BIBLIOGRAPHY

1) Boonleart Eiamtudsana. " The memory interface card. " Semiconductor Electronics Journal Vol.82 (October-December 1984) : 190-194.

2) Borland International. TURBO PASCAL V.4.0 Owner's Handbook. USA : Borland International, 1987.

3) Bottomley, P.A. " NMR IMAGING TECHNIQUES AND APPLICATIONS: A REVIEW. " Technical Information Series (GENERAL ELECTRIC) pp. 1-21, August, 1981.

4) Chaiwat Laowattanakul. " Robot Arm. " Bachelor's Project, Department of Physics, Faulty of Science, Chulalongkorn Unitversity, 1987.

5) Cho, Z.H. " COMPUTERIZED TOMOGRAPHY. " ENCYCLOPEDIA OF PHYSICAL SCIENCE AND TECHNOLOGY pp. 507-544, Vol. 3, 1987.

6) Edwards, Charles C. Advanced Techniques in Turbo Pascal. Singapore : Tech Publications, 1987.

7) Fisher, Eugene; and Jensen, C.W. PET/CBM and the IEEE 488 Bus (GPIB). 2 nd ed. Berkeiey, California: Osbrone/McGraw-Hill, 1982.

8) General Electric Company (GE). NMR A PERSPECTIVE ON IMAGING. Wisconsin, USA : General Electric Company.

9) Kampel, Ian Practical Design of Digital Circuits. Engiand: Whitstable Litho Ltd., 1983.

10) Kanok Janejirapongvetch. " GPIB (IEEE488) Standard Bus Interface

and its applications part1 : The structure and general applications. " Semiconductor Electronics Journal Vol.78 (May 1987) : 214-218.

11) Kanok Janejirapongvetch. " GPIB (IEEE488) Standard Bus Interface and its applications part2 : The working commands. " Semiconductor Electronics Journal Vol.80 (August 1987) : 196-201.

12) Kanok Janejirapongvetch. " GPIB (IEEE488) Standard Bus Interface and its applications part3 : The polling and the data format. " Semiconductor Electronics Journal Vol.81 (September 1987) : 201-207.

13) Marston, R.M. 110 Integrated Circuit Projects for the Home Constructor. 2nd ed. LONDON: Butterworth & Co (Publisher) Ltd., 1978.

14) Morris, Peter G. Nuclear Magnetic Resonance Imaging in Medicine and Biology. New York : Oxford University Press, 1986.

15) Ongoard Wattanajitsarie. " Pulse Programming Unit. " Bachelor's Project, Department of Physics, Faulty of Science, Chulalongkorn Unitversity, 1988.

16) Philips ECG, Inc. ECG Semiconductor Master Replacement Guide. 13th ed. USA: Philips ECG, Inc., 1985.

17) Pippenger, D.E.; and Tobaben, E.J. Linear and Interface Circuits Applications Vol. 2, USA: Texas Instruments Incorporated, 1985.

18) Poole, Charles P.; Jr.; and Farach, Horacio A. Relaxation in Magnetic Resonance. New York: Academic Press Inc., 1971.

19) Rogers, David F. Procedural Elements for Computers Graphics. 3rd

ed. Singapore : McGraw Hill, Inc., 1988.

20) Rushworth, F.A.; Tunstall, D.P. Nuclear Magnetic Resonance. New York: Gordon and Breach Science Publishers, Inc., 1973.

21) Schildt, Herbert Advanced Turbo Pascal Programming & Techniques. Berkeley, California : Osborne/McGraw-Hill, 1986.

22) Science, Engineering & Education Co. Chip Support and Memory data book. 1st ed. Bangkok : Science, Engineering & Education Co., 1986.

23) Science, Engineering & Education Co. The World TTL, IC DATA & CROSS-REFERENCE GUIDE. 1st ed. Bangkok : Science, Engineering & Education Co., 1986.

24) Science, Engineering & Education Co. Microprocessor data book. 1st ed. Bangkok : Science, Engineering & Education Co., 1986.

25) Slichter, Charles P. Principles of Magnetic Resonance. New York: Harper & Row Publishers, Inc., 1963.

26) Somyot Lohavittayavigran; and Junrachai Rungvichaviwatt. " The 2-way memory. " Semiconductor Electronics Journal Vol.70 (April-May 1986) : 122-128.

27) Stone, Harold S. Microcomputer Interfacing. USA. Addison-wesley Publishing Company, 1982.

28) Taylor, D.G.; Inamdar, R.; and Bushell, M.C. " NMR imaging in theory and in practice. " Phys. Med. Biol. pp. 635-670, Vol. 33, No. 6, 1988.

29) Wiwat Sidhisoradej, "IMAGE PROCESSING FROM NMR SIGNAL," Master thesis, Department of Physics Graduate school Chulalongkorn University, 1989.

APPENDIX

Appendix A

IEEE 488 (GPIB)

The basic concept of IEEE std 488 or General Purpose Interface Bus (GPIB) is that it is electrical/mechanical interfacing standard whereby an instrument manufactured by one company can be interconnected with any other instrument of another company conforming to the same electrical interface.

The GPIB has been painstakingly specified in its electrical, mechanical and communication operation. This standard bus and its interconnecting cabling provide a means for the mini- or microcomputer user to interconnect a computer controller with a variety of instruments and equipment and to program the entire system for optimum operation. (Fisher, Jensen 1982: 4)

The GPIB consists of two groups of lines (see Fig.A-1). One group (a) consists of eight data lines, each carrying one bit of data. The other group (b) has eight control/management lines. The eight bidirection GPIB data lines carry parallel, simultaneous bits transferred between the controller and a selected device on the GPIB, or between any talker and listener. The eight GPIB control/management lines supply information as it is needed immediately before, during, or after the transmission of data. (Fisher, Jensen 1982: 9)

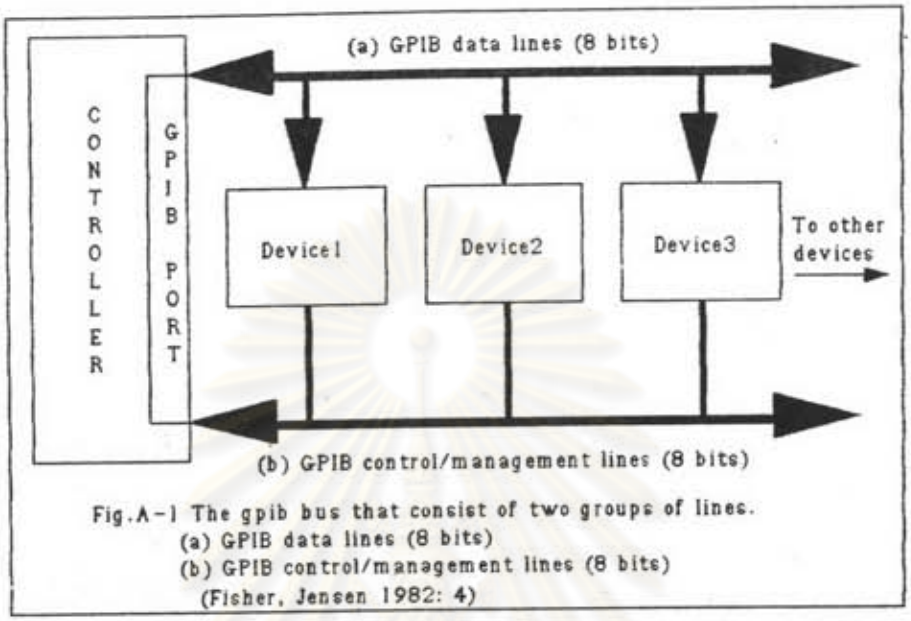


Fig.A-1 The gpib bus that consist of two groups of lines.
(a) GPIB data lines (8 bits)
(b) GPIB control/management lines (8 bits)
(Fisher, Jensen 1982: 4)

The Types of GPIB devices

The types of GPIB devices consist of the talker, the listener and the controller. (Kanok Janejirapongvetch 1987: 1: 216)

Talker - Its function is to send the information. In system that has many talkers, there is active one at some time.

Listener - Its function is to receive the information. In system that has many listeners, there is active one at some time.

Controller - Its function is to control all signals on the bus, by receiving the requirement of a request device, assigning the talker to send information and assigning the listener to receive the information.

The GPIB devices can be provided by the following function:

(1) Talker - Multimeter, ..., etc.
(2) Listener - printer, recorder, ...,

etc.

(3) Talker and Listener - computer, ..., etc.

(4) Talker, Listener and Controller - computer, ..., etc.

Electrical properties of GPIB

Electrical properties is the limitation of the communication of GPIB. (Kanok Janejirapongvetch 1987: 1: 216)

(1) The number of devices (talker, listener and controller) in GPIB bus must not more than 15 devices.

(2) The GPIB bus lines between two adjacent devices must not more than 4 meters, and the length of the total bus lines in the system must not more than 20 meters.

(3) The rate of the transfer data must not more than 1 Mb/sec.

(4) If the number of devices more than half of the system then they must turn on.

Note : If all devices not more than 10 devices then the length of the total bus lines (L) can more than 20 meters, but there are the following relation

$$L \leq 2N \text{ meters}$$

where N is the number of total devices in the system.

The maximum rate of the communication is 1 Mb/sec, in case the DAV line must be active at least t_1 (see Fig.A-2) as data is on DIO_1-DIO_8 lines.

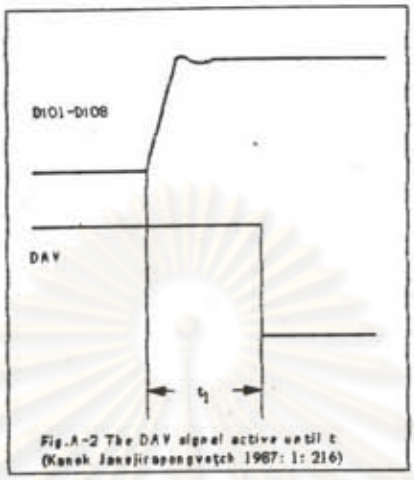


Fig.A-2 The DAV signal active until t
(Kanok Janejirapongvatch 1987: 1: 216)

This time (t_1) is the limiting factor of the maximum rate of the communication by the following :

(1) In case that uses the 48 mA open collector driver, t_1 is much more than 2 μs and if every 2 meters has the standard loads, the maximum rate of the communication in 20 meters must less than 250 Kb/sec.

(2) In case that uses the 48 mA 3-state driver and has the standard loads every 2 meters, t_1 is 500 ns and the maximum rate of communication in 20 meters must less than 500 Kb/sec.

(3) In case that uses the 3-state driver, t_1 is 350 ns and has the standard loads every 1 meter, the maximum rate of communication in 15 meters must less than 1 Mb/sec.

Caution for using GPIB

(1) Do not extend the bus between the connectors, Because it occur the crosstalk effect and increase capacitance in the bus.

(2) Do connect the end of the shield lines (pin #12 in Fig.A-3) to the frame that nears the connector.

(3) Do connect carefully the devices with the cables not have loop. Since the devices which use the switching power supply may generate the noise signal in the bus.

(4) Do not take the noise signal to outside of the bus, by connecting the shield line of the bus to the shielded connectors with the FCC standard metal.

The bus structure

The basic bus structure (Fisher, Jensen 1982: 10-13), IEEE 488 is illustrated in Fig.A-1, of which the bidirectional data bus is along the bottom. To process the information (data flow) on the bidirectional data bus, up to eight control and status signals are required, they form two groups of lines. These two groups are labelled "Interface Management lines" and "Transfer Control lines".

(1) Transfer Control Lines

DAV (Data Valid) The Data Valid line is asserted low (true) by a talker after it places its data on the DIO lines. This tells the listener that the information on the data lines valid.

NRFD (Not Ready For Data) The Not Ready For Data line, when asserted low (true), indicates that not all devices on the GPIB are ready to receive data. Each instrument, in its own time, releases this line. However, the line cannot return to its high (inactive) state until the slowest responding device lets go.

NDAC (Not Data Accepted) The Not Data Accepted line is controlled by the device or devices receiving the data. This line is held low (true) until all the receiving devices (listeners) capture the particular address or data byte; then the line is set high.

(2) Interface Management lines

ATN (Attention) The ATN line is asserted only by the controller during the addressing or command sequence. It can be activated low (true) only by the controller-in-charge to get (as the name implies) the attention of devices residing on the bus. Before sending command on the eight data lines to peripheral devices, the computer (acting as the controller-in-charge) activates this ATN line low (true) to tell all devices that signal levels present (both true and false) on the data bus represent address and control messages for them.

When the ATN line is returned high, only the talker and listeners previously activated take part in the subsequent data exchange.

IFC (Interface Clear) The Interface Clear line transports a reset signal that can be initiated only by the system controller.

When this line is driven low (true), all bus devices are returned or set to their idle (inactive) states.

REN (Remote Enable) The Remote Enable line can be activated only by the system controller. Bus-residing devices will then respond to the controller commands to those of another talker. If this REN line were allowed to become inactive high (false), all bus devices would return to local control.

SRQ (Service Request) The Service Request line is a type of interrupt line, and it can be activated low (true) by any device residing on the bus and needing service from the controller. Such needs for service can, for example, be initiated by a digital meter having a voltage reading available, or a bus device having an internal error that has been detected.

EOI (End Or Identify) The End Or Identify line can be asserted by the controller-in-charge or a bus compatible talker. This line may be pulled low (true) by a talker during the transfer of its last data byte in a multiple-byte message to signal the END of the message. A talker has the option of using EOI. However, EOI is always pulled low (true) by the talker during the transfer of its last data byte onto the DIO lines. By using the same EOI signal line together with the ATN message, the controller-in-charge initiates a parallel poll sequence.

The GPIB line functions are summarized in Table A-1 and the GPIB connector is shown in Fig.A-3.

(3) Data Bus Lines

The GPIB data bus lines (DIO_1-DIO_8) are similar to microprocessor input and output ports. These lines carry 8-bit parallel data to or from the computer.

Table A-1 Functions of the GPIB lines (Fisher and Jensen 1982: 14)

Name of line	Description
ATN	**Attention.** Issued only by controller to gain attention of bus drives before beginning handshake sequence and to denote address/control information on the data bus.
DAV	**Data Valid.** Issued by talker to notify listener that data has been placed on the DIO lines.
DIO_1-DIO_8	**Data Input/Output.** Eight data transfer lines. Also called data bus.
EOI	**End Or Identify.** Issued by talker to notify listener that the data byte currently on the DIO lines is the last one. Issued by controller together with ATN to initiate a parallel poll sequence.
IFC	**Interface Clear.** Issued only by controller to bring all active bus devices to a known state.
NDAC	**Not Data Accepted.** Issued by listener while fetching data from the DIO lines.
NRFD	**Not Ready For Data.** Issued by all listeners. Released by each listener as it becomes ready to accept data.
REN	**Remote Enable.** Issued by controller to control all devices in Local/Remote mode.
SRQ	**Service Request.** Issued by any device needing service from the controller.

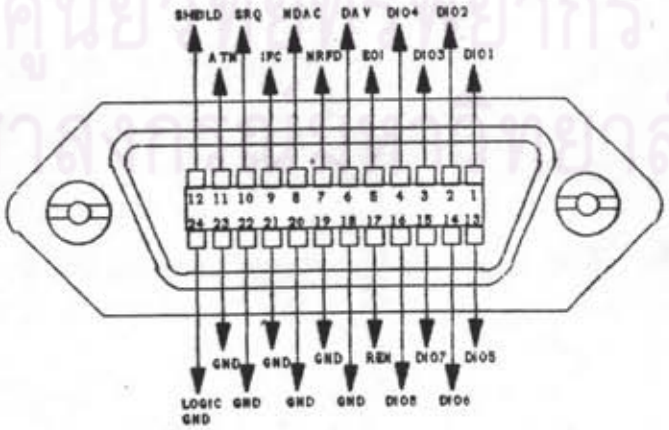


Fig.A-3 The GPIB connector showing pin designations (Fisher, Jensen 1982 : 48)

The GPIB commands

The controller can send the commands to set parameters on all devices in GPIB system. As the ATN line is low the codes on the data bus lines is the command. Similarly, as the ATN line is high the codes on the data bus lines is the data. The GPIB standard commands have 128 commands in Table A-2, provided into 5 groups (Kanok Janejirapongvetch 1987: 2: 199-201) :

(1) The addressed command group The commands in this group will be send to the specified device. These commands are

GTL (Go To Local) : used to control the specified device go to local mode - it can be controlled by manual.

SDC (Selected Device Clear) : used to clear the specified device go to the initial state.

PPC (Parallel Poll Configure) : used to check the status of the specified device by parallel poll (see service request and polling section).

GET (Group Execute Trigger) : used to trigger the specified device to execute.

TCT (Take Control To) : used to assign the specified talker device to the controller.

(2) The universal command group The commands in this group will be send to all devices on the bus. These commands are

LLO (Local Lockout) : used to lockout all devices to state that controlled by manual (local mode).

DCL (Device Clear) : used to clear all devices to the initial state.

PPU (Parallel Poll Unconfigure) : used to cancel the checking of the status of all devices.

SPE (Serial Poll Enable) : used to check the status of all devices by serial poll (see service request and polling section).

(3) The listener address group The commands in this group used to assign the specified device to the listener, which has 0 to 30 addresses and used the UNL (unlisten) command to cancel.

(4) The talker address group The commands in this group used to specified device to the talker, which has 0 to 30 address, and used the UNT (untalk) command to cancel.

All commands above are in the primary command group which is the exact meaning, but the following group is in the secondary command group.

(5) The secondary command group The commands in this group used to set the parameters of each device on the bus to function along the objective of that device, which the same as the button on the front panel of that device. The command group is used after assigning the initial state to all devices.

Table A-2 ASCII-IEEE 488 messages (commands and addresses) hex codes. (Fisher and Jensen 1982: 275)

LSD \ MSD	0		1		2		3		4		5		6		7	
	ASCII	MSG	ASCII	MSG	ASCII	MSG	ASCII	MSG	ASCII	MSG	ASCII	MSG	ASCII	MSG	ASCII	MSG
0	NUL		DLE		SP	00	0	16	@	00	P	16	`	↑	p	↑
1	SOH	GTL	DC1	LLO	!	01	1	17	A	01	Q	17	a	MEANING DEFINED BY PCG CODE	q	MEANING DEFINED BY PCG CODE
2	STX		DC2		"	02	2	18	B	02	R	18	b		r	
3	ETX		DC3		#	03	3	19	C	03	S	19	c		s	
4	EOT	SDC	DC4	DCL	$	04	4	20	D	04	T	20	d		t	
5	ENQ	PPC	NAK	PPU	%	05	5	21	E	05	U	21	e		u	
6	ACK		SYN		&	06	6	22	F	06	V	22	f		v	
7	BEL		ETB		'	07	7	23	G	07	W	23	g		w	
8	BS	GET	CAN	SPE	(	08	8	24	H	08	X	24	h		x	
9	HT	TCT	EM	SPD	)	09	9	25	I	09	Y	25	i		y	
A	LF		SUB		*	10	:	26	J	10	Z	26	j		z	
B	VT		ESC		+	11	;	27	K	11	[	27	k		{	
C	FF		FS		,	12	<	28	L	12	\	28	l		|	
D	CR		GS		-	13	=	29	M	13	]	29	m		}	
E	SO		RS		.	14	>	30	N	14	^	30	n		~	
F	SI		US		/	15	?	UNL	O	15	_	UNT	o	↓	DEL	↓
	ADDRESSED COMMAND GROUP		UNIVERSAL COMMAND GROUP		LISTEN ADDRESS GROUP				TALK ADDRESS GROUP				SECONDARY COMMAND GROUP			

PRIMARY COMMAND GROUP (PCG)

Notes:
1. Device Address messages shown in decimal.
2. Message codes are:
   DCL-Device Clear — LLO-Local Lockout — SDC-Selected Device Clear
   GET-Device Trigger — PPC-Parallel Poll Configure — SPD-Serial Poll Disable
   GTL-Go To Local — PPU-Parallel Poll Unconfigure — SPE-Serial Poll Enable
3. ATN off. Bus data is ASCII; ATN on. Bus data is an IEEE MSG.

Remote/Local

The assignment the device in the system to be controlled by the another one is called "Remote state". Inversely, the device that is controlled by manual (the front panel key operation become valid) is called "Local state". The Remote/Local in the GPIB system have 4 types (Kanok Janejirapongvetch 1987: 2: 200-201) :

(1) LOCS (Local State) is the state that the front panel key operation become valid. This state occur when they is turned on, the REN lines is high or there is the GTL (go to local) command for the specified device.

(2) REMS (Remote State) is the state that can not be controlled by manual, but by the another device. This state occur when the REN lines is low, but except the local switch on that device indicate the local state.

(3) RWLS (Remote With Lockout State) is the state that the same as REMS, but cannot lock by the local switch. This state can be cancelled by the LLO (local lockout) command.

(4) LWLS (Local With Lockout State) is the state that the same as LOCS, but different as receiving the command that assign the listener device, it will change to RWLS immediately. This state occur when the device is in the LOCS, then received the LLO (local lockout) command or when the device is in the RWLS, then received the GTL (go to local) command.

In Fig.A-4, illustrate all changed states, which are the same as the description above.

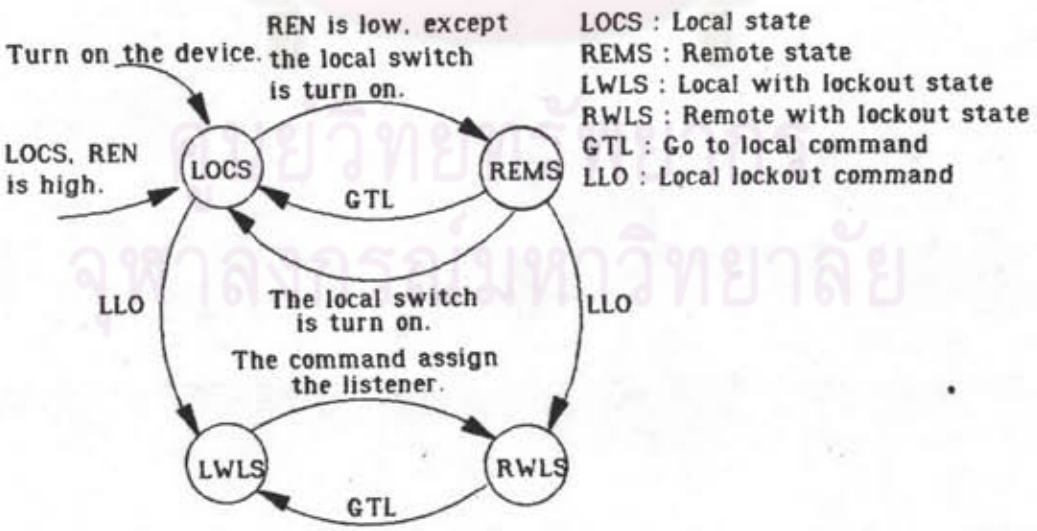


Fig.A-4 All types of Remote/Local and the state changing diagram. (Kanok Janejirapongvetch 1987: 2: 201)

Simplified handshake procedure for data transfer

The entire bus can be visualized as a single communication like between one talker and (at least) one listener. (Fisher and Jensen 1982: 17-19)

To communicate, the talker must let the listener know when data is available on the bus, and the listener must recognize that fact. Also, there must be an agreement as to which device - talker or listener - will activate which control lines.

The three lines that do the communication handshaking between the controller (talker) and the listener are DAV (Data Valid), NRFD (Not Ready For Data), and NDAC (Not Data Accepted). A simplified handshake timing diagram for these three lines is shown in Fig.A-6 and the analogous picture for this communication is shown in Fig.A-5.

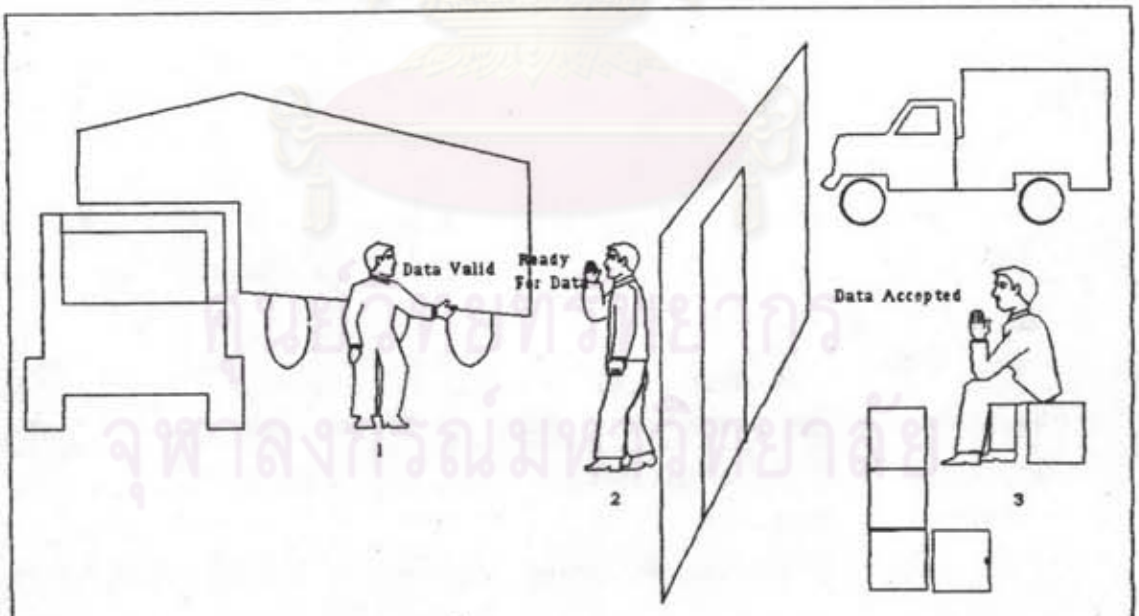


Fig.A-5 The analogous picture for handshake procedure.

The DAV line belongs to the talker; it identifies valid data. The two remaining lines, NRFD and NDAC, belong to the listener. The listener uses NRFD to indicate when it has received and accepted the data.

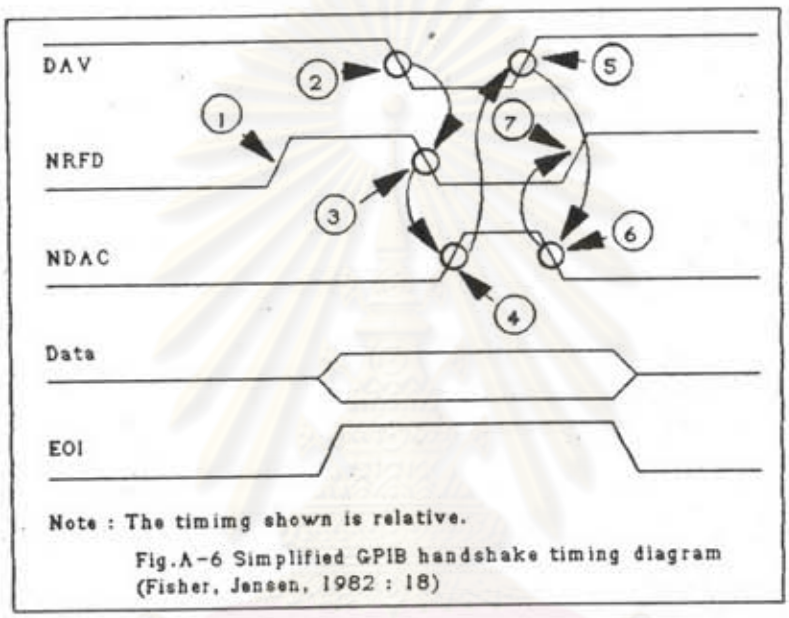


Fig.A-6 Simplified GPIB handshake timing diagram
(Fisher, Jensen, 1982 : 18)

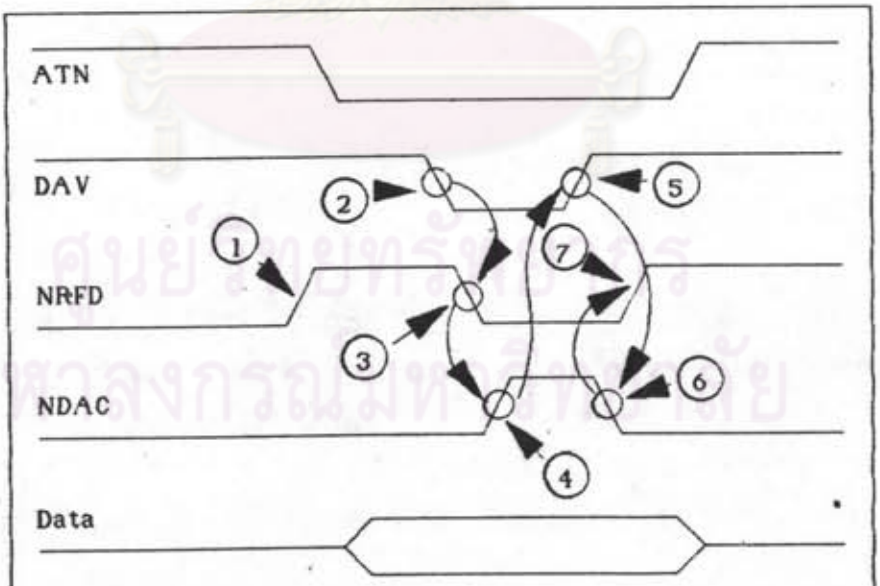
Fig.A-6 illustrates the timing of the handshake as follows:

Assuming that the addressing information has already been transferred down the bus to a listener, the talker knows it is talking, the listener knows it is going to listener, and the listener raises its NRFD line (1). Thus, the listener says that it is now ready to accept data that the talker will send down the bus. The talker then places its data on the 8-bit parallel GPIB data lines. After allowing enough time for the data to settle on the line, the talker drops its DAV lines (2) to a low (true) state, saying that the data is now available for the listener.

Upon sensing DAV low (true), the listener answers by lowering its NRFD line (3) when it is ready to accept data. After the listener has strobed the data into its internal buffers, it raises its NDAC line (4) saying that it has accepted the input data from the bus.

The talker sensing that the NDAC line has been pulled high (false), raises its DAV line (5) to indicate to the listener that the data on the bus is no longer valid. When the listener detects the DAV lines high (false) again,it drop NDAC low (6), acknowledging that data is being removed from the bus, The listener the raises NRFD high (false) (7), saying it is ready for the next data byte to be sent down the bus.

The sequence is now complete. The listener is waiting for the next data byte to be put on the GPIB, and the talker is processing the next data byte before placing it on the bus; it allows for signal propagation delays and processing time required by either talker or listener. If the next data byte is put on the GPIB and the EOI line is high, saying that it is the last data which send on the GPIB. In case of command that has 8 bits is put on the 8-bit parallel GPIB data lines as the same as sending data, but the ATN line is low (true) until end of command sending, that is shown in Fig.A-7.



Note : The timimg shown is relative.

Fig.A-7 The GPIB handshake timing diagram for command sending.

Service Request and polling

When the device occurs error, problem or finishing process, it will send the service request signal (SRQ) to the controller (Kanok Janejirapongvetch 1987: 3: 201-202). In case of the printer, for example, when paper is run out it will send the SRQ signal to the controller, then the controller will check it with

1) Whose the SRQ signal is send ? (If there are many devices in the GPIB system)

2) What is happen ?

This checking will be done by the controller which is the so-called "polling" that has 2 types - serial poll and parallel poll.

When the controller is received the interruption from the requesting device - the SRQ lines is low, it will send the SPE (Serial Poll Enable) command to each device. Each device receives this command will send its working status byte return to the controller. The format of the working status byte is shown in Table A-3.

Bit No.	8	7	6	5	4	3	2	1
Meaning	extend	sending the SRQ line	error	working	x	x	x	x

Note : x is not used (and bit 8, 6, 5 is not also assigned, but the general format is above).

Table A-3 The format of the working status byte for serial poll.

In case of the bus has one device that can request the service. When it request the service, of which the controller is known the objective, the controller not necessary to poll it, except there are the objectives of services requests or devices are more than one.

The following process of the serial poll.

1) The ATN line is pulled low after receiving the SRQ signal.

2) The controller sends the UNL (Unlisten) command on the bus.

3) The controller sends the address of themselves and address of the device that will be polled on the bus, respectively.

4) The controller sends the SPE (Serial Poll Enable) command on the bus and then pulled the ATN line to high and received the working status byte from the device that is being polled. The bit 7 of the working status byte indicates that this polled device is (or not) requesting. If there are one device which requests the service, the SRQ line will change to high. If there are more than one device to request the service and some request devices are already received the service then the SRQ line remains low.

5) The ATN line is pulled low again, and then the controller will send the SPD (Serial Poll Disable) command on the bus.

6) The controller sends the UNT (Untalk) command and UNL (Unlisten) command on the bus, respectively.

If the SRQ line is low, the controller will poll to the another device until the SRQ line return to high. In this last process is controlled by user or program is written by user.

The parallel poll can do rapid more than serial poll. Since each bit on the working status byte for parallel poll represent the bit 7 of each device, so that the controller can poll the eight device in time, if these devices have the function which responds the parallel poll. Typically, the parallel poll is not popular used.

GPIB for NMR imaging

In NMR imaging system used the GPIB system to communicate data between computer and the programmable pulse generator with GPIB card that is inserted in computer IBM AT and controlled by GPIB software

package that is also given. This software is written with assembly language which is called under BASICA. Since NMR imaging used pascal language which is the main language so GPIB software for pascal language is created. To call assembly language directly from pascal language, instead of from BASICA. In this appendix, it will only describe GPIB software package which is modified for pascal language.

1) Using the GPIB software package

A) Preparation

Since the GPIB software package is a resident program, so you must execute it before executing Turbo Pascal V.5.0 by type the command :

GPIB <cr>

where this subdirectory has the following files; GPIB.EXE, IEEE488.EXE and IEEE488.COM.

B) Using the GPIB statements in Turbo Pascal V.5.0.

Before using the GPIB statements, you must use GPIB.TPU by typing the following command at the top of the program file :

uses gpib;

Then you can using GPIB statements.

2) The GPIB statement description.

(1) Gpsys procedure (Gpib system controller)

Declaration Gpsys(adr : word; var successful : boolean)

Remarks adr is a decimal number between 0 to 30 - used to assign the controller to itself, and successful returns true if statement calls successfully, otherwise return false.

Function Initialize IBM PC as the controller, and assign address of IBM PC.

Example

```
uses gpib;
var
   IBM_adr    : word;
   successful : boolean;
begin
   IBM_adr := 0;
   Gpsys(IBM_adr,successful);
   ...
end.
```

(2) Gpnsys procedure (Gpib not system controller)

Declaration Gpnsys(adr : word; var successful : boolean)

Remarks adr is a decimal number between 0 to 30 - used to assign the listener to the talker to itself, and successful returns true if statement calls successfully, otherwise return false.

Function Initialize IBM PC as the talker or the listener, and assign address of IBM PC.

Example

```
uses gpib;
var
   IBM_adr    : word;
   successful : boolean;
begin
   IBM_adr := 0;
```

```
    Gpnsys(IBM_adr,successful);
    ...
end.
```

(3) Gpifc procedure (Gpib interface clear) **Gpsys**

Declaration Gpifc(var successful : boolean)

Remarks successful returns true if statement calls successfully, otherwise return false.

Function Initialize IBM PC as the talker or the listener, and assign address of IBM PC.

Communication the controller ---> all the instruments.

(4) Gpsclr procedure (Gpib specified clear) **Gpsys**

Declaration Gpsclr(lsn : word; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30) and successful returns true if statement call successful, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNT	UNL	listener address	SDC	UNL
Hex	$5F	$3F	$27	$04	$3F
Dec	95	63	39	4	63

{listener address = 7}

Function set the specified listener in the initial state peculiar to the instrument.

Communication the controller ---> the specified instrument.

Example

```
uses gpib;
var
   IBM_adr    : word;
   lsn_adr    : word;
   successful : boolean;
begin
   IBM_adr := 0;
   lsn_adr := 7;
   Gpsys(IBM_adr,successful);
   ...
   Gpsclr(lsn_adr,successful);
   ...
end.
```

(5) Gpaclr procedure (Gpib all clear) **Gpsys**

Declaration Gpaclr(var successful : boolean)

Remarks successful returns true if statement calls successfully, otherwise return false.

Physical result send the following commands.

Meaning	DCL
Hex	$14
Dec	20

Function to set all devices in the GPIB bus to their respective initial state.

Communication the controller ---> all devices.

(6) Gpllo procedure (Gpib local lockout) **Gpsys**

Declaration Gpllo(var successful : boolean)

Remarks successful returns true if statement calls successfully, otherwise return false.

Physical result send the following commands.

Meaning	LLO
Hex	$11
Dec	17

Function to prohibit all devices in the GPIB bus to return to local mode (the front panel key operation becomes invalid).

Communication the controller ---> all devices.

(7) Gpren procedure (Gpib remote enable) Gpsys

Declaration Gpren(var successful : boolean)

Remarks successful returns true if statement calls successfully, otherwise return false.

Physical result to change state on the REN line. If the REN line is high, it will change to low and if it is low, it will not change.

Function to set the all devices in the GPIB bus to the remote mode.

Communication the controller ---> all devices.

(8) Gpsloc procedure (Gpib specified local) Gpsys

Declaration Gpsloc(lsn : word; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30) and successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNT	UNL	listener address	GTL	UNL
Hex	$5F	$3F	$27	$01	$3F
Dec	95	63	39	1	63

{listener address = 7}

Function to set the specified listener to the local mode.

Communication the controller ---> the specified device.

Example
```
uses gpib;
var
   IBM_adr    : word;
   lsn_adr    : word;
   successful : boolean;
begin
   IBM_adr := 0;
   lsn_adr := 7;
   Gpsys(IBM_adr,successful);
   ...
   Gpsloc(lsn_adr,successful);
   ...
end.
```

(9) Gpaloc procedure (Gpib all local) **Gpsys**

Declaration Gpaloc(var successful : boolean)

Remarks successful returns true if statement calls successfully, otherwise return false.

Physical result to change state on the REN line. If the REN line is high, it will change to low and then change to high again. If the REN line is low, it will only change to high.

Function to set all devices in the GPIB bus to the local mode.

Communication the controller ---> all devices.

(10) Gptrg procedure (Gpib trigger) **Gpsys**

Declaration Gptrg(lsn : word; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30) and successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNT	UNL	listener address	GTL	UNL
Hex	$5F	$3F	$27	$08	$3F
Dec	95	63	39	8	63

{listener address = 7}

Function causes the specified listener to start operation.

Communication the controller ---> the specified device.

Example
```
uses gpib;
var
   IBM_adr    : word;
```

```
    lsn_adr    : word;
    successful : boolean;
begin
    IBM_adr := 0;
    lsn_adr := 7;
    Gpsys(IBM_adr,successful);
    ...
    Gptrg(lsn_adr,successful);
    ...
end.
```

(11) Gpwrt procedure (Gpib write) **Gpsys**

Declaration Gpwrt(lsn : word; wrt : text255; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30), wrt is a text255-type expression (text255 type is a string type that not more than 255 characters) and successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNL	listener address	talker address
Hex	$3F	$27	$40
Dec	63	39	64

{listener address = 7 and talker address = 0}

Function to transmit data (keep in wrt parameter) to the specified listener.

Communication the controller ---> the specified listener.

Timeout value 5 seconds.

Example

```
uses gpib;
var
   IBM_adr    : word;
   lsn_adr    : word;
   wrt        : text255;
   successful : boolean;
begin
   IBM_adr := 0;
   lsn_adr := 7;
   wrt := 'gptest';
   Gpsys(IBM_adr,successful);
   ...
   Gpwrt(lsn_adr,wrt,successful); {send wrt to
lsn_adr}
   ...
end.
```

(12) Gpred procedure (Gpib read) **Gpsys**

Declaration Gpred(tlk : word; var red : text255; var successful : boolean)

Remarks tlk is a talker address value (decimal number between 0 to 30), wrt is a text255-type expression (text255 type is a string type that not more than 255 characters), it should be assigned arbitrary literal string to a maximum length user want to receive (as : red = space255). After statement call, user can type red, which contains the data received from the talker. Successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNL	listener address	talker address	receive data
Hex	$3F	$20	$47	
Dec	63	32	71	

{listener address = 7 and talker address = 0}

Function to receive data (keep in red parameter) from the specified talker.

Communication the controller <--- the specified device.

Timeout value 5 seconds

Example

```
uses gpib;
var
   IBM_adr    : word;
   tlk_adr    : word;
   red        : text255;
   successful : boolean;
begin
   IBM_adr := 0;
   tlk_adr := 7;
   red := space255;
   Gpsys(IBM_adr,successful);
   ...
   Gpred(tlk_adr,red,successful);  {receive red from
   ...                              tlk_adr when IBM
is
end.                                the controller}
```

(13) Gptlk procedure (Gpib talk) **Gpnsys**

Declaration Gptlk(wrt : text255; var successful : boolean)

Remarks — wrt is a text255-type expression (text255 type is a string type that not more than 255 characters), it contains the transmit data and successful returns true if statement calls successfully, otherwise return false.

Physical result — It does not send signals on GPIB bus lines, but it receives signals from the listener (or the controller) before transmission data until 5 seconds, then it failure.

Function — waits to be addressed to be a talker, and transmits data.

Communication — the talker (IBM PC) ---> the listener or the controller.

Timeout value — 5 seconds

Example

```
uses gpib;
var
   IBM_adr    : word;
   wrt        : text255;
   successful : boolean;
begin
   IBM_adr := 0;
   wrt := 'gpib test';
   Gpnsys(IBM_adr,successful);
   ...
   repeat
     ...
     Gptlk(red,successful);
     ...
   until keypressed or successful;
end.
```

(14) Gplsn procedure (Gpib listen) **Gpnsys**

Declaration: Gplsn(var red : text255; var successful : boolean)

Remarks: red is a text255-type parameter (text255 type is a string type that not more than 255 characters), it should be assigned arbitrary literal string to a maximum length user requires to receive (as : red = space255). After statement calls, user can type red, which contains the data received from the talker and successful returns true if statement calls successfully, otherwise return false.

Physical result: It does not send signals on GPIB bus lines, but it receives signals from the talker (or the controller) before receiving data until 5 seconds, then it fails.

Function: waits to be addressed to be a talker, and receives data into red.

Communication: the listener(IBM PC) <--- the talker or the controller.

Timeout value: 5 seconds

Example:

```
uses gpib;
var
   IBM_adr    : word;
   red        : text255;
   successful : boolean;
begin
   IBM_adr := 0;
   red := space255;
   Gpnsys(IBM_adr,successful);
   ...
   repeat
```

```
    ...
    Gplsn(red,successful);
    ...
  until keypressed or successful;
end.
```

(15) Gpspol procedure (Gpib serial poll) **Gpsys**

Declaration Gpspol(lsn : word; var rec : word; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30), rec contains valid status byte if statement call successful, and successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNL	SPE	listener addr.	talker addr.	receive status byte	SPD	UNT	UNL
Hex	$3F	$18	$20	$47	(ATN is low)	$19	$5F	$3F
Dec	63	24	32	71		25	95	63

{listener address = 0 and talker address = lsn = 7}

Function Executes serial poll for specified listener.

Communication the controller <--- the specified device.

Note see service request and polling section.

(16) Gppct procedure (Gpib pass control to) **Gpsys**

Declaration Gppct(lsn : word; var successful : boolean)

Remarks lsn is a listener address value (decimal number between 0 to 30) and successful returns true if statement calls successfully, otherwise return false.

Physical result send the following sequence commands.

Meaning	UNL	talker address	TCT
Hex	$3F	$47	$09
Dec	63	71	9

{talker address = lsn = 7}

Function Passes the right of control to the specified listener equipped with controller function.

Communication the controller ---> the specified device.

The following program is created to call the GPIB assembly language. This program is written in pascal language and made the unit for Turbo Pascal V.4.0-V.5.0 :

Program A-1 The unit of the GPIB controller card.

```
unit gpib;
interface
type
  text255 = string[255];
var
  space255 : text255;
```

```
  procedure gpsys(address:word;var state:boolean);
  procedure gpnsys(address:word;var state:boolean);
  procedure gpspol(lsn:word;var rec:word;var state:boolean);
  procedure gpwrt(lsn:word; text:text255;var state:boolean);
  procedure gpred(tlk:word;var text:text255;var state:boolean);
  procedure gplsn(var text:text255;var state:boolean);
  procedure gptlk(var text:text255;var state:boolean);
  procedure gpsclr(lsn:word;var state:boolean);
  procedure gpsloc(lsn:word;var state:boolean);
  procedure gptrg(lsn:word;var state:boolean);
  procedure gppct(lsn:word;var state:boolean);
  procedure gpifc(var state:boolean);
  procedure gpaclr(var state:boolean);
  procedure gpllo(var state:boolean);
  procedure gpren(var state:boolean);
  procedure gpaloc(var state:boolean);

implementation
type
  text3 = string[3];
const
  ofs_gpsys   : word = 67;
  ofs_gpnsys  : word = 10;
  ofs_gpifc   : word = 124;
  ofs_gpsclr  : word = 190;
  ofs_gpaclr  : word = 239;
  ofs_gpllo   : word = 305;
  ofs_gpren   : word = 371;
  ofs_gpsloc  : word = 437;
  ofs_gpaloc  : word = 486;
  ofs_gptrg   : word = 552;
  ofs_gpwrt   : word = 601;
  ofs_gpred   : word = 800;
  ofs_gptlk   : word = 686;
```

```
  ofs_gplsn    : word = 885;
  ofs_gpspol   : word = 995;
  ofs_gppct    : word = 1049;

var
  Adr          : word;
  Lsn          : word;
  sta          : word;
  Red          : text255;
  Tlk          : word;
  Wrt          : text255;
  Rec          : word;
  Segment      : word;
  NewDs        : word;
  dummy        : char;
  route_addr   : longint;
  i            : byte;
  successful   : boolean;

procedure set_route_addr;
begin
    Segment := 0;
    NewDs := MemW[Segment:1266];
      {result the same as Mem[Segment:1266]+(256*Mem[Segment:1267])}
    Segment := NewDs;
end;

procedure gpib1(NewDs,Offset:word;var sta:word);
  inline($55/                    {push bp}
         $8b/$ec/                {mov  bp,sp}
         $50/                    {push ax}
         $53/                    {push bx}
         $51/                    {push cx}
         $52/                    {push dx}
         $56/                    {push di}
```

```
        $57/                    {push si}
        $1e/                    {push ds}
        $8b/$76/$02/            {mov  si,[bp+2]}
        $56/                    {push si}
        $8e/$5e/$04/            {mov  ds,[bp+4]}
        $ff/$5e/$06/            {call far [bp+6]}
        $89/$76/$02/            {mov  [bp+2],si}
        $1f/                    {pop  ds}
        $5f/                    {pop  di}
        $5e/                    {pop  si}
        $5a/                    {pop  dx}
        $59/                    {pop  cx}
        $5b/                    {pop  bx}
        $58/                    {pop  ax}
        $5d);                   {pop  bp}

procedure gpib2A(NewDs,Offset,data_segment,ofs_adr:word;ofs_sta:word);
  inline($55/                   {push bp}
        $8b/$ec/                {mov  bp,sp}
        $50/                    {push ax}
        $53/                    {push bx}
        $51/                    {push cx}
        $52/                    {push dx}
        $56/                    {push di}
        $57/                    {push si}
        $1e/                    {push ds}
        $8b/$76/$04/            {mov  si,[bp+4]}
        $56/                    {push si}
        $8b/$76/$02/            {mov  si,[bp+2]}
        $56/                    {push si}
        $8e/$5e/$06/            {mov  ds,[bp+6]}
        $ff/$5e/$08/            {call far [bp+8]}
        $89/$76/$02/            {mov  [bp+2],si}
        $1f/                    {pop  ds}
```

```
        $5f/                    {pop  di}
        $5e/                    {pop  si}
        $5a/                    {pop  dx}
        $59/                    {pop  cx}
        $5b/                    {pop  bx}
        $58/                    {pop  ax}
        $5d);                   {pop  bp}

procedure gpib2B(NewDs,Offset,data_segment,ofs_adr_text,ofs_sta:word);
  inline($55/                   {push bp}
        $8b/$ec/                {mov  bp,sp}
        $50/                    {push ax}
        $53/                    {push bx}
        $51/                    {push cx}
        $52/                    {push dx}
        $56/                    {push di}
        $57/                    {push si}
        $1e/                    {push ds}
        $8b/$76/$04/            {mov  si,[bp+04]}
        $46/                    {inc  si}
        $56/                    {push si}
        $8b/$76/$02/            {mov  si,[bp+02]}
        $56/                    {push si}
        $8e/$5e/$06/            {mov  ds,[bp+06]}
        $ff/$5e/$08/            {call far [bp+08]}
        $89/$76/$02/            {mov  [bp+02],si}
        $1f/                    {pop  ds}
        $5f/                    {pop  di}
        $5e/                    {pop  si}
        $5a/                    {pop  dx}
        $59/                    {pop  cx}
        $5b/                    {pop  bx}
        $58/                    {pop  ax}
        $5d);                   {pop  bp}
```

```
procedure gpib3A(NewDs,Offset,data_segment,ofs_adr,ofs_adr_text,
        ofs_sta:word);
  inline($55/                    {push bp}
        $8b/$ec/                 {mov  bp,sp}
        $50/                     {push ax}
        $53/                     {push bx}
        $51/                     {push cx}
        $52/                     {push dx}
        $56/                     {push di}
        $57/                     {push si}
        $1e/                     {push ds}
        $8b/$76/$06/             {mov  si,[bp+06]}
        $56/                     {push si}
        $8b/$76/$04/             {mov  si,[bp+04]}
        $46/                     {inc  si}
        $56/                     {push si}
        $8b/$76/$02/             {mov  si,[bp+02]}
        $56/                     {push si}
        $8e/$5e/$08/             {mov  ds,[bp+08]}
        $ff/$5e/$0a/             {call far [bp+0a]}
        $89/$76/$02/             {mov  [bp+02],si}
        $1f/                     {pop  ds}
        $5f/                     {pop  di}
        $5e/                     {pop  si}
        $5a/                     {pop  dx}
        $59/                     {pop  cx}
        $5b/                     {pop  bx}
        $58/                     {pop  ax}
        $5d);                    {pop  bp}

procedure gpib3B(NewDs,Offset,data_segment,ofs_lsn,ofs_rec,
        ofs_sta:word);
  inline($55/                    {push bp}
        $8b/$ec/                 {mov  bp,sp}
```

```
        $50/                    {push ax}
        $53/                    {push bx}
        $51/                    {push cx}
        $52/                    {push dx}
        $56/                    {push di}
        $57/                    {push si}
        $1e/                    {push ds}
        $8b/$76/$06/            {mov  si,[bp+06]}
        $56/                    {push si}
        $8b/$76/$04/            {mov  si,[bp+04]}
        $56/                    {push si}
        $8b/$76/$02/            {mov  si,[bp+02]}
        $56/                    {push si}
        $8e/$5e/$08/            {mov  ds,[bp+08]}
        $ff/$5e/$0a/            {call far [bp+0a]}
        $89/$76/$02/            {mov  [bp+02],si}
        $1f/                    {pop  ds}
        $5f/                    {pop  di}
        $5e/                    {pop  si}
        $5a/                    {pop  dx}
        $59/                    {pop  cx}
        $5b/                    {pop  bx}
        $58/                    {pop  ax}
        $5d);                   {pop  bp}
procedure gpspol(lsn:word;var rec:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_lsn,ofs_sta,ofs_rec : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
```

```
    ofs_rec := (Seg(rec)*16+Ofs(rec)+1) - data_segment*16;
    gpib3B(segment,ofs_gpspol,data_segment,ofs_lsn,ofs_rec,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gpred(tlk:word;var text:text255;var state:boolean);
var sta:word;
    adr_text : text3;
    ofs_text : word;
    strlen   : byte absolute text;
    data_segment : word;
    ofs_tlk,ofs_sta,ofs_adr_text : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_tlk := (Seg(tlk)*16+Ofs(tlk)) - data_segment*16;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_text :=(Seg(text)*16+Ofs(text)+1) - data_segment*16;
    ofs_adr_text := (Seg(adr_text)*16+Ofs(adr_text)) - data_segment*16;
    adr_text := chr(strlen)+chr(lo(ofs_text))+chr(hi(ofs_text));
    gpib3A(segment,ofs_gpred,data_segment,ofs_tlk,ofs_adr_text,
          ofs_sta);
    strlen := Mem[data_segment:(ofs_text+1)];
    if sta = 1 then state := true else state := false;
end;

procedure gplsn(var text:text255;var state:boolean);
var sta:word;
    adr_text : text3;
    ofs_text : word;
    strlen   : byte absolute text;
    data_segment : word;
    ofs_sta,ofs_adr_text : word;
begin
```

```
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_text :=(Seg(text)*16+Ofs(text)+1) - data_segment*16;
    ofs_adr_text := (Seg(adr_text)*16+Ofs(adr_text)) - data_segment*16;
    adr_text := chr(strlen)+chr(lo(ofs_text))+chr(hi(ofs_text));
    gpib2B(segment,ofs_gplsn,data_segment,ofs_adr_text,ofs_sta);
    strlen := Mem[data_segment:(ofs_text+1)];
    if sta = 1 then state := true else state := false;
end;

procedure gptlk(var text:text255;var state:boolean);
var sta:word;
    adr_text : text3;
    ofs_text : word;
    strlen   : byte absolute text;
    data_segment : word;
    ofs_sta,ofs_adr_text : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_text :=(Seg(text)*16+Ofs(text)+1) - data_segment*16;
    ofs_adr_text := (Seg(adr_text)*16+Ofs(adr_text)) - data_segment*16;
    adr_text := chr(strlen)+chr(lo(ofs_text))+chr(hi(ofs_text));
    gpib2B(segment,ofs_gptlk,data_segment,ofs_adr_text,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gpnsys(address:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_address,ofs_sta : word;
begin
```

```
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_address := (Seg(address)*16+Ofs(address)) - data_segment*16;
    gpib2A(segment,ofs_gpnsys,data_segment,ofs_address,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gpsys(address:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_address,ofs_sta : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_address := (Seg(address)*16+Ofs(address)) - data_segment*16;
    gpib2A(segment,ofs_gpsys,data_segment,ofs_address,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gpsclr(lsn:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_lsn,ofs_sta : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    gpib2A(segment,ofs_gpsclr,data_segment,ofs_lsn,ofs_sta);
    if sta = 1 then state := true else state := false;
end;
```


```
procedure gpsloc(lsn:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_lsn,ofs_sta : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    gpib2A(segment,ofs_gpsloc,data_segment,ofs_lsn,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gptrg(lsn:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_lsn,ofs_sta : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    gpib2A(segment,ofs_gptrg,data_segment,ofs_lsn,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gppct(lsn:word;var state:boolean);
var sta:word;
    data_segment : word;
    ofs_lsn,ofs_sta : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
```

```
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    gpib2A(segment,ofs_gppct,data_segment,ofs_lsn,ofs_sta);
    if sta = 1 then state := true else state := false;
end;

procedure gpifc(var state:boolean);
var sta : word;
begin
   sta := 0;
   gpib1(segment,Ofs_gpifc,sta);
   if sta = 1 then state := true else state := false;
end;

procedure gpaclr(var state:boolean);
var sta : word;
begin
   sta := 0;
   gpib1(segment,Ofs_gpaclr,sta);
   if sta = 1 then state := true else state := false;
end;

procedure gpllo(var state:boolean);
var sta : word;
begin
   sta := 0;
   gpib1(segment,Ofs_gpllo,sta);
   if sta = 1 then state := true else state := false;
end;

procedure gpren(var state:boolean);
var sta : word;
begin
   sta := 0;
```

```
   gpib1(segment,Ofs_gpren,sta);
   if sta = 1 then state := true else state := false;
end;

procedure gpaloc(var state:boolean);
var sta : word;
begin
   sta := 0;
   gpib1(segment,Ofs_gpaloc,sta);
   if sta = 1 then state := true else state := false;
end;

procedure gpwrt(lsn:word; text:text255;var state:boolean);
var sta:word;
    adr_text : text3;
    ofs_text : word;
    strlen   : byte absolute text;
    data_segment : word;
    ofs_lsn,ofs_sta,ofs_adr_text : word;
begin
    sta := 0;
    data_segment := dseg;
    ofs_lsn := (Seg(lsn)*16+Ofs(lsn)) - data_segment*16;
    ofs_sta := (Seg(sta)*16+Ofs(sta)) - data_segment*16;
    ofs_text :=(Seg(text)*16+Ofs(text)+1) - data_segment*16;
    ofs_adr_text := (Seg(adr_text)*16+Ofs(adr_text)) - data_segment*16;
    adr_text := chr(strlen)+chr(lo(ofs_text))+chr(hi(ofs_text));
    gpib3A(segment,ofs_gpwrt,data_segment,ofs_lsn,ofs_adr_text,
          ofs_sta);
        if sta = 1 then state := true else state := false;
        {gpllo(successful);  This command is used to correct gpwrt
         command, that has bug.}
end;
```

```
begin
  set_route_addr;
  space255 :='';
  for i:=1 to 255 do space255 := space255 + ' ';
end.
```

Appendix B

The tools that used in building the programmable pulse generator for NMR imaging.

It is very difficult in building the programmable pulse generator, if it is without the instruments that develop the system programs and the instruments that are used to study, or to check the GPIB system. These instruments can be provided into two sets. The first is the software development set, consists of the 2-way memory and the memory interface card using to interface with the AppleII computer. Since the system software is written by the 8048 assembly language and the AppleII computer has this 8048 assembler, so the software development is based on the AppleII computer. The second is the GPIB investigation set, consists of the GPIB display and 6522 interface card using to interface with the AppleII computer as the same as the memory interface card. This set uses the AppleII computer to investigate the GPIB card of the IBM PC computer.

The software development instruments

This section consists of the 2-way memory and the memory interface card, both instruments are used together. This section will tell how they are built and how user uses them.

1) The 2-way memory

This instrument is built for software development. In this thesis, it can keep the assembled program writing with 8048 assembly language on the AppleII computer. This instrument has 2 bus. The first bus is connected with the Apple II computer by passing the memory interface card. The second is connected with the 8035 microcomputer in MCS-48 series (as the same as 8048 microcomputer). The user can control the direction of the data transfer and control the write enable for writing

process. Typically, this instrument is connected with the computer - for example, single board or ET board (Somyot Lohavittayavigran and Junrachai Rungvichaviwatt 1986: 122-128). The circuit of this instrument uses 6116 RAM IC to keep the program and has the battery backup to prevent the program lose, when user turns it off.

The working description

This instrument uses the 2-way switch to select the 6116 RAM IC connect with the first bus (the Apple II computer) or the second bus (MCS-48). Since each bus consists of 8-bit data bus lines, 11-bit address bus lines and 3-bit control lines - such as $\overline{CE}$, $\overline{OE}$ and $R/\overline{W}$ (see Fig.B-1).

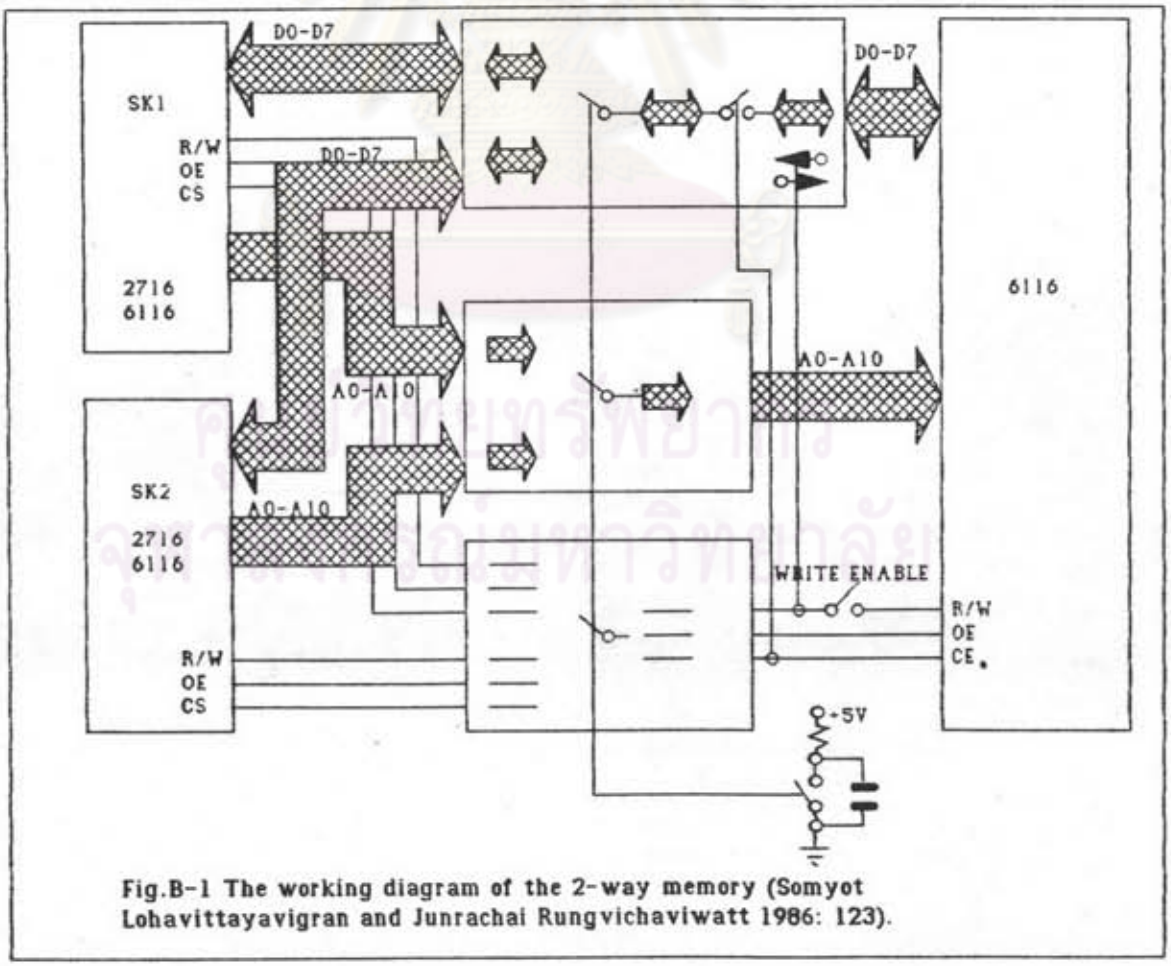


Fig.B-1 The working diagram of the 2-way memory (Somyot Lohavittayavigran and Junrachai Rungvichaviwatt 1986: 123).

Since the address bus lines and control lines are one-directional, so they are designed by using the 74LS257, 2-line-to-1 data selector/multiplexer with 3-state output. The data bus lines are bidirectional and tri-state logic. They are designed together with the $R/\overline{W}$ lines for control direction and the $\overline{CE}$ lines for control 3-state output whether the 6116 RAM is or not connected.

The real circuit is shown in Fig.B-2. The address bus selector part of the circuit is built by IC_3 - IC_6 (74LS257). The data bus selector part of the circuit is built by IC_1 and IC_2 (74LS245 - octal bus transceiver with 3-state outputs). The lines for control IC_1 and IC_2 are the $\overline{CE}$ line and SEL line (select the first bus or the second bus). They are used to control the direction of the data bus lines by the $R/\overline{W}$ line. The truth table of IC_1 and IC_2 is shown in table B-1.

Table B-1 The truth table of IC_1 and IC_2 (Somyot Lohavittayavigran and Junrachai Rungvichaviwatt 1986: 126)

Input			Output				
CS	SEL	R/W	EN_1	EN_2	DIR_1	DIR_2	Note
1	x	x	1	1	x	x	not connect with the 6116 RAM.
0	0	0	0	1	1	x	The first instrument can write data.
0	0	1	0	1	0	x	The first instrument can read data.
0	1	0	1	0	x	1	The second instrument can write data.
0	1	1	1	0	x	0	The second instrument can read data.

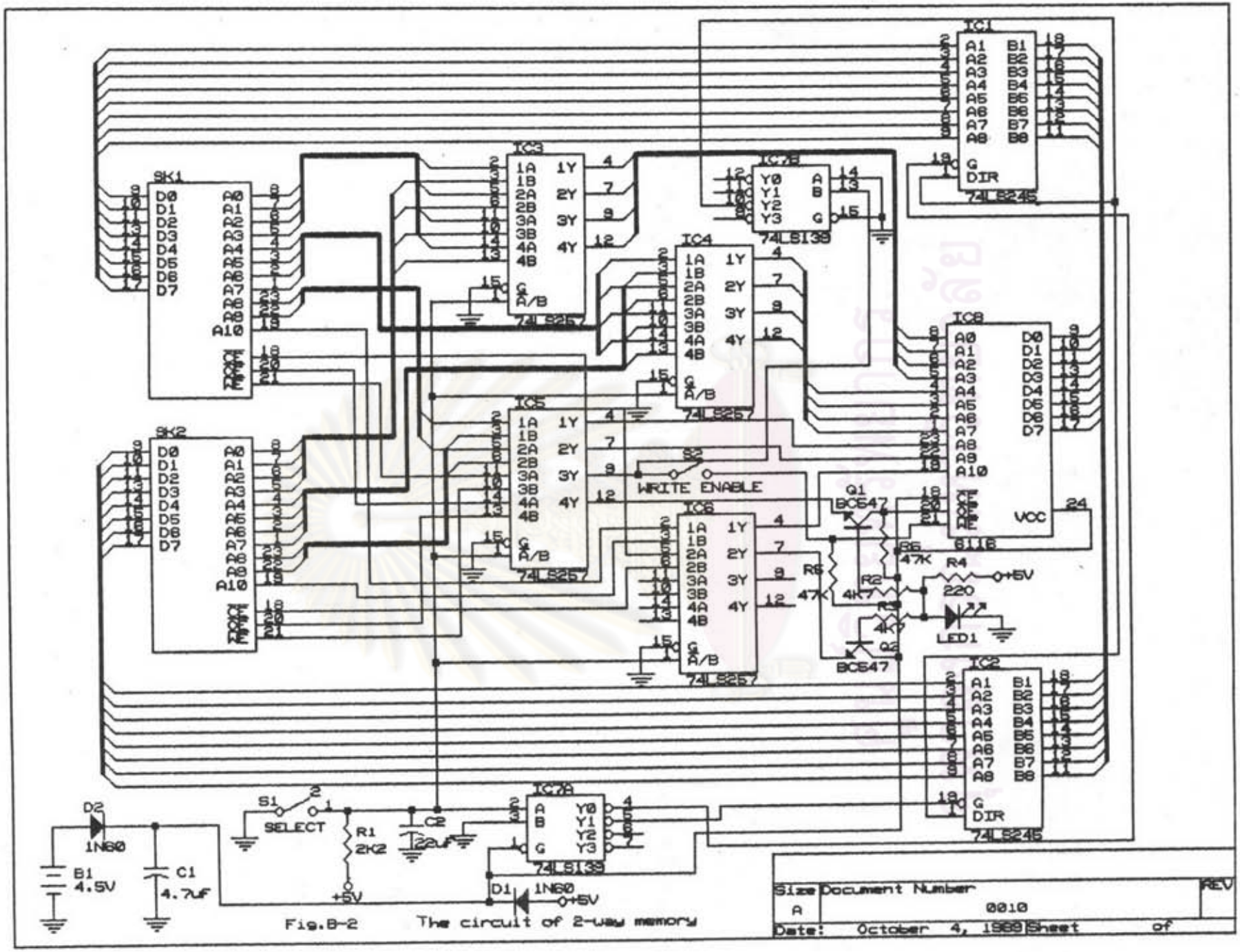


Fig.B-2 The circuit of 2-way memory

From table B-2, the appropriate IC that decodes the signal for this table is the 74LS139 (IC7), dual 2-line-to-4-line decoder/demultiplexer. It connects to some lines as in Fig.B-2. The printed circuit board of this circuit is shown in Fig.B-3 and Fig.B-4 illustrates component layout of this PCB.

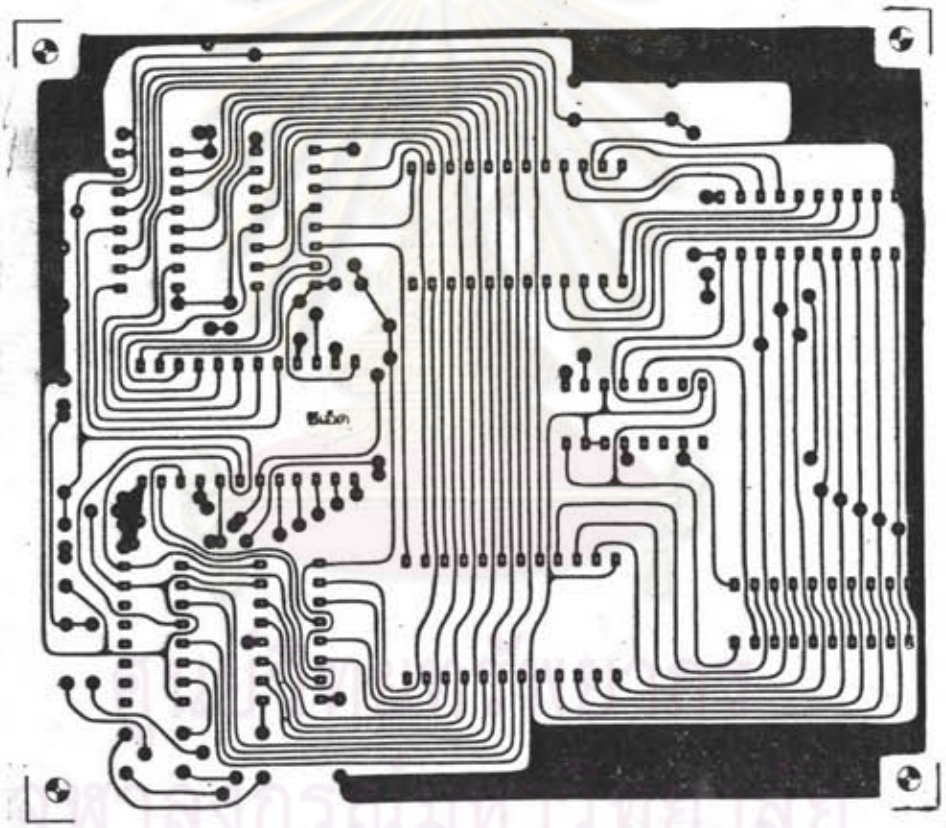


Fig.B-3 The printed circuit board of the 2-way memory, not include power supply (Somyot Lohavittayavigran and Junrachai Rungvichaviwatt 1986: 127).

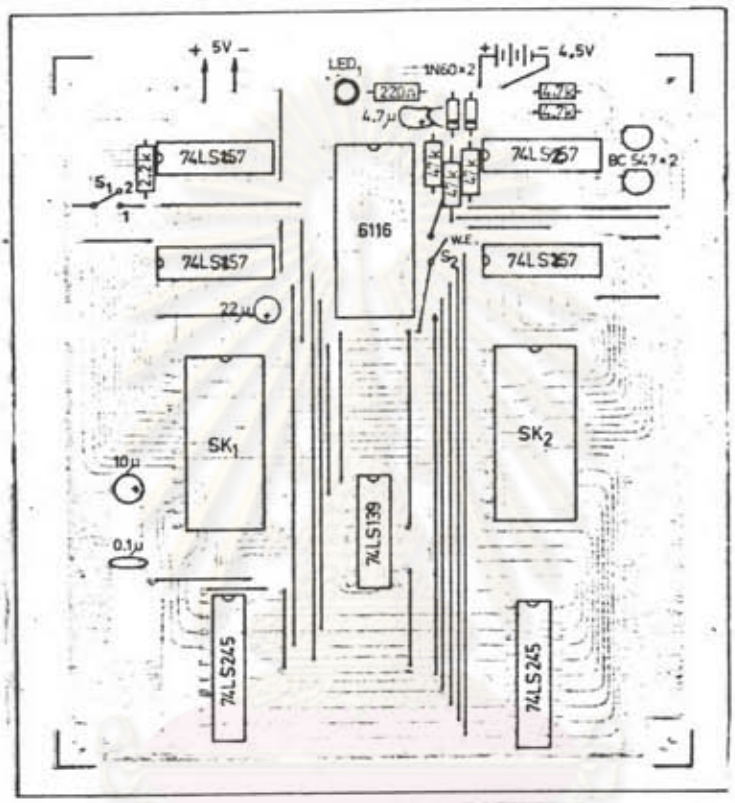


Fig.B-4 The component layout of the PCB (Somyot Lohavittayavigran and Junrachai Rungvichaviwatt 1986: 127).

The operation of this 2-way memory

After building this instrument, the front panel key operation is shown in Fig.B-5.

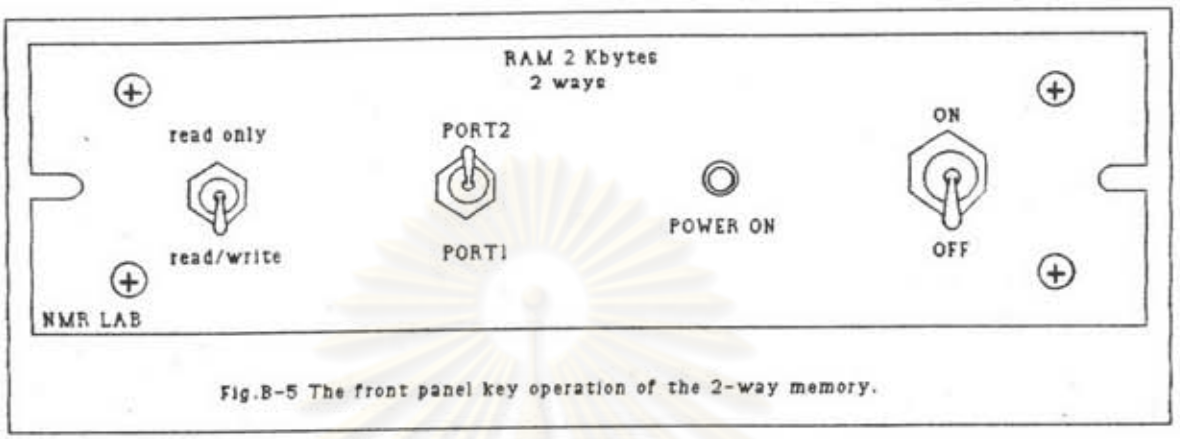


Fig.B-5 The front panel key operation of the 2-way memory.

In this thesis, the port2 of this instrument connects to the memory interface card, which will be described in the next section. The port1 connects to the 8035 microcomputer single board which controls the GPIB system of the programmable pulse generator (the pulse programmer and the pulse shaping unit). To insert the memory interface card in a slot of the AppleII computer, then transfer the 8048 assembled program to the 2-way memory by using the memory interface card. When it is set the front panel key operation as in Fig.B-5, but power is turned on. This program will keep in the 6116 RAM. Then user turns the left switch to "read only" and the next switch to "port1" - the 8035 microcomputer single board can only read this program on the 6116 RAM. The user should turn this single board off before the middle switch of the 2-way memory is turned to "port1". If this program is not correct then the single board will work mistakenly. So user can improve this program on the AppleII computer until program has no error. This is the software development process.

2) The memory interface card

This instrument is built to connect the 2-way memory with the AppleII computer for software development. Its circuit is shown in Fig.B-6 (Boonleart Elamtudsana 1984: 192).

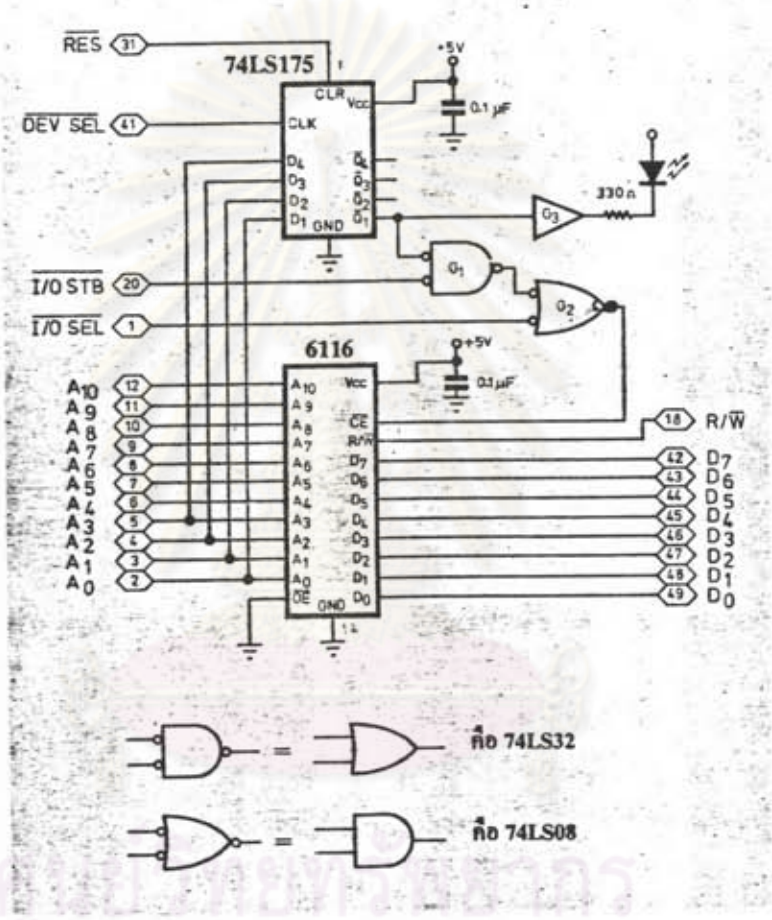


Fig.B-6 The circuit of the memory interface (Boonleart Eiamtudsana 1984: 192).

Since the AppleII computer provides the 2 Kbytes memory from address $C800 to address $CFFF for all slots (8 slots). As user calls this interval address, the I/O STROBE line of all slots will be active. Yet the AppleII computer provides the 256 address memory from address $C000 to address $C0FF by the first half is used for soft switch and the other half is provided to 16 addresses for each slots. As user calls these 16

addresses of one particular slot, the $\overline{\text{DEVICE SELECT}}$ line of that slots will be active. From providing memory of the AppleII computer, the memory interface card is designed as shown in Fig.B-6. The printed circuit board is shown in Fig.B-7 and the component layout is shown in Fig.B-8.

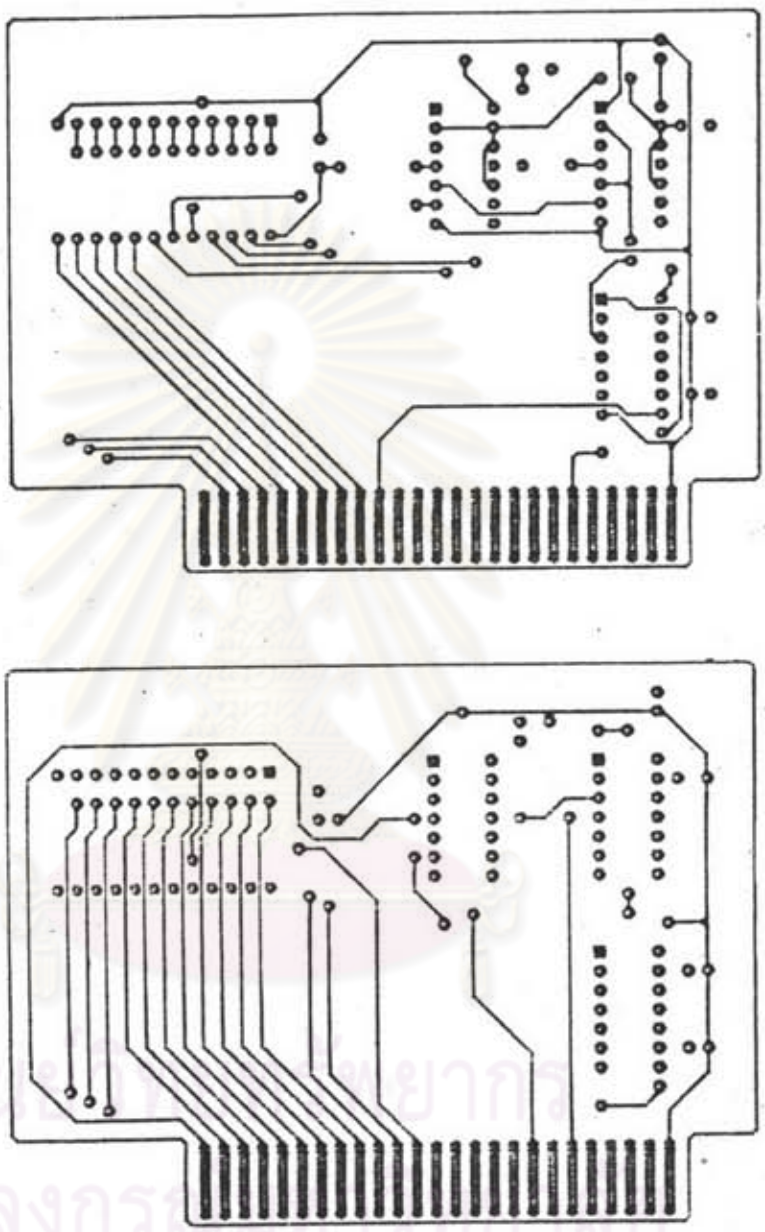


Fig.B-7 The printed circuit board of the memory interface card. (a) The mirror of solder side. (b) The component side.

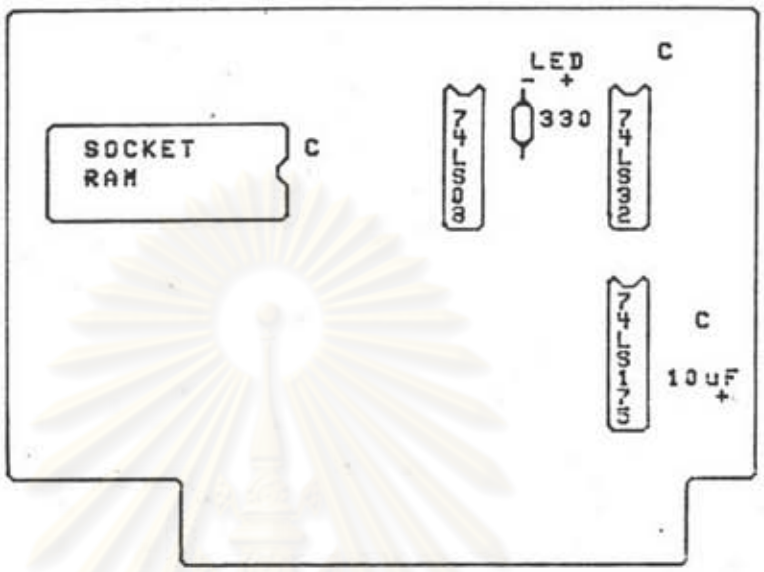


Fig.B-8 The component layout of the memory interface card.

The operation of this memory interface card

To connect this memory interface with the 2-way memory by instead of the 6116 RAM on this memory interface, then type the following command:

]CALL-151 ; go to monitor ROM. (type MNTR for 8048 Macro Assembler).

*C0C1 ; enable the memory interface card, the LED on card will be turn on (when the memory interface card must insert in slot #4).

*C800<0800.0FFFM ; move the data from address $0800-$0FFF to address $C800-$CFFF.

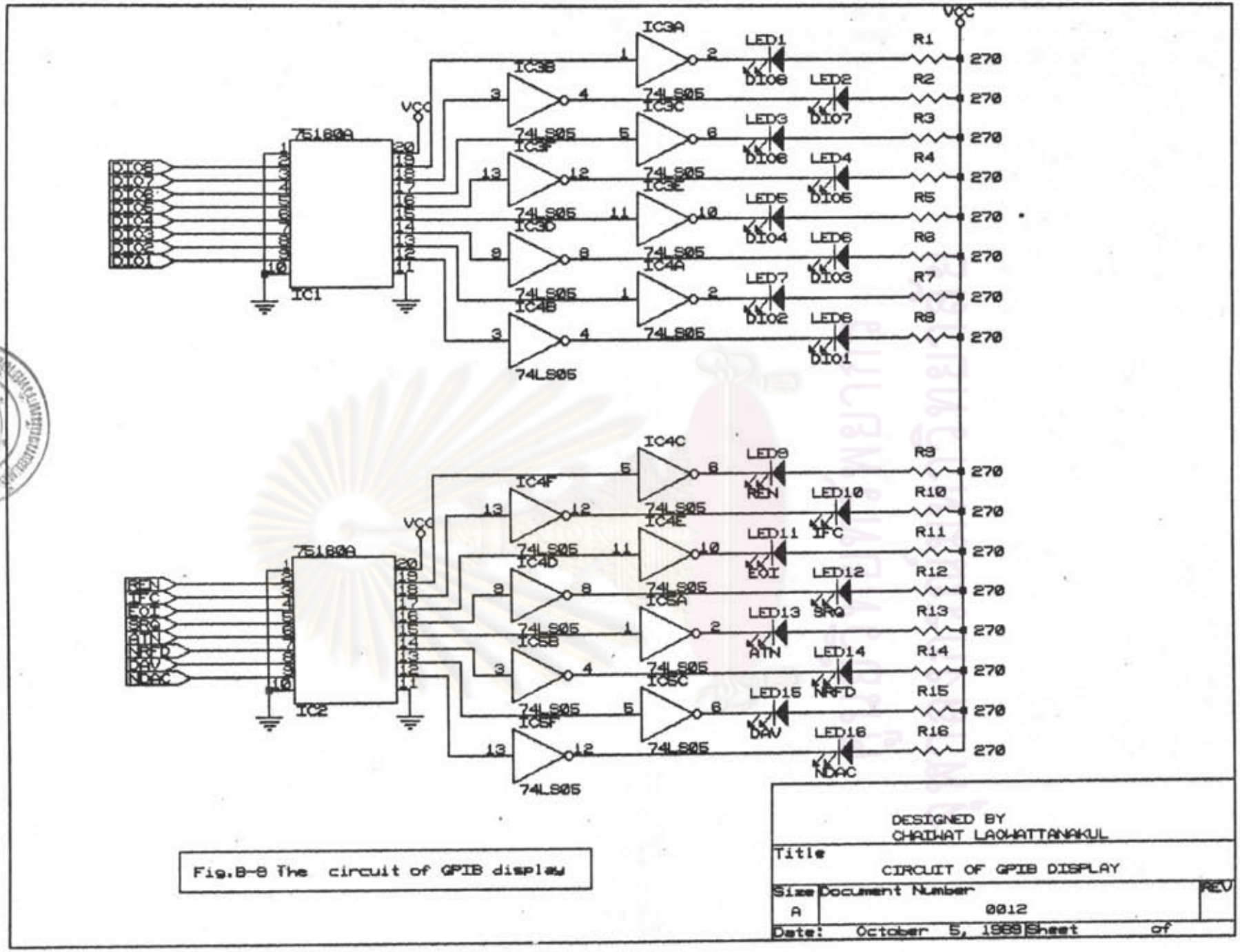


Fig.8-8 The circuit of GPIB display

DESIGNED BY
CHAIWAT LAOWATTANAKUL

Title
CIRCUIT OF GPIB DISPLAY

Size	Document Number	REV
A	0012	

Date: October 5, 1989 Sheet of

*C800<0800.0FFFV ; verify the data between address $0800-$0FFF with address $C800-$CFFF.

*C0C0 ; disable the memory interface card, the LED on card will be turn off.

*CTRL-C ; go back to BASIC (or to 8048 Macro Assembler).

In this thesis, transferred program is written with 8048 Macro Assembler and is assembled to the starting address $0800 by command "TA 0800H" at the top of program (see program 7.1 or program 7.2, for example).

The GPIB Investigation Instruments.

This section has two instruments, such as the GPIB display and the 6522 interface card, both instruments are used together. This section will tell how they can be built and how user can be used them.

1) The GPIB display

This instrument is built for investigation the GPIB system. It is used to display states of 16-bit GPIB bus lines and to connect between the 6522 interface card and the GPIB bus lines. Its circuit is shown in Fig.B-9. The LED of this circuit will light when the signal on this line is high. Similarly, it will not light when the signal is low. This circuit uses the 75160A ICs (driver buffer). It connectes to the 74LS05 ICs (inverter with open collector output).

2) The 6522 interface card

This instrument is the two 8-bit bidirectional input/output ports with or without 4 handshake lines. It has many applications to use it. In this thesis it is used as the GPIB card for the AppleII computer, but it is not perfect because of no longer time and difficulty to write the

control program. This circuit, the printed circuit board and the component layout of this instrument are shown in Fig.B-10, Fig.B-11 and Fig.B-12, respectively.

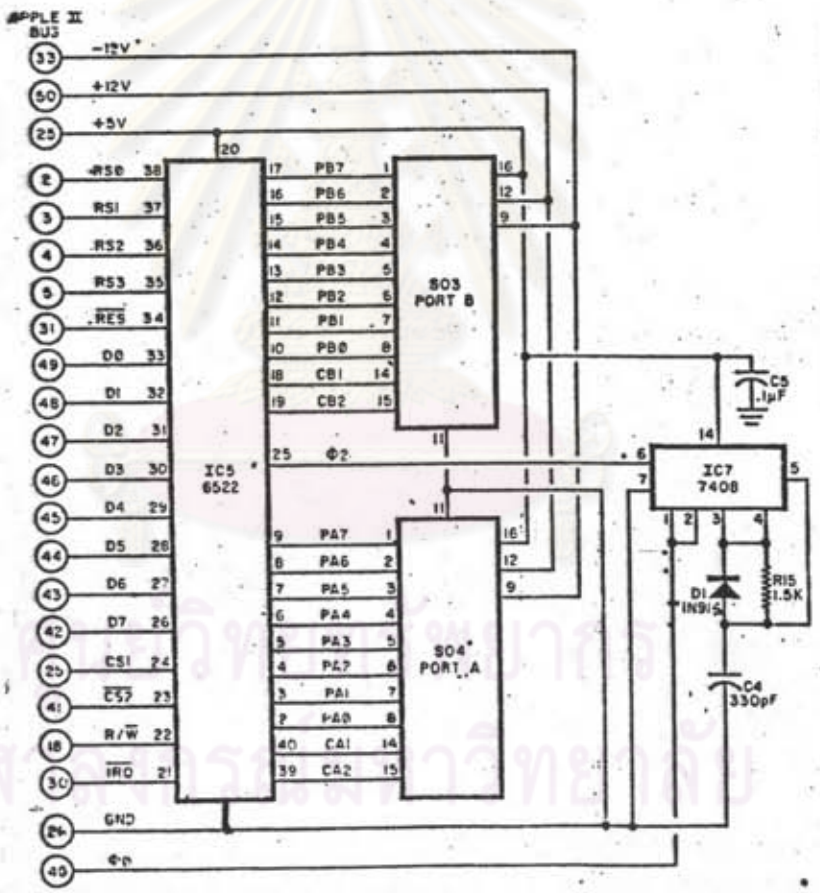


Fig.B-10 The circuit of the 6522 interface card.

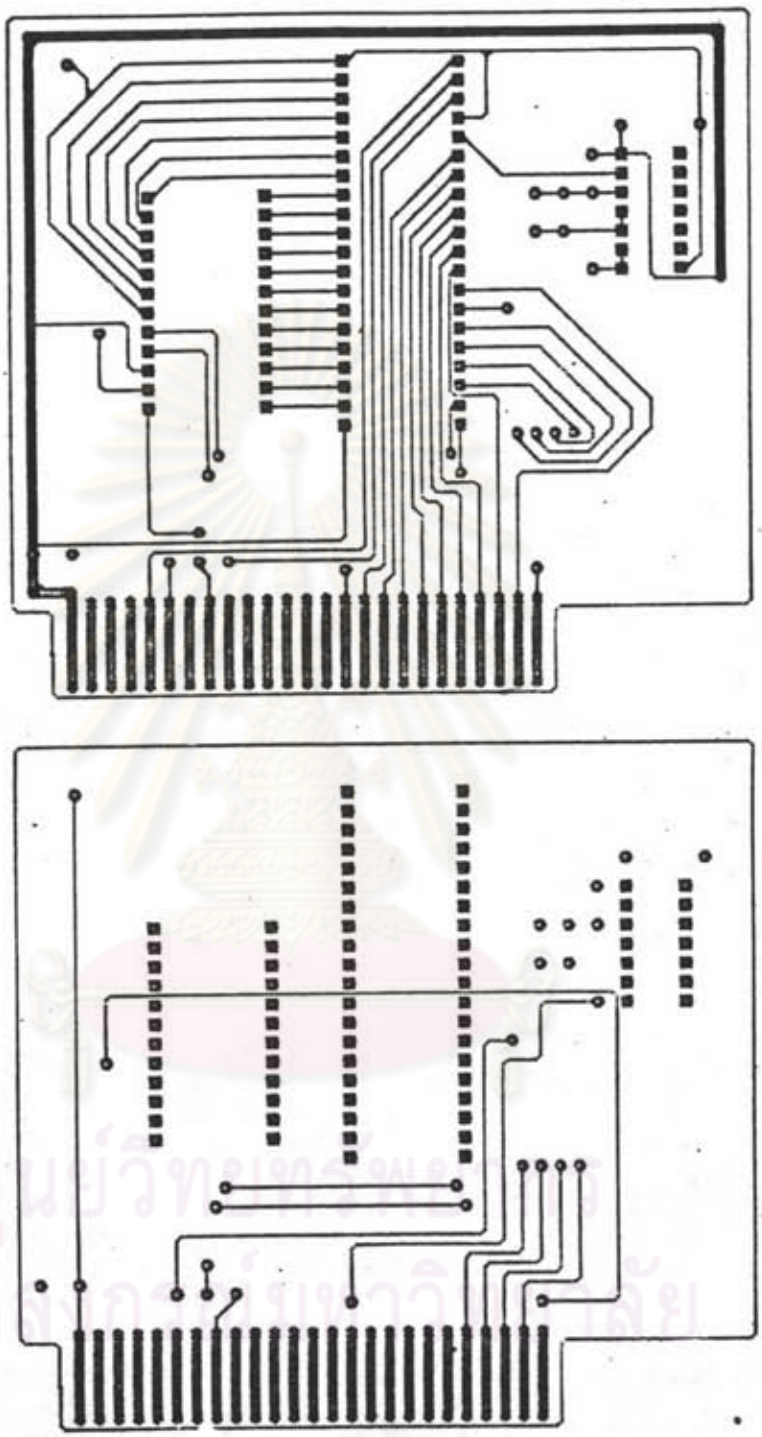


Fig.B-11 The printed circuit board of the 6522 interface card. (a) The solder side.
(b) The mirror of the component side.

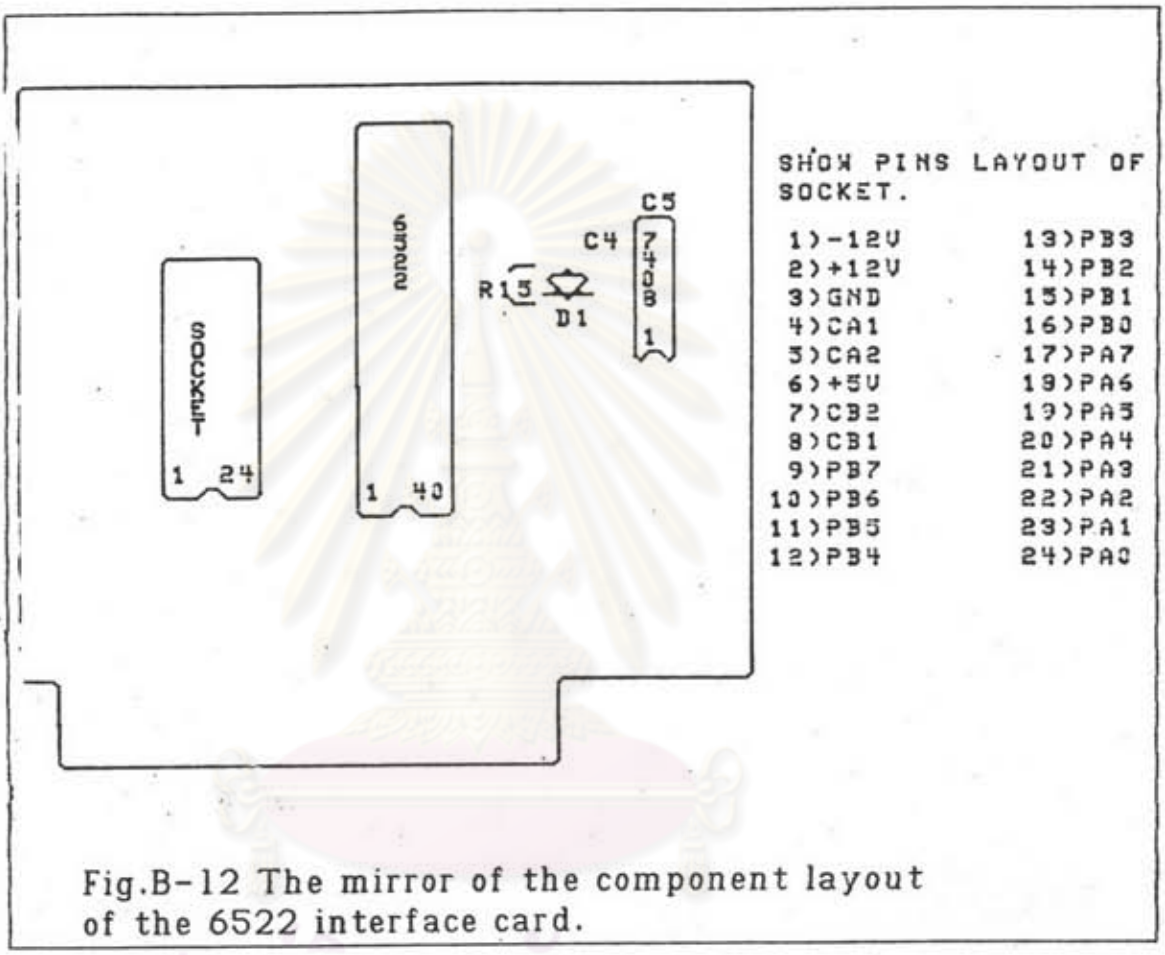


Fig.B-12 The mirror of the component layout of the 6522 interface card.

This instrument is used to together with Turbo Pascal V.2.0 language program on the AppleII computer.

Appendix C

Frequency divider

Since the pulse programmer and the pulse shaping unit must use the time bases for synchronous with together. So the frequency divider is used for this system. This frequency divider consists of 4 channels. Each channel has a frequency input and a frequency output that frequency is divided by 2N - where N is the arbitrary number between 1 and 9999. This frequency divider gets the clock from the external clock source - it is not necessary that is square wave or oscillate between 0V and 5V, but it may be the sine wave or oscillate between -10V and 10V. The frequency that is divided is oscillating between 0V and 5V for the TTL system.

The working process of the frequency divider

From Fig.C-1, it illustrates the working diagram of the frequency divider for one channel.

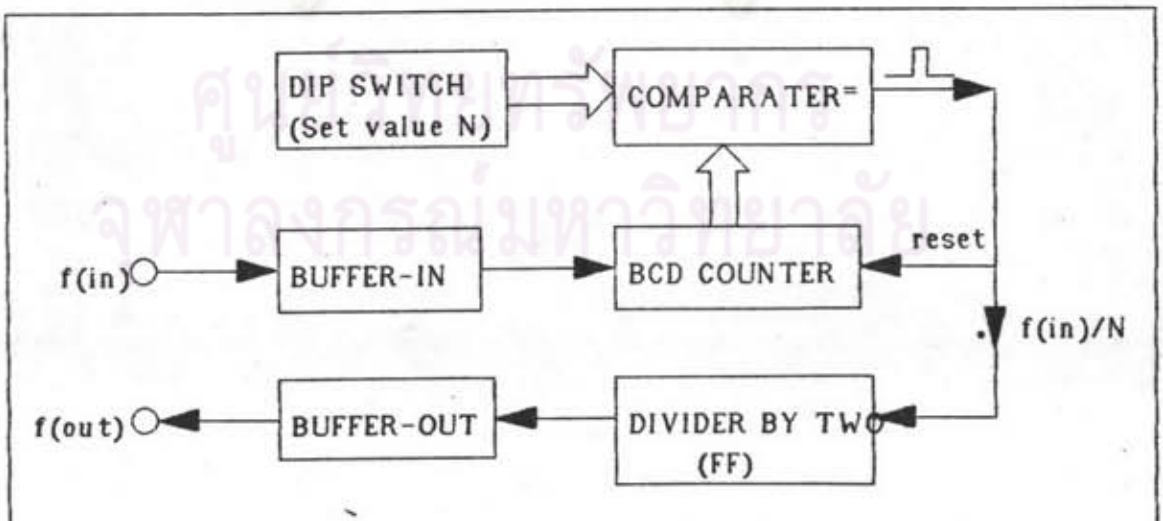


Fig.C-1 The working diagram of frequency divider for one channel.

Buffer-in will be clipped on the input frequency to be not over TTL level - 0V and 5V, and adjust waveform to square wave. this square wave is used for clock of BCD counter. The dip switch on the front panel key is used to set value N. If the output of BCD counter is equal to value N then the output of comparator (normally high) will reset the BCD counter to zero and will send to flipflop. This flipflop will divides the signal by two for duty cycle 50%. In the other word, the BCD counter will restart every value N and the frequency on buffer-out is $f_{in}/2N$.

From Fig.C-2 IC11, D1-D2 and R1 are used for the buffer-in. IC1-IC4 are used for the comparator. SW1-SW2 and R2-R18 are used for setting value N. IC5-IC9 are used for BCD counter. IC10, IC12 are used for divider by two and the buffer-out, respectively. The supply for this circuit is the same as PPM and PSP. The printed circuit board and the component layout of this circuit are illustrated in Fig.C-3 and Fig.C-4, respectively.

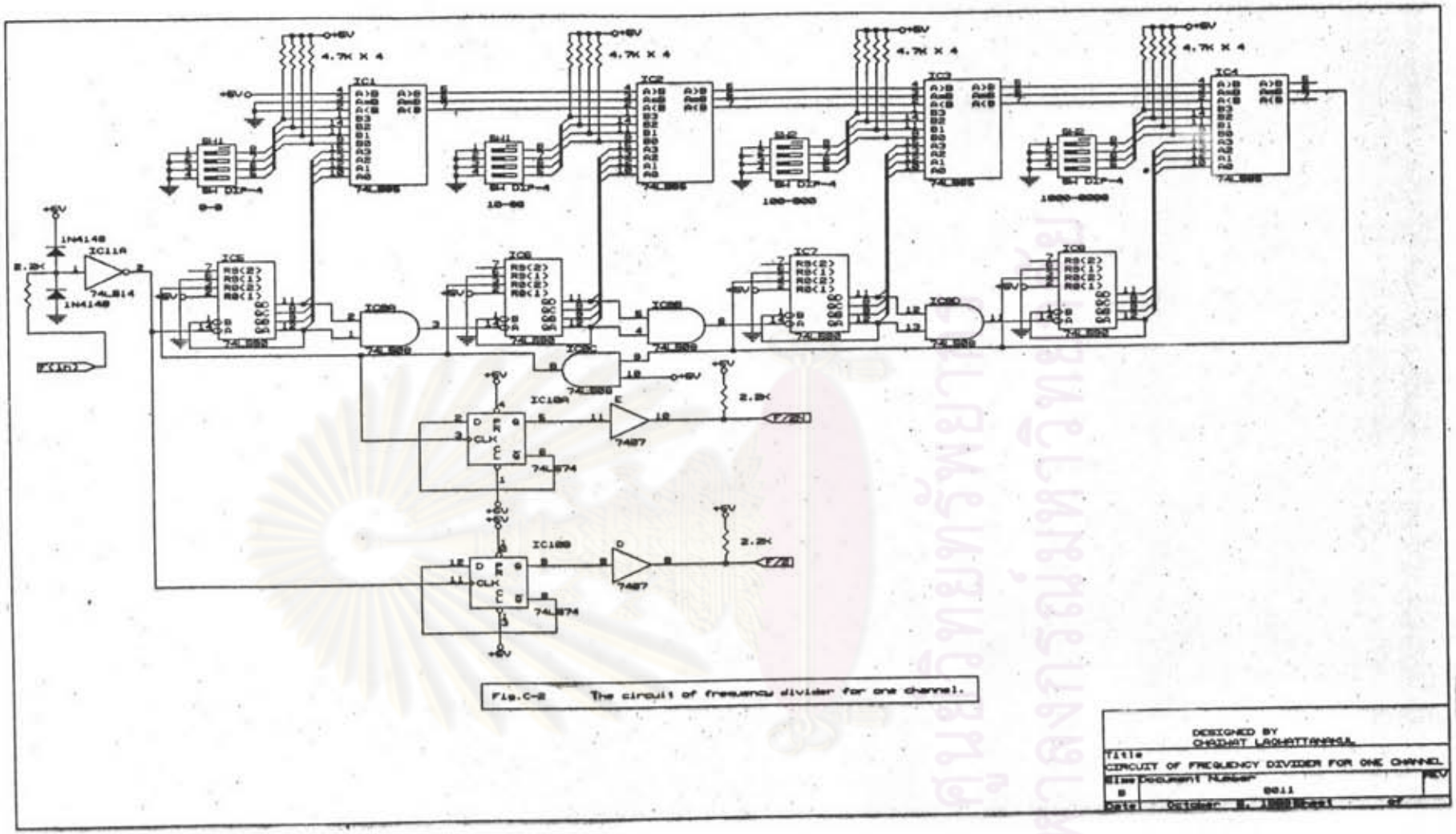


Fig.C-2 The circuit of frequency divider for one channel.

DESIGNED BY CHAIWAT LAOHATTANANSA.		
Title: CIRCUIT OF FREQUENCY DIVIDER FOR ONE CHANNEL		
Size: B	Document Number: 0011	REV
Date: October 8, 1988	Sheet	of

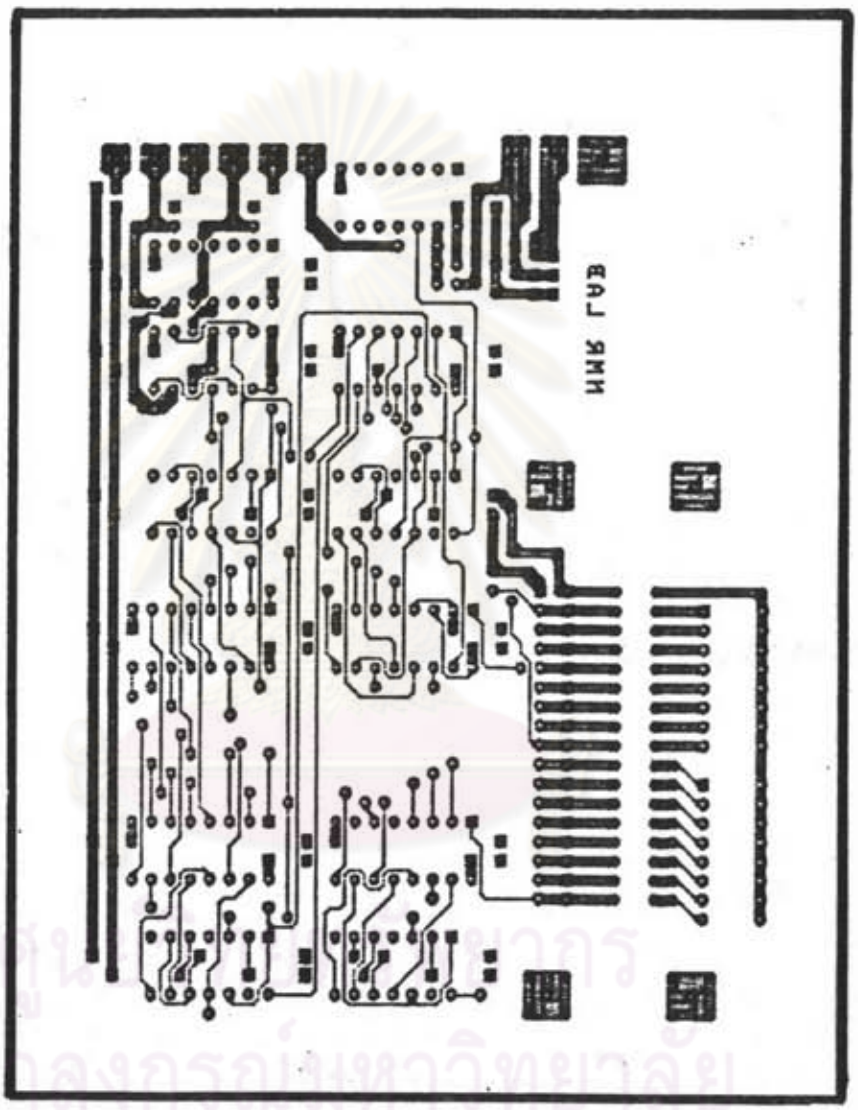


Fig.C-3(a) The mirror of the solder side of the printed circuit board for the frequency divider.

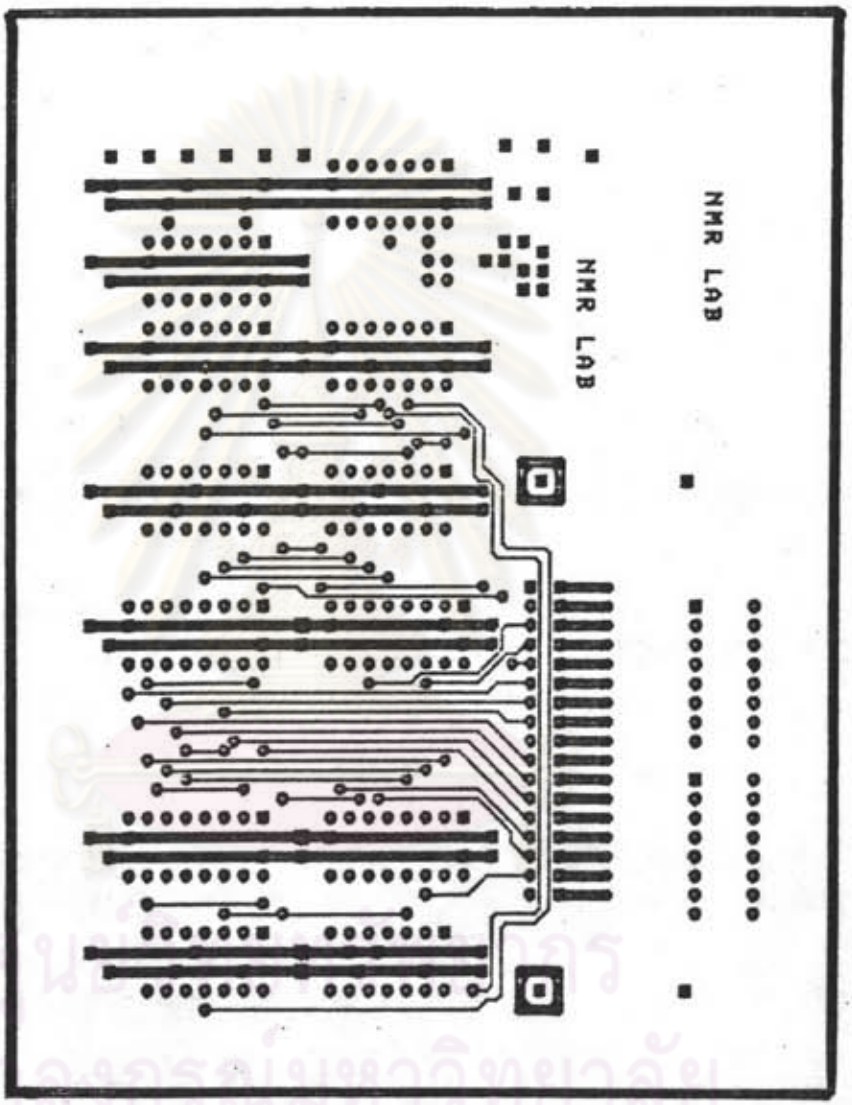


Fig.C-3(b) The component side of the printed circuit board for the frequency divider.

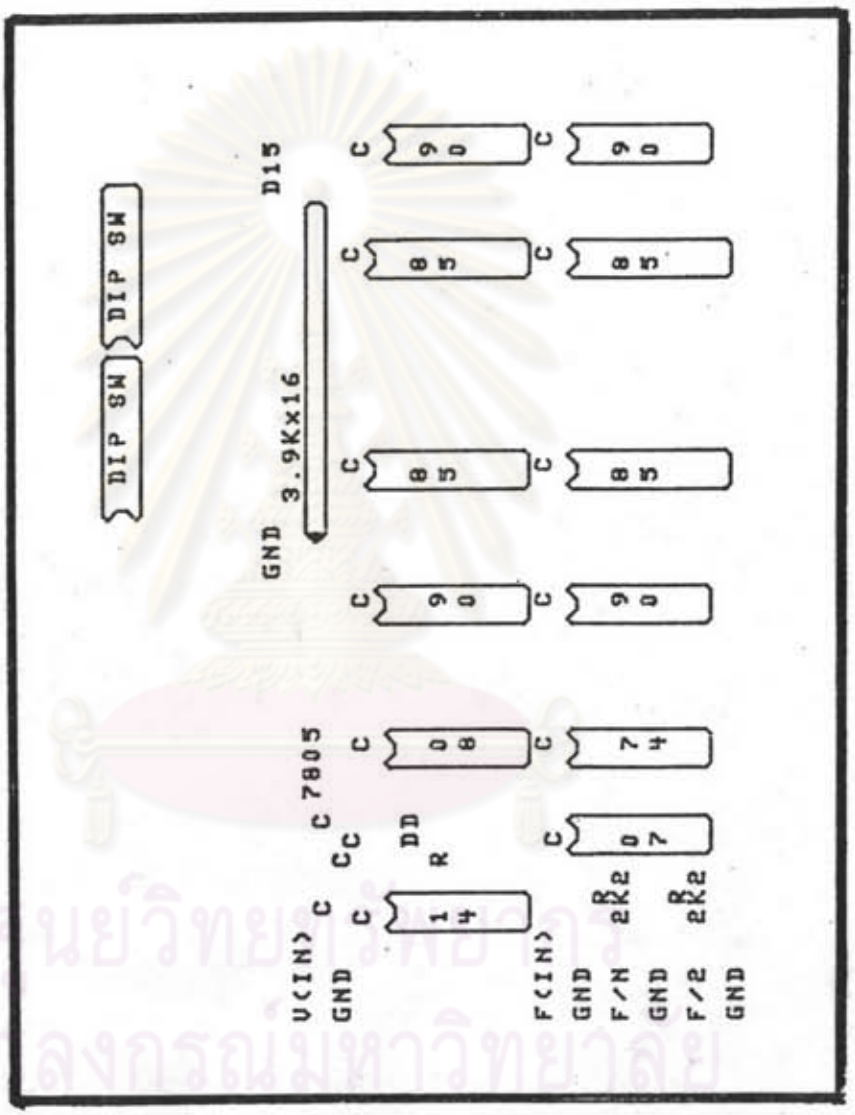


Fig.C-4 The component layout of the frequnecy divider.

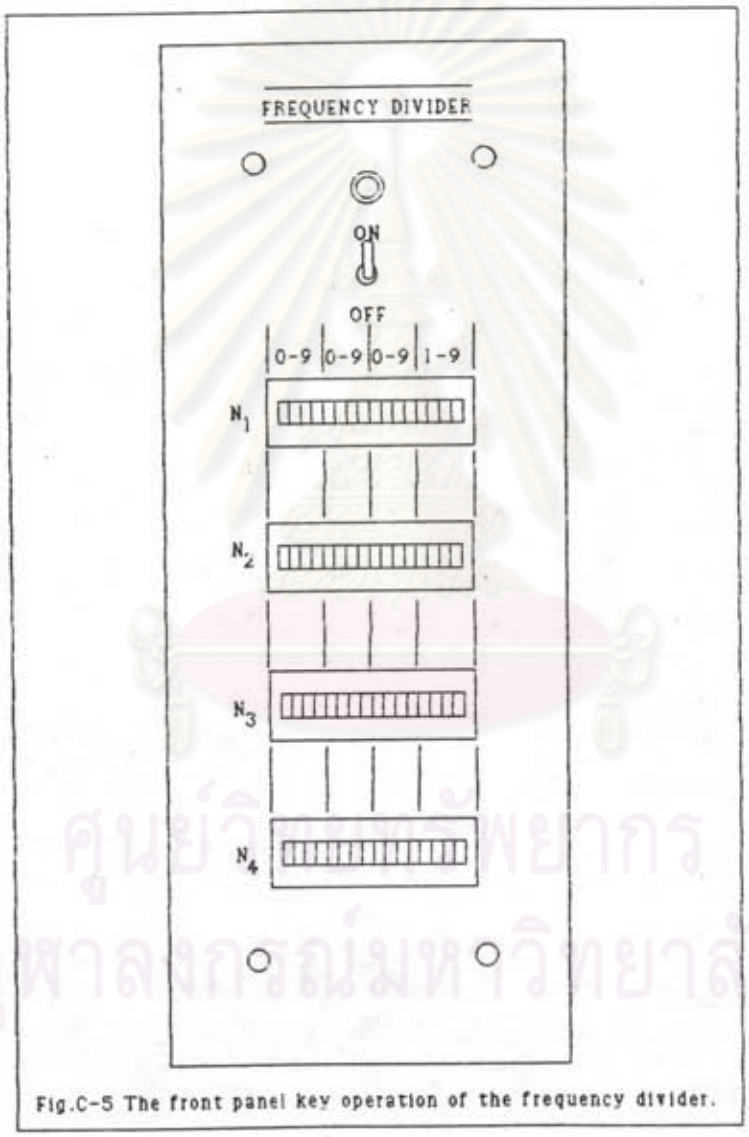


Fig.C-5 The front panel key operation of the frequency divider.

VITA

My name is Chaiwat Laowattanakul. I was born in Bangkok on February 24, 1964. I graduted on 1986 from Department of Physics, Chulalongkorn University. Before graduating, I builded a robot arm that was Bachelor's project.