มิดี้การ์ด

MIDI CARD

โดย

นายทิพากร โสภา

นายยรรยง อิ่มเอมพร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาตรีต่อเนื่อง (อส.บ.)

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

สาขาวิชาเทคโนโลยีคอมพิวเตอร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

ปีการศึกษา 2535

มิดี้การ์ด

MIDI CARD

โดย

นายทินากร โสภา 34162157

นายยรรยง อิ่มเอมพร 34162165

อาจารย์ที่ปรึกษา

อาจารย์ เกษตร์ ศิริสันติสัมฤทธิ์

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

สาขาวิชาเทคโนโลยีคอมพิวเตอร์อุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง มิดี้การ์ด (MIDI CARD)

ผู้จัดทำ

1. นายทิพากร โสภา
2. นายยรรยง อิ่มเอมพร

.....อาจารย์ที่ปรึกษา

(อาจารย์ เกษตร์ ศิริสันติสัมฤทธิ์)

มิดี้การ์ด

นายทิฆากร โสภา

นายธรรธง อิ่มเอมพร

อ. เกษตร์ ศิริสันติสัมฤทธิ์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2535

บทคัดย่อ

การนำคอมพิวเตอร์มาใช้ในงานดนตรี เริ่มขึ้นเมื่อเครื่องดนตรีเปลี่ยนจากการใช้สัญญาณอนาลอกมาเป็นสัญญาณดิจิตอล และได้มีการกำหนดมาตรฐานทางการสื่อสารระหว่างเครื่องดนตรีอิเล็กทรอนิกส์ ที่เรียกว่า "มิดี้" (MIDI - Musical Instrument Digital Interface) งานวิจัยชิ้นนี้ ได้ทำการศึกษาเกี่ยวกับมิดี้ และได้มีการสร้างการ์ดที่สามารถเปลี่ยนข้อมูลที่เป็นข้อมูลของเครื่องคอมพิวเตอร์ให้เป็นข้อมูลที่เป็นมาตรฐานของมิดี้ ส่งออกไปทางสายสัญญาณมิดี้ พร้อมทั้งสามารถแปลงข้อมูลในทางกลับกันได้

MIDI CARD

TIPAKORN SOPA

YANYONG IMAEMPORN

KASET SIRISANTISUMRIT ADVISOR

1992

Abstract

The application of computer in music began when musical instrument changed from analog instrument to digital instrument.A standard "MIDI" is made for communication between differrent kind and manufacture instrument. This paper describes a development of MIDI and we use the necessary musical rules to comfort the user in placing notes. In the interface part, our card use the MIDI standard in communication with the instrument.

สารบัญ

สารบัญ (ต่อ)

เรื่อง หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมา

เครื่องดนตรี เป็นศิลปะอันประณีต บรรจง ของมนุษย์ในอันที่จะสรรสร้าง หรือถ่ายทอดเอาจินตนาการ ความรู้สึกนึกคิด ออกมาเป็นเสียงดนตรีอันไพเราะสู่ผู้ฟัง จากความประณีตบรรจงอันนี้ เป็นสื่อสร้างสรรอันหนึ่ง ที่ทำให้มนุษย์ในยุคปัจจุบันนี้ พยายามทดลอง ศึกษาค้นคว้า เพื่อหาเครื่องดนตรีชิ้นใหม่ที่สามารถถ่ายทอดอารมณ์และความรู้สึก ให้ล้ำหน้าเข้าไปอีก โดยอาศัยเทคโนโลยีที่ทันสมัย อันได้แก่ การนำเอาเทคโนโลยีของอุปกรณ์และวงจรทางด้านอิเล็กทรอนิกส์เข้ามาประยุกต์ใช้ทำเครื่องดนตรีไฟฟ้า ซึ่งได้แก่ กลองไฟฟ้า(DRUM MACHINE),KEYBOARD,SYNTHESIZER และGUITAR เป็นต้น

จากการที่มีการพัฒนาเครื่องดนตรี ที่เป็นไฟฟ้ามากมายหลายชนิดขึ้น ผนวกกับความเจริญก้าวหน้าของคอมพิวเตอร์มีมากขึ้น จึงเกิดมีแนวความคิดในอันที่จะทำให้เครื่องดนตรีไฟฟ้า ชนิดต่างๆ สามารถติดต่อสื่อสารกันได้ภายใต้มาตรฐานเดียวที่เรียกว่า "มิดี้" (MIDI - Musical Instrument Digital Interface) ซึ่งสามารถจะอธิบาย ความหมายและขอบข่ายของ มิดี้ได้ว่า มิดี้คือ มาตรฐานของการสื่อสารข้อมูลระหว่างเครื่องดนตรีไฟฟ้าที่มีไมโครโปรเซสเซอร์เป็นอุปกรณ์หลัก ซึ่งมีลักษณะของการเชื่อมต่อกัน ระหว่างอุปกรณ์หลักและเครื่องดนตรีต่างๆ โดยมีการควบคุมจากอุปกรณ์หลักผ่านไปยังเครื่องดนตรีแต่ละตัว ตามมาตรฐานของมิดี้ สามารถจะควบคุมเครื่องดนตรีไฟฟ้าได้สูงสุดถึง 16 เครื่อง

1.2 แนวความคิดในการทำโครงงาน

แนวความคิดในการทำโครงงานครั้งนี้ เกิดขึ้นจากความปรารถนาในอันที่จะทำให้เครื่องดนตรีไฟฟ้าสามารถบรรเลงบทเพลงต่างๆ ออกมาได้ โดยการอ่านบทเพลงจากโน๊ตดนตรีสากล คล้ายดังกับมนุษย์หรือนักดนตรีเป็นผู้อ่านโน๊ตดนตรีแล้วบรรเลงบทเพลงออกมา

จากแนวความคิดอันนี้ จึงเกิดคำถามขึ้นมาว่า จะทำอย่างไร ให้เครื่องดนตรีบรรเลงบทเพลงออกมาได้ โดยมีหลักการคล้ายกับมนุษย์หรือนักดนตรีบรรเลงบทเพลง ฉะนั้น เพื่อให้บรรลุวัตถุประสงค์ในแนวความคิดของโครงงาน จึงได้มีการทำโครงงานออกเป็น 2 ภาค คือ

1 ภาคแรก มีหน้าที่หลัก คือ

อ่านโน๊ตดนตรีสากลจากแผ่นกระดาษเพื่อเก็บเป็นรหัสข้อมูล หลักการทำงานในภาคแรกนี้ คล้ายกับการอ่านและการทำความเข้าใจโน๊ตของนักดนตรี เพื่อพร้อมที่จะบรรเลงบทเพลงตามโน๊ตที่อ่านมา

2 ภาคหลัง มีหน้าที่หลัก คือ

นำรหัสข้อมูลที่ได้จากภาคแรก ส่งให้เครื่องดนตรีไฟฟ้าบรรเลงบทเพลงนั้นออกมา ถ้าเปรียบกับการเล่นดนตรีของนักดนตรี ก็เป็นการนำเอาความเข้าใจจากการอ่านโน๊ตมาบรรเลง

ซึ่งในการทำโครงงานชุดนี้ กลุ่มของเรามีหน้าที่รับผิดชอบในการทำภาคหลัง คือ ทำให้เครื่องดนตรีสามารถบรรเลงบทเพลงออกมาได้เมื่อมีการป้อนรหัสข้อมูลให้ คำว่า "รหัสข้อมูล" หมายถึง รหัสมิดี้ ซึ่งจะกล่าวไว้ในบทที่ 2

1.3 ขอบเขตในการทำโครงงาน

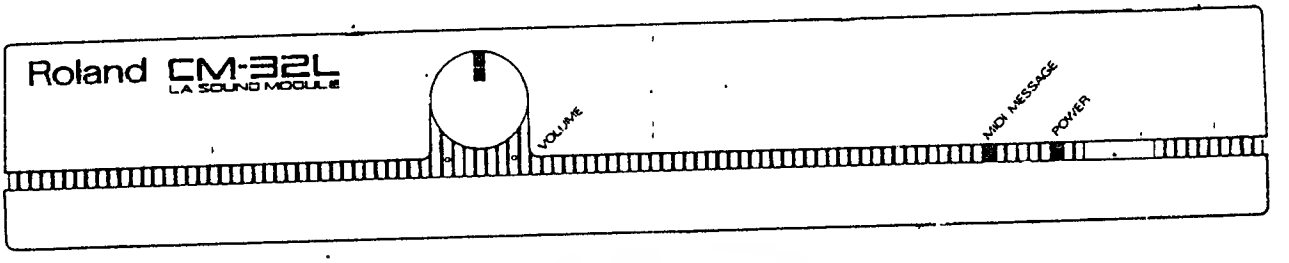
ในโครงงานภาคหลัง เป็นลักษณะของการเชื่อมต่อกันระหว่าง เครื่องไมโครคอมพิวเตอร์กับเครื่อง LA SOUND MODULE CM-32L ของบริษัท Roland โดยผ่านทาง MIDI CARD ที่สร้างขึ้นเพื่อให้ไมโครคอมพิวเตอร์กับเครื่อง LA SOUND MODULE CM-32L สามารถติดต่อสื่อสารกันได้

เนื่องจากว่า กลุ่มที่ทำโครงงานในภาคแรกยังทำไม่เสร็จ กลุ่มเราจึงทำการสมมติเอาว่า ได้มีรหัสข้อมูลที่อ่านมาจากโน๊ตดนตรีแล้ว (ทำการแปลงรหัสโดยสมาชิกในกลุ่ม) โดยทำและเก็บข้อมูลไว้เป็นไฟล์ เมื่อต้องการจะให้บรรเลงเพลงก็ทำการโหลดไฟล์เพลงที่ต้องการออกมาให้เครื่องดนตรีเล่นได้เลย

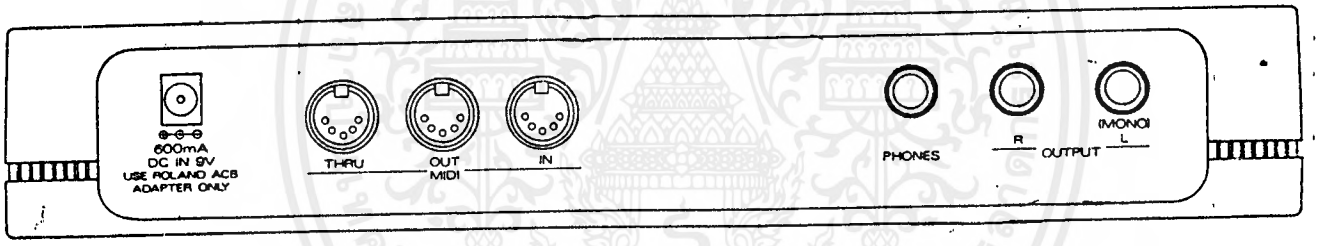
ในโปรแกรมได้มีการออกแบบให้ผู้ใช้งานสามารถเปลี่ยนเครื่องดนตรีได้สูงสุดถึง 128 ชิ้นสามารถเลือกให้เล่นเครื่องดนตรีหลายชิ้นพร้อมกันหรือให้เล่นแยกชิ้นกันก็ได้ เปลี่ยนจังหวะการเล่นให้เร็ว หรือช้าก็ได้

1.4 LA SOUND MODULE CM-32L ของบริษัท Roland

LA SOUND MODULE CM-32L ของบริษัท Roland เป็นเครื่องที่เราใช้ในการทดลอง ซึ่งสามารถอธิบายให้พอที่จะเข้าใจได้ว่า ภายในเครื่อง SOUND MODULE เปรียบเสมือนว่า ประกอบไปด้วยเครื่องซินธิไซเซอร์ (Synthesizer) 8 เครื่อง(CH2-CH9)กับอีก 1 เครื่อง Rhythm Machine (CH10) โดยเครื่องซินธิไซเซอร์นี้ สามารถที่จะสังเคราะห์เสียงธรรมชาติได้ 128 ชนิด



ก) ภาพด้านหน้า



ข) ภาพด้านหลัง

รูปที่ 1.1 ภาพด้านหน้า-ภาพด้านหลัง ของ LA SOUND MODULE CM-32L

ตารางการสังเคราะห์เสียงของ LA SOUND MODULE CM-32L

PROG#	TONE	Pt 1 #
1/00H	AcouPiano 1	4
2/01H	AcouPiano 2	2
3/02H	AcouPiano 3	1
4/03H	ElecPiano 1	3
5/04H	ElecPiano 2	2
6/05H	ElecPiano 3	2
7/06H	ElecPiano 4	1
8/07H	Honkytonk	3
9/08H	Elec Org 1	3
10/09H	Elec Org 2	3
11/0AH	Elec Org 3	2
12/0BH	Elec Org 4	2
13/0CH	Pipe Org 1	3
14/0DH	Pipe Org 2	3
15/0EH	Pipe Org 3	2
16/0FH	Accordion	2
17/10H	Harpsi 1	4
18/11H	Harpsi 2	3
19/12H	Harpsi 3	1
20/13H	Clavi 1	3
21/14H	Clavi 2	2
22/15H	Clavi 3	1
23/16H	Celesta 1	4
24/17H	Celesta 2	2
25/18H	SynBrass 1	2
26/19H	SynBrass 2	3
27/1AH	SynBrass 3	2
28/1BH	SynBrass 4	2
29/1CH	Syn Bass 1	2
30/1DH	Syn Bass 2	2
31/1EH	Syn Bass 3	2
32/1FH	Syn Bass 4	1

PROG #	TONE	Pt 1 #
33/20H	Fantasy	3
34/21H	Harmo Pan	3
35/22H	Chorale	3
36/23H	Glasses	2
37/24H	Soundtrack	4
38/25H	Atmosphere	4
39/26H	Warm Bell	4
40/27H	Funny Vox	1
41/28H	Echo Bell	3
42/29H	Ice Rain	3
43/2AH	Oboe 2001	2
44/2BH	Echo Pan	2
45/2CH	DoctorSolo	2
46/2DH	Schooldaze	2
47/2EH	Bellsinger	1
48/2FH	SquareWave	2
49/30H	Str Sect 1	4
50/31H	Str Sect 2	3
51/32H	Str Sect 3	2
52/33H	Pizzicato	3
53/34H	Violin 1	3
54/35H	Violin 2	2
55/36H	Cello 1	3
56/37H	Cello 2	2
57/38H	Contrabass	2
58/39H	Harp 1	3
59/3AH	Harp 2	2
60/3BH	Guitar 1	2
61/3CH	Guitar 2	2
62/3DH	Elec Gtr 1	4
63/3EH	Elec Gtr 2	3
64/3FH	Sitar	4

PROG #	TONE	Pt 1 #
65/40H	AcouBass 1	2
66/41H	AcouBass 2	1
67/42H	ElecBass 1	2
68/43H	ElecBass 2	1
69/44H	SlapBass 1	3
70/45H	SlapBass 2	2
71/46H	Fretless 1	4
72/47H	Fretless 2	2
73/48H	Flute 1	4
74/49H	Flute 2	2
75/4AH	Piccolo 1	3
76/4BH	Piccolo 2	2
77/4CH	Recorder	2
78/4DH	Pan Pipes	3
79/4EH	Sax 1	4
80/4FH	Sax 2	3
81/50H	Sax 3	2
82/51H	Sax 4	1
83/52H	Clarinet 1	3
84/53H	Clarinet 2	2
85/54H	Oboe	2
86/55H	Engl Horn	2
87/56H	Bassoon	2
88/57H	Harmonica	2
89/58H	Trumpet 1	3
90/59H	Trumpet 2	2
91/5AH	Trombone 1	3
92/5BH	Trombone 2	2
93/5CH	Fr Horn 1	3
94/5DH	Fr Horn 2	2
95/5EH	Tuba	2
96/5FH	Brs Sect 1	4

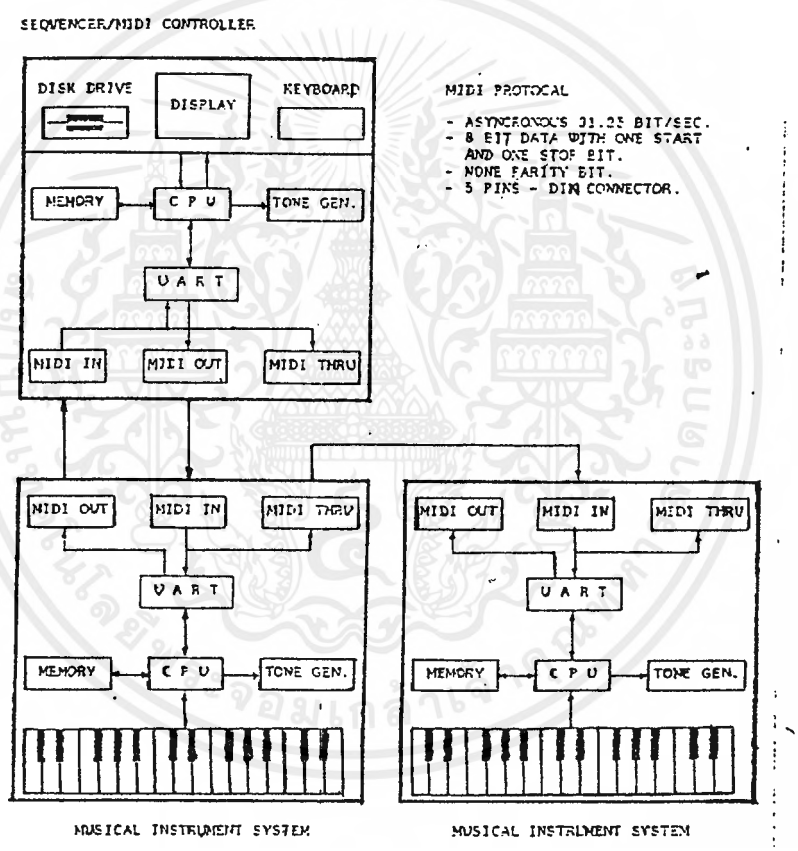
PROG #	TONE	Pt 1 #
97/60H	Brs Sect 2	3
98/61H	Vibe 1	3
99/62H	Vibe 2	2
100/63H	Syn Mallet	1
101/64H	Windbell	3
102/65H	Glock	2
103/66H	Tube Bell	4
104/67H	Xylophone	1
105/68H	Marimba	3
106/69H	Koto	2
107/6AH	Sho	4
108/6BH	Shakuhachi	4
109/6CH	Whistle 1	2
110/6DH	Whistle 2	1
111/6EH	Bottleblow	4
112/6FH	Breathpipe	3
113/70H	Timpani	2
114/71H	MelodicTom	1
115/72H	Deep Snare	2
116/73H	ElecPerc 1	2
117/74H	ElecPerc 2	2
118/75H	Taiko	3
119/76H	Taiko Rim	1
120/77H	Cymbal	2
121/78H	Castanets	2
122/79H	Triangle	2
123/7AH	Orche Hit	4
124/7BH	Telephone	1
125/7CH	Bird Tweet	4
126/7DH	OneNoteJam	4
127/7EH	WaterBells	3
128/7FH	JungleTune	4

PROG # : MIDI Program Change Number (decimal indication / hexadecimal indication).
Ptl # : The number of partials used for a sound.

บทที่ 2
ทฤษฎีของระบบมิดี้

2.1 หลักการ

มิดี้ (MIDI) ย่อมาจาก Musical Instrument Digital Interface เป็นระบบมาตรฐานสำหรับการสื่อสารข้อมูลระหว่างเครื่องดนตรี โดยจะมีการสื่อสารกันแบบอซิงโครนัส (Asynchronous) ซึ่งสามารถเขียนแผนภาพแสดงการทำงานได้ดังนี้



รูปที่ 2.1 แผนภาพแสดงโครงสร้างของเครื่องดนตรี

จากรูป อุปกรณ์ควบคุม (Controlling) เช่น คอมพิวเตอร์ สามารถควบคุมการทำงานของเครื่องดนตรีได้โดยคอมพิวเตอร์จะส่งรหัสข้อมูลมิดี้ไปยังคีย์บอร์ดตัวที่ 1 และ ใช้ออฟโต้ไอโซเลเตอร์ (Optoisolator) เป็นตัวคัปปลิ้งสัญญาณส่วน UART (Universal Synchronous Asynchronous Receiver) จะเปลี่ยนข้อมูลแบบขนานให้เป็นแบบอนุกรม (และเปลี่ยนข้อมูลแบบอนุกรมให้เป็นแบบอนุกรมด้วย)

ข้อมูลที่ได้จะเข้ามายัง ส่วนควบคุมภายในเครื่องดนตรี (Micro Controllor) เพื่อตีความแล้วส่งให้คีย์บอร์ดทำงานตามรหัสมิดี้ที่รับได้

ข้อมูลจากออฟโต้ไอโซเลเตอร์ ส่วนหนึ่งจะผ่านไปยังบัฟเฟอร์ แล้วส่งออกไปเป็นมิดี้ทรู (MIDI thru) ซึ่งก็คือการลอกแบบข้อมูลที่ได้รับมานั่นเอง ข้อมูลส่วนนี้จะถูกส่งไปให้คีย์บอร์ดตัวที่ 2 จะเห็นได้ว่า คีย์บอร์ดตัวแรกกับตัวที่สองจะติดต่อกันได้ โดยคีย์บอร์ดตัวแรกจะควบคุมคีย์บอร์ดตัวที่สอง ซึ่งอาศัยข้อมูลที่ได้จากคอมพิวเตอร์

ถ้าไม่อาศัยข้อมูลจากคอมพิวเตอร์ เมื่อมีการเล่นคีย์บอร์ด 1 (ใช้คนเล่น) ส่วนควบคุมคีย์บอร์ด 1 จะส่งรหัสมิดี้ออกไป และคอมพิวเตอร์จะรับข้อมูลนี้ แล้วนำมาทำขบวนการต่างๆ ได้ เช่นเก็บข้อมูลนี้ไว้ในดิสก์ ในลักษณะนี้จะคล้ายกับเทปบันทึกเสียง แต่แทนที่จะเก็บเป็นสัญญาณเสียง คอมพิวเตอร์จะเก็บในรูปของรหัสมิดี้ และเมื่อมีการเล่นซ้ำ ทางคอมพิวเตอร์ก็จะส่งรหัสมิดี้นี้กลับไปยังเครื่องดนตรี และเครื่องดนตรีก็จะทำงาน (เล่น) ตามรหัสมิดี้เดิม ซึ่งเป็นการสร้างเสียงขึ้นมาใหม่นั่นเอง

จะเห็นได้ว่า เมื่อเครื่องดนตรีสามารถสื่อสารกันภายใต้มาตรฐานเดียวกัน จึงทำให้เราสามารถนำไปประยุกต์ใช้งานได้อย่างกว้างขวาง เช่น

1. ควบคุมเครื่องดนตรีได้ทีละหลายๆเครื่อง เนื่องจากเครื่องดนตรีทุกเครื่องสื่อสารด้วยภาษาเดียวกัน

2. สามารถส่งรหัสมิดี้ไปควบคุมกลองไฟฟ้า (DRUM MACHINE) เพื่อให้มีจังหวะตรงกับเครื่องดนตรีอื่นๆ ได้

3. อุปกรณ์แต่งเสียง (Sound Effect) ต่างๆ สามารถได้ด้วยรหัสมิดี้

4. ใช้ในการสร้างเสียง (Voice Quality) ของเครื่องดนตรีประเภทซินธิไซเซอร์ (Voice Editing Software)

5. ข้อมูลจากเครื่องดนตรีเครื่องหนึ่ง อาจนำไปใช้กับอีกเครื่องหนึ่งได้

6. ถ้าใช้การเก็บรหัสมิดี้ แทนที่จะเก็บเป็นสัญญาณเสียง เราสามารถเล่นซ้ำได้อีก ดังที่ได้กล่าวไว้แล้ว ซึ่งจะคล้ายกับการบันทึกเสียง

7. ประโยชน์จากการสื่อสาร เช่น สามารถส่งข้อความมิดี้ผ่านทางโมเด็ม(Modem) ได้ ดังนั้นจึงทำให้สามารถส่งรหัสมิดี้ ผ่านทางสายโทรศัพท์ไปได้ในระยะทางไกลๆ โดยไม่มีวามผิดเพี้ยนของสัญ

สัญญาณเสียง เนื่องจากมีการส่งเป็นรหัส

2.2 การสื่อสารแบบมิดี้

ข้อมูลที่ใช้สื่อสารในระบบมิดี้มีหลายประเภท แต่ก่อนที่จะกล่าวถึงข้อมูลเหล่านั้น มีคำศัพท์ที่จำเป็นที่จะต้องทำความเข้าใจก่อน คือ

แชลแนล (Channel) ในมาตรฐานมิดี้ สามารถส่งข้อมูลได้พร้อมกันถึง 16 แชลแนล นั่นหมายถึงสามารถส่งข้อมูลไปให้เครื่องดนตรี 16 ชิ้นเล่นได้พร้อมๆกัน ข้อมูลที่เป็นข้อมูลแชลแนลแต่ละตัวสามารถกำหนดได้ว่า เป็นข้อมูลของแชลแนลใด ส่วนทางเครื่องดนตรีที่เป็นตัวรับก็สามารถที่จะกำหนดได้ว่าจะฟังข้อมูลจากแชลแนลใดบ้าง หรืออาจจะฟังพร้อมกันทุกแชลแนลก็ได้ หลักการนี้คล้ายกับเครื่องรับโทรทัศน์ที่สามารถเลือกที่จะให้สัญญาณของสถานีใดที่มีอยู่หลายๆสัญญาณปนกันออกมาได้

โหมด (Mode) โหมดต่างๆ ของเครื่องดนตรีแบ่งออกได้เป็น

1.โหมด 1 (Omni - On,Poly) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากทุกแชลแนล (Omni-On) และสามารถเล่นได้ครั้งละหลายๆ โน๊ตพร้อมกัน (Polyphony)

2.โหมด 2 (Omni - On,Mono) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากทุกแชลแนล (Omni-On) และสามารถเล่นได้ครั้งโน๊ตเท่านั้น (Monophony)

3.โหมด 3 (Omni - Off,Poly) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้ จะรับฟังข่าวสารจากแชลแนลที่ได้ตั้งไว้เพียงแชลแนลเดียวเท่านั้น แต่สามารถเล่นได้ครั้งละหลายๆ โน๊ตพร้อมกัน (Polyphony)

4.โหมด 4 (Omni - Off,Mono) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากแชลแนลที่ได้ตั้งไว้เพียงแชลแนลเดียวเท่านั้น และสามารถเล่นได้ครั้งโน๊ตเท่านั้น (Monophony)

ทั้ง 4 โหมด สามารถสรุปได้เป็นตารางดังนี้

รูปที่ 2.2 ตารางโหมด

	Poly	Mono
Omni On	Mode 1	Mode 2
Omni Off	Mode 3	Mode 4

2.3 รูปแบบของข้อมูลมิดี้

ข้อมูลมิดี้ หมายถึง ข้อมูลต่างๆ ที่เครื่องดนตรีใช้ติดต่อสื่อสารกัน แบ่งออกได้เป็น 2 ส่วน ใหญ่ๆ คือ

2.3.1 ข้อมูลที่มีการกำหนดแชลแนล (Channel Message) เป็นข้อมูลที่ส่งออกไป พร้อมๆ กันกับหมายเลขแชลแนล ยังสามารถแบ่งย่อยออกไปได้อีก คือ

ก) ข้อมูลเสียง (Voice Message)

-ข้อมูลที่เกี่ยวกับโน๊ต จะเป็นข้อมูลที่บอกว่า เป็นโน๊ตตัวใด

-ข้อมูลเปลี่ยนโปรแกรม (Program change) เป็นข้อมูลที่บอกให้เครื่องดนตรีที่เป็นตัวรับว่าให้เล่นเป็นเสียงอะไร เช่น ถ้าตัวรับเป็นซินธิไซเซอร์ ก็จะสามารถใช้กับข้อมูลนี้ สั่งให้ซินธิไซเซอร์เปลี่ยนเป็นเสียงที่ซินธิไซเซอร์ สามารถสร้างเสียงใดก็ได้

-ข้อมูลเปลี่ยนการควบคุม (Control Change) เป็นข้อมูลที่ทำให้เสียงที่ออกมาแตกต่างๆ ไปจากธรรมดาเล็กน้อย อาจจะเป็นเสียงสั่น เสียงห้วน ซึ่งจะไม่มีในเครื่องดนตรีทุกชนิด ส่วนมากจะเป็นเครื่องดนตรีประเภทเปียโนไฟฟ้า

-ข้อมูลหลังการกดคีย์ (After Touch) เป็นข้อมูลที่เกิดขึ้นหลังการกดคีย์ของเครื่องดนตรีประเภทคีย์บอร์ด ข้อมูลเหล่านี้ก็เช่น แรงกดคีย์ ความเร็วที่ใช้กดคีย์

-ข้อมูลการเปลี่ยนแปลงเสียง (Pitch Bender) เป็นข้อมูลที่เกิดขึ้นเนื่องจากการใช้ ตัว Pitch Bender ที่มีอยู่บนเครื่องดนตรีประเภทคีย์บอร์ด จะทำให้ความถี่ของโน๊ตที่กดอยู่ตอนนั้นค่อยๆ สูงขึ้นหรือต่ำลง

ข) ข้อมูลโหมด (Mode Message)

เป็นการส่งข้อมูลในการเปลี่ยนโหมด เพื่อทำการเปลี่ยนโหมดของตัวรับ

2.3.2 ข้อมูลที่ไม่มีการกำหนดแชลแนล (System Message) เป็นการส่งข้อมูลไปยังอุปกรณ์ทุกชิ้นที่ต่ออยู่ในข่ายการสื่อสารของมิดี้ ข้อมูลเหล่านี้ก็คือ

ก) ข้อมูลในการซิงโครไนซ์และควบคุมเวลาของระบบ (System Synchronization) เรียกข้อมูลนี้ว่า ข่าวสารระบบควบคุมเวลาจริง (Syatem Real Time Message) สามารถแบ่งย่อยออกได้ เป็น

-ข้อมูลสัญญาณนาฬิกา (Time Clock Message) ใช้ในการซิงโครไนซ์อุปกรณ์ต่างๆ ในระบบ เช่น เครื่องสร้างเสียงกลอง เครื่องจัดลำดับตัวโน๊ต ซึ่งข้อมูลนี้จะถูกส่งออกไป 24 ครั้งใน 1 ควอเตอร์โน๊ต

-ข้อมูลบอกการเริ่มต้น (Start Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทุกชิ้นในเครือข่ายมิดี้เริ่มทำงาน

-ข้อมูลต่อเนื่อง (Continuous Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทำงานที่สัญญาณนาฬิกาถัดไปหลังจากที่หยุดทำงาน

-ข้อมูลบอกการสิ้นสุด (Stop Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทุกชิ้นในเครือข่ายมิดี้หยุดทำงาน

-ข้อมูลบอกการเริ่มต้นใหม่ (System Reset Message) เป็นการเริ่มต้นทำงานใหม่ทั้งหมดเหมือนการเปิดเครื่องใหม่

ข) ข้อมูลระบบปกติ (system Common message) ใช้ในการกำหนดการทำงานและหน้าที่ของอุปกรณ์ต่างๆ ในระบบ แบ่งย่อยได้ดังนี้

-ข้อมูลบอกตำแหน่งของเพลง (Song Position Pointer Message) เป็นการกำหนดการนับจังหวะ (Beat Counter) ของอุปกรณ์ให้มีค่าตามที่ต้องการ

-ข้อมูลเลือกเพลง (Song Select Message) เป็นข้อมูลที่ใช้ในการกำหนดเพลงที่จะเล่น สามารถเลือกได้ถึง 128 เพลง

-ข้อมูลการขอปรับแต่ง (Tune Request Message) ใช้กับซินธิไซเซอร์แบบอนาลอก เป็นการปรับแต่งวงจรสร้างความถี่

2.3.3 ข้อมูลระบบพิเศษ (System Executive Message) เป็นการกำหนดเลขหมายที่แน่นอนให้กับเครื่องดนตรี (Manufacturer' ID Number) เพื่อใช้เครื่องดนตรีหลายๆ ยี่ห้อ สามารถร่วมเล่นกันได้

บทที่ 3

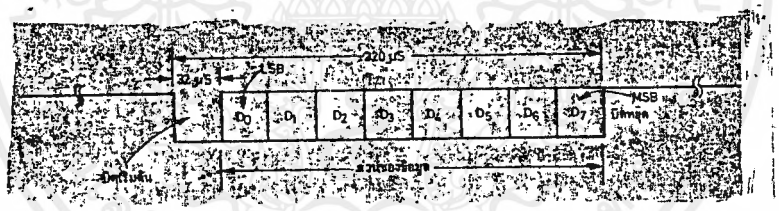
การออกแบบและการสร้าง

3.1 หลักการออกแบบมิดี้การ์ด

3.1.1 มาตรฐานการเชื่อมต่อแบบมิดี้

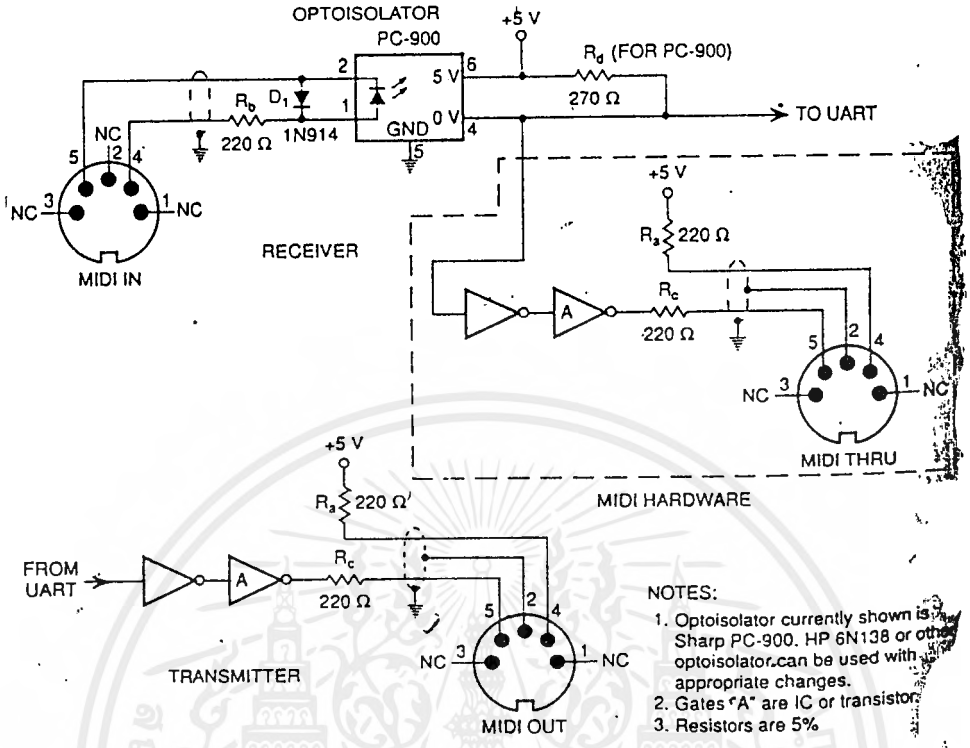
จากในบทที่แล้วได้กล่าวถึง หลักในการเชื่อมต่ออุปกรณ์ที่ใช้มาตรฐานมิดี้ในการสื่อสารเข้าด้วยกัน ในส่วนนี้จึงจะขอกล่าวเพิ่มเติม ดังต่อไปนี้

การส่งของข้อมูลแบบมิดี้ นั้นเป็นการส่งแบบอะซิงโครนัส (Serial Asynchrononus) โดยมีบิทเริ่มต้น 1 บิท (1 Start Bit), บิทสิ้นสุด 1 บิท (1 Stop Bit) และบิทข้อมูล 8 บิท (8 Data Bits) โดยไม่มีบิทตรวจสอบความผิดพลาด (Parity Bit) สายที่ใช้ในการสื่อสารเป็นแบบ 1 เส้น สัญญาณ 1 เส้นสายดิน เพราะฉะนั้น ในสายแต่ละเส้นจึงเป็นการสื่อสารทางเดียว สิ่งที่สำคัญที่ต้องคำนึงถึงมากที่สุดก็คือ ความเร็วในการส่งจะต้องส่งด้วยความถี่ 31250 Hz และต้องมีความผิดพลาดได้ไม่เกิน 1 % ส่วนแจ็คที่ใช้จะต้องเป็นแจ็คดิน (DIN) 5 ขา



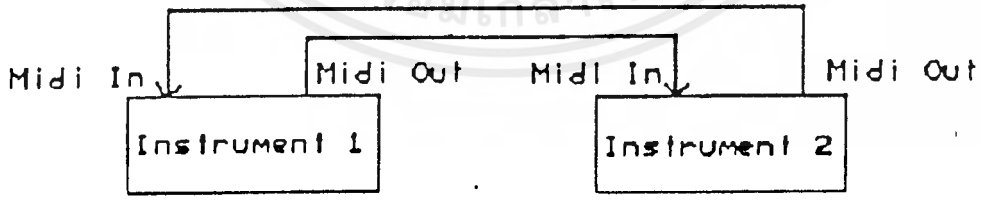
รูปที่ 3.1 รูปแบบของข้อมูลมิดี้มาตรฐาน

เนื่องจากมาตรฐานในการส่งกำหนดให้มีการส่งแบบ Cunrrent Loop (หมายถึง การที่สัญญาณที่ได้รับจะเป็น 0 หรือเป็น 1 พิจารณาได้จากทิศทางของกระแสที่ไหลในสาย) โดยที่กำหนดให้กระแสที่ไหลในสาย มีค่าไม่เกิน 5 มิลลิแอมป์ และสายสัญญาณเชื่อมต่อควรมีความยาวไม่เกิน 50 ฟุต



รูปที่ 3.2 วงจรอินเตอร์เฟสและลักษณะการต่อสายสัญญาณกับแจ็คดิน

3.1.2 โครงสร้างวงจร

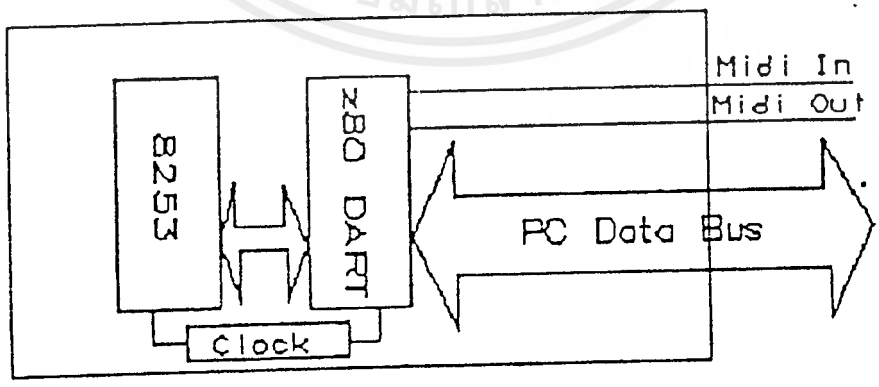


รูปที่ 3.3 ลักษณะการต่อเชื่อมของข่ายการสื่อสารแบบมิดี้

พิจารณาจากรูปที่ 3.3 ส่วนหลักในเครือข่ายการสื่อสารที่ใช้ในโครงงานนี้ คือการ์ดที่ทำหน้าที่ในการส่งสัญญาณมิดี้ และเครื่องดนตรีที่รับสัญญาณมิดี้ โดยพอร์ตที่รับสัญญาณมิดี้เข้าไปของอุปกรณ์ เรียกว่า **พอร์ตมิดี้อิน** (MIDI in Port) และพอร์ตที่ทำหน้าที่ส่งสัญญาณมิดี้ออกมานั้นเรียกว่า พอร์มิดี้เอาท์ (MIDI out Port) ส่วนการต่อเครื่องดนตรีในเครือข่ายมากกว่า 1 ชิ้นจะต้องผ่านทาง พอร์ตมิดี้ทรู (MIDI thru Port) ซึ่งพอร์ทนี้จะส่งสัญญาณที่เข้ามาทางพอร์ตมิดี้อินออกไปทางพอร์ตมิดี้ทรูอีกต่อหนึ่ง เนื่องจากข้อมูลมิดี้ สามารถกำหนดแชลแนลในการรับส่งได้ถึง 16 แชลแนล จึงสามารถใช้สายเส้นเดียวในการติดต่อกับเครื่องดนตรีหลายๆ ชิ้น (โดยที่เครื่องดนตรีแต่ละชิ้นจะถูกตั้งให้รับสัญญาณมิดี้ไม่ซ้ำแชลแนลกัน)

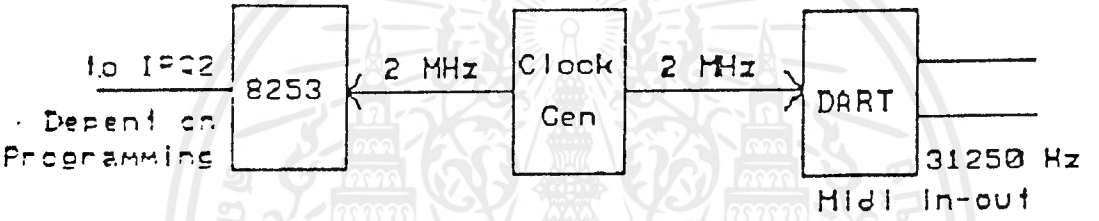
การ์ดมิดี้ที่สร้างขึ้นจะถือเป็นเสมือนเครื่องดนตรีชิ้นหนึ่ง คือ สามารถรับส่งสัญญาณมิดี้ผ่านทางพอร์ทมิดี้อินและมิดี้เอาท์ได้เช่นเดียวกัน แต่ในโครงงานนี้ ความสามารถของพอร์ทมิดี้อินของการ์ดไม่ได้ถูกนำมาใช้ ดังนั้นการ์ดจึงทำตัวเสมือนเป็นแหล่งกำเนิดสัญญาณมิดี้เพียงอย่างเดียว การทำงานของการ์ดมิดี้ที่สร้างขึ้นจะมีส่วนสำคัญอยู่ 3 ส่วน คือ

ส่วนที่ 1 เป็นส่วนที่ทำการเปลี่ยนข้อมูลที่เป็นแบบขนานใน Data Bus ของเครื่อง PC มาเป็นข้อมูลแบบอนุกรม ให้ได้ความเร็วและลักษณะของข้อมูลตามมาตรฐานของมิดี้ (และในทางกลับกันก็สามารถแปลงจากข้อมูลแบบอนุกรมมาเป็นข้อมูลแบบขนานได้เช่นกัน) ในโครงงานนี้ใช้ Z-80 DART มาทำหน้าที่แปลงลักษณะของข้อมูลดังที่กล่าวไว้ เนื่องจาก Z-80 DART ออกแบบมาเพื่อการสื่อสารข้อมูลแบบ Asynchrononus โดยผ่านทางขา RxDA (เป็นขาที่ใช้รับข้อมูลจากสายสัญญาณมิดี้) และขา TxDA(เป็นขาที่ใช้ส่งข้อมูลจากเครื่อง PC ไปยังเครื่องดนตรี) เพื่อการรับส่งที่ถูกต้องตามมาตรฐานมิดี้จึงรับสัญญาณนาฬิกา 2 MHz จากส่วนผลิตสัญญาณนาฬิกามาหารด้วย 64 เพื่อให้ได้ความถี่ในการสื่อสารเป็น 31250 Hz ตามมาตรฐาน



รูปที่ 3.4 บล็อกไดอะแกรมของมิดี้การ์ด

ส่วนที่ 2 ส่วนเคาน์เตอร์ (Counter) ในโครงงานนี้เลือกใช้เบอร์ 8253 เป็นส่วนที่ใช้ในการสร้างความเร็วช้า (Tempo) ทั้งหมดในโปรแกรมส่วนที่ควบคุมการส่งสัญญาณมิดี้ไปยังเครื่องดนตรีซึ่งตัว 8253 นี้จะถูกป้อนด้วยสัญญาณนาฬิกาความถี่ 2 MHz ให้กับเคาร์เตอร์ 0 เป็นตัวหารความถี่ในชั้นแรก จากนั้นสัญญาณนาฬิกาที่ออกจากเคาร์เตอร์ 1 จะถูกส่งไปให้เคาร์เตอร์ 0 ทำการหารความถี่ในชั้นที่ 2 และสัญญาณนาฬิกาที่ออกจากเคาร์เตอร์ 0 ก็จะถูกส่งผ่านไปยังขา IORQ 3 ของสล็อตของเครื่อง PC เพื่อใช้เป็นสัญญาณในการสร้าง ความเร็วช้า (Tempo) ต่างๆ กันไปในแต่ละเพลง ซึ่งค่าของเคาร์เตอร์ 0 และเคาร์เตอร์ 1 สามารถโปรแกรมได้ด้วยซอฟต์แวร์



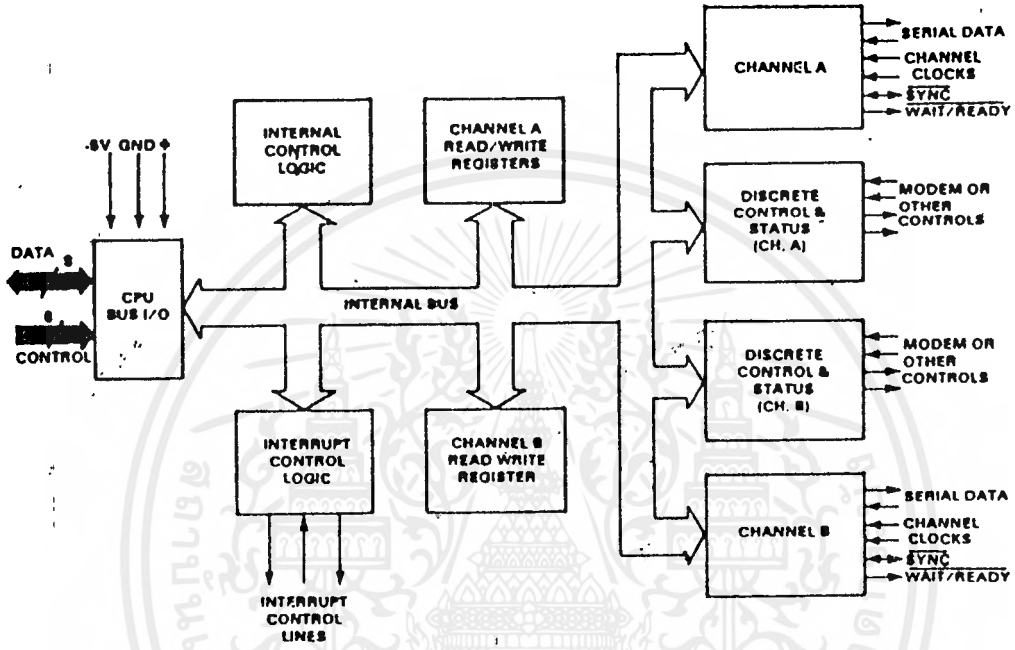
รูปที่ 3.5 สัญญาณนาฬิกาที่จุดต่างๆ บนบอร์ด

และในส่วนที่ 3 เป็นส่วนที่ใช้ในการเลือกพอร์ท (Decoder Port) ที่ใช้ในการติดต่อของเครื่อง PC กับชิพต่างๆ บนการ์ด โดยใช้ 74138 และอุปกรณ์ประกอบ (AND ,OR Gate) มาเป็นตัวเลือก ซึ่งในโครงงานนี้จะทำการเลือกพอร์ทต่างๆ ดังนี้

พอร์ท 301 เป็นพอร์ทข้อมูล (ทั้งอินและเอาท์) ของ Z-80 DART
พอร์ท 303 เป็นพอร์ทควบคุมของแชลแนล B ของ Z-80 DART
พอร์ท 304 เป็นพอร์ทข้อมูลของเคาร์เตอร์ 0 ของ 8253
พอร์ท 305 เป็นพอร์ทข้อมูลของเคาร์เตอร์ 1 ของ 8253

3.1.3 การทำงานและการเริ่ม (Initialize) ของ Z-80 DART

Z-80 DART เป็นชิฟที่ออกแบบมาเพื่อการสื่อสารแบบอนุกรมอะซิงโครนัสโดยเฉพาะ โดยมีบล็อกไดอะแกรม ดังนี้



รูปที่ 3.6 บล็อกไดอะแกรมของ Z-80 DART

จากรูปจะเห็นว่า Z-80 DART มีแชลแนลในการรับส่งถึง 2 แชลแนล คือ แชลแนล A และแชลแนล B แต่ในโครงงานนี้จะใช้แชลแนล B เพียงแชลแนลเดียว สามารถควบคุมการทำงานและอ่านค่าสถานะด้วยรีจิสเตอร์ต่างๆ จึงทำให้สามารถควบคุมการทำงานทั้งหมดได้ด้วยซอฟแวร์ ซึ่งรีจิสเตอร์ต่างๆ เหล่านี้ ประกอบด้วยรีจิสเตอร์ในการเขียน (internal write register)ของแชลแนล A มี 7 ตัว และของแชลแนล B มี 8 ตัว โดยที่แชลแนล B มีรีจิสเตอร ที่ใช้เก็บอินเตอร์รัพท์เวคเตอร์อีก 1 ตัว นอกจากนั้น ยังมีรีจิสเตอร์ที่ใช้ในการอ่านค่าสถานะต่างๆ อีก 3 ตัว (ข้อมูลของรีจิสเตอร์สามารถอ่านเพิ่มเติมได้จากภาคผนวก) การทำงานของ Z-80 DART สามรถทำได้ทั้ง แบบอินเตอร์รัพท์ (Interrupt),โพลิ่ง (Polling) และ DMA

การตั้งค่าเริ่มต้นของ Z-80 DART สามารถทำได้ดังนี้ คือ

เริ่มแรก ต้องทำการตั้งค่าให้ Z-80 DART มีการรับส่งข้อมูลที่เป็นมาตรฐานของมิดี้ คือ ข้อมูล 8 บิท,มีบิทเริ่มต้นเป็น 0 มีบิทสิ้นสุดเป็น 1 และมีการหารความถี่ที่เข้ามาด้วย 64 เพื่อนำไปใช้ในการรับส่งข้อมูล (2 MHz / 64 = 31250 Hz) ซึ่งสัญญาณนี้จะมีผลต่อขา RxA,RxB,TXA และTxB โดยมีขั้นตอนในการตั้งค่าดังนี้

1.เขียนข้อมูลให้ รีจิสเตอร์เขียน 0 (WR0)เพื่อเริ่มการทำงาน

2.เขียนข้อมูลให้ รีจิสเตอร์เขียน 1 (WR1)เพื่อเป็นการกำหนดค่าที่จะนำมาใช้หารความถี่ของสัญญาณนาฬิกาที่เข้ามา โดยในโครงงานนี้ ต้องกำหนดให้มีค่า 64 และกำหนดขนาดขนาดของบิทสิ้นสุดเท่ากับ 1

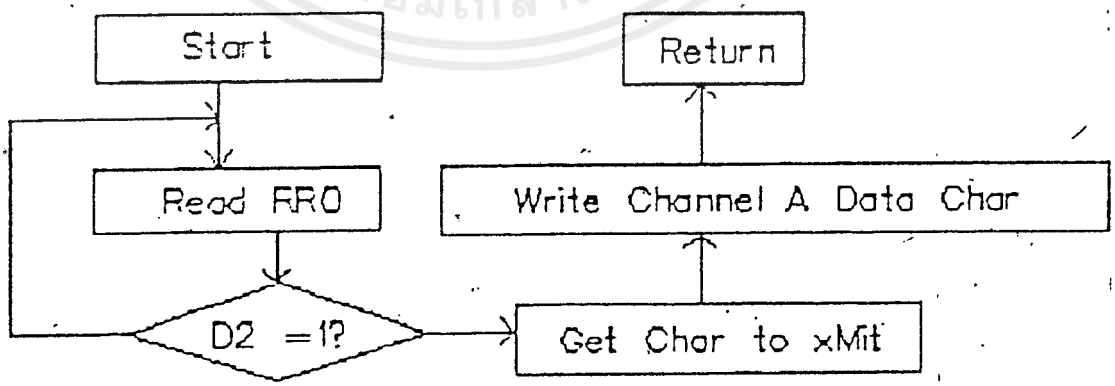
3.เขียนข้อมูลให้ รีจิสเตอร์เขียน 3 (WR3)เพื่อเป็นการกำหนดจำนวนบิทของข้อมูลที่ทำการรับให้เป็น 8 บิท และทำการอินาเบิ้นตัวรับ (Receiver Enable) กับเซ็ทให้เป็นอินาเบิ้ลแบบอัตโนมัติ (Auto Eable)

4.เขียนข้อมูลให้ รีจิสเตอร์เขียน 5 (WR5)เพื่อเป็นการกำหนดจำนวนบิทของข้อมูลที่ทำการส่งให้เป็น 8 บิทและทำการอินาเบิ้ลตัวส่ง (Transmitter Eneble)

5.เขียนข้อมูลให้รีจิสเตอร์เขียน 1 (WR1)เพื่อทำการอินาเบิ้ล การขออินเตอร์รัพท์จาก Z-80 DART

ขั้นตอนในการส่งข้อมูลอนุกรม สามารถทำได้ดังนี้ คือ

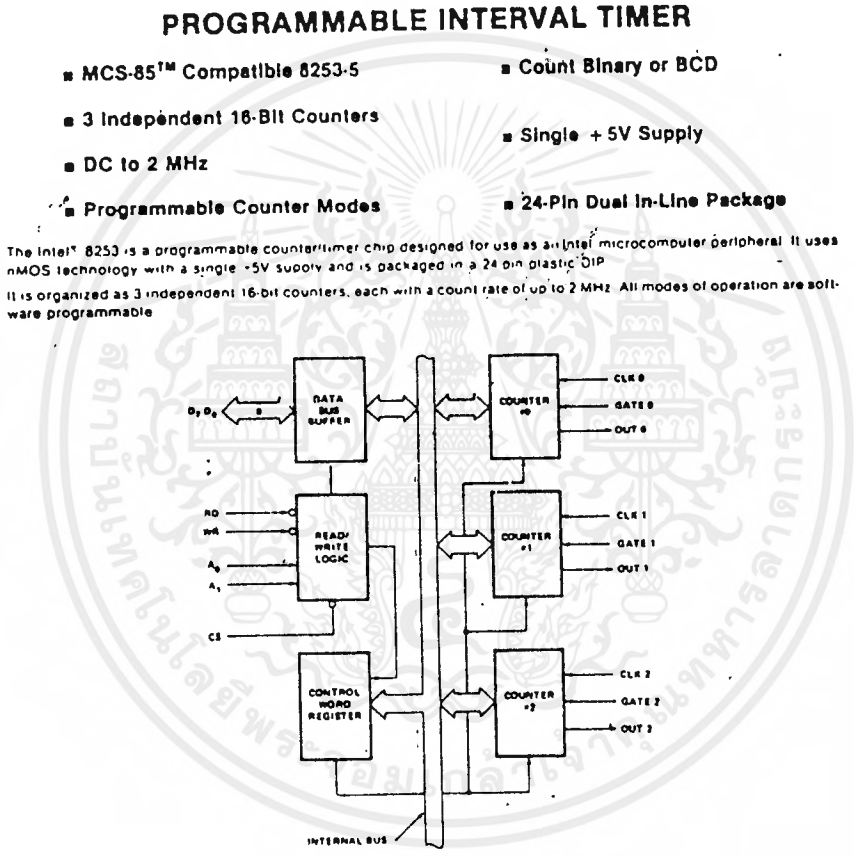
ในขั้นตอนแรกสด จะทำการตรวจสอบสถานะของบัฟเฟอร์ตัวส่ง ว่าว่างหรือที่จะรับข้อมูลหรือไม่ โดยอ่านจากรีจิสเตอร์สถานะตัวที่ 0 (RR0) และตรวจสอบบิทที่ 3 ถ้ามีค่า 1 แสดงว่า บัฟเฟอร์ว่างสามารถที่จะส่งข้อมูลได้ แต่ถ้ามีค่าเป็น 0 จะต้องทำการรอแล้วตรวจสอบใหม่จนมีค่าเป็น 1 จึงจะส่งข้อมูลออกไปได้



รูปที่ 3.7ขั้นตอนในการส่งข้อมูล

3.1.4 การทำงานและการเริ่มต้นของ 8253

8253 เป็นไอซีที่สามารถทำหน้าที่เป็น ไทม์เมอร์ (Timer)หรือ เคาร์เตอร์ (Counter) ก็ได้ โดยจะมีตัวนับเวลาอยู่ 3 ชุด (ดังรูปที่ 3.8) ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิท โดยแต่ละตัวสามารถทำงานได้เป็นอิสระต่อกัน มีขา Clk เป็นขาที่รับสัญญาณเข้ามา ขา Out เป็นขาที่ส่งสัญญาณเอาท์พุทออกไป การทำงานของตัวนับแต่ละตัว สามารถทำงานได้หลายโหมด โดยสามารถควบคุมได้โดยซอฟแวร์ผ่านทางรีจิสเตอร์ควบคุมของ 8253



รูปที่ 3.8 บล็อกไดอะแกรมของ 8253

โหมดการทำงานต่างๆ ของ 8253

1.โหมด 0 Event Counter

2.โหมด 1 Programmable one-shot

3.โหมด 2 Reate Generator

4.โหมด 3 Square Wave Generator

5.โหมด 4 Software Triggered Strobe

6.โหมด 5 Hardware Triggered Strobe

โหมดที่เลือกใช้ในโครงงานนี้ คือ ให้เคาร์เตอร์ทั้ง 2 ตัวทำงานที่โหมด 2 ซึ่งมีลักษณะเป็นการหารความถี่ที่เข้ามาทางขา Clk ด้วยค่าที่โปรแกรมไว้ นั่นคือสัญญาณนาฬิกาเข้ามา 1 ลูกก็จะลดค่าที่ตั้งไว้ในรีจิสเตอร์ เมื่อนับลงถึง 0 ก็จะให้เอาท์พุทอีกหนึ่งลูก จากนั้นเริ่มนับถอยหลังลงมาใหม่เมื่อถึง 0 ก็จะให้เอาท์พุทอีกหนึ่งลูกเป็นอย่างนี้เรื่อยๆ สามารถเขียนเป็นขั้นตอนตั้งค่าได้ ดังนี้

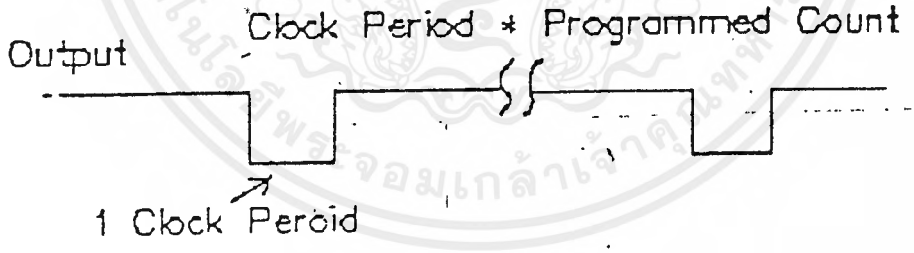
1.ตั้งค่ารีจิสเตอร์ควบคุม ตั้งเคาร์เตอร์ 0 ให้ทำงานในโหมด 2 นับแบบฐาน 2 แล้วให้ใส่ไบท์ต่ำก่อน แล้วตามด้วยไบท์สูง

2.ใส่ค่าไบท์ต่ำในเคาร์เตอร์ 0 ตามด้วยไบท์สูง

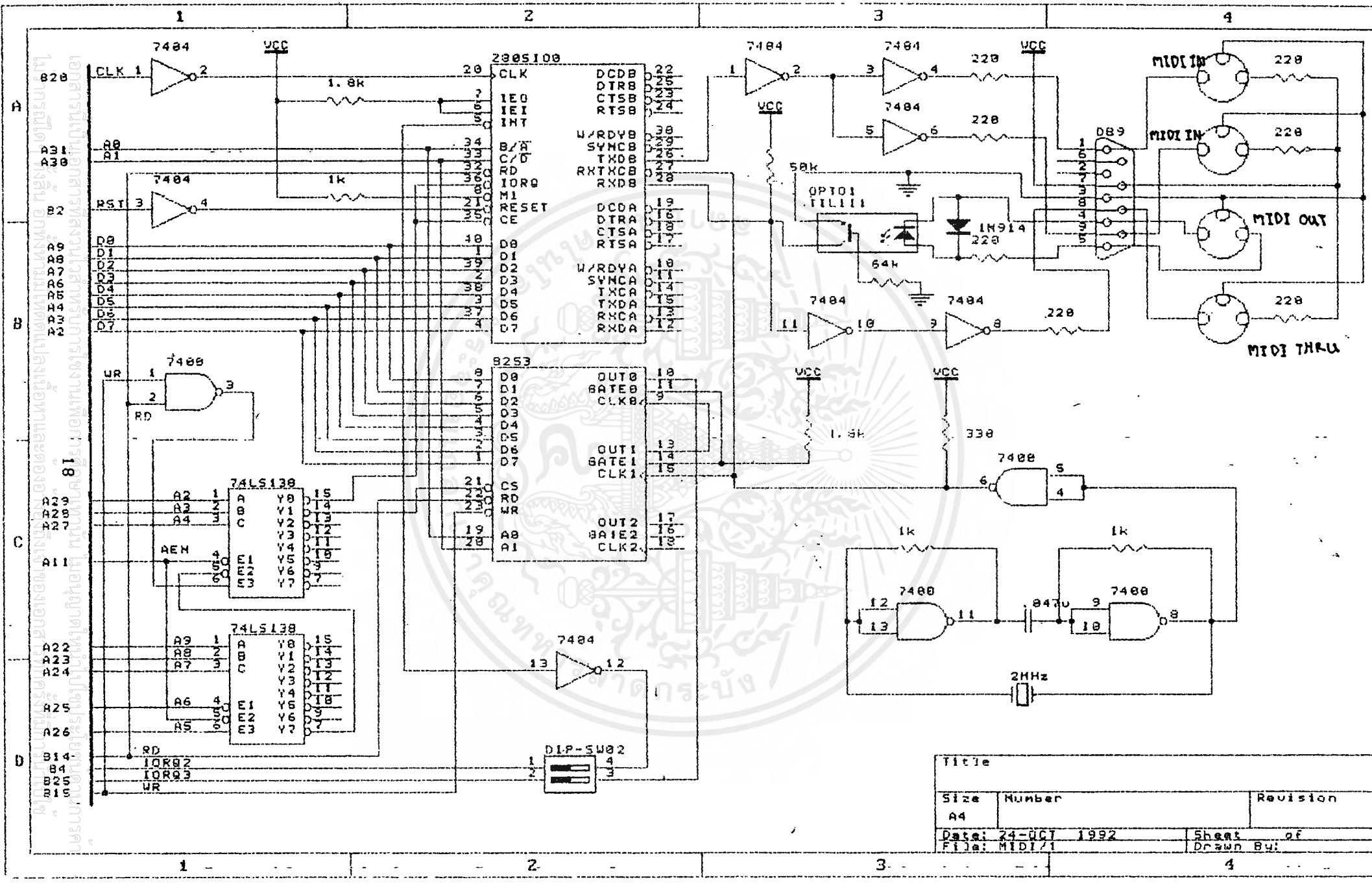
3.ตั้งค่ารีจิสเตอร์ควบคุม ตั้งเคาร์เตอร์ 1 ให้ทำงานในโหมด 2 นับแบบฐาน 2 แล้วให้ใส่ไบท์ต่ำก่อน แล้วตามด้วยไบท์สูง

4.ใส่ค่าไบท์ต่ำในเคาร์เตอร์ 1 ตามด้วยไบท์สูง

ส่วนการทำงานในโหมดอื่นๆ นั้น สามารถอ่านเพิ่มเติมได้จาก ภาคผนวก



รูปที่ 3.9 ลักษณะการหารความถี่ของ โหมด 2



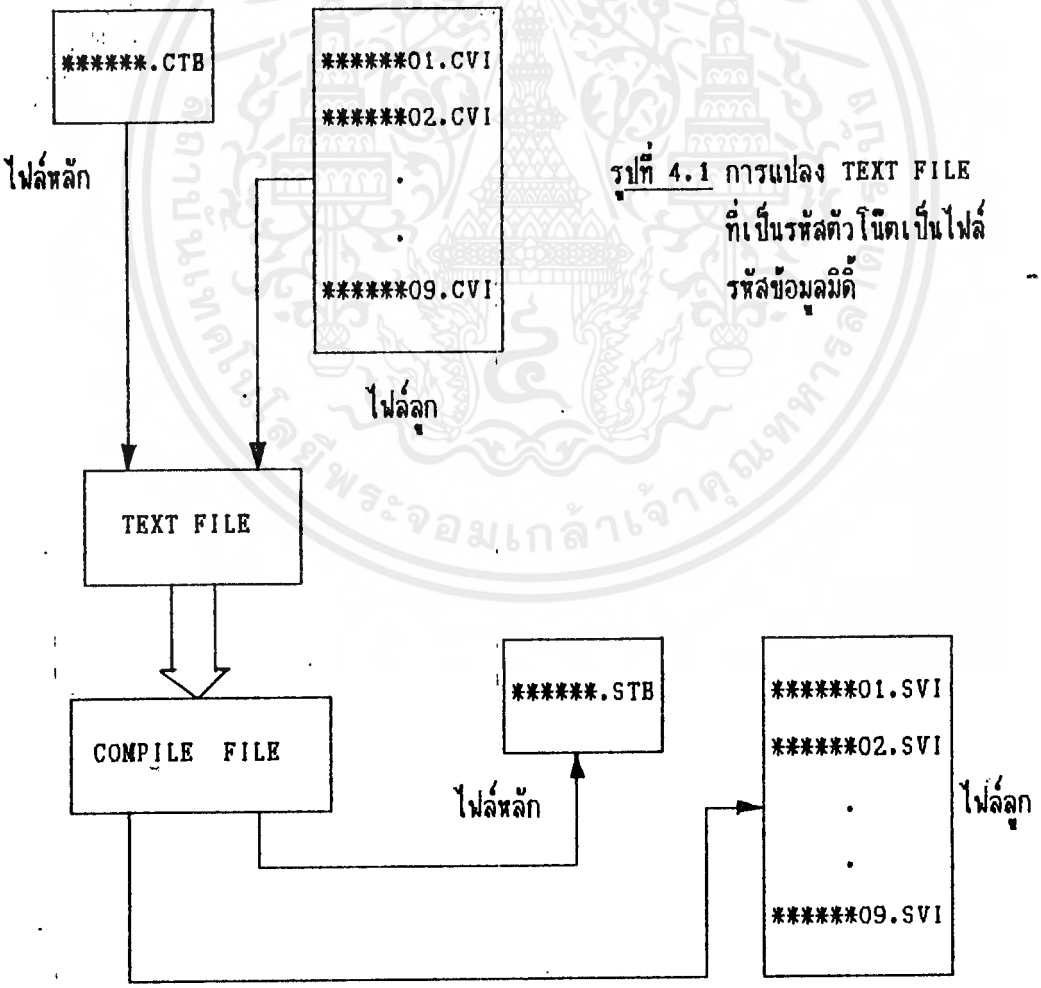
รูปที่ 3.10 วงจรของมิดี้การ์ด

บทที่ 4
การสร้างไฟล์รหัสข้อมูล

4.1 ที่มา

ด้วยเหตุที่ว่าการแปลงจากโน๊ตดนตรีสากลไปเป็นรหัสข้อมูลมิดี้ ในภาคแรกยังทำได้ไม่สำเร็จ ดังนั้นจึงจำเป็นที่จะต้อง ทำการอ่านโน๊ตดนตรีโดยผู้ที่สามารถอ่านโน๊ตดนตรี เพื่อนำมาเข้ารหัสที่ทำให้สามารถอ่านทำความเข้าใจได้ง่ายขึ้น จากนั้นก็นำเอาไฟล์รหัสตัวโน๊ตอันนี้ไปผ่านโปรมแกรมคอมไพล์ไฟล์ที่เขียนขึ้นเพื่อแปลงไฟล์รหัสข้อมูลตัวโน๊ตให้เป็นรหัสข้อมูลมิดี้ ซึ่งรหัสข้อมูลมิดี้อันนี้ จะมีลักษณะเป็นเลขฐานสิบหกโดยมีการจัดเก็บไว้ที่ไฟล์อีกไฟล์หนึ่งที่สามารถโหลดออกมาแล้วส่งไปควบคุมการทำให้เกิดเสียงแก่เครื่องดนตรีได้เลย

4.2 หลักการสร้างไฟล์



รูปที่ 4.1 การแปลง TEXT FILE ที่เป็นรหัสตัวโน๊ตเป็นไฟล์รหัสข้อมูลมิดี้

จากรูปที่ 4.1 เป็นแผนภาพที่แสดงถึงการแปลง TEXT FILE ที่เขียนขึ้นเป็นรหัสตัวโน๊ตให้เป็นไฟล์รหัสข้อมูลมิดี้ โดยมีรายละเอียดของไฟล์ต่างๆ ดังนี้

-ไฟล์หลัก *.CTB เป็นไฟล์ที่เขียนขึ้นเพื่อเก็บรายละเอียดของไฟล์ลูก เป็นต้นว่า มีไฟล์ลูกกี่ไฟล์ แต่ละไฟล์เก็บรหัสของเครื่องดนตรีชิ้นไหน ชื่อเพลงอะไร

-ไฟล์ลูก *.CVI เป็นไฟล์ของการแปลงตัวโน๊ตดนตรีของเครื่องดนตรีแต่ละชิ้น เป็นต้นว่าไฟล์ที่ชื่อ ******01.CVI เป็นไฟล์โน๊ตดนตรีของเปียโน ส่วนไฟล์ที่ชื่อ ******02.CVI เป็นไฟล์โน๊ตดนตรีของไวโอลิน (มีเครื่องดนตรีกี่ชิ้นก็มีจำนวนไฟล์เท่านั้นไฟล์)

-ไฟล์ COMPILE FILE เป็นไฟล์ที่เขียนขึ้นเพื่อแปลงไฟล์หลัก *.CTB และไฟล์ลูก *.CVI ให้เป็นไฟล์หลัก ******.STB และไฟล์ลูก ******01.STB ถึง ******09.STB ตามลำดับ

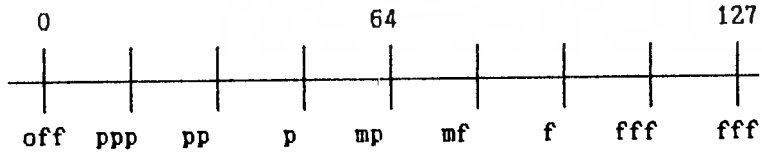
-ไฟล์ *.STB และไฟล์ *.SVI เป็นไฟล์รหัสข้อมูลมิดี้ ที่พร้อมที่จะส่งไปควบคุมเครื่องดนตรีแล้ว

4.3 ตัวอย่างการสร้างไฟล์จากเพลงชื่อ POLKA

จากโน๊ตดนตรีสากลในรูปที่ 4.2 เป็นโน๊ตของกีต้าร์คลาสสิคที่ชื่อ POLKA ซึ่งใช้กีต้าร์เล่นสามตัวด้วยกัน (เรียงจากบนไปล่างตามลำดับ) ตัวที่หนึ่งและตัวที่สองเล่นเป็น Melody ส่วนตัวที่สามเล่นเป็น Chord มีคุณแจซอลเป็นคุณแจประจำหลัก มีอัตราจังหวะ $\frac{4}{4}$

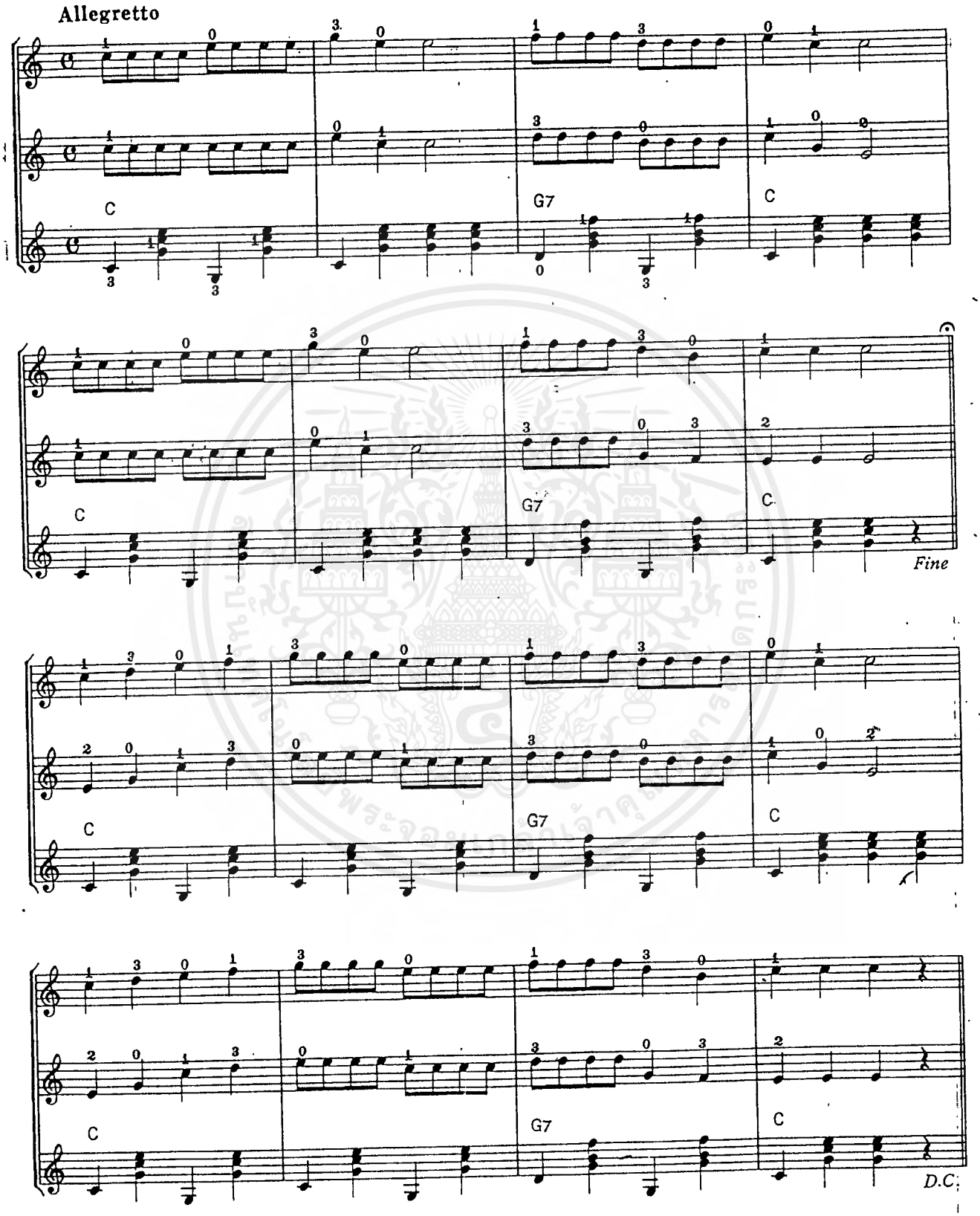
สำหรับการที่จะบ่งบอกว่าเป็นโน๊ตอะไรนั้น ตามมาตรฐานของมิดี้ไม่ได้กำหนดไว้ ดังนั้นเพื่อความสะดวกในการทดลองจึงได้กำหนดมาตรฐานขึ้นมาเอง ดังในตาราง 4-1 ภายในตารางประกอบไปด้วย ตัวโน๊ต,ตัวหยุดตัวโน๊ต,ชื่อตัวโน๊ต และค่าจังหวะตัวโน๊ต จากตารางจะเห็นได้ว่า ประกอบไปด้วยตัวโน๊ตและตัวหยุดตัวโน๊ต อย่างละหกตัวและมีค่าจังหวะเท่ากัน เช่น ตัวโน๊ตตัวกลมและตัวหยุดตัวกลมมีค่าจังหวะเท่ากันคือ 96 เป็นต้น

ค่าความดังตัวโน๊ตแต่ละตัวซึ่งกำหนดไว้ตามมาตรฐานของมิดี้ ดังรูป 4.2



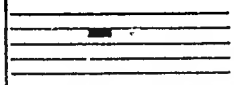
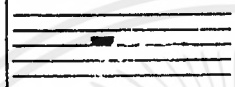
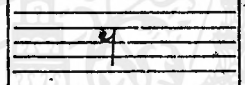
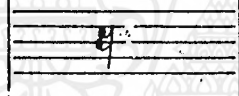
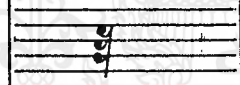
รูปที่ 4.2 ระดับค่าความดังที่กำหนดตามมาตรฐานมิดี้

POLKA



รูปที่ 4.2 โน๊ตเพลงชื่อ POLKA

ตาราง 4-1

ตัวโน๊ต	ตัวหยุดตัวโน๊ต	ชื่อ	ค่าจังหวะ
		ตัวกลม	96
		ตัวขาว	48
		ตัวดำ	24
		เขบ็จ 1 ชั้น	12
		เขบ็จ 2 ชั้น	06
		เขบ็จ 3 ชั้น	03

ตัวอย่างการให้ความหมายในไฟล์ POLKA01.CVI โดยอ่านจากโน๊ตกีต้าร์ตัวที่หนึ่ง

$G -----> เป็นหัวกุญแจซอล

@4/4 -----> อัตราจังหวะ $\frac{4}{4}$

12/0/064/1/C /04 -----> ตัวเขบ็จ 1ชั้น/ไม่มีเครื่องหมายโยงเสียง/ความดัง
โน๊ต 64/เล่นโน๊ต 1 ตัว/คือ โน๊ตตัวโด/ที่ Octave 4

.....

ตัวอย่างการให้ความหมายในไฟล์ POLKA03.CVI โดยอ่านจากโน๊ตกีต้าร์ตัวที่สาม

$G -----> เป็นหัวกุญแจซอล

@4/4 -----> อัตราจังหวะ $\frac{4}{4}$

24/0/064/1/C /03

24/0/064/2/G /03/C /04/E /04 -----> ตัวเขบ็จ 1ชั้น/ไม่มีเครื่องหมายโยงเสียง
/ความดังโน๊ต 64/เล่นโน๊ต 3 ตัว/คือ
โน๊ตตัวซอล/ที่ Octave 3/โน๊ตตัวโด/
ที่ Octave 4/โน๊ตตัวมี/ที่ Octave 4

ในการเขียนแปลความหมายของโน๊ตเพลง POLKA ของกีต้าร์ทั้งสามตัว สามารถแปลและเขียนเป็น Text File ได้เป็นสามไฟล์คือ POLKA01.CVI , POLKA02.CVI และ POLKA03.CVI ดังหน้าถัดไปนี้

สำหรับในการทดลองครั้งนี้ได้มีการแปลงโน๊ตเพลง RIGHT HERE WAITING มาเป็นเพลงตัวอย่าง ซึ่งเพลงนี้ประกอบไปด้วยเครื่องดนตรี 6 ชิ้นใช้เวลาในการแปลและเขียนเป็น Text File ประมาณ 1 สัปดาห์ จึงทำให้เห็นว่า ถ้าหากสามารถอ่านโน๊ตเพลงจากกล้องมาเป็นรหัสข้อมูลมิดี้ได้สำเร็จ ก็จะทำให้ใช้เวลาในการแปลงได้น้อยลง

ไฟล์ POLKA01.CVI

```
$G
@4/4
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/E /04
12/0/064/1/E /04
12/0/064/1/E /04
12/0/064/1/E /04
24/0/064/1/G /04
24/0/064/1/E /04
48/0/064/1/E /04
12/0/064/1/F /04
12/0/064/1/F /04
12/0/064/1/F /04
12/0/064/1/F /04
12/0/064/1/D /04
12/0/064/1/D /04
12/0/064/1/D /04
12/0/064/1/D /04
24/0/064/1/E /04
24/0/064/1/C /04
48/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/E /04
12/0/064/1/E /04
    ...
```

ไฟล์ POLKA02.CVI

```
$G
@4/4
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
24/0/064/1/E /04
24/0/064/1/C /04
48/0/064/1/C /04
12/0/064/1/D /04
12/0/064/1/D /04
12/0/064/1/D /04
12/0/064/1/D /04
12/0/064/1/B /03
12/0/064/1/B /03
12/0/064/1/B /03
12/0/064/1/B /03
24/0/064/1/C /04
24/0/064/1/G /03
48/0/064/1/E /03
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
12/0/064/1/C /04
   ...
```

ไฟล์ POLKA03.CVI

```
$G
@4/4
24/0/064/1/C /03
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/G /02
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/C /03
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/D /03
24/0/064/3/G /03/B /03/F / 04
24/0/064/1/G /02
24/0/064/3/G /03/B /03/F / 04
24/0/064/1/C /03
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/C /03
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/G /02
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/C /03
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/3/G /03/C /04/E / 04
24/0/064/1/D /03
24/0/064/3/G /03/B /03/F / 04
24/0/064/1/G /02
24/0/064/3/G /03/B /03/F / 04
```

. . .

บทที่ 5
การแปลงโน๊ตดนตรีสากลจากภาพให้เป็นรหัสมิดี้

สำหรับโครงงานในส่วนนี้ ได้จัดทำขึ้นเพื่อเป็นแนวทางให้กับผู้ที่จะทำการศึกษาและพัฒนาต่อ จะเห็นได้ว่าในการทำโครงงานในขั้นตอนนี้ เป็นการทำงานในส่วนของการศึกษาในภาคแรก เนื่องจากมีขีดจำกัดในด้านเวลาและความรู้ในด้านการวิเคราะห์รูปภาพของเรามีน้อย จึงสามารถแสดงให้เห็นพอเป็นแนวทางในการศึกษาได้บ้างเท่านั้น

5.1 แนวความคิดในการแปลงโน๊ต

จากการสังเกตเราสามารถแบ่งสัญญลักษณ์ของโน๊ตดนตรีสากลตามรูปแบบการเขียนได้ 2 พวก คือ

1. สัญญลักษณ์ที่มีรูปแบบแน่นอนตายตัว เช่น กุญแจประจำหลัก, ตัวเลขบอกอัตราจังหวะ, ตัวหยุด, ตัวอักษรต่างๆ ที่ใช้บอกตำแหน่ง การเล่นวนซ้ำ การบอกให้เล่นช้าหรือเร็ว เป็นต้น
2. สัญญลักษณ์ที่มีหลายรูปแบบ ซึ่งก็คือสัญญลักษณ์ที่เป็นตัวโน๊ตนั่นเอง คือ ตัวดำ, ตัวกลม, ตัวขาว, เขบ็จหนึ่งชั้น, เขบ็จสองชั้น และเขบ็จสามชั้น

สำหรับการแปลงตัวโน๊ตในแต่ละแบบ จะใช้วิธีการที่แตกต่างกัน โดยในสัญญลักษณ์แบบที่หนึ่งสามารถเก็บเป็นลักษณะของฐานข้อมูล (database)ได้ เมื่อต้องการที่จะแปลความหมายก็เพียงแต่นำมาเปรียบเทียบกันเท่านั้น ในสัญญลักษณ์แบบที่สองนั้น เนื่องจากตัวโน๊ตสามารถเขียนได้หลายรูปแบบ จนไม่สามารถเก็บรูปแบบของตัวโน๊ตทั้งหมดไว้เป็นฐานข้อมูลเหมือนแบบที่หนึ่งได้ ทางแก้คือ เราจะแยกตัวโน๊ตออกเป็นส่วนๆ ซึ่งในโน๊ตแต่ละตัว จะมีส่วนประกอบที่แตกต่างกัน จากนั้นนำส่วนของตัวโน๊ตทั้งหมดที่แยกได้ มาวิเคราะห์ว่าเป็นส่วนของตัวโน๊ตอะไร

5.2 ขั้นตอนในการแปลงโน๊ต

1. อ่านไฟล์ข้อมูลภาพที่ได้จากแสกนเนอร์(ชื่อไฟล์ *.BMP)ลงสู่หน่วยความจำ ซึ่งข้อมูลที่ได้จะเป็นแบบ Gray Sdale คือเป็นภาพขาวดำ ที่มีระดับความเข้มของสี 256 ระดับ ข้อมูลที่ได้มีค่า 00-FF

2. การทำ Fillter เนื่องจากภาพที่ได้จากแสกนเนอร์ มีสัญญาณรบกวนติดมาด้วย ทำให้ภาพไม่คมชัดและผิดเนื่องจากภาพเดิมที่ถ่ายมา เราจึงต้องทำการ fillter ก่อน เพื่อให้ได้ภาพที่ถูกต้องพร้อมที่จะนำไปประมวลผลในขั้นตอนต่อไป

3. ทำการแปลงข้อมูลจาก Gray Sdale ให้เป็นข้อมูล"1" และ"0" โดย "1" คือ ส่วนที่เป็นพื้นของตัวโน๊ต และ "0" คือ สีพื้น

4. ลบบรรทัด 5 เส้นออก เพื่อให้ตัวโน๊ตแยกออกจากกัน ซึ่งจะทำให้หาขอบเขตของตัวโน๊ตได้และจดจำตำแหน่ง ของบรรทัด 5 เส้น

5. หาขอบเขตของชุดตัวโน๊ตแต่ละชุดและจดจำตำแหน่งของชุดตัวโน๊ต โดยโน๊ตที่ติดกันคือชุดโน๊ต

เดียวกัน

6. ทำ Skeletonizing คือ หาโครงร่างของตัวโน๊ตเพื่อให้สามารถแยกโน๊ตออกเป็นส่วนๆได้

7. จากขั้นตอนที่ 5 นำชุดของตัวโน๊ตที่ได้จากขั้นตอนที่ 5 โดยทั้งหมดได้ใช้หลักการ Skeletonizing แล้วมาแยกโน๊ตออกเป็นส่วนๆ จากนั้นเข้ารหัสของแต่ละส่วนไว้ และนำมาวิเคราะห์ว่า ส่วนต่างๆ ที่ตรวจเช็คได้ ประกอบกันเป็นตัวโน๊ตอะไร โดยในขั้นตอนนี้ เราจะเช็คได้ว่าโน๊ตนั้น เป็นตัวอะไร (ตัวดำ, ตัวกลม, ตัวขาว, เขบ็จหนึ่งชั้น, เขบ็จสองชั้น และเขบ็จสามชั้น)

8. ตรวจเช็คตำแหน่งของตัวโน๊ตเทียบกับบรรทัด 5 เส้น เพื่อหาว่าตัวโน๊ตอยู่ที่ตำแหน่งใดบนบรรทัด 5 เส้น และเมื่อทราบตำแหน่งแล้ว เราก็สามารถเช็คได้ว่า ตัวโน๊ตนั้นเป็นเสียงอะไร (โด , เร ,มี ,ฟา, ซอล ,ลา ,ที)

9. เข้ารหัสให้กับตัวโน๊ต เก็บไว้เป็นข้อมูลเพลงและเช็คชุดของตัวโน๊ตชุดต่อไปจนกว่าจะหมด

5.3 ทฤษฎีโครงกระดูก (Skeleton)

วิธีการหาโครงร่างของภาพมีหลายวิธี และแต่ละวิธีก็ใช้งานแตกต่างกัน ซึ่งเราจะต้องเลือกเอาวิธีใดวิธีหนึ่ง มาใช้กับโครงงานนี้ ซึ่งในที่นี้เราใช้วิธีการของ Zhang และ Suen ซึ่งถูกพัฒนามาขึ้นในปี 1984 ใช้สำหรับ Thinning binary regions หมายถึงภาพที่มีลักษณะบางและข้อมูลที่เก็บจะเป็นลักษณะไบนารี คือ จุดที่เป็นภาพจะมีค่าเป็น "1" และจุดที่เป็นฉากจะมีค่าเป็น "0" ซึ่งวิธีการที่จะทำเราได้เส้นโครงร่าง ที่มีความหนา 1 Pixel อยู่ที่กึ่งกลางภาพ

วิธีการนี้ประกอบด้วยขั้นตอน 2 ขั้นตอน ซึ่งจะใช้กับ Contour point ที่อยู่ภายในขอบเขตของภาพ โดย Contour point เป็นจุดใดๆ ที่มีค่าเป็น "1" และมีจุดรอบๆ อย่างน้อย 1 จุดที่มีค่าเป็น"0"

ในการตรวจเช็คจุดภาพโดยการนำเอา Matrix ขนาด 3x3 ไปวางครอบจุดที่ต้องการเช็ค และเราจะใช้จุดรอบๆ ตัวมันเป็นการตรวจสอบ จากรูป 5.1 เป็น Matrix ขนาด 3x3 ที่นำไปครอบจุด

P8	P1	P2
P7	P0	P3
P6	P5	P4

รูปที่ 5.1 Matrix ขนาด 3x3

5.4 ภาพที่ต้องการตรวจเช็ค

ขั้นแรก

เมื่อเราใช้ Matrix ขนาด 3x3 ไปครอบจุดที่ต้องการตรวจเช็ค และจุดนั้นจะถูกลบ เมื่อเงื่อนไขต่อไปนี้เป็นจริง

$$\left.\begin{array}{l} (a)\ 2 \leqslant N(P0) \leqslant 6\ , \\ (b)\ S(P0) = 1\ , \\ (c)\ P1*P3*P5 = 0 \\ (d)\ P3*P5*P7 = 0 \end{array}\right\} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

โดยที่ N(P0) คือ จำนวนจุดรอบจุด P0 ซึ่งมีค่าเป็น "1" เราจึงได้

$$N(P0) = P1 + P2 + \ldots + P7 + P8 \quad \ldots\ldots\ldots\ldots\ldots (2)$$

และ S(P0) คือ จำนวนของการเปลี่ยนแปลงจาก "0" ไปเป็น "1" ในลำดับของ P1,P2,..P7,P8 ตัวอย่างเช่น ในรูปที่ 5.2 จะได้ N(P0) = 4 และ S(P0) = 3

0	0	1
1	P0	0
1	0	1

รูปที่ 5.2 แสดงการตจวจสอบเงื่อนไข (a) และ (b) จากสมการที่ (1) จะได้ N(P0) = 4 และ S(P0) = 3

ขั้นตอนที่สอง

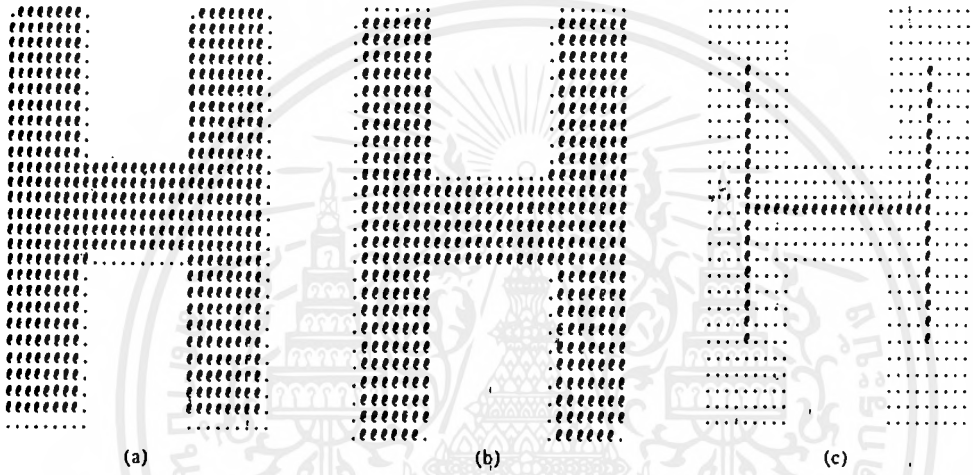
สำหรับขั้นตอนนี้ จะใช้เงื่อนไข (a) และ (b) ในการตรวจสอบเหมือนขั้นตอนแรก แต่เงื่อนไข (c) และ (d) จะถูกเปลี่ยนไปเป็น

$$\left.\begin{array}{l} (c')\ P1*P3*P7 = 0\ , \\ (d')\ P1*P5*P7 = 0 \end{array}\right\} \quad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

วิธีการ

จากขั้นตอนแรก เราจะตรวจสอบจุดทุกจุดในภาพ เพื่อหาจุดที่มีเงื่อนไขเป็นจริงตาม (a) ถึง (d) ถ้าพบจุดที่ตรงตามเงื่อนไข ก็ทำสัญญลักษณ์ที่จุดนั้นก่อน รอจนกว่าตรวจสอบจุดทุกจุดในภาพครบแล้ว จึงทำการลบจุดที่ได้ ทำสัญญลักษณ์ไว้ทั้งหมด จากนั้นทำการตรวจสอบใหม่ทั้งภาพ ตามเงื่อนไขในขั้นตอนที่ 2 และลบจุดที่ทำสัญญลักษณ์ไว้ทั้งหมดเช่นกัน ทำซ้ำในขั้นตอนแรกใหม่ จนกว่าจะไม่มีจุดที่ถูกต้องตามเงื่อนไขใดๆ เลย ขั้นสุดท้ายเราจะได้โครงร่างของภาพมีลักษณะเป็นเส้นขนาด 1 Pixel ที่อยู่กลางภาพ

จากรูปที่ 5.3 รูป (a) แสดงผลของการประมวลผลในขั้นตอนที่ 1 จะเห็นว่าจุดที่ถูกลบจะเป็นจุดที่อยู่ที่ขอบด้านขวา, ด้านล่างและจุดของมุมบนด้านซ้าย รูป (b) เป็นผลที่ได้จากการประมวล ในขั้นตอนที่ 2 จะเห็นว่า จุดที่ถูกลบจะเป็นจุดที่ขอบด้านซ้าย, ด้านบนและมุมล่างด้านซ้าย รูป (c) คือโครงร่างของภาพที่ได้ เมื่อทำการทำซ้ำๆ ในขั้นตอนที่ 1 และ 2 จนไม่มีจุดใดที่ถูกต้องตามเงื่อนไข



รูปที่ 5.3 (a) ผลจากขั้นตอนที่ 1 ในการประมวลผลรอบแรก
(b) ผลจากขั้นตอนที่ 2 ในการประมวลผลรอบแรก
(c) ผลสุดท้ายจากการประมวลผล

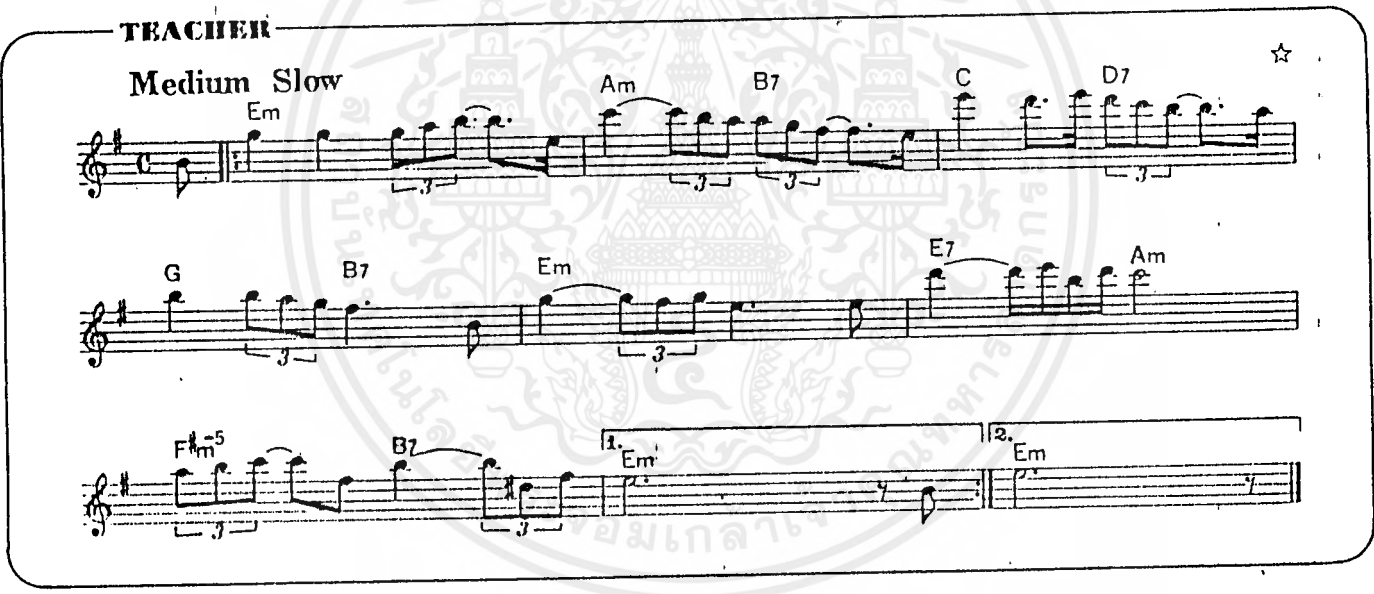
5.5 การลบบรรทัด 5 เส้นออกจากภาพ

หลังจากที่เราอ่านภาพลงสู่หน่ายความจำ ,Fillter ภาพ และ แปลงข้อมูลให้เป็นแบบไบนารี ก่อนที่เราจะไปสู่ขั้นตอนการหาโครงร่างของภาพนั้น เราจะต้องทำการลบบรรทัด 5 เส้น ออกจากภาพก่อน เพื่อแยกโน๊ตออกจากกัน และให้ได้ลักษณะของโน๊ตที่แท้จริง ดังรูปที่ 5.4 เป็นโน๊ตเพลงดนตรีสากล เราจะสังเกตเห็นได้ว่า ในภาพประกอบด้วยเส้นตรงตามแนวนอนที่มีขนาดต่างๆ กัน หลายขนาด เช่น

เส้นโยงเสียง , เส้นที่เป็นส่วนประกอบของตัวเบย์จ และเส้นที่เป็นบรรทัด 5 เส้นเอง สำหรับบรรทัดที่ต้องการลบมี 2 ขนาด คือ เส้นยาวปกติที่ต้องมีการเขียนโน๊ตอยู่แล้ว และเส้นที่จะมีเมื่อมีการเขียนโน๊ตที่มี Octave สูงหรือต่ำกว่าบรรทัด 5 เส้น (เรียกว่า เส้นน้อย)

ทางหนึ่งที่เป็นไปได้คือ เราจะต้องกำหนดช่วงความยาวของเส้นที่ต้องการจะลบ เมื่อได้ทราบว่าเส้นนั้นเป็นเส้นที่ต้องการจะลบจริง เช่น เรากำหนดให้บรรทัดที่กำหนดเป็นเส้นยาวมากกว่า 300 Pixel และบรรทัดที่เป็นเส้นสั้น มีขนาดมากกว่า 5 Pixel และ น้อยกว่า 10 Pixel ซึ่งขนาดที่กำหนดจะต้อง กำหนดให้เหมาะสม จึงจะทำให้การลบเส้นเป็นไปอย่างถูกต้อง จากนั้นทำการลบเส้นที่ต้องการ โดยการลบก็จะต้องเฉพาะส่วนที่เป็นเส้น แต่ไม่ลบส่วนของตัวโน๊ตที่มีเส้นตัดผ่าน

ในขั้นตอนที่จะต้องจดจำตำแหน่งทางแนวตั้งของบรรทัด 5 เส้นไว้ด้วย เพื่อนำไปใช้ในการตรวจสอบระดับเสียงของตัวโน๊ต ในขั้นตอนสุดท้าย



รูปที่ 5.4 โน๊ตเพลงดนตรีสากล

31

5.6 การเข้ารหัส

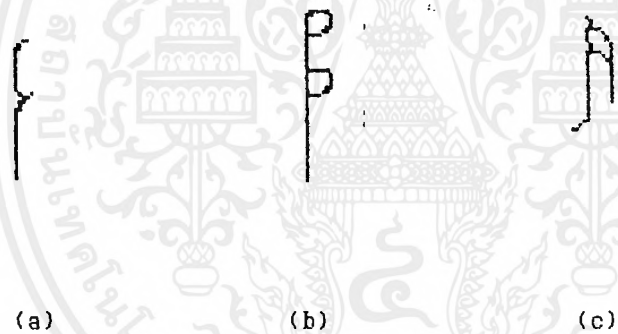
เมื่อเราได้โครงร่างของตัวโน๊ตแล้ว เราจะแยกตัวโน๊ตออกเป็นส่วนย่อยๆ ซึ่งตัวโน๊ตจะประกอบด้วยส่วนย่อยๆ ดังนี้คือ ส่วนที่เป็นเส้นตรงทางแนวตั้ง ,ส่วนตัวของตัวโน๊ตและถ้าเป็นตัวเขบ็จจะประกอบด้วยหางของเขบ็จด้วย แต่ส่วนที่เป็นเส้นตรงทางแนวตั้งจะไม่นำมาพิจราณา เพราะมีในทุกตัวเพราะฉะนั้นส่วนที่จะนำมาพิจารณา คือ ส่วนหัวของตัวโน๊ตและหางของเขบ็จ

ส่วนหัวของตัวโน๊ต ประกอบด้วย

1. ตัวของตัวขาว
2. หัวของตัวดำหรือตัวเขบ็จ

ในที่นี้เราจะแทนค่า ส่วนย่อยของตัวโน๊ต ดังนี้ คือ

1. หัวตัวขาว = 0
2. หัวตัวดำหรือตัวเขบ็จ = 1
3. หางเขบ็จ = 2



(a) (b) (c)

รูปที่ 5.5

จากรูปที่ 5.5 (a) เมื่อทำการแยกส่วนย่อยแล้ว จะประกอบด้วยค่า "1" 2 ค่า ซึ่งจะตีความได้ว่า ประกอบด้วย หัวดำ 2 หัว ทำให้เราทราบว่า ชุดข้อมูลนี้ประกอบด้วยตัวดำ 2 ตัว

จากรูปที่ 5.5 (b) จะประกอบด้วยค่า "0" 2 ค่า ซึ่งตีความได้ว่า ประกอบด้วยตัวขาว 2 ตัว ทำให้ทราบว่า ชุดข้อมูลนี้ประกอบด้วยตัวขาว 2 ตัว

จากรูปที่ 5.5 (c) จะประกอบด้วยค่า "1" 1 ตัว และค่า "2" 2 ตัว จะได้ว่าประกอบด้วยหัวดำ 1 หัวและ หางเขบ็จ 2 หาง ทำให้ทราบว่า ชุดข้อมูลชุดนี้เป็นตัวเขบ็จ 2 ชั้น

บทที่ 6
การทดลองและผลที่ได้

6.1 อุปกรณ์ฮาร์ดแวร์ หรือ MIDI CARD

การทำงานของอุปกรณ์ฮาร์ดแวร์หรือ MIDI CARD สามารถใช้งานได้ผลเป็นที่น่าพอใจเพราะสามารถเป็นตัวกลางในการส่งรหัสมิดี้จากเครื่องคอมพิวเตอร์ไปควบคุมเครื่องดนตรีได้อย่างถูกต้องและอีกอย่างหนึ่งที่ทดลองได้ผลเป็นที่น่าพอใจก็คือ การทำจังหวะของตัวทำจังหวะ(Timer/Counterในการ์ด) ที่สามารถควบคุมระยะเวลาในการส่งตัวโน๊ตจากตัวแรกไปจนถึงตัวโน๊ตตัวถัดๆไป ตามค่าจังหวะที่กำหนดให้

6.2 การทดลองโปรแกรม COMPILE FILE และการเข้ารหัสจากการรอ่านตัวโน๊ต

จากการทดลองในการเขียนรหัสจากการอ่านตัวโน๊ตลงใน Editor ที่ทำให้เราเข้าใจได้ง่ายขึ้นนี้สามารถที่จะให้รายละเอียดได้ในระดับหนึ่ง กล่าวคือ สามารถเล่นคอร์ดได้ (การเล่นโน๊ตมากกว่าหนึ่งโน๊ตออกมาพร้อมกัน) และสามารถให้รายละเอียดในเครื่องหมายโยงเสียง

6.3 โปรแกรมในส่วนที่ติดต่อกับผู้ใช้

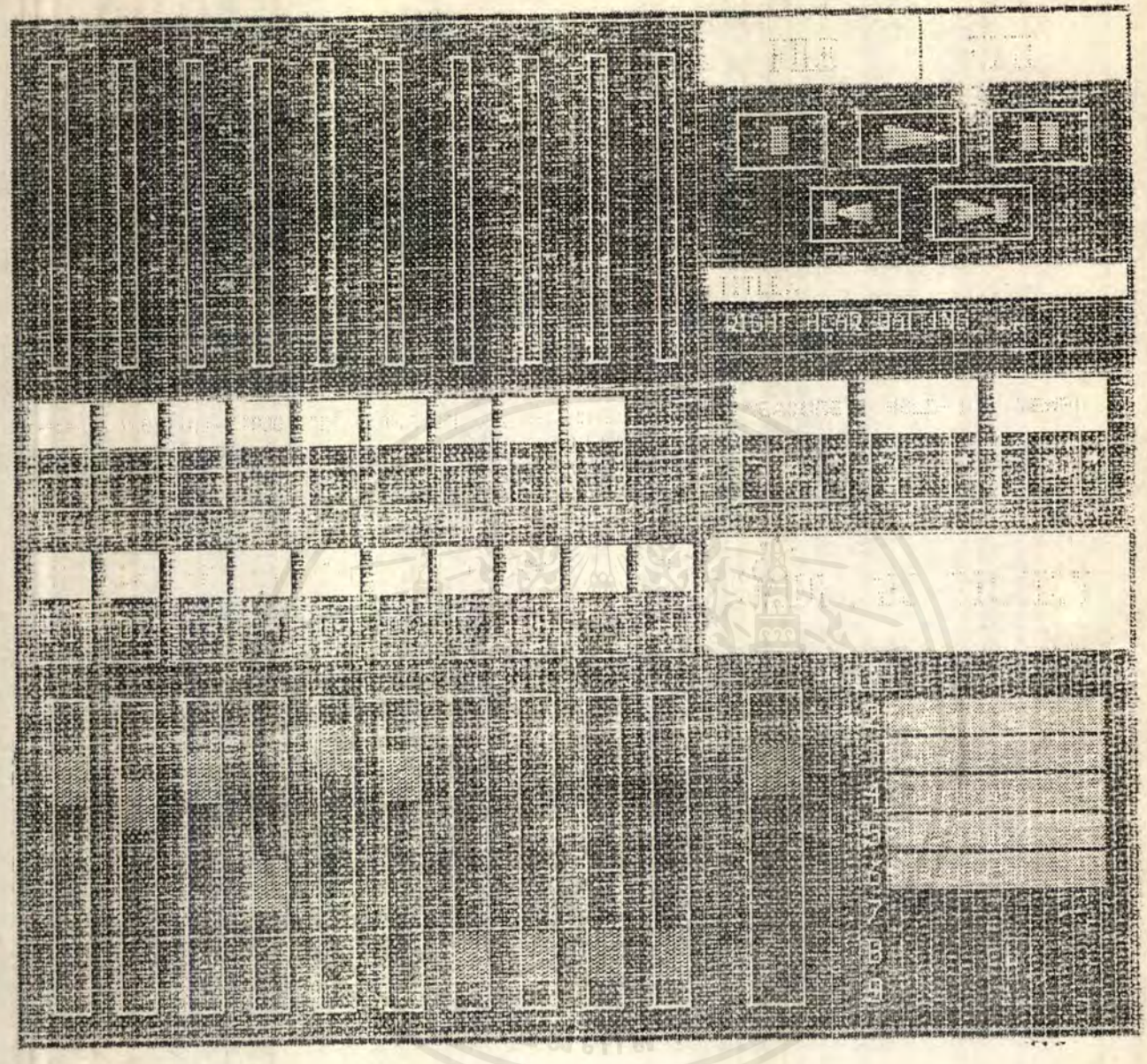
ในการทดลองครั้งนี้ ได้มีการเขียนโปรมแกรมในโหมดกราฟิคส์ โดยมีการวาดหน้าจอให้มีรูปร่างลักษณะคล้ายกับมิกเซอร์ขนาด 10 แชลแนล และได้มีการวาดหน้าจอให้มีปุ่มฟังก์ชั่นต่างๆ เพื่ออำนวยความสะดวกแก่ผู้ใช้หลายปุ่ม เช่น STOP , PLAY , PUSH , FR และ FF นอกจากนั้น ยังได้แสดงถึงอัตราจังหวะ (TEMPO),การเลือกลำโพงซ้ายขวา (PANPOT) และการแสดงให้เห็นถึงลำดับที่ของห้องเพลงว่าอยู่ที่ห้องที่เท่าไร และยังสามารถที่จะเลือกเล่นที่ห้องเพลงลำดับที่เท่าใดก็ได้

เพื่อเป็นการแสดงให้เห็นการเปลี่ยนแปลงของสัญญาณเสียงของแต่ละแชลแนล จึงได้มีการวาดหน้าจอให้มีลักษณะคล้ายไดโอดเปล่งแสง (LED) ติดดับตามสัญญาณเสียงที่ออกทางเอ้าท์พุท

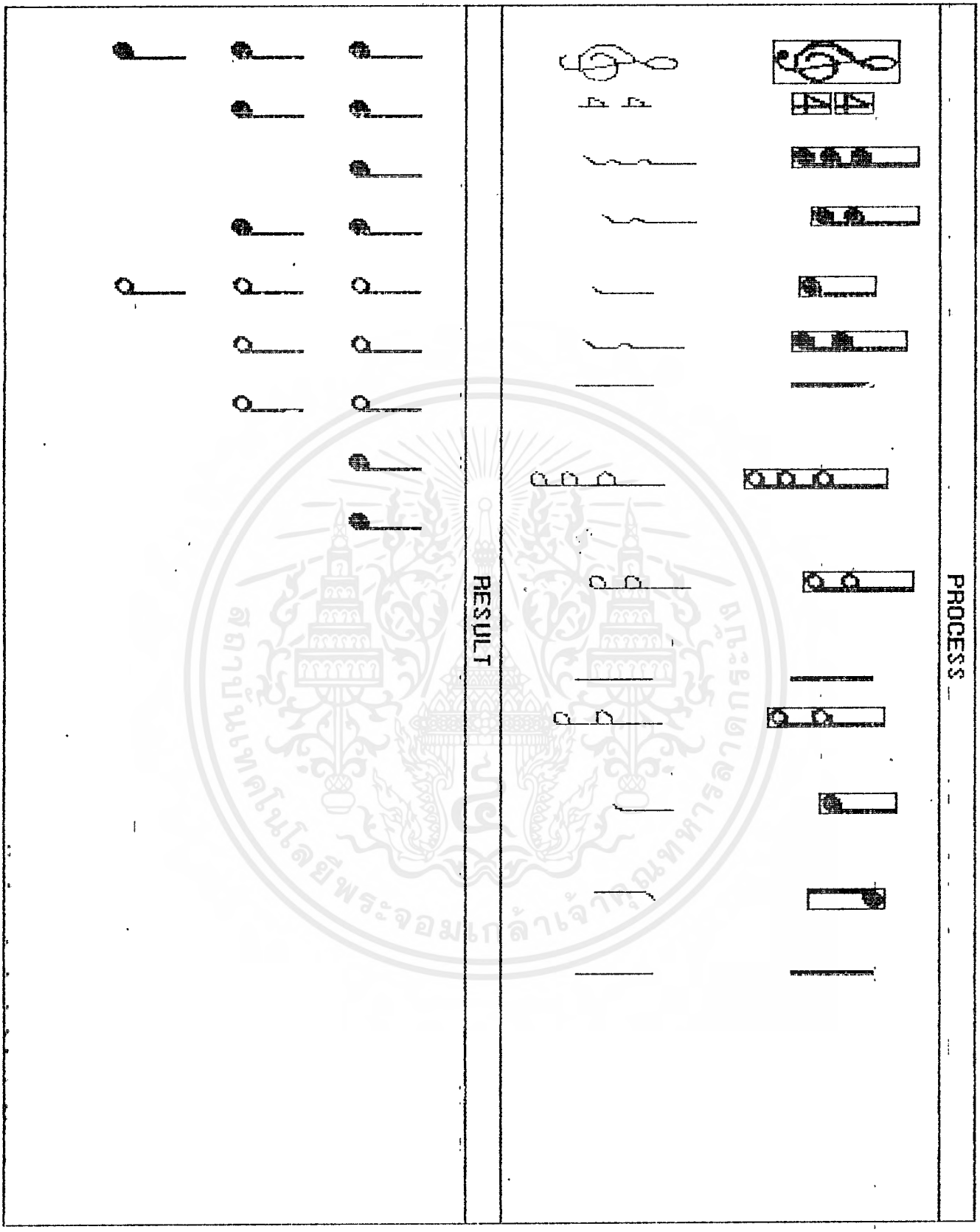
6.4 การทดลองอ่านตัวโน๊ต

ในการทดลองช่วงแรกๆ ปัญหาที่พบคือ โครงร่างของโน๊ตที่ได้ไม่แน่นอน ทำให้การ detect โน๊ตไม่ถูกต้อง เมื่อลองไปตรวจสอบและแก้ไขโปรแกรม ในส่วน Skeleton ปรากฏว่าโปรแกรมในส่วนนี้ผิด จากนั้นทำการแก้ไข จนสามารถหาโครงร่างภาพได้อย่างถูกต้อง และการ detect โน๊ตถูกต้องด้วย

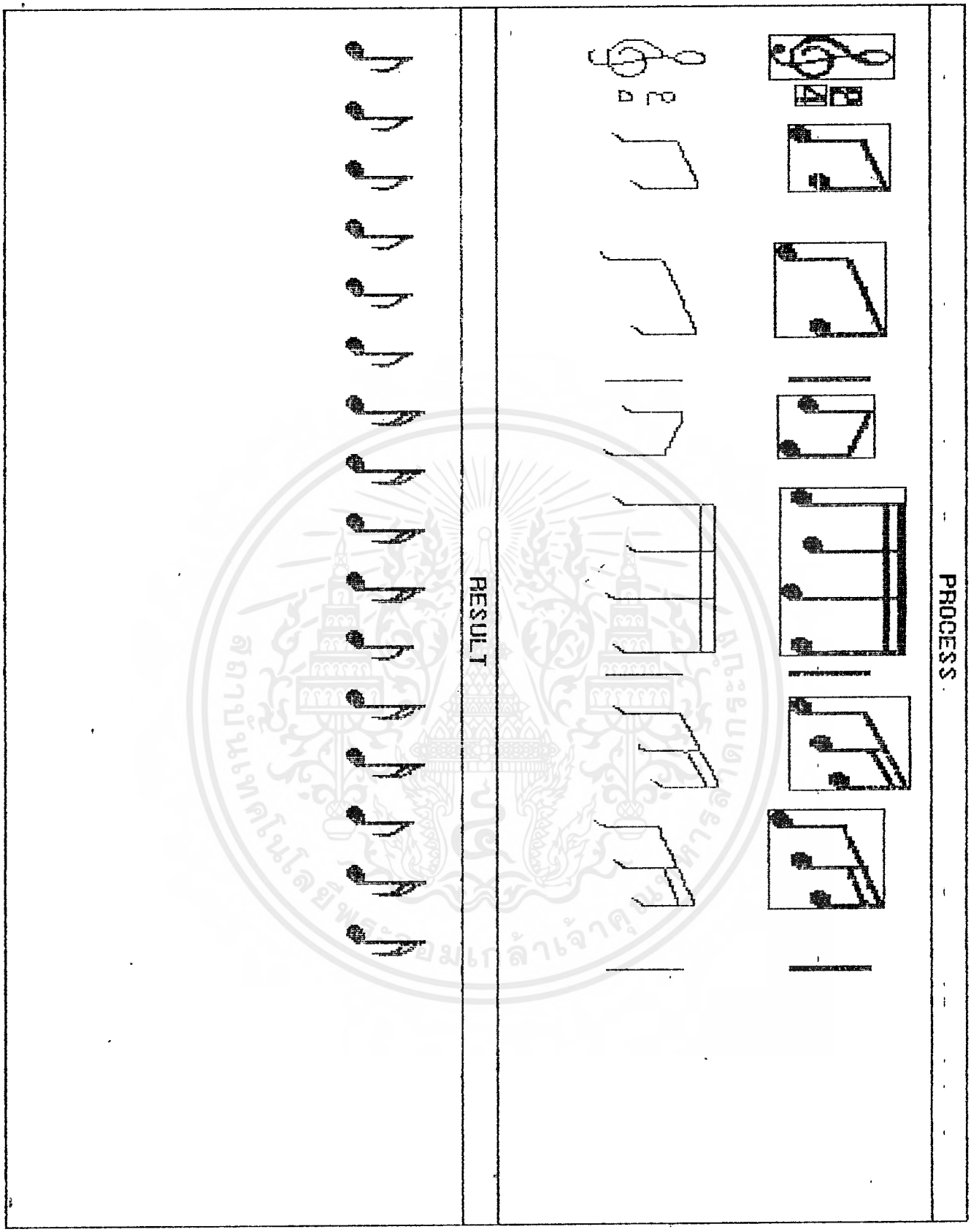
สำหรับในโครงงานนี้ จะสามารถ detect โน๊ตเพลงที่มีส่วนประกอบของเส้นตรงอยู่ได้ เช่น ตัวดำ, ตัวขาว, เขบ็จ 1 ชั้น, เขบ็จ 2 ชั้น และโน๊ตที่เป็นคอร์ด (Chord) แต่ยังไม่สามรถ detect สัญญลักษณ์ ที่นอกเหนือจากสัญญลักษณ์ที่เป็นตัวโน๊ต เช่น กุญแจประจำหลัก, สัญญลักษณ์บอกจังหวะดนตรี, สัญญลักษณ์บอกคอร์ดของกีต้าร์ และโน๊ตที่เขียนในลักษณะคอร์ด โดยมีหัวดำติดกัน ซึ่งทำให้ไม่ทราบว่าโน๊ตนั้น มีหัวดำกี่หัว



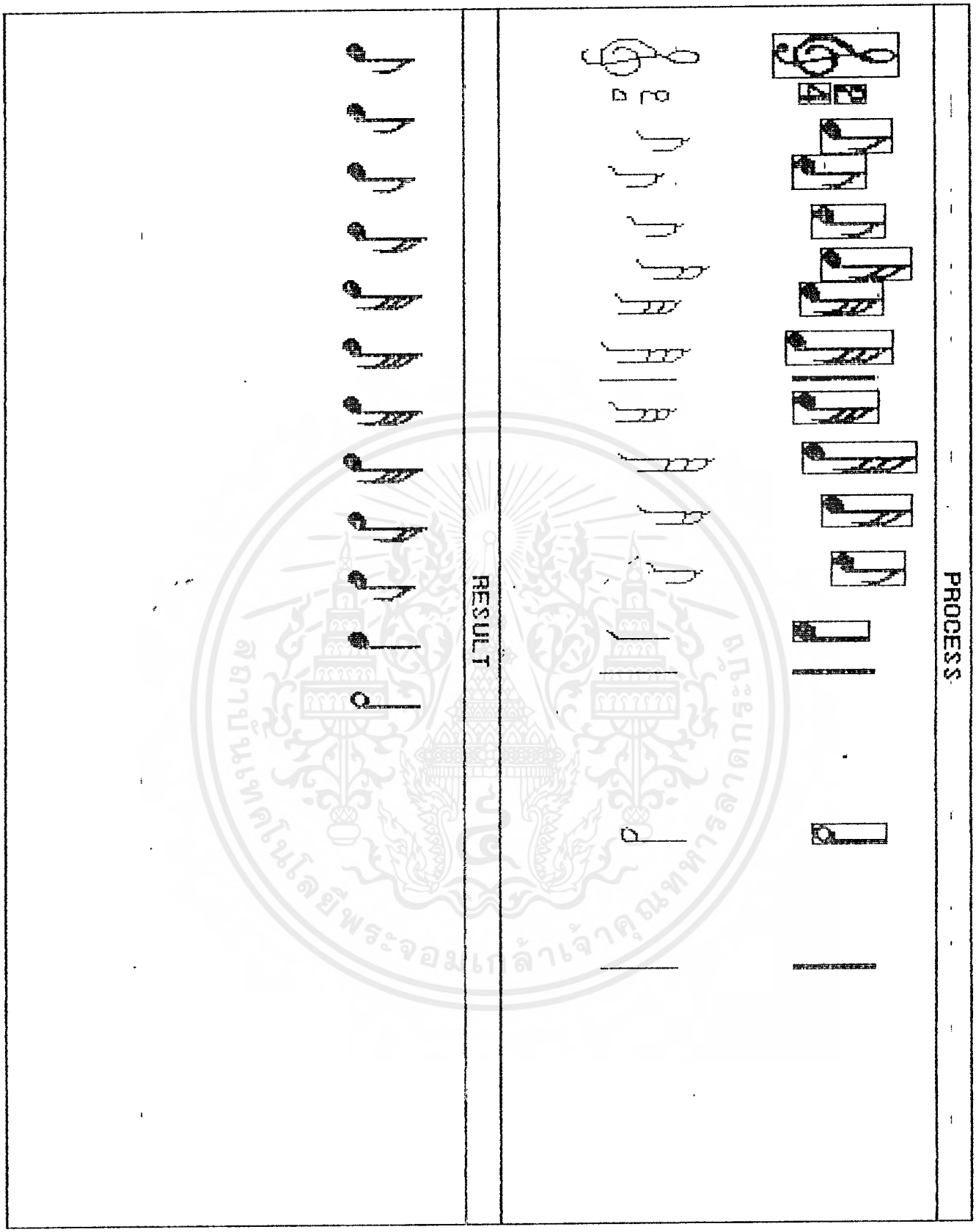
รูปที่ 6.1 แสดงหน้าจอของโปรแกรมส่วนติดต่อกับผู้ใช้



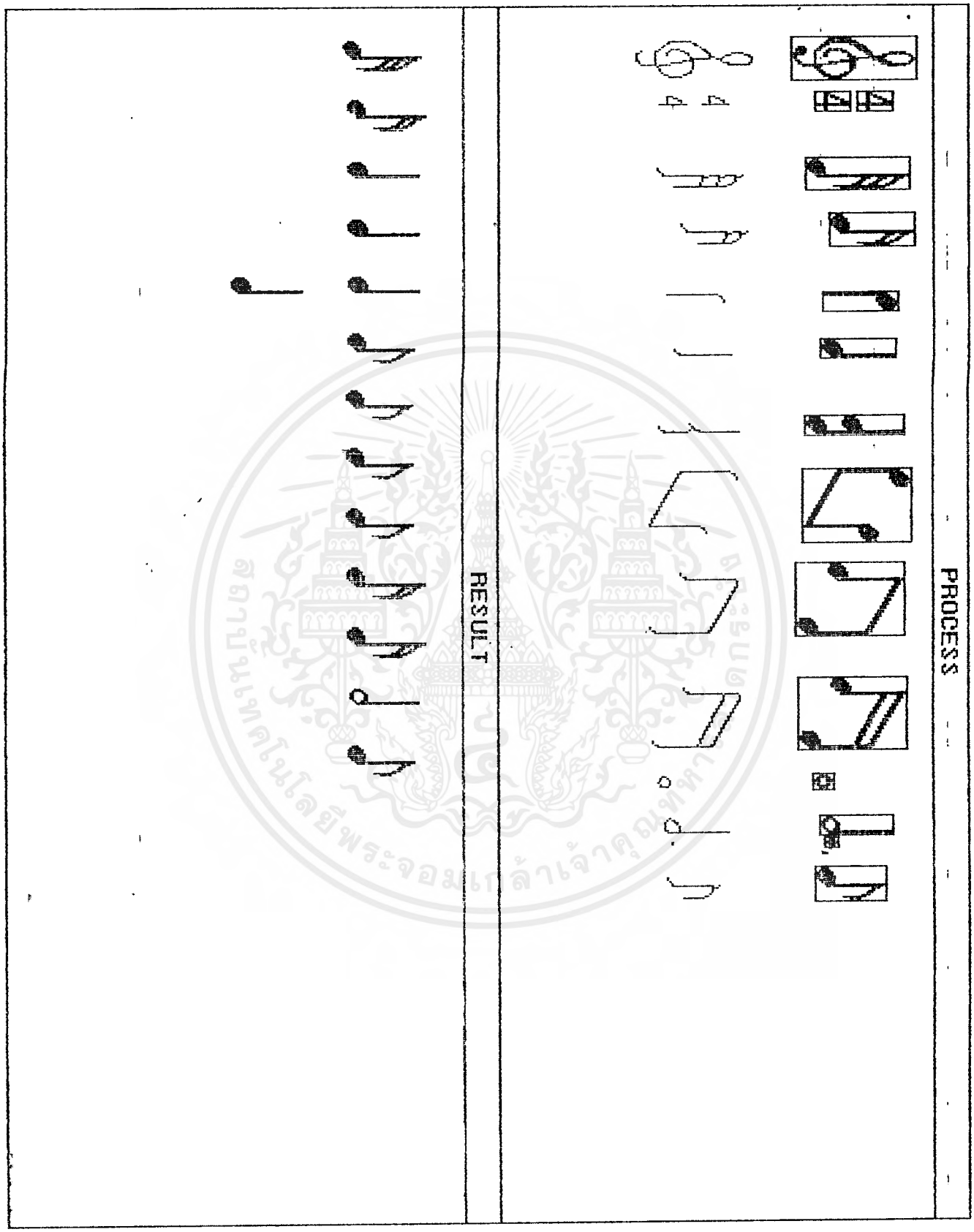
รูปที่ 6.2 แสดงผลการ dectect note จาก file img1.bmp



รูปที่ 6.3 แสดงผลการ dectect note จาก file img2.bmp



รูปที่ 6.4 แสดงผลการ dectect note จาก file img3.bmp



รูปที่ 6.5 แสดงผลการ dectect note จาก file img4.bmp

บทที่ 7

สรุปผล ปัญหา และแนวทางการพัฒนาต่อ

7.1 สรุปผลการทดลอง

-ฮาร์ดแวร์

วงจรในมิดี้การ์ด มีจุดประสงค์หลักในการทำงาน 2 ประการ คือ

1. แปลงรหัสข้อมูลมิดี้แบบขนานจากเครื่องคอมพิวเตอร์ ให้เป็นรหัสข้อมูลแบบอนุกรมแล้วส่งไปควบคุมเครื่องดนตรีอีกทีหนึ่ง
2. เป็นตัวควบคุมจังหวะ กล่าวคือ เป็นการควบคุมระยะเวลาในการส่งตัวโน๊ตจากตัวแรกๆ ไปจนถึงตัวถัดๆไป

จากการทดลอง ก็สามารถที่จะทำให้บรรลุจุดประสงค์ตาม 2 ประการดังกล่าว

-ซอฟแวร์

ซอฟแวร์ที่เขียนขึ้นถึงแม้ว่าจะสามารถทำการส่งรหัสควบคุมได้บรรลุตามวัตถุประสงค์ที่ตั้งไว้ก็ตามแต่ก็ไม่เป็นที่น่าพอใจเท่าใดนักด้วยเหตุที่ว่า ยังไม่สามารถอำนวยความสะดวกต่อผู้ใช้งานได้มากเท่านัก เป็นต้นว่า รหัสข้อมูลที่จะทำการส่งยังไม่สามารถแก้ไขหรือเปลี่ยนแปลงรหัสตัวโน๊ตใหม่ได้เมื่อจะแก้ไขก็ต้องไปแก้ที่ไฟล์โน๊ตเพลง (*.CVI) แล้วคอมไพล์ใหม่อีกรอบหนึ่ง ทำให้เสียเวลาไม่สะดวก

โปรแกรมในส่วน Detect โน๊ตสามารถที่จะ Detect โน๊ตได้ในระดับหนึ่ง คือ Detectโน๊ตได้เป็นตัวๆ บางตัว ซึ่งต้องมีการพัฒนากันต่อไป

7.2 ปัญหา

ปัญหาที่เห็นได้ชัดก็คือ การเชื่อมต่อกันระหว่างภาคแรกกับภาคหลัง เพราะข้อมูลที่อ่านจากกล้องในภาคแรกยังไม่มีผลการทดลองออกมามากนัก จึงทำให้ไม่สามารถจะล่วงรู้ได้ว่า ภาคแรกเก็บข้อมูลแบบใดแล้วในภาคหลังจะนำรหัสข้อมูลมาใช้ได้หรือไม่ แต่อย่างไรก็เป็นที่แน่ใจได้ว่า เมื่อมีข้อมูลรหัสมิดี้แล้วก็สามารถที่จะส่งไปควบคุมการสร้างเสียงของเครื่องดนตรีได้แน่นอน

ปัญหาการจัดเก็บไฟล์ เท่าที่ศึกษาได้ข้อมูลมาค้นพบว่า ได้มีการกำหนดมาตรฐานในการจัดเก็บไฟล์เอาไว้ แต่ก็ไม่เคยเห็นเหมือนกันว่า ได้มีการจัดเก็บเช่นใด เพราะไม่เคยเห็นในหนังสือเล่มใด ซึ่งในตรงส่วนนี้มีความสำคัญมากเช่นกัน เพราะการจัดเก็บไฟล์ที่เป็นมาตรฐานเดียวกันทำให้ใช้ไฟล์ข้อมูลได้กับเครื่องทุกรุ่นทุกยี่ห้อ เนื่องจากว่าในขณะนี้เรายังไม่ทราบว่า มีการจัดเก็บไฟล์มาตรฐานเป็นเช่นไร ดังนั้นในการทดลองของเราในครั้งนี้ เราจึงได้กำหนดรูปแบบการจัดเก็บไฟล์ขึ้นมาเอง จะเห็นได้ว่า ในการใช้งานเราจะสามารถใช้งานได้เฉพาะในการทดลองของเราในครั้งนี้เท่านั้น เพื่อความเป็นมาตรฐาน

ในการทดลองใช้งานในอนาคตควรจะมีการจัดเก็บไฟล์ข้อมูลให้เป็นมาตรฐานด้วย

ปัญหาโปรแกรมในส่วนที่ติดต่อกับผู้ใช้ ในการทดลองได้มีการเขียนโปรแกรมให้ใช้งานในโหมดกราฟิก โดยจำลองรูปคล้ายกับมิกเซอร์ 10 แชลแนล มีปุ่มเลือกใช้งานที่เป็นรูปภาพทำให้สะดวกในการใช้งานได้ในระดับหนึ่ง แต่อย่างไรก็ดีควรจะต้องมีการปรับปรุงโปรแกรมการใช้งานในส่วนนี้ ให้มีฟังก์ชั่นการใช้งานให้มากขึ้นไปกว่านี้

7.3 แนวทางการพัฒนาต่อ

1. พิจารณาหาวิธีเก็บไฟล์และส่งรหัสมิดี้ให้สามารถทำได้สะดวกและแก้ไขได้ง่ายกว่าที่เป็นอยู่นี้ โดยอิงให้เก็บตามมาตรฐานที่มิดี้กำหนดเอาไว้ เพื่อให้สามารถใช้ได้กับเครื่องอื่นทั่วๆไป
2. พัฒนาโปรแกรมส่วนติดต่อกับผู้ใช้ให้สามารถใช้ได้สะดวกขึ้น เช่น การแก้ไขข้อมูลที่อ่านมา, การปรับแต่งเสียง, การเปลี่ยนเครื่องดนตรีที่ใช้เล่น เป็นต้น
3. เนื่องจากโปรแกรมในส่วน Detect โน๊ต การตรวจสอบเส้นบางเส้นยังอาศัยการกำหนดขนาดของเส้นที่จะประมวลผล ทำให้ไม่สมารถที่จะทำการ Detect โน๊ตที่มีขนาดแตกต่างจากภาพที่ใช้ในการทดลองมากไม่ได้ จึงควรที่จะพัฒนาในส่วนนี้ และทำให้โปรแกรมสามรถเช็คเสียง, สัญญลักษณ์อื่นๆ ที่ไม่ใช่ตัวโน๊ตได้, โน๊ตที่เป็นคอร์ดและมีหัวดำอยู่บรรทัดเดียวกัน

7.4 คำแนะนำในการศึกษาและการพัฒนาต่อ

ด้วยเหตุที่ว่ากลุ่มผู้ทำโครงงานมิดี้การ์ด ไม่สามารถที่จะเขียนคำอธิบายที่ได้จากประสบการในการทำโครงงานได้อย่างหมดสิ้น ทางกลุ่มของพวกเราจึงมีความปรารถนาดีแก่ทุกท่านในการที่จะศึกษาทำโครงงานชิ้นนี้ต่อ หรือจะศึกษาเพื่อทำอย่างอื่น ให้สามารถศึกษาได้ภายในระยะเวลาอันสั้น ดังมีแนวทางดังนี้ คือ

1. ควรมีพื้นฐานในการอ่านและการเข้าใจโน๊ตดนตรีสากลเป็นอย่างดี
2. ควรเริ่มศึกษาจากลักษณะการใช้งานอย่างกว้างๆก่อน เป็นต้นว่า มิดี้คืออะไร มีการเชื่อมต่อระหว่างเครื่องดนตรีแต่ละชิ้นเป็นอย่างไร เมื่อทราบเช่นนี้ก็จะทำให้สามารถมองภาพอย่างกว้างๆ ของการสื่อสารข้อมูลตามมาตรฐานมิดี้ได้ เพื่อเราจะได้ศึกษาต่อไปว่า ทำอย่างไรจึงจะสามารถสื่อสารต่อกันระหว่างเครื่องดนตรีได้รู้เรื่อง
3. ควรศึกษามาตรฐานการติดต่อสื่อสารจาก The MIDI 1.0 Specification (ในภาคผนวก) ภายในจะอธิบายถึงมาตรฐานที่กำหนดขึ้นทาง หลักๆ 2 หัวข้อ คือ

-Hardware อธิบายสั้นๆ ดังนี้

เป็นการสื่อสารอนุกรมแบบ Asynchronous มี Start bit และ Stop bit อย่างละ 1 Bit มี Data bitจำนวน 8 bit เป็นต้น

-Data Format อธิบายสั้นๆ ดังนี้

มีการอธิบายถึงรหัสมิดี้ต่างๆ เช่น รหัส Note On, Note Off เป็นต้น ซึ่งมีอธิบาย ค่อนข้างละเอียดในบทต้นๆ

ในการศึกษารหัสมิดี้จะเห็นว่า มีมากมายหลายตัว ไม่ต้องกังวลว่าจะใช้งานอย่างไรหรือใช้เมื่อไร ในขั้นต้นขอให้ทราบแต่เพียงว่า เป็นรหัสอะไร มีค่าเป็นเลขฐานสิบหกหรือฐานสิบเท่าใด ก็เพียงพอ

4. ในการใช้งานจริงนั้นจะเห็นได้ว่าใช้งานไม่ครบทุกรหัส ทั้งนี้ขึ้นอยู่กับ Implementation Chart (ในภาคผนวกมี The MIDI Implementation Chart) ซึ่งเป็นความสามารถเฉพาะเครื่องแต่ละรุ่นว่าจะสามารถใช้รหัสได้ตัวไหนบ้าง ในการใช้งานต้องคำนึงถึงการใช้งานตาม Implementation chart ด้วย
5. ศึกษาจากคู่มือการใช้งานของเครื่อง เพราะในนั้นจะบอกการใช้งานเฉพาะย่อๆ และใช้งานได้เลย หากอ่านแล้วไม่เข้าใจก็ให้ไปอ่านแบบละเอียดในหนังสือที่ให้ไว้ในหนังสืออ้างอิง

ภาคผนวก ก

Z80-SIO Internal Register Descriptions

Z80-SIO Internal Register Descriptions

Appendix

WRITE REGISTERS

The Z80-SIO contains eight registers (WR0-WR7) in each channel that are programmed separately by the system program to configure the functional personality of the channels. With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80-SIO.

Note that the programmer has complete freedom, after pointing to the selected register, of either reading to test the read register or writing to initialize the write register. By designing software to initialize the Z80-SIO in a modular and structured fashion, the programmer can use powerful block I/O instructions.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits (D_0-D_2) to point to WR0.

The basic commands (CMD_0-CMD_2) and the CRC controls (CRC_0, CRC_1) are contained in the first byte of any write register access. This maintains maximum flexibility and system control. Each channel contains the following control registers. These registers are addressed as commands (not data).

WRITE REGISTER 0

WR0 is the command register; however, it is also used for CRC reset codes and to point to the other registers.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

Pointer Bits (D_0-D_2). Bits D_0-D_2 are pointer bits that determine which other write register the next byte is to be written into or which read register the next byte is to be read from. The first byte written into each channel after a reset (either by a Reset command or by the external reset input) goes into WR0. Following a read or write to any register (except WR0), the pointer will point to WR0.

Command Bits (D_3-D_5). Three bits, D_3-D_5, are encoded to issue the seven basic Z80-SIO commands

COMMAND	CMD_2	CMD_1	CMD_0	
0	0	0	0	Null Command (no effect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Enable Interrupt on next Rx Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A)

Command 0 (Null). The Null command has no effect. Its normal use is to cause the Z80-SIO to do nothing while the pointers are set for the following byte

Command 1 (Send Abort). This command is used only with the SDLC mode to generate a sequence of eight to thirteen 1's

Command 2 (Reset External/Status Interrupts). After an External/Status interrupt (a change on a modem line or a break condition, for example) the status bits of RR0 are latched. This command re-enables them and allows interrupts to occur again. Latching the status bits captures short pulses until the CPU has time to read the change.

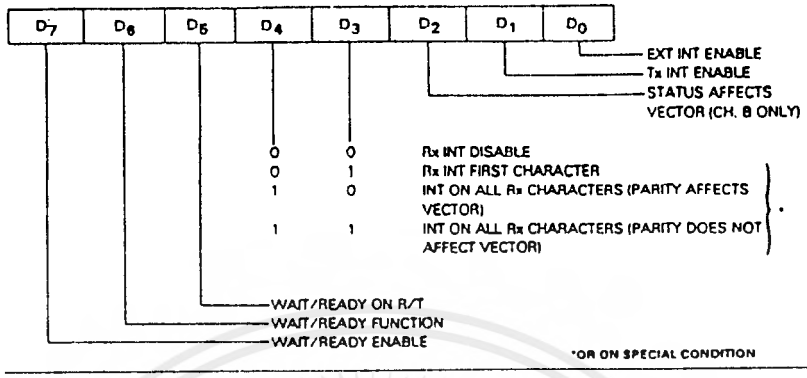
Command 3 (Channel Reset). This command performs the same function as an External Reset, but only on a single channel. Channel A Reset also resets the interrupt prioritization logic. All control registers for the channel must be rewritten after a Channel Reset command.

WRITE REGISTER BIT FUNCTIONS

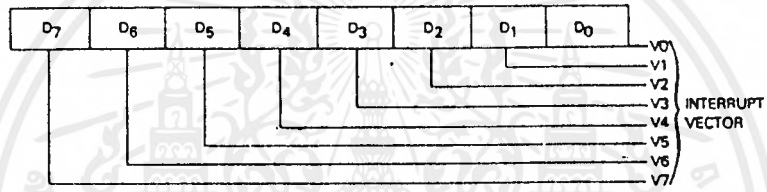
WRITE REGISTER 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
					0	0	0	REGISTER 0
					0	0	1	REGISTER 1
					0	1	0	REGISTER 2
					0	1	1	REGISTER 3
					1	0	0	REGISTER 4
					1	0	1	REGISTER 5
					1	1	0	REGISTER 6
					1	1	1	REGISTER 7
		0	0	0	NULL CODE			
		0	0	1	SEND ABORT (SDLC)			
		0	1	0	RESET EXT/STATUS INTERRUPTS			
		0	1	1	CHANNEL RESET			
		1	0	0	ENABLE INT ON NEXT Rx CHARACTER			
		1	0	1	RESET Tx INT PENDING			
		1	1	0	ERROR RESET			
		1	1	1	RETURN FROM INT (CH-A ONLY)			
0	0	NULL CODE						
0	1	RESET Rx CRC CHECKER						
1	0	RESET Tx CRC GENERATOR						
1	1	RESET Tx UNDERRUN/EOM LATCH						

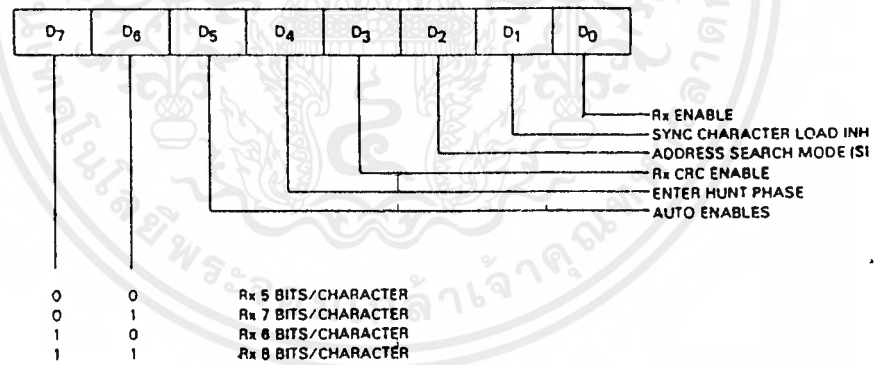
WRITE REGISTER 1



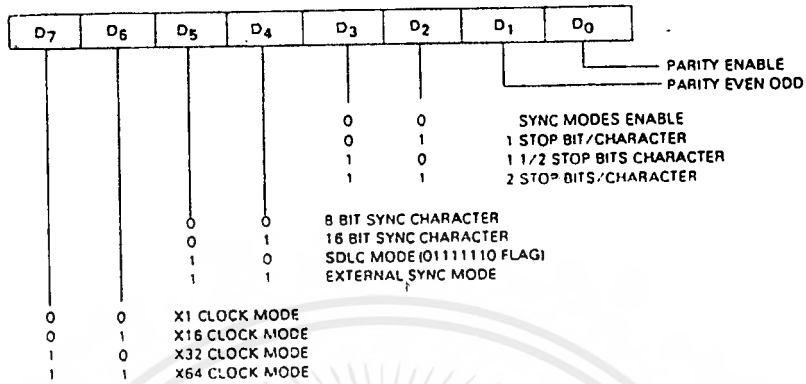
WRITE REGISTER 2 (CHANNEL B ONLY)



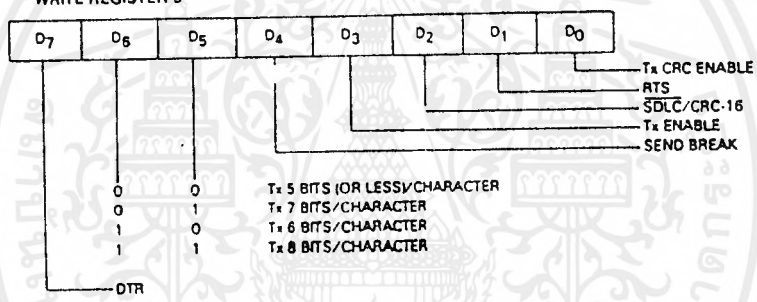
WRITE REGISTER 3



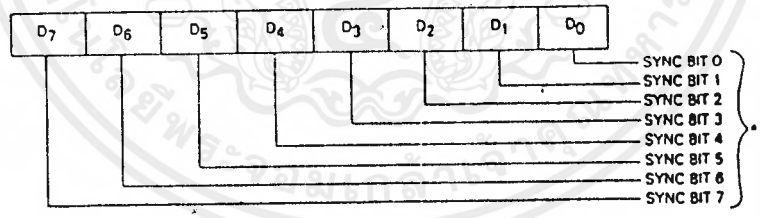
WRITE REGISTER 4



WRITE REGISTER 5

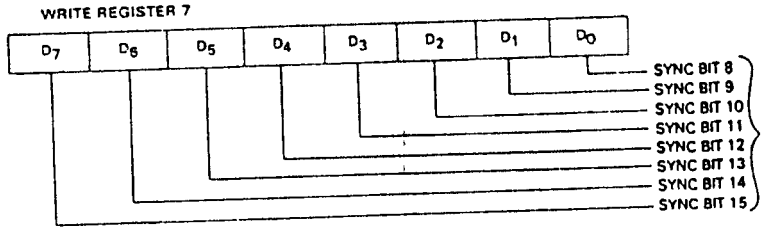


WRITE REGISTER 6



*ALSO SDLC ADDRESS FIELD

WRITE REGISTER 7



*FOR SDLC, IT MUST BE PROGRAMMED TO "01111110" FOR FLAG RECOGNITION

After a Channel Reset, four extra system clock cycles should be allowed for Z80-SIO reset time before any additional commands or controls are written into that channel. This can normally be the time used by the CPU to fetch the next op code.

Command 4 (Enable Interrupt On Next Character). If the Interrupt On First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the Z80-SIO for the next message.

Command 5 (Reset Transmitter Interrupt Pending). The transmitter interrupts when the transmit buffer becomes empty if the Transmit Interrupt Enable mode is selected. In those cases where there are no more characters to be sent (at the end of message, for example), issuing this command prevents further transmitter interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent.

Command 6 (Error Reset). This command resets the error latches. Parity and Overrun errors are latched in RR1 until they are reset with this command. With this scheme, parity errors occurring in block transfers can be examined at the end of the block.

Command 7 (Return From Interrupt). This command must be issued in Channel A and is interpreted by the Z80-SIO in exactly the same way it would interpret a RETI command on the data bus. It resets the interrupt-under-service latch of the highest-priority internal device under service and thus allows lower priority devices to interrupt via the daisy chain. This command allows use of the internal daisy chain even in systems with no external daisy chain or RETI command.

CRC Reset Codes 0 and 1 (D_6 and D_7). Together, these bits select one of the three following reset commands:

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no effect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset Tx Underrun/End Of Message Latch

The Reset Transmit CRC Generator command normally initializes the CRC generator to all 0's. If the SDLC mode is selected, this command initializes the CRC generator to all 1's. The Receive CRC checker is also initialized to all 1's for the SDLC mode.

WRITE REGISTER 1

WR1 contains the control bits for the various interrupt and Wait/Ready modes.

D_7 Wait/Ready Enable	D_6 Wait Or Ready Function	D_5 Wait/Ready On Receive/Transmit	D_4 Receive Interrupt Mode 1
D_3 Receive Interrupt Mode 0	D_2 Status Affects Vector	D_1 Transmit Interrupt Enable	D_0 External Interrupts Enable

External/Status Interrupt Enable (D_0). The External/Status Interrupt Enable allows interrupts to occur as a result of transitions on the $\overline{DCD}$, $\overline{CTS}$ or $\overline{SYNC}$ inputs, as a result of a Break/Abort detection and termination, or at the beginning of CRC or sync character transmission when the Transmit Underrun/EOM latch becomes set.

Transmitter Interrupt Enable (D_1). If enabled, the interrupts occur whenever the transmitter buffer becomes empty.

Status Affects Vector (D_2). This bit is active in Channel B only. If this bit is not set, the fixed vector programmed in WR2 is returned from an interrupt acknowledge sequence. If this bit is set, the vector returned from an interrupt acknowledge is variable according to the following interrupt conditions:

	V_3	V_2	V_1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch B Receive Character Available
	0	1	1	Ch B Special Receive Condition*
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition*

*Special Receive Conditions: Parity Error, Rx Overrun Error, Framing Error, End Of Frame (SDLC).

Receive Interrupt Modes 0 and 1 (D_3 and D_4). Together, these two bits specify the various character-available conditions. In Receive Interrupt modes 1, 2 and 3, a Special Receive Condition can cause an interrupt and modify the interrupt vector.

D_4 Receive Interrupt Mode 1	D_3 Receive Interrupt Mode 0	
0	0	0. Receive Interrupts Disabled
0	1	1. Receive Interrupt On First Character Only
1	0	2. Interrupt On All Receive Characters—parity error is a Special Receive condition
1	1	3. Interrupt On All Receive Characters—parity error is not a Special Receive condition

Wait/Ready Function Selection (D_5-D_7) The Wait and Ready functions are selected by controlling D_5. D_6 and D_7. Wait/Ready function is enabled by setting Wait/Ready Enable (WR1 D_7) to 1. The Ready Function is selected by setting D_6 (Wait/Ready function) to 1. If this bit is 1, the $\overline{WAIT/READY}$ output switches from High to Low when the Z80-SIO is ready to transfer data. The Wait function is selected by setting D_6 to 0. If this bit is 0, the $\overline{WAIT/READY}$ output is in the open-drain state and goes Low when active.

Both the Wait and Ready functions can be used in either the Transmit or Receive modes, but not both simultaneously. If D_5 (Wait/Ready on Receive/Transmit) is set to 1, the Wait/Ready function responds to the condition of the receive buffer (empty or full). If D_5 is set to 0, the Wait/Ready function responds to the condition of the transmit buffer (empty or full).

The logic states of the $\overline{WAIT/READY}$ output when active or inactive depend on the combination of modes selected. Following is a summary of these combinations.

	And D_6 = 1	If D_7 = 0	And D_6 = 0
	$\overline{READY}$ is High		$\overline{WAIT}$ is floating
	And D_5 = 0	If D_7 = 1,	And D_5 = 1
$\overline{READY}$	Is High when transmit buffer is full	$\overline{READY}$	Is High when receive buffer is empty.
$\overline{WAIT}$	Is Low when transmit buffer is full and an SIO data port is selected.	$\overline{WAIT}$	Is Low when receive buffer is empty and an SIO data port is selected.
$\overline{READY}$	Is Low when transmit buffer is empty.	$\overline{READY}$	Is Low when receive buffer is full.
$\overline{WAIT}$	Is floating when transmit buffer is empty	$\overline{WAIT}$	Is Floating when receive buffer is full.

The $\overline{WAIT}$ output High-to-Low transition occurs when the delay time $t_{DIC}(WR)$ after the I/O request. The Low-to-High transition occurs with the delay $t_{DH\Phi}(WR)$ from the falling edge of Φ. The $\overline{READY}$ output High-to-Low transition occurs with the delay $t_{DL\Phi}(WR)$ from the rising edge of Φ. The $\overline{READY}$ output Low-to-High transition occurs with the delay $t_{DIC}(WR)$ after $\overline{IORQ}$ falls.

The Ready function can occur any time the Z80-SIO is not selected. When the $\overline{READY}$ output becomes active (Low), the DMA controller issues $\overline{IORQ}$ and the corresponding B/$\overline{A}$ and C/$\overline{D}$ inputs to the Z80-SIO to transfer data. The $\overline{READY}$ output becomes inactive as soon as $\overline{IORQ}$ and $\overline{CS}$ become active. Since the Ready function can occur internally in the Z80-SIO whether it is addressed or not, the $\overline{READY}$ output becomes inactive when any CPU data or command transfer takes place. This does not cause problems because the DMA controller is not enabled when the CPU transfer takes place.

The Wait function—on the other hand—is active only if the CPU attempts to read Z80-SIO data that has not yet been received, which occurs frequently when block transfer instructions are used. The Wait function can also become active (under program control) if the CPU tries to write data while the transmit buffer is still full. The fact that the $\overline{WAIT}$ output for either channel can become active when the opposite channel is addressed (because the Z80-SIO is addressed) does not affect operation of software loops or block move instructions.

WRITE REGISTER 2

WR2 is the interrupt vector register; it exists in Channel B only. V_4-V_7 and V_0 are always returned exactly as written; V_1-V_3 are returned as written if the Status Affects Vector (WR1, D_2) control bit is 0. If this bit is 1, they are modified as explained in the previous section.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

WRITE REGISTER 3

WR3 contains receiver logic control bits and parameters

D_7	D_6	D_5	D_4
Receiver Bits/ Char 1	Receiver Bits/ Char 0	Auto Enables	Enter Hunt Phase
D_3	D_2	D_1	D_0
Receiver CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

Receiver Enable (D_0). A 1 programmed into this bit allows receive operations to begin. This bit should be set only after all other receive parameters are set and receiver is completely initialized.

Sync Character Load Inhibit (D_1). Sync characters preceding the message (leading sync characters) are not loaded into the receive buffers if this option is selected. Because CRC calculations are not stopped by 'sync character stripping, this feature should be enabled only at the beginning of the message.

Address Search Mode (D_2). If SDLC is selected, setting this mode causes messages with addresses not matching the programmed address in WR6 or the global (11111111) address to be rejected. In other words, no receive interrupts can occur in the Address Search mode unless there is an address match.

Receiver CRC Enable (D_3). If this bit is set, CRC calculation starts (or restarts) at the beginning of the last character transferred from the receive shift register to the buffer stack, regardless of the number of characters in the stack. See "SDLC Receive CRC Checking" (SDLC Receive section) and "CRC Error Checking" (Synchronous Receive section) for details regarding when this bit should be set.

Enter Hunt Phase (D_4). The Z80-SIO automatically enters the Hunt phase after a reset; however, it can be re-entered if character synchronization is lost for any reason (Synchronous mode) or if the contents of an incoming message are not needed (SDLC mode). The Hunt phase is re-entered by writing a 1 into bit D_4. This sets the Sync/Hunt bit (D_4) in RR0.

Auto Enables (D_5). If this mode is selected, $\overline{DCD}$ and $\overline{CTS}$ become the receiver and transmitter enables, respectively. If this bit is not set, $\overline{DCD}$ and $\overline{CTS}$ are simply inputs to their corresponding status bits in RR0.

Receiver Bits/Character 1 and 0 (D_7 and D_6). Together, these bits determine the number of serial receive bits assembled to form a character. Both bits may be changed during the time that a character is being assembled, but they must be changed before the number of bits currently programmed is reached.

D_7	D_6	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

WRITE REGISTER 4

WR4 contains the control bits that affect both the receiver and transmitter. In the transmit and receive initialization routine, these bits should be set before issuing WR1, WR3, WR5, WR6, and WR7.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate 1	Clock Rate 0	Sync Modes 1	Sync Modes 0	Stop Bits 1	Stop Bits 0	Parity Even/$\overline{\text{Odd}}$	Parity

Parity (D_0). If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data. In the Receive mode, the parity bit received is transferred to the CPU as part of the character, unless 8 bits/character is selected.

Parity Even $\overline{\text{Odd}}$ (D_1). If parity is specified, this bit determines whether it is sent and checked as even or odd (1 = even).

Stop Bits 0 and 1 (D_2 and D_3). These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. A special mode (00) signifies that a synchronous mode is to be selected.

D_3 Stop Bits 1	D_2 Stop Bits 0	
0	0	Sync modes
0	1	1 stop bit per character
1	0	1½ stop bits per character
1	1	2 stop bits per character

Sync Modes 0 and 1 (D_4 and D_5). These bits select the various options for character synchronization.

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC mode (01111110 flag pattern)
1	1	External Sync mode

Clock Rate 0 and 1 (D_6 and D_7). These bits specify the multiplier between the clock ($\overline{\text{TxC}}$ and $\overline{\text{RxC}}$) and data rates. For synchronous modes, the x1 clock rate must be specified. Any rate may be specified for asynchronous modes, however, the same rate must be used for both the receiver and transmitter. The system clock in all modes must be at least 5 times the data rate. If the x1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate x1 Clock Rate
0	1	Data Rate x16 Clock Rate
1	0	Data Rate x32 Clock Rate
1	1	Data Rate x64 Clock Rate

WRITE REGISTER 5

WR5 contains control bits that affect the operation of transmitter, with the exception of D2, which affects the transmitter and receiver

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Tx Bits/ Char 1	Tx Bits/ Char 0	Send Break	Tx Enable	CRC-16/ SDLC	RTS	Tx CRC Enable

Transmit CRC Enable (D_0). This bit determines if CRC is calculated on a particular transmit character. If it is set at the time the character is loaded from the transmit buffer into the transmit shift register, CRC is calculated on the character. CRC is not automatically sent unless this bit is set when the Transmit Underrun condition exists.

Request To Send (D_1). This is the control bit for the $\overline{RTS}$ pin. When the $\overline{RTS}$ bit is set, the $\overline{RTS}$ pin goes Low; when reset, $\overline{RTS}$ goes High. In the Asynchronous mode, $\overline{RTS}$ goes High only after all the bits of the character are transmitted and the transmitter buffer is empty. In Synchronous modes, the pin directly follows the state of the bit.

CRC-16/$\overline{SDLC}$ (D_2). This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial ($X^{16} + X^{15} + X^2 + 1$) is used; when reset, the SDLC polynomial ($X^{16} + X^{12} + X^5 + 1$) is used. If the SDLC mode is selected, the CRC generator and checker are preset to all 1's and a special check sequence is used. The SDLC CRC polynomial must be selected when the SDLC mode is selected. If the SDLC mode is not selected, the CRC generator and checker are present to all 0's (for both polynomials).

Transmit Enable (D_3). Data is not transmitted until this bit is set and the Transmit Data output is held marking. Data or sync characters in the process of being transmitted are completely sent if this bit is reset after transmission has started. If the transmitter is disabled during the transmission of a CRC character, sync or flag characters are sent instead of CRC.

Send Break (D_4). When set, this bit immediately forces the Transmit Data output to the spacing condition, regardless of any data being transmitted. When reset, TxD returns to marking.

Transmit Bits/Character 0 and 1 (D_5 and D_6). Together, D_6 and D_5 control the number of bits in each byte transferred to the transmit buffer.

D_6 Transmit Bits/ Character 1	D_5 Transmit Bits/ Character 0	Bits/Character
0	0	Five or less
0	1	7
1	0	6
1	1	8

Bits to be sent must be right justified, least-significant bits first. The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown in the following table

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	D	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
D	0	0	D	D	D	D	D	Sends five data bits

Data Terminal Ready (D_7). This is the control bit for the $\overline{DTR}$ pin. When set, $\overline{DTR}$ is active (Low), when reset, $\overline{DTR}$ is inactive (High).

WRITE REGISTER 6

This register is programmed to contain the transmit sync character in the Monosync mode, the first eight bits of a 16-bit sync character in the Bisync mode or a transmit sync character in the External Sync mode. In the SDLC mode, it is programmed to contain the secondary address field used to compare against the address field of the SDLC frame.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 7	Sync 6	Sync 5	Sync 4	Sync 3	Sync 2	Sync 1	Sync 0

WRITE REGISTER 7

This register is programmed to contain the receive sync character in the Monosync mode, a second byte (last eight bits) of a 16-bit sync character in the Bisync mode and a flag character (01111110) in the SDLC mode. WR7 is not used in the External Sync mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 15	Sync 14	Sync 13	Sync 12	Sync 11	Sync 10	Sync 9	Sync 8

READ REGISTERS INTRODUCTION

The Z80-SIO contains three registers, RR0-RR2 (Figure 7-1), that can be read to obtain the status information for each channel (except for RR2-Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

READ REGISTER 0

This register contains the status of the receive and transmit buffers, the $\overline{DCD}$, $\overline{CTS}$ and $\overline{SYNC}$ inputs, the Transmit Underrun/EOM latch; and the Break/Abort latch.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break Abort	Transmit Underrun/ EOM	$\overline{CTS}$	$\overline{Sync}$/ Hunt	$\overline{DCD}$	Transmit Buffer Empty	Interrupt Pending (Ch A only)	Receive Character Available

Receive Character Available (D_0). This bit is set when at least one character is available in the receive buffer; it is reset when the receive FIFO is completely empty.

Interrupt Pending (D_1). Any interrupting condition in the Z80-SIO causes this bit to be set; however, it is readable only in Channel A. This bit is mainly used in applications that do not have vectored interrupts available. During the interrupt service routine in these applications, this bit indicates if any interrupt conditions are present in all Z80-SIO. This eliminates the need for analyzing all the bits of RR0 in both Channels A and B. Bit D_1 is reset when all the interrupting conditions are satisfied. This bit is always 0 in Channel B.

Transmit Buffer Empty (D_2). This bit is set whenever the transmit buffer becomes empty, except when a CRC character is being sent in a synchronous or SDLC mode. The bit is reset when a character is loaded into the transmit buffer. This bit is in the set condition after a reset.

Data Carrier Detect (D_3). The DCD bit shows the inverted state of the $\overline{DCD}$ input at the time of the last change of any of the five External/Status bits (DCD, $\overline{CTS}$, Sync/Hunt, Break/Abort or Transmit Underrun/EOM). Any transition of the $\overline{DCD}$ input causes the DCD bit to be latched and causes an External/Status interrupt. To read the current state of the DCD bit, this bit must be read immediately following a Reset External/Status Interrupt command.

Sync/Hunt (D_4). Since this bit is controlled differently in the Asynchronous, Synchronous and SDLC modes, its operation is somewhat more complex than that of the other bits and, therefore, requires more explanation.

In Asynchronous modes, the operation of this bit is similar to the DCD status bit, except that Sync/Hunt shows the state of the $\overline{SYNC}$ input. Any High-to-Low transition on the $\overline{SYNC}$ pin sets this bit and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of $\overline{SYNC}$ pin at the time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNC}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External

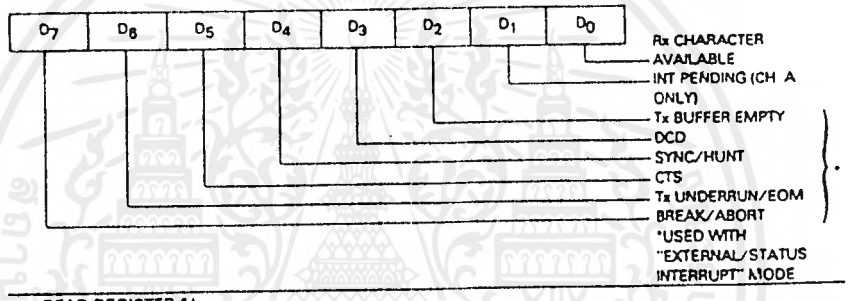
Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{SYNC}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{SYNC}$ input holds the Sync/Hunt status bit in the reset condition.

When external synchronization is achieved, $\overline{SYNC}$ must be driven Low on the second rising edge of $\overline{RxC}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive clock cycles to activate the $\overline{SYNC}$ input. Once $\overline{SYNC}$ is forced Low, it is a good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Refer to Figure 8-6 for timing details. The High-to-Low transition of the $\overline{SYNC}$ input sets the Sync/Hunt bit, which—in turn—sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt command.

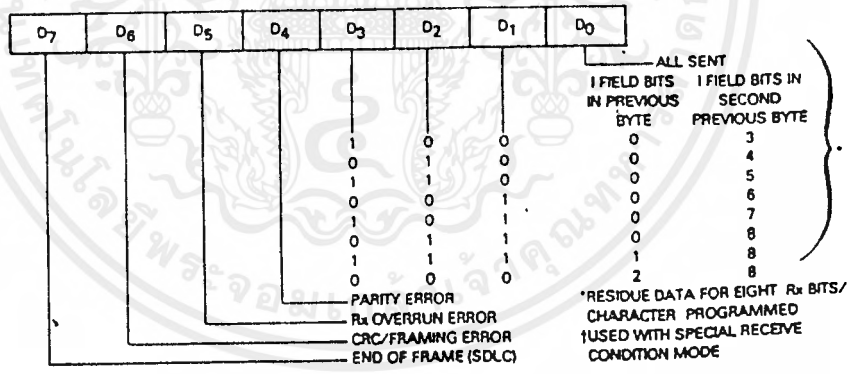
When the $\overline{SYNC}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the Z80-SIO again looks for a High-to-Low transition on the $\overline{SYNC}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the Z80-SIO is waiting for $\overline{SYNC}$ to become active.

READ REGISTER BIT FUNCTIONS

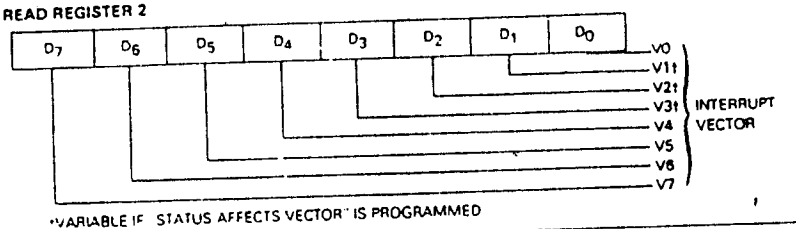
READ REGISTER 0



READ REGISTER 1†



READ REGISTER 2



*VARIABLE IF "STATUS AFFECTS VECTOR" IS PROGRAMMED

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the Z80-SIO establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the Z80-SIO to detect the next transition of other External/Status bits.

When the CPU detects the end of message of that character and synchronization is lost, it sets the Enter Hunt Mode control bit, which—in turn—sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status interrupt, which must also be cleared by the Reset External/Status Interrupt command. Note that the SYNC pin acts as an output in this mode and goes Low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the Z80-SIO. The External/Status interrupt is also generated and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The Z80-SIO automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit or by disabling the receiver.

Clear to Send (D_5). This bit is similar to the DCD bit, except that it shows the inverted state of the $\overline{CTS}$ pin.

Transmit Underrun/End of Message (D_6). This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set, its becoming set causes the External/Status interrupt, which must be reset by issuing the Reset External/Status Interrupt command bits (WR0). This status bit plays an important role in conjunction with other control bits in controlling a transmit operation. Refer to "Bisync Transmit Underrun" and "SDLC Transmit Underrun" for additional details.

Break/Abort (D_7). In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when Break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, CMD_2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break, it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

READ REGISTER 1

This register contains the Special Receive condition status bits and Residue codes for the I-field in the SDLC Receive Mode

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

All Sent (D_0). In Asynchronous modes, this bit is set when all the characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. The bit is always set in Synchronous modes.

Residue Codes 0, 1, and 2 (D_1-D_3). In those cases of the SDLC receive mode where the I-field is not an integral multiple of the character length, these three bits indicate the length of the I-field. These codes are meaningful only for the transfer in which the End Of Frame bit is set (SDLC). For a receive character length of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field Bits In Previous Byte	I-Field Bits In Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

I-Field bits are right-justified in all cases

If a receive character length different from eight bits is used for the I-field, a table similar to the previous one may be constructed for each different character length. For no residue (that is, the last character boundary coincides with the boundary of the I-field and CRC field), the Residue codes are:

Bits per Character	Residue Code 2	Residue Code 1	Residue Code 0
8 Bits per Character	0	1	1
7 Bits per Character	0	0	0
6 Bits per Character	0	1	0
5 Bits per Character	0	0	1

Parity Error (D_4). When parity is enabled, this bit is set for those characters whose parity does not match the programmed sense (even/odd). The bit is latched, so once an error occurs, it remains set until the Error Reset command (WR0) is given.

Receive Overrun Error (D_5). This bit indicates that more than three characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the Error Reset command. If Status Affects Vector is enabled, the character that has been overrun interrupts with a Special Receive Condition vector.

CRC/Framing Error (D_6). If a Framing Error occurs (asynchronous modes), this bit is set (and not latched) for the receive character in which the Framing error occurred. Detection of a Framing Error adds an additional one-half of a bit time to the character time so the Framing Error is not interpreted as a new start bit. In Synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command. The bit is

not latched, so it is always updated when the next character is received. When used for CRC error and status in Synchronous modes, it is usually set since most bit combinations result in a non-zero CRC, except for a correctly completed message.

End of Frame (D_7). This bit is used only with the SDLC mode and indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame.

READ REGISTER 2 (Ch. B Only)

This register contains the interrupt vector written into WR2 if the Status Affects Vector control bit is not set. If the control bit is set, it contains the modified vector shown in the Status Affects Vector paragraph of the Write Register 1 section. When this register is read, the vector returned is modified by the highest priority interrupting condition at the time of the read. If no interrupts are pending, the vector is modified with V_3=0, V_2=1, and V_1=1. This register may be read only through Channel B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
				Variable if Status Affects Vector is enabled			

ภาคผนวก ข

The MIDI 1.0 Specification

The MIDI 1.0 Specification*

Introduction

MIDI is the acronym for Musical Instrument Digital Interface.

MIDI enables synthesizers, sequencers, home computers, rhythm machines, etc. to be interconnected through a standard interface.

Each MIDI-equipped instrument usually contains a receiver and a transmitter. Some instruments may contain only a receiver or transmitter. The receiver receives messages in MIDI format and executes MIDI commands. It consists of an opto-isolator, Universal Asynchronous Receiver/Transmitter (UART), and other hardware needed to perform the intended functions. The transmitter originates messages in MIDI format, and transmits them by way of a UART and line driver.

The MIDI standard hardware and data format are defined in this specification.

Conventions

Status and Data bytes given in Tables I through VI are given in binary.

Numbers followed by an "H" are in hexadecimal.

All other numbers are in decimal.

*MIDI Manufacturers Association, 5316 West 57th Street, Los Angeles, CA. 90056

Hardware

The interface operates at 31.25 (± 1%) Kbaud, asynchronous, with a start bit, 8 data bits (D0 to D7), and a stop bit. This makes a total of 10 bits for a period of 320 microseconds per serial byte.

Circuit: 5 mA current loop type. Logical 0 is current ON. One output shall drive one and only one input. The receiver shall be opto-isolated and require less than 5 mA to turn on. Sharp PC-900 and HP 6N138 opto-isolators have been found acceptable. Other high-speed opto-isolators may be satisfactory. Rise and fall times should be less than 2 microseconds.

Connectors: DIN 5 pin (180 degree) female panel mount receptacle. An example is the SWITCHCRAFT 57 GB5F. The connectors shall be labelled "MIDI IN" and "MIDI OUT." Note that pins 1 and 3 are not used, and should be left unconnected in the receiver and transmitter.

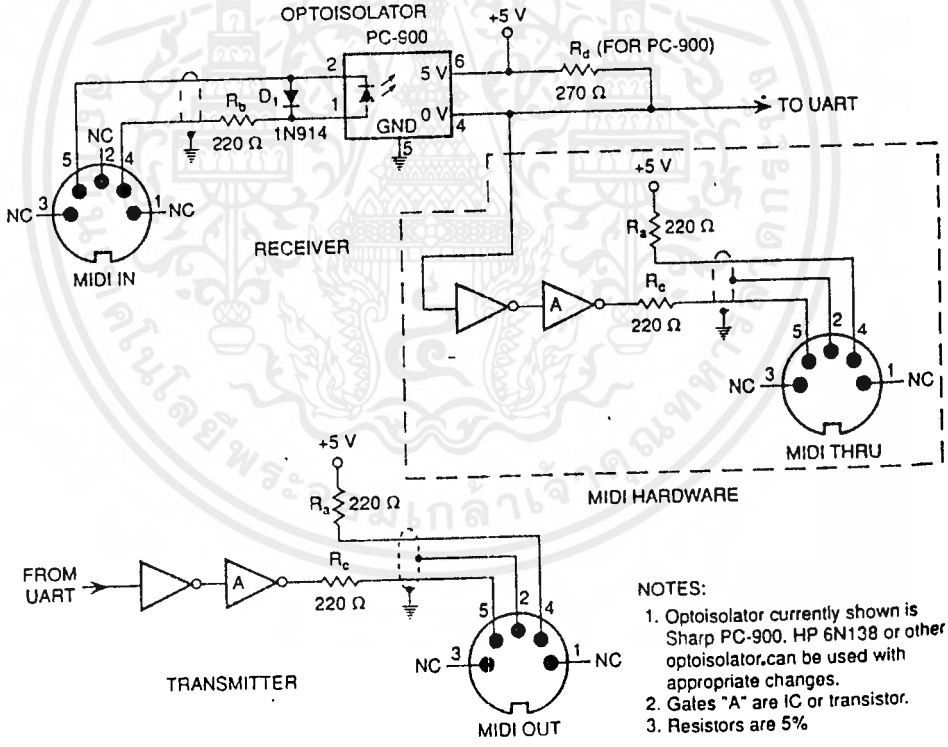


Fig. A-1. Standard hardware configuration for MIDI in, out, and thru ports.

Cables shall have a maximum length of fifty feet (15 meters), and shall be terminated on each end by a corresponding 5-pin DIN male plug, such as the SWITCHCRAFT 05GM5M. The cable shall be shielded twisted pair, with the shield connected to pin 2 at both ends.

A "MIDI THRU" output may be provided if needed, which provides a direct copy of data coming in MIDI IN. For very long chain lengths (more than three instruments), higher-speed opto-isolators must be used to avoid additive rise/fall time errors which affect pulse width duty cycle.

Data Format

All MIDI communication is achieved through multi-byte "messages" consisting of one Status byte followed by one or two Data bytes, except Real-Time and Exclusive messages (see below).

Message Types

Messages are divided into two main categories: Channel and System.

Channel

Channel messages contain a four-bit number in the Status byte which address the message specifically to one of sixteen channels. These messages are thereby intended for any units in a system whose channel number matches the channel number encoded into the Status byte.

There are two types of Channel messages: Voice and Mode.

Voice–To control the instrument's voices, Voice messages are sent over the Voice Channels.

Mode–To define the instrument's response to Voice messages, Mode messages are sent over the instrument's Basic Channel.

System

System messages are not encoded with channel numbers.

There are three types of System messages: Common, Real-Time, and Exclusive.

Common–Common messages are intended for all units in a system.

Real-Time–Real-Time messages are intended for all units in a system.

They contain Status bytes only—no Data bytes. Real-Time messages may be sent at any time—even between bytes of a message which has a different status. In such cases the Real-Time message is either ignored or acted upon, after which the receiving process resumes under the previous status.

Exclusive–Exclusive messages can contain any number of Data bytes, and are terminated by an End of Exclusive (EOX) or any other Status byte. These messages include a Manufacturer's Identification (ID) code. If the receiver does not recognize the ID code, it should ignore the ensuing data.

So that other users can fully access MIDI instruments, manufacturers should publish the format of data following their ID code. Only the manufacturer can update the format following their ID.

Data Types

Status Bytes

Status bytes are eight-bit binary numbers in which the Most Significant Bit (MSB) is set (binary 1). Status bytes serve to identify the message type; that is, the purpose of the Data bytes which follow the Status byte.

Except for Real-Time messages, new Status bytes will always command the receiver to adopt their status, even if the new Status is received before the last message was completed.

Running Status—For Voice and Mode messages only, when a Status byte is received and processed, the receiver will remain in that status until a different Status byte is received. Therefore, if the same Status byte would be repeated, it may (optionally) be omitted so that only the correct number of Data bytes need be sent. Under Running Status, then, a complete message need only consist of specified Data bytes sent in the specified order.

The Running Status feature is especially useful for communicating long strings of Note On/Off messages, where "Note On with Velocity of 0" is used for Note Off. (A separate Note Off Status byte is also available.)

Running Status will be stopped when any other Status byte intervenes, except that Real-Time messages will only interrupt the Running Status temporarily.

Unimplemented Status—Any status bytes received for functions which the receiver has not implemented should be ignored, and subsequent data bytes ignored.

Undefined Status—Undefined Status bytes must not be used. Care should be taken to prevent illegal messages from being sent during power-up or power-down. If undefined Status bytes are received, they should be ignored, as should subsequent Data bytes.

Data Bytes

Following the Status byte, there are (except for Real-Time messages) one or two Data bytes which carry the content of the message. Data bytes are eight-bit binary numbers in which the MSB is reset (binary 0). The number and range of Data bytes which must follow each Status byte are specified in the tables which follow. For each Status byte the correct number of Data bytes must always be sent. Inside the receiver, action on the message should wait until all Data bytes required under the current status are received. Receivers should ignore Data bytes which have not been properly preceded by a valid Status byte (with the exception of "Running Status," above).

Channel Modes

Synthesizers contain sound generation elements called voices. Voice assignment is the algorithmic process of routing Note On/Off data from the keyboard to the voices so that the musical notes are correctly played with accurate timing.

When MIDI is implemented, the relationship between the sixteen available MIDI channels and the synthesizer's voice assignment must be defined. Several Mode messages are available for this purpose (see Table III). They are Omni (On/Off), Poly, and Mono. Poly and Mono are mutually exclusive, i.e., Poly Select disables Mono, and vice versa. Omni, when on, enables the receiver to receive Voice messages in all Voice Channels without discrimination. When Omni is off, the receiver will accept Voice messages from only the selected Voice Channel(s). Mono, when on, restricts the assignment of Voices to just one voice per Voice Channel (Monophonic.) When Mono is off (=Poly On), any number of voices may be allocated by the Receiver's normal voice assignment algorithm (Polyphonic).

For a receiver assigned to Basic Channel "N," the four possible modes arising from the two Mode messages are:

Mode	Omni		
1	On	Poly	Voice messages are received from all Voice Channels and assigned to voices polyphonically.
2	On	Mono	Voice messages are received from all Voice Channels, and control only one voice, monophonically.
3	Off	Poly	Voice messages are received in Voice Channel N only, and are assigned to voices polyphonically.
4	Off	Mono	Voice messages are received in Voice Channels N thru N+M−1, and assigned monophonically to voices 1 thru M, respectively. The number of voices M is specified by the third byte of the Mono Mode Message.

Four modes are applied to transmitters (also assigned to Basic Channel N). Transmitters with no channel selection capability will normally transmit on Basic Channel 1 (N=0).

Mode	Omni		
1	On	Poly	All voice messages are transmitted in Channel N.
2	On	Mono	Voice messages for one voice are sent in Channel N.
3	Off	Poly	Voice messages for all voices are sent in Channel N.
4	Off	Mono	Voice messages for voices 1 thru M are transmitted in Voice Channels N thru N+M−1, respectively. (Single voice per channel.)

A MIDI receiver or transmitter can operate under one and only one mode at a time. Usually the receiver and transmitter will be in the same mode. If a mode cannot be honored by the receiver, it may ignore the message (and any subsequent data bytes), or it may switch to an alternate mode (usually Mode 1, Omni On/Poly).

Mode messages will be recognized by a receiver only when sent in the Basic Channel to which the receiver has been assigned, regardless of the current mode. Voice messages may be received in the Basic Channel and in other channels (which are all called Voice Channels), which are related specifically to the Basic Channel by the rules above, depending on which mode has been selected.

A MIDI receiver may be assigned to one or more Basic Channels by default or by user control. For example, an eight-voice synthesizer might be assigned to Basic Channel 1 on power-up. The user could then switch the instrument to be configured as two four-voice synthesizers, each assigned to its own Basic Channel. Separate Mode messages would then be sent to each four-voice synthesizer, just as if they were physically separate instruments.

Power-Up Default Conditions

On power-up all instruments should default to Mode #1. Except for Note On/Off Status, all Voice messages should be disabled. Spurious or undefined transmissions must be suppressed.

Table I. Summary of Status Bytes

Status D7— D0	# of Data Bytes	Description
		Channel Voice Messages
1000nnnn	2	Note Off event
1001nnnn	2	Note On event (velocity=0: Note Off)
1010nnnn	2	Polyphonic key pressure/after touch
1011nnnn	2	Control change
1100nnnn	1	Program change
1101nnnn	1	Channel pressure/after touch
1110nnnn	2	Pitch bend change
		Channel Mode Messages
1011nnnn	2	Selects Channel Mode
		System Messages
11110000	*****	System Exclusive
11110sss	0 to 2	System Common
11111ttt	0	System Real Time

Notes:

nnnn: N-1, where N = Channel #,
i.e., 0000 is Channel 1.
0001 is Channel 2.
.
.
.
1111 is Channel 16.

*****: 0iiiiiii, data, ..., EOX

iiiiiii: Identification

sss: 1 to 7

ttt: 0 to 7

Table II. Channel Voice Messages

Status	Data Bytes	Description
1000nnnn	0kkkkkkk 0vvvvvvv	Note Off (see notes 1-4) vvvvvvv: note off velocity
1001nnnn	0kkkkkkk 0vvvvvvv	Note On (see notes 1-4) vvvvvvv - 0: velocity vvvvvvv = 0: note off
1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (After-Touch) vvvvvvv: pressure value
1011nnnn	0ccccccc 0vvvvvvv	Control Change ccccccc: control # (0-121) (see notes 5-8) vvvvvvv: control value ccccccc = 122 thru 127: Reserved. (See Table III)
1100nnnn	0ppppppp	Program Change ppppppp: program number (0-127)
1101nnnn	0vvvvvvv	Channel Pressure (After-Touch) vvvvvvv: pressure value
1110nnnn	0vvvvvvv 0vvvvvvv	Pitch Bend Change LSB (see note 10) Pitch Bend Change MSB

Notes:

1. nnnn: Voice Channel # (1-16, coded as defined in Table I notes)

2. kkkkkkk: note # (0 - 127)
   kkkkkkk = 60: Middle C of keyboard

0	12	24	36	48	60	72	84	96	108	120	127
		ac	c	c	c	c	c	c	c		
		|——			piano range				——|		

3. vvvvvvv: key velocity
   A logarithmic scale would be advisable.

0	1			64				127
off	ppp	pp	p	mp	mf	f	fff	fff

   vvvvvvv = 64: in case of no velocity sensors
   vvvvvvv = 0: Note Off, with velocity = 64

4. Any Note On message sent should be balanced by sending a Note Off message for that note in that channel at some later time.

5. ccccccc: control number

ccccccc	Description
0	Continuous Controller 0 MSB
1	Continuous Controller 1 MSB (MODULATION BENDER)
2	Continuous Controller 2 MSB
3	Continuous Controller 3 MSB
4-31	Continuous Controllers 4-31 MSB
32	Continuous Controller 0 LSB
33	Continuous Controller 1 LSB (MODULATION BENDER)
34	Continuous Controller 2 LSB
35	Continuous Controller 3 LSB
36-63	Continuous Controllers 4-31 LSB
64-95	Switches (On/Off)
96-121	Undefined
122-127	Reserved for Channel Mode messages (see Table III).

6. All controllers are specifically defined by agreement of the MIDI Manu facturers Association (MMA) and the Japan MIDI Standards Committee (JMSC). Manufacturers can request through the MMA or JMSC that logical controllers be assigned to physical ones as necessary. The controller allocation table must be provided in the user's operation manual.

7. Continuous controllers are divided into Most Significant and Least Signifi cant Bytes. If only seven bits of resolution are needed for any particular controllers, only the MSB is sent. It is not necessary to send the LSB. If more resolution is needed, then both are sent, first the MSB, then the LSB. If only the LSB has changed in value, the LSB may be sent without re-sending the MSB.

8. vvvvvvv: control value (MSB)
(for controllers)

0 ——————————————— 127
min max

(for switches)

0 - - - - - - - - - - 127
off on

Numbers 1 through 126, inclusive, are ignored.

9. Any messages (e.g., Note On), which are sent successively under the same status, can be sent without a Status byte until a different Status byte is needed.

10. Sensitivity of the pitch bender is selected in the receiver. Center position value (no pitch change) is 2000H, which would be transmitted EnH-00H-40H.

Table III. Channel Mode Messages

Status	Data Bytes	Description
1011nnnn	0ccccccc 0vvvvvvv	Mode Messages ccccccc = 122: Local Control vvvvvvv = 0, Local Control Off vvvvvvv = 127, Local Control On ccccccc = 123: All Notes Off vvvvvvv = 0 ccccccc = 124: Omni Mode Off (All Notes Off) vvvvvvv = 0 ccccccc = 125: Omni Mode On (All Notes Off) vvvvvvv = 0 ccccccc = 126: Mono Mode On (Poly Mode Off) (All Notes Off) vvvvvvv = M, where M is the number of channels. vvvvvvv = 0, the number of channels equals the number of voices in the receiver. ccccccc = 127: Poly Mode On (Mono Mode Off) vvvvvvv = 0 (All Notes Off)

Notes:

1. nnnn: Basic Channel # (1-16, coded as defined in Table I)

2. Messages 123 thru 127 function as All Notes Off messages. They will turn off all voices controlled by the assigned Basic Channel. Except for message 123, All Notes Off, they should not be sent periodically, but only for a specific purpose. In no case should they be used in lieu of Note Off commands to turn off notes which have been previously turned on. Therefore any All Notes Off command (123-127) may be ignored by receiver with no possibility of notes staying on, since any Note On command must have a corresponding specific Note Off command.

3. Control Change #122, Local Control, is optionally used to interrupt the internal control path between the keyboard, for example, and the sound-generating circuitry. If 0 (Local Off message) is received, the path is disconnected: the keyboard data goes only to MIDI and the sound-generating circuitry is controlled only by incoming MIDI data. If a 7FH (Local On message) is received, normal operation is restored.

4. The third byte of "Mono" specifies the number of channels in which Monophonic Voice messages are to be sent. This number, "M", is a number between 1 and 16. The channel(s) being used, then, will be the current

Basic Channel (=N) thru N+M−1 up to a maximum of 16. If M=0, this is a special case directing the receiver to assign all its voices, one per channel, from the Basic Channel N through 16.

Table IV. System Common Messages

Status	Data Bytes	Description
11110001		Undefined
11110010		Song Position Pointer
	0lllllll	lllllll: (Least significant)
	0hhhhhhh	hhhhhhh: (Most significant)
11110011	0sssssss	Song Select
		sssssss: Song #
11110100		Undefined
11110101		Undefined
11110110	none	Tune Request
11110111	none	EOX: "End of System Exclusive" flag

1. Song Position Pointer: Is an internal register which holds the number of MIDI beats (1 beat = 6 MIDI clocks) since the start of the song. Normally it is set to 0 when the START switch is pressed, which starts sequence playback. It then increments with every sixth MIDI clock receipt, until STOP is pressed. If CONTINUE is pressed, it continues to increment. It can be arbitrarily preset (to a resolution of 1 beat) by the SONG POSITION POINTER message.
2. Song Select: Specifies which song or sequence is to be played upon receipt of a Start (Real-Time) message.
3. Tune Request: Used with analog synthesizers to request them to tune their oscillators.
4. EOX: Used as a flag to indicate the end of a System Exclusive transmission (see Table VI).

Table V. System Real Time Messages

Status	Data Bytes	Description
11111000		Timing Clock
11111001		Undefined
11111010		Start
11111011		Continue
11111100		Stop
11111101		Undefined
11111110		Active Sensing
11111111		System Reset

Notes:

1. The System Real Time messages are for synchronizing all of the system in real time.

2. The System Real Time messages can be sent at any time. Any messages which consist of two or more bytes may be split to insert Real Time messages.

3. Timing clock (F8H)
   The system is synchronized with this clock, which is sent at a rate of 24 clocks/quarter note.

4. Start (from the beginning of song) (FAH)
   This byte is immediately sent when the PLAY switch on the master (e.g., sequencer or rhythm unit) is pressed.

5. Continue (FBH)
   This is sent when the CONTINUE switch is hit. A sequence will continue at the time of the next clock.

6. Stop (FCH)
   This byte is immediately sent when the STOP switch is hit. It will stop the sequence.

7. Active Sensing (FEH)
   Use of this message is optional, for either receivers or transmitters. This is a "dummy" Status byte that is sent every 300 ms (max), whenever there is no other activity on MIDI. The receiver will operate normally if it never receives FEH. Otherwise, if FEH is ever received, the receiver will expect to receive FEH or a transmission of any type every 300 ms (max). If a period of 300 ms passes with no activity, the receiver will turn off the voices and return to normal operation.

8. System Reset (FFH)
   This message initializes all of the system to the condition of just having turned on power. The system Reset message should be used sparingly, preferably under manual command only. In particular, it should not be sent automatically on power up.

Table VI. System Exclusive Messages

Status	Data Bytes	Description
11110000		Bulk dump etc.
	0iiiiiii	iiiiiii: identification
	(0*******)	Any number of bytes may be sent here, for any purpose, as long as they all have a zero in the most significant bit.
	(0*******)	
	11110111	EOX: "End of System Exclusive"

Notes:

1. iiiiiii: identification ID (0-127)
2. All bytes between the System Exclusive Status byte and EOX or the next Status byte must have zeroes in the MSB.
3. The ID number can be obtained from the MMA or JMSC.
4. In no case should other Status or Data bytes (except Real-Time) be interleaved with System Exclusive, regardless of whether or not the ID code

ภาคผนวก ค

The MIDI Implementation Chart

The MIDI Implementation Chart

It is not always necessary for a MIDI device to transmit or receive every type of MIDI message that is defined by the MIDI specification. Certain messages may not relate to the function of a device. For example, it is only required of a synth module that it respond to note-on/off messages since there is no built-in keyboard controller for transmitting them. Other devices might limit various MIDI messages due to such factors as design limitations or cost-effectiveness. For instance, should a velocity-sensitive keyboard controller be used to control a synthesizer that does not respond to velocity messages, no amount of keyboard banging will result in a change in volume. As a result, the velocity message is simply ignored by the synthesizer.

To insure that two or more MIDI devices will be able to communicate MIDI events effectively, the MMA and the JMSC have devised the *MIDI Implementation Chart* (Fig. B-1), which relates all of the MIDI capabilities of a specific MIDI device to the user at a glance using a standardized printed format.

From the user's standpoint, when considering a new piece of equipment, it is always wise to compare its implementation chart with other devices within the existing MIDI system. This will ensure that the device will recognize existing messages and/or add to the capabilities of the current system.

Guidelines for Using the Chart

The MMA specifies that the MIDI implementation chart be printed the same size using a standardized spreadsheet format consisting of 4 columns by 12 rows. The first column lists the MIDI function in question. The second lists information relating to whether (or how) the device transmits this function's data. The third lists whether (or how) the device recognizes (receives) this data, and the final column is used for additional remarks by the manufacturer.

- *Note off*: Indicates whether the device is capable of transmitting and responding to variable release velocity messages. Many devices use a message (note-on velocity = 0) to indicate a note-off condition. This is often indicated in the chart by *9NH v=0* or *$9n 00*, which is the hexadecimal equivalent for this message.

After Touch

After touch indicates how pressure data is transmitted or received. The subheadings for this function are *key's* and *ch's*.

- *Key's*: This indicates if the device will transmit or receive independent polyphonic-pressure messages for each key.
- *Ch's*: Indicates whether the device is capable of transmitting or receiving channel-pressure changes (a common after-touch mode, providing one pressure value for an entire MIDI channel).

Pitch Bender

Pitch bender indicates if the device is capable of transmitting or receiving pitch-bend information. If so, the remarks column will often give information as to the pitch bend range and resolution.

Control Change

Control change indicates whether the device is capable of transmitting or receiving continuous-controller messages. The chart will often list which of these messages are supported in addition to providing a detailed breakdown of their parameters within the remarks column.

Program Change

This category indicates if the device is capable of transmitting or receiving *program-change messages*. *True #* indicates the message numbers that are actually supported by the device's program-change buttons.

System Exclusive

This indicates if the device is capable of transmitting and receiving *system exclusive data*. The remarks column will often give general information as to which type of SysEx data is supported. However, more detailed data will generally be provided within the device's manual.

Despite efforts at standardization, slight inconsistencies within the chart's specifications allow for variations in the symbols, abbreviations, spelling, etc. that can be used by different manufacturers. The following guidelines provide a basic understanding of these differences.

- In general, the symbol O is used to indicate that a MIDI function *is* implemented, while an X is used to show that the function is *not* implemented. However, some charts may use an X to equal a *yes* and an O to equal a *no*. This will usually be indicated within a key at the lower right-hand corner of the chart.
- OX or "*" is used to indicate a selectable function. Further information on the range or type of selectability will be placed within the remarks column.
- MIDI modes are listed as follows:

  Mode 1 (omni on, poly)
  Mode 2 (omni on, mono)
  Mode 3 (omni off, poly)
  Mode 4 (omni off, mono)

  These will often be listed at the bottom of the chart. Occasionally abbreviations of these modes (i.e., omni on/off, omni on or poly) may be used by a manufacturer.

Detailed Explanation of the Chart

The following is a detailed explanation of the various functions and their related categories that are found within the chart.

Header

The *header* provides the user with the model number, brief description, date, and version number of the device.

Basic Channel

Basic channel indicates which MIDI channels are used by the device to transmit and receive data. The subheadings for this function are ***default*** and ***changed***.

- *Default*: This indicates which MIDI channel is in use when the device is first turned on.
- *Changed*: This indicates which of the MIDI channels can be addressed after the device is first turned on.

Mode

Mode indicates which of the MIDI modes may be used by the device. The subheadings for this function are default, messages, and altered.

- *Default*: This indicates which of the four MIDI modes is active when the device is first turned on.
- *Messages*: This describes which of the four MIDI modes can be transmitted or recognized by the device.
- *Altered*: This refers to mode messages which cannot be recognized by the device. It may be followed by a description of the mode that the device automatically enters into upon receiving a request message for an unavailable mode.

Note Number

The transmitted *note number* indicates the range of MIDI note numbers that are transmitted by a device. The maximum possible range spans from 0–127, while 21–108 corresponds to the 88 keys of an extended keyboard controller. Should the note number be greater than the actual number of keys on a keyboard device, a key transposition feature is indicated.

The recognized note number indicates the range of MIDI note numbers that can be recognized by a device. MIDI notes that are out of this range shall be ignored by this device. A second note number range, known as *true voice*, indicates the number of notes the device can actually play. Recognized notes that are out of the actual voice range are transposed up or down in octaves until they fall within this range.

Velocity

This category indicates whether the device is capable of transmitting or receiving attack- and release-velocity messages. The subheadings for this function are *note on* and *note off*.

- *Note on*: This indicates if the device is capable of transmitting and responding to variable-velocity (attack) messages. Not all dynamically controllable devices respond to the full velocity range (1–127). Some devices, such as drum machines, respond to a finite number of velocity steps.

Function ···		Transmitted	Recognized	Remarks
Basic Channel	Default	1 – 16	1 – 16	memorized
	Changed	1 – 16	1 – 16	
Mode	Default	Mode 3, 4	Mode 3	memorized
	Messages	OMNI OFF, MONO POLY	×	
	Alterd	*********		
Note Number		0 – 127	0 – 127	
	True Voice	*********	12 – 108	
Velocity	Note ON	○ v = 1 – 127	○ v = 1 – 127	
	Note OFF	× 9n v = 0	×	
After Touch	Key's	×	×	
	Ch's	×	×	
Pitch Bender		○	○ 0 – 24 semitone	
Control Change	1	○	○	Modulation
	2 – 5	×	×	
	6	**	**	Data Entry MSB
	7	○	○	Volume
	8 – 15	×	×	
	16	×	○	General Purpose Control-1
	17 – 37	×	×	
	38	**	×	Data Entry LSB
	39 – 63	×	×	
	64	×	○	Hold 1
	65 – 80	×	×	
	81	×	○	General Purpose Control-1
	82 – 99	×	×	
	100 – 101	** (0)	** (0)	RPC LSB, MSB
	102 – 120	×	×	
	121	○	○	Reset All Controllers
Prog Change		○ 0 – 127	○ 0 – 127.	
	True #	*********	0 – 127	
System Exclusive		○	○	
System Common	Song Pos	×	×	
	Song Sel	×	×	
	Tune	×	×	
System Real Time	Clock	×	×	
	Commands	×	×	
Aux Message	Local ON/OFF	×	×	
	All Notes OFF	×	○	
	Active Sense	○	○	
	Reset	×	×	
Notes		* Control Change messages from 0 to 95 which are recognized through Control channel are transmitted throgh all the channels which are used in Branches. However, General Purpose Control -1 and General Purpose Control – 6 are converted into the same functions as the FC-100 EV-5 assign and the FC-100 Switch assign in the System Setup, and are transmitted. ** RPC = Registered Parameter Control Number RPC # 0 : Bender Range The value of parameter is to be determined by entering data.		

Mode 1 : OMNI ON, POLY Mode 2 : OMNI ON, MONO ○ : Yes
Mode 3 : OMNI OFF, POLY Mode 4 : OMNI OFF, MONO × : No

Fig. B-1. Example of a MIDI Implementation Chart.

System Common

This indicates whether the device is capable of transmitting or receiving the different types of *system common messages*, such as SPP, MIDI time code, song select, and tune-request messages.

System Real Time

This category indicates whether the device can transmit or receive *system real-time messages*. The subheadings for this function are *clock* and *commands*.

- *Clock*: This refers to the device's ability to receive or transmit MIDI clock messages. A device that can transmit MIDI clock may be used to provide master timing information within a MIDI system, while a device capable of receiving clock data may only be slaved to other MIDI devices.
- *Commands*: This indicates whether the device is capable of transmitting or responding to start, stop, and continue messages.

Aux Messages

This indicates if a device is capable of transmitting or receiving local control-on/off, all notes-off, active-sensing, and system-reset messages.

Notes

This area is used by the manufacturer to comment on any function or implementation particular to the specific MIDI device.

ภาคผนวก ง

MIDI Implementation Chart
of
LA SOUND MODULE
Modle CM - 32L

LA SOUND MODULE
Model CM – 32L

MIDI Implementation Chart

Date : Apr. 18. 1989
Version : 1.00

Function ...		Transmitted	Recognized	Remarks
Basic	Default	×	2 – 10	
Channel	Changed	×	×	
Mode	Default	×	3	
	Messages	×	×	
	Alterd	*********	×	
Note Number		×	0 – 127	
	True Voice	*********	12 – 108	
Velocity	Note ON	×	○ v = 1 – 127	
	Note OFF	×	×	
After Touch	Key's	×	×	
	Ch's	×	×	
Pitch Bender		×	○	
Control Change	1	×	○	Modulation
	2 – 5	×	×	
	6	×	*	Data Entry
	7	×	○	Volume
	8, 9	×	×	
	10	×	○	Pan
	11	×	○	Expression
	12 – 63	×	×	
	64	×	○	Hold 1
	65 – 99	×	×	
	100, 101	×	* (0)	RPN LSB, MSB
	102 – 120	×	×	
	121	×	○	Resets All Controllers
Prog Change		×	○ 0 – 127	
	True #	*********	0 – 127	
System Exclusive		○	○	
System Common	Song Pos	×	×	
	Song Sel	×	×	
	Tune	×	×	
System Real Time	Clock	×	×	
	Commands	×	×	
Aux Message	Local ON/OFF	×	×	
	All Notes OFF	×	○ (123 – 127)	
	Active Sense	×	○	
	Reset	×	×	

Notes

* RPN = Registered Parameter Number
RPN # 0 : Pitch Bend Sensitivity
The value of parameter is to be determined by entering data.

Mode 1 : OMNI ON, POLY
Mode 2 : OMNI ON, MONO
Mode 3 : OMNI OFF, POLY
Mode 4 : OMNI OFF, MONO

○ : Yes
× : No

เอกสารอ้างอิง

1.ซีเอ็ดยูเคชั่น, คู่มือไอซีชิพซัพพอร์ทและหน่วยความจำ ,บริษัทซีเอ็ดยูเคชั่น จำกัด 605 หน้า ,2529

2.ซีเอ็ดยูเคชั่น, คู่มือ/เทียบเบอร์ไอซี TTL ,บริษัทซีเอ็ดยูเคชั่น จำกัด 338 หน้า,2529

3.Roger Powell, The Challenge of Music Software , BYTE ,Vol.11, No.6,1986,pp.145-150

4.Jay Kubicky, A MIDI Project , BYTE , Vol.11 , No.6,1986,pp.199-208

5.James W.coffron, Z-80 Application , SYBEX Inc. ,295 p.,1983

6.David Miles Huber, The MIDI Manual , SAMS , pp. 217-236 ,1990

7.Steve De Furia & Joe Scacciaferro, THE MIDI RESOURCE BOOK , Ferro Technologies , pp. 14-62 ,1988

กิตติกรรมประกาศ

โครงงานนี้คงไม่สามารถดำเนินลุล่วงไปได้ หากปราศจากการสนับสนุนจากบุคคลที่มีพระคุณยิ่งเหล่านี้

1. อาจารย์เกษตร์ ศิริสันติสัมฤทธิ์ อาจารย์ที่ปรึกษา
2. ผู้ช่วยศาสตราจารย์ ดรรชิต ไมตรี อาจารย์ผู้ให้คำปรึกษาเพิ่มเติม
3. อาจารย์เจอดจรีย์ (ขออภัยไม่ทราบนามสกุล) แห่งโรงเรียนดนตรีสยามกลการ สุขุมวิท อาจารย์ผู้สอนกีต้าร์คลาสสิค
4. ท่านพุทธทาสภิกขุและสวนโมกขพลาราม ผู้แต่งหนังสือให้อ่านยามว่างจากการทำโครงงาน
5. เพื่อนๆ สภานักศึกษา องค์การนักศึกษา ที่คอยให้กำลังใจเสมอมา และขอบคุณเพื่อนๆ ชมรมพุทธฯ ทุกคนที่หางานอดิเรกที่ดีมีประโยชน์ให้ทำอย่างต่อเนื่องเสมอมา
6. พี่แพดล นี่ผู้ที่ช่วยหาอุปกรณ์เกินทุกอย่าง ตลอดจนหนังสือดีมีประโยชน์ต่อโครงงาน
7. เพื่อน เกียรติชัย เอื้อมคุณากร, สุปรีดิ วัฒนการุณ และสุรศักดิ์ อุไรรัตน์ ผู้ให้ความช่วยเหลือค้นอุปกรณ์ที่ใช้ในการทดลอง ตลอดจนเครื่องคอมพิวเตอร์ และคำแนะนำดีๆ
8. สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง สำหรับทุกสิ่งทุกอย่างที่ก่อให้เกิดเป็นบัณฑิต

ท้ายที่สุด ขอขอบทุกท่านที่ให้ความช่วยเหลือ แต่ไม่ได้กล่าวนาม