แบบรายงานโครงการวิจัย
โดยใช้เงินรายได้คณะวิศวกรรมศาสตร์
ประจำปี 2551

ชื่อโครงการ
อุปกรณ์เสริมคีย์บอร์ดสำหรับผู้พิการทางสายตา
(Keyboard accessory for blind)

ผู้รับผิดชอบโครงการ

หัวหน้าโครงการวิจัย.....รศ.ดร.อรรถสิทธิ์ หล่าสกุล

หน่วยงาน

ภาควิชาวิศวกรรมสารสนเทศ คณะวิศวกรรมศาสตร์

พระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

อันเนื่องมาจาก การใช้งานคอมพิวเตอร์ไม่ว่าจะเพื่อการสืบค้นหาข้อมูล หรืองานด้านเอกสารต่างๆได้เข้ามามีส่วนสำคัญต่อบุคคลทั่วไปในปัจจุบัน แต่หากผู้ใช้งานเป็นผู้ที่พิการทางสายตาแล้ว อุปสรรคในการใช้งานคอมพิวเตอร์ก็จะมีมากมายยิ่งขึ้น มีผู้วิจัยหลายท่านและหลายองค์กรณ์ ได้พยายามสร้างอุปกรณ์ช่วยเหลือผู้พิการทางสายตาออกมา มีมากมายซึ่งส่วนมากจะเป็นลักษณะของซอฟร์แวร์ เช่น ซอฟร์แวร์ช่วยอ่านข้อมูลในเว็ปไซด์ เหล่านี้เป็นต้น แต่อุปสรรคที่สำคัญอย่างหนึ่งที่น่าจะนำมาพิจารณาทำวิจัยคือการใช้งานตัวคีย์บอร์ด ซึ่งผู้พิการทางสายตาที่เป็นผู้เริ่มต้นที่ไม่ชำนาญการใช้งานคีย์บอร์ดที่จำเป็นจะต้องใช้ในการฝึกฝน ดังนั้นอุปกรณ์เสริมคีย์บอร์ดที่มีลักษณะเฉพาะพิเศษสำหรับผู้พิการทางสายตาจึงเป็นสิ่งจำเป็นอย่างมากที่จะช่วยแก้ปัญหาให้กับผู้พิการทางสายตา

โครงงานวิจัยนี้จึงเสนอเพื่อของบประมาณการทำวิจัย เพื่อทำอุปกรณ์เสริมคีย์บอร์ดสำหรับผู้พิการทางสายตา โดยจะมีลักษณะที่สำคัญคือเปล่งเสียงออกมาทั้งภาษาไทยและภาษาอังกฤษในอักษรที่กด และยังมีคุณสมบัติอื่นที่จำเป็นอีกมาก เพื่อให้ผู้พิการทางสายตาได้สามารถใช้งานกับคีย์บอร์ดมาตฐานทั่วไปได้อย่างมีสิทธิภาพ.

Abstract

Currently, Personal Computer (PC) becomes the most efficiency machine and widely use in many kind of working area. Furthermore, by using GUI programming system makes its more fun to learning, especially, for a sighted person. Otherwise, for blind persons or those who are visually impaired, using Personal Computer (PC) become hard and harder to learn. Standard input system of Personal Computer (PC) such as keyboard still being common and simple way of computer accessible. This device should be adapted to suite with those persons. According to above reason, sound keyboard (keyboard accessory for blind person) is implemented in this research.

This device (sound keyboard) has been optimized for accessibility to those who are visually impaired. The device has characteristic such as: can be used with standard keyboard (both USB/PS2 types), small size and moveable. Its give sound out in both (Thai/English) languages. It is very easy to use (less switch configuration). This characteristic makes device very useful for blind person. It can be use to training and improve typing efficiency without help of a sighted person.

สารบัญ

หัวข้อ	หน้า
สารบัญตาราง	4
สารบัญรูป	4
บทที่ 1 บทนำ	5
บทที่ 2 เอกสารและงานวิจัยที่เกี่ยวข้อง/การทบทวนวรรณกรรม	6
บทที่ 3 วิธีดำเนินการวิจัยและผลการวิจัย	7
3.1 แผนงานระยะต่างๆของการดำเนินงานวิจัย	7
3.2 การสร้างส่วนของฮาร์ดแวร์	8
3.2.1 ส่วนติดต่อกับคีย์บอร์ด	12
3.2.1.1 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ AT	12
3.2.1.2 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ PS/2	13
3.2.1.3 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ USB	15
3.3 การสร้างส่วนของซอฟร์แวร์	16
3.3.1 ซอฟร์แวร์ส่วนของการทำงานโหมดที่หนึ่ง	16
3.3.2 ซอฟร์แวร์ส่วนของการทำงานโหมดที่สอง	18
3.4 การใช้งาน	18
บที่ 4 อภิปรายผลการวิจัยและวิจารณ์	21
บทที่ 5 สรุปและข้อเสนอแนะ	21
บรรณานุกรม	22
ภาคผนวก ส่วนฮาร์แวร์ที่สำคัญ และซอฟร์แวร์	23

สารบัญตาราง

ตารางที่ 1 แสดงช่วงเวลาการทำงาน
ตารางที่ 2 ค่าสแกนโค้ดของแป้นพิมพ์ (Keyboard Scan Code)
ตารางที่ 3 ความสัมพันธ์การส่งบิตข้อมูลระหว่างบิตและข้อมูล
ตารางที่ 4 แสดงคีย์ โค๊ดที่ให้จำนวนค่าไม่เท่ากัน

สารบัญรูป

รูปที่ 1 แสดงซอฟร์แวร์บางตัวสำหรับออกเสียงคีย์ที่ใช้กับคอมพิวเตอร์เท่านั้น
รูปที่ 2 บล็อกไดอะแกรมของระบบ
รูปที่ 3 แสดงคีย์บอร์ดและ Code ประจำตัว
รูปที่ 4 แสดงรูปสัญญาณที่ออกไปบันทึกตัวไอซี APR6016
รูปที่ 5 ขั้วต่อของแป้นพิมพ์ แบบ 5 Pin DIN (AT/XT)
รูปที่ 6 ขั้วต่อของแป้นพิมพ์ แบบ 6 Pin Mini-DIN (PS/2)
รูปที่ 7 แผนผังการส่งข้อมูลจากแป้นพิมพ์ส่งไปยังคอมพิวเตอร์ (Keyboard to Host)
รูปที่ 8 แสดงการเชื่อมต่อระหว่างตัว MCS-51 กับหัวต่อแบบ PS/2 และ USB
รูปที่ 9 แสดงรูปหัวต่อ USB
รูปที่ 10 แสดงบล็อกไดอะแกรมของซอฟร์แวร์ที่ทำงานในโหมดที่หนึ่ง
รูปที่ 11 ลักษณะการทำงานของโปรแกรมในโหมดที่สอง
รูปที่ 12 รูปแสดงส่วนด้านใน,ด้านข้าง ของเครื่อง
รูปที่ 13 รูปการต่อใช้งานในโหมดที่หนึ่ง
รูปที่ 14 รูปแสดงโปรแกรมที่รันไว้เป็น Background ตรวจจับคีย์
รูปที่ 15 แสดงการเชื่อมต่อใช้งานในโหมดที่สอง (ใช้งานกับคอมพิวเตอร์)

บทที่ 1
บทนำ

ปัจจุบันนี้ การใช้งานคอมพิวเตอร์มีใช้อย่างกว้างขวาง และส่วนติดต่อกับตัวคอมพิวเตอร์เองก็จะเป็นคีย์บอร์ด สำหรับคนปกติทั่วไปแล้ว จะสามารถเรียนรู้และใช้งานคอมพิวเตอร์ผ่านคีย์บอร์ดจะเป็นไปได้อย่างสะดวกไม่มีปัญหามากนัก แต่สำหรับผู้พิการทางสายตาแล้ว การใช้งานคีย์บอร์ดจะเป็นไปด้วยความยากลำบากมาก โดยเฉพาะผู้ที่หัดเริ่มใช้งานคีย์บอร์ด (อันที่จริงก็สามารถใช้กับผู้ที่สายตาดี แต่ต้องการฝึกพิมพ์แบบสัมผัส) ดังนั้น โครงงานวิจัยนี้จึงได้นำเสนอการสร้างเครื่องเชื่อมต่อคีย์บอร์ดเพื่อใช้ในการออกเสียง คีย์ที่กดนั้นๆ ออกมา (ทั้งภาษาไทยและอังกฤษ) ซึ่งจะทำให้ผู้ฝึก หรือผู้พิการทางสายตา สามารถนำไปฝึกพิมพ์ใช้งานได้อย่างสะดวก ทั้งนี้เพราะ ในงานวิจัยนี้ได้กำหนดคุณสมบัติพื้นฐานที่สำคัญดังนี้

คุณสมบัติมาตฐานโดยรวมเป็นดังนี้

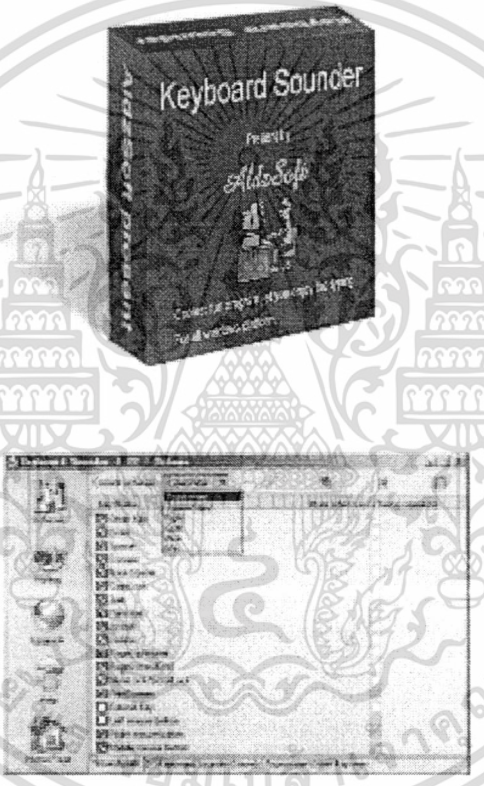
- ใช้ต่อเชื่อมกับคีย์มาตฐานทั่วไปได้
- ทำงานได้โดยทั้งที่มีหรือไม่มีการต่อเชื่อมกับคอมพิวเตอร์
- ออกลียงได้ทั้งไทยและอังกฤษตามคีย์ที่กด
- ใช้แบตเตอร์รี่เป็นแหล่งพลังงาน
- ขนาดเล็ก พกพาได้สะดวก

ทั้งนี้ ทั้งหมดที่ได้กล่าวข้างต้นเป็นคุณสมบัติมาตรฐานที่จะต้องมี ส่วนลักษณะฟังก์ชั่นพิเศษอื่นๆที่อาจมีเพิ่มเติม นั้น เช่น สามารถออกเสียงทวนคีย์ที่กดได้ หรือ ลักษณะอื่นๆได จะมีหรือไม่มีมากน้อยแค่ไหนนั้น จะขึ้นอยู่ในขั้นตอนการทดสอบ กับผู้การทางสายตาจริงๆ เพื่อ นำข้อคิดเห็นต่างๆจากผู้ใช้จริงนั้นมาปรับปรุงเพิ่มเติมให้ตรงจุดประสงค์ของผู้ใช้งานให้มากที่สุด ซึ่งแน่นอนว่า อาจไม่สมบูรณ์เต็มร้อยเปอร์เซนต์ อาจมีข้อที่ต้องปรับปรุงให้ดีขึ้นไปอีก ซึ่งอันนี้ก็จะได้มี สรุปไว้ในตอนท้ายของ รายงานวิจัยนี้ อย่างชัดเจน ซึ่งก็จะให้ได้เครื่องที่สมบูรณ์แบบต่อไป

ดังนั้น ประโยชน์ของเครื่องช่วยเหลือคนพิการทางสายตานี้ จะเป็นเป็นโยชน์อย่างสูง และยังสมารถนำไปใช้งาน กับคนสายตาปกติที่ต้องการฝึกฝนประสิทธิภาพการพิมพ์แบบสัมผัสได้อีกเป็นอย่างด

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง/การทบทวนวรรณกรรม

ในการดำเนินงานวิจัยนี้ ก็ได้มีการค้นคว้าหาข้อมูลที่เกียวข้องกับงานที่ทำทั้งในเว็ปไซด์ [3], [4] และสิ่งพิมพ์ต่างๆ แต่ยังไม่พบอุปกรณ์ฮาร์ดแวร์ที่ตรงเป้าหมายนัก จะมีก็เป็นเพียงแต่เป็นซอฟร์แวร์ที่รันบน windows ซึ่งสามารถใช้ได้กับคอมพิวเตอร์ที่มีระบบปฏิบัติการ windows เท่านั้น ดังในเอสารอ้างอิง [5] ซึ่งเป็นเว็ปไซด์ที่ให้ข้อมูลเกี่ยวกับซอฟร์แวร์ดังกล่าว ดังแสดงตัวอย่างบางซอฟร์แวร์ในรูปข้างล่างนี้



รูปที่ 1 แสดงซอฟร์แวร์บางตัวสำหรับออกเสียงคีย์ที่ใช้กับคอมพิวเตอร์เท่านั้น

ดังนั้น ในงานวิจัยนี้จึงได้ออกแบบและสร้างระบบขึ้นมาเอง โดยให้ได้ดังจุดประสงค์ดังที่ได้กล่าวมาในบทนำ โดยเลือกอุปกรณ์ที่มีขนาดเล็ก เช่น เลือกหน่วนยประมวลผล (CPU) ขนาดเล็กคือ MCS-51 ซึ่งหาได้ง่ายและราคาถูกและมีประสิทธิภาพที่พอเพียงต่อความต้องการ, ส่วนของการบันทึกเสียง ก็เลือกใช้ ชิฟไอซี ที่มีจำหน่ายในเมืองไทย นำมาประกอบกับส่วนอื่นๆ อีกเล็กน้อย เช่น ส่วนของภาคขยายเสียง, แบตเตอร์รี่ ดังรายละเอียดการสร้างและการทดลองและผลการทดลองต่างๆ ก็จะได้ นำเสนอในส่วนของ บทท้ายๆ นี้ต่อไป

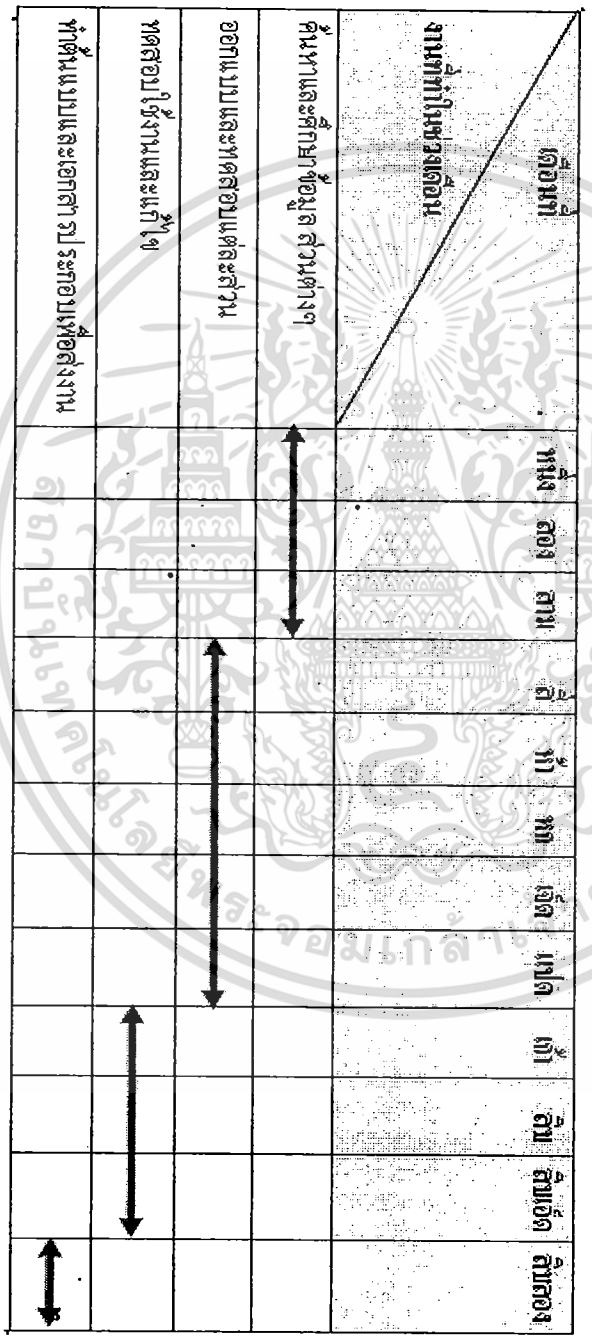
บทที่ 3

วิธีดำเนินการวิจัยและผลการวิจัย

3.1 แผนงานระยะต่างๆของการดำเนินงานวิจัย

ระยะเวลาวิจัยรวม หนึ่งปี ดังแสดงตารางช่วงเวลาการทำงาน ในแต่ละส่วน ในตารางที่ 1

หมายเหตุ เดือนที่หนึ่ง หมายถึง เดือนที่นับจากเดือนที่ได้รับอนุมัติโครงงานวิจัย และเดือนที่สิบสอง หมายถึง เดือนสุดท้ายของการทำโครงงานวิจัย

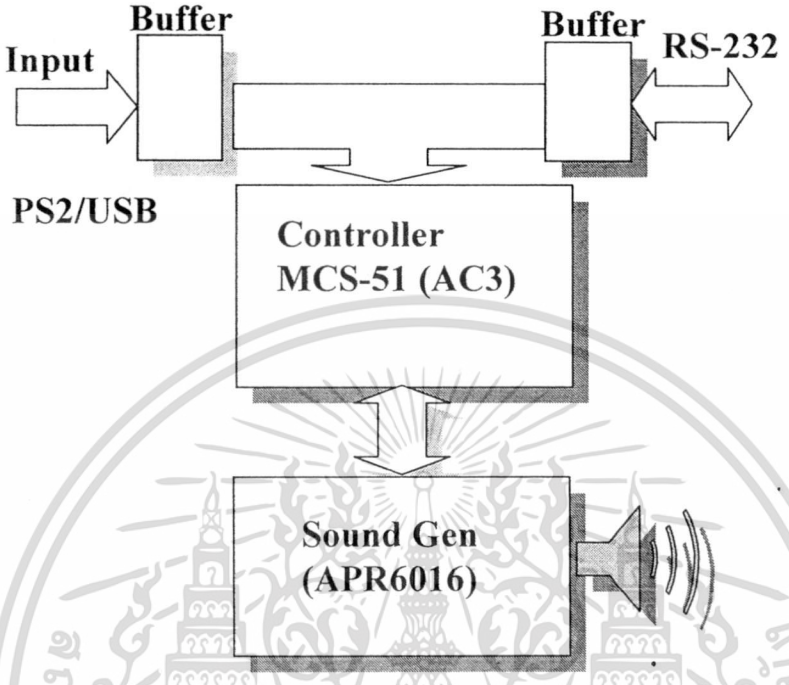


ตารางที่ 1 แสดงช่วงเวลาการทำงาน

งานวิจัยนี้ได้แบ่งออกเป็นสองส่วน คือส่วนของ ฮาร์ดแวร์และซอฟร์แวร์ ดังนี้

3.2 การสร้างส่วนของฮาร์ดแวร์

การสร้างสามารถแสดงได้ดังรูป บล็อกไออะแกรม ดังรูปที่ 2



รูปที่ 2 บล็อกไดอะแกรมของระบบ

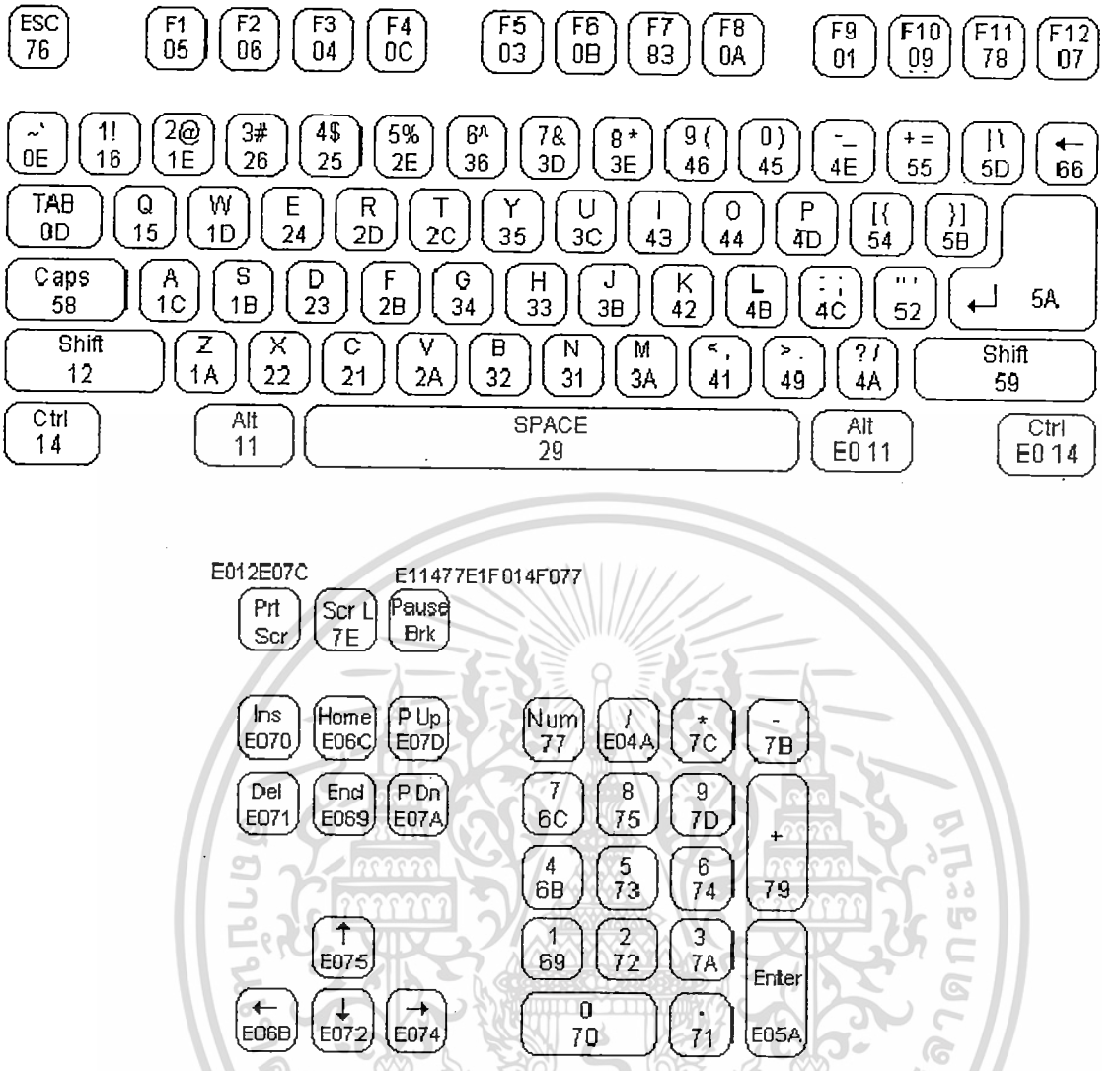
การทำงานเป็นดังนี้ โดยตัวเครื่องจะมีการทำงานเป็นสองโหมด คือ

<u>โหมดที่หนึ่ง</u> จะใช้งานกับคีย์บอร์ด (อาจเป็นได้ทั้ง PS2และ USB) สัญญาณ จากคีย์บอร์ดจะถูกนำเข้ามาสู่ตัว CPU เพื่อถอดระหัสคีย์บอร์ด แล้วนำ ระหัสที่ได้นี้ไปเปิดตารางที่จะชี้ไปสู่ส่วนออกเสียง ที่ถูกบันทึกไว้แล้วในตัวของ APR6016 นำเสียงนั้น ออกลำโพงผ่านภาคขยายเสียงต่อไป ซึ่งในส่วนของระหัสที่ได้จากคีย์บอร์ดนั้น แสดงไว้ดัง ตารางที่ 2 ซึ่งในส่วนของ Code ที่แสดงนี้จะได้จากการตรวจสอบสัญญาณที่ออกมาจากคีย์บอร์ด โดยวงจรในการตรวจสอบจะแสดงไว้แล้วในส่วนของ ภาคผวก ซึ่งค่าของ Code ที่ ตรวจได้นี้จะถูกนำไปชี้หาตำแหน่งของเสียงคีย์ นั้นๆ โดยเสียงของ คีย์ ต่างๆ ทั้ง ไทยและ อังกฤษ ก็จะถูกบันทึกไว้แล้วในตัวของ ไอซี APR6016 โดยจะมีการ บันทึก โดยการ นำฟล์ของเสียง (WAV) ที่ได้บันทึกแล้วไป เข้าสู่โปรแกรมคอมพิวเตอร์ และสร้างไฟล์เสียงที่มีสถานะ เป็น ศูนย์และ หนึ่งอีกหนึ่งไฟล์ เพื่อใช้ในการบันทึกเสียงนี้ ลงไปสู่ ไอซีเสียงอย่างอัตโนมัติ อันนี้ทำได้โดยนำไฟล์เสียงและไฟล์ที่มีสถานะเป็นศูนย์หรือหนึ่งนี้ เข้าสู่ตัวของ

ตารางที่ 2 ค่าสแกนโค้ดของแป้นพิมพ์ (Keyboard Scan Code)

Key	Code	Key	Code
	0E	A	1C
1	16	S	1B
2	1E	D	23
3	26	F	2B
4	25	G	34
5	2E	H	33
6	36	J	3B
7	3D	K	42
8	3E	L	4B
9	46	;	4C
0	45	'	52
-	4E	Enter	5A
=	55	Left Shift	12
Back Space	66	Z	1A
Tab	0D	X	11
Q	15	C	21
W	1D	V	2A
E	24	B	32
R	2D	N	31
T	2C	M	3A
Y	35	,	41
U	3C	.	49
I	43	/	4A
O	44	Right Shift	59
P	4D	Left Ctrl	14
[	54	Left Alt	11
]	5B	Spec	29
\	5D	Right Alt	E011

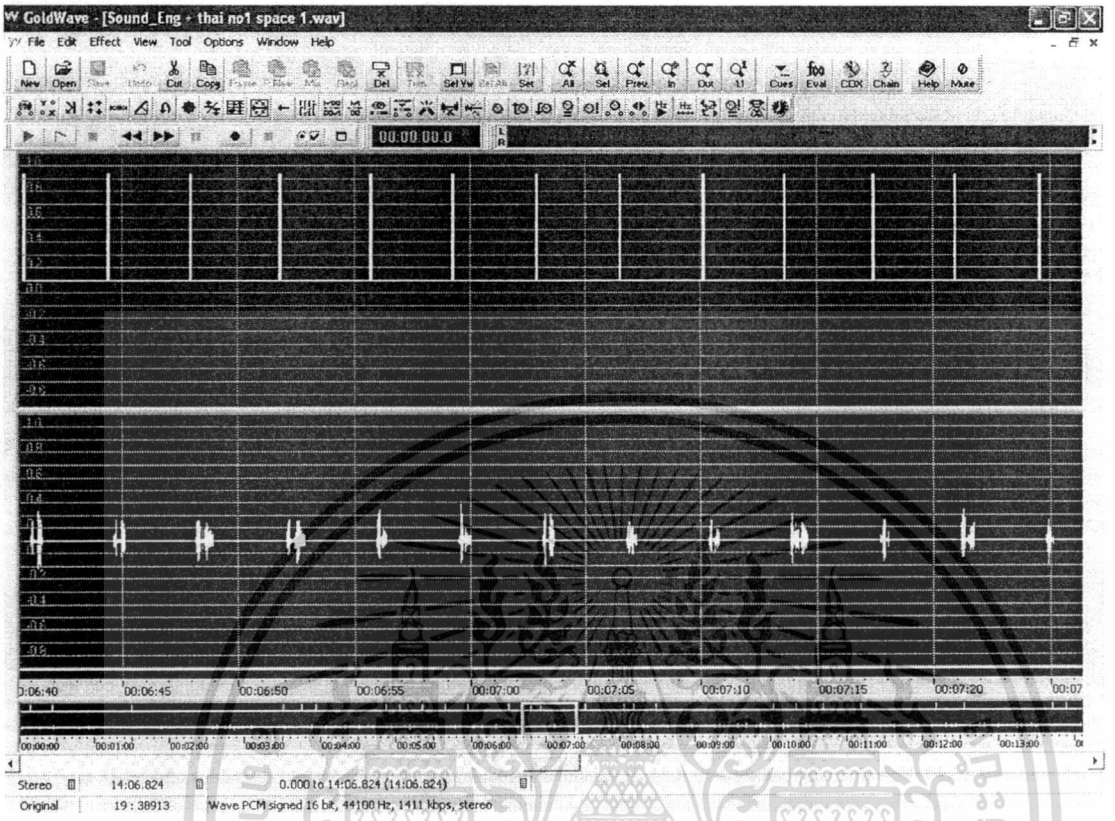
Key	Make Code	Key	Make Code
Right Ctrl	E014	Home	E06C
F1	05	Pg Up	E07D
F2	06	Del	E071
F3	04	End	E069
F4	0C	Pg Dn	E07A
F5	03	Up Arrow	E075
F6	0B	Left Arrow	E06B
F7	83	Right Arrow	E074
F8	0A	Down Arrow	E072
F9	01	PrtSC	E012E07C
F10	09	Ctl-PrtSc	E07C
F11	78	Alt-PrtSc	84
F12	07	ScrLK	7E
Num Lock	77	Pause	E01477E1F01444
KP-	7B	Ctrl-Break	E07EE0F07E
KP/	E04A		
KP.	71		
KP*	7C		
KP+	79		
KPEnter	E05A		
KP0	70		
KP1	69		
KP2	72		
KP3	7A		
KP4	6B		
KP5	73		
KP6	74		
KP7	6C		
KP8	75		
KP9	7D		



รูปที่ 3 แสดงคีย์บอร์ดและ Code ประจำตัว

โปรแกรม Gold wave ซึ่งเป็นโปรแกรมประเภท Edit sound ก็จะทำให้ได้เสียงออก ช่องที่หนึ่งและสอง ตามลำดับ จากนั้น ก็นำไปส่งสัญญาณเสียงและสัญญาณ ศูนย์และหนึ่งที่สร้างขึ้นนี้ใช้เพื่อออกไปเข้าสู่ตัววงจรของ APR6016 เพื่อบันทึกต่อไป โดยช่องสัญญาณเสียงที่บันทึกไว้ก็เข้าสู่ ช่องเสียงอินพุทของไอซี และส่วนช่องสัญญาณศูนย์/หนึ่ง ก็เข้าสู่วงจรควบคุมกำหนดการอัดเสียง (ดูวงจรได้จากคู่มือไอซี) ในรูปที่ 3 เป็นการแสดงสัญญาณที่จะบันทึกสู่ตัว ไอซีเสียง

เมื่อได้ทำการอัดเสียงลงในตัวไอซีแล้ว ต่อไปก็นำไอซีมาต่อใช้งานร่วมกับ MCS-51 ซึ่งสามารถต่อวงจรได้ตาม ภาคผนวก ซึ่งสามารถดัดแปลงเป็นตัวสแกนหาตำแหน่งที่อยู่ของเสียงแต่ละเสียงได้เพื่อจะได้นำไปใช้ในโปรแกรม การชี้ตารางเสียงต่อไป

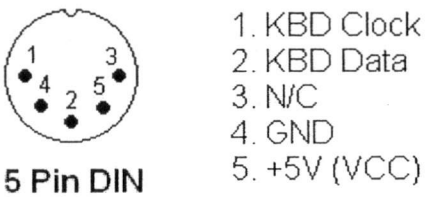


รูปที่ 4 แสดงรูปสัญญาณที่ออกไปบันทึกตัวไอซี APR6016

3.2.1 ส่วนติดต่อกับคีย์บอร์ด

3.2.1.1 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ AT

(มาตรฐาน IBM 102/104/107 คีย์) มีลักษณะเป็นสาย 4 เส้น ต่อแบบปลั๊กตัวผู้แบบ DIN ดังรูปที่ 5



รูปที่ 5 ขั้วต่อของแป้นพิมพ์ แบบ 5 Pin DIN (AT/XT)

การทำงานของขาแต่ละตำแหน่ง มีการหน้าที่ต่อดังนี้

ขาที่ 1 คือ KBD Clock

ขาที่ 2 คือ KBD Data

ขาที่ 3 คือ Not Implemented

ขาที่ 4 คือ Ground

ขาที่ 5 คือ +5V (VCC)

3.2.1.2 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ PS/2

(มาตรฐานIBM 102/104/107 คีย์) มีลักษณะเป็นสาย 4 เส้น ต่อแบบปลั๊กตัวผู้แบบ DIN ดังรูปที่ 6

รูปที่ 6 ขั้วต่อของแป้นพิมพ์ แบบ 6 Pin Mini-DIN (PS/2)

การทำงานของขาแต่ละตำแหน่ง มีการหน้าที่ต่อดังนี้

ขาที่ 1 คือ KBD Clock

ขาที่ 2 คือ GND

ขาที่ 3 คือ KBD DATA

ขาที่ 4 คือ N/C

ขาที่ 5 คือ +5V (VCC)

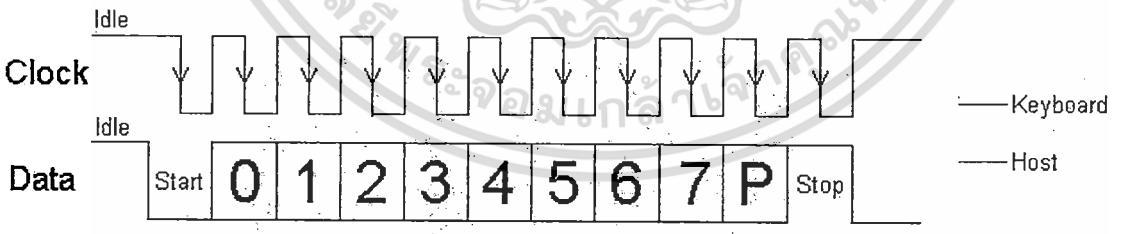
ขาที่ 6 คือ Not Implemented

การส่งข้อมูลของแป้นพิมพ์แบบ AT และ PS/2 นี้จะใช้โปรโตคอลเป็น Clock แบบอนุกรมขนาดของข้อมูล 11 บิต มีค่าอัตราของ Clock ประมาณ 10-20 KHz ลักษณะข้อมูลที่ส่งจากแป้นพิมพ์มีความหมายดังตารางที่ 3

Bit	Function
1	START Bit (Logic "0")
2	Data Bit 0
3	Data Bit 1
4	Data Bit 2
5	Data Bit 3
6	Data Bit 4
7	Data Bit 5
8	Data Bit 6
9	Data Bit 7
10	Parity Bit
11	Stop Bit (Logic "1")

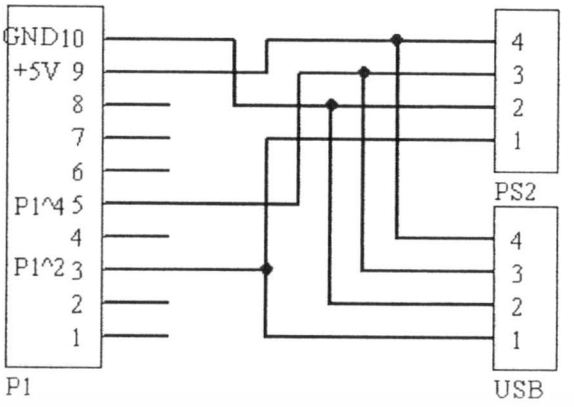
ตารางที่ 3 ความสัมพันธ์การส่งบิตข้อมูลระหว่างบิตและข้อมูล

จากตารางที่ 3 บิตแรกเป็นบิตเริ่มต้น (Start Bit) มีระดับลอจิกเป็น"0" แล้วตามด้วย 8 บิต ข้อมูล (บิตที่มีน้ำหนักน้อย LSB ถูกส่งไปก่อน) หนึ่งบิตพาริตี้แบบ Odd (Odd Parity Bit) และปิดท้ายด้วยบิตสิ้นสุด (Stop Bit) ต้องเป็นลอจิก"1" ทั้งหมดสัมพันธ์กับสัญญาณนาฬิกาที่ขอบขาลงดังรูปสัญญาณในรูปที่ 7



รูปที่ 7 แผนผังการส่งข้อมูลจากแป้นพิมพ์ส่งไปยังคอมพิวเตอร์ (Keyboard to Host)

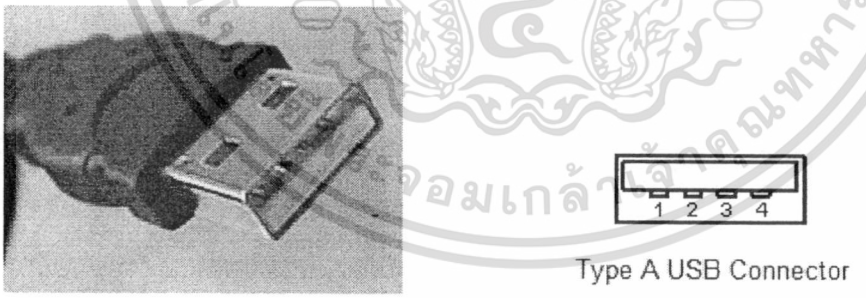
จากรูป สัญญาณข้างบน เราสามารถนำมาเขียนเป็นโปรแกรมบน MCS-51 เพื่อรับข้อมูลได้โดยไม่ยากนัก ซึ่งส่วนนี้ ก็ได้มีไว้แล้วใน ภาคผนวก



รูปที่ 8 แสดงการเชื่อมต่อระหว่างตัว MCS-51 กับหัวต่อแบบ PS/2 และ USB

3.2.1.3 ขั้วต่อของแป้นพิมพ์ (Keyboard's Connector) แบบ USB

ในส่วนของหัวต่อแบบ USB นี้ เนื่องจากเราไม่ได้ต่อเชื่อมกับคอมพิวเตอร์ในโหมดที่หนึ่งนี้ ทำให้ไม่มี Operating system ที่จะต้องตรวจสอบกัน การใช้งานจึงสามารถทำเช่น เดียวกันกับการต่อแบบ PS2 ซึ่งลักษณะการต่อกันของสายสัญญาณแสดงไว้แล้วในรูปที่ 8 และรูปลักษณะของหัวต่อ USB แสดงไว้ดังรูปที่ 9



รูปที่ 9 แสดงรูปหัวต่อ USB

<u>โหมดที่สอง</u> การทำงานในโหมดนี้ เป็นการใช้งานกับคอมพิวเตอร์ซึ่งปกติจะมีคีย์บอร์ดต่อเชื่อมอยู่แล้ว ดังนั้น ในแนวคิดของการใช้งานก็คือ จะมีการรันโปรแกรมที่เป็น แบบทำงานเบื้องหลัง (background) ไว้บนคอมพิวเตอร์ เพื่อคอยตรวจจับการกดคีย์บอร์ด และส่งข้อมูลคีย์บอร์ดนั้นออกมา ทาง พอร์ทมาตฐาน ที่มีใช้อยู่ทั่วไปคือ พอร์ท RS-232 ดังนั้น ตัวของเครื่องสร้างเสียงนี้ จึง

ทำหน้าที่เพียงรับค่าคีย์นั้นเพื่อไปเปิดตารางออกเสียงให้ถูกต้องต่อไป โดยกำหนดรูปแบบการรับส่งไว้ที่ตัวเครื่องคือ กำหนดให้ รับและส่งข้อมูลกันใน อัตตรา 9600 บิทต่อวินาที และ ไม่มี พาริตี้บิท และ หนึ่ง บิทหยุด (9600 8 N 1)

3.3 การสร้างส่วนของซอฟร์แวร์

ดังที่กล่าวมาแล้วในหัวข้อ ฮาร์ดแวร์ว่า มีการทำงานด้วยกัน สองโหมด โดยโหมดที่หนึ่งคือการทำงานแบบต่อเฉพาะคีย์บอร์ดก็ใช้งานได้เลย ส่วนในโหมดที่สองนั้น เป็นการต่อใช้งานกับเครื่องคอมพิวเตอร์ ดังนั้น จึงต้องมี ซอฟร์แวร์เป็น สองส่วนแยกทำงานในแต่ละโหมด โดยในตัวเครื่องก็จะมี สวิทช์ที่ใช้สำหรับการเลือกว่าจะใช้งานเครื่องในโหมด ใด

3.3.1 ซอฟร์แวร์ส่วนของการทำงานโหมดที่หนึ่ง

ในโหมดนี้ สามารถแสดงได้ดัง ไออะแกรม ในรูปที่ 10 ในโหมดนี้เป็นการรับค่าของคีย์บอร์ดจาก พอร์ท PS2 หรือ USB ซึ่งค่าของคีย์นั้นได้มีการกำหนดมาแล้วดัง แสดงในบทที่ผ่านมา และจากรูปแบบของค่าคีย์ที่ส่งมา จะสังเกตุเห็นได้ว่า มีคีย์บางคีย์ที่ ส่งเฉพาะ ค่า code เพียงค่าเดียว บางปุ่ม ก็จะเป็นคีย์พิเศษ เช่น คีย์ PAUSE ซึ่งจะส่งค่าที่หลายค่า และที่สำคัญบางปุ่มก็มีการทำงานเป็นแบบ Toggle (สลับไปมา เช่น ปุ่ม Numlock) เหล่านี้เป็นต้น ดังนั้นการเขียนโปรแกรมรับคีย์จึงต้องมีการตรวจสอบหลายชั้น เพราะการส่งค่าคีย์ที่ไม่เท่ากันดังที่กล่าวมา

คีย์	ค่าโค๊ด
PrtSC	E012E07C
Ctl-PrtSc	E07C
Alt-PrtSc	84
ScrLK	7E
Pause	E01477E1F01444
Ctrl-Break	E07EE0F07E

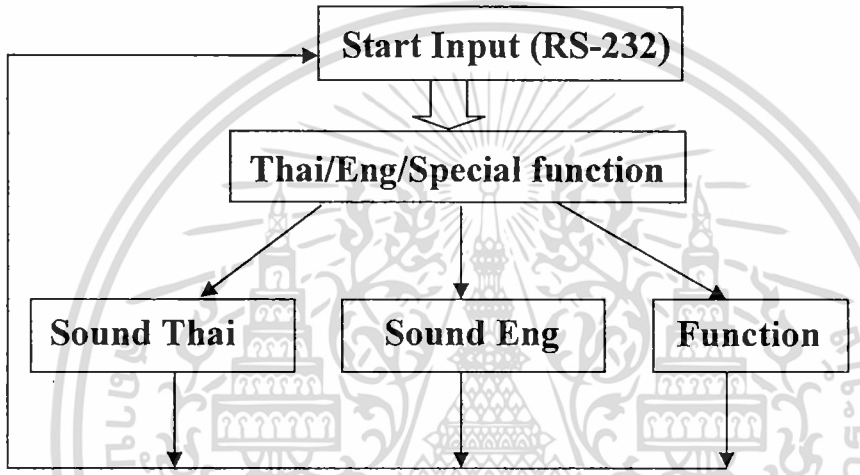
ตารางที่ 4 แสดงคีย์ โค๊ดที่ให้จำนวนค่าไม่เท่ากัน

จากรูปไดอะแกรม แสดงให้เห็นว่า จะมีการเก็บค่าต่างๆไว้ ก่อนเช่น เป็นการกดคีย์เปลี่ยนไทยเป็น อังกฤษหรือเปล่า, มีการกดู คีย์ Caps Lock หรือเปล่า เป็นต้น เพื่อที่จะได้นำไปชี้ตารางเสียงที่ถูกต้องต่อไป ทั้งนี้เพราะตารางเสียงจะมีถึง สี่ตาราง คือ ตารางเสียง Eng Big, Eng Small, Thai Big และ Thai Small เพราะทุกปุ่มจะมีเสียงที่แตกต่างกันอยู่แล้ว

รูปที่ 10 แสดงบล็อกไดอะแกรมของซอฟร์แวร์ที่ทำงานในโหมดที่หนึ่ง

3.3.2 ซอฟร์แวร์ส่วนของการทำงานโหมดที่สอง

ในการทำงานในโหมดนี้จะเป็นการรับค่า คีย์ผ่านมาทาง RS-232 ของคอมพิวเตอร์ ซึ่งจะเป็นคีย์ของแต่ละตัวที่ให้ค่าไม่หมือนกัน ซึ่งในทางโปรแกรมมิ่งแล้ว เราสามารถกำหนดค่าใหม่ให้กับคีย์ได้เลย ซึ่งหน้าที่ส่วนนี้ในงานวิจัยนี้ได้ทำการเขียนโดย VC#2205 ซึ่งมีแสดงไว้แล้วใน ส่วนของภาคผนวก และการทำงานของโปรแกรมเป็นลักษณะของโปรแกรมที่ทำงานแบบฝังตัว ทำงานตลอดเวลา คอยตรวจจับการกดคีย์แล้วจัดการส่งค่าออกมาทาง พอร์ท RS-232 ทำให้สะดวกมากในการเขียนโปรแกรมรับค่าในส่วนของ**โปรแกรมในโหมดที่สอง**

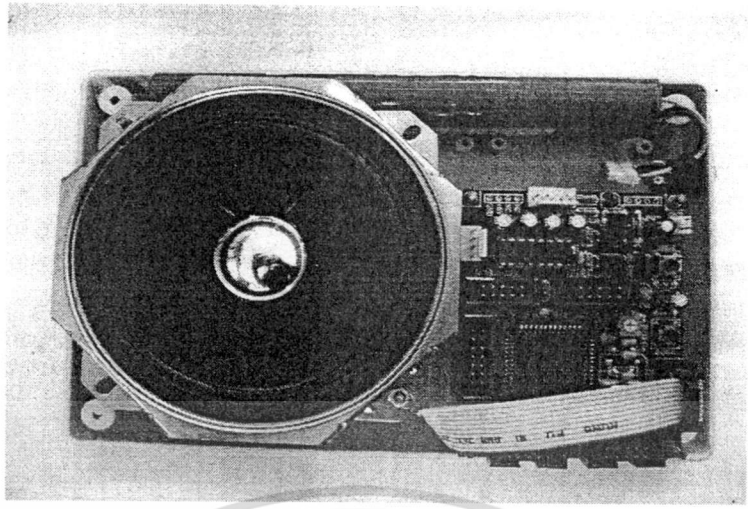


รูปที่ 11 ลักษณะการทำงานของโปรแกรมในโหมดที่สอง

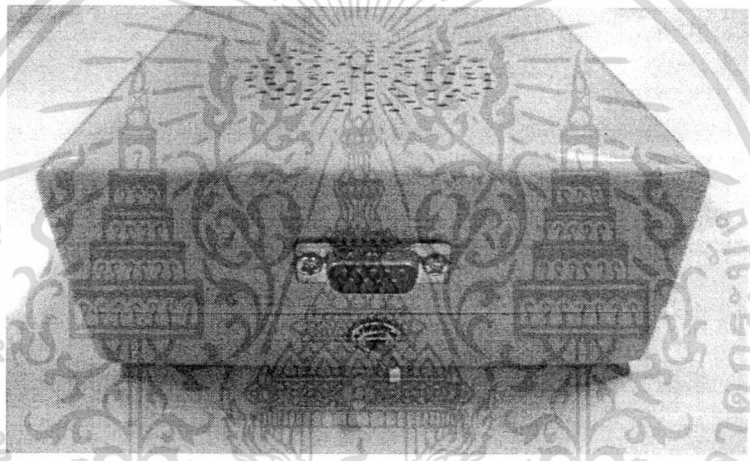
3.4 การใช้งาน

ในการใช้งานนั้น เริ่มด้วยให้ดูรูปของตัวเครื่องดังแสดงในรูปที่ 11 ว่า มีปุ่มน้อยไม่ซับซ้อนเลย และตัวแบตเตอร์รี่เองก็สามารถใช้งานได้หลาย ชั่วโมงและมีระบบประจุไฟในตัวแล้วด้วย

การใช้งานสามารถอธิบายได้ดังนี้ คือ ดังที่ทราบแล้วว่าการทำงานมีด้วยกัน สองโหมด คือ โหมดที่หนึ่ง ซึ่งเป็นการทำงานแบบไม่ต่อกับคอมพิวเตอร์ เพียงแต่ต่อเข้ากับคีย์บอร์ด(ไม่ว่าจะเป็นแบบ PS2 หรือ USB) ก็ได้ ก็สามารถทำงานได้เลย ดังนั้นแสดงได้ดังรูปที่ 13 ส่วนรูปที่ 15 เป็นการใช้งานในโหมดสอง โดยการต่อสัญญาณจาก RS-232 และมีการรันโปรแกรมบน PC ไว้ก่อนเมื่อต้องการใช้งาน ซึ่งจะเห็นได้ว่าใช้งานได้ง่ายมากทีเดียว



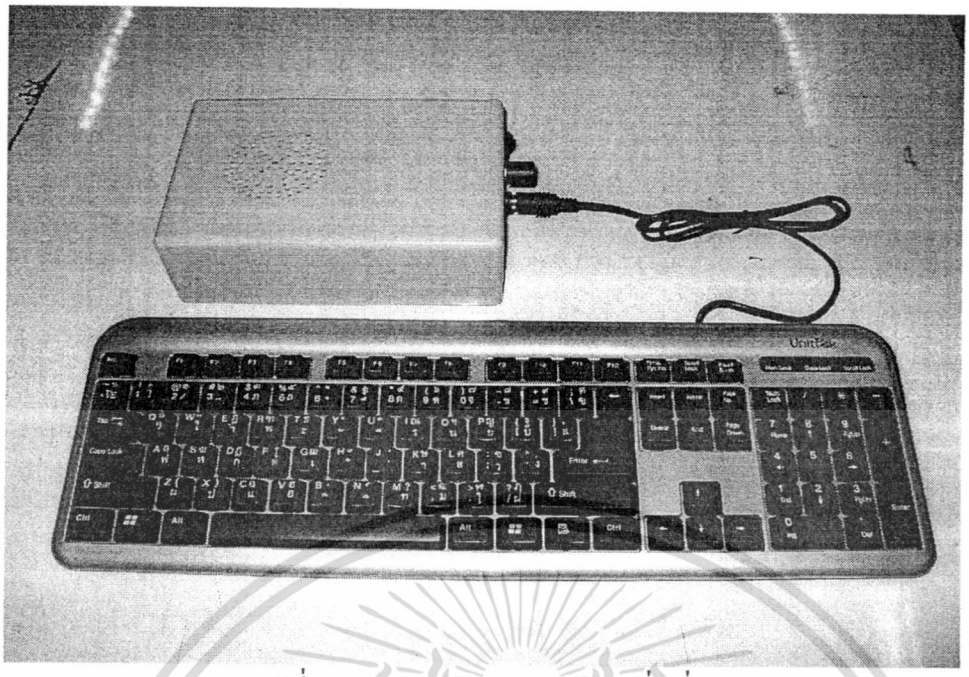
แสดงด้านใน



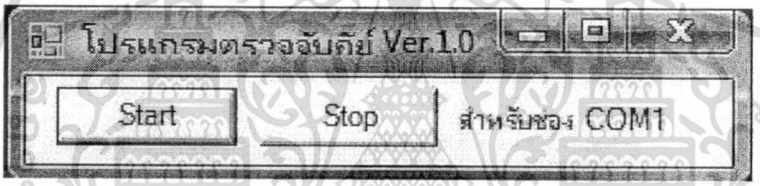
แสดงส่วนสวิชท์และหัวต่อพอร์ท RS-232

แสดงส่วนต่อสัญญาณคีย์บอร์ด, สวิชท์เปิดปิดเครื่องและปุ่มหมุนปรับระดับเสียง

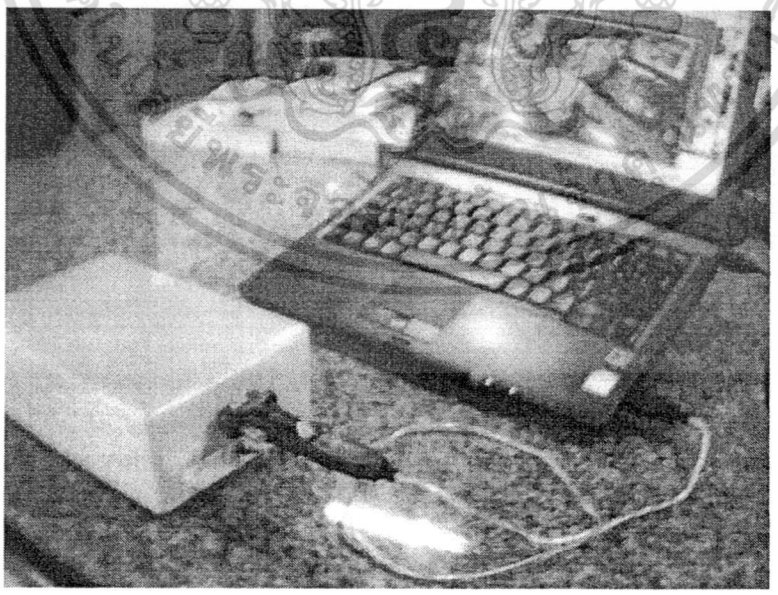
รูปที่ 12 รูปแสดงส่วนด้านใน, ด้านข้าง ของเครื่อง



รูปที่ 13 รูปการต่อใช้งานในโหมดที่หนึ่ง



รูปที่ 14 รูปแสดงโปรแกรมที่รันไว้เป็น Background เพื่อตรวจจับคีย์



รูปที่ 15 แสดงการเชื่อมต่อใช้งานในโหมดที่สอง (ใช้งานกับคอมพิวเตอร์)

บทที่ 4
อภิปรายผลการวิจัยและวิจารณ์

จากผลการวิจัย ที่ผ่านมาในบที่ 3 นั้น จะเห็นได้ว่า ส่วนของตัวเครื่องนั้นมีปุ่มให้ปรับน้อยมาก คือมีเพียงปุ่มสำหรับเลือกว่าจะใช้งานในโหมดใด ก่อนการเปิดเครื่องใช้งาน และ ปุ่มปรับความดังของเสียงที่เปล่งออกมา นอกนั้นก็ไม่ต้องการปรับแต่งอะไรเลย

จากการทดลองนำไปใช้งาน สามารถใช้งานได้ดีในระดับหนึ่ง หากแต่ถ้าสามารถปรับปรุงให้มีขนาดของเครื่องที่เล็กลงไปอีกก็จะเป็นประโยชน์ได้ไม่น้อย อนึ่งขอ ที่ควรปรับปรุงอีกอย่างคือการออกเสียงที่อาจช้าไปบ้าง หากผู้ฝึกนั้นมีความคล่องและชำนาญในการใช้คีย์บอร์ดมากขึ้นแล้วนั้น เครื่องต้นแบบนี้ ได้ใช้ไอซีเก็บบันทึกคำออกเสียงคีย์บอร์ด ทั้งไทยและอังกฤษ คือ เบอร์ APR6016 ซึ่งสามารถบันทึกได้ถึง 16 นาทีและ มีจุดที่บกพร่องคือในการสั่งอ่านออกเสียงจะต้องมีการส่งคำสั่งหลายคำสั่งและทั้งมีการติดต่อกับ CPU ของระบบเป็นแบบ SPI ทำให้อาจล้าช้าต่อการออกเสียงต่อการตอบสนองคีย์บอร์ดได้ อันนี้เป็นส่วนหนึ่งที่จะได้เสนอแนะต่อไป

บทที่ 5
สรุปและข้อเสนอแนะ

งานวิจัยนี้ เป็นการสร้างเครื่องออกเสียงคีย์บอร์ดเพื่อใช้งานกับคนพิการทางสายตา จะเห็นได้ว่า เครื่องออกเสียงคีย์บอร์ด สำหรับคนพิการทางสายตานี้ สามารถทำงานได้เป็นอย่างดีในระดับพื้นฐาน คือ โดยหลักๆก็สามารถออกเสียงทั้งไทยและอังกฤษได้อย่างถูกต้อง และก็เป็นไปตามจุดมุ่งหมายของงานวิจัยที่ตั้งไว้ในเบื้องต้น ที่นอกจากจะออกเสียงได้สมบูรณ์แล้วยัง มีน้ำหนักที่เบา , ใช้งานง่ายไม่ซับซ้อน, เคลื่อนย้ายได้สะดวก และทำงานได้ทั้งกับคีย์บอร์ด PS2/USB ทั้งยังใช้ได้กับคีย์บอร์ดเท่านั้น หรือจะใช้กับคอมพิวเตอร์ก็ได้ อีกเช่นกัน

แต่เพื่อให้เกิดประโยชน์และใช้งานได้หลากหลายขึ้น ก็ยังมีสิ่งที่ต้องปรับปรุงต่อไปอีก เช่น ปรับปรุง ฮาร์ดแวร์ให้มีขนาดเล็กลงไปอีกได้ โดย ออกแบบเป็น แผ่นวงจรปริ้นเดียวทั้งระบบ และย่อลงบน Chip เดียว เช่น บน FPGA เป็นต้น สร้างฟังก์ชั่น พิเศษอื่นๆ เพิ่มเติมเช่น การทบทวนประโยคที่พิมพ์ การตอบสนองของคีย์บอร์ดให้ออกเสียงได้เร็วขึ้นสำหรับ คนที่พิมพ์สัมผัส ได้รวดเร็ว หรือแม้แต่ เพิ่มฟังชั้นแแบฝึก การพิมพ์เข้าไปในตัวเครื่องเลย เหล่านี้ เป็น ต้น ดังนั้น จะเห็นได้ว่าแนวทางการพัฒนาต่อเพื่อให้เกิดการนำไปใช้ประโยชน์ให้ได้กว้างขวางและจริงจัง นั้น ยังมีอีกมาก.

บรรณานุกรม

[1] http://www.keil.com/

[2] http://www.ett.co.th/

[3] http://www.sunrom.com/files/APR6016.pdf

[4] http://www.clickykeyboards.com/

[5] http://www.aldzsoft.com/

[6] Han-Way Huang , "Using the MCS-51 Microcontroller" , Dec 16, 1999.

[7] ประจิน พลังสันติกุล และ ชัยวัฒน์ ลิ้มพรจิตรวิไล, " ปฏิบัติการ ไมโครคอนโทรลเลอร์ MCS-51 กับ Keil C51 ", บริษัท อินโนเวชั่น เอ็กเพอริเม้นต์ จำกัด., พ.ศ. 2550

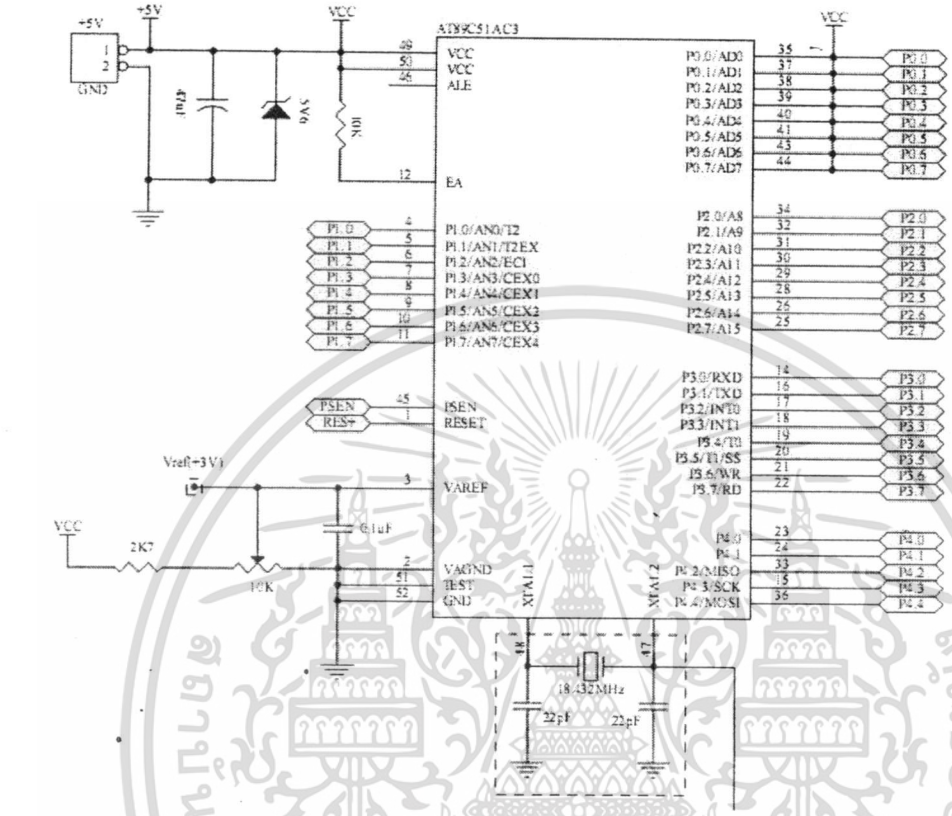
[8] สัจจะ จรัสรุ่งรวีวร , " คู่มือ Visual C# 2005 ฉบับสมบูรณ์ ", บริษัท ไอดีซีฯ, พ.ศ. 2550

[9] ลาภลอย วานิชอังกูร, " เรียนรู้ด้วยตนเอง OOP C# ASP.NET ", บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) , พ.ศ. 2550.

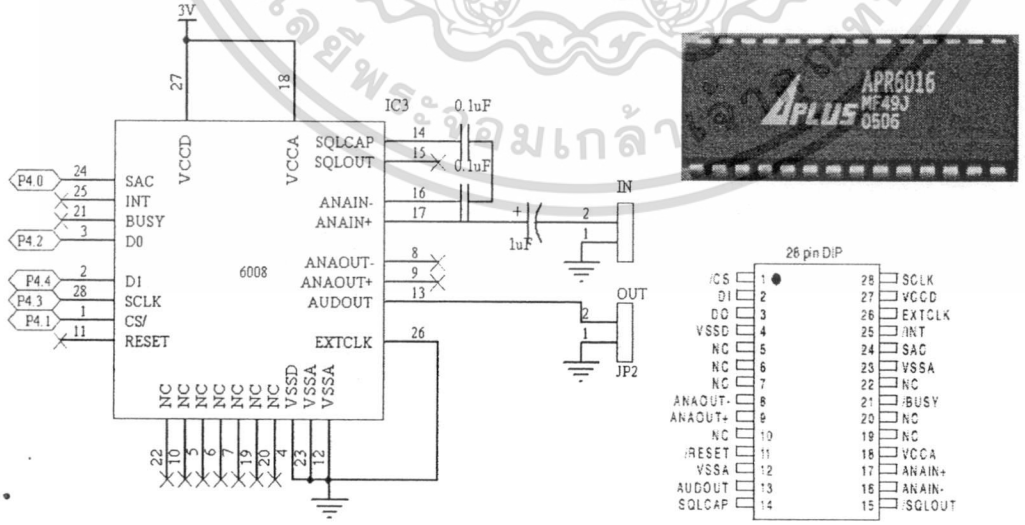
ภาคผนวก

ส่วนฮาร์แวร์ที่สำคัญ

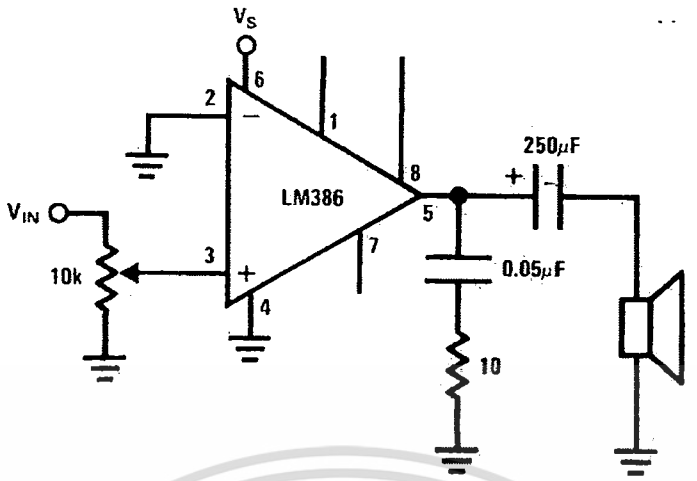
ก. วงจรเครื่องอุปกรณ์เสริมคีย์บอร์ดสำหรับผู้พิการทางสายตา



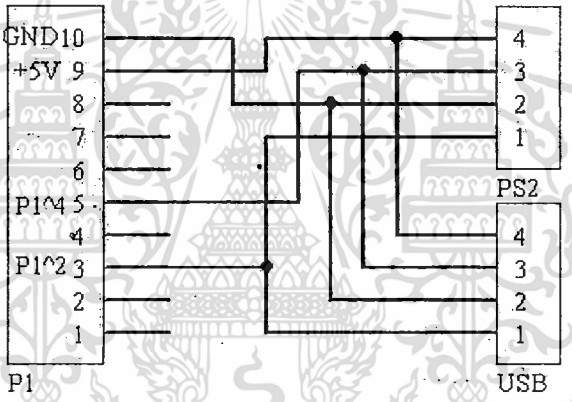
วงจร CPU



วงจรบันทึกและสร้างสัญญาณเสียง



วงจรขยายเสียง



วงจรเชื่อมต่อคีย์บอร์ด

ข. โปรแกรมส่วนของตัวเครื่อง (ภาษาซี บน MCS-51 AC3)

โปรแกรมจะมีอยู่สองส่วนสำหรับแต่ละโหมดการทำงานซึ่งยาวมากจึงนำไปใส่ไว้ใน CD-ROM จะสะดวกต่อการนำไปพัฒนาต่อ

ค. โปรแกรม ส่วนของเครื่องคอมพิวเตอร์ (ภาษา C#)

โปรแกรมส่วนของ Main

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using gma.System.Windows;


namespace GlobalHookDemo
{
    class MainForm : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button buttonStart;
        private System.Windows.Forms.Button buttonStop;
        delegate void SetCallback(string text_temp);

        private rs_232 myRs232Comport1;
        string strDown;
        string strPress;
        string strUp;
        string two_str;
        private Label label1;
        int   str_1;


        public MainForm()
        {
            InitializeComponent();
        }

        // THIS METHOD IS MAINTAINED BY THE FORM DESIGNER
       // DO NOT EDIT IT MANUALLY! YOUR CHANGES ARE LIKELY TO BE LOST
        void InitializeComponent()
        {
            this.buttonStop = new System.Windows.Forms.Button();
            this.buttonStart = new System.Windows.Forms.Button();
            this.label1 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // buttonStop
```

```
            //
            this.buttonStop.Location = new System.Drawing.Point(92,
4);
            this.buttonStop.Name = "buttonStop";
            this.buttonStop.Size = new System.Drawing.Size(71, 23);
            this.buttonStop.TabIndex = 1;
            this.buttonStop.Text = "Stop";
            this.buttonStop.Click += new
System.EventHandler(this.ButtonStopClick);
            //
            // buttonStart
            //
         this.buttonStart.Location = new System.Drawing.Point(12, 4);
            this.buttonStart.Name = "buttonStart";
            this.buttonStart.Size = new System.Drawing.Size(71, 23);
            this.buttonStart.TabIndex = 0;
            this.buttonStart.Text = "Start";
            this.buttonStart.Click += new
System.EventHandler(this.ButtonStartClick);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(169, 9);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(89, 13);
            this.label1.TabIndex = 2;
            this.label1.Text = "สำหรับช่อง COM1";
            //
            // MainForm
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(279, 34);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.buttonStop);
            this.Controls.Add(this.buttonStart);
            this.Name = "MainForm";
            this.Text = "โปรแกรมตรวจจับคีย์ Ver.1.0";
            this.Load += new System.EventHandler(this.MainFormLoad);
            this.ResumeLayout(false);
            this.PerformLayout();
        }
        [STAThread]
        public static void Main(string[] args)
        {
            Application.Run(new MainForm());
        }
        void ButtonStartClick(object sender, System.EventArgs e)
        {
            actHook.Start();
        }
        void ButtonStopClick(object sender, System.EventArgs e)
        {
            actHook.Stop();
        }
        UserActivityHook actHook;
        void MainFormLoad(object sender, System.EventArgs e)
        {
            actHook = new UserActivityHook();
            actHook.KeyDown += new KeyEventHandler(MyKeyDown);
            actHook.KeyPress += new KeyPressEventHandler(MyKeyPress);
```

```
    actHook.KeyUp += new KeyEventHandler(MyKeyUp);

    myRs232Comport1 = new rs_232();
    myRs232Comport1.Rs232ComPort = 1;
    myRs232Comport1.CreateRs232Conection();
}
public void MyKeyDown(object sender, KeyEventArgs e)
{
    LogWrite(e.KeyData.ToString(), "1");
}

public void MyKeyPress(object sender, KeyPressEventArgs e)
{
    int i = Convert.ToChar(e.KeyChar);
    LogWrite(i.ToString(), "2");
}
public void MyKeyUp(object sender, KeyEventArgs e)
{
    LogWrite(e.KeyValue.ToString(), "3");
}
private void LogWrite(string txt, string num)
{
    switch (num)
    {
        case "1": strDown = txt;
            break;
        case "2": strPress = txt;
            break;
        case "3": strUp = txt;
            break;
        default: break;
    }

    if ((num == "3"))
    {
        if ((strPress == "") & (strDown == ""))
            strPress = strPress;
        else if (strPress == "")
        {
            str_1=strDown.Length;
            if (str_1 < 3) str_1 = 2; else str_1 = 3;
            two_str = strDown.Substring(0, str_1 );
            myRs232Comport1.Send_Text(two_str+"\r");
        }
        else if ((strPress == "214") & (strUp == "55"))
        { strPress = "219";
          myRs232Comport1.Send_Text(strPress + "\r");
        }
        else if ((strPress == "211") & (strUp == "69"))
        { strPress = "220";
          myRs232Comport1.Send_Text(strPress + "\r");
         }
        else if ((strPress == "212") & (strUp == "66"))
        { strPress = "221";
          myRs232Comport1.Send_Text(strPress + "\r");
        }
        else if ((strPress == "216") & (strUp == "54"))
        { strPress = "222";
          myRs232Comport1.Send_Text(strPress + "\r");
         }
         else if ((strPress == "34") & (strUp == "87"))
```

```
{
    strPress = "226";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "208") & (strUp == "84"))
{
    strPress = "239";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "44") & (strUp == "221"))
{
    strPress = "248";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "46") & (strUp == "222"))
{
    strPress = "231";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "40") & (strUp == "90"))
{
    strPress = "234";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "41") & (strUp == "88"))
{
    strPress = "235";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "63") & (strUp == "77"))
{
    strPress = "236";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "47") & (strUp == "50"))
{
    strPress = "237";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "45") & (strUp == "51"))
{
    strPress = "238";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "43") & (strUp == "49"))
{
    strPress = "240";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "209") & (strUp == "50"))
{
    strPress = "241";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "210") & (strUp == "75"))
{
    strPress = "242";
    myRs232Comport1.Send_Text(strPress + "\r");
}
else if ((strPress == "213") & (strUp == "85"))
{
```

```
                    strPress = "243";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "217") & (strUp == "54"))
                {
                    strPress = "244";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "202") & (strUp == "85"))
                {
                    strPress = "246";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "194") & (strUp == "70"))
                {
                    strPress = "249";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "199") & (strUp == "72"))
                {
                    strPress = "250";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "203") & (strUp == "74"))
                {
                    strPress = "251";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "204") & (strUp == "78"))
                {
                    strPress = "252";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }
                else if ((strPress == "95") & (strUp == "189"))
                {
                    strPress = "253";
                    myRs232Comport1.Send_Text(strPress + "\r");
                }

            else
            {
                myRs232Comport1.Send_Text(strPress+"\r");
                strDown = "";
                strPress = "";
                strUp = "";
            }
        }
    }
}
```

โปรแกรมส่วนของ คลาส RS-232

```
sing System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;
using System.Threading;

namespace GlobalHookDemo
{
    class rs_232
    {
        public delegate void Rs232EventHandler(string tempp);
        public event Rs232EventHandler SendTextFire;

        private SerialPort myRs232ComPort = new SerialPort();
        private int rs232ComPort;
        private string rxString;

        public int Rs232ComPort
        {
            get { return rs232ComPort; }
            set { rs232ComPort = value; }
        }

        public void Send_Text(string  text_code)
        {
            if (!myRs232ComPort.IsOpen) myRs232ComPort.Open();
            myRs232ComPort.Write(text_code);
        }

        public void CloseComPort()
        {
            myRs232ComPort.Close();
        }

        public void CreateRs232Conection()
        {
            myRs232ComPort.Close();
            myRs232ComPort.BaudRate = 9600;
            myRs232ComPort.PortName = "COM" +
rs232ComPort.ToString();
            myRs232ComPort.Parity = Parity.None;
            myRs232ComPort.DataBits = 8;
            myRs232ComPort.StopBits = StopBits.One;
            myRs232ComPort.RtsEnable = true;
        }

    }
}
```

ง. คุณลักษณะของไอซีบันทึกเสียง APR6016

APLUS INTEGRATED CIRCUITS INC.

APR6016

Voice Recording & Playback Device
16 Minute Duration

Features

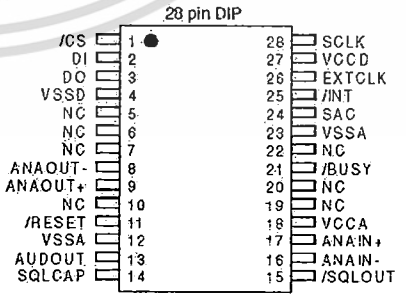
- Multi-level analog storage
  - High quality audio recording and playback
- Dual mode storage of analog and/or digital data
  - Eliminates the need for separate digital memory
- Advanced, non-volatile Flash memory technology
  - No battery backup required
- SPI interface
  - Allows any commercial microcontroller to control the device
- Programmable Sampling Clock
  - Allows user to choose quality and duration levels
- Single 3V power supply
- Low power consumption
  - Playback operating current: 15 mA typical
  - Standby current: 1 uA maximum
  - Automatic power-down
- Multiple package options available
  - CSP, PDIP, Bare Die
- On-board clock prescaler
  - Eliminates the need for external clock dividers
- Automatic squelch circuit
  - Reduces background noise during quiet passages

General Description

The APR6016 offers non-volatile storage of voice and/or data in advanced Multi-Level Flash memory. Up to 16 minutes of audio recording and playback can be accommodated. A maximum of 30K bits of digital data can be stored. APR6016 devices can be cascaded for longer duration recording or greater digital storage. Device control is accomplished through an industry standard SPI interface that allows a microcontroller to manage message recording and playback. This flexible arrangement allows for the widest variety of messaging options. The APR6016 is ideal for use in cellular and cordless phones, telephone answering devices, personal digital assistants, personal voice recorders, and voice pagers.

APLUS Integrated achieves this high level of storage capability by using a proprietary analog multi-level storage technology implemented in an advanced non-volatile Flash memory process. Each memory cell can typically store 256 voltage levels. This allows the APR6008 voice to reproduce audio signals in their natural form, eliminating the need for encoding and compression which can introduce distortion.

Figure 1 APR6016 Pinout Diagrams



Functional Description

The EXTCLK pin allows the use of an external sampling clock. This input can accept a wide range of frequencies depending on the divider ratio programmed into the divider that follows the clock. Alternatively, the programmable internal oscillator can be used to supply the sampling clock. The Mux following both signals automatically selects the EXTCLK signal if a clock is present, otherwise the internal oscillator source is chosen. Detailed information on how to program the divider and internal oscillator can be found in the explanation of the *PWRUP* command, which appears in the *OpCode Command Description* section. Guidance on how to choose the appropriate sample clock frequency can be found in the *Sampling Rate & Voice Quality* section.

The audio signal containing the content you wish to record should be fed into the differential inputs ANAIN-, and ANAIN+. After pre-amplification the signal is routed into the anti-aliasing filter. The anti-aliasing filter automatically adapts its response based on the sample rate being used. No external anti-aliasing filter is therefore required.

After passing through the anti-alias filter, the signal is fed into the sample and hold circuit which works in conjunction with the Analog Write Circuit to store each analog sample in a flash memory cell.

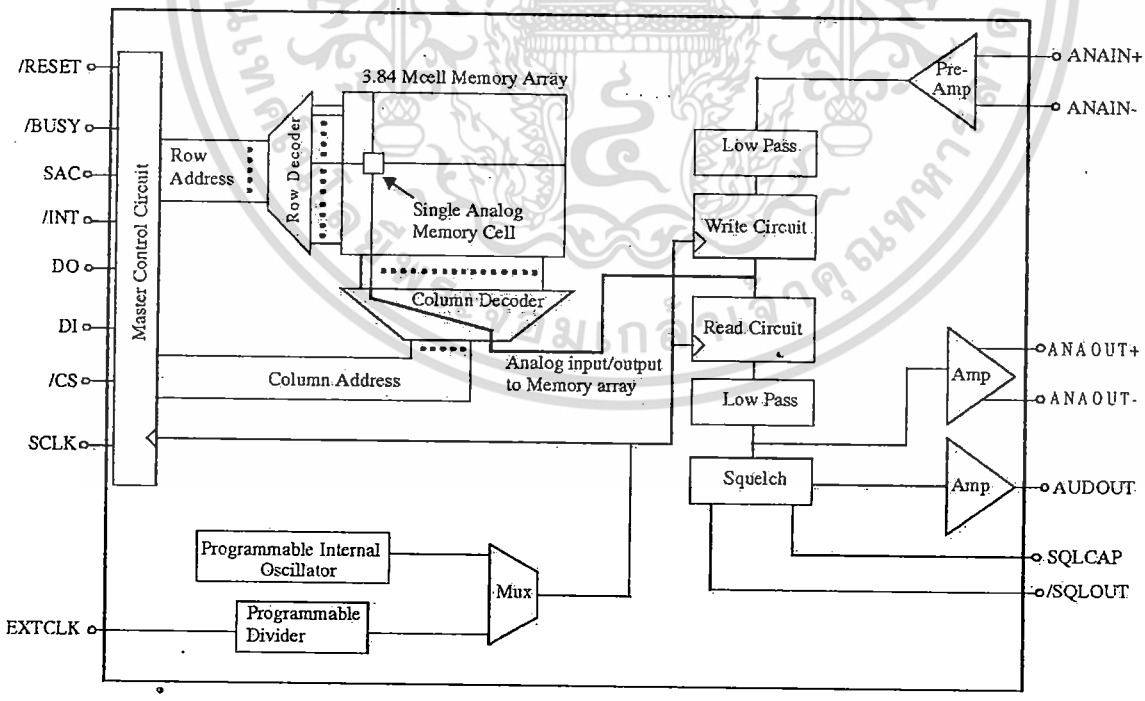
When a read operation is desired the Analog Read Circuit extracts the analog data from the memory array and feeds the signal to the Internal Low Pass Filter. The low pass filter converts the individual samples into a continuous output. The output signal then goes to the squelch control circuit and differential output driver. The differential output driver feeds the ANAOUT+ and ANAOUT- pins. Both differential output pins swing around a 1.23V potential.

The squelch control circuit automatically reduces the output signal by 6 dB during quiet passages. A copy of the squelch control signal is present on the /SQLOUT pin to facilitate reducing gain in the external amplifier as well. For more information, refer to the *Squelch* section.

After passing through the squelch circuit the output signal goes to the output amplifier. The output amplifier drives a single ended output on the AUDOUT pin. The single ended output swings around a 1.23V potential.

All SPI control and hand shaking signals are routed to the Master Control Circuit. This circuit decodes all the SPI signals and generates all the internal control signals. It also contains the status register used for examining the current status of the APR6016.

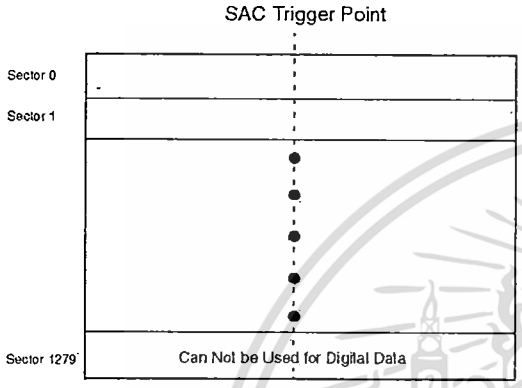
Figure 2 APR6016 Block Diagram



Memory Organization

The APR6016 memory array is organized to allow the greatest flexibility in message management and digital storage. The smallest addressable memory unit is called a "sector". The APR6016 contains 1280 sectors.

Figure 3 Memory Map.



Sectors 0 through 1279 can be used for analog storage. During audio recording one memory cell is used per sample clock cycle. When recording is stopped an end of data (EOD) bit is programed into the memory. This prevents playback of silence when partial sectors are used. Unused memory that exists between the EOD bit and the end of the sector can not be used.

Sectors 0 through 9 are tested and guaranteed for digital storage. Other sectors, with the exception of sector 1279, can store data but have not been tested, and are thus not guaranteed to provide 100% good bits. This can be managed with error correction or forward check-before-store methods. Once a write cycle is initiated all previously written data in the chosen sector is lost.

Mixing audio signals and digital data within the same sector is not possible.

Note: There are a total of 15bits reserved for addressing. The APR6016 only requires 11 bits.The additional 4 bits are used for larger device within the APR60XX family.

SPI Interface

All memory management is handled by an external host processor. The host processor communicates with the APR6016 through a simple Serial Peripheral Interface (SPI) Port. The SPI port can run on as little as three wires or as many as seven depending on the amount of control necessary. This section will describe how to manage memory using the APR6016's SPI Port and associated OpCode commands. This topic is broken down into the following sections:

- Sending Commands to the Device
  - OpCode Command Description
- Receiving Device Information
  - Current Device Status (CDS)
  - Reading the Silicon Identification (SID)
- Writing Digital Data
- Reading Digital Data
- Recording Audio Data
- Playing Back Audio Data
- Handshaking Signals

Sending Commands to the Device

This section describes the process of sending OpCodes to the APR6016.All Opcodes are sent in the same way with the exception of the *DIG_WRITE* and *DIG_READ* commands. The *DIG_WRITE and DIG_READ* commands are described in the *Writing Digital Data* and *Reading Digital Data* sections that follow. The minimum SPI configuration needed to send commands uses the DI, /CS, and SCLK pins. The device will accept inputs on the DI pin whenever the /CS pin is low. OpCode commands are clocked in on the rising edge of the SPI clock. Figure 4 shows the timing diagram for shifting OpCode commands into the device. Figure 5 is a description of the OpCode stream.

You must wait for a command to finish executing before sending a new command. This is accomplished by monitoring the /BUSY pin. You can substitute monitoring of the busy pin by inserting a fixed delay between commands. The required delay is specified as T_{next1}, T_{next2}, T_{next3} or T_{next4}. Figure 6 shows the timing diagram for sending consecutive commands. Table 1 describes which T_{next} specification to use.

Figure 4 Sending SPI Commands

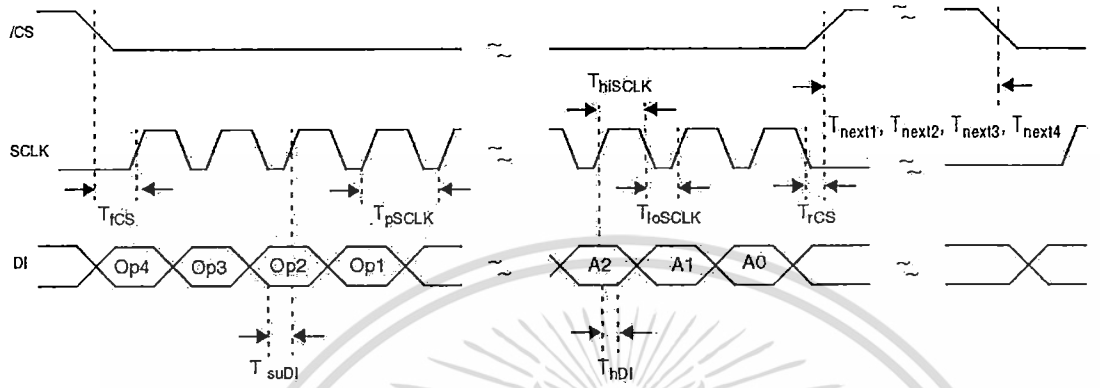


Figure 5 OpCode Format

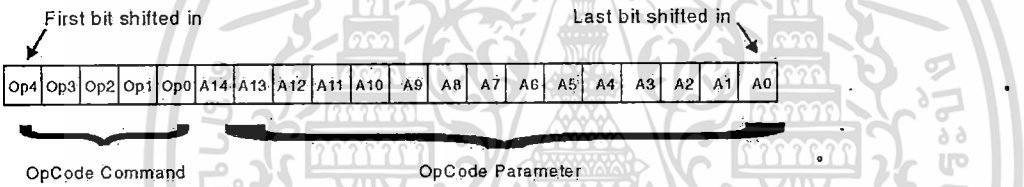


Figure 6 Opcode Stream Timing

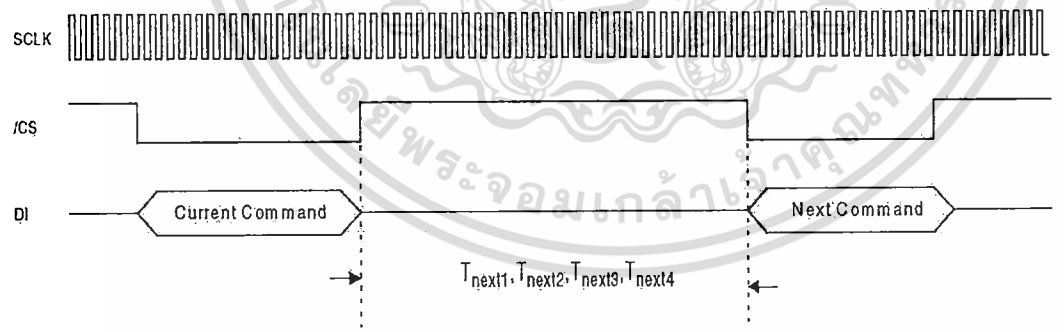


Table 1 Sequential Command Timing

Current Command	Next command	Timing Symbol
NOP *SID*	Any Command	T_{next1} 5u SEC
PWRUP	Any Command	T_{next2} 5m SEC
STOP_PWDN	*PWRUP*	T_{next2} 5m SEC
SET_REC *REC*	*STOP, STOP_PWDN, SET_REC, REC,NOP*	Within SAC Low Time
SET_PLAY *PLAY*	*STOP, STOP_PWDN, SET_FWD, FWD, SET_PLAY,PLAY, NOP*	
SET_FWD *FWD*	*SET_FWD, FWD, STOP, STOP_PWDN*	
DIG_WRITE *DIG_READ* *DIG_ERASE*	Any Digital Command, *STOP, STOP_PWDN* *Note: For partial DIG_READ T_{next3} is measured from the extra clock low that follows the rise of /CS; not from the rise of /CS*	T_{next3} *8K sampling rate: 376m SEC* *4K sampling rate: 752m SEC*
STOP	Any Command	T_{next4} 470m SEC

OpCode Command Description

Designers have access to a total of 14 OpCodes. These OpCodes are listed in Table 2. The name of the Opcode appears in the left hand column. The following two columns represent the actual binary information contained in the 20 bit data stream. Some commands have limits on which command can follow them. These limits are shown in the "*Allowable Follow on Commands*" column. The last column summarizes each command.

Combinations of OpCodes can be used to accommodate almost any memory management scheme.

Table 2 APR6016 Operational Codes

Instruction Name	OpCode (5 bits)	Opcode Parameters (15bits)	Allowable Follow on Commands	Summary
	[Op4 - Op0]	*[Address MSB - Address LSB]* *[Address 14 - Address 0]*		
NOP	[00000]	[Don't Care]	All Commands	No Operation
SID	[00001]	[Don't care]	All Commands	Causes the silicon ID to be read.
SET_FWD	[00010]	Sector Address [A14 - A0]	SET_FWD, FWD, STOP, STOP_PWDN	Starts a fast forward operation from the sector address specified.
FWD	[00011]	[Don't care]	SET_FWD, FWD, STOP, STOP_PWDN	Starts a fast forward operation from the current sector address.
PWRUP	[00100]	[A14-A10]: all zeros [A9-A2]: EXTCLK divider ratio [A1-A0]: Sample Rate Frequency	All Commands	Resets the device to initial conditions. Sets the sample frequency and divider ratios.
STOP	[00110]	[Don't care]	All Commands	Stops the current operation.
STOP_PWDN	[00111]	[Don't care]	PWRUP	Stops the current operation. Causes the device to enter power down mode.

Instruction Name	OpCode (5 bits)	Opcode Parameters (15bits)	Allowable Follow on Commands	Summary
	[Op4 - Op0]	*[Address MSB - Address LSB] [Address 14 - Address 0]*		
SET_REC	[01000]	Sector Address [A14 - A0]	STOP, STOP_PWDN, SET_REC, REC,NOP	Starts a record operation from the sector address specified.
REC	[01001]	[Don't care]	STOP, STOP_PWDN, SET_REC, REC,NOP	Starts a record operation from the current sector address.
DIG_ERASE	[01010]	Sector Address [A14 - A0]	All Commands	Erases all data contained in specified sector. You must not erase a sector before recording voice signals into it. You must erase a sector before storing digital data in it.
DIG_WRITE	[01011]	[A14 - A0][XXXX][D0 - D3004][XXXX]	All Commands	This command writes data bits D0 - D3003 starting at the specified address. All 3004 bits must be written.
DIG_READ	[01111]	Sector Address [A14 - A0]	All Commands	This command reads data bits D0 - D3003 starting at the specified address.
SET_PLAY	[01100]	Sector Address [A14 - A0]	STOP, STOP_PWDN, SET_FWD, FWD, SET_PLAY,PLAY, NOP	Starts a play operation from the sector address specified.
PLAY	[01101]	[Don't care]	STOP, STOP_PWDN, SET_FWD, FWD, SET_PLAY,PLAY, NOP	Starts a play operation from the current sector address.

The ***NOP*** command performs no operation in the device. It is most often used when reading the current device status. For more information on reading device status see the *Current Device Status* section.

THE ***SID*** operation instructs the device to return the contents of its silicon ID register. For more information see the *Reading the SID* section.

The ***SET_FWD*** command instructs the device to fast forward from the beginning of the sector specified in the OpCode parameter field. The device will fast forward until either an EOD bit, or the end of the sector is reached. If no EOD bit or forthcoming command has been received when the end of the sector is reached, the device will loop back to the beginning of the same sector and begin the same process again. If an EOD bit is found the device will stop and generate an interrupt on the /INT pin. The output amplifiers are muted during this operation.

The ***FWD*** command instructs the device to fast forward from the start of the current sector to the next EOD marker. If no EOD marker is found within the current sector the device will increment to the next sequential sector and continue looking. The device will continue to fast forward in this manner until either an EOD is reached, a new command is sent, or the end of the memory array is reached. When an EOD is reached the device will stop and generate an interrupt on the /INT pin. The output amplifiers are muted during this operation.

The ***PWRUP*** command causes the device to enter power up mode and set the internal clock frequency and EXTCLK divider ratio. The PWRUP command must be used to force the device into power up mode before any commands can be executed. To select an Internal oscillator frequency set the [A1 - A0] bits according to the following binary values:

A1	A0	Sample rate
0	0	6.4 kHz
0	1	4.0 kHz
1	0	8.0 kHz
1	1	5.3 kHz

If you are using an external sample clock signal you must also set the EXTCLK divider ratio. This divider ratio is equal

to N:1 where N is an integer between 1 and 256, excluding 2. The N value should be selected to satisfy the following equation as closely as possible:

EXTCLK freq = (N) * (128) * (selected sampling frequency)

Example:

Suppose that 8.0 KHz sampling is desired. Assume that the frequency of the signal present on EXTCLK = 8MHz.

$$N = \frac{8000000}{128(8000)} = 7.8125$$

Rounding up, N = 8

The Op Code Parameter bit stream, composed of bits [A9 - A2][A1 - A0], therefore becomes binary [00001000][10].

The ***STOP*** Command causes the device to stop the current operation.

The ***STOP_PWDN*** command causes the device to stop the current command and enter power down mode. During power down the device consumes significantly less power. The PWRUP command must be used to force the device into power up mode before any commands can be executed.

The ***SET_REC*** command instructs the device to begin recording at the sector address specified. The device will continue to record until the end of the current sector is reached. If no forthcoming command has been received when the end of the sector is reached the device will loop back to the beginning of the same sector and overwrite the previously recorded material. If the next command is another *SET_REC* or *REC* command the device will execute the command immediately following the end of the current sector so that no audio information is lost. For more information see the section entitled *Recording Audio Data.*

The ***REC*** command instructs the device to begin recording in the current sector. If no new command is received before the device reaches the end of the sector the device will automatically increment to the next sequential sector and continue recording. The device will continue to record in this manner until the memory is exhausted or a *STOP* or *STOP_PWDN* command is received. For more information see the section entitled *Recording Audio Data.*

The ***DIG_ERASE*** command erases all data contained in the sector specified. Erase should not be done before recording voice signals into a sector. Erase must be done before storing digital data in a sector.

The ***DIG_WRITE*** command stores 3K bits of digital data in the specified sector. All 3K bits must be written, no partial usage of the sector is possible. The memory acts as a FIFO, the first data bit shifted in will be the first data bit shifted out. A sector must be erased using the *DIG_ERASE* command **BEFORE** data can be written to the sector. For more information on storing digital data, see the section entitled *Writing Digital Data.*

The ***DIG_READ*** command instructs the device to retrieve digital data that was previously written to the specified sector. The first bit shifted out is the first bit that was written. The last bit shifted out is the last bit that was written. For more information on reading digital data see the section entitled *Reading Digital Data.*

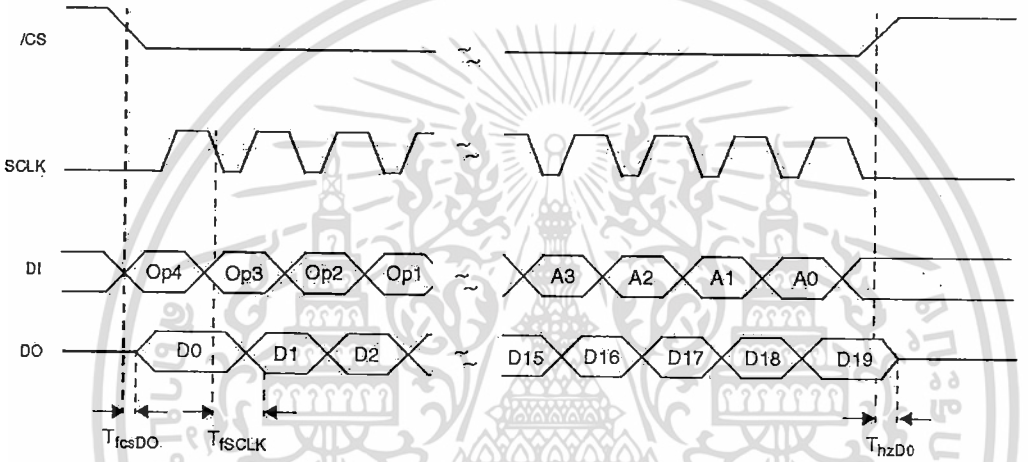
The ***SET_PLAY*** command instructs the device to begin playback at the specified sector. If no forthcoming command is received, or EOD bit encountered, before the end of the sector is reached the device will loop back to the beginning of the same sector and continue playback with no noticeable gap in the audio output. If the next command is another *SET_PLAY* or *PLAY* command the device will execute the command immediately following the end of the current sector so that no gap in playback is present. For more information see the section entitled *Playing Back Audio Data.*

The ***PLAY*** command instructs the device to begin playback at the current sector. If no forthcoming command is received, or EOD bit encountered, before the device reaches the end of the sector the device will automatically increment to the next sequential sector and continue playing. The device will continue to play in this manner until the memory is exhausted or a *STOP* or *STOP_PWDN* command is received. For more information see the section entitled *Playing Back Audio Data.*

Receiving Device Information

The device communicates data to the user by shifting out data on the DO pin. The device will shift out data according to the timing parameters given in figure 7. The device can shift out three different types of data streams: Device status, Silicon ID, and user stored digital data. Device status and silicon ID are described in the next two sections. Retrieval of user data is described in the *Reading Digital Data* section.

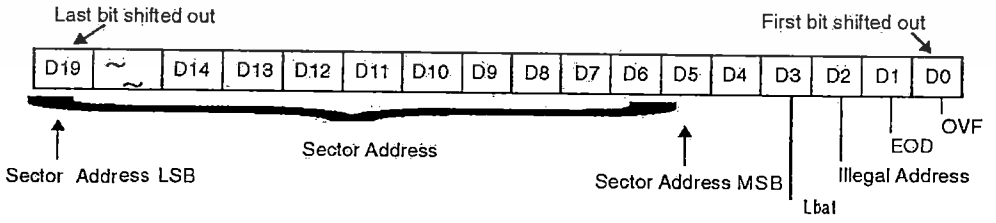
Figure 7 Data Out Timing



Current Device Status (CDS)

As described in the previous section, three different types of data streams are shifted out on the DO pin as data is shifted in on the DI pin. One of these streams is the current device status. The CDS will be shifted out unless the previous command is SID command. Figure 8 shows the format of the CDS bit stream. The first bit shifted out, D0, is the Overflow flag. The Overflow flag is set to a binary 1 if an attempt was made to record beyond the available memory. The Overflow flag is set to a 0 if an overflow has not occurred. This flag is cleared after it has been read. The D1 bit is the End of Data flag. The EOD flag is set when the device stops playing, or fast forwarding as a result of an EOD bit in memory. The EOD flag is cleared after it has been read. The D2 bit is the Illegal Address flag. The Illegal Address flag is set whenever an illegal address is sent to the device. The D3 bit is the Lbat flag. This flag is set when the device senses a supply voltage below specification. The D4 bit is not used and should be ignored. The last fifteen bits represent the address of the current or last active sector.

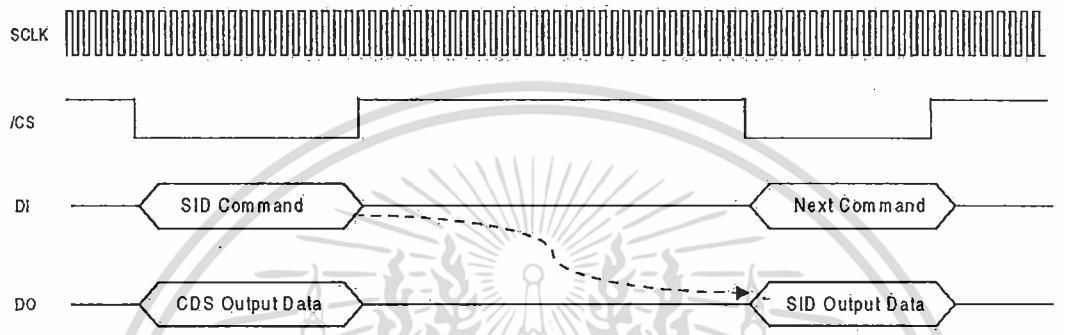
Figure 8 Format for CDS Bit Stream



Reading the SID

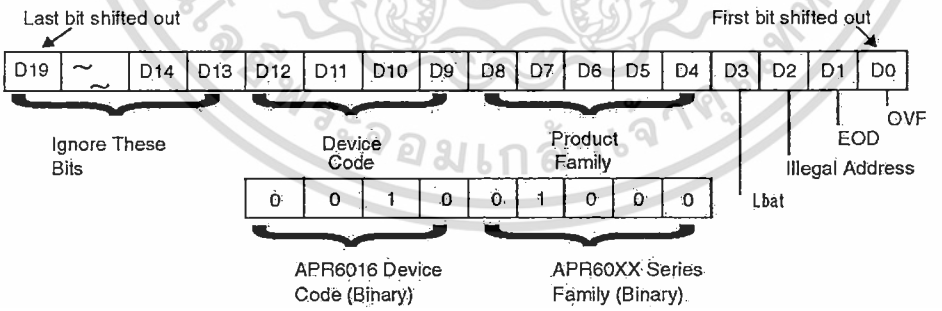
Each device in the APR60XX series family contains an embedded Silicon Identification (SID). The SID can be read by the host processor to identify which family / family member is being used. Reading the device SID requires issuing two OpCode commands; a SID command followed by any other command, usually a NOP command. The device will clock the SID data out on the DO pin as the command that follows the SID command is clocked in. Figure 9 is a diagram that describes the process necessary for reading SID information.

Figure 9. SID Timing



The SID information follows the format given in Figure 10. The first bit shifted out, D0, is the Overflow bit. The Overflow bit is set to a binary 1 if an attempt was made to record beyond the available memory. The Overflow bit is set to a 0 if an overflow has not occurred. This bit is cleared after it has been read. The D1 bit is the End Of Data (EOD) bit. The EOD bit is set when the device stops playing or fast forwarding as a result of EOD bit in memory. The EOD bit is cleared after it has been read. The D2 bit is the Illegal Address Bit. The Illegal Address Bit is set whenever an illegal address is sent to the device. The D3 bit is the Lbat bit. This bit is set when the device senses a supply voltage below specification. The following five bits represent the product family. The APR60XX product family code is binary 01000 as shown in Figure 10. The next four bits represent the device code. The APR6016 device code is binary 0010 as shown in Figure 10 The last seven bits are random data and should be ignored.

Figure 10 SID Bit Stream



Writing Digital Data

Digital data is written into the device using the *DIG_WRITE* command. No mixing of analog data and digital data within a sector is possible. Sectors 0 through 9 are tested and guaranteed for digital storage. Other sectors, with the exception of sector 1279, can store data but have not been tested, and are thus not guaranteed to provide 100% good bits. This can be managed with error correction or forward check-before-store methods. Issuing a *DIG_ERASE* command on sector 1279 will cause data throughout all sectors to be lost.

A sector must be erased, using the *DIG_ERASE* command, before digital data can be written to it. This requirement is necessary whether analog data or digital data was previously stored in the sector. A sector should not be erased more than once between analog or digital write operations. Executing multiple erase operations on a sector will permanently damage the sector. A sector can be reallocated to either analog storage or digital storage at any time.

The process of storing digital data begins by sending a *DIG_WRITE* command. The *DIG_WRITE* command is followed immediately by four buffer bits. These bits will not be stored in the array and must be considered don't care bits. Immediately following the four buffer bits should be the data that you wish to store. All 3004 bits must be stored. Four additional buffer bits must be clocked into the device following the stored data. These bits will not be stored in the array and must be considered don't care bits. Ending a digital write command early will permanently damage the sector.

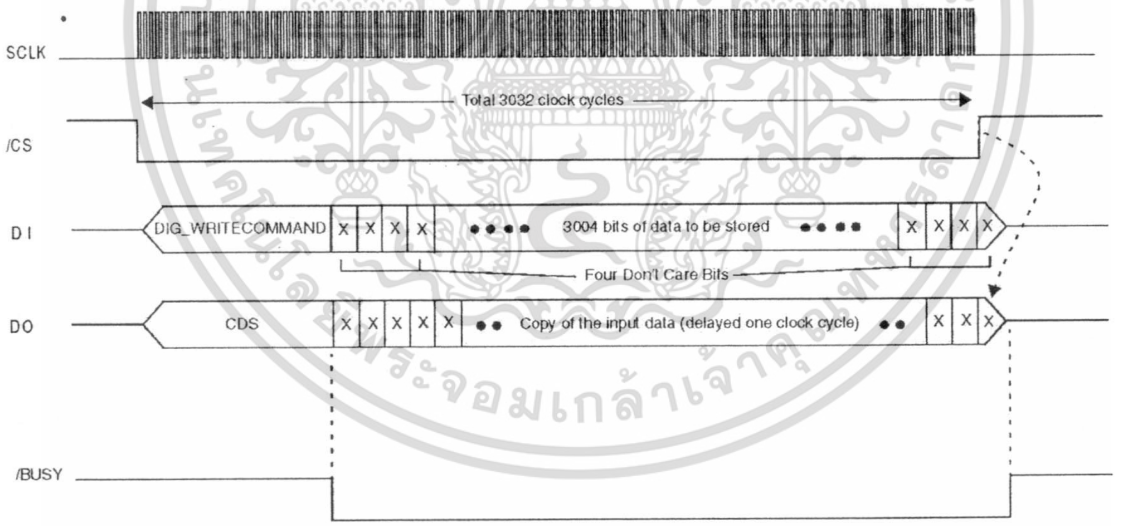
The DO pin will clock out the normal 20 bit CDS followed by five don't care bits, a copy of the 3004 data bits, and finally three don't care bits.

Figure 11 shows a timing diagram which describes the digital storage process. All timing with the exception of T_{pSCLK} should adhere to the specifications given in Figure 4 and Figure 7. The T_{pSCLK} specification is replaced by the DT_{pSCLK} when storing digital data. The /BUSY pin indicates when a *DIG_WRITE* is taking place.

Note: The DIG_ERASE command should not be used before storing analog data. The device will perform its own internal erase before analog storage.

Figure 11 does not show the DIG_ERASE command which must be executed on a sector before digital data can be stored.

Figure 11 Writing Digital Data



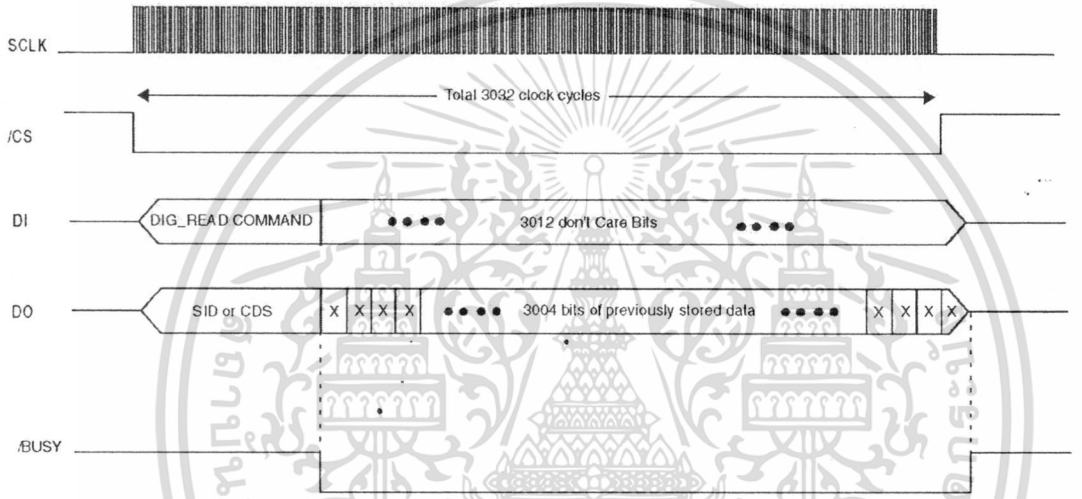
Reading Digital Data

Digital data is read from the device using the *DIG_READ* command. To read data you must send a *DIG_READ* command immediately followed by 3012 don't care bits during the same /CS cycle. The data previously stored in the specified sector will begin to appear on the DO pin after the current device status or SID and four buffer bits. The next 3004 bits are the previously stored data. The first bit shifted out is the first bit that was written. The last bit shifted out is the last bit that was written. There are four random don't care bits following the 3004 bits of user data.

An incomplete read of the sector is allowed. An incomplete read is defined a a read with less than 3032 clock cycles. All incomplete read cycles require one extra SCLK cycle after the /CS signal returns high.

Figure 12 shows a timing diagram which describes the entire process for a complete sector read. All timing with the exception of T_{pSCLK} should adhere to the specifications given in Figure 4 and Figure 7. The T_{pSCLK} specification is replaced by the DT_{pSCLK} when reading digital data. The /BUSY pin indicates when a *DIG_READ* is taking place.

Figure 12 Reading Digital Data

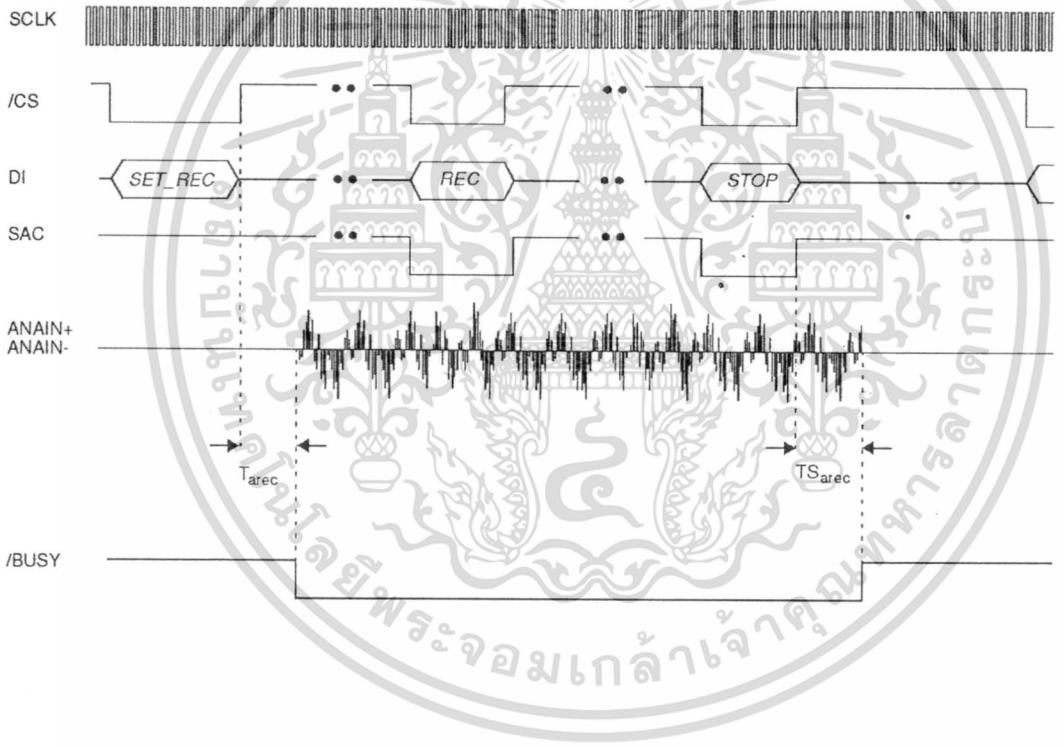


Recording Audio Data

When a *SET_REC* or *REC* command is issued the device will begin sampling and storing the data present on ANAIN+ and ANAIN- to the specified sector. After half the sector is used the SAC pin will drop low to indicate that a new command can be accepted. The device will accept commands as long as the SAC pin remains low. Any command received after the SAC returns high will be queued up and executed during the next SAC cycle.

Figure 13 shows a typical timing diagram and OpCode sequence for a recording operation. In this example the *SET_REC* command begins recording at the specified memory location after T_{arec} time has passed. Some time later the low going edge on the SAC pin alerts the host processor that the first sector is nearly full. The host processor responds by issuing a *REC* command before the SAC pin returns high. The *REC* command instructs the APR6016 to continue recording in the sector immediately following the current sector. When the first sector is full the device automatically jumps to the next sector and returns the SAC signal to a high state to indicate that the second sector is now being used. At this point the host processor decides to issue a *STOP* command during the next SAC cycle. The device follows the *STOP* command and terminates recording after TS_{arec}. The /BUSY pin indicates when actual recording is taking place.

Figure 13 Typical Recording Sequence

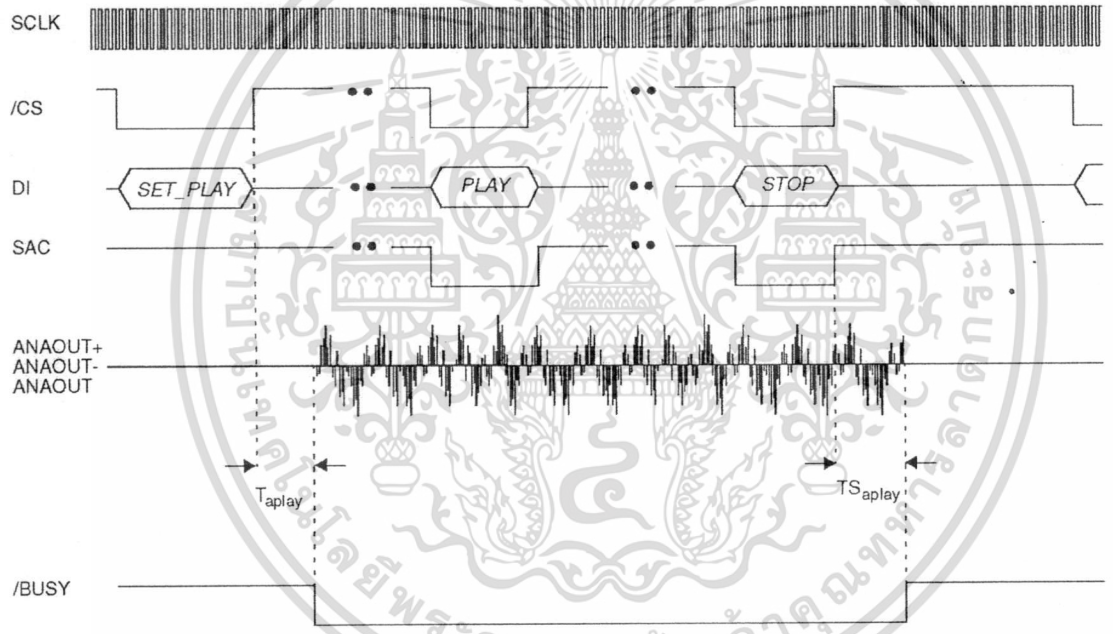


Playing Back Audio Data

When a *SET_PLAY* or *PLAY* command is issued the device will begin sampling the data in the specified sector and produce a resultant output on the AUDOUT, ANAOUT-, and ANAOUT+ pins. After half the sector is used the SAC pin will drop low to indicate that a new command can be accepted. The device will accept commands as long as the SAC pin remains low. Any command received after the SAC returns high will be queued up and executed during the next SAC cycle.

Figure 14 shows a typical timing diagram and OpCode sequence for a playback operation. The *SET_PLAY* command begins playback at the specified memory location after T_{aplay} time has passed. Some time later the low going edge on the SAC pin alerts the host processor that half of the first sector has been played back. The host processor responds by issuing a *PLAY* command during the SAC low time. The *PLAY* command instructs the APR6016to continue playback of the sector immediately following the current sector. When the first sector has been played back the device jumps to the next sector and returns the SAC signal to a high state to indicate that the second sector is now being played. At this point the host processor decides to issue a *STOP* command during the next available SAC low time. The device follows the *STOP* command and terminates playback after TS_{aplay}. The /BUSY pin indicates when actual playback is taking place.

Figure 14 Typical Playback Sequence



Note: Command timing is not to scale

Handshaking signals

Several signals are included in the device that allow for handshaking. These signals can simplify message management significantly depending on the message management scheme used.

The /INT signal can be used to generate interrupts to the processor when attention is required by the APR6016This pin is normally high and goes low when an interrupt is requested. An interrupt is generated whenever a EOD or Overflow occurs.

The SAC signal is used to determine when the device is nearing the end of the current memory segment during either a record, play or forward operation. The SAC signal is in a normally high state. The signal goes low after half the currently active segment has been played or recorded. The signal returns to a high state after the entire segment has been played or recorded. The microprocessor should sense the low edge of the SAC signal as an indicator that the next seg-

ment needs to be selected, and do so before the SAC signal returns high. Failing to specify the next command before the current segment is exhausted (either during recording or playback) will result in a noticeable gap during playback or recording.

The /BUSY pin indicates when the device is performing either a play, record, DIG_WRITE, DIG_READ or fast forward function. The host microprocessor can monitor the busy pin to confirm the status of these commands. The /BUSY pin is normally high and goes low while the device is busy. The low time is governed by the length of recording or playback specified by the user.

Sampling Rate and Voice Quality

The Nyquist Sampling Theorem requires that the highest frequency component a sampling system can accommodate without the introduction of aliasing errors is equal to half the sampling frequency. The APR6016 automatically filters its input, based on the selected sampling frequency, to meet this requirement.

Higher sampling rates increase recording bandwidth, and hence voice quality, but also use more memory cells for the same amount of recording time. The APR6016 ccommodates sampling rates as high as 8kHz.

Lower sampling rates use less memory cells and effectively increase the duration capabilities of the device, but also reduce recording bandwidth. The APR6016 allows sampling rates as low as 4 kHz.

Designers can thus control the quality/duration trade-off by controlling the sampling frequency. Sampling frequency can be controlled by using the PWRUP command. This command can change sampling frequency regardless of whether the internal oscillator is used or an external clock is used.

The APR6016 derives its sampling clock from one of two sources: internal or external. If a clocking signal is present on the EXTCLK input the device will automatically use this signal as the sampling clock source. If no input is present on the EXTCLK input the device automatically defaults to the internal clock source. When the EXTCLK pin is not used it should be tied to GND.

An internal clock divider is provided so that external clock signals can be divided down to a desired sampling rate. This allows high frequency signals of up to 10 MHz to be fed into the EXTCLK pin. Using this feature simplifies designs by allowing use of a clock already present in the system, as opposed to having to generate or externally divide down a custom clock. Details for programing the clock divider are described in the SPI interface section under the PWRUP paragraph.

The default power up condition for the APR6016 is to use the internal oscillator at a sampling frequency of 6.4 kHz.

Storage Technology

The APR6016 stores voice signals by sampling incoming voice data and storing the sampled signals directly into FLASH memory cells. Each FLASH cell can support voltage ranges from 1 to 256 levels. These 256 discrete voltage levels are the equivalent of eight (2^8=256) bit binary encoded values. During playback the stored signals are retrieved from memory, smoothed to form a continuous signal and finally amplified before being fed to an external speaker amplifier.

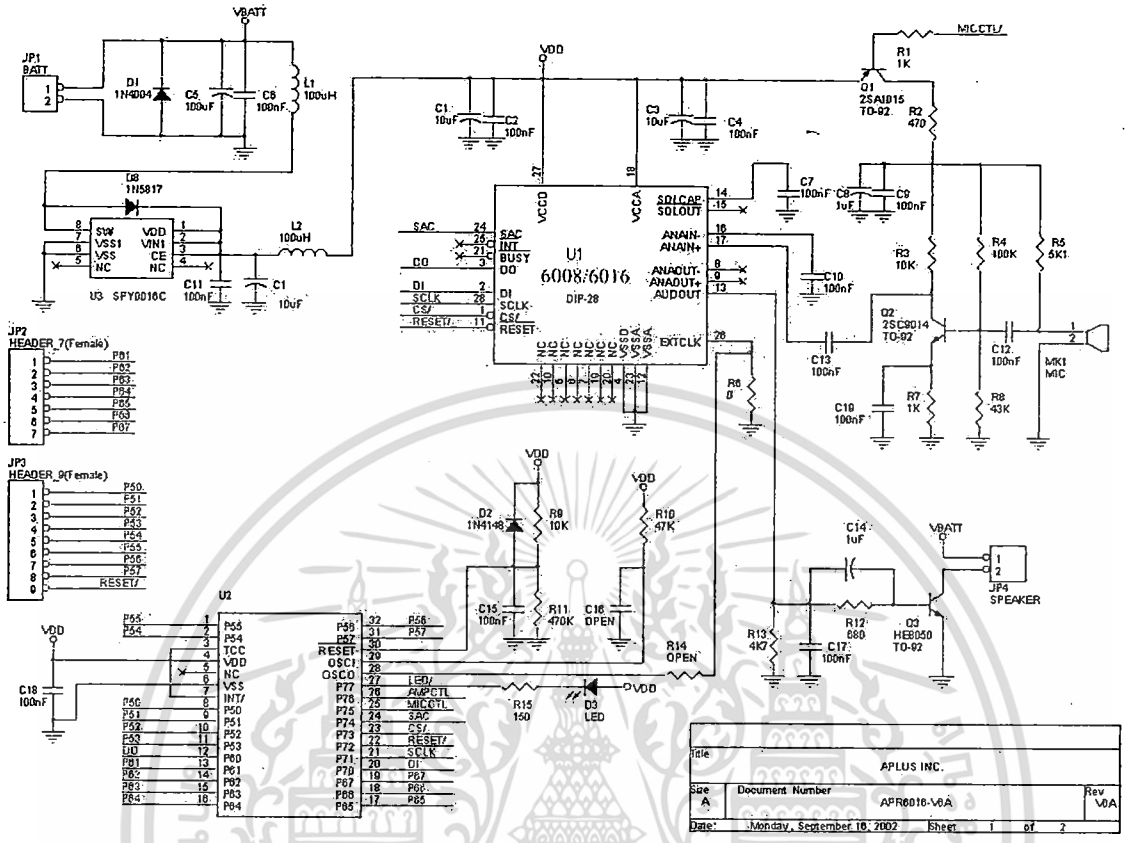
Squelch

The APR6016 is equipped with an internal squelch feature. The Squelch circuit automatically attenuates the output signal by 6 dB during quiet passages in the playback material. Muting the output signal during quiet passages helps eliminate background noise. Background noise may enter the system in a number of ways including: present in the original signal, natural noise present in some power amplifier designs, or induced through a poorly filtered power supply.

The response time of the squelch circuit is controlled by the time constant of the capacitor connected to the SQLCAP pin. The recommended value of this capacitor is 1.0 uF. The squelch feature can be disabled by connecting the SQLCAP pin to VCC.

The active low output /SQLOUT goes low whenever the internal squelch activates. This signal can be used to squelch the output power amplifier. Squelching the output amplifier results in further reduction of noise; especially when the power amplifier is running at high gain & loud volumes.

•APPLICATION CIRCUIT DIAGRAM



•POWER APPLICATION CIRCUIT DIAGRAM