55	" '	52
| \	5D	Enter	5A

ปุ่มคีย์บอร์ด	Scan Code	ปุ่มคีย์บอร์ด	Scan Code
Left Shift	12	Page Down	E0 7A
Z	1A	Up	E0 75
X	22	Down	E0 72
C	21	Left	E0 6B
V	2A	Right	E0 74
B	32	Num	77
N	31	/	E0 4A
M	3A	*	7C
< ,	41	-	7B
> .	49	+	79
? /	4A	.	71
Right Shift	59	0	70
Left Ctrl	14	1	69
Left Alt	11	2	72
SPACE	29	3	7A
Right Alt	E0 11	4	6B
Right Ctrl	E0 14	5	73
Ins	E0 70	6	74
Del	E0 71	7	6C
Home	E0 6C	8	75
End	E0 69	9	7D
Page Up	E0 7D	Num Enter	E0 5A

Source code

```
////////////////////////////////////
////Interface VS1002d ////
////////////////////////////////////
#include<p30f4011.h>
#include<spi.h>
_FOSC(CSW_FSCM_ON & XT_PLL4)
_FWDT(WDT_OFF)

#define  SCI_MODE                0x00
#define  SCI_STATUS              0x01
#define  SCI_BASS                0x02
#define  SCI_CLOCKF              0x03
#define  SCI_DECODE_TIME         0x04
#define  SCI_AUDATA              0x05
#define  SCI_WRAM                0x06
#define  SCI_WRAMADDR            0x07
#define  SCI_HDAT0               0x08
#define  SCI_HDAT1               0x09
#define  SCI_AIADDR              0x0A
#define  SCI_VOL                 0x0B
#define  SCI_AICTRL0             0x0C
#define  SCI_AICTRL1             0x0D
#define  SCI_AICTRL2             0x0E
#define  SCI_AICTRL3             0x0F

#define  VS1002D_XCS_HIGH()      LATBbits.LATB1  = I     // XCS#  = '1'
#define  VS1002D_XCS_LOW()       LATBbits.LATB1  = 0     // XCS#  = '0'
#define  VS1002D_RES_HIGH()      LATBbits.LATB2  = 1     // RES#  = '1'
#define  VS1002D_RES_LOW()       LATBbits.LATB2  = 0     // RES#  = '0'
#define  VS1002D_XDCS_HIGH()     LATBbits.LATB3  = 1     // XDCS  = '1'
#define  VS1002D_XDCS_LOW()      LATBbits.LATB3  = 0     // XDCS  = '0'
```

```
const unsigned int sound[sound_size] = {……};

/*****************************************/
/* Long Delay Time Function(1..4294967295) */
/*****************************************/
void delay(unsigned int ms)
{
 unsigned int x,a;
 for(x=0;x<ms;x++)
  {
 for(a=0;a<443;a++);
  }
}

/***************************/
/* Wait VS1002 DREQ Ready */
/* Wait DREQ = 1(FIFO OK) */
/***************************/
char VS1002D_Wait_DREQ_Ready(void)

{
 unsigned int DREQ_Status;

 DREQ_Status = PORTBbits.RB0;


 if(DREQ_Status == 0)
 {
  return 0;

 }
 else
```

```
  {
    return 1;

  }
}


/************************/
/* Write 1 Byte to SPI0 */
/************************/
void SPI_WriteByte(unsigned int DataByte)
{
  unsigned int Dummy;

  SPI1BUF = DataByte;

  while(SPI1STATbits.SPITBF);
  Dummy = DataByte;

}

/************************/
/* VS1002 Write Command */
/************************/
void VS1002D_Write_SCI(unsigned int SCI_Reg,unsigned int SCI_MSBData,unsigned int
SCI_LSBData)
{
  VS1002D_XCS_LOW();

  SPI_WriteByte(0x02);
  SPI_WriteByte(SCI_Reg);
  SPI_WriteByte(SCI_MSBData);
```

```
  SPI_WriteByte(SCI_LSBData);

  VS1002D_XCS_HIGH();
}

/************************/
/* VS1002D Setup Volume */
/* Left,Right = 0...255  */
/************************/
void VS1002D_Setup_Volume(unsigned int Left,unsigned int Right)
{
  while(VS1002D_Wait_DREQ_Ready());
  VS1002D_Write_SCI(SCI_VOL,Left,Right);
}

/***********************/
/* VS1002D Write Data */
/***********************/
void VS1002D_Write_SDI(unsigned int SDI_Data)
{
  SPI_WriteByte(SDI_Data);

}

/************************/
/* Send Zero to VS1002D */
/************************/

void VS1002D_Write_Zero(unsigned int count)
{
 unsigned int a;
```

```
  VS1002D_XDCS_LOW();
  for(a=0;a<count;a++)
     {
      VS1002D_Write_SDI(0x00);
     }
  VS1002D_XDCS_HIGH();
}

/***************************/
/* VS1002D Hardware Reset */
/***************************/
void VS1002D_HW_Reset(void)

{
  VS1002D_RES_LOW();

  delay(100);
  VS1002D_RES_HIGH();

  delay(100);
}

/***************************/
/* VS1002D Software Reset */
/***************************/

void VS1002D_SW_Reset(void)

{
  while(VS1002D_Wait_DREQ_Ready());
  VS1002D_Write_SCI(SCI_MODE,0x08,0x04);
```

```
 /* Wait 100mS After Reset Complete */
 delay(100);


 while(VS1002D_Wait_DREQ_Ready());
 VS1002D_Write_SCI(SCI_CLOCKF,0x98,0x00);

 while(VS1002D_Wait_DREQ_Ready());
 VS1002D_Write_SCI(SCI_AUDATA,0xBB,0x80);

 while(VS1002D_Wait_DREQ_Ready());
 VS1002D_Write_Zero(2048);


 while(VS1002D_Wait_DREQ_Ready());
 VS1002D_Setup_Volume(0,0);
}

/*****************************/
/* Initial VS1002D Function */
/*****************************/
void VS1002D_Initial()

{
 SPI1CON = 0x033D;

 SPI1STATbits.SPIEN = 1;
 SPI1STATbits.SPISIDL = 1;
 SPI1STATbits.SPIROV = 0;

 TRISBbits.TRISB0 = 1;
```

```
  TRISBbits.TRISB1 = 0;
  TRISBbits.TRISB2 = 0;
  TRISBbits.TRISB3 = 0;

  /* Initial GPIO Signal Interface VS1002D */
  VS1002D_RES_HIGH();
  VS1002D_XCS_HIGH();
  VS1002D_XDCS_HIGH();

  /* Initial VS1002D Function */
  VS1002D_HW_Reset();

  VS1002D_SW_Reset();

  delay(10);
}


/*****************************/
/* Start Main program Here */
/*****************************/
int main (void)
{
  unsigned int  Sound_Pointer;

  VS1002D_Initial();


  while(1)
  {
   VS1002D_XDCS_LOW();
```

```
        Sound_Pointer = 0;

        while(Sound_Pointer < sound_size)
        {
          while(VS1002D_Wait_DREQ_Ready());

          VS1002D_Write_SDI(sound[Sound_Pointer]);
          Sound_Pointer++;
        }
        delay(1000);
  }
}
```

```
//////////////////////////////////////
//////Interface Keyboard//////
//////////////////////////////////////
```

```
///// lcd.h/////
#define NUMBER_OF_DIGITS 32
#define LINE1 0x80
#define LINE2 0xC0
#define delay_ms(x)  lcd_delay(x)
#define lcd_clear()  lcd_command(1)
#define lcd_origin() lcd_command(2)
#define RS LATFbits.LATF0
#define E LATFbits.LATF1

// Function delay 1 ms //
void lcd_delay(unsigned int ms)
{
 unsigned int x,a;
 for(x=0;x<ms;x++)
 {
  for(a=0;a<443;a++);
 }
}

// Function send command //
void lcd_command(unsigned char com)
{
 unsigned char buff;
 buff = (com & 0xF0)>>4;
 RS = 0;
 E = 1;
 LATD = (LATD & 0xF0)|buff;
 lcd_delay(1);
```

```
  E = 0;
  lcd_delay(1);

  buff = (com & 0x0F);
  RS = 0;
  E = 1;
  LATD = (LATD & 0xF0)|buff;
  lcd_delay(1);
  E = 0;
  lcd_delay(1);
}

// Function send data //
void lcd_text(char text)
{
  unsigned char buff;
  buff = (text & 0xF0)>>4;
  RS = 1;
  E = 1;
  LATD = (LATD & 0xF0)|buff;
  lcd_delay(1);
  E = 0;
  lcd_delay(1);

  buff = (text & 0x0F);
  RS = 1;
  E = 1;
  LATD = (LATD & 0xF0)|buff;
  lcd_delay(1);
  E = 0;
  lcd_delay(1);
}
```

```
// Function show string message //
void lcd_puts(unsigned char line,char *p)
{
 lcd_origin();
 lcd_command(line);
 while(*p)
 {
  lcd_text(*p);
  p++;
 }
}

// Convert long integer to ascii //
void _ultoa(unsigned long value,char *string,unsigned char radix)
{
 unsigned char index;
 char buffer[NUMBER_OF_DIGITS];
 index = NUMBER_OF_DIGITS;

 do
 {
  buffer[-index] = '0' + (value % radix);
  if (buffer[index] > '9') buffer[index] += 'A' - '9' - 1;
  value /= radix;
 }while (value != 0);

 do
 {
  *string++ = buffer[index++];
 } while (index < NUMBER_OF_DIGITS);
 *string = 0;
```

```
}

// Convert long integer to ascii //
void _ltoa(long value_1,char *string_1,unsigned char radix_1)
{
 if (value_1 < 0 && radix_1 == 10)
  {
   *string_1++ = '-';
   _ultoa(-value_1,*string_1,radix_1);
  }
else
  {
   _ultoa(-value_1,*string_1,radix_1);
  }
}

// Convert integer to acii for display on LCD //
void inttolcd(unsigned char posi,long value)
{
 char buff[12];
 _ultoa(value,&buff[0],10);
 lcd_puts(posi,&buff[0]);
}

// Function set initial //
void lcd_init()
{
 TRISD = 0;
 TRISFbits.TRISF0 = 0;
 TRISFbits.TRISF1 = 0;
 lcd_delay(500);
 lcd_command(0x33);
```

```
  lcd_command(0x32);
  lcd_command(0x28);
  lcd_command(0x0C);
  lcd_command(0x01);
}

//// Main Program////
#include<p30f4011.h>
#include<lcd.h>
#include<uart.h>

_FOSC(CSW_FSCM_ON & XT_PLL4)
_FWDT(WDT_OFF)

#define PORTEbits.RE4 dat

// Function display character //
void lcd_puts_step(unsigned char line,char *p,unsigned int msec)
{
 lcd_command(0x02);
 lcd_command(line);
 while(*p)
 {
  lcd_text(*p);
  p++;
  delay_ms(msec);
 }
}


// interrupt TX //
void _ISR _U1TXInterrupt(void)
```

```
{
 IFS0bits.U1TXIF = 0;
}

// interrupt RX //
void _ISR _U1RXInterrupt(void)
{
 IFS0bits.U1RXIF = 0;
}

// Function config //
void uart1_init()
{
 unsigned int baudvalue;
 unsigned int U1MODEvalue;
 unsigned int U1STAvalue;

 CloseUART1();
 ConfigIntUART1(UART_RX_INT_EN &
  UART_RX_INT_PR6 &
  UART_TX_INT_EN &
  UART_TX_INT_PR2);

 baudvalue = 11;

 U1MODEvalue = UART_EN &
  UART_IDLE_CON &
  UART_RX_TX &
  UART_DIS_WAKE &
  UART_DIS_LOOPBACK &
  UART_DIS_ABAUD &
  UART_NO_PAR_8BIT &
```

```
   UART_1STOPBIT;

  U1STAvalue = UART_INT_TX_BUF_EMPTY &
   UART_TX_PIN_NORMAL &
   UART_TX_ENABLE &
   UART_INT_RX_3_4_FUL &
   UART_ADR_DETECT_DIS &
   UART_RX_OVERRUN_CLEAR;
  OpenUART1(U1MODEvalue,U1STAvalue,baudvalue);
}

// Main //
int main()
{
  char dat;
  uart1_init();
  lcd_init();

  while(1)
  {
  if(DataRdyUART1())
  {
   dat = ReadUART1();
   lcd_clear();
   lcd_puts(LINE1,&dat);
   delay_ms(3000);
  }
  }
  return 0;
}
```