Table of Contents

Article 1 Principle of Interface

With the rapid development of computer technology, peripheral equipment and interface technology has also developed very quickly. The ISA interface of the so-called computer interface technology PC XT/AT, have not been manufactured since 2000. Computer motherboards with ISA interface are also no longer seen on the market, while PCI have already became a main interface for internal transmission of computer. Although computer host machine still have new types, the slow speed interface and AGP graphic interface, its main structure is unable to break away from PCI interface. In the future, speedier interface technologies will be developed based on this interface technology.

The so-called PCI interface has existed in computer system for more than ten years, however, people are still mostly familiar with the interface technology of ISA, which have been put into use for over 20 years and are appreciably different from PCI interface in bandwidth, frequency, related circuit design and research. This article will describe current computer structure and the principle of ISA interface, including a brief introduction of PC XT/AT and ISA, related I/O address configuration, interruption methods, introduction of ISA interface-related chips and their principles of operating. However, for in-depth discussion of the ISA interface, please refer to books associated with designs and applications of PC XT/AT and ISA interface first, and then proceed to discuss the PCI interface.

The chips mentioned in this book are all embodied in the chip specifications folder in the disk attached to this book, which can be used by the reader for references. This book mainly focuses on principles and practices of PCI interface, describes the basic principle of ISA interface in brief only for the reader to know the evolution of computer interface, thus to understand the related principle of PCI interface. Meanwhile, for practice and experiments of related ISA, please refer to interface books published in the market.

Chapter 1 Brief Introduction of Computer and ISA Interface

The development of ISA interface begins with PC XT/AT, so far, it has gone through decades of development and evolution. In the past, interface textbook practices were based on this. However, in recent years, due to the rapid development of semi-conductor and computer technology, the IAS interface is no longer the mainstream, and is gradually replaced by a more speedy computer interface. In spite of this, interface-related basic knowledge is still limited to ISA interface, far lagging behind current interface technology. This chapter describes the basic principle of the ISA interface, which can provide a basis and comparison for PCI interface and move ahead to build the conception of interface.

1.1 PC XT/AT brief introduction

The personal computer or the so-called PC, which is the standard for personal computer established by IBM in the 1980s, belongs to an open system framework; and based on this standard, many manufacturers produced compatible personal computers, which greatly increased its popularity. Basic PC system uses INTEL X86 series CPU as operating core, initially it should work with X87 series floating-point unit to enhance its operating ability; after 80486 series, floating-point units are already included in the CPU, making its unnecessary to improve operating ability via selection. Several 8 bit extension slots are provided on PC XT motherboard for the use of related extension cards; however, in addition to several 8 bits extension slots on PC AT motherboard, another segment of the 8 bits extension slot is added, increasing the data bus to the standard of 16 bits. The above-mentioned extension slots are commonly known as ISA bus, after which, more compatible 32 bits buses have emerged, and finally, the PCI bus has became the mainstream.

Figure 1-1-1 shows the basic PC system block diagram, in which internal exchange power supply provides electricity demands for the motherboard, on which are placed the microprocessor (CPU), memory (RAM/ROM), keyboard and speaker interface, counter, interruption controller and dynamic access memory (DMA) controller, via the extension slots, a few more interface cards, functions such display and serial communication can be added.

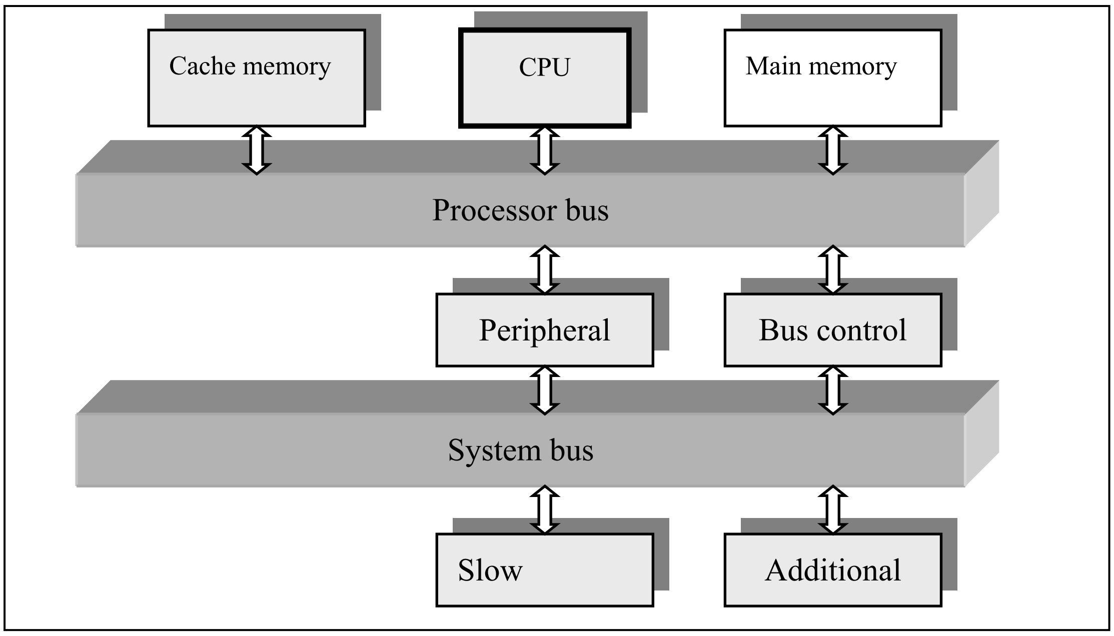


Figure 1-1-1 Basic PC system block diagram

1.2 PC XT/AT I/O address and memory configuration

PC XT/AT data input and output are achieved by defining output and input ports. The address scope of CPU is 00000H~0FFFFH, between which there are 65536 output/input ports available for use. However, PC XT/AT system only uses 1024 ports between 0000H~03FFH. Moreover, these 1024 ports are divided into two parts, with A9/SA9 address line as the dividing line: when A9/SA9 is low, 512 ports between address 0000H~01FFH can be used, these ports are provided for the system motherboard to use; and when A9/SA9 is high, 512 ports between address 0200H~03FFH can be used, these ports are provided for related interface cards to use. Table 1-3-1 shows the I/O address configuration of PC XT/AT.

The maximum memory that can be addressed for PC XT is 1MB, while the maximum memory that can be addressed for PC AT is 16 MB, there is little difference between the two parts: 00000H~FFFFFH, the part that exceeds 1 MB is extended memory, which is the basis of today’s computer memory. Table 1-2-2 shows the basic memory configuration diagram.

Table 1-2-1 I/O address configuration

PC XT	
Address used (hexadecimal)	Description
0000H~000FH	8237A DMA
0020H~0021H	8259A interrupt controller
0040H~0043H	8253 timer
0060H~0063H	8255A peripheral interface controller
0080H~0083H	DMA page controller
00A0H~00BFH	NMI mask bit
00C0H~01FFH	Reserved
0200H	Reserved
0201H	Pc game control interface card
0202H~0277H	Reserved
0278H~027FH	The second print port interface card
0280H~02F7H	Reserved
02F8H~02FFH	The second serial port interface card
0300H~031FH	Reserved
0320H~033FH	PC XT hard drive
0338H~0377H	Reserved
0378H~037FH	Print port interface card
0380H~03AFH	Reserved
03B0H~03BFH	Single color and print port interface card
03C0H~03CFH	Reserved
03D0H~03DFH	Color graphics interface card
03E0H~03EFH	Reserved
03F0H~03F7H	5 1/4 inch drive interface card
03F8H~03FFH	Serial port interface card

PC AT	
Address used (hexadecimal)	Description
0000H~001FH	DMA controller 1 (8237A-5)
0020H~003FH	Interrupt controller 1 (8259A,main)
0040H~005FH	Timer 8254
0060H~006FH	8042 keyboard controller
0070H~007FH	Real-time clock and NMI mask register
0080H~009FH	DMA page register (74LS612)
00A0H~00BFH	Interrupt controller 2 (8259A)
00C0H~00DFH	DMA controller 2 (8237A-5)
00F0H	Clear mathematics auxiliary processor
00F1H	Reset mathematics auxiliary processor
00F8H~00FFH	Mathematics auxiliary processor
01F0H~01F8H	Hard drive control card
0200H~0207H	Computer game control game
0278H~027FH	Parallel printer control card2
02F8H~02FFH	Serial transmission control card 2
0300H~031FH	Prototyping card,
0360H~036FH	Reserved
0378H~037FH	Parallel printer control card 1
0380H~038FH	SDLC, binary synchronous communication 2
03A0H~03AFH	SDLC, binary synchronous communication 1
03B0H~03BFH	Single color display interface card and printer control
03C0H~03CFH	Enhanced color drawing control care
03D0H~03DFH	Color drawing control card
03E0H~03E7H	Drive control card
03F8H~03FFH	Serial transmission control card 1

Table 1-2-2 Memory configurations

PC XT		
Address (hexadecimal)	Function	Description
0000~3FFF	128~256K system version, RAM	System basic memory
40000~9FFFF	Memory extension card plugged into system intension slot	Intension memory
A0000~AFFFF	Reserved	Use of interface cards compatible with PC / XT
B0000~B3FFF	Single color image display	
B4000~B7FFF	Reserved	
B8000~BBFFF	Color / graphic image displayed (only 16K is used)	
BC000~BFFFF	Reserved	
C0000~C7FFF	Reserved	Extension and control of 192K ROM
C8000~CBFFF	Hard disk drive interface card	
CC000~EFFFF	Reserved	
F0000~F3FFF	Reserved	
F4000~F5FFF	Empty sockets left on the system board	
F6000~FDFFF	BASIC interpreter	
FE000~FFFFF	BIOS	

PC AT		
Address (hexadecimal)	Function	Description
000000H~07FFFFH	System board memory	512KB Memory on the motherboard is 512 KB in total
080000H~09FFFFH	Memory extension board	Extend memory to 640K
0A0000H~0BFFFFH	Display buffer	Buffer zone of character and painting
0C0000H~0DFFFFH	Output/input read only extension area	Use of interface card output/input program
0E0000H~0EFFFFH	Read only memory area reserved by the system	Reserved for the user to extend.
0F0000H~0FFFFFH	(BIOS) System basic input/output program area	System's start-up self-test, interrupt service program storage area (BIOS)
100000H~FDFFFFH	Maximum memory area	Extending 15 MB memory
FE0000H~FEFFFFH	System memory reserved area	Reserved for the user to extend.
FF0000H~FFFFFFH	Basic I/O system program area	System's start-up self-test, interrupt service program storage area (BIOS)

1.3 Brief introduction of PC XT/AT interrupt concept

PC XT/AT computer system I/O service can be divided into 3 types: POLLING, INTERRUPT and DMA; to carry out I/O service by means of POLLING, the overall system efficiency is the worst, because its system CPU needs to continuously check related peripherals, causing the system to waste a lot of time on I/O checks; to carry out I/O service by means of INTERRUPT, the system efficiency is high, interrupt request is sent by I/O device to the system, in comparison with the POLLING method, it can reduce the time for CPU needed to check related peripherals; and to carry out I/O service by means of DMA in order to make use of DMA. When interruption occurs, I/O device sends out DMA request, allowing I/O device to exchange data with system memory directly and the data need not be read and written by CPU, thus the overall transmission efficiency is the highest.

As for practice, interruption types can be divided into software interrupts and hardware interrupts. Also PC XT/AT can use 256 interrupts in total and use the method of interrupt vector to facilitate processing-interrupt vector from the computer is achieved by assigning a memory block, which is the so-called interrupt vector table;

the interrupt vector table is located in the memory absolute address scope 0000:0000H~0000:03FFH, each interrupt vector can allocate 4 memory addresses, so that the first address of each interrupt is interrupt vector multiplies 4. Software interrupt is to achieve the goal of interrupt by using INT instruction, still needs to refer to and use the contents of the interrupt vector table, which is quite convenient in practice. PC XT/AT hardware interrupts handle peripheral interrupt request signal with one or two 8259, while peripheral devices informs the system to interrupt by means of hardware implementation. Table 1-3-1 shows the functions of PC XT/AT hardware interrupt request; table 1-3-2 describes the function of hardware direct memory access; and table 1-3-3 lists bit configuration currently used for common X86 series platforms.

Table 1-3-1 Function description of hardware interrupt request

Interrupt signal (interrupt group)	PC XT		PC AT	
	Vector value	Function	Vector value	Function
IRQ0 (1)	8	System counter	8(08H)	System counter
IRQ1 (1)	9	Keyboard	9(09H)	Keyboard
IRQ2 (1)	10	Reserved	The second 8259	
IRQ3 (1)	11	COM2	11(0BH)	COM2
IRQ4 (1)	12	COM1	12(0CH)	COM1
IRQ5 (1)	13	HDD	13(0DH)	LPT2
IRQ6 (1)	14	FDD	14(0EH)	FDD
IRQ7 (1)	15	LPT	15(0FH)	LPT1
IRQ8 (2)	Unused		70(46H)	RTC
IRQ9 (2)			71(47H)	Point to IRQ2
IRQ10 (2)			72(48H)	Reserved
IRQ11 (2)			73(49H)	Reserved
IRQ12 (2)			74(4AH)	Reserved
IRQ13 (2)			75(4BH)	Floating-point unit
IRQ14 (2)			76(4CH)	HDD
IRQ15 (2)			77(4DH)	Reserved
Interrupt priority sequence		0>1>2(8>9>10>11>12>13>14>15)>3>4>5>6>7		

Table 1-3-2 Hardware direct memory access function

Channel number (DMA controller groups)	Function
0 (1)	DRAM update
1 (1)	The secondary DMA controller
2 (1)	Use of floppy disk
3 (1)	Open to use
4 (2)	Connected to the first group of DMA controller
5 (2)	Open to use
6 (2)	Open to use
7 (2)	Open to use
Priority sequence	0>1 (4>5>6>7) >2>3

Table 1-3-3 Current I/O address configuration

Name of I/O system or peripheral interface	I/O address (group)
DMA Controller	00h~0Fh (1) C0h~DFh (2)
Programmable Interrupt Controller	20h, 21h (1) A0h, A1h (2)
Programmable counter (Programmable Interval Timer)	40h~43h (1) 44h~47h (2)
Keyboard Controller	60h~64h
Programmable Option Controller	90h~96h
Floating-point auxiliary operation processor (Math Co-Processor, X87)	F0h~FFh
Secondary IDE interface	170h~177h
Primary IDE interface card	1F0h~1F7h
GAME I/O joystick interface (Game Port)	200h~201h
Sound Card interface	220h~22Fh
PnP Configuration Register	279h, A79h
(Serial Port 4 interface	2E8h~2EFh
Serial Port 2 interface	2F8h~2FFh
MIDI interface (Midi Port)	330h, 331h
Parallel Port interface	378h~37Ah
Single color graphic display interface (MDA/MGA)	3B0h~3BFh
Color graphic display interface (EGA/VGA)	3C0h~3CFh
Display cache (CGA/CRT)	3D4h~3D9h
Serial Port 3 interface	3E8h~3EFh
Floppy Diskette interface	3F0h~3F7h
Enhanced IDE interface	3F6h, 3F7h
Serial Port 1 interface	3F8h~3FFh
PCIconfiguration address cache (PCI Configuration Register/Address)	0CF8h
PCIconfiguration data read/write port (PCI Configuration Register/Data)	0CFCh

1.4 Brief introduction of chips like 8255/8254

For PC XT/AT conventional ISA interfaces, chips like 8255 and 8254 play very important roles. 8255 is a 40 pin programmable peripheral interface chip, which can achieve very control functions through program language software planning; this chip has 24 I/O bits, and is generally divided into three 8 bits I/O ports: A, B, C, or divided into two 12 bits I/O groups: A, B, whereby group A is made up of A port and upper half 4 bits on C port, while group B is made up of B port and lower half 4 bits on port C. The I/O mode of this chip can be divided into 3 types: mode 0, mode 1 and mode 2, port A can operate the above 3 modes, port B can only operate mode 0 and mode 1, and operating mode is determined by controlling the control field of register. 8254 is a 24 pin programmable timer/counter chip, can be used to solve timing control problems, inside which there are 3 independent 16 bit counting-backward counters, which can handle binary and decimal counting, yet its operating mode can be planned into 6 modes such as mode 0 to 5; chip planning action is carried out by controlling the control field of the cache. Figure 1-4-1 shows the external pin diagram and internal structure diagram of 8255 and 8254.

Other ISA interface related chips include 8259 of interrupt control chip, keyboard and display interface chip 8279 and communication element 8251,etc. 8259 is a 28 pin programmable interrupt controller, which can handle the priority Sequence of 8 groups of 8 interrupt requests; PC XT uses a 8259 chip, while PC AT connects two 8259 chips in series, which is connected to the auxiliary 8259 interrupt controller via main 8259 interrupt controller IRQ2, solving the problem of interrupt controller compatibility and extension; request input can be interrupted by connecting several groups of 8259 extension in series. For the uses of ISA interface-related chips, they can be further discussed in books about PC XT/AT.

As with ISA interface operation, when used under peripheral chips like 8255, 8259, etc. It is necessary to set the action mode of peripheral chips, plan the operating modes of chips by defining control characters so as to write the relevant control programs. Table 1-4-1 shows the I/O addresses of relevant chips.

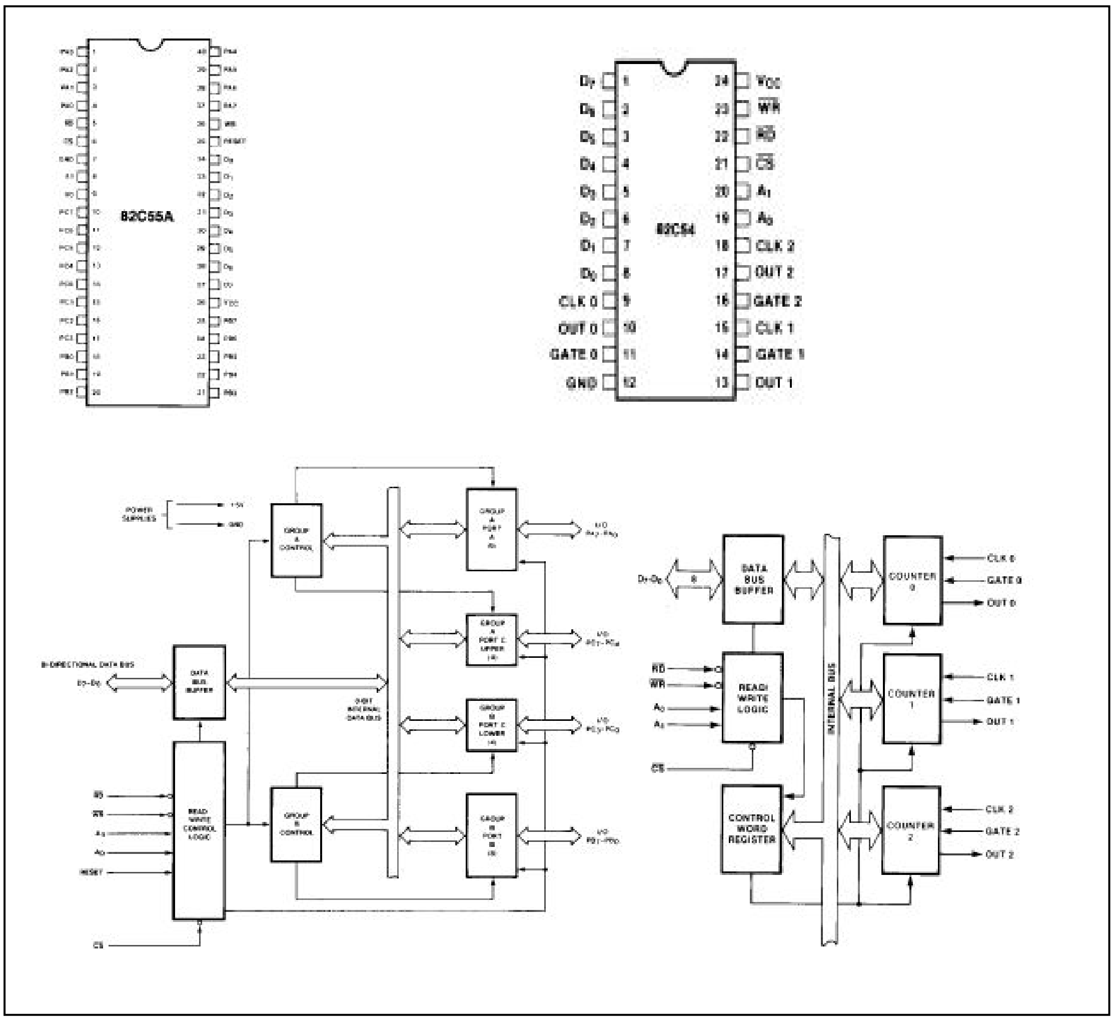


Figure 1-4-1 8255, 8254 pin and internal structure diagrams

Table 1-4-1 ISA interface-related chip I/O address

Chip	Function	Address	Description
8255	PPI_PA	0x300	8255 A port address
	PPI_PB	0x301	8255 B port address
	PPI_PC	0x302	8255 C port address
	PPI_CW	0x303	8255 control character address
8279	D79	0x310	8279 data address
	C79	0x311	8279 control address
8254	C0_54	0x320	8254 counter 0 address
	C1_54	0x321	8254 counter 1 address
	C2_54	0x322	8254 counter 2 address
	C3_54	0x323	8254 control address
8251	D51	0x330	8251 data address
	C51	0x331	8251 control address

1.5 ISA interface brief introduction

Table 1-5-1 is the ISA interface pin diagram, including the definition of another 8 bits extension pins. The signal pins are described by dividing into 4 types: table 1-5-2 describes data bus, table 1-5-3 describes address bus, and table 1-5-4 describes control bus, while the remaining power portion is described in 1-5-5. Figure 1-5-1 is the sketch of ISA interface card, describing the definition of interface card A/B sides.

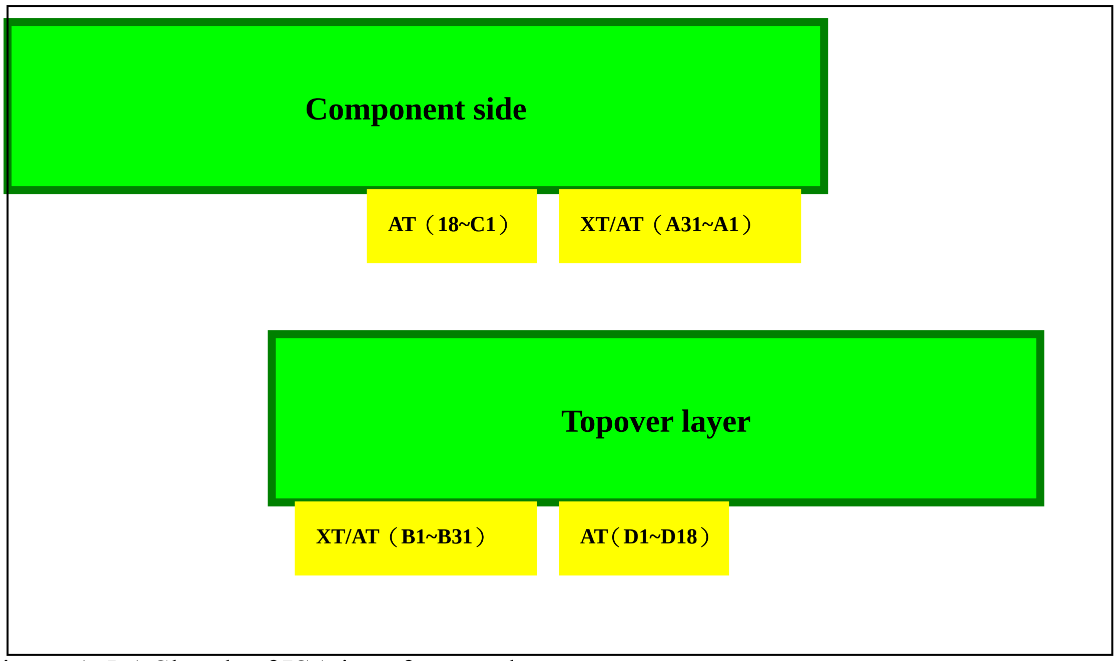


Figure 1-5-1 Sketch of ISA interface card appearance

Table 1-5-1 ISA interface pins

PC AT		PC XT		Pin	Slot	Pin	PC XT		PC AT	
I/O	Signal	I/O	Signal				Signal	I/O	Signal	I/O
G	GND	G	GND	B1		A1	-I/OCHCK	I	-I/OCHCK	I
O	RESET	O	RESET	B2		A2	D7	I/O	SD7	I/O
P	+5	P	+5	B3		A3	D6	I/O	SD6	I/O
I	IRQ9	I	IRQ2	B4		A4	D5	I/O	SD5	I/O
P	-5	P	-5	B5		A5	D4	I/O	SD4	I/O
I	DRQ2	I	DRQ2	B6		A6	D3	I/O	SD3	I/O
P	-12	P	-12	B7		A7	D2	I/O	SD2	I/O
I	OWS	I	NON	B8		A8	D1	I/O	SD1	I/O
P	+12	P	+12	B9		A9	D0	I/O	SD0	I/O
G	GND	G	GND	B10		A10	-I/OCHRDY	I	-I/OCHRDY	I
O	-SMEMW	O	-MEMW	B11		A11	AEN	O	AEN	O
O	-SMEMR	O	-MEMR	B12		A12	A19	I/O	SA19	I/O
I/O	-IOW	I/O	-IOW	B13		A13	A18	I/O	SA18	I/O
I/O	-IOR	I/O	-IOR	B14		A14	A17	I/O	SA17	I/O
O	-DACK3	O	-DACK3	B15		A15	A16	I/O	SA16	I/O
I	DRQ3	I	DRQ3	B16		A16	A15	I/O	SA15	I/O
O	-DACK1	O	-DACK1	B17		A17	A14	I/O	SA14	I/O
I	DRQ1	I	DRQ1	B18		A18	A13	I/O	SA13	I/O
I/O	-REFRESH	I/O	-DACK0	B19		A19	A12	I/O	SA12	I/O
O	CLK	O	CLK	B20		A20	A11	I/O	SA11	I/O
I	IRQ7	I	IRQ7	B21		A21	A10	I/O	SA10	I/O
I	IRQ6	I	IRQ6	B22		A22	A9	I/O	SA9	I/O
I	IRQ5	I	IRQ5	B23		A23	A8	I/O	SA8	I/O
I	IRQ4	I	IRQ4	B24		A24	A7	I/O	SA7	I/O
I	IRQ3	I	IRQ3	B25		A25	A6	I/O	SA6	I/O
O	-DACK2	O	-DACK2	B26		A26	A5	I/O	SA5	I/O
O	T/C	O	T/C	B27		A27	A4	I/O	SA4	I/O
O	ALE	O	ALE	B28		A28	A3	I/O	SA3	I/O
P	+5	P	+5	B29		A29	A2	I/O	SA2	I/O
O	OSC	O	OSC	B30		A30	A1	I/O	SA1	I/O
G	GND	G	GND	B31		A31	A0	I/O	SA0	I/O

PC AT (ONLY)		Pins	Slots	Pins	PC AT (ONLY)	
I/O	Signal				Signal	I/O
I	-MEMCS16	D1		C1	SBHE	I/O
I	-I/OCS16	D2		C2	LA23	I/O
I	IRQ10	D3		C3	LA22	I/O
I	IRQ11	D4		C4	LA21	I/O
I	IRQ12	D5		C5	LA20	I/O
I	IRQ15	D6		C6	LA19	I/O
I	IRQ14	D7		C7	LA18	I/O
O	-DACK0	D8		C8	LA17	I/O
I	DRQ0	D9		C9	-MEMR	I/O
O	-DACK5	D10		C10	-MEMW	I/O
I	DRQ5	D11		C11	SD8	I/O
O	-DACK6	D12		C12	SD9	I/O
I	DRQ6	D13		C13	SD10	I/O
O	-DACK7	D14		C14	SD11	I/O
I	DRQ7	D15		C15	SD12	I/O
P	+5	D16		C16	SD13	I/O
I	-MASTER	D17		C17	SD14	I/O
G	GND	D18		C18	SD15	I/O
P: power		G: ground		I: input	O:output	

Table 1-5-2 data bus

Pin signal	**Pin signal description**	**Transmission direction**
SD0~sD7	Low byte	Bi-directional transmission
SD8~sD15	High byte	Bi-directional transmission

Table1-5-3 address bus

Pin signal	**Pin signal description**	**Transmission direction**
$sA0 \sim sA19$	PC XT base address bus	Bi-directional Transmission
$LA17 \sim LA23$	PC At high byte addressing signal	Bi-directional Transmission

Table 1-5-4 control buses

Signal group	**Pin signal**	**Pin signal description**	**Transmission direction**
System control signals	$RESET$	System reset, power on, system starts	Output
	ALE	Address signal locking	Output
	AEN	DMA and CPU cycle mode	Output
	$\overline{sMEMR}$	Memory read	Output
	$\overline{MEMR}$		Bi-directional
	$\overline{sMEMW}$	Memory write	Output
	$MEMW$		Bi-directional
	$\overline{IOR}$	I/O read	Bi-directional
	$\overline{IOW}$	I/O write	Bi-directional
	$\overline{MEM\quad CS\ 16}$	(O/C, T/S) memory data transmission instruction	Input
	$\overline{I\ /\ O\quad CS\ \ 16}$	I/O data transmission	Input
	$\overline{sBHE}$	High byte transmission start-up	Bi-directional
	$\overline{REFRESH}$	DRAM regeneration instruction	Bi-directional
Clock control signal	OSC	Extension slot clock	Output
	CLK	OSC/3, 1/3 system clock	Output
Asynchronous control signal	$\overline{I/OCH\,RDY}$	Make CPU access slow	Input
	$\overline{OWS}$	Place system access memory into waiting status	Input
Interrupt request signal	IRQ	Hardware interrupt signal	Input
	$\overline{I/OCHCK}$	Parity check error detecting	Input
DMAcontrol signal	DRQ	Direct memory access signal	Input
	$\overline{DACK}$	Response signal sent by DMA	Output
	T/C	TERMINAL COUNT	Output
Double processor coordination	$\overline{MASTER}$	Multi-processor bus coordination	Input

Table 1-5-5 power signals

Voltage value	+5V	-5V	+12V	-12V	GND
Output voltage specification	4.75V~5.25V ±5%	-4.75V~-5.25V ±5%	11.4V~12.6v ±5%	-10.8V~13.2V ±10%	-----
Output current specification	<15A	<0.5A	<5A	<0.3A	-----

1.6 Summary of computer in recent years

In recent years, computers have been breaking away from traditional structure by means of Legacy Free PCs. New generations of computers are designed with brand-new specifications; and in conjunction with newly defined hardware/software interface specifications, a new generation of personal computers have been manufactured. As seen from the motherboard platform layout, free of the constraints of traditional structures in the past and with ISA bus, ISA extension slots discarded, it supports only the latest, self-detecting/setting bus and interface specifications, such as PCI, AGP, USB, SM, Bus, etc; besides, sound card, modem interface cards are also gradually replaced by AMR or CNR interface.

Since Intel advocated USB, so far, most of the motherboards have a built-in USB interface, and USB devices are becoming popular gradually; currently the most popular motherboard is USB1.1 interface, even USB2.0 began to become popular so that traditional medium or low speed interface such as serial ports, parallel ports, joystick and PS/2 interface cards have been gradually abandoned. In addition to the USB interface, PCI interface, AGP display interface card or IEEE1394 interface are enough for common users to use. Table 1-6-1 describes the differences between “traditional” and “new generation” computers.

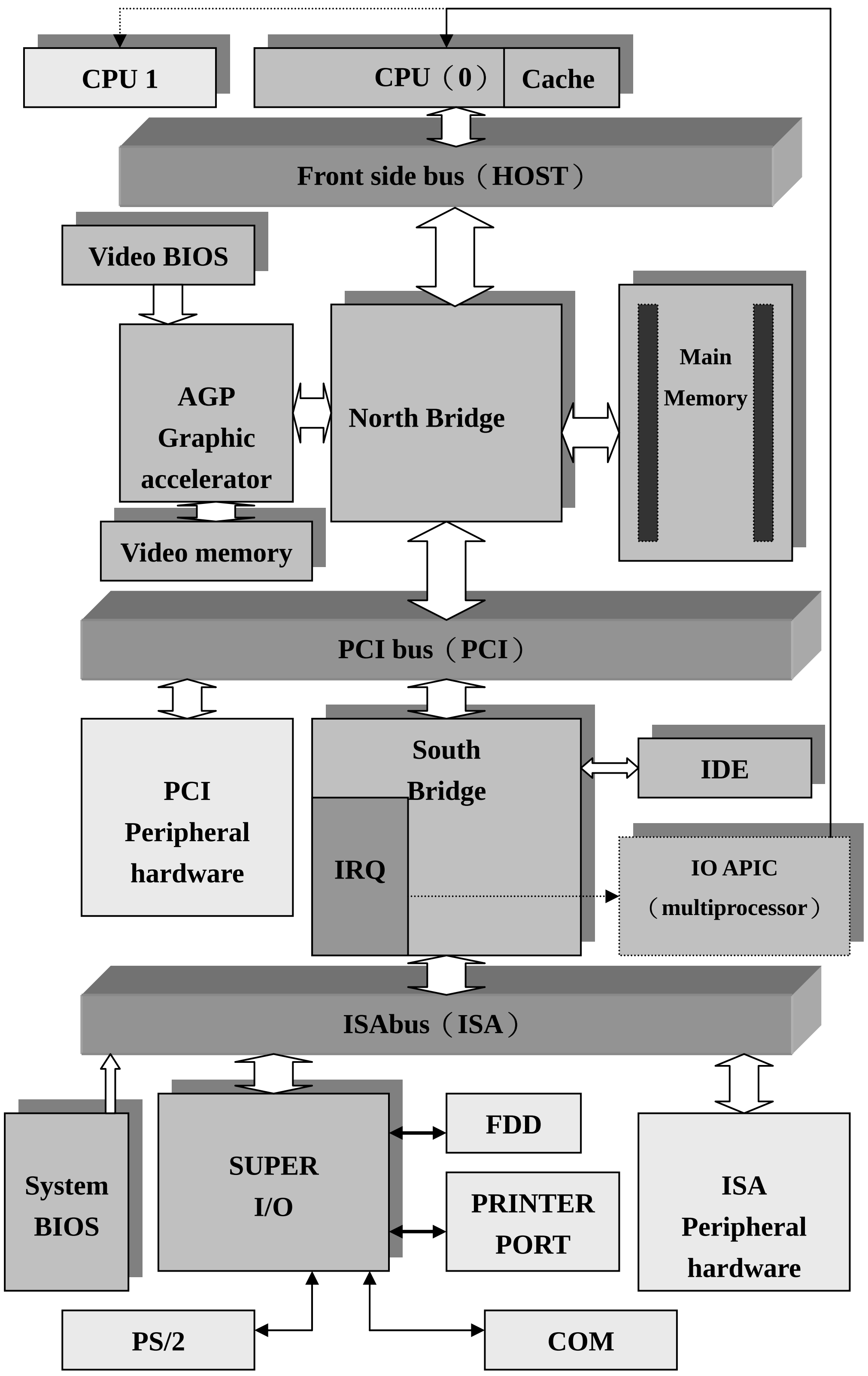


Figure 1-6-1 computer system structure diagrams

Nowadays, computer technology is developing very quickly and there are many chip manufacturers, so the chips are largely identical in terms of specifications and functions, however, there are still some differences in design details. Generally speaking, the structures of today's personal computers are described synoptically with X86 system. Figure 1-6-1 shows the X86 system structure with single/double CPUs. The difference between single and double CPUs lies in the structure of multi-mission hardware environment, so double CPUs need IO-APIC to manage interrupt. Host/PCI is commonly called the North Bridge, connects main processor bus and PCI bus. PCI-to-ISA Bridge is commonly called the South Bridge, connects PCI bus and ISA (or LPC) bus, usually integrating Interrupt Controller, IDE Controller, USB Host Controller, and DMA Controller, the South and North Bridges chipset. Today, many chipsets have replaced PCI interface to the South Bridge end.

Table 1-6-1 the difference between traditional computer and new generation of computers

	Legacy PC (traditional PC)	**Legacy Free PC**
Shape design	Fixed format, dull	Diversified
Main host dimension	ATX, MicorATX	MicorATX, FlexATX
ISABus, ISASlot PCI Bus AGP Bus USB Bus	Yes (via LPC) Yes Yes (not a must) Yes (a must)	None (fully abandoned) Yes Yes (not a must) Yes (a must)
Serial Port COM1＆COM2 Infrared Port (IR) Parallel Port (LPT1) Game I/O	Yes Yes (not a must) Yes Yes	None (completely a must) Yes (not a must) None (fully abandoned) None (fully abandoned)
The function USB device can directly start up the computer	Yes (not a must)	Yes (a must)

1.7 Recent computer structure

The configuration of memory and I/O address for recent computers was nearly completed in the era of 80386, except for some large main hosts, the configuration of memory is largely the same as that of PC AT, and the maximum memory limitation is determined by the chipset. In theory, the maximum memory capacity that can be addressed on computers is 4GB, and basic I/O address is similar to the I/O address of PC AT.

Generally speaking, X86 system can use address line drives, access memory addresses or I/O port drive, there are only 16 I/O ports, so from 00000h~0FFFFh, I/O ports is 2^{16}. In common ISA interface system, only 2^{10}ports (0000h~03FFh) are used, while PCI interfacees use I/O ports over 0400h. PCI interface cards used in this practice are driven by I/O ports.

Also, X86 system designs 256 interrupt vectors from INT 01h to INT Fifth respectively, each vector contains 4 bytes; under real models, the memory address of interrupt vector tables are addressed between 0000: 0000 and 0000: 03FFh, using the 1KB memory space in the forefront. Table 1-7-1 shows the lists of interrupt vector numbers, which is related with the writing of assembly languages.

Table 1-7-1 Interrupt vectors numbers lists

Interrupt number	Property	The description of interruption meaning
INT 00h	(Error)	CPU Divide Error
INT 01h	(Hardware)	CPU Single Step
INT 02h	(Hardware)	NMI (Non Mask able Interrupt)
INT 03h	(Hardware)	Break Point code (instruction break point, instruction code 0CCh)
INT 04h	(Error)	CPU Overflow (Data overflow break)
INT 05h	(Software) (Error)	BIOS process, press Print screen button's print service program interface BOUND Range Exceed
INT 06h	(Error)	Invalid Op Code
INT 07h	(Error)	Floating Point Processor/FPU Not Available
INT 08h	(Hardware)	IRQ0－System Timer (system timer interrupt) INT 8 is generated every 1/18.2 second, used to carry out the work of system timing
INT 09h	(Hardware) (Error)	IRQ1－Keyboard Interrupt Processor Extension Protection ERROR (block selection extension directional error)
INT 0Ah	(Hardware) (Error)	IRQ2－Connected to the auxiliary program of IRQ8~15 Invalid Task State Segment (TSS), Invalid Task State Segment error
INT 0Bh	(Hardware) (Error)	IRQ3－COM2 (secondary serial port) Segment not present (actual memory corresponding to the segment not present)
INT 0Ch	(Hardware) (Error)	IRQ4－COM1　(Primary serial port) CPU Stack Fault, (CPU stack processing fault)
INT 0Dh	(Hardware) (Error)	IRQ5－LPT2 (secondary print port), now it is reserved for PnP system configuration GENERAL PROTECTION VIOLATION CPU produces protection violation error
INT 0Eh	(Hardware) (Error)	IRQ6－Floppy Diskette read/write interrupt PAGEFAULT Switch page fault
INT 0Fh	(Hardware)	IRQ7－LPT1 primary print port
INT 10h	(Software)	Screen I/O interface, switch character/graphics mode. Providing display/paint scroll service
INT 11h	(Software)	PC peripheral equipment check

INT 11h	(Software)	Data/address alignment check error, occurring to CPU above 486
INT 12h	(Software) (Error)	Sending back PC main memory size check MACHINE CHECK EXCEPTION
INT 13h	(Software)	Disk I/O interface (services such as floppy diskette, hard disk read/write, format)
INT 14h	(Software)	Serial Port Communication port interrupt service routine
INT 15h	(Software)	Cassette interface service program, AT extension interrupt service call, numerous programs
INT 16h	(Software)	Keyboard read service program
INT 17h	(Software)	Print service program
INT 18h	(Software)	ROM BASIC entry point address, network card Diskette Start-up interception point
INT 19h	(Software)	Entry point of starting up the operating system
INT 1Ah	(Software)	BIOS time interface/CMOS Real-time Clock battery clock interface
INT 1Bh	(Software)	Interrupt entry point handling program of CTRL BREAK for BIOS
INT 1Ch	(Software)	User timing interrupt, process auxiliary program call from INT 8
INT 1Dh	(Hardware)	Video interface card address table
INT 1Eh		Floppy diskette parameter address table
INT 1Fh		Character dot matrix fonts table address (ASC80h ~ 0FFh)
INT 20h		MS-DOS end program execution
INT 21h	(Software)	MS-DOS API
INT 22h	(Software)	MS-DOS program ends jump address
INT 23h	(Software)	MS-DOS program presses the interrupt entry point of CTRL BREAK
INT 24h	(Software)	MS-DOS Fatal Error Handler
INT 25h	(Software)	DOS Absolute Read
INT 26h	(Software)	DOS Absolute Write
INT 27h	(Software)	Permanent memory interface in. COM file
INT 28h	(Software)	System call of DOS idle state
INT 29h	(Software)	DOS Console Output
INT 2Ah	(Software)	Network software interface layer, Net BIOS
INT 2Bh~2Dh	(Software)	Reserved and unused

INT 2Eh	(Software)	Transmit command queue parameters to DOS command interpreter
INT 2Fh	(Software)	Multiplex Interrupt Multiplex system Interrupt CD-ROM, HIMEM.SYS (XMS), Windows＆DPMI Check
INT 30h	(Software)	Reserved and unused
INT 31h	(Software)	DOS protection mode
INT 32h	(Software)	Reserved and unused
INT 33h	(Software)	Mouse interrupt service
INT 34h~3Eh	(Software)	Used for floating-point unit
INT 3Fh	(Software)	Overlay Manager
INT 40h	(Software)	Disk interrupt service program
INT 41h		Primary hard disk parameter address table
INT 42h	(Software)	INT 10h display service
INT 43h		Dot matrix fonts data table address
INT 44h		Dot matrix fonts data table address
INT 45h	(Software)	Reserved and unused
INT 46h		Secondary hard disk parameter address table
INT 47h~49h	(Software)	Reserved and unused
INT 4Ah	(Software)	CMOS/RTC Alarm Interrupt
INT 4Bh~64h	(Software)	Reserved and unused
INT 65h	(Software)	Audio call service program
INT 66h	(Software)	Reserved and unused
INT 67h	(Software)	LIMEMS service
INT 68h~6Fh	(Software)	Reserved and unused
INT 70h	(Hardware)	IRQ8 CMOS/RTC Time Interrupt
INT 71h	Hardware	IRQ9 (pointing to IRQ2－INT 0Ah)
INT 72h	Hardware	IRQ10 (PnP)
INT 73h	Hardware	IRQ11 (PnP)
INT 74h	Hardware	IRQ12 (PS/2, USB)
INT 75h	Hardware	IRQ13 (Co-Processor – X87)
INT 76h	Hardware	IRQ14 (Primary IDE)
INT 77h	Hardware	IRQ15 (Secondary IDE)
INT 78h~7Fh	(Software)	Reserved and unused
INT 80h~Efh	(Software)	BASIC interpretation program
INT F0h~FFh	(Software)	Reserved and unused
(Software): services of software call; (hardware): IRQ hardware interruption; (error): error detection		

1.8 Brief introduction of related interface

Nowadays, computer systems are developing very quickly and specifications of CPU, memory, peripheral hardware and motherboard chipset are likewise expanding rapidly and varied. However, transmission specifications and interfaces are not easy to change. Interfaces commonly seen on common personal computers include the SM Bus, USB, IEEE1394, IDE, AGP and PCI, etc, while AC97, LPC and SCSI are rarely seen. In the following paragraphs, we will briefly describe interfaces such as AC97, SM Bus, USB, LPC, while AGP, SCSI, IEEE1394 will not be described.

AC'97 (Audio Codec 97)

AC97 is Analog Component 97 (for short AC97), which was introduced by Intel96 when developing NSP MULTI-MEDIA; its latest version is V2.1 (issued on May 22nd, 1998), and became the later CNR 1.0 specification through subsequent extension. AC 97 mainly include analog/digital conversion circuit functions such as computer platforms (motherboard), sound card chip, modem transmission chips, which are divided into Analog Codec and Digital Codec; pure Digital Codec is placed on motherboard, while Analog Codec is located on Riser Card of extension slot. Signals are controlled and transmitted by motherboards (chipsets) with AC-Link, which can reduce the interference from high frequency signals of the motherboard and improve the sound quality of built-in audio chip motherboard. This extension slot is called the AMR slot.

Another characteristic of AC97 is that it can enable low pin and low cost Analog Codec designed in compliance with AC97 specifications, through the increasingly powerful operating abilities of CPU, to simulate 16 bit sound blaster level recording, playing sound and MIDI sound play function. Modem Codec in conformity with AC 97 simulates the basic 56Kbps modem function of V90 specification by means of software, or the two can be integrated into AMC (Audio/Modem Codec) of audio/modem transmission. As with the mixer, it can be outputted from traditional speaker Lineout, and buzzers on the motherboard can be omitted.

Most of the motherboards manufactured recently conform to AC 97 specifications, attaining a set of simple and cheap sound card or 56 Kbps modem functions by opening BIOS, which is enough for users who only type the words, merely surfs the net or places little emphasis on acoustic-optic effects, however, opening the Audio or Modem functions of AC97 can consume the execution efficiency and resources of the CPU, when installing high operating efficiency and sound quality sound card or external modems, motherboards in conformity with AC 97 specifications will automatically detect external sound cards or modems.

Meanwhile, close the AMC elements on the motherboard, and let the hardware sound card/modem to take over.

SM Bus (System Management Bus)

SM Bus (System Management Bus) is the bus contact interface of the two signal lines designed following I2C protocol, which is a low speed interface (80KHz~400KHz) for setting of the detection, positioning, reading and writing parameters of peripheral parts in compliance with SM Bus.

Normally, computer motherboards have built-in SM Bus control circuits (SM Bus Controller) inside south bridge chipsets. The motherboard can, via SM Bus. Detect DRAM and automatically grab timing parameters (SPD) and read the parameters of hardware monitoring chips, monitor CPU, operating temperature and voltage of motherboard and motor speed (RPM) of coolant fan, etc. And, the motherboard of notebook computer detects the electrifical power index of the battery and the possibility to independently close or restart peripheral devices, parts via the SM Bus, temporarily closes peripheral devices and parts power supplies that are not used currently via the SM Bus through ACPI protocol when the entire system needs to enter power saving status to optimize power management.

USB (Universal Serial Bus)

USB is the computer peripheral bus standard jointly developed by computer and communication industry manufacturer such as Compaq, IBM, DEC, NEC, Intel, Microsoft and Northern Telecom, which is a so-called general-purpose serial bus. X86 motherboard was introduced in 1997. In 1999, Apple iMac also adopted USB, speeding the popularity of USB peripheral devices.

USB bus provides extension ability to medium and low speed peripheral devices. Peripheral devices such as keyboard, mouse, joystick, speaker, microphone, modem, cinematograph, through USB interface designs, can be directly connected or removed by means of hot-plugs; computers and OS may automatically detect and enable/disable the device to achieve the objectives of Plug and Play. USB has strong expansion abilities, capable of connecting at most 127 sets of peripheral devices (including USB HUB), the connection distance of each hub to device runs to 5 meters, and USB connector has a special design, which makes it very convenient to install or remove. USBv1.0/1.1 provides maximally 12 Mbps(=1.5MB/s) transmission rate; so far, the motherboards meet the specifications of version 1.1 and the newly approved version 2.0 increased transmission rate to 480Mbps (=60MB/s) , what is more, relevant interface cards, peripheral devices and motherboards have already achieved this function and become the supporting item of future chipsets.

Hardware specification mainly includes Universal HCI (Universal Host Controller Interface) developed by Intel and Open HCI (Open Host Controller Interface) designed and opened by COMPAQ, common USB peripheral devices support these two protocols, with the differences between the two shown in table 1-8-1 below.

Table 1-8-1 USB device hardware specification comparisons

Specifications / Different items	UHCI (Universal Host Controller Interface	OHCI (Open Host Controller Interface)
First published	Jan 15th 1996	Nov 22nd 1995
Manufacturer (IP source)	Intel, exclusive intellectual property	Compaq (Compaq), open intellectual property
Chipset manufacturer	Intel, VIA	Ali, SiS
Setting characteristic	Circuit implementation is easy; the cost is low and easy to integrate, however the efficiency is poor in case of mass transmission	The design of the circuit is complicated, however, it will not affect the CPU and bus efficiency.
Control/addressing mode	IO base (IO Port)	Memory address

Due to the complicated driving methods of USB, normally, USB devices are controlled by operating systems and loaded into drive program so as to be driven and used; in recent years, some BIOSs also support USB control programs and evolve with devices such as USB keyboard, USB floppy diskette, hard disk and drive, provide basic USB device drive and read functions, are capable of start-up using USB keyboard, mouse, USB floppy diskette, hard disk, and even USB CD-ROM, and have even acquired the ability of the USB Device Boot.

IEEE-1394

IEEE-1394 can support high data transmission rate equipment, such as digital video equipment, high performance consumer electronics and PC equipment. IEEE-1394 has heat exchange and Plug-and-Play characteristic. Its equivalent structure provides a method to connect more than two pieces of equipment without the need for a special adapter and complicated settings. Its current transmission rate can reach 400Mbps, much faster than the speeds of serial, parallel, USB, and even PCIs. The IEEE-1394A specification of the standard can support transmission rates of 100, 200, 400Mbps. With the continued development of the standard, new generations of specification-IEEE-1394B will be able to support transmission rates up to 800Mbps. Actually, new generations of products that work at 800Mbps have emerged, and equipment with transmission rates of 1600Mbp will be delivered to the market soon as well. In addition to speed, real reciprocal interface is the key advantage of IEEE-1394. IEEE-1394 is a module made up of physical layer interface equipment and link level controller. It is very difficult to integrate analog and digital technologies, and with transmission rate striding toward 800Mbps, chip designs will face more challenges. Due to the different number of ports supported, some 100Mbps, single TQFP packaging IEEE-1394 chips have only 48 pins, while some have up to 100 pins, capable of supporting 800Mbps and /or more ports, therefore products with IEEE-1394 standards in the market include video editing board, camera, video set-top box, VCR and PC. IEEE-1394 is becoming a PC industrial standard. Consequently, this makes some new multi-media applications possible.

LPC (Low Pin Count) Interface

Published by Intel on Sep 29th, 1997, LPC (Low Pin Count interface) was the new interface specification used to replace ISA Bus. As with control I/O interface such as old ISA extension slot/interface card, ROM BIOS chip, south bridge chip must retain one ISA Bus connecting Super I/O chip to control traditional peripheral devices. Traditional ISA Bus clock falls between 7.159 (14.318MHz Frequency Divider) and ~8.33MHz (PCI clock divides 4); in theory, peak transmission value is 16MB/s (actually it is less than 7 MB/s), but ISA Bus is largely different from PCI Bus in terms of electrical characteristics, signal definition methods, south bridge chips, super I/O chips need to spend more pins to process and clock/line designs of the motherboard also appear to be complicated.

As with LPC interface definition, separate and decode the old ISA address/data and change to the shared decoding methods of PCI address/data signal so that the number of signal lines is greatly reduced and the working clock is synchronously driven by the PCI bus. The improved LPC interface likewise maintains maximum transmission value at about 16MB/s and the number of signal pins is significantly reduced to 25~30. Both Super I/O chips and Flash chips designed with LPC interface can enjoy the benefits of reduced number of pin and smaller sizes, also , the design of the motherboard may be simplified, which is the purpose of LPC－Low Pin Count. Figure 1-8-2 below compares the differences between ISA and LPC.

In case traditional ISA extension slots must be added, 33MHz, 4bit signal of LPC interface is converted into 8MHz, 16bit signal via PCI/ISA bridge chip of LPC interface and connected to ISA interface card. If ISA bridge chip is converted with PCI, it will lead to increases in motherboard design areas and costs, also, the south bridge needs to sacrifice the driving ability of a set of PCI extension slots to drive this single ISA extension slot and its devices. In specification PC99, it was suggested that ISA extension slots and the ISA Bus be abandoned and fully replaced with PCI extension slots and LPC interface. The PC2001 specification presented the concept of Legacy Free PC, whereby all peripheral devices are connected and extended via USB and only four types of interface cards remain: PCI, AGP and CNR/ACR; Super I/O, IR, floppy diskette interface still exist and the I/O methods of control are fully identical, without obvious differences in the writing of software.

Table 1-8-2: Difference between ISA and LPC

	ISA	LPC
Data width	16bit	4bit
Operating frequency	8.33MHz	33.3MHz*1
Addressing space	2^{24}=16MB	2^{32}=4GB
Max transmission bandwidth	16MB/s	16MB/s

1.9 Operation rules applied to computer bits

For computer interfaces, the definition of byte and different scale rules must be first understood. The definition of bytes is as follows: a byte is equal to 8 bits (1-Byte = 8-Bits); a word is equal to 2 bytes (1-Word = 2-Bytes), that is, equal to 16 bits; and a double word is equal to 4 bytes (1-DoubleWord = 4-Bytes), name equal to 32-Bits, and so on, QuadWord is 64 bits.

For different scale rules, binary, decimal and hexadecimal are the commonly seen ones. Binary is expressed as Bin, or the English letter b is added to the mantissa, for instance: “0000 1101b”, “0010b”, “10b” etc. Decimal is mostly expressed as dec. mostly it is omitted, or letter “d” is added to the end to facilitate identification, for example: “29d”, “3729d”, “4d, etc: and hexadecimal is expressed as hex, or “h” is added after the numbers for easier identification, for instance: “9C6Ah”, “4Ah”, “Eh” etc.

Different scale rules can be interchanged. Take 0100 1010b as example for binary, when converted into hexadecimal, it is $0*2^7+ 1*2^6+ 0*2^5+ 0*2^4+ 1*2^3+ 0*2^2+$

$1*2^1+0*2^0 = 74d$"; and because $2^4=16$, with four bits as a group, it can be converted into hexadecimal 0100b = 4d = 4h, 1010b = 10d = Ah", so it is "4Ah" for hexadecimal. To convert decimal number "74d" into binary, first divide 74 by2, the resulting quotient is 37, and the arithmetical compliment is 0, the arithmetical compliment obtained is the constant of 2^0; then divide 37 by 2, the quotient obtained is 18, arithmetical compliment is the constant of 2^1, and so on, the constant of 2^2 is 0, the constant of 2^3 is 1, the constant of 2^4 is 0,and now the quotient obtained is 2, then divided by 2 again, the quotient is 1, the arithmetical compliment is 0, now the arithmetical compliment obtained is the constant of 2^5 , the quotient is the constant of 2^6. In summing up, the converted value is 100 1010b. And for decimal/hexadecimal conversion, we first convert the decimal into binary to compensate 100 1010b" bits to be 0100 1010b", then convert, or convert directly in the manner similar to decimal/binary conversion, divide 16 by 74 to obtain arithmetical compliment and quotient, the quotient is the constant of 16^0 and the arithmetical compliment is the constant of 16^1.

This is because the hexadecimal expresses the numbers more then 10 with table 1-6-1 below, while the conversion result is "4Ah". Hexadecimal/binary conversion is made as shown in table 1-6-2 below. Take "4Ah" as an example. When it is converted from hexadecimal to decimal, it becomes "$4*16^1+ 10*16^0 = 74$".

Those are the three main scale rules in this book, for conversion between different scale rules, please carryout more operational exercises. As PC XT/AT ISA interface mentioned previously is only briefly covered in this chapter, please refer to the many books with in-depth research and discussion of this to acquire more knowledge about computer interfaces. With the rapid development of computers in recent years, the powerful plug and play peripheral equipment and operating system software's have combined to make it easier for users to use computers; consequently, the previous difficulty in setting hardware and software is overcome. In the next chapter, we will discuss the trends of PCs since the year 2000.

Table 1.6.1 Table of definition of hexadecimal numbers

Decimal	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Hexadecimal	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0

Table 1.6.2 Comparison table of hexadecimal/binary conversion

Binary	Hexadecimal	Binary	Hexadecimal
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F
1000	8	0000	0

Exercises:

1.Convert the following binary numbers into decimal numbers.
“00011101”,”1011”,”11”,”11100111”

2.Convert the following binary numbers into hexadecimal numbers
“01011011”,”1101”,”10”,”11000101”

3.Convert the following decimal numbers into binary numbers.
“1024”,”512”,”255”,”65535”,”100”,”4096”,”80”

4.Convert the following decimal numbers into hexadecimal numbers.
“1024”,”512”,”255”,”65535”,”100”,”4096”,”80”

5.Convert the following hexadecimal numbers into binary numbers.
“Abh”,”D1h”,”00FAh”,”10h”,”80h”

6.Convert the following hexadecimal numbers into decimal numbers
“Abh”,”D1h”,”00FAh”,”10h”,”80h”

7.Refer to books on ISA interface, have a try at describing the definitions of ISA pins.

8. Have a try at describing binary addition operation and multiplication

9. Have a try at describing the hexadecimal addition operation.

10. Refer to books on ISA interface, have a try at describing the structure of relevant chips and principle of actions.

Chapter 2 PCI interface

PCI interfaces have many specifications. In this chapter, we will discuss basic principles such as PCI interface signal types, signal pin definitions and instruction sets and transmission modes, etc. The address of this interface is shared with the data bus, in any transmission mode, address signal is transmitted in advance of the data signal, additionally, the signal timing control is very important. For PCI interface-related knowledge, the descriptions in this book are simply fundamental without discussing advanced theory and extended specifications. Table 2-0-1 roughly illustrates the specifications and evolution of PCI interfaces.

Table 2-0-1 Specifications and evolution of PCI interfaces.

Operating frequency	33MHz	33 MHz	66 MHz	133 MHz	266 MHz	533 MHz
Working bits	32bits	64bits	64bits	64bits	64bits	64bits
Peak transmission	133MB/s	266 MB/s	533 MB/s	1066 MB/s	2133 MB/s	----
Commercial	Yes	Yes	Yes	Yes	No	No
Examples of applications	Audio card, network card, U2W SCSIcard, I/O extension card	U160 advanced SCSI card, 1Giga network card	U160 advanced SCSI card, 1Giga network card	U160/U320 advanced SCSI card		
			PCI-X			

2.1 PCI (Peripheral Component Interconnect) brief introduction

Intel developed the PCI bus to ensure the compatibility of different special processor's area bus architectures. The earliest PCI specification published is version 1.0, which became effective on June 2nd, 1992. Rev 2.0 became effective in April 1993. Rev 2.1 was published in the first quarter of 1995. The latest version 2.2 was completed on Dec 18th, 1998 and became effective in Feb 1999.

Devices on the PCI bus can access each other or the system memory very quickly and can be accessed by the processor at speeds close to that of its built-in bus. The transmission of all PCI bus reads and writes can be executed by means of burst, which is superior to interfaces such as ISA and VESA in efficiency, with bus master deciding the length of burst. At the beginning of data transaction, starting addresses and transaction types are assigned to target. However, the target is not informed of its transmission length. When a master is prepared from transmitting data, it will inform the target whether this data is the last batch. The transaction is completed when the last batch of data is transmitted. Table 2-1-1 illustrates the characteristics of PCI interface.

PCI devices can be divided into three types: Target/Slave, PCI Master and Initiator. In normal computer systems, PCI master and initiator are considered as the same device, and both can control PCI bus. The North Bridge of the system usually plays the role of initiator, when there is SCSI interface card that can be turned off (with read only memory) on PCI bus, this interface card can have the actions of PCI master, and can also control this bus. So when the interface cards of bus-related PCI Master function chips have only PCI Master in the system, it must be PCI Initiator.

Typical PCI devices contain a peripheral device adapter packaged in an IC or integrated into a PCI extension card, with commonly seen examples such as network cards, display cards or SCSI cards. After PCI specifications are devised, still many manufacturers continue to use the old devices incompatible with PCI as interface's connected to system motherboard PCI bus, these devices can be used on PCI bus by using Programmable Logic Arrays.

Again, display interface (AGP) can be seen as the extension of PCI interface, mainly provides high bandwidth for the graphic interfaces to use. So far, its product specification has evolved from the early AGP1.0 (1×/2×) to today's AGP3.0 (8×) and the transmission speed has also increased from 32 bits to 256 bits. As with AGP interfaces, they are only used in graphic interfaces and classified as PCI peripherals; however, this is not discussed in this book.

Table 2-1-1 PCI interface characteristics

Characteristics	Description
Irrelevant with processor	Element designed by PCI bus is PCI-specific, separating device design from processor.
At most 80 PCI functions for each PCI bus	Normal PCI bus supports about 10 electricity loads, each device is considered as a load, each device can contain at most 8 PCI functions.
At most 256 PCI bus	Providing at most 256 PCI bus supports.
Low power consumption	Reduce system design that wastes power as possible as practicable, operate with 0MHz in idle state.
Burst transmission can be executed in all read/write transmission	32 bits PCI bus supports read/write transmission of peak transmission rate of 133Mbytes per second. The peak transmission rate of 64 bits PCI is 266Mbytes per second. And the maximum transmission rate of 64 bits, 66MHz PCI bus can reach 533Mbytes per second.
Bus speed	The highest PCI bus speed supported by Rev 2.0 specification is 33MHz., while 66MHz bus operation is added to Rev 2.1.
64 bits bus width	Complete 64 bits extension definition
Access time	60ns (When initiator or Master of PCI bus is writing PCI Target and bus clock is 33MHz).
Parallel bus operation	Bridge supports complete bus parallel, processor, PCI bus and extension bus can be used at the same time.
Support bus master	Complete bus master support allows peer-to-peer PCI bus access, access, via PCI-to-PCI and extension bus bridge, it accesses main memory extension bus device. PCI master can access the target of lower level PCI bus in the system.
Hidden bus arbitration	PCI bus arbitration is required when multiple groups of bus masters are transmitting on PCI bus.
Low pin counts	Reduce the use of bus signal pins, functional PCI target uses only 47 pins, initiator uses only 49 pins
Check of transaction integrality	Parity check with address, instructions and data.
Automatic configuration	With specification that supports automatic device detecting and configuration bit-level Configuration Register.
The thoroughness of software	When the software drive program is communicating with PCI device or devices associated with its extension bus, the same instruction sets and status definitions can be used.

2.2 PCI connector and pin

For the PCI interfaces used, the sketch of PCI interface signal group is shown in Figure 2-2-1, connector specification description is shown in Figure 2-2-2, while the configuration of signal pins is shown in Figure 2-2-1, with the definition of each signal described in the following section.

Generally speaking, PCI pins are divided into required pins and optional pins, as shown in Figure 2-2-1. The required pins include data and address bus pins (AD[31:0]), interface control pins (FRAME#, TRDY#, IRDY#, etc.), error signal pins (PERR#, SERR#), etc. Each PCI peripheral must have this type of pins; optional pins include 64 bits extension pins, LOCK#, interrupt pins (INT#[3:0]), JTAG insect; this type of pins can be selected as required. Figure 2-2-2 shows the specification of 32 bits PCI interface slots; on the left of the figure is the rear baffle of computer. Form observing normal motherboard, we can learn that most of them use PCI slots of 5V specification, this slot can use general or 5V specification PCI interface cards; 64 bits PCI slots are mostly 3 V specification. However, these types of slot and interface cards are rarely seen.

And 64 bits connectors are the extension of basic interface and are listed in Figure 2-2-1. Attention should be paid to B49 pin, which is ground signal when operating at 3.3V/33MHz and M66EN when operating at 3.3V/66MHz; there should be a pull-up resistor on the system board and small capacitor need to be connected to interface cards to remove electrical coupling.

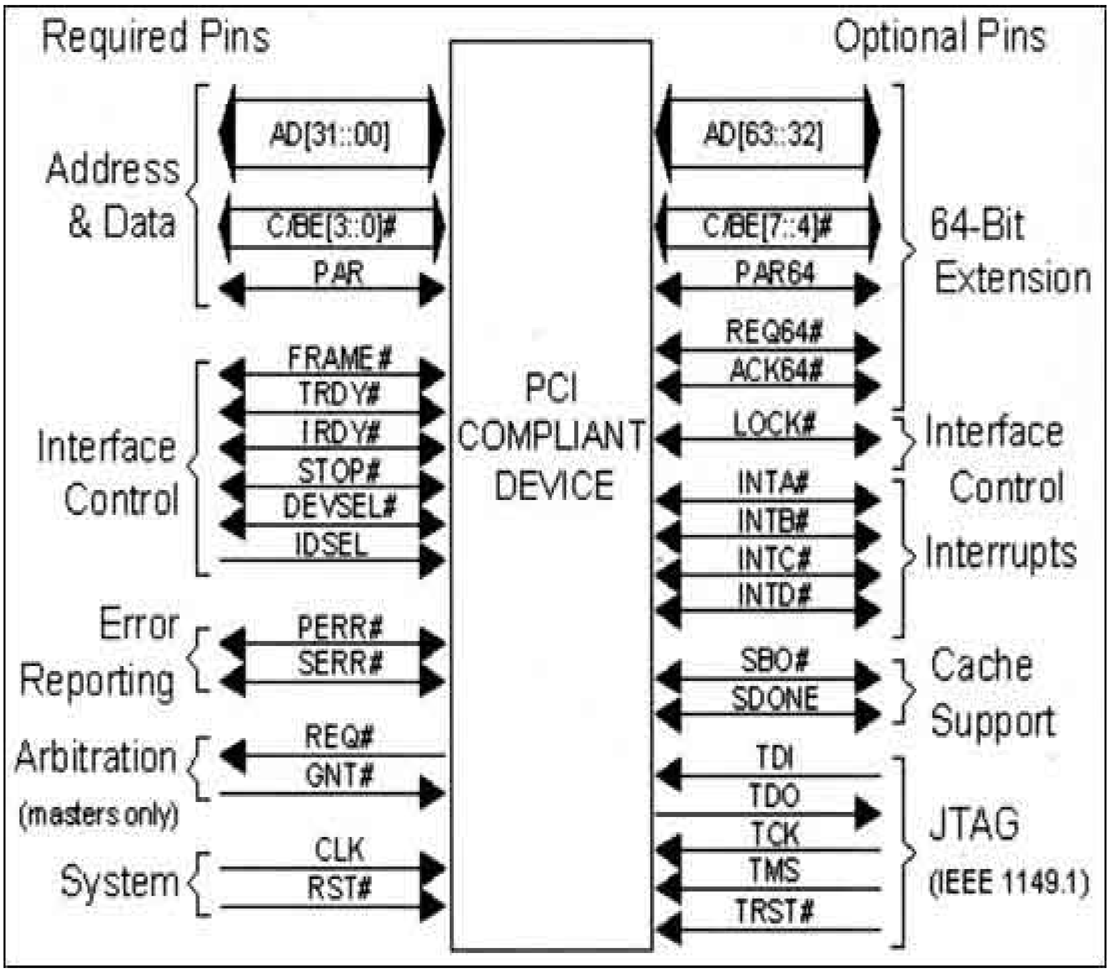


Figure 2-2-1 PCI signal groups

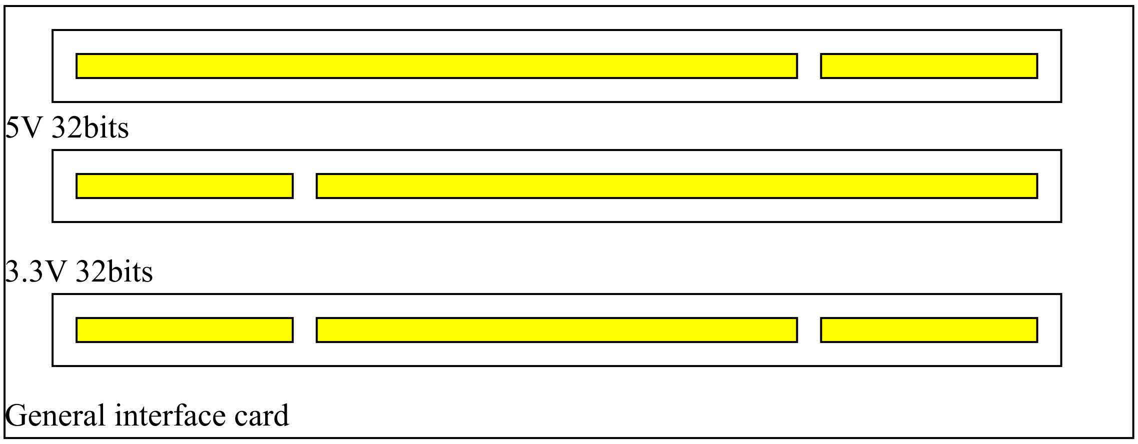


Figure 2-2-2 PCI 32 bits connector specification

Table 2-2-1 Pin configuration

Pins	5V interface card		3.3V interface card		General interface card		Remark
	A side	B side	A side	B side	A side	B side	
1	-12V	TRST#	-12V	TRST#	-12V	TRST#	Baffle head
2	TCK	+12V	TCK	+12V	TCK	+12V	
3	GND	TMS	GND	TMS	GND	TMS	
4	TDO	TDI	TDO	TDI	TDO	TDI	
5	+5V	+5V	+5V	+5V	+5V	+5V	
6	+5V	INT#A	+5V	INT#A	+5V	INT#A	
7	INT#B	INT#C	INT#B	INT#C	INT#B	INT#C	
8	INT#D	+5V	INT#D	+5V	INT#D	+5V	
9	PRSNT#1	Reserved	PRSNT#1	Reserved	PRSNT#1	Reserved	
10	Reserved	+5V	Reserved	+3.3V	Reserved	+Vi/o	
11	PRSNT#2	Reserved	PRSNT#2	Reserved	PRSNT#2	Reserved	
12	GND	GND	KEY				3.3V_KEY
13	GND	GND					
14	Reserved	+3.3Vaux	Reserved	+3.3Vaux	Reserved	+3.3Vaux	
15	GND	RST#	GND	RST#	GND	RST#	
16	CLK	+5V	CLK	+3.3V	CLK	+Vi/o	
17	GND	GNT#	GND	GNT#	GND	GNT#	
18	REQ#	GND	REQ#	GND	REQ#	GND	
19	+5V	PME#	+3.3V	PME#	+Vi/o	PME#	
20	AD [31]	AD [30]	AD [31]	AD [30]	AD [31]	AD [30]	

21	AD [29]	+3.3V	AD [29]	+3.3V	AD [29]	+3.3V	
22	GND	AD [28]	GND	AD [28]	GND	AD [28]	
23	AD [27]	AD [26]	AD [27]	AD [26]	AD [27]	AD [26]	
24	AD [25]	GND	AD [25]	GND	AD [25]	GND	
25	+3.3V	AD [24]	+3.3V	AD [24]	+3.3V	AD [24]	
26	C/BE#[3]	IDSEL	C/BE#[3]	IDSEL	C/BE#[3]	IDSEL	
27	AD [23]	+3.3V	AD [23]	+3.3V	AD [23]	+3.3V	
28	GND	AD [22]	GND	AD [22]	GND	AD [22]	
29	AD [21]	AD [20]	AD [21]	AD [20]	AD [21]	AD [20]	
30	AD [19]	GND	AD [19]	GND	AD [19]	GND	
31	+3.3V	AD [18]	+3.3V	AD [18]	+3.3V	AD [18]	
32	AD [17]	AD [16]	AD [17]	AD [16]	AD [17]	AD [16]	
33	C/BE#[2]	+3.3V	C/BE#[2]	+3.3V	C/BE#[2]	+3.3V	
34	GND	FRAME#	GND	FRAME#	GND	FRAME#	
35	IRDY#	GND	IRDY#	GND	IRDY#	GND	
36	+3.3V	TRDY#	+3.3V	TRDY#	+3.3V	TRDY#	
37	DEVSEL#	GND	DEVSEL#	GND	DEVSEL#	GND	
38	GND	STOP#	GND	STOP#	GND	STOP#	
39	LOCK#	+3.3V	LOCK#	+3.3V	LOCK#	+3.3V	
40	PERR#	Reserved	PERR#	Reserved	PERR#	Reserved	
41	+3.3V	Reserved	+3.3V	Reserved	+3.3V	Reserved	
42	SERR#	GND	SERR#	GND	SERR#	GND	
43	+3.3V	PAR	+3.3V	PAR	+3.3V	PAR	
44	C/BE#[1]	AD [15]	C/BE#[1]	AD [15]	C/BE#[1]	AD [15]	
45	AD [14]	+3.3V	AD [14]	+3.3V	AD [14]	+3.3V	
46	GND	AD [13]	GND	AD [13]	GND	AD [13]	
47	AD [12]	AD [11]	AD [12]	AD [11]	AD [12]	AD [11]	
48	AD [10]	GND	AD [10]	GND	AD [10]	GND	
49	GND	AD [09]	G/M66EN	AD [09]	G/M66EN	AD [09]	Used for 66MHz
50	KEY		GND	GND	KEY		5V_KEY
51			GND	GND			
52	AD [08]	C/BE#[0]	AD [08]	C/BE#[0]	AD [08]	C/BE#[0]	
53	AD [07]	+3.3V	AD [07]	+3.3V	AD [07]	+3.3V	
54	+3.3V	AD [06]	+3.3V	AD [06]	+3.3V	AD [06]	
55	AD [05]	AD [04]	AD [05]	AD [04]	AD [05]	AD [04]	
56	AD [03]	GND	AD [03]	GND	AD [03]	GND	

57	GND	AD [02]	GND	AD [02]	GND	AD [02]	
58	AD [01]	AD [00]	AD [01]	AD [00]	AD [01]	AD [00]	
59	+5V	+5V	+3.3V	+3.3V	Vi/o	Vi/o	
60	ACK#64	REQ#64	ACK#64	REQ#64	ACK#64	REQ#64	
61	+5V	+5V	+5V	+5V	+5V	+5V	
62	+5V	+5V	+5V	+5V	+5V	+5V	
(The above are 32bits_PCI) ------KEY------ (The followings are 64bits_PCI)							64bits Separation
63	Reserved	GND	Reserved	GND	Reserved	GND	
64	GND	C/BE#[7]	GND	C/BE#[7]	GND	C/BE#[7]	
65	C/BE#[6]	C/BE#[5]	C/BE#[6]	C/BE#[5]	C/BE#[6]	C/BE#[5]	
66	C/BE#[4]	+5V	C/BE#[4]	+3.3V	C/BE#[4]	Vi/o	
67	GND	PAR64	GND	PAR64	GND	PAR64	
68	AD [63]	AD [62]	AD [63]	AD [62]	AD [63]	AD [62]	
69	AD [61]	GND	AD [61]	GND	AD [61]	GND	
70	+5V	AD [60]	+3.3V	AD [60]	Vi/o	AD [60]	
71	AD [59]	AD [58]	AD [59]	AD [58]	AD [59]	AD [58]	
72	AD [57]	GND	AD [57]	GND	AD [57]	GND	
73	GND	AD [56]	GND	AD [56]	GND	AD [56]	
74	AD [55]	AD [54]	AD [55]	AD [54]	AD [55]	AD [54]	
75	AD [53]	+5V	AD [53]	+3.3V	AD [53]	Vi/o	
76	GND	AD [52]	GND	AD [52]	GND	AD [52]	
77	AD [51]	AD [50]	AD [51]	AD [50]	AD [51]	AD [50]	
78	AD [49]	GND	AD [49]	GND	AD [49]	GND	
79	+5V	AD [48]	+3.3V	AD [48]	Vi/o	AD [48]	
80	AD [47]	AD [46]	AD [47]	AD [46]	AD [47]	AD [46]	
81	AD [45]	GND	AD [45]	GND	AD [45]	GND	
82	GND	AD [44]	GND	AD [44]	GND	AD [44]	
83	AD [43]	AD [42]	AD [43]	AD [42]	AD [43]	AD [42]	
84	AD [41]	+5V	AD [41]	+3.3V	AD [41]	Vi/o	
85	GND	AD [40]	GND	AD [40]	GND	AD [40]	
86	AD [39]	AD [38]	AD [39]	AD [38]	AD [39]	AD [38]	
87	AD [37]	GND	AD [37]	GND	AD [37]	GND	
88	+5V	AD [36]	+3.3V	AD [36]	Vi/o	AD [36]	
89	AD [35]	AD [34]	AD [35]	AD [34]	AD [35]	AD [34]	
90	AD [33]	GND	AD [33]	GND	AD [33]	GND	
91	GND	AD [32]	GND	AD [32]	GND	AD [32]	

92	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	
93	Reserved	GND	Reserved	GND	Reserved	GND	
94	GND	Reserved	GND	Reserved	GND	Reserved	
A side		Component side		B side		Top over layer	
Remark: it is low logic when the signals are driven, all such signals end with "#", conversely when it is high, then signals have no "#" followed.							

2.3 PCI interface pin definition

Electricity definition of digital circuits used on PCI interface may be classified into five types: INPUT, OUTPUT, T/S (Tri-State), S/T/S (Sustain Tri-State) and O/D (Open Drain). I/O signal is a standard signal only; T/S is standard tristate I/O signal, whose logic diagram and truth table are shown in Figures 2-3-1 and 2-3-1; similar to T/S signal, S/T/S is the continuous signal that can be driven by only one device. After initiating driving of this signal, to stop driving this signal, it is necessary to pull up this signal for one clock and then floating this pin and pull-up resistor is required to keep this signal strong during intervals. O/D is an open drain signal, requiring an additional pull-up resistor to maintain its backdrive, roughly, it requires 3 clock cycles to meet the demands of backdrive actions, with logic diagrams and truth tables shown in Figures 2-3-2 and 2-3-2. The following is the necessary signal definition and signal type description.

As with signals on PCI BUS, normal PCI device needs to use 47 signal pins, excluding PERR# and SERR#, with the definitions of these signals described as follows:

◎ **CLK (Input) "working frequency"**
CLK provides the timing of all transmission on PCI BUS, except RST#, INTA#, INTB#, INTC#, and INTD#, the remaining PCI signals are sampled on rising edges of CLK.

◎ **RST# (Input) "RESET SIGNAL"**
As long as RST# ACTS, all PCI output signals must enter Tri-State (it is floating for O/D type signals), to prevent signal such as AD, C/BE# and PAR floating during resetting, system chips may pull these signals low. Although RST# is asynchronous with CLK, system chips must ensure that it is a clean signal without bounces.

◎ **AD[31:00] (T/S) "data and address bus"**
Address and data lines operate using AD[31:00] pins. When the first Frame# on

the low CLK rising edge, AD[31:00] represents address line; when IRDY# and TRDY# are both on low CLK rising edges, it represents a data line. When this type of pin is used for the address line, for I/O, AD[31:00]32 are needed, for configuration and memory, only AD[31:02] is address line, and [01:00] is otherwise used. When this type of pin is used for data line, AD[07:00] is the data of lowest byte (LSB), while AD[31:24] is the data of highest byte (MSB).

◎ **C/BE[3:0]# (T/S) "control signal"**

Not only is the address line and data line multiplexed together, but also bus instruction and byte enable are multiplexed together. When the first Frame# is on the rising edge of low CLK, C/BE[3:0]# represents bus instruction, in the rest of the time, C/BE[3:0]# represents byte enable, indicating which data transmitted is byte enable among the four bytes of data line, so C/BE[3:0] is used on PCI BUS to identify, and its corresponding relation is that C/BE#[0] corresponds lowest byte; C/BE#[3] corresponds highest byte(MSB), and low represents corresponding byte, namely actually transmitted data.

◎ **PAR (T/S) "parity check signal"**

PCI BUS uses the method of Even Parity", that is , all "1" numbers on AD[31:00], C/BE[3:0] and PAR should be even. The key is to check whether there is data transmission error, as with timing, it will lag behind by only one CLK.

◎ **FRAME# (S/T/S) "Transmission control signal"**

As with data transmission on bus, PCI_Bus_Master converts Frame# from high low, if Frame# continues to low, it means that data is transmitting; when Frame# is changed from low to high, it means that the last batch of data is waiting to be transmitted.

◎ **IRDY# (S/T/S) "Initiator Ready signal"**

For writing, when it is placed low, it means PCI_Bus_Master has got the data ready on bus. While for reading, it means that PCI_Bus_Master is ready to receive data.

◎ **TRDY# (S/T/S) "Target Ready signal"**

For writing, when it is low, it means that the PCI_Target is ready to receive data; while for reading, it means that data is ready on bus.

◎ **STOP# (S/T/S) “interrupting transmission signal”**

PCI_Target may request PCI_Master to interrupt the current transfer cycle via this signal.

◎ **DEVSEL# (S/T/S) “Device Select signal”**

The return signal that is used by PCI_Target to notify PCI_Master to select the signal

◎ **IDSEL (Input) “Initialization Device Select”**

A special signal that is used for Configuration Read/Write transfer Each PCI Slot in the system can select any of the signals in AD[31:11]. It is not connected to its IDSEL and can not be repeated, mainly used in Plug & Play operating environment.

◎ **REQ# (T/S)**

The device that has the ability to master PCI sends requests to acquire the bus control from PCI_Initiator of the system via this signal, thus becoming the PCI_Bus_Master.

◎ **GNT# (T/S) “signal Grant”**

Enable the device that wants to acquire bus control to become new matter on the bus. REQ# and GNT# become pairs of point-to-point signal, on each slot, there are independent REQ# and GNT# connected with system chips.

The above signals are commonly used signal pins for PCI interfaces, and the remainders are optional pins. Apart from pins added by 64 bits extension mode, JTAG is also a characteristic of PCI interface. Because this type of signal is capable of monitoring chips on PCI peripherals, it can significantly increase the reliability of this interface in use, however, only a few advanced peripheral use this signal pins. Figure 2-3-3 shows the pin diagram of 33MHz/32 bits PCI interface relative to PCI interface pins definitions. In this chapter, we just briefly describe it and in the next chapter, we will discuss the types of C/BE#[3:0] and configuration cache, which can be considered as important information in using PCI interface device.

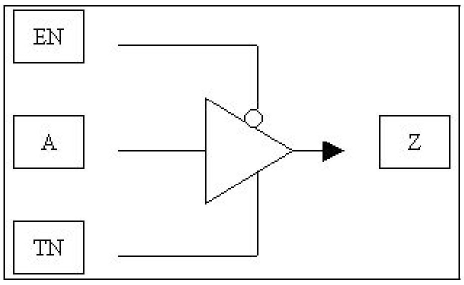


Figure 2-3-1 Tri-state logic symbol diagram

Table 2-3-1 Tri-state logic truth table

Input			Output
EN	A	TN	Z
L	L	H	L
L	H	H	H
H	×	×	Hold
×	×	L	Hold

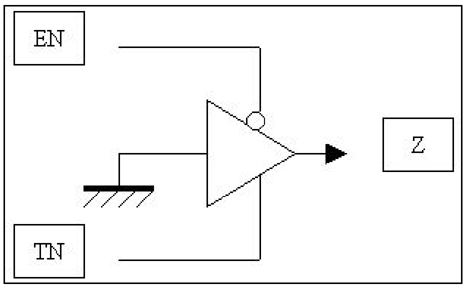


Figure 2-3-2 Open drain logic symbol diagram

Table 2-3-2 Open drain logic truth table

Input		Output
EN	TN	Z
L	H	L
H	×	Hold
×	L	Hold

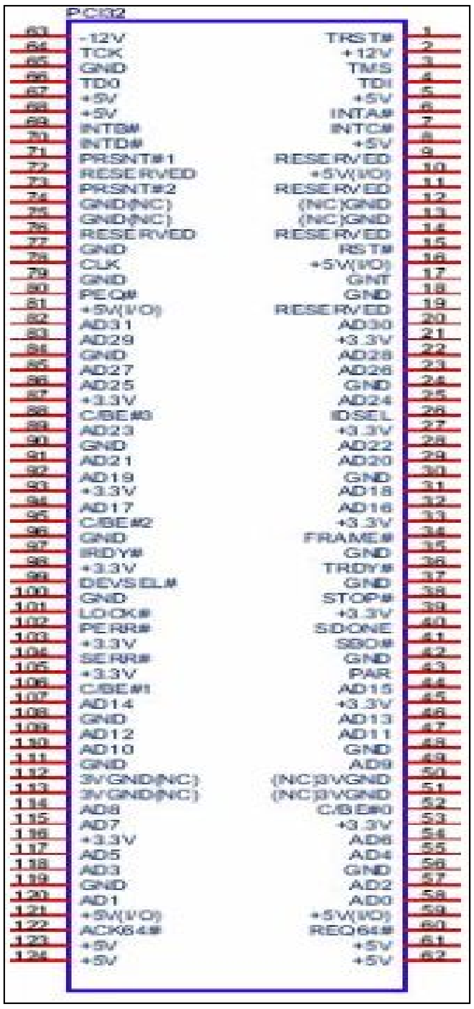


Figure 2-3-3 33MHz/32 bits PCI pin diagram

2.4 PCI instructions

C/BE#[3:0] and configuration cache are important information in designing PCI interfaces. Table 2-4-1 illustrates PCI instructions types, where PCI bus is Little_Endian bus; table 2-4-2 describes the significance of data phase byte enable.

PCI bus transmits data on AD[31:0], while C/BE#[3:0] transmits instruction and address enable signal. Generally, PCI signal transmission can be divided into two parts: address phase and data phase. During address phase, signal on C/BE#[3:0] is PCI instruction, as is shown in Figure 2-4-1 below is the signal configuration of this phase of C/BE#[3:0], now data address bus AD[31:0] transmits address, during data phase, signal on C/BE#[3:0] is byte enable signal, mainly defining AD[31:0] bus in this phase to transmit data. As with what are the effective bytes on AD[31:0], table 2-4-2 will give a definition for it.

Table 2-4-1 PCI instructions types

C/BE#				**Instructions types**
3	2	1	0	
0	0	0	0	Interrupt Acknowledge
0	0	0	1	Special Cycle
0	0	1	0	I/O Read
0	0	1	1	I/O Write
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Memory Read
0	1	1	1	Memory Write
1	0	0	0	Reserved
1	0	0	1	Reserved
1	0	1	0	Configuration Read
1	0	1	1	Configuration Write
1	1	0	0	Memory Read Multiple
1	1	0	1	Dual Address Cycle
1	1	1	0	Memory Read Line
1	1	1	1	Memory Write and Invalidate

Table 2-4-2 Definition of data phase byte enables

C/BE#				Definition	
3	2	1	0	Data path numbers	Byte of transmission addressing dword
0	0	0	0	4	All the 4
0	0	0	1	3	The higher 3
0	0	1	0	3	The higher 2 and the lowest 1
0	0	1	1	2	The higher 2
0	1	0	0	3	The highest 1 and the lower 2
0	1	0	1	2	The highest 1 and the third highest 1
0	1	1	0	2	The highest 1 and the lowest 1
0	1	1	1	1	The highest 1
1	0	0	0	3	The lower 3
1	0	0	1	2	The second highest 1 and the third highest 1
1	0	1	0	2	The second highest 1 and the lowest 1
1	0	1	1	1	The second highest 1
1	1	0	0	2	The lower 2
1	1	0	1	1	The third highest 1
1	1	1	0	1	The lowest 1
1	1	1	1	0	Unused

Byte enable signal	Data path	Addressing location
C/BE#[3]	3 AD[31:24]	The fourth location in the addressed dword
C/BE#[2]	2 AD[23:16]	The third location in the addressed dword
C/BE#[1]	1 AD[15:08]	The second location in the addressed dword
C/BE#[0]	0 AD[07:00]	The first location in the addressed dword

2.5 Configuration address port and configuration transaction

Generally, the configuration mechanism of PCI peripheral is divided into two types: configuration mechanism #1 and configuration mechanism #2. PCI 2.2 Version has deleted configuration mechanism #2 and each PCI function on each PCI device needs 64-dword special configuration cache; for X86 system, I/O address is 64KB maximally. When it is not in conflicts with old interface, the I/O addresses that can be used by the PCI interface fall between 0400h~04FFh, 0800h~08FFh and 0C00h~0CFFh.

Configuration mechanism #1 uses two 32 bits I/O ports: address 0CF8h and 0CFCh. 0CF8h~0CFBh is 32 bits Configuration Address Port, while 0CFCh~0CFFh32 is Configuration Data Port. Their definitions are shown in table 2-5-1, with definition of each bit described as follows:

Table 2-5-1 Configuration address port

0CFBh								0CFAh								0CF9h								0CF8h							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved							bus no.								device no.					function no.			D word						0__0	

bit [0:1]:hardware connection, read only, must be 0 when writingBit [7:2]: d_word that confirms Target function configuration

Bit [10:8]: function number parameters of PCI Target device

bit [15:11]: PCI Target device numbers

bit [23:16]: PCI Target bus numbers

bit [30:24]: all are “0”bit [31]: configuration access is “0” , bus I/O access is “1”

Configuration transaction can be divided into two types, Type 0 and Type 1. When conducting configuration transactions, the lowest valid bit for AD bus is 00b in case of Type 0 and 01b in case of Type 1 configuration type. FRAME# is driven low, IDSEL signal can only correspond to Type 0 configuration transaction, C/BE#[3:0] instructs the actions of configuration transactions. In the following paragraph, we will describe the two types of configuration transactions respectively.

As with configuration transaction Type 0, AD [1:0] is 00b, it is Type 0 configuration transaction, AD [7:2], AD [7:2] means dword of Target configuration, AD [10:8] means functions of Target device, AD [31:11] is reserved and not read, and is interpreted by IDSEL into device numbers.

IDSEL implementation examples are shown in Figures 2-5-1 and 2-5-2. Figure 2-5-1 connects IDSEL to the unused AD lines, AD [31:11] is not used during Type 0 configuration transaction address phase. Host/PCI bridge configuration address port bit [15:11] is responsible for interpretation. This method can correspond to 21 devices and reduce the numbers of Host/PCI bridges pins and connections. Figure 2-5-2 shows the connecting methods of added resistance coupling, which has been added to bus stability; and the other IDSEL implementation method is that Host/PCI Bridge completes decoding actions together with digital logic circuits.

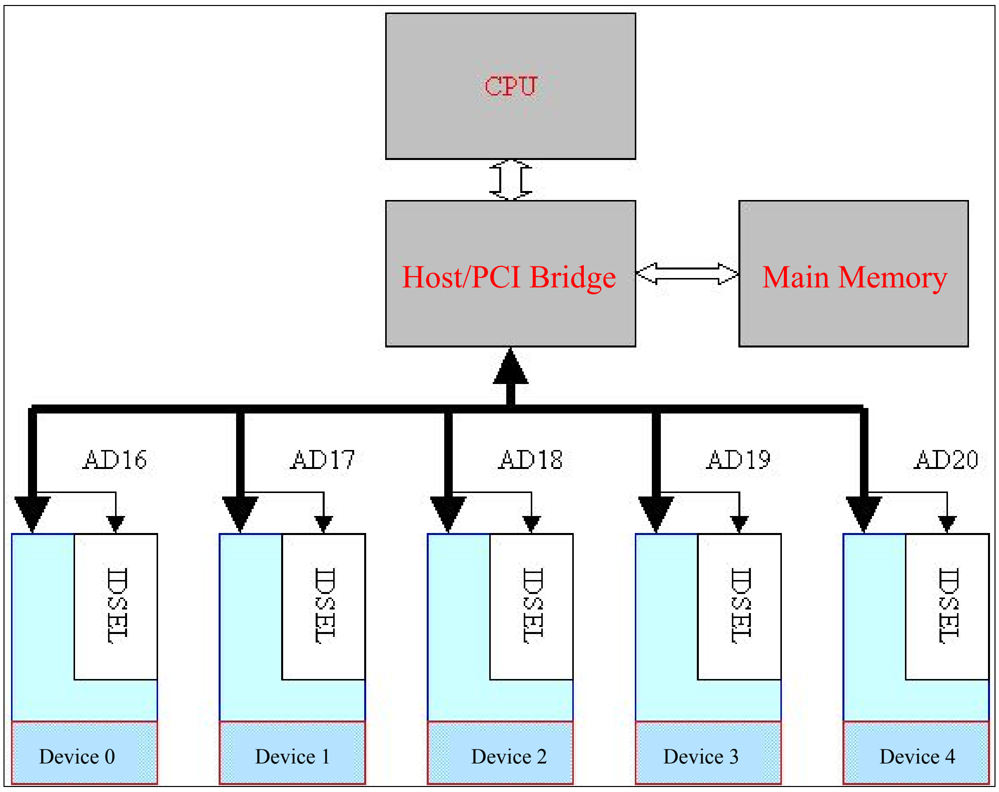


Figure 2-5-1 IDSEL example of implementation (without decoupling resistance)

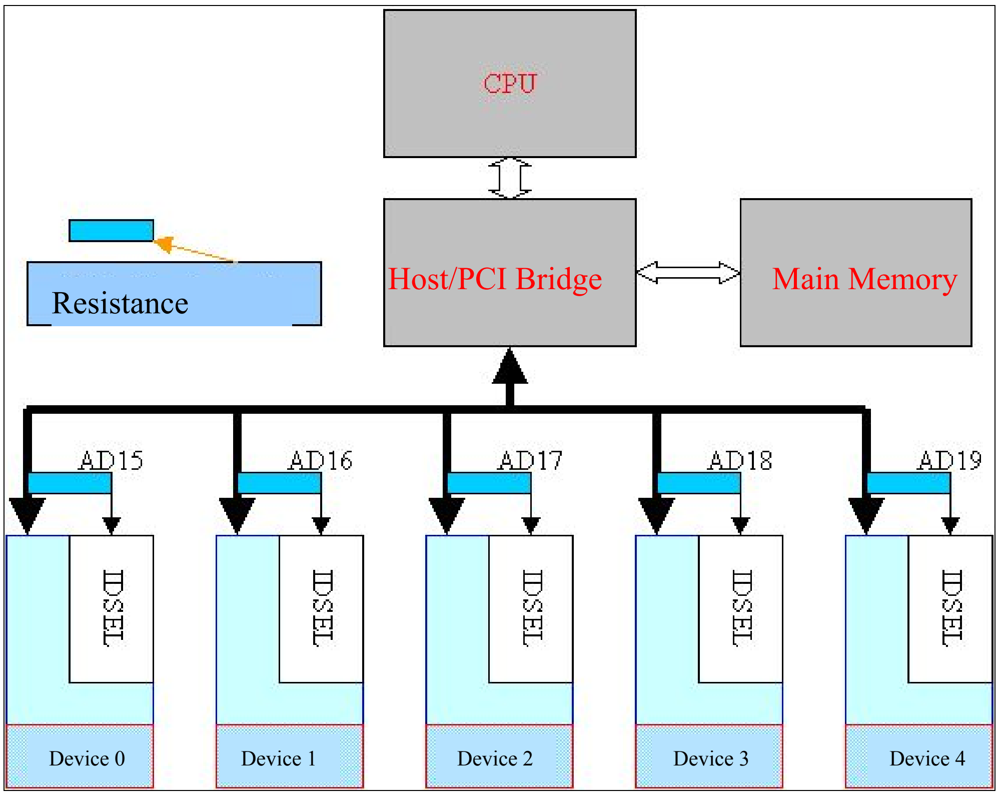


Figure 2-5-2 IDSEL example of implementation (with coupled resistance)

Address phase: Another IDSEL implementation method is to interpret with digital logic circuit. Figure 2-5-3 shows example of configuration read using this method, which is described with by the address and data phase.

・PCI Initiator reads PCI Target
・ FRAME# to below, transaction begins
・AD[1:0]=00b－＞Type 0
AD[7:2] －＞ configuration dword address
AD[10:8] －＞function numbers
・C/BE#[3:0]==1010b (configuration read)
・PCI Initiator shows it can read data.
・PCI Target shows it can transmit data.
・Type 0 needs IDSELto select device.

Data phase
・PCI Initiator reads PCI Target
・FRAME# counter drive shows it is ready to complete the last data transmission phase.
・PCI Target transmits configuration data to PCI Initiator.
・Bridge duplicates byte enable of processor to C/BE#[3:0]
・ PCI Initiator reads data in case of IRDY# and TRDY#

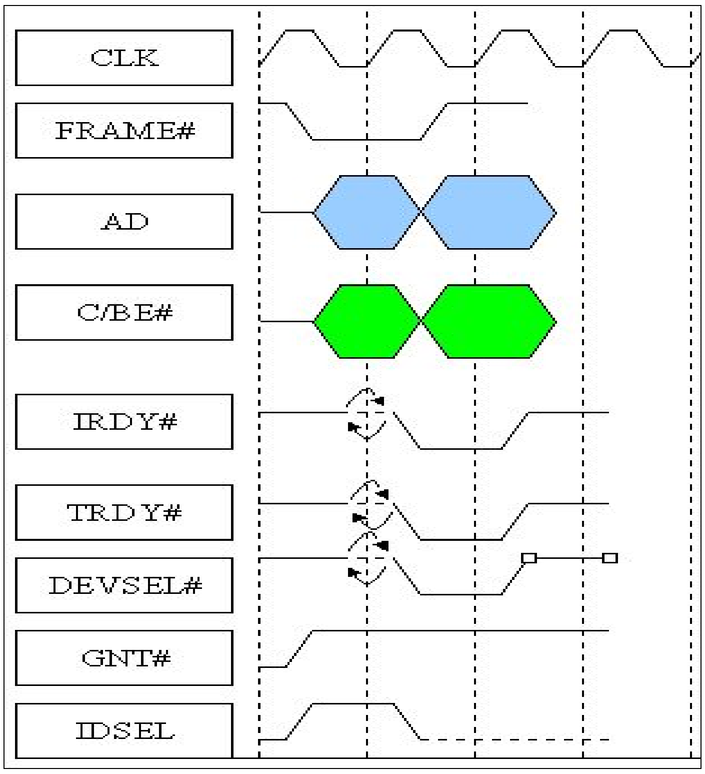


Figure 2-5-3 Type 0 configurations read

Figure 2-5-4 is Type 0 configuration write example, which is described in the address and data phase.

Address phase:

- PCI Initiator writes PCI Target
- FRAME# low, and transaction begins
- AD[1:0]=00b－＞Type 0
  AD[7:2] －＞ Address of configuration dword
  AD[10:8] －＞function numbers
- C/BE#[3:0]==1011b (configuration write)
- PCI Initiator indicates it can transmit data.
- PCI Target indicates it can read data.
- Type 0 needs IDSEL to select device.

Data phase

- PCI Initiator writes PCI Target.
- FRAME# counter drive indicates it is ready to complete the last data transfer phase.
- PCI Initiator transmits configuration data to PCI Target
- Bridge duplicates processor's byte enable to C/BE# [3:0]
- PCI Target receives data in case of IRDY# and TRDY#

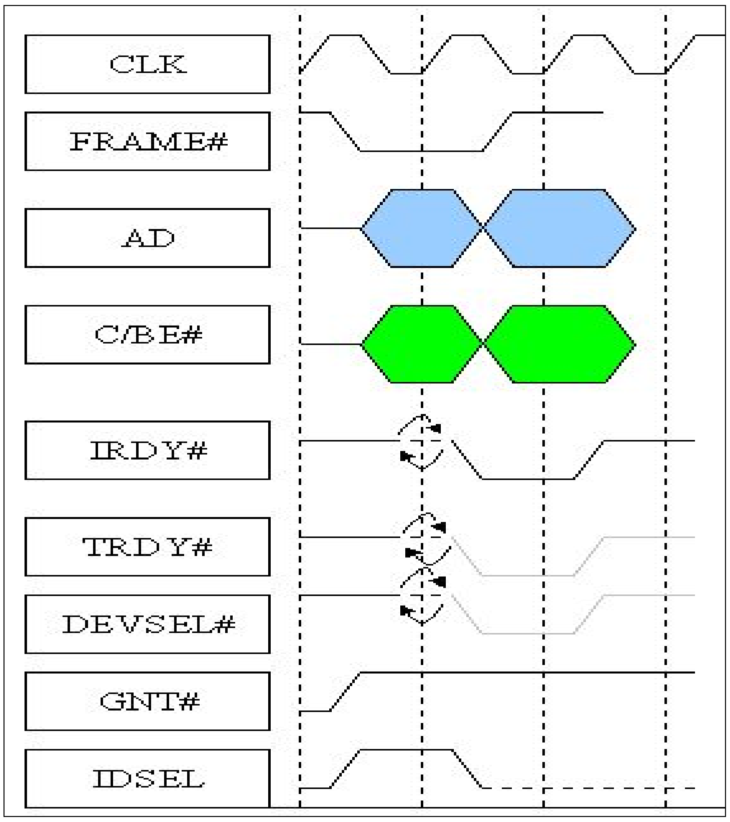


Figure 2-5-4 Type 0 configurations write

As with Type 1, AD [1:0] is 01b is Type 1 configuration transaction, AD [7:2] confirms 64 dword in Target configuration, AD [10:8] confirms 8 functions of Target, AD [15:11] confirms 32 entity devices, selects different device IDSEL lines, AD [23:16] confirms 256 buses in the system, AD [31:24] are reserved and are all 0.

Bus numbers on AD buses are different from sub-bus numbers and are not within the same bus scope, so this configuration must be ignored. If bus numbers are equal to sub-bus numbers, then they can be configuration accessed through Type 0 and be transmitted to sub-bus. If bus numbers are not equal to sub-bus numbers, but sub-bus is within the scope of the bus, Host/PCI make configuration access by means of Type 1. Figure 2-5-5 is Type 1configuration read, which is described in the address and data phase respectively

Address phase

・FRAME# driven low
・AD[1:0]=01b－＞Type 1 configuration transaction
・AD[7:2] －＞ configuration dword address
・AD[10:8] －＞function numbers
・AD[15:11] －＞device numbers
・AD[23:16] －＞bus numbers
・AD[31:24] －＞unused
・C/BE#[3:0]==1010b (configuration access)
・IDESL can be ignored.

Data phase

・FRAME# counter drive indicates the last data transfer phase has been completed.
・PCI Target sends back the requested configuration data.
・Bridge duplicates processor’s byte enable to C/BE#[3:0]
・PCI Initiator device is ready
・PCI Target provides data and announces transactions
・PCI bridge receives data

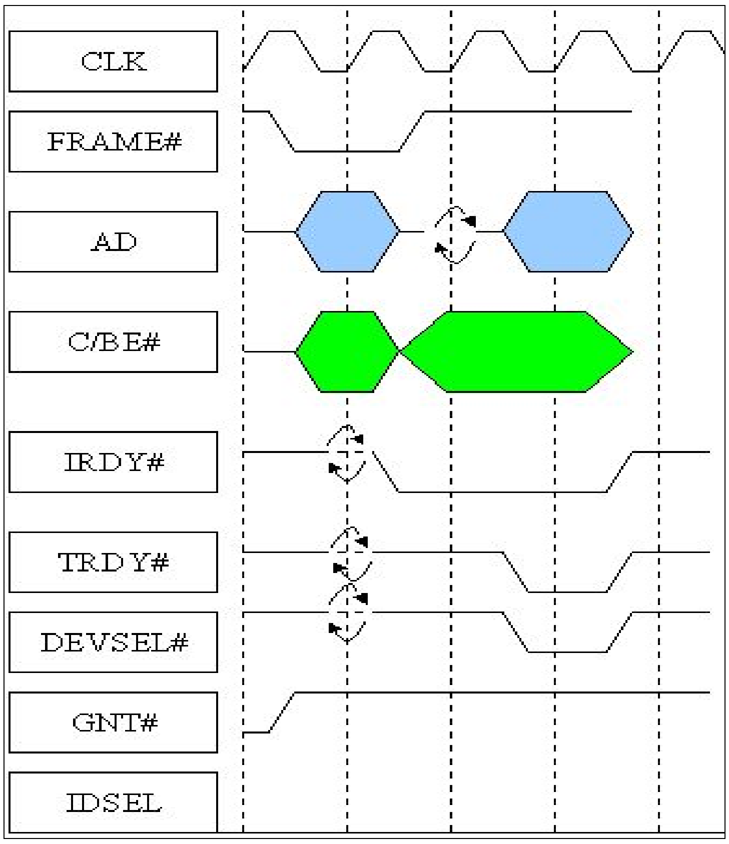


Figure 2-5-5 Type 1 Configuration read

Figure 2-5-6 shows Type 1 configuration writes, which are described in the address and data phases respectively.

Address phase

・FRAME driven low

・AD[1:0]=01b－＞Type 1 configuration transaction

・AD[7:2] －＞ configuration dword address

・AD[10:8] －＞function number

・AD[15:11] －＞device number

・AD[23:16] －＞bus number

・AD[31:24] －＞unused

・C/BE#[3:0]==1011b (configuration writes)

・IDESL can be ignored

Data phase

・FRAME# counter drive indicates that the last data transfer phase is ready

・PCI Target receives bridge’s configuration data

・Bridge duplicates processor’s byte enable to C/BE#[3:0]

・PCI Initiator device is ready

・PCI Target device is ready
・PCI Target receives data

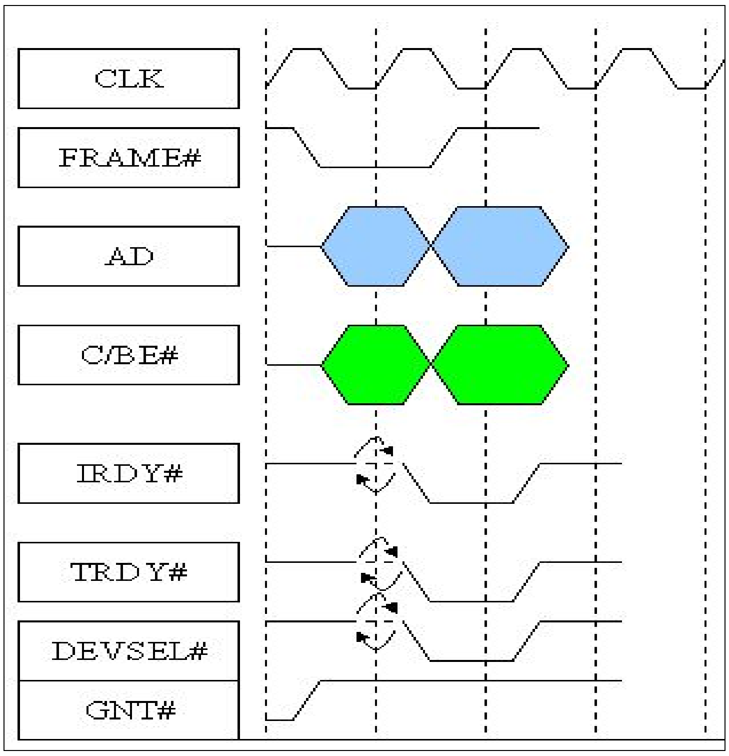


Figure 2-5-6 Type 1 configurations write

2.6 Configuration cache

The first 16 dword of PCI Configuration cache is Configuration cache header, defining the parameters of PCI devices. This header field is divided into 3 types: Type 0, Type 1, Type 2. Type 1 defines PCI-to-PCI Bridge, Type 2 header defines PCI-to-Card Bus Bridge, and Type 0 defines the remaining PCI devices. Generally speaking, Type 0 is more common. Table 2-6-1 shows type 0 configuration cache

Regarding Type 0 configuration header, this chapter only describes more important caches such as Vendor ID, Device ID, Revision ID, Class Code, Sub-System Vendor ID, Sub-System ID / Sub-System Device ID.

Vendor ID is 16 bits cache assigned by PCISIG, with the information of the PCI bendor number inside; Device ID 16 bits cache defined by vendor himself, if the interface card used is PLX 9052 chip, then Vendor ID=10B5, Device ID=9052. Revision ID is 8 bits cache defined by vendor himself, via which programs of different drive versions are defined; Sub-System Vendor ID, (Sub-System ID)/ (Sub-System Device ID) is 16 bits cache defined by the vendor himself respectively, and are connected to bridge Host/PCI, PCI-to-EISA ISA, PCI-to-Micro Channel, PCI-to-PCI, and computer Interrupt Controller. DMA Controller, Programmable

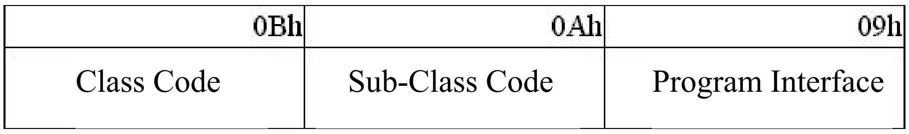
Timer, RTC Controller, need not contain this cache, which mainly identifies the type of sub-system and Sub-System Vendor ID =10B5, Sub-System ID=9052 of the interface car used in this practice

Class code cache consists of three 8 bits caches, as shown in Figure 2-6-1, and the 3 caches are: class code, sub-class code and program interface. This cache is developed and is freely used by the vendor. Table 2-6-2 shows the class code cache definitions. Instruction cache is 16 bits cache, defining basic device response and access ability. Table 2-6-3 is instruction cache definitions. State cache is 16 bits cache, keeps track of functions states aimed at functions defined by PCI device instruction cache, with format definition shown in table 2-6-4. Header type register is 8 bits cache, bit [6:0] is header type, bit [7] defining device is a single/multi-function device. If [7]= 0 is single function device, [7]= 1 is a multi-function device.

In addition to the above mentioned more important Type 0 configuration caches, there are several other caches such as Cache Line Size cache, Latency Timer cache, Built-In-Self-Test cache, Base Address cache, Extension ROM base address cache, Card Bus CIS, Interrupt Pin cache, Interrupt Line cache, Min_Gnt: time block request, Max_Lat: priority level request, ability index cache, etc. For these types of caches and Type 1 configuration cache, the reader can refer to the book--PCI SYSTEM ARCHITECTURE.

Table 2-6-1 PCI configuration cache

00h	02h	04h	06h	08h	09h	0Ch	0Dh	0Eh	0Fh
Vendor ID	Device ID	Instruction Cache	Status Cache	Revision ID	Class Code	Cache Line Size	Latency Timer	Header Type	BIST
10h	14h	18h	1Ch						
Base address 0	Base address 1	Base address 2	Base address 3						
20h	24h	28h	2Ch	2Eh					
Base address 4	Base address 5	Card Bus/CIS	Sub System Vendor ID	Sub System ID					
30h	34h	35h	3Ch	3Dh	3Eh	3Fh			
Extension ROM base address	Ability index	Reserved	Interrupt line	Interrupt line	Min GNT	Max LAT			



0Bh	0Ah	09h
Class Code	Sub-Class Code	Program Interface

Figure 2-6-1 class caches

Table 2-6-2 class code cache

Basic Class Code	Device description	Sub-Class code	Program interface	Device description
00h	Device built before class code definition	00h	00h	All non-VGA device
		01h	01h	Device compatible with VGA
01h	Mass storage media controller	00h	00h	SCSI Controller
		01h	××h	IDE Controller
		02h	00h	FDD Controller
		03h	00h	IPI Controller
		04h	00h	RAID Controller
		80h	00h	The other storage peripherals
02h	Network controller	00h	00h	Ethernet Controller
		01h	00h	Token Ring Controller
		02h	00h	FDDI Controller
		03h	00h	ATM Controller
		04h	00h	ISDN Controller
		80h	00h	The other network peripherals
03h	Display controller	00h	00h	Device compatible with VGA
			01h	Device compatible with 8514
		01h	00h	XGA Controller
		02h	00h	3D Controller
		80h	00h	The other display peripherals
04h	Multi-media device	00h	00h	Video device
		01h	00h	Audio device
		02h	00h	Computer telephone devices
		80h	00h	The other multi-media devices
05h	Memory controller	00h	00h	RAM Controller
		01h	00h	Flash Controller
		80h	00h	The other memory controllers
06h	Bridge device	00h	00h	Host/PCI
		01h	00h	PCI/ISA
		02h	00h	PCI/EISA
		03h	00h	PCI/Micro Channel

		04h	00h	PCI/PCI
			01h	Subtraction decoding PCI/PCI
		05h	00h	PCI/PCMCIA
		06h	00h	PCI/ N u Bus
		07h	00h	PCI/Card Bus
		08h	××h	Race Way
		80h	00h	The other bridge device
07h	Simple communication controller	00h	00h	XT compatible serial controller
			01h	16450 compatible serial controller
			02h	16550 compatible serial controller
			03h	16650 compatible serial controller
			04h	16750 compatible serial controller
			05h	16850 compatible serial controller
			06h	16950 compatible serial controller
		01h	00h	Parallel port
			01h	Bi-directional parallel port
			02h	Following ECP1.×parallel port
			03h	IEEE128 controller
			FE h	IEEE1284 Target
		02h	00h	Multi-port serial controller
		03h	00h	General purpose modem
			01h	Interface compatible with 16450
			02h	Interface compatible with 16550
			03h	Interface compatible with 16650
			04h	Interface compatible with 16750
		80h	00h	The other communication device
08h	Basic system peripheral device	00h	00h	8259 interrupt controller
			01h	ISA PIC
			02h	EISA PIC
			10h	IO APIC
			20h	IO APIC interrupt controller

		01h	00h	8237 DMA controller
			01h	ISA DMA Controller
			02h	EISA DMA Controller
		02h	00h	8254 timer
			01h	ISA system timer
			02h	EISA system timer
		03h	00h	RTC controller
			01h	ISA RTC Controller
		04h	00h	PCI hot-plug controller
		80h	00h	The other system peripheral devices
09h	Input device	00h	00h	Keyboard controller
		01h	00h	Digitizer (PEN)
		02h	00h	Mouse controller
		03h	00h	Scanner controller
		04h	00h	Common Game Port controller
			10h	Game Port controller
		80h	00h	The other input peripheral device
0Ah	Docking Station	00h	00h	General docking station system
		80h	00h	The other docking station system
0Bh	Processor	00h	00h	386
		01h	00h	486
		02h	00h	Pentium
		10h	00h	Alpha
		20h	00h	PowerPC
		30h	00h	MIPS
		80h	00h	Co-Processor
0Ch	Serial bus controller	00h	00h	Fire wire (IEEE1304)
			10h	Open HCI IEEE1394
		01h	00h	ACCESS bus
		02h	00h	SSA serial storage architecture
		03h	00h	UHC-USB
			10h	OHC-USB
			80h	USB without particular program interface
			FE h	USB device

		04h	00h	Fiber Channel
		05h	00h	SM-Bus
0Dh	Wireless controller	00h	00h	IRDA compatible controller
		01h	00h	Consumer IR controller
		10h	00h	RF controller
		80h	00h	Other wireless controller
0Eh	Smart IO Controller	00h	××h	Following I2O controller
			00h	Signal FIFO of address
0Fh	Satellite communication controller	01h	00h	(TV)
		02h	00h	(Audio)
		03h	00h	(Voice)
		04h	00h	(Data)
10h	Encryption/deciphering Controller	00h	00h	Applied to network and operation
		01h	00h	Used in entertainment
		80h	00h	Used for other functions
11h	Data acquisition and signal processing controller	00h	00h	DPIO module
		80h	00h	Other controllers of this kind
12h~FE h	Reserved			
FF h	Device not in conformity with the defined class code			

Table 2-6-3 instruction registers

Bit	Description	Preset value
15	Reserved	0
14		
13		
12		
11		
10		
09	Fast Back-to-Back enable	0
08	SERR#enable	0
07	Gradual drive control	1 when used, 0 when unused
06	Parity error response	0

05	VGA color table monitoring	NON-VGA is1, VGA is 0
04	Memory Write and Invalidate enable	0
03	Special cycle	0
02	Bus Master	0
01	Memory space	0
00	IO space	0

Table 2-6-4 Status register

Bit	read (R) /write (W)	Function	Preset value
15	R/W	Parity error detected	1: Parity error detected
14	R/W	System error signal has been sent out	1: The device may generate SERR# signal
13	R/W	Master Abort received	Receive Target Abort signal
12	R/W	Target Abort received	Receive Master Abort signal
11	R/W	Target Abort signal has been sent out	1:error occurs
10	R	DEVSEL timing	00b:fast 11b: reserved
09			
08	R/W	Master data parity error	1:error occurs
07	R	Fast Back-to-Back ability	1:supports this function, 0:does not support
06	R	UDF support	1:supports UDF, 0:does not support
05	R	66MHz ability	1:Capable of 66MHz operation, 0:33MHz
04	R	Ability serial	1:with ability index cache0: without this function
03	R	Reserve	0
02			
01			
00			

2.7 Read transmission

As with PCI read transfer, take PCI Initiator reading PCI Target as an example, single data read and burst read are described respectively. Figure 2-7-1 shows single read transfer; Figure 2-7-2 shows burst read transfer.

Single read transfer, with actions of each described as follows:

Clock 1 When it is detected that bus is in idle (both FRAME# and IRDY#are counter-driven high), initiator begins to conduct transaction on the rising edge of clock 1. Initiator drives address on AD[31:0], drives instructions on C/BE#[3:0], and drive FRAME# low, indicating the transaction begins and there is an effective address and instruction on bus.

Clock 2 Clock rising edge, targets on the bus are sampled address, instructions and FRAME#, address phase are completed. All targets begin to decipher, deciding the target of this transaction. Initiator drives IRDY# low, indicating it is ready to receive the first batch of data read from target. While initiator drives IRDY# low, it back drive FRAME# high, indicating it is ready to complete the last data phase of this transaction. Initiator stops driving instructions to C/BE#[3:0] and starts to drive byte enable so as to indicate the location of the first dword to be read. There is not any target driving DEVSEL# low and announcing the transaction.

Clock 3 Clock rising edge, initiator is sampled so that DEVSEL# is backdriven high, indicating that the transaction has not need announced by target. So, the data phase is not completed, it was prolonged by a clock (a wait state). In wait state, initiator must continue to drive byte enable and drive IRDY# until target drives DEVSEL# low so as to announce the transaction. Meanwhile, target drives TRDY# low, indicating that it is driving the first dword onto AD bus.

Clock 4 Rising edge, initiator and target are both sampled and IRDY# & TRDY# are driven low. At the same time, initiator latches data and the setting of TRDY#, indicating the data is valid. The first (also the only one) data is read successfully. When target is sampled and FRAME# is backdriven high, indicating this is the last data phase. The transaction is completed, so initiator drives IRDY# high and stops driving byte enable, target backdrives TRDY# and DEVSEL# high and stops driving data.

Clock 5 Clock rising edge, bus returns to idle state.

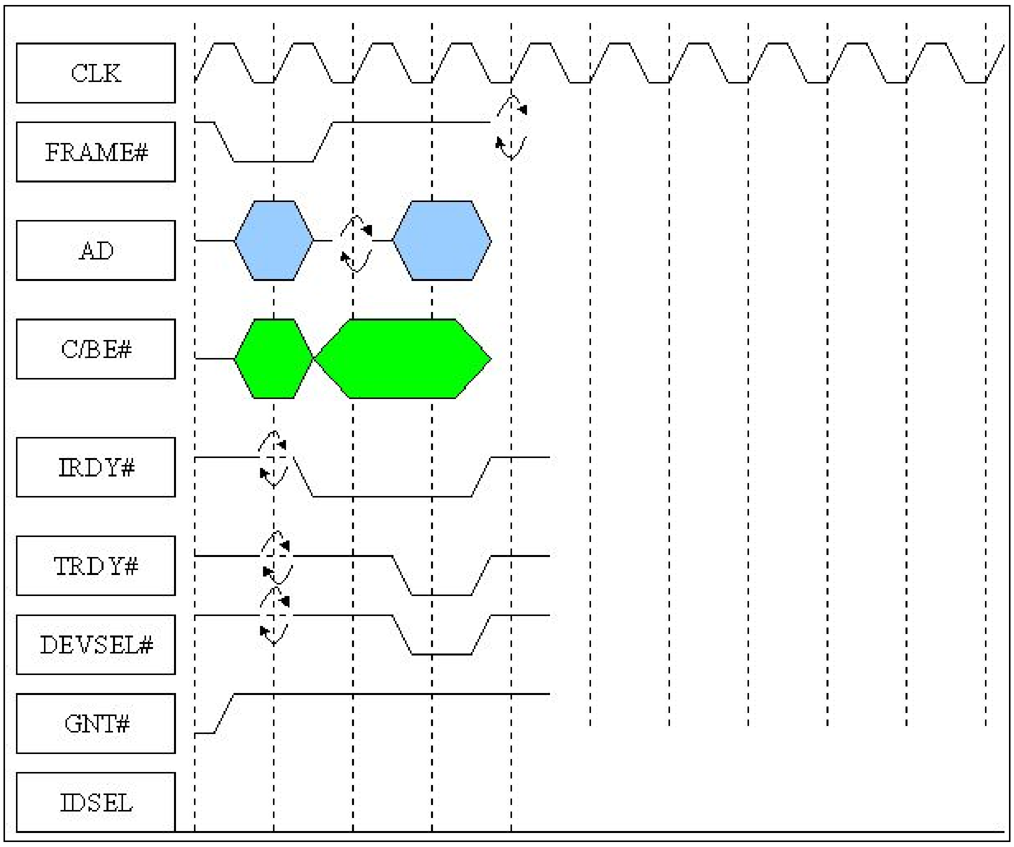


Figure 2-7-1 single read transfer

As with Burst read transfers, the actions of each clock are described as follows:

Clock 1 Initiator drives FRAME# low, indicating that transaction begins, and there is a valid starting address and instruction. FRAME# must continue to be driven low until initiator is prepared to complete the last data phase. When initiator is driving FRAME# low, it will drive the starting address on AD bus and drive transaction types on instruction/byte enable line. Address and transaction type are driven onto the bus during clock period 1. During the clock period 1, IRDY#, TRDY# and DEVSEL may not be driven (prepared to be taken over by new target). It is maintained to logic high by pull-up resistor on the system board (requiring system board resources).

Clock 2 Initiator stops driving AD bus. On all signals that may be driven by more than 1 PCI agent, a reverse cycle (namely dead cycle) is needed. This cycle is intended to avoid that when one agent is closing its output driver, another agent begins to drive the same signal (resulting in data collision), target will acquire the control of AD bus so as to drive the first requested data item (one to four bytes), transmitting it to initiator. During the process of reading, clock 2 is defined as reverse cycle, because the ownership of AD bus is converted into the addressed target by initiator. Target is responsible for continuously backdriving TRDY# high to accomplish this cycle. At the same time, initiator stops driving instructions onto instruction/byte enable lines, and uses them to indicate the bytes to be transferred (and the data path used during the process of data transfer) in the dword that has been addressed so far. Usually, during the process of reading, initiator may drive all byte enable. Meanwhile, initiator drives IRDY# low, indicating it is prepared to receive the first data item from the target. While driving IRDY#, initiator does not backdrive FRAME# high, therefore indicating that this is the last data phase of the transaction. During the last data phase, it is required to drive IRDY# low and counter drive FRAME# high at the same time, indicating it is prepared to complete the last data phase.

Clock 3 Target drives DEVSEL# low, indicating its recognizes its address and takes part in this transaction. Meanwhile, target begins to drive the first data item (one to four bytes, as requested by the setting of C/BE lines) onto AD bus and drives TRDY# low to indicate the emerge of the request data.

Clock 4 When initiator and the currently addressed target are on the rising edges of clock four, they both are sampled and TRDY# & IRDY# are driven low, initiator will read the first data item from the bus. The first data phase consists of clock 2 and the wait state of target insertion (clock 3, a reverse cycle). At the beginning of the second data phase (clock edge 4), initiator sets byte enable, indicating the byte to be transmitted in the first dword. The rule is when a data phase is in progress, initiator must immediately output byte enabling necessary to the data phase. If initiator does not know what will the byte enable of the next data phase will be set to, it may maintain the backdriving state of IRDY#, preventing the current data phase from ending. After entering into the second data phase, initiator continues to drive IRDY# low rather than backdrive FRAME# high, indicating that initiator is prepared to read the second data item. In multi-data transactions, target (if it supports burst) is responsible for latching the starting address-to-address counter and managing the address counters of each data phase.

Clock 5 Initiator is sampled and TRDY# is backdriven high, confirming target is requesting more time so as to input the second data item, so it inserts a wait state into the second data phase (clock 5). In wait state, target begins to drive the second date item onto the bus and drive TRDY# low, indicating the emerge of data item.

Clock 6 When initiator is sampled and both IRDY# and TRDY# are driven low, it reads the second data item from the bus. This is the end of the second data phase, which consists of clock cycle 4 and 5. Initiator sets byte enable, indicating the bytes to be inputted in the next dword. Meanwhile, it backdrives IRDY# high, indicating that it needs to exceed the time of a clock cycle before getting ready to receive data. Target continues to drive TRDY# low, indicating it will immediately drive the third requested data item onto AD bus.

Clock 7 Target must continue to drive the third data item onto AD bus, initiator drives IRDY# low, indicating it wants to receive the third data item at the rising edge of the next clock. Meanwhile, it backdrives FRAME# high, indicating this is the last data phase.

Clock 8 When it is sampled that both IRDY# and TRDY# are driven low, initiator reads the third data item from the bus. The third data phase consists of clock 6 and 7. On the rising edge of clock 8, sampling to FRAME# is backdriven high, instructing target this is the last data phase and the burst transfer made up of the 3 data phases is fully completed. Initiator backdrives IRDY# high, allowing the bus to return to idle state (on the rising edge of clock 9), while target backdrives TRDY# and DEVSEL# high.

Clock 9 The bus returns to idle state (both TRDY# and DEVSEL# are driven high)

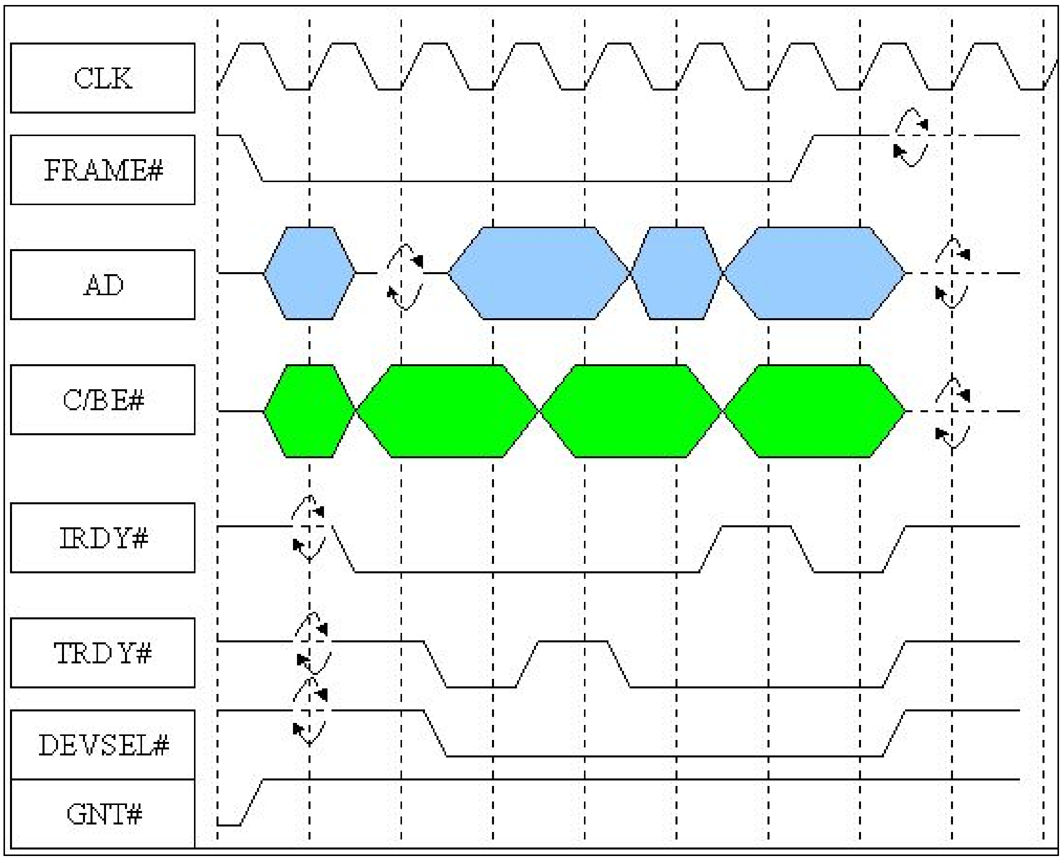


Figure 2-7-2 Burst read transfer

2.8 Write transfer

AS with PCI device write transfer, take PCI initiator writing PCI target as example, single data write and burst write are described respectively. Figure 2-8-1 is single write transfer, while Figure 2-8-2 is burst write transfer.

As with single write transfer, the actions of each clock signal are as follows:

Clock 1. When it is detected that the bus is idle (both TRDY# and DEVSEL# are backdriven to high), initiator begins to conduct the transaction on clock rising edge. Initiator drives the address on AD[31:0] , drives instructions on C/BE#[3:0], and drives FRAME# low, indicating that the transaction begins and there is a valid address and instruction on the bus.

Clock 2. Clock rising edge, all the targets on the bus are sampled and address, instruction, and FRAME#, the data phase are completed, all the targets start to decipher to decide the target of this transaction, initiator drives IRDY# low, indicating its is driving the first written data item onto the AD bus, as long as the time that initiator drives IRDY# low is within the 7 clock cycles after entering data phase.

When initiator drives IRDY# low, it also cunterdrives FRAME# high, thus indicating it is prepared to complete the last data phase of this transaction. Initiator stops driving instructions onto C/BE#[3:0] and begins to drive byte enable, indicating it wants to be written into the first dword. During clock 2, target drives DEVSEL# low to announce the transaction; meanwhile, target drives TRDY# low, indicating it is prepared to receive the first written data item.

Clock 3. Master is sampled and DEVSEL# is backdriven low, indicating that the target has announced the transaction. Target is sampled to IRDY# and data on AD bus. The setting of IRDY# also shows that it has latched the first valid written data item. The setting of IRDY# instructs initiator and target are prepared to receive, meanwhile, target is also sampled so that FRAME# is backdriven high, indicating that this is the final data phase. The transaction is completed, initiator is backdriven by IRDY# high and stops driving byte enable, target backdrives TRDY#and DEVSEL# high and stops driving data.

Clock 4. Clock rising edge, the bus returns to idle state

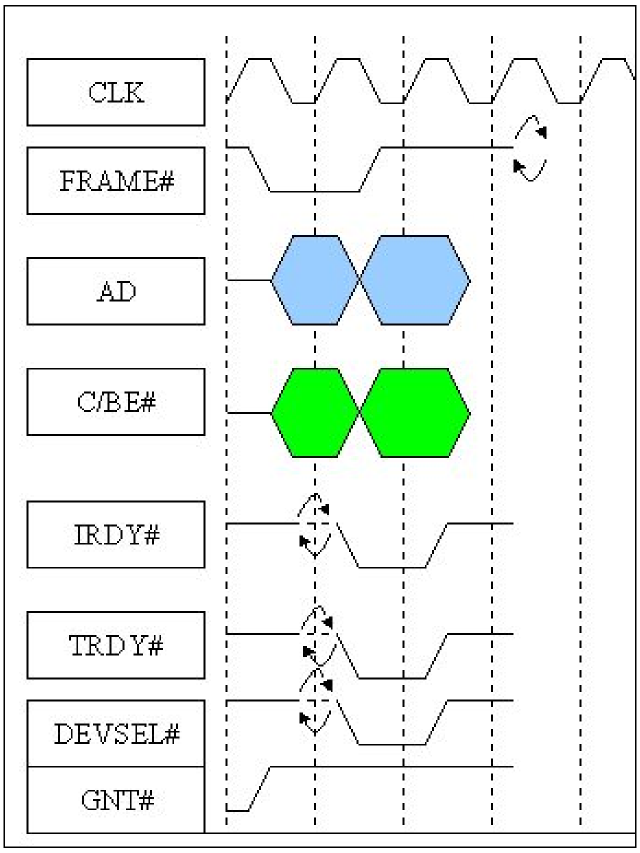


Figure 2-8-1 Single write transfer

Burst writes transfer: the actions of each clock signal are described as follows:

Clock 1. FRAME# and IRDY# are backdriven high (on the rising edge of clock one), the bus remains in idle state. Initiator drives FRAME# low, indicating the transaction begins and there is a valid starting address and instruction on the bus, FRAME# must continue to be driven low until initiator is ready (already driven IRDY# low) to complete the last data phase. While initiator drives IRDY# low, it drives the starting address on AD bus and drives transaction types to instruction/byte enable line- C/BE#[3:0]. Address and transaction types are driven onto the bus during clock period one. IRDY#, DY# and DEVSEL# will not be driven (prepared to be taken over by new initiator and target). And it maintains high with the pull-up resistor on the system board.

Clock 2. Initiator stops driving AD bus and begins to drive the first written data item and may immediately starts to drive the first data item onto AD bus. Initiator stops driving instructions and begins to drive byte enable to indicate the byte that has been written into target in the currently addressed dword. Initiator drives written data onto AD bus and drives IRDY# low, indicating that data has emerged in the bus. While initiator drives IRDY# low, it does not backdrive FRAME# high (because this is not the last data phase). Target deciphers address and instructions and drives DEVSEL# low, announcing the transaction. At the same time, it drives TRDY# low, indicating that it is prepared to receive the first written data item.

Clock 3. When initiator and the currently addressed target are on the rising edges of target and are sampled and TRDY# and TRDY# are driven low, indicating that it is ready to complete the first data phase. This is a transfer (single clock data phase) of zero wait state. Target receives the first data item from the bus (and sampling byte enable t decide which bytes to be written), completes the first data phase, and adds 4 to address counter to point to the next dword. Initiator drive the second data onto AD bus and sets byte enable, indicating the byte it will write in the next dword during the second data phase and the data path used, also continues to drive IRDY# low, and it will not backdrive FRAME# high, therefore indicating that initiator is prepared to read the second data phase rather than the last data phase. The setting of IRDY# indicates that the data to be written appears on the bus.

Clock 4. Initiator and the currently addressed target are sampled and TRDY# and TRDY# are driven low, indicating they are both prepared to complete the second data phase, which is a data phase of zero wait state. Target receives the second data item from the bus (and sampling byte enable to decide which data paths are valid data), completing the second data phase. Before initiator begins to drive the next data item, it needs some time (because it encounters empty cache). So it backdrives IRDY# high to insert a wait state into the third data phase, immediately setting the proper byte enable necessary to the third data phase. Before target is prepared to receive the third data item, it also needs more time, to indicate more demands for time, target backdrives TRDY# high during the clock period. Target again adds 4 to address counter to point to the next dword, although initiator has not the third data item that can be driven, it must drive a stable signal onto the data path to prevent AD bus from floating.

Clock 5. When initiator and target are sampled and IRDY# and TRDY# are driven high, they insert a wait state (clock cycle 5) into the third data phase. Initiator drives IRDY# low and drives the last data item onto AD bus.

Meanwhile, it drives FRAME# high, indicating this is the final data phase, and continues to drive the byte enable necessary to the third data phase until it is completed. Target continues to backdrive TRDY# high, indicating it is not prepared to receive the third data item.

Clock 6 Initiator is sampled and IRDY# is driven low, indicating it is transmitting data, but TRDY# is still high (backdrive state) (because target is not prepared to receive the third data item). Target is also sampled and FRAME# is backdriven to high, indicating that the final data phase is in progress, continuing to backdrive TRDY# high, until it is prepared to receive the final data item. TRDY# is backdriven to high sampling, target and initiator insert the second wait state into the third data phase. During the second waits state, initiator continues to backdrive TRDY# high, indicating it is not prepared to receive the third data item.

Clock 7. Target and initiator are sampled and IRDY# is driven low, indicating initiator is still transmitting data, but TRDY# is still high. In response, target and initiator inset the third wait state into the third data phase, during which, initiator continues to drive the third data item onto AD bus and maintain the setting of byte enable. Now, target drives TRDY# low, indicating it is prepared to complete the last data phase.

Clock 8. Initiator and target are sampled and IRDY# and TRDY# are driven low, indicating that they are prepared to end the third, also the final data phase. In response, the third data phase is completed on rising edge of clock 8. Target receives the third data item on AD bus. The third data phase consists of 4 clock cycles (the first clock cycle of data phase, clock cycle 4, and the 3 wait states). Initiator stops driving data onto AD bus, stops driving C/BE# bus, and back drives IRDY# high (enabling bus to return to idle state). Target back drives TRDY# and DEVSEL# high.

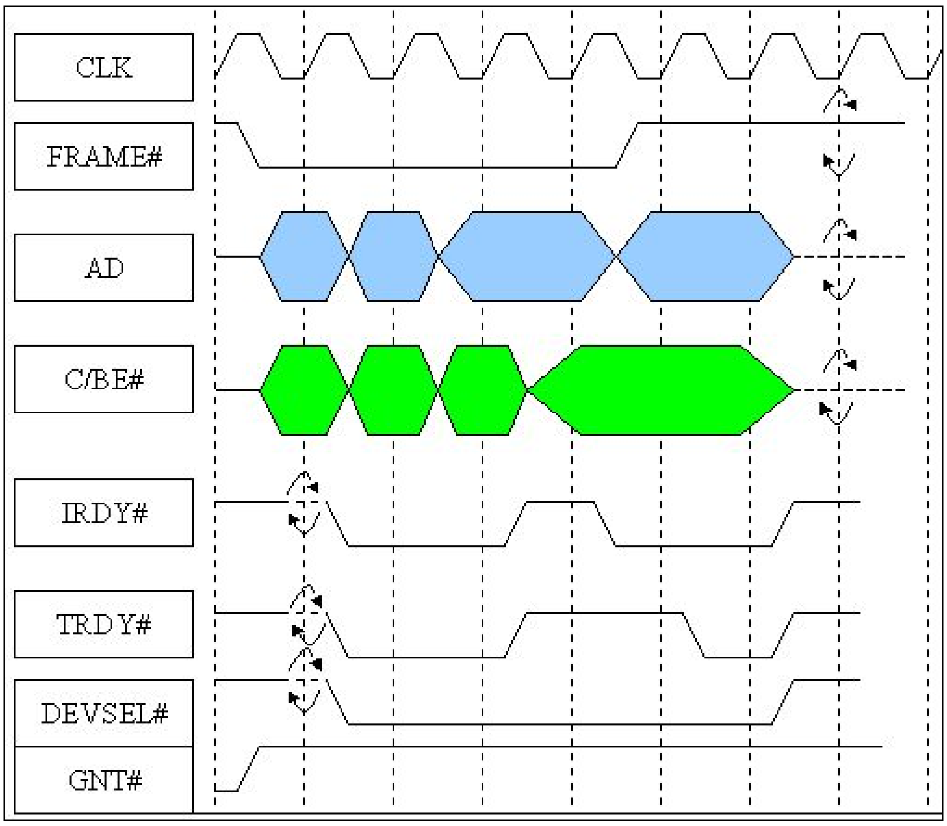


Figure 2-8-2 Burst writes transfer

This chapter has described the hardware architecture of PCI interface, pin definition and configuration cache, etc. Usually, PCI interface configuration cache can write C/C++ program languages to read and write. However, changing configuration cache may cause PCI interface card unable to operate. When it comes to this circumstance of incomplete data, it is better not to change the data in configuration cache without authorization. Regarding the implementation of PCI interface, it will be described in the next chapter with PLX PCI_9052 chips and actual examples of practice kits made of it.

Exercises

1. Give examples to describe PCI_Master, PCI_Slave, and PCI_Initiator, PCI_Target device.
2. To understand the definition of cache by reading PCI configuration cache, please have a try to describe with actual computer device.
3. Have a try to describe the relevant data that will replace PCI interface in the future.
4. About the relevant applications of PCI interface, give practical examples to describe its functions.

5. Compare the differences between ISA interface and PCI interface.
6. Give examples to describe the advantages and disadvantages PCI interface.

Article 2 PCI-IO/LAB hardware description

This article is divided into chapter 3 and 4, which describe hardware lines of PCI-IO/CAB and the important specifications of chips used. For PLX9050/9052 chips please refer to chip specification documents of PLX Company. The specifications of other parts can be found in the disk attached with this manual.

Chapter 3 PCI-IO/LAB hardware

The hardware of PCI-IO/CAB is designed and manufactured by Leaper for the electronic circuits of this exercise equipment. This manual will discuss the products functions in two chapters to help the reader to understand the basic concepts of this device.

3.1 PCI-IO hardware

Shown in Figure 3-1-1 is the picture of PCI-IO interface card, with PLX-9050/9052 chip in the middle of this figure as the core of this interface card; PLX-9050/9052 chip is briefly described in next chapter. To use this chip, a serial EEPROM as cache is needed, storing characteristics related with this interface card. Figure 3-1-2 shows the pin of this interface PCI slot; Figure 3-1-3 shows the producing circuit of 8254 chips and related clock. Figure 3-1-4 shows the decoupling capacitor, mainly eliminating the noise of interface card.

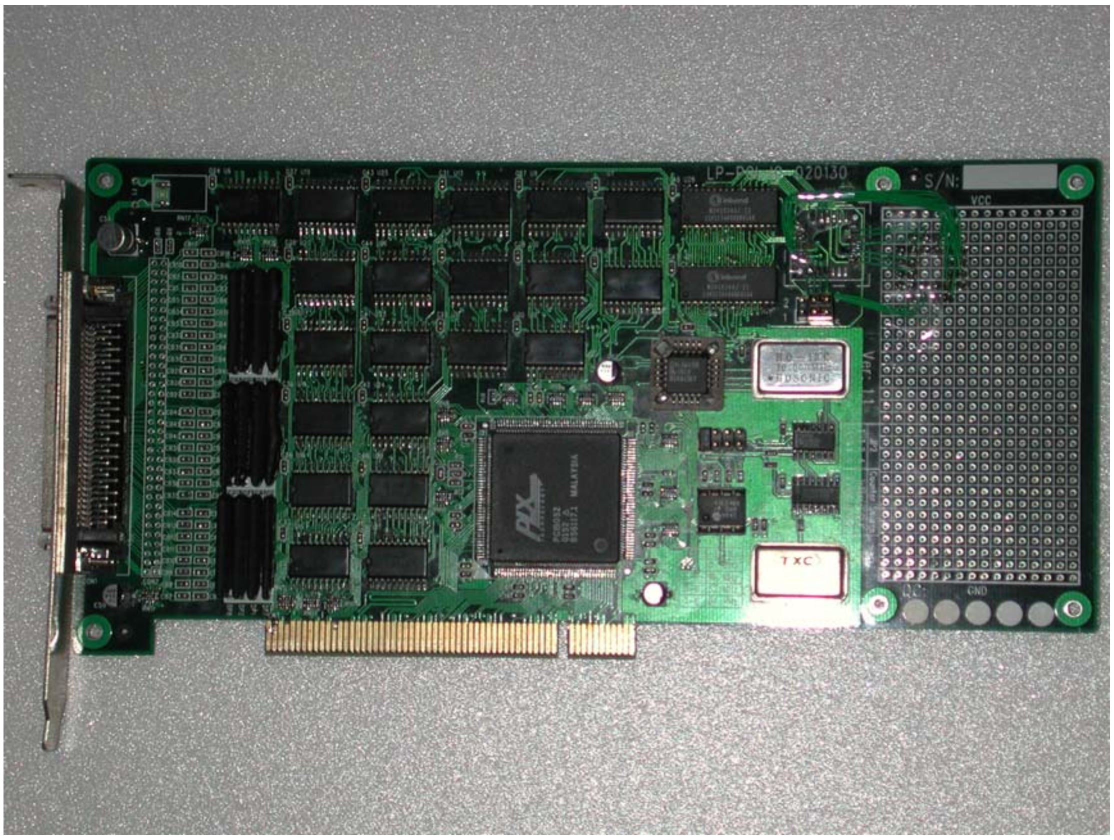


Figure 3-1-1 PCI-IO interface card (engineering version)

Figure 3-1-5 shows part of the decoupling capacitance and chip mode setting. Figure 3-16 shows Chip LOCAL end interrupted pin, PCI-IC is set to operate under Non-Multiplexed and the LOCAL end does not produce interrupted input, only uses the interface card as IO output/input. Figure 3-1-7 shows the serial EEPROM of PLX-9050/9052, storing settings related with this interface card. Figure 3-1-8 is PLX-9050/9052 connection diagram. Figure 3-1-9 is resistance network, mainly connecting PLX-9052 chips and allowing this interface card to have IO function in conjunction with related setting. Figure 3-1-10 is also a decoupling capacitance. Plenty of decoupling capacitance is needed on the PCI interface card to mitigate electricity and noise interference. Figure 3-1-11 is an output clock frequency selection, 4 different kinds of LOCAL end operating clocks can be selected. Figure 3-1-12 and 3-1-13 show RAM and IO port input or output selection, common output may be unnecessary to be set. However, it must be set when used in input status. Figure 3-1-14, 3-1-15, 3-1-16, 3-1-17, 3-1-18, 3-1-19 shows the circuit diagrams of IO respectively, from the diagram we can learn that output and input signals switch channels here. Figure 3-1-20 is output /input end protection loop. Figure 3-1-21 shows output joint, output end on the interface card respectively and another output end on the baffle, which is connected to PCI-LAB with a 68-pin connecting wire, while 3-1-22 is IO-BANK selection circuit.

There is an IO-BANK0 inside this interface card, while IO-BANK1~IO-BANK4 may also be in it. Normally, implementation can use internal and external IO-BANK, however, internal IO-BANK mechanism has been fixed to use 8254 chips. The read himself can also implement 8254-related practice in IO-BANK1~IO-BANK4.

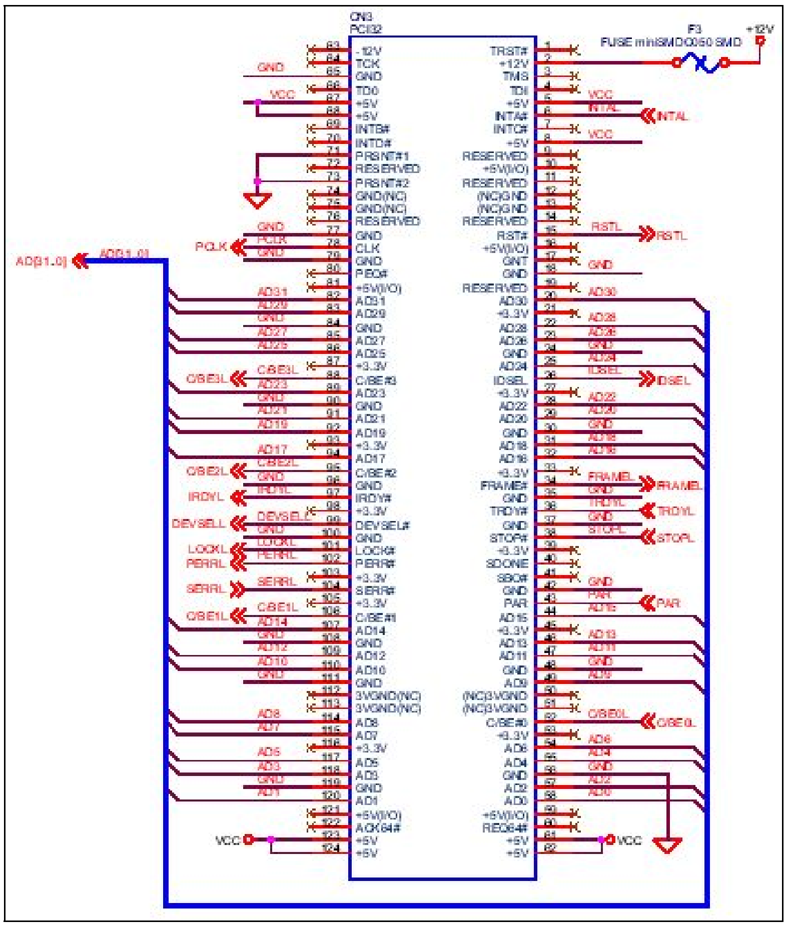
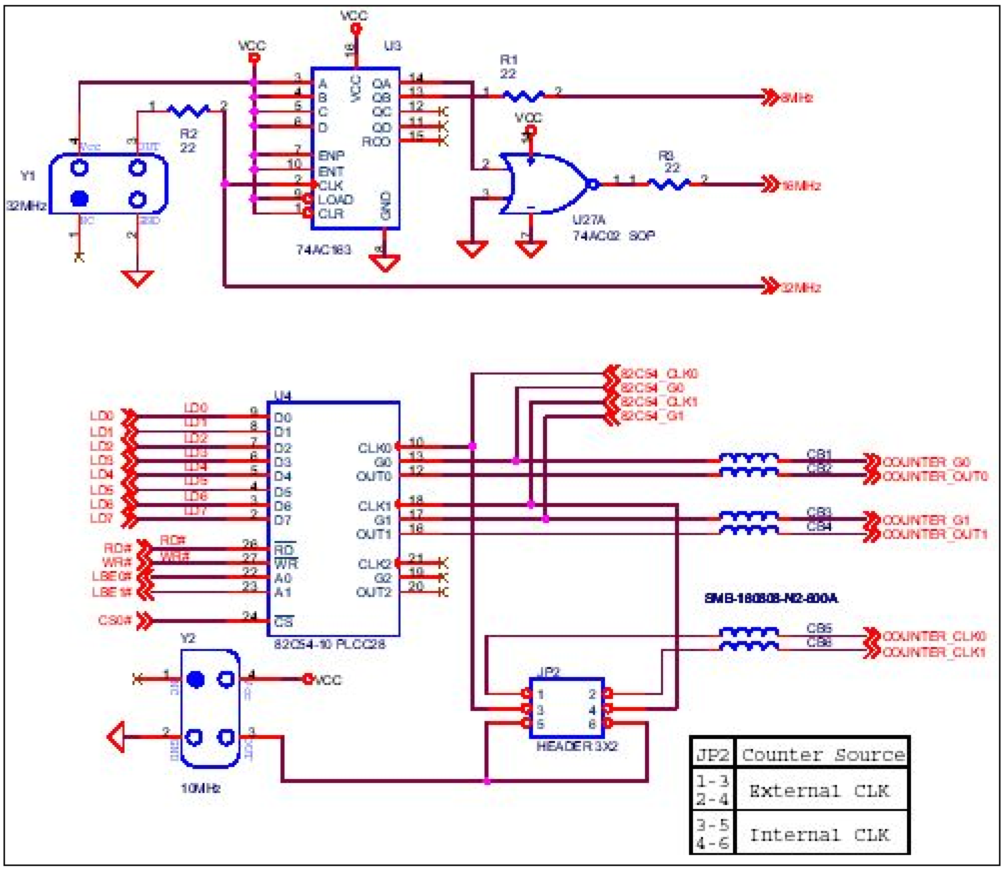


Figure 3-1-2 PCI-IO card PCI slot



JP2	Counter Source
1-3 2-4	External CLK
3-5 4-6	Internal CLK

Figure 3-1-3 Built-in 8254 and clock circuit

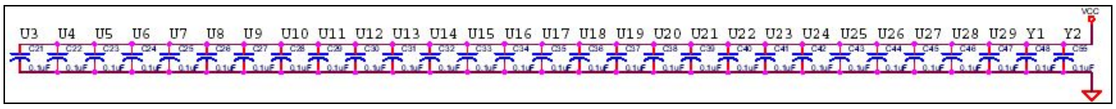

U3 U4 U5 U6 U7 U8 U9 U10 U11 U12 U13 U14 U15 U16 U17 U18 U19 U20 U21 U22 U23 U24 U25 U26 U27 U28 U29 Y1 Y2
VCC
0.1uF

Figure 3-1-4 Decoupling capacitance

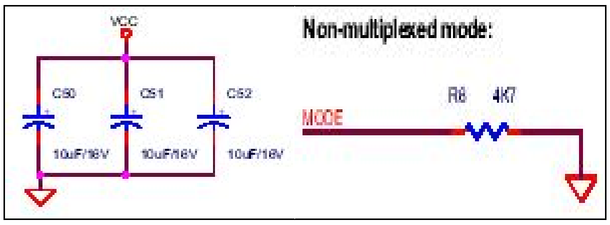


Figure 3-1-5 Decoupling capacitance and chip mode setting

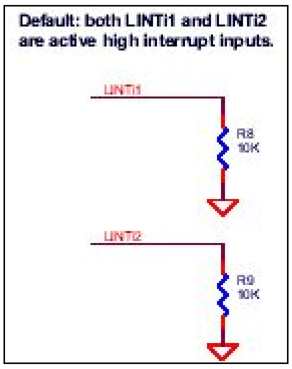

Default: both LINTi1 and LINTi2 are active high interrupt inputs.
LINTi1
R8
10K
LINTi2
R9
10K

Figure 3-1-6 LOCAL end interruption pin

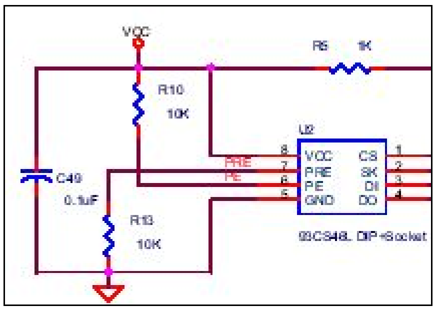


Figure 3-1-7 93C46 serial EEPROM

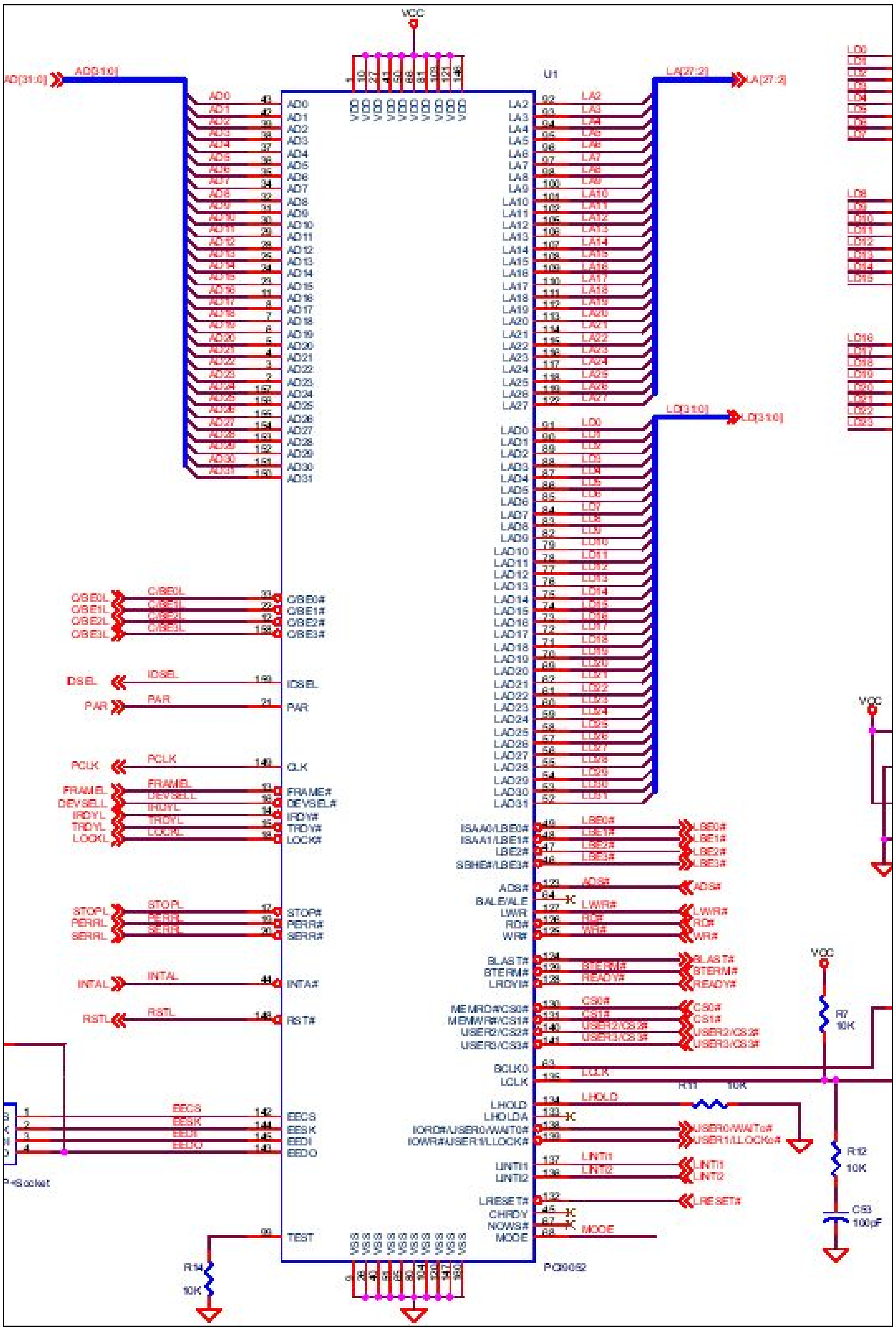


Figure 3-1-8 PLX-9052 connection diagrams

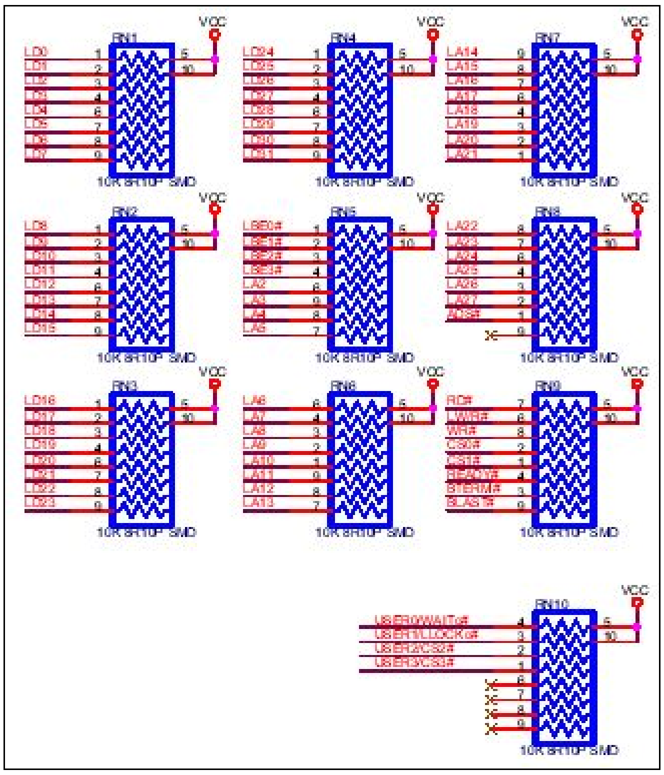


Figure 3-1-9 Resistance network

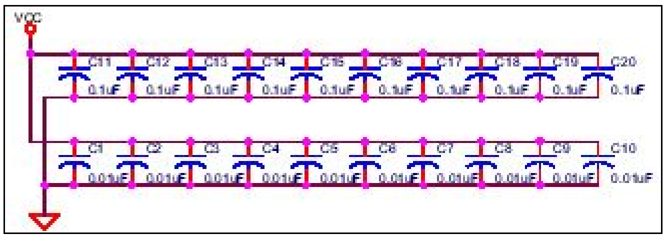
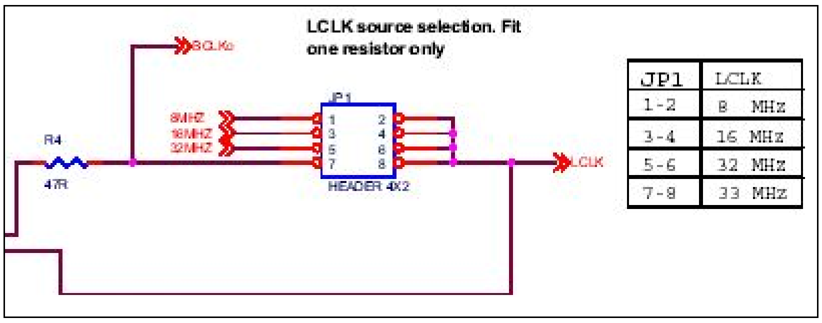


Figure 3-1-10 Decupling capacitance



JP1	LCLK
1-2	8 MHz
3-4	16 MHz
5-6	32 MHz
7-8	33 MHz

Figure 3-1-11 Clock selections

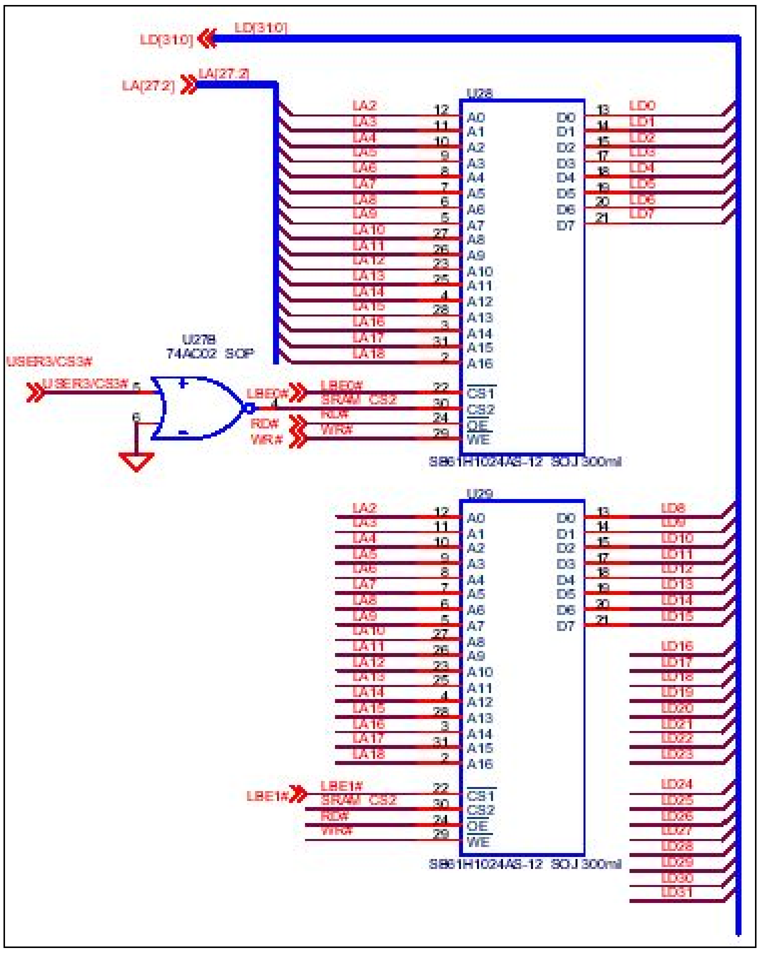


Figure 3-1-12 RAM control port (only engineering version has RAM)

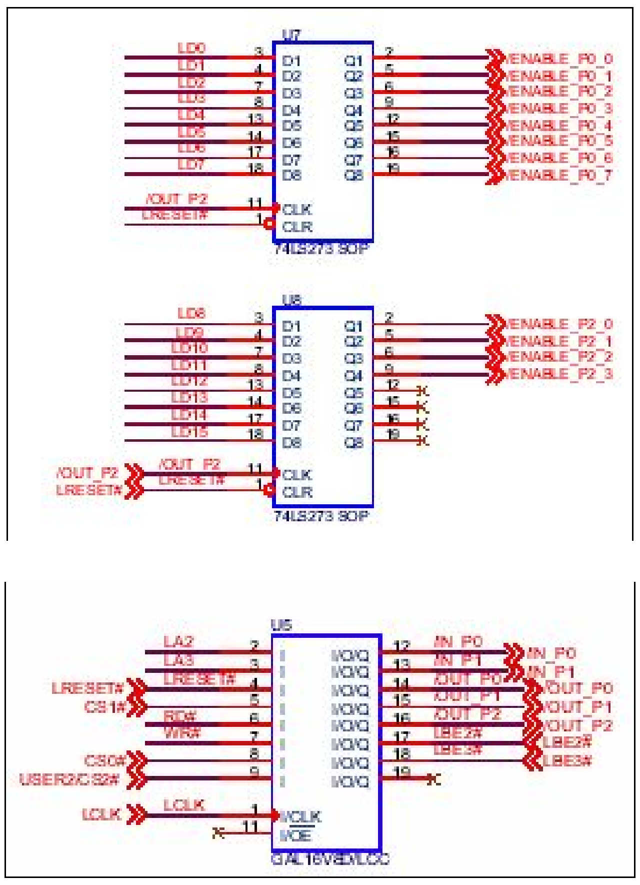


Figure 3-1-13 Output/input selections

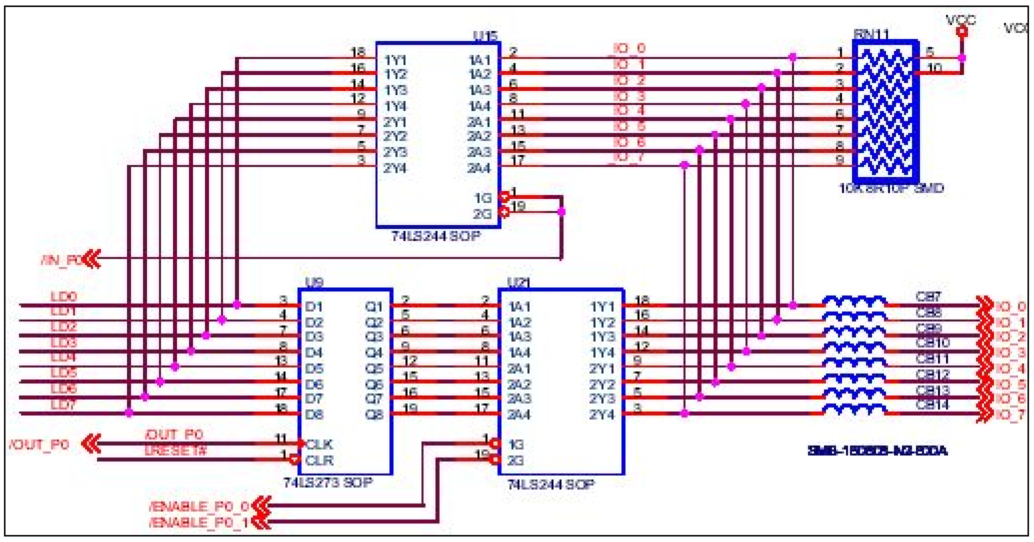


Figure 3-1-14 IO_PORT0 circuit diagram

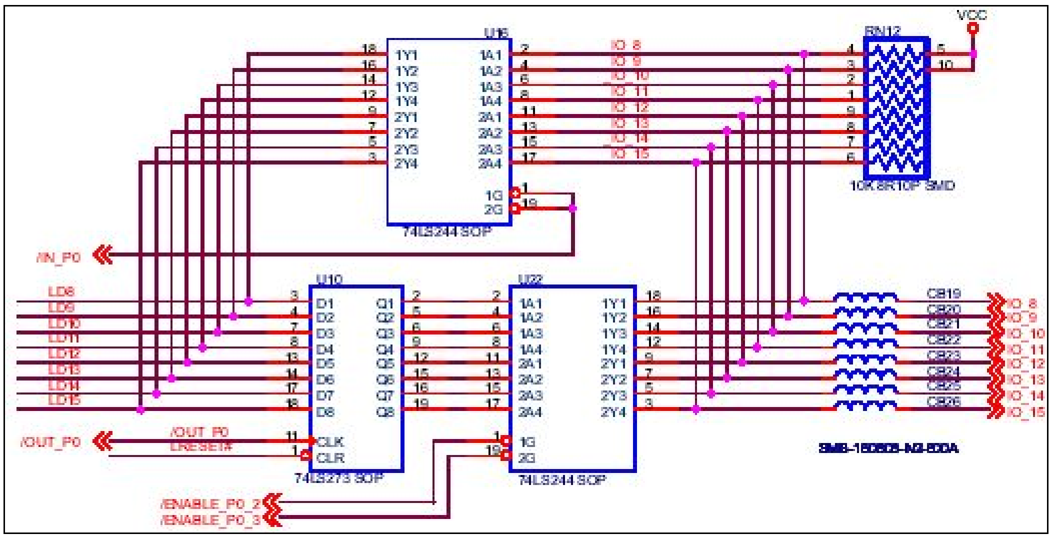


Figure 3-1-15 IO_PORT1 circuit diagram

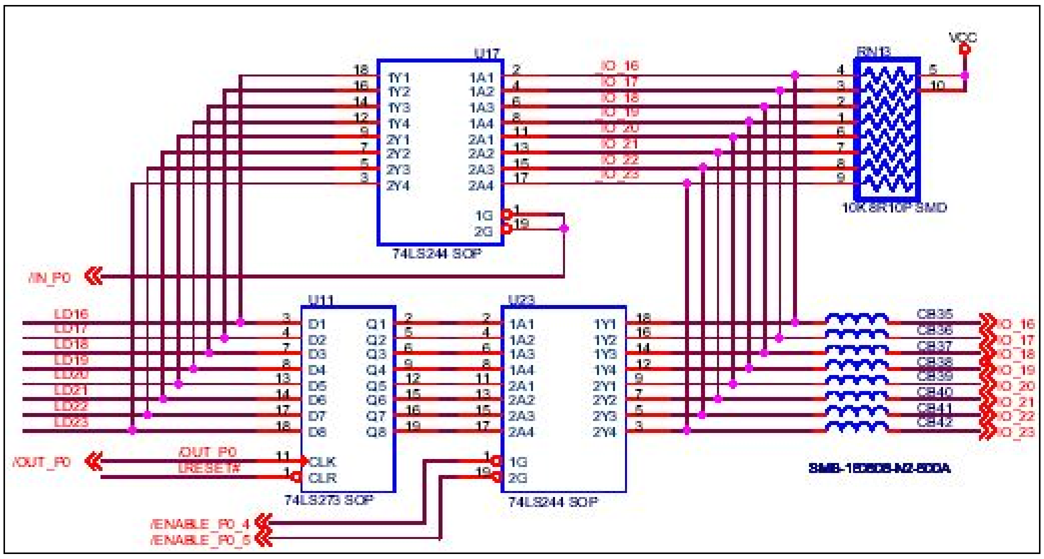


Figure 3-1-16 IO_PORT2 circuit diagram

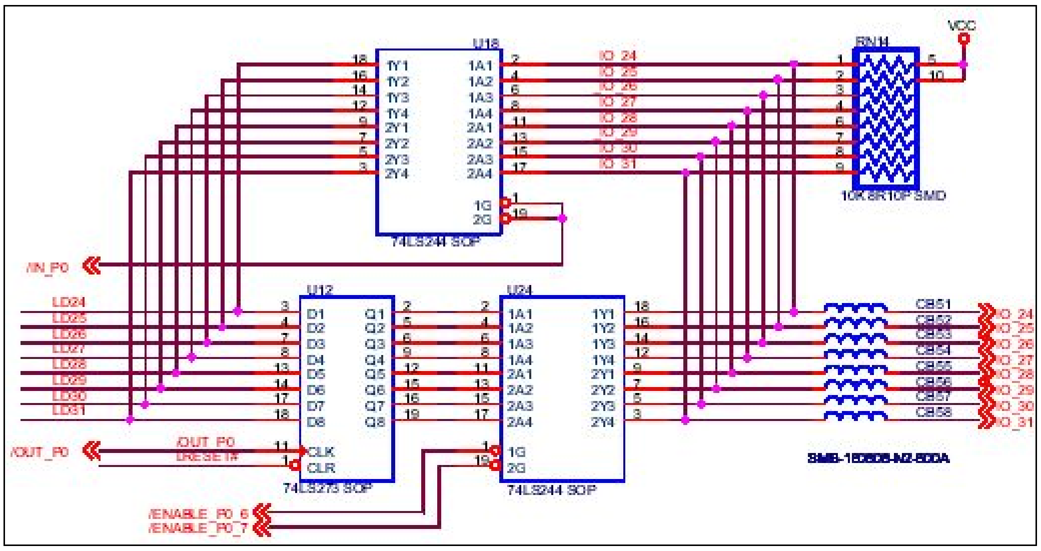


Figure 3-1-17 IO-PORT3 circuit diagram

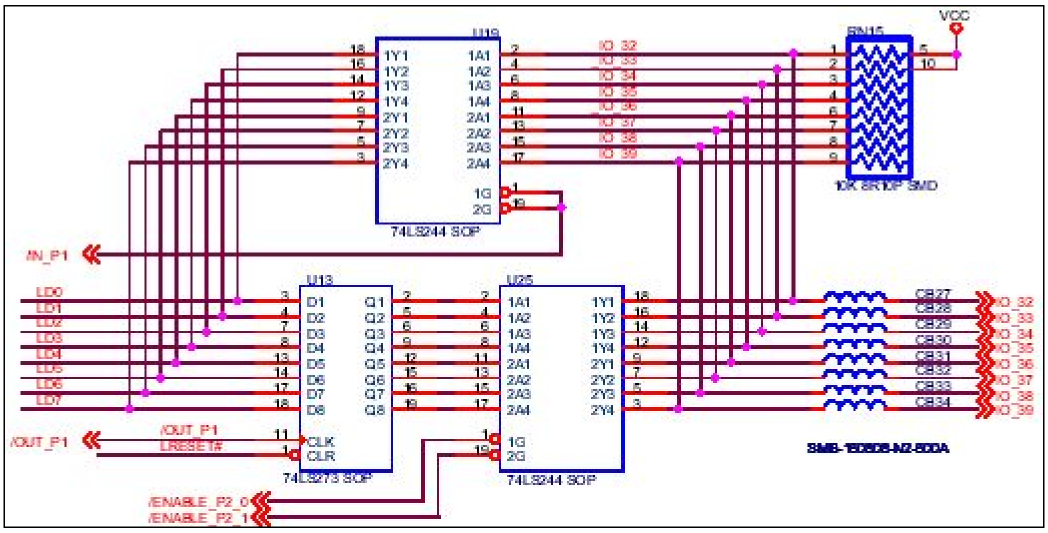


Figure 3-1-18 IO-PORT4 circuit diagram

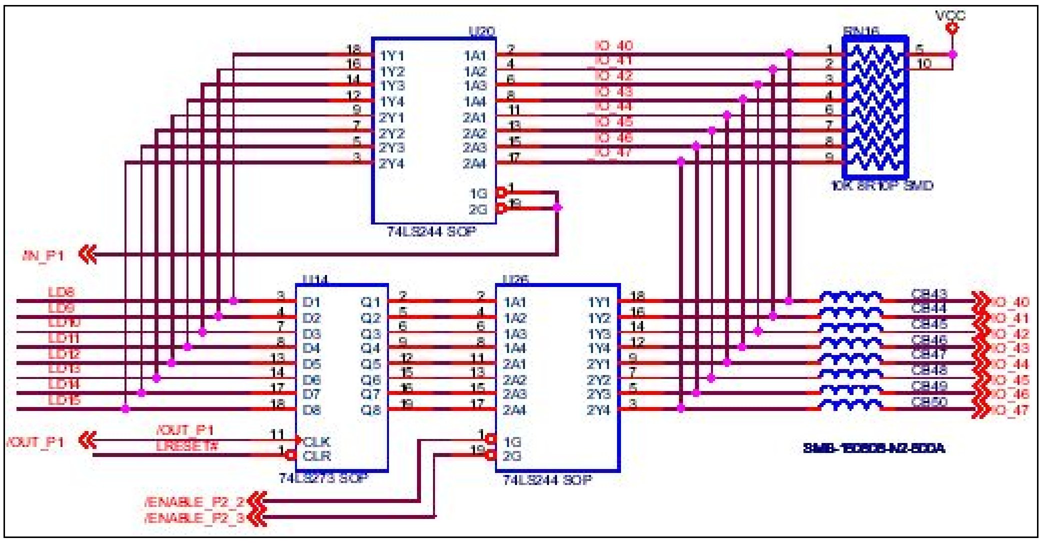


Figure 3-1-19 IO-PORT5 circuit diagram

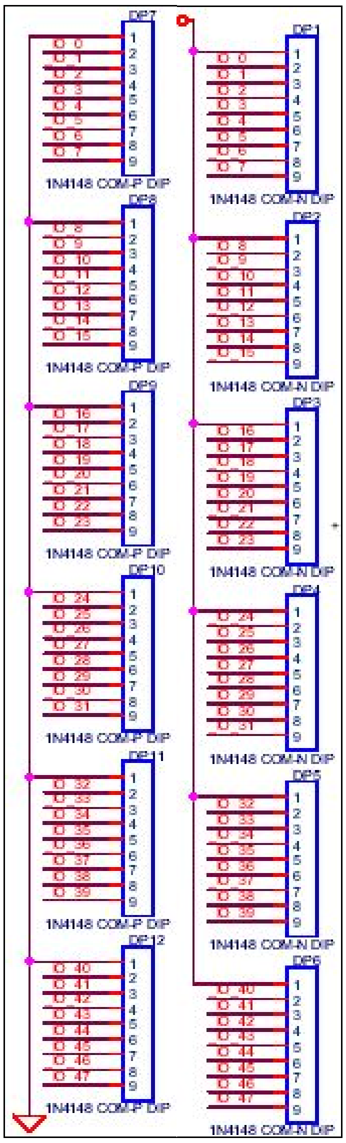


Figure 3-1-20 IO protection circuit

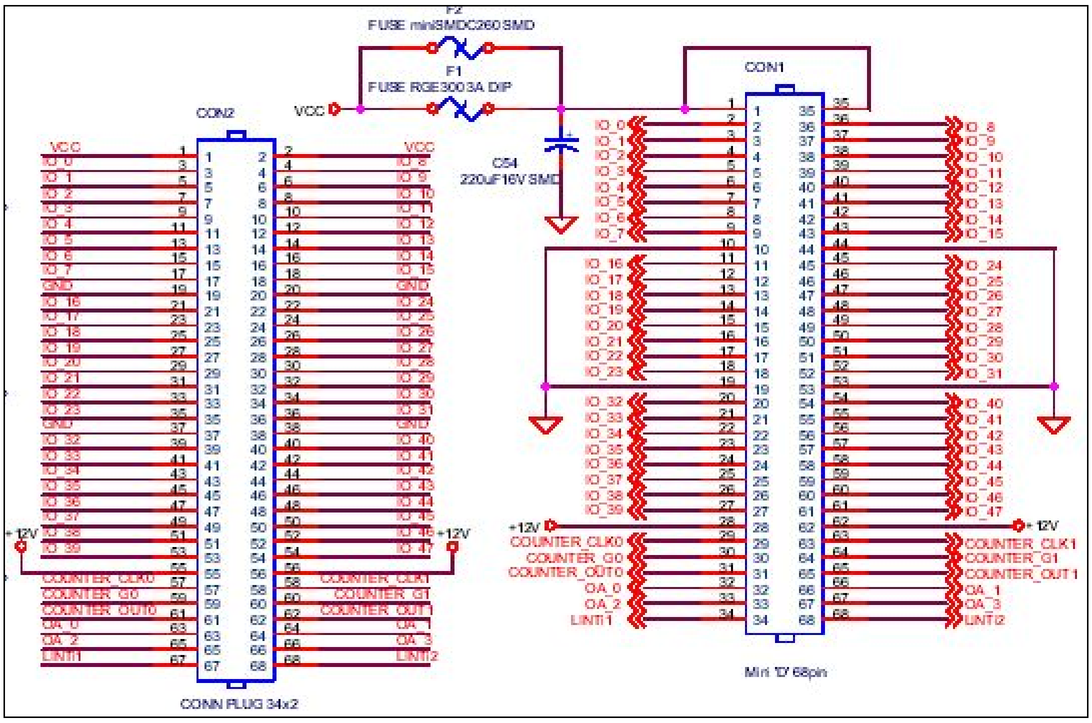


Figure 3-1-21 Output joint

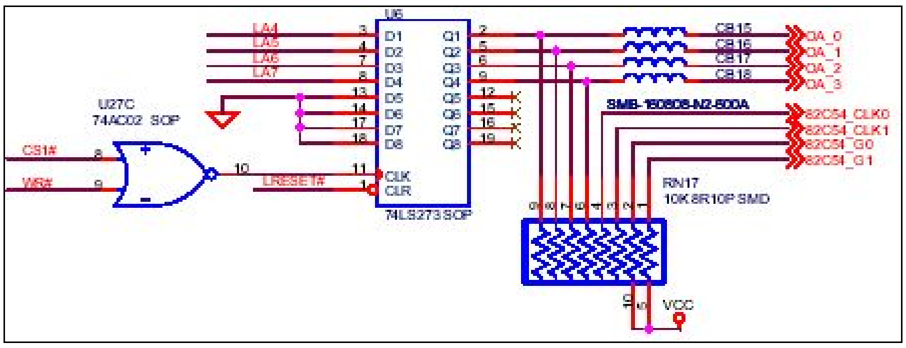


Figure 3-1-22 IO-BANK selection

3.2 PCI-LAB hardware

Figure 3-2-1 shows the hardware diagram of PCI-LAB, this section only describes the common circuit, while the various module circuits are described respectively in implementation.

Figure 3-2-2 shows the power switch of PCI-LAB, when changing PCI-LAB modules, please be sure to cut off the power supply to prevent the PCI-LAB from being burned. As with the power supply of PCI-LAB, we can learn from observing the

LED display in Figure 3-2-3 that when PCI-LAB is operating, the red light of this LED must be on. Close the PCI-LAB, then this LED light is off.

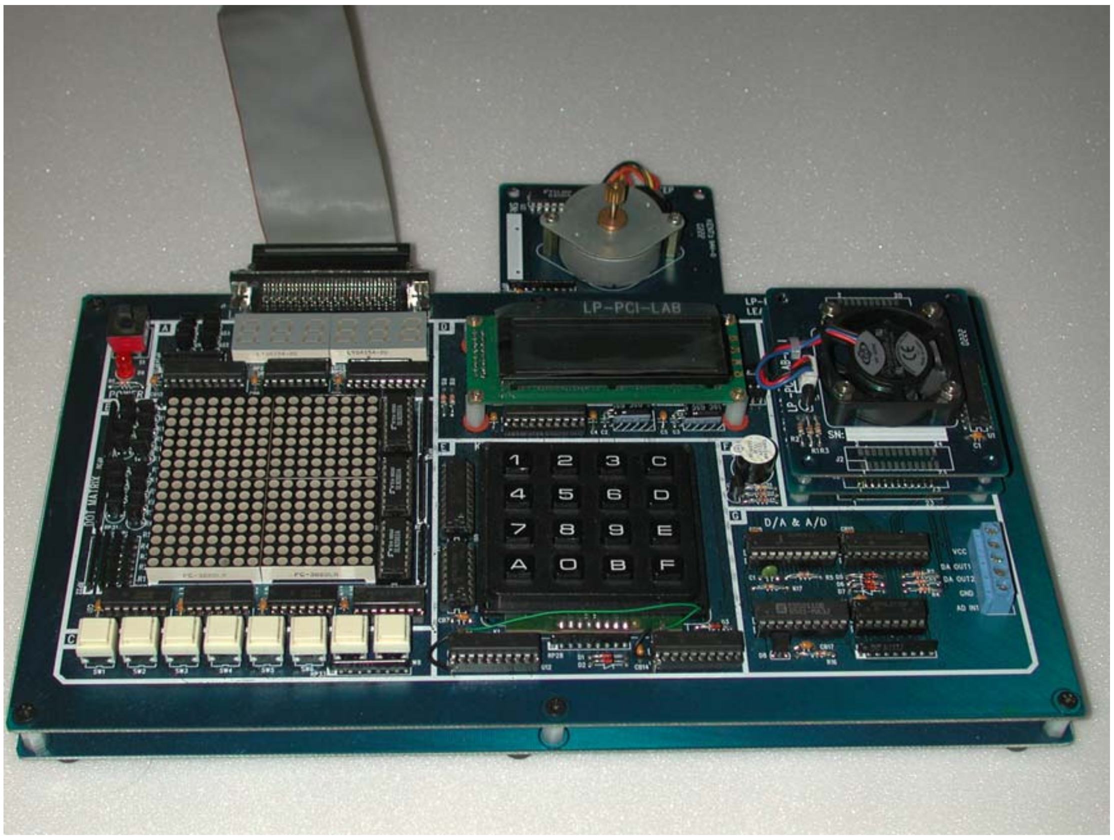


Figure 3-2-1 PCI_LAB experiment module

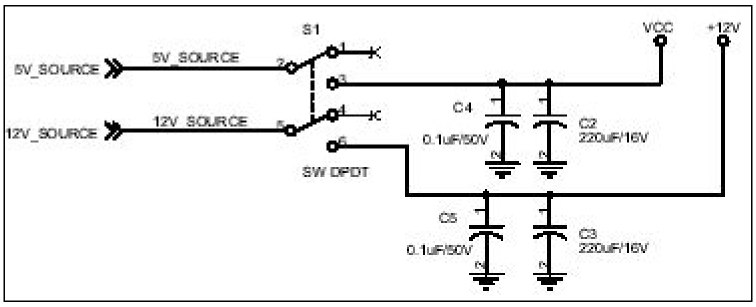


Figure 3-2-2 PCI_LAB power switch

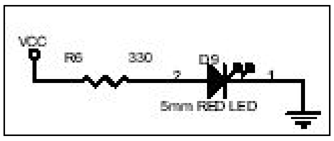


Figure 3-2-3 PCI_LAB power indicator LED light

Figure 3-2-4 shows the 68-pin of PCI-LAB; which is connected to PCI-IO through this, while Figure 3-2-5 is the interpretation circuit of IO-BANK1~IO-BANK4. Figure 3-2-6 and 3-2-7 shows the external pin of the PCI extension module, which can be connected to the external heater, step motor and DC motor external module. Figure 3-2-8 is the output footer of 8254, and is the output measurement point of 8254 experiment. Figure 3-2-9 shows the lines of IO-BANK1 IO-PORT5 on PCI-LAB, which are used to switch the functions of port-used modules. The circuits of the rest of the modules are described in detail in examples.

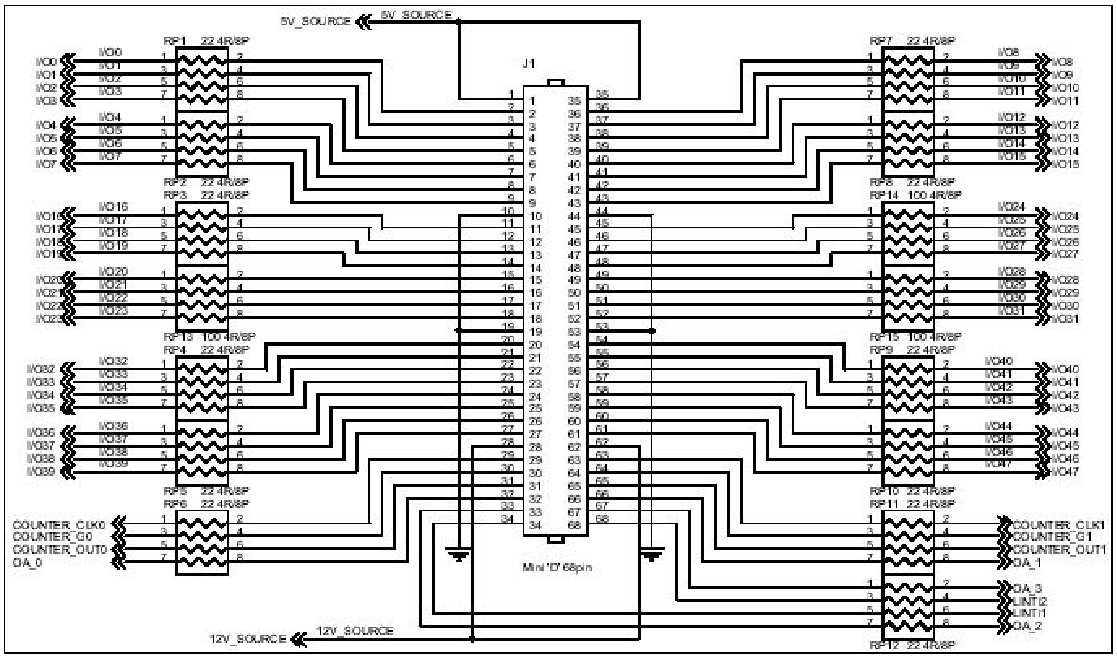


Figure 3-2-4 PCI-LAB pins

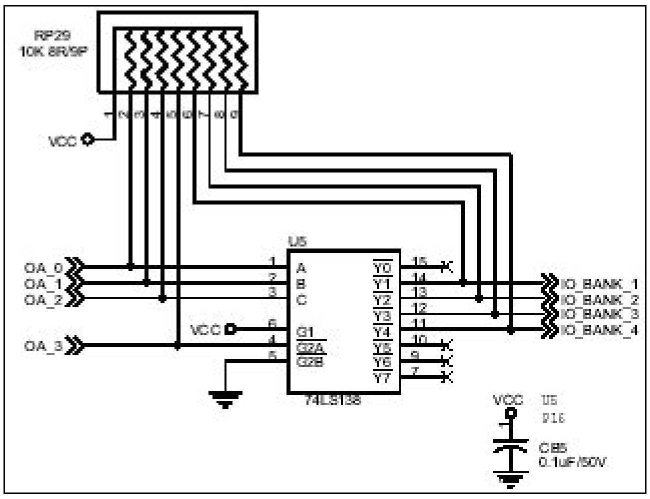


Figure 3-2-5 Explanation of IO-BANK circuit

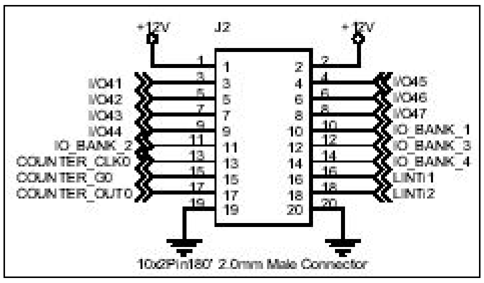


Figure 3-2-6 cartridges pin1

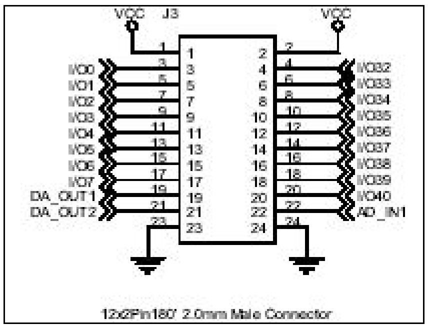


Figure 3-2-7 extension module pins 2

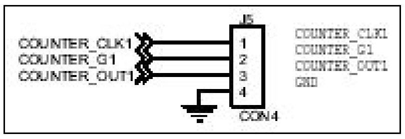


Figure 3-2-8 8254-output ends

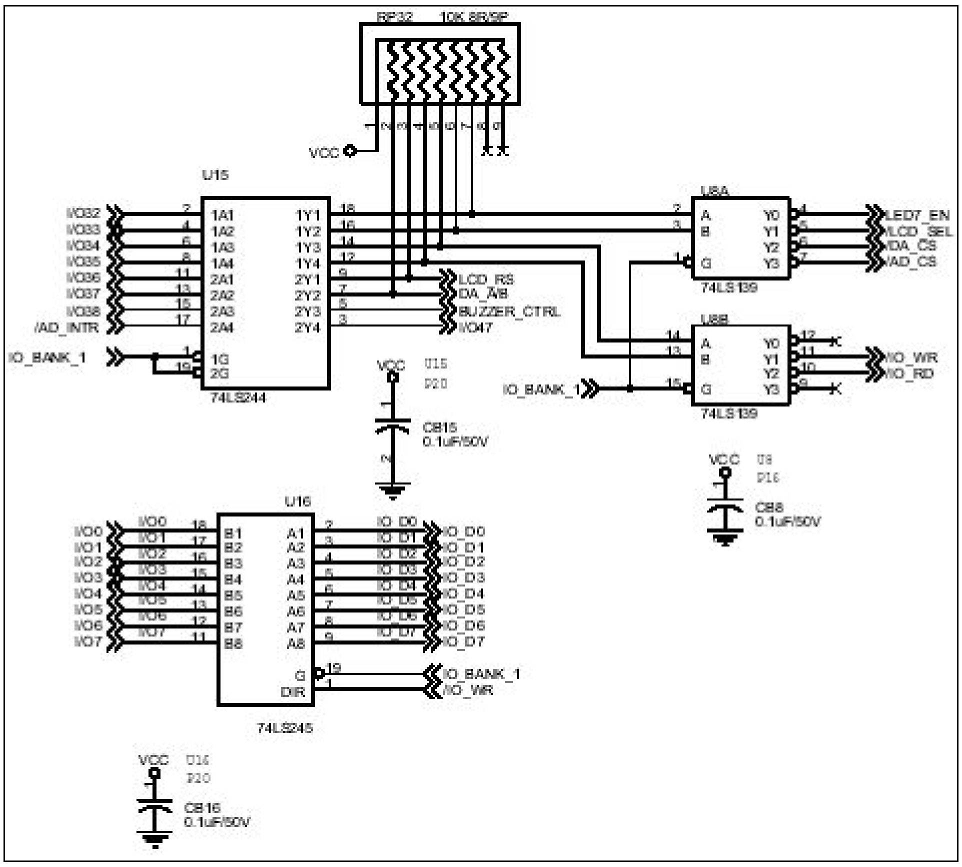


Figure 3-2-9 PCI-LAB function and parameter switch

Chapter 4 Brief introduction of related chips

This chapter introduces PLX PCI9050/9052, 93C46 serial EEPROM and 8254 chips. 9050/9052 chip is the core of the interface card in this experiment, and 93C46 used to memorize the PCI cache data in this interface card, while 8254 is the built-in counter of this interface card, the rest of the parts are 74 series logic gates, so this article only briefly introduces these three kinds of chips.

4.1 PLX9050/9052 chip

PLX9050/9052 chip is PCI Slave chip, mainly connects the PCI bus and common logic bus. This series of chips have powerful functions, which can be designed into a PCI-to-ISA keyed and can use IO output, with several kinds of modes available for choosing. Shown in Figure 4-1-1 below is the sketch of external framework of this series of chip, with serial EEPROM providing 9050/9052 chip PCI Host and Local end configuration information; one end connects PCI bus, the other end can be connected to the IO controller or memory. Generally, this type of chip connects the DSP or FPGA chips, etc., forming an IO control board. Figure 4-1-2 shows the pin diagram of this type of chip.

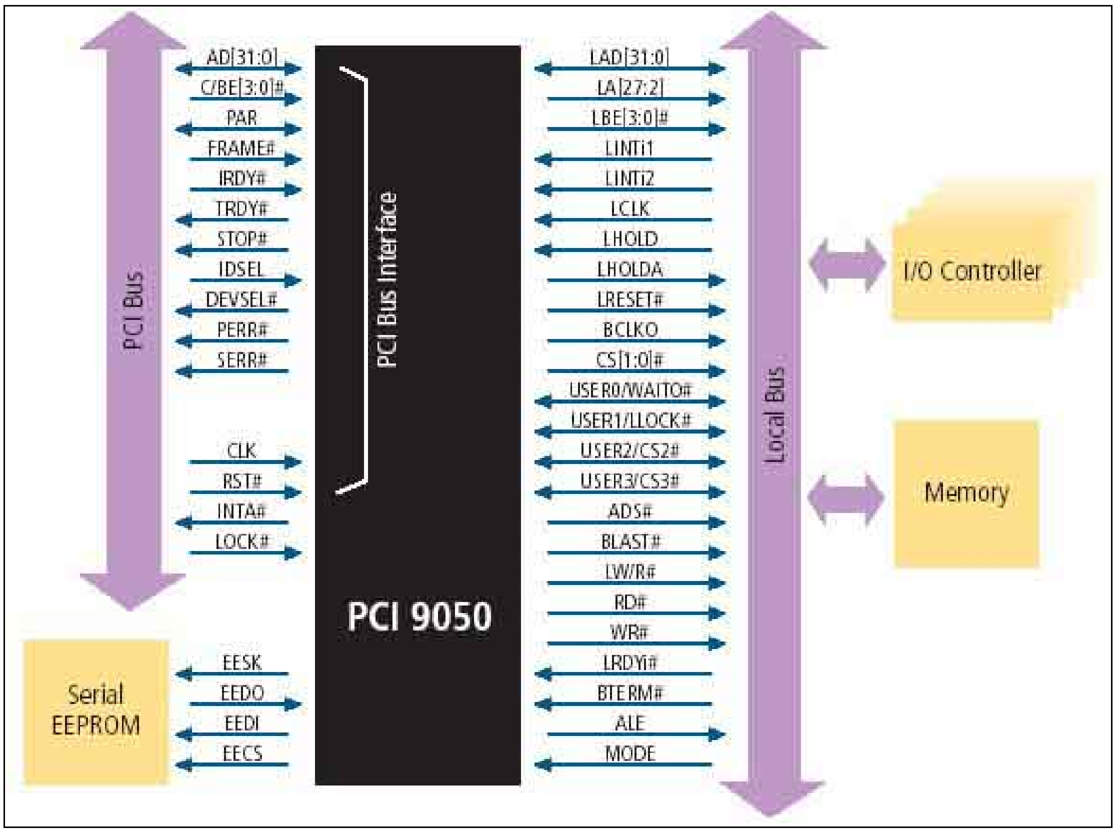


Figure 4-1-1 PCI 9050/9052 external structure diagram

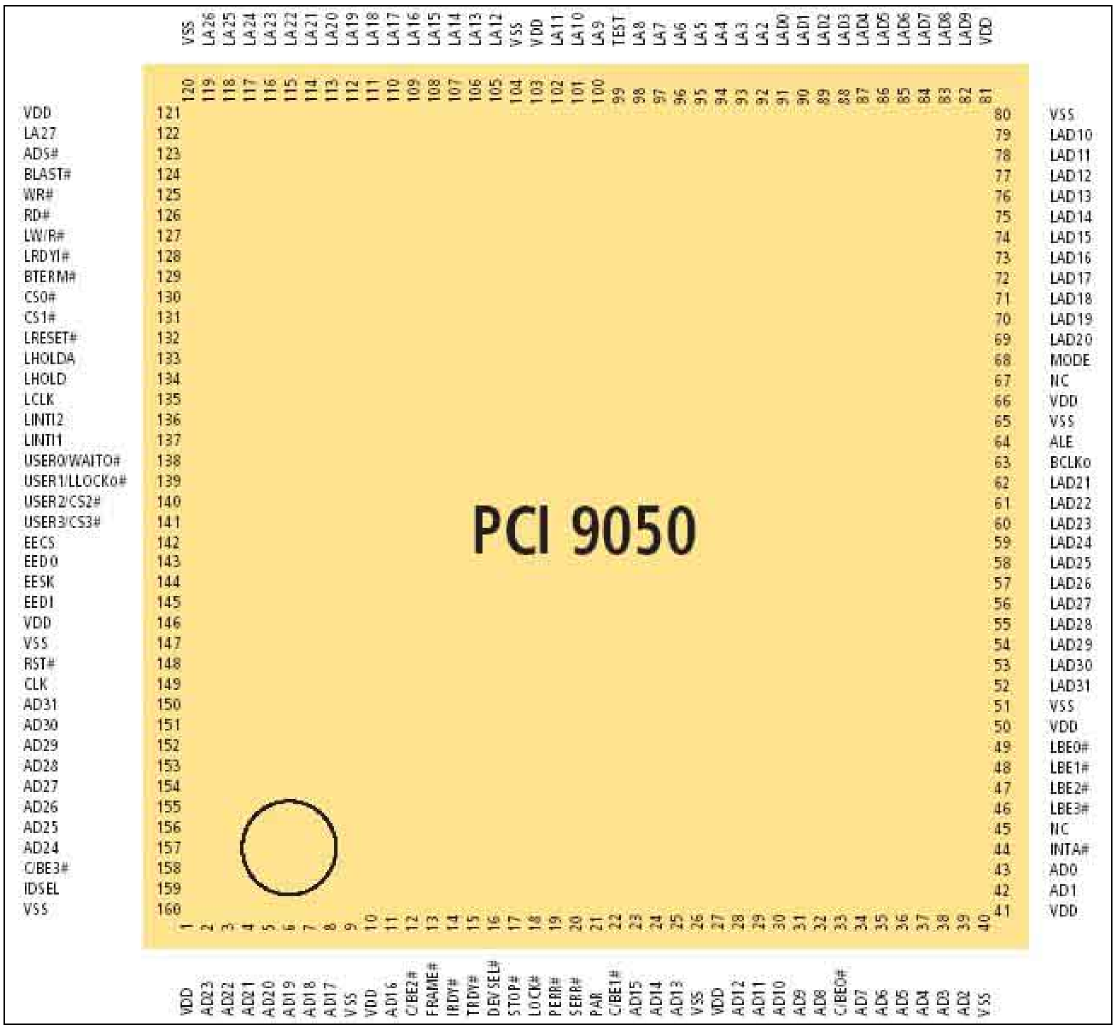


Figure 4-1-2 PCI 9050/9052 pin definition diagram

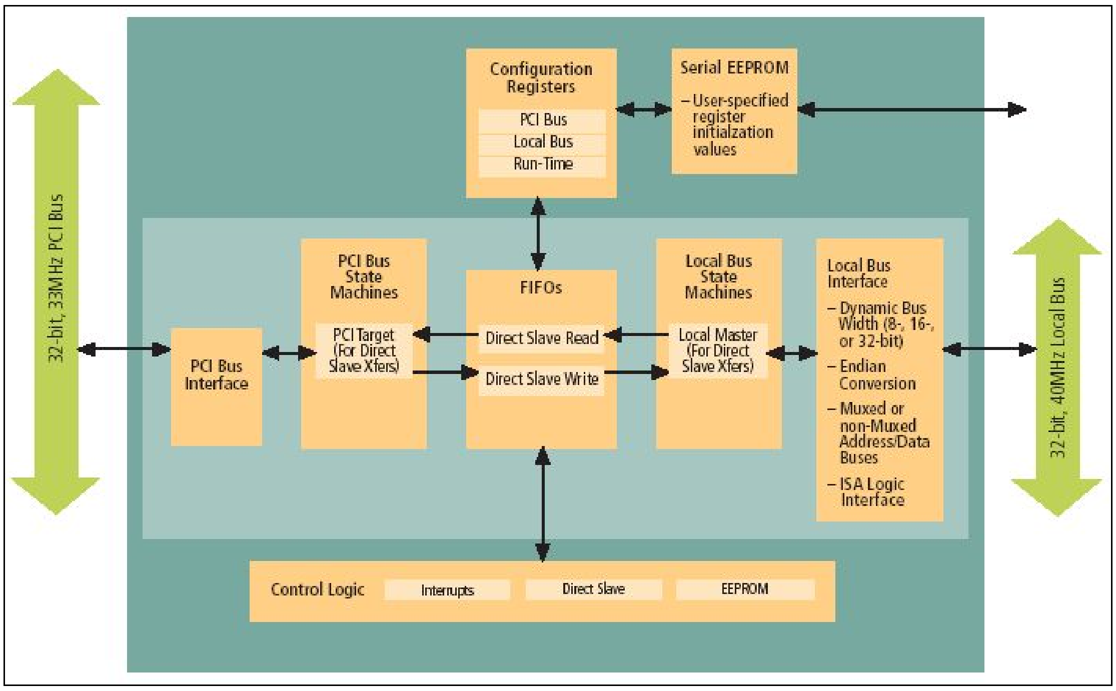


Figure 4-1-3 PCI 9050/9052 internal structure diagram

Figure 4-1-3 is the internal structure diagram of PLX 9050/9052. This chip structure can use PCI function of standard C/C++ language without the need to use the development library supplied by the PLX company and PLX 9050/9052 RDK kit (Rapid Develop Kit) had been issued for many years. If the reader wants to design the advanced PCI experiment card by himself, this kit is another development mode.

PCI instructions supported by PLX PCI9050/9052 chips are shown in table 4-1-1. Because this chip is PCI Slave chip, part of the functions cannot be used. The basic operating mode is divided into 2 modes: Non-Multiplexed and Multiplexed, which needs to be adjusted from hardware as listed in table 4-1-2, while LEAP PCI-IO interface card used in this manual is set to be Non-Multiplexed. The rest of the settings are software settings, stored in 93C46 EEPROM; most of the settings of this chip are done by means of software, which can use PLXMON supplied by the PLX Company to conduct configuration modifications. However, this manual does not provide this function so as to avoid interface card damages.

Here, this manual describes the differences between Big Endian and Little Endian. Figure 4-1-3 shows the comparison between Big Endian and Little Endian. From this, it can be learned that the high, low bits of the two are arrayed in reverse order. Figure 4-1-4 shows the sketch of the conversion between the two, the operating modes of these two memories depend on the differences of the IO controller.

Table 4-1-1 Instructions of this PCI chip

Command Type	Code (C/BE[3:0]#)
I/O Read	0010 (2h)
I/O Write	0011 (3h)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Configuration Read	1010 (Ah)
Configuration Write	1011 (Bh)
Memory Read Multiple	1100 (Ch)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

Table 4-1-2 operating mode

MODE Pin	Mode	Bus Width
0	Non-Multiplexed	32-, 16, or 8-Bit
1	Multiplexed	

Byte Number		
Big Endian	Little Endian	Byte Lane
3	0	LAD[7:0]
2	1	LAD[15:8]
1	2	LAD[23:16]
0	3	LAD[31:24]

Figure 4-1-3 Comparison
Little Endian with Big Endian

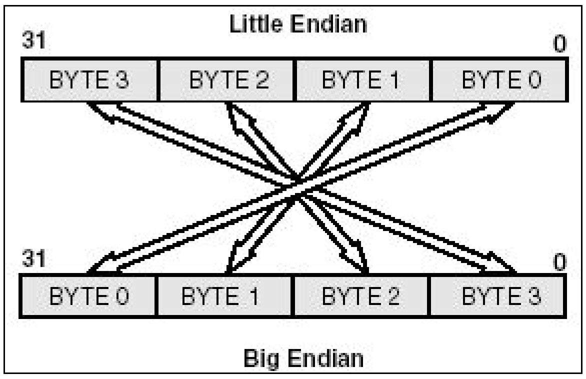


Figure 4-1-4 Changeover between
Little Endian and Big Endian

4.2 Serial RRPROM

Many computer products use memory as its initialization function, which can change the character of the internal firmware data. Additionally, its internal data can be reserved even if the system is closed. The memory used is called by a joint name as non-volatile memory, including ROM, PROM, EPROM, EEPROM and FLASH ROM, etc., EEPROM and FLASH ROM can even be used as RAM. Regarding its use, this type of memory can be divided into serial and parallel data transmission methods, whereby 27Cxx and 29Cxx are serial types, while 93Cxx is parallel. Due to the small sizes and low costs of serial memory, it has been wisely used on the configuration and memory of TV channel selectors, software protectors (Keypro) and PCI-related interface cards and so on.

Serial electronics can erase a larger number of memory pins. This type of memory are mostly packaged by means of DIP, PQFP and PLCC, has 8 bits data bus, and controls memory position through more than 10 address buses. It is often used in BIOS of computer motherboard, BOOT ROM of PCI network card and disk firmware. The writing and burning of its internal programs are common and convenient. It only appears to be complicated because its circuit board has a larger number of pins, but it can still be considered as the mainstream of current erasable programmed memory. Figure 4-2-1 shows the internal structure of 29F002, this type of memory can use a memory capacity of 2048Kbits (8×256Kbits) in total. This is the same amount as other serial memory structure except that the memory capacity and memory block vary with its types. Serial electronic erasable memory usually has only 8 pins, is mostly packaged by means of DIP and PLCC. Due to its smaller size compared with serial memory, it is widely used in the memory of PCI peripheral interface card. This type of memory transmits data and bus in series, reducing the number of pins so as to reduce the size of circuit board significantly. This will be the mainstream standard of future erasable programmed memory.

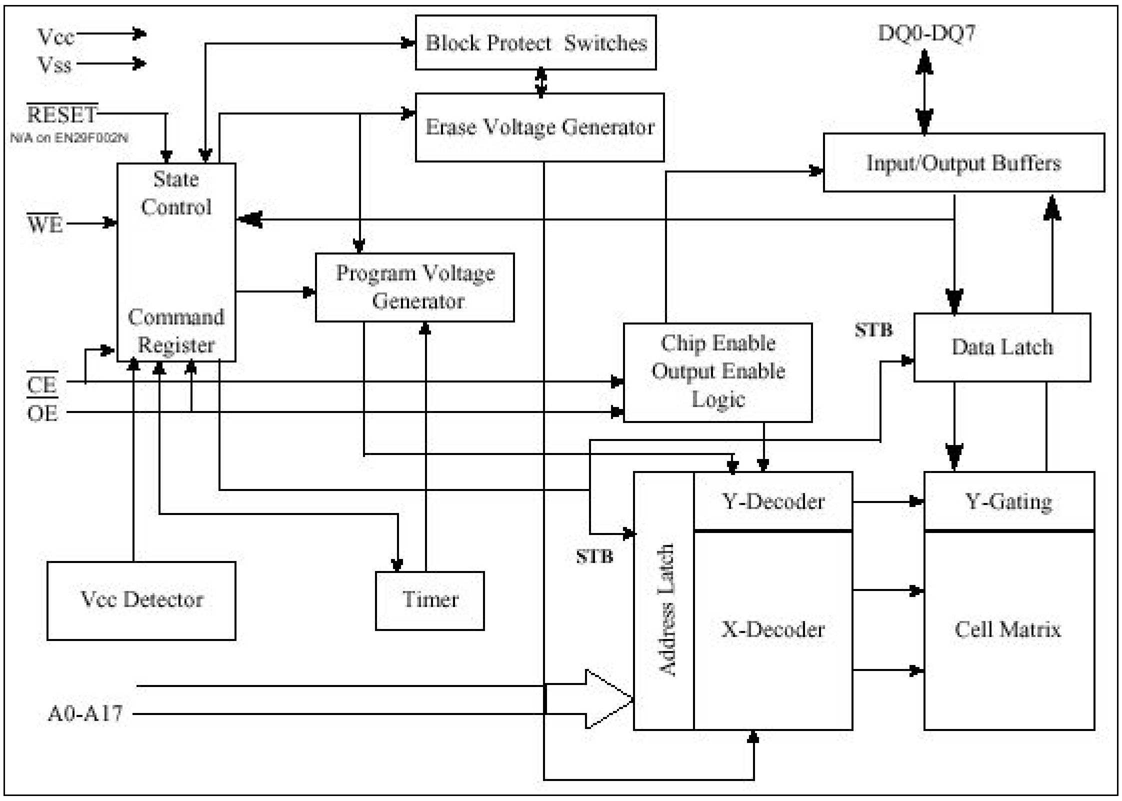


Figure 4-2-1 29F002 internal structures

This section will describe the interface characteristic and application methods of serial EEPROM (96C46, 56, 66series) , type 93C46, 93C56, 93C66 is the difference of memory capacity: 93C46 has1024 bits, 93C56 has 2048 bits，93C66 is 4096bits. Some further instructions, 93C46 single instruction needs to use 9 bits, while 93C56 and 93C66 are both 11 bits, with the rest of the structure and functions being the same. Because this experiment version uses 93C46 as the memory of PLX-9052 chip configuration, we use 93C46 as an example to describe its principle of operation.

Functions and internal structure of 93C46

93C46 is the serial EEPROM that can access 1024bits, with 64 registers inside, the length of each register is 16 bits, 1024 bits (1024 bits) in total. Seven 9 bits instruction controls all actions of this IC, whereby the data can be kept for 10 to 40 years. Figure 4-2-2 and 4-2-3 show its block diagram and pin diagram. This memory has only 8 pins, with the definition of pins shown in table 4-2-1.

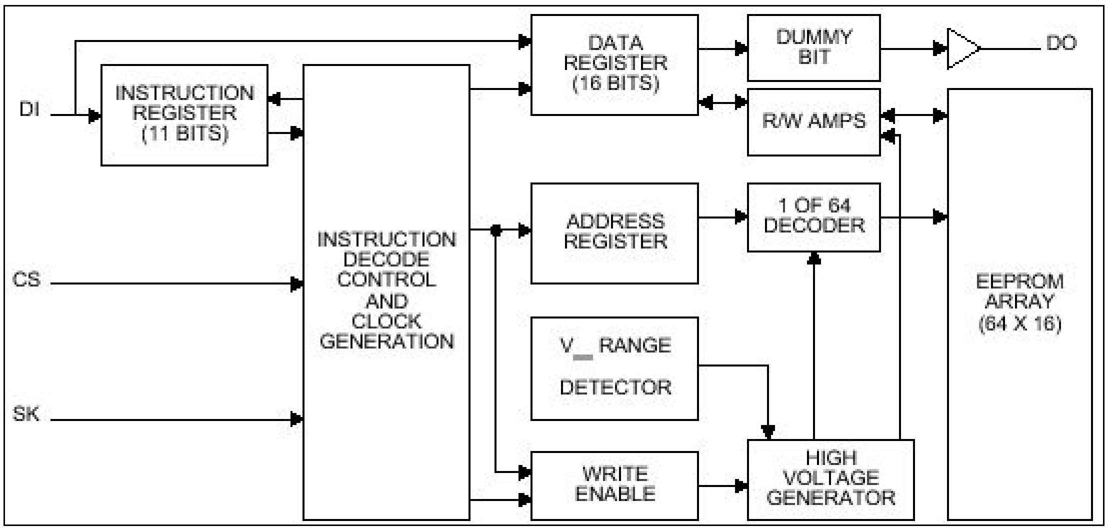


Figure 4-2-2 29C46 internal structures

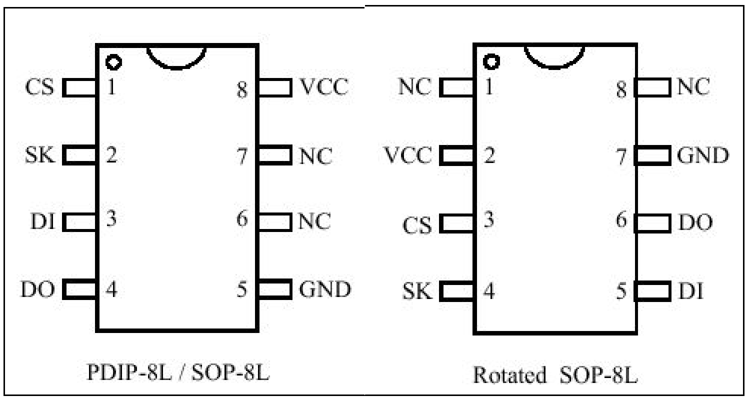


Figure 4-2-3 29C46 pin diagram

Table 4-2-1 29C46 pin definition

Pin	Chinese definition	English definition	Description
CS	Chip select pin	Chip Select	Chip read/write places this signal at high potential
SK	Serial Data Clock pin	Serial Data Clock	Read or write synchronous clock of various bits data action
DI	Serial Data Input pin	Serial Data Input	Serial Data Input pin (go with SK signal)
DO	Serial Data Output pin	Serial Data Output	Serial Data Output pin (go with SKsignal)
Vcc	Power Supply pin	Power Supply	Chip power cord
GND	Ground pin	Ground	Chip ground wire
NC	No Connection	No Connection	No connection (unused)

93C46 instruction structure

7 instruction structures of 93C46 are shown in table 4-2-2, the length of each instruction is 9 bits, which transmits address and data signals in a serial manner mostly in conjunction with the SK clock signal. The first bit is the starting bit and must be placed high. It judges the actions to be taken through the second and third bit control code (OP Code). The subsequent 6 bits is address signal, transmits 16bits data signal again, it will remain in the original action status if it works with SK signal. Thus it needs to wait for clock signals to access data.

Table 4-2-2 instruction structures

Instruction	Definition	Starting bit	OP Code	Address	Input signal
READ	Read	1	10	A5~A0	—
WEN	Write Enable	1	00	11XXXX	—
WRITE	Write	1	01	A5~A0	D15~D0
WRALL	Write All Registers	1	00	01XXXX	D15~D0
WDS	Write Disable	1	00	00XXXX	—
ERASE	Erase	1	11	A5~A0	—
ERAL	Erase All Registers	1	00	10XXXX	—
Description	1: digital signal high potential 0: digital signal low potential				
	A5~A0: Address arranged from the highest bit to the lowest bit D15~D0: Data arranged form the highest bit to the lowest bit				

Read instruction (READ)

The Read instruction is to read the data stored in 93C46 through DO pin in a serial manner. To read memory data, first enter the instruction and address signal in sequence, this action will select the memory register, and place its contents in a 16 bits serial offset register and read the data when the SK signal is on the rising edge. The serial data read is outputted in the order of “high bits first and low bits later” with timing diagram shown in Figure 4-2-4.

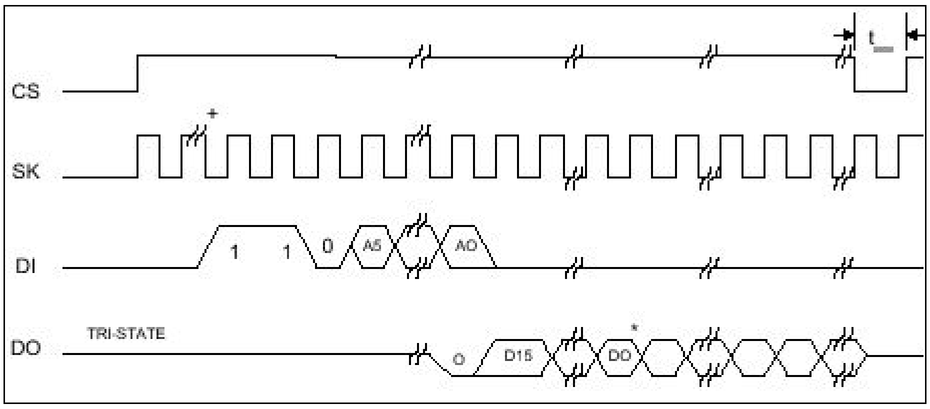


Figure 4-2-4 read timing

Write Enable instruction (Write Enable) WEN

To write data into the EEPROM 93C46, this instruction must be issued before data can be written. Otherwise the write action will be invalid. For this chip, after adding power is turned on, EEPROM is in an unwritable status, and after this instruction is issued, it enters the writable status. Data write is controlled through write instructions. After the writable status is enabled, it will be maintained until the power supply disappears or issues instructions that forbid the write function to close write. Figure 4-25 contains the timing diagram of write enable.

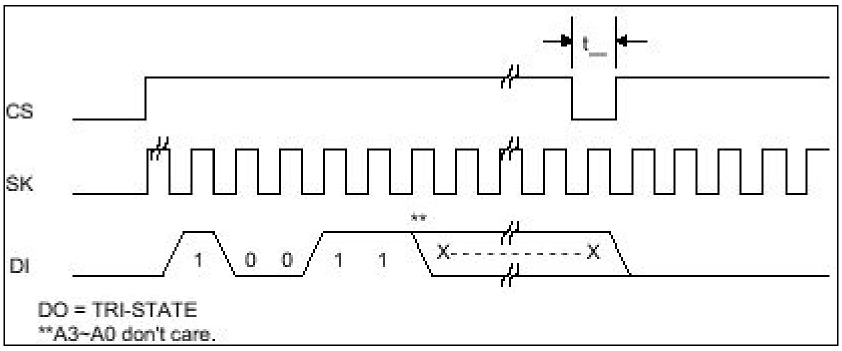


Figure 4-2-5 Write enables timing

Write instruction

Write instruction writes 16 bits data into the designated memory address. Input the instruction and data in conjunction with SK clock signal in a serial manner. After write is completed; CS should be kept at a low potential (at least 250ns). When CS changes back to high potential, DO pin is at low potential and indicates that the write action is not completed (busy). Oppositely, if DO pin is high potential, it indicates that the write action has been completed (ready) and the next instruction may be implemented. Before proceeding to this instruction, write enable instruction must be executed first so that write action can operate properly. Figure 4-2-6 is the timing diagram of writes action.

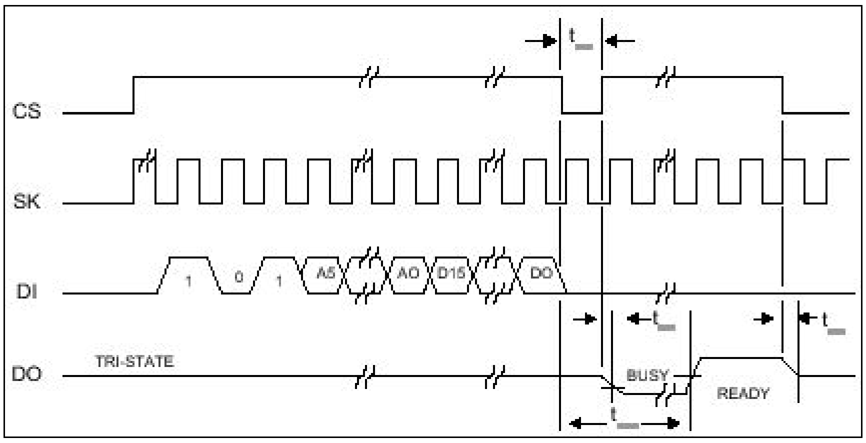


Figure 4-2-6 Write timing

Write all instruction

Write all instruction writes 16 bits data after the instruction into all memory registers, whose contents are the same. The address column of this instruction is useless. After completing inputting instructions and data, CS must be kept low potential (250ns). When CS changes back to high potential, DO pin can be used to indicate whether the write action is completed or not, which is the same as the method of write instruction in operating. Figure 4-2-7 shows the timing diagram of this instruction.

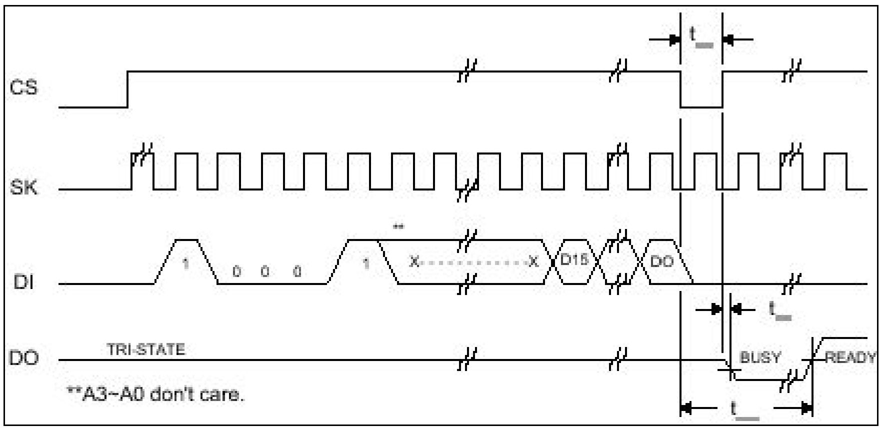


Figure 4-2-7 Write all timing

Write disable instruction

Write disable instruction forbids any write action, protects the data from being modified unexpectedly. After the data is written, this instruction can be issued to ensure the safety of the data. Its timing diagram is shown in Figure 4-2-8

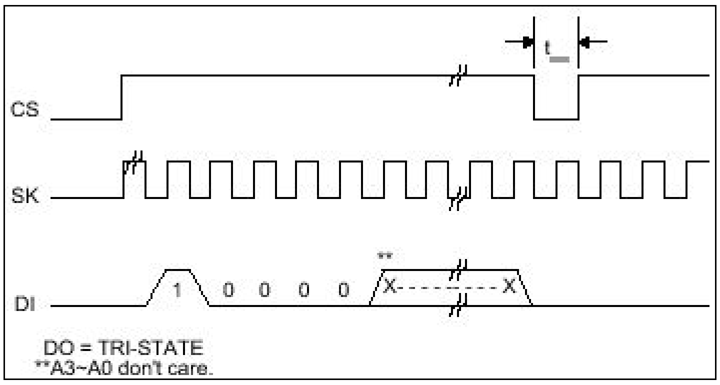


Figure 4-2-8 Write disable

The instruction to erase register contents (Erase)

When erasing the contents of a memory register, this instruction, which erases the register contents of designated addresses, may be issued. After this instruction has been issued, it is necessary to judge whether this instruction has been executed, which is of the same manner as write instruction in operating, with timing diagrams as shown in Figure 4-2-9.

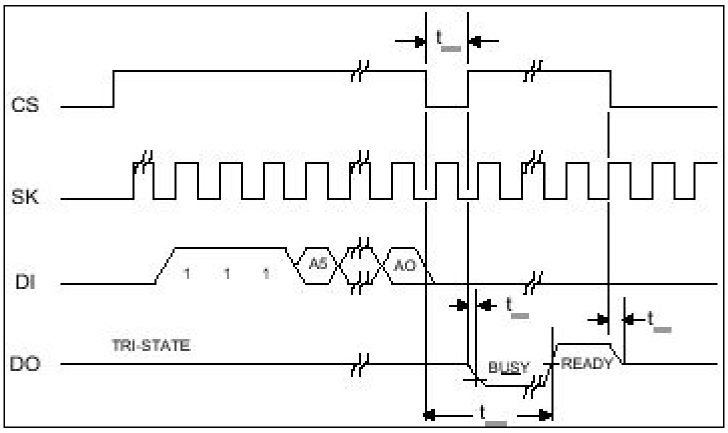


Figure 4-2-9 Erasing register contents

(Erase All instruction) RAL

After this instruction is issued, all the contents of registers change to high potential. The same thing occurs with write instructions in operating mode. With the timing diagram as shown in Figure 4-2-10. Generally, before data is written, the contents of the registers may be erased first, and the write action may be carried out.

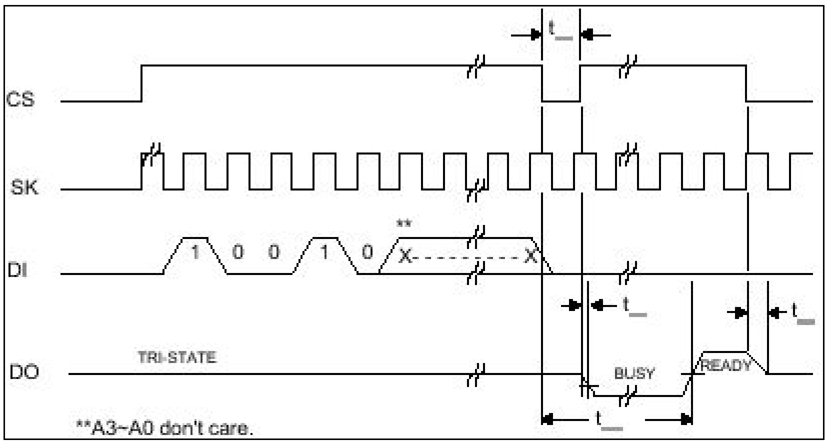


Figure 4-2-10 Erase all timing

4.3 8254 timing/counting chip

It is a simple method to count time with software. However, the accuracy is not high. The timing speed is affected by operating systems and peripheral hardware, unable to achieve the accuracy of "within one 1ms". When the software timing program is executed, the execution of other programs is interrupted. Moreover, it consumes the time for the microprocessor to process. Consequently, the hardware counter appears to be more important to achieving real-time execution efficiency and peripheral execution.

8254 is the timing/counting chip that can be programmed, is widely used in design and experiments of ISA interface, with maximum operating frequency of 8 MHz. The modified 8254 are 10MHz, can still be used in experiment of PCI interface. Generally there are speedier timer/count chips available for use. This chip has a read-back function, which can lock information such as count timer value or status in any case to facilitate reading. Each 8254 has 3 sets of independent counters, the count length is 16 bits, the highest count frequency is 8MHz, 2 sets of counters can be connected in a series to be used to achieve a count length of 32 bits. Figure 4-3-1 is the function block diagram of 8254 timer interface, while 4-3-2 is the pin diagram of 8254.

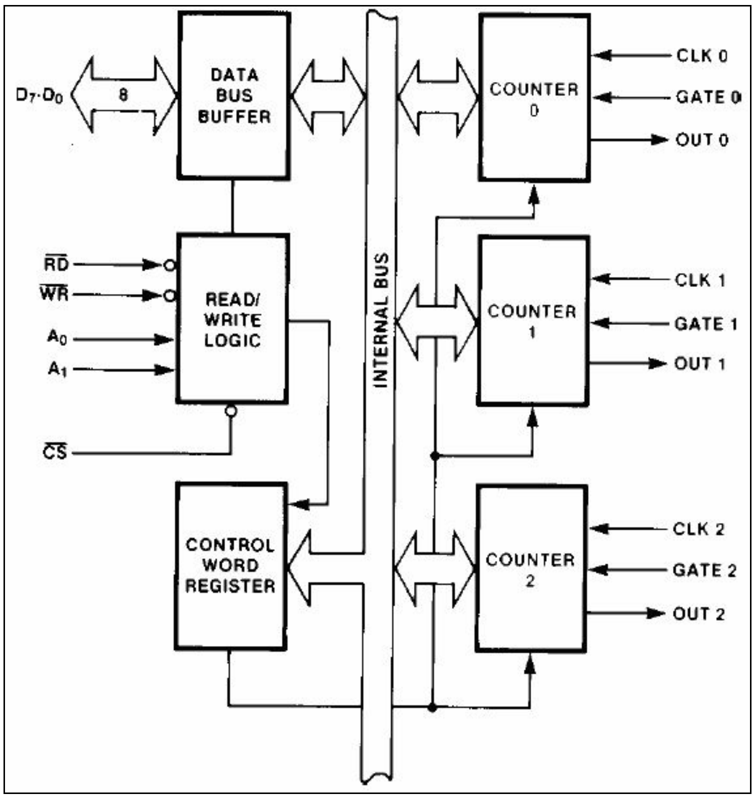


Figure 4-3-1 8254-function block diagrams

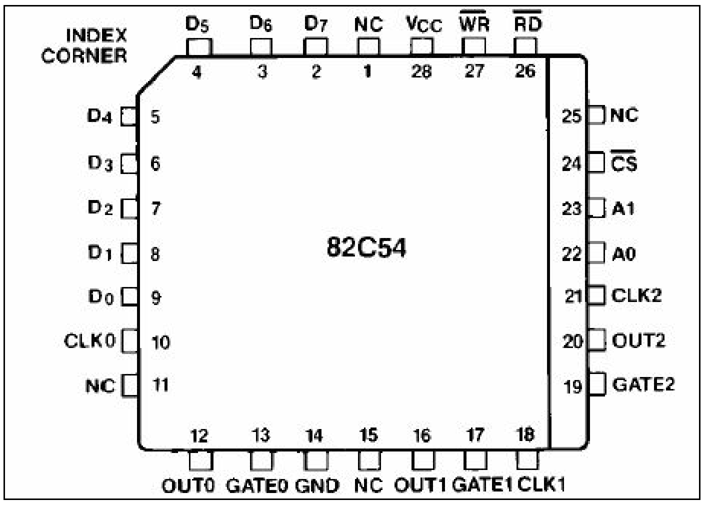


Figure 4-3-2 8254-pin diagrams

Like 8255A, the use of 8254 chips needs 4 I/O addresses, which are selected by $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A0 and A1 pins of 8254. $\overline{RD}$, $\overline{WR}$, Pins are I/O read/write control signals, $\overline{CS}$ chip selection signal is connected to PLXPCI9052 chip, so I/O address assigned by 8254 is A40h~A403h of IO BANK0. A0 and A1 pins are connected to low base address line A0 and A1, used to select any of the four I/O addresses in 8254.

The 3 counters of 8254 each use an I/O address (A400h~A402h), and the remaining I/O addresses are used for 8254 control ports (A403h), like the control port of 8255A, control bits are written into 8254 control ports so plan the working modes of the counters. Each set of counter of 8254 has 3 pins: clock input end (CLK), GATE control input end (GATE), and signal output end (out), with the functions of each pin described as follows:

Table 4-3-1 Counter pins

CLK	8 MHz The input end of base clock for timing; the maximum clock frequency allowed by 8254 is 8 MHz.
GATE	Used to control the start and close of counter, its function is determined by the working modes set by 8254 control ports.
OUT	The output end of counter, which can be connected to CLK input ends of other counters to achieve a counting length.

8254 chip has 24 pins, including bi-directional data bus D0~D7, control lines A1, A0, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$ and 3 sets of counter pins CLK[2:0], GATE[2:0], OUT[2:0], the function of each pin is shown in table 4-3-2 below. Because address lines are used to select 8254 internal counters and control the register, table 4-3-3 below describes its actions:

Table 4-3-2 Pin functions

Pin	Definition	Input/output
D0 ~ D7	Bi-directional data bus	Bi-directional
CLK0 ~ 2	Counter timing input	Unidirectional input
OUT0 ~ 2	Counter output	Unidirectional output
GATE0 ~ 2	Counter gate input	Unidirectional input
$\overline{CS}$	Chip selection	Unidirectional input
$\overline{RD}$	Read control	Unidirectional input
$\overline{WR}$	Write control	Unidirectional input
A1 A2	Address line	Unidirectional input
Vcc	power (+5V)	------
GND	Earth wire	------

Table 4-3-3 counter selection

A0	A1	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control bit register

8254-control field

When 8254 is started initially, its internal counter working modes, count value, and output signals are in a undefined status, the counter will not enter the usable status before a proper control bit is written into the control port and the working modes of the counter and count values are planned. In the summaries of the above tables and figures, the basic read and write modes of 8254 are described as shown in table 4-3-4.

Table 4-3-4 8254-control fields

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	Function
0	1	0	0	0	Date write counter 0
0	1	0	0	1	Date write counter 1
0	1	0	1	0	Date write counter 2
0	1	0	1	1	Date write control register
0	0	1	0	0	Read counter 0
0	0	1	0	1	Read counter 1
0	0	1	1	0	Read counter 2
0	0	1	1	1	No action
1	×	×	×	×	No action
0	1	1	×	×	No action

8254 working mode

8254 counters can be planned into 6 working modes via control bits. The following is the description of these 6 working modes.

◎Mode 0— (Interrupt on Terminal Count)

Mode 0 is mainly used for event counter, with 8254 counter controlling a certain amount. If the preset amount is 500, first set the counter to mode 0, load count value 500, CLK signal will count accordingly, the count value, subtracting 1 each time, until the count value is subtracted to 0. The “OUT” output end sends out a signal, which can be taken as Interrupt request signals to make data processing easier. Under mode 0, the counter actions are:

1. When it is set to mode 0, the OUT output signal of counter is low
2. GATE signal is used to control the counting actions of counter. When GATE=high, the counter counts backwards properly; conversely; if GATE =low, then current counting is paused. The counter continues to count until GATE signal again rises to high.
3. After new count values are loaded into the counter, the counter begins to act, now GATE signal must be high, and count value automatically subtracts 1 on the decreasing edges of CLK signal. At the terminal of counting, OUT signal

will be outputted high and the current counting is stopped. Unless new count values are loaded or new working mode is set, the OUT signal will maintain high and the count values will continue to be 0.

4. Again, when a new count value is loaded during counting, the counter will count from the new count value until the end of counting. If a 16 bits count value is loaded, the following two events may take place: first when writing high byte count values, the current counting is terminated; then when writing low byte count value or loading new count value, the counting work may continue.

◎Mode 1—Hardware Retriggerable One-Shot output

Mode 1 is to take the counter as a one-shot generator that can be planned. The GATE input end is taken as a trigger signal, when the GATE signal is triggered, a pulse wave signal that can plan time width will be generated, which is called a one shot signal. The pulse wave width of this one shot signal is jointly determined by the count value of 8254 counter and pulse wave frequency of CLK input end. Under mode 1, the counter actions are:

1. After mode 1 is set, the OUT signal of the counter will be outputted to high.
2. After loading the count value and on the rising edge of GATE signal, the user adjusts the OUT signal to low and the counter begins to count backwards. When the counter stops counting backwards, OUT signal will again rise to high.
3. One shot signals can be repeatedly generated; if after the end of counting, GATE signal is again triggered, another shot signal will be generated in the same order as in item 2, with pulse wave width being the same as the old ones.
4. This mode has the retriggerable function. If during counting (that is, OUT signal is low), there is another trigger signal of GATE rising edge, the counter will start to count backwards from the original count value until the end of counting. This retriggerable function will make the pulse wave width of one-shot signal become longer.
5. If a new count value is loaded during the process of counting, the original pulse wave width may not be affected. The counter will count from the

loaded count value when the next GATE signal was triggered.

◎Mode 2—Rate Generator

When it is set to mode 2, the counter may divide input pulse waves from CLK with N, and can often be used as interrupt signal generators of real-time pulse waves. The value of N here is determined by the count valued loaded, but the value should not be 1. During each counting cycle, OUT signal is low. Except for one short CLK cycle, during the rest of the time, the OUT signal is high. Under mode 2, the counter actions are:

1. After it is set to mode 2, OUT signal will maintain high. The counter will begin to count backward when count value is loaded.
2. When GATE = high, the counter can count backwards; conversely, if GATE= low, the current counting is stopped and forces OUT signal to be outputted high.
3. During the process of counting (that is, GATE is high, and after count value is loaded), OUT signal is maintained at H. Till the time when the counter counts backward to 1, OUT output is low, then to 0 (that is the end of counting). Now OUT signal will again rise to high. If after the end of counting, GATE signal remains high, the counter will again begin to count from the original count value, and repeats in cycles until the counter changes to another working mode, or a new count value is loaded, or the GATE signal comes to high.
4. If a new count value is loaded during the process of counting, the current counting cycle may not be affected. It will count from the newly loaded count value till the next counting cycle.

◎Mode 3-- Square Wave generator mode (Square Wave Mode)

Except for different output of OUT signals, the counter action of mode three is similar to that of mode 2. When operating under this mode, the counter, after dividing the signal frequency from the CLK input end with N, outputs a square wave whose duty cycle is about 50% from the output end. That is, the square pulse wave whereby high and low account for half of the time respectively. When N is even, the square wave is half of the time occupied by high and low multiplies cycle time of the CLK pulse wave. When N is odd, then the time taken by high is-- (N+1) / 2 multiplies cycle time of CLK pulse wave, while the time taken by low is (N－1) / 2 multiplies cycle time of CLK pulse wave. In other words, if N is odd, then the high of the OUT signal will have one more CLK time then the low.

◎Mode 4-- Software triggered strobe control signal output (Software Triggered Strobe)

This working mode is to take the counter as strobe control signal generator triggered by software. Under mode four, the actions of counter are:

1. After it is set to this mode, OUT signal will be outputted high.
2. GATE signal can control the actions of counter. When GATE is high, the counter can count properly. When GATE is low, the counter stops counting. However, GATE signal may not affect output to OUT.
3. After the count value is loaded (now the GATE signal is high, output of OUTPUT signal should be low), the counter starts to count backwards. At the end of counting, OUT output is low, and counts to 0 again. After lasting a period of one CLK cycle, it again rises to high, which is the control signal. To send out a new strobe control signal, a new count value may be loaded upon the end of counting.
4. If a new count value is loaded during the process of counting, the counter will continue to count from this new count value until the end of counting. If a 16 bit count value is loaded, the following 2 events may take place: first, when the high byte count value is written, the current counting is stopped; then when writing low byte count value, the counting work may continue from the count value loaded.

◎Mode 5--Hardware Triggered Strobe signal output (Hardware Triggered Strobe)

Like mode 4, this mode also takes counter as the generator of strobe control signal. However, mode 5 takes the rising edge of GATE signal as the trigger signal of strobe control. Under mode 5, the actions of the counter are:

1. After it is set to this mode, the OUT signal will be outputted high.
2. After the count value is loaded, the counter will not count backwards, it will

not begin to count before the trigger signal of GATE is on a rising edge. At the end of counting, OUT is low, and will rise to high after lasting for a period of a CLK cycle.

3. If before the end of counting, GATE signal is again triggered (rising edge), the counter will count from the original count value until the end of counting, namely the counter now has the function to be triggered again.
4. If a new count value is loaded during the process of counting, the current counting work may not be affected. Only when the next GATE signal is again triggered, the counter will count from the new count value.

Read the count value of 8254

It is usually desirable to read the present count value of 8254 without interfering with the counting work of 8254. Generally speaking, there are 3 methods to read 8254 count values, which are described as follows:

◎ Common read command

Read count value with common control port command. The shortcomings of this method is that it is unable to read the count value that is being counted, to read the right count values, the counter must stop working.

◎ The locking command of counter

In the control bit of 8254, there is the command of counter locking. This command can lock the count value of designated counters, then read count value with common counters. Like control bits, this command must be written into the control port of 8254 in the following formats:

SCI and SCO are used to select 3 counters, D4, D5 must be 0, while D0~D3 is any value; usually we set them to be 0. After this command is written into the control port, the count value of selected counter will be locked in the latch circuit inside 8254, and be kept until it is read. After the count value in the latch circuit is read, data in the latch circuit will change with the count value.

◎ Read-back command

This command can not only read the count value of the counter, but also read the status of the counter. This command is also written into the control port by means of a control bit, which is described as follows:

1. CNT0~CNT2 are used to designate counters, that is, when either of CNT0~CNT2 is set to 1, one counter is designated. For instance, when CNT0=1, this read-back command is only valid for counter 0.
2. When COUNT=0, lock the count value of the selected counter for reading, this command has the same function as the above-mentioned counter locking command.
3. When STATUS=0, lock the selected counter status data, then the status data of the counter can be read.

As with initial value setting of 8254, when power is turned on, 8254 remains in an undefined status, the output of various counters, count values and operating modes are not defined. To use 8254 to make it act properly, control field data needed must be written into registers, the unused counters need not be set.

D7, D6 (SC1, SC0)

D7 and D6, the two highest bits in control field, are used to select any of the three counters or read back instruction. To select counter 0, SC1 and SC0 in control field must be set to 0; to select counter 1, then SC[1:0] =01. Similarly to select counter 2, SC[1:0] =10. It is read back instruction when SC[1:0]=11.

D5, D4 (RW1, RW0)

The two bits D5, D4 are the high/low bytes that set read/write into various counters, or locking instructions of the counter; the counter is 16-bit register, which is divided into high bytes and low bytes. If high byte (D15~D8) is selected, then low byte (D7~D0) is cleared to be 0; conversely, if low byte (D7~D0) is selected, then high byte (D15~D8) is cleared to be 0. If high and low bytes are selected at the same time, the low byte action will be done first and the high byte actions second when reading or writing data. When RW[1:0] =00, it is counter locking instruction.

D3~D1 (M2~M0)

The value of this bit determines the operating modes of various 8254 counters. The operating modes of various counters that can be set include six types.

D0 (BCD)

This bit is used to set write counter value to be hexadecimal or decimal. When D0=0, the count value written is hexadecimal. In this case, the maximum initial value of counter can be FFFFH; when D0=1, the count value written is decimal, and now the maximum initial value of counter can be 9999. As for which scale to use, it may depend on the actual situations: if the count value exceeds 9999, hexadecimal code can be used.

Article 3 PCI interface experiment software hardware basic setting

Leaper electronic company manufactures PCI interface experiment cards used in this book. The use of PCI_9052 chip of US PLX company as its core, together with 8254 timer/counter, 16V8(PLD), memory 93C46 and 74 series logic switches enable this interface card to have the functions of basic I/O cards. This interface card is designed without hardware interrupt but with many IO so that it can replace the old ISA interface experiment textbook.

With the current trend of Legacy-Free, ISA/EISA interface cards are about to fall into disuse, and the R&D of interface circuit is sure to move toward more advanced interfaces. However, since more advanced PCI interfaces are based on 32 bit, 33 MHz PCI_32 bits interfaces, this type of experiment textbook must be important in the areas related with future interface design. This article will describe in brief the experiment board used, and describe the software and hardware setting of this experiment board (the hardware portion is divided into 3 parts: assembly language, Debug Mode, and Visual C++), then describe the setting and operating methods of the 3 types of program languages.

Chapter 5 PCI_LAB/IO software setting and description

PCI_LAB/IO is the PCI I/O practice board manufactured by Leaper electronic company, the hardware portion of which is divided into two major parts: PCI_IO interface card and PCI_LAB, which are connected by a 68-pin cable. PCI_IO needs to be connected to a PCI slot, while PCI_LAB is the experiment board for this practice, used to observe the experiment result. This chapter will describe its hardware building and drive program setting, while the software setting and operation will be described in the next chapter.

5.1 PCI experiment board hardware building

This chapter will describe the installation of the PCI experiment board step by step. First, PCI-IO interface card must be installed in the PC. Figure 5-1-1 shows how the PCI-IO interface card to be used. The IO chip used is PLX 9052 chip. While Figure 5-1-2 shows the sketch of common computer motherboard chassis interface slot, containing AGP interface, PCI interface and ISA interface. Generally speaking, the AGP interface slot on the motherboard chassis is brown, PCI interface slot is white, and ISA interface slot is black. Figure 5-1-3 shows the sketch of PCI slot for this interface card, any of the PCI slots can be used to plug this interface in. Never plug into different types of slots to avoid damage to the motherboard chassis slot, and lock out the screws on the rear board to prevent the interface card from coming off. Figure 5-1-4 is the connection diagram of PCI-IO interface card and PCI-LAB experiment board, 2 devices are connected using 68-pin connecting wire; the end of which is wide at the top and narrow at the bottom, and not easy to reverse plugs. With the above steps taken, software settings can be made.

The change of modules on PCI-LAB and PCI-LAB can be made on this experiment platform; usually, according to PCI specifications, the power supply of the computer must be shut off, then can the changes of module and PCI-LAB be made, which is the safest changing method. However, both this experiment board and interface board have protection circuit, changes of PCI-LAB or modules can be made by cutting off the power supply on the PCI-LAB, and power indicator LED is below PCI-LAB power switch. To make changes, please make sure that the power supply has been cut off, otherwise software errors may occur to the PCI-LAB or PCI-IP or there may be risks of hardware being burned.

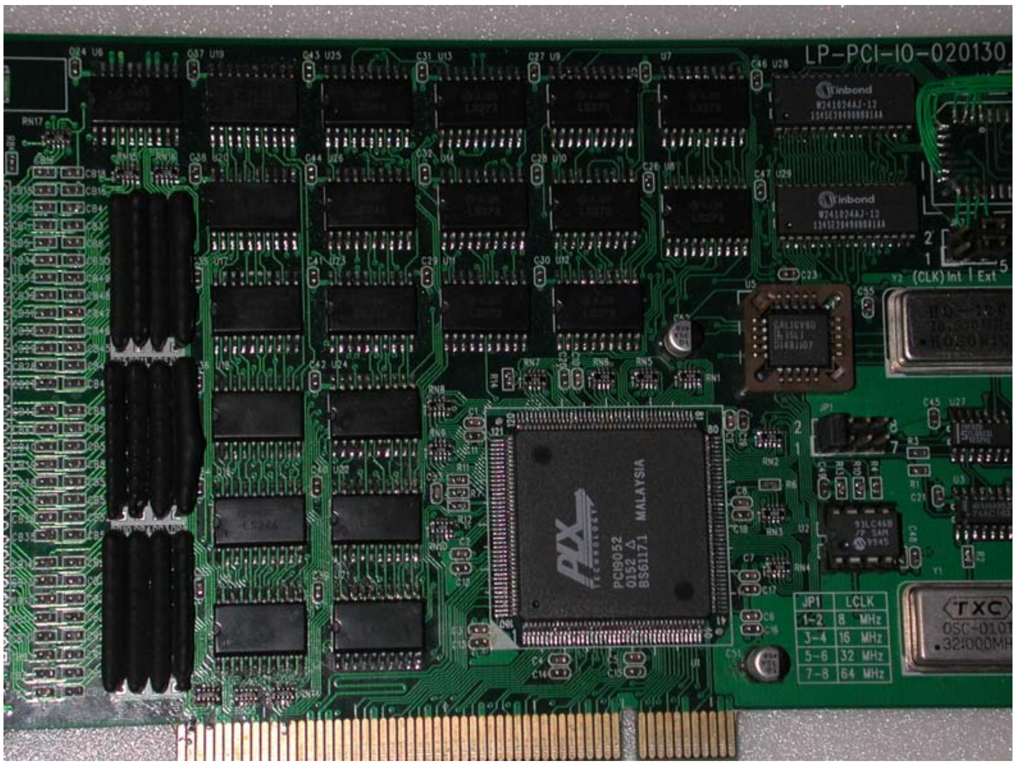


Figure 5-1-1 Leaper PCI-IO experiment board

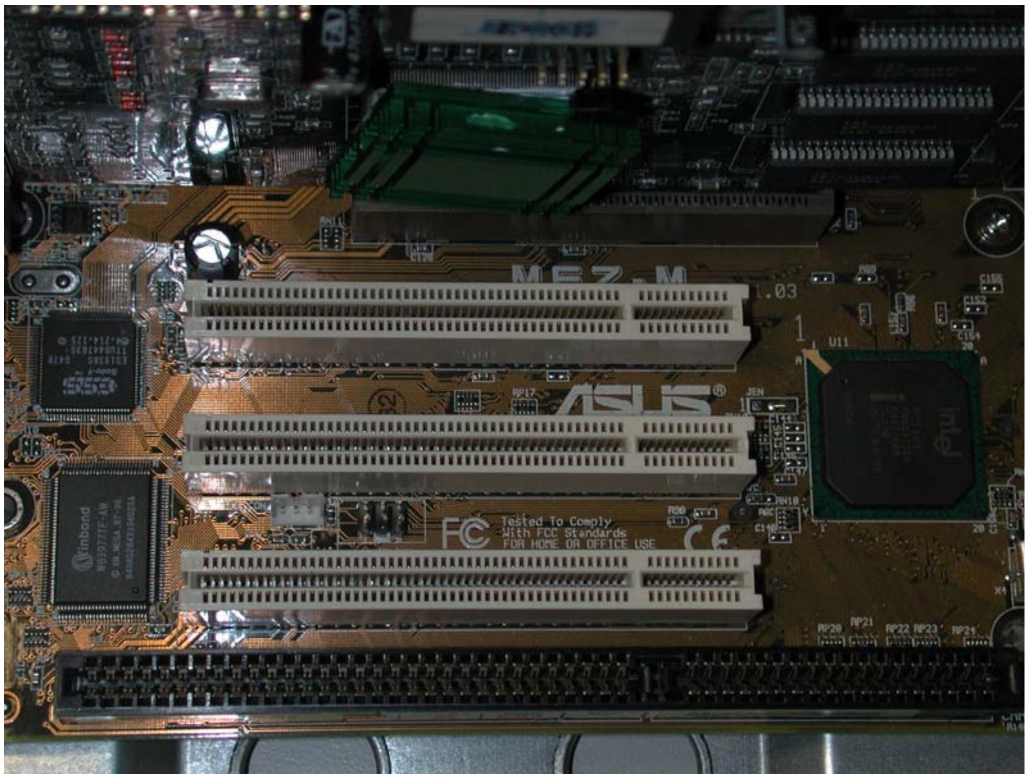


Figure 5-1-2 Interface slot sketch

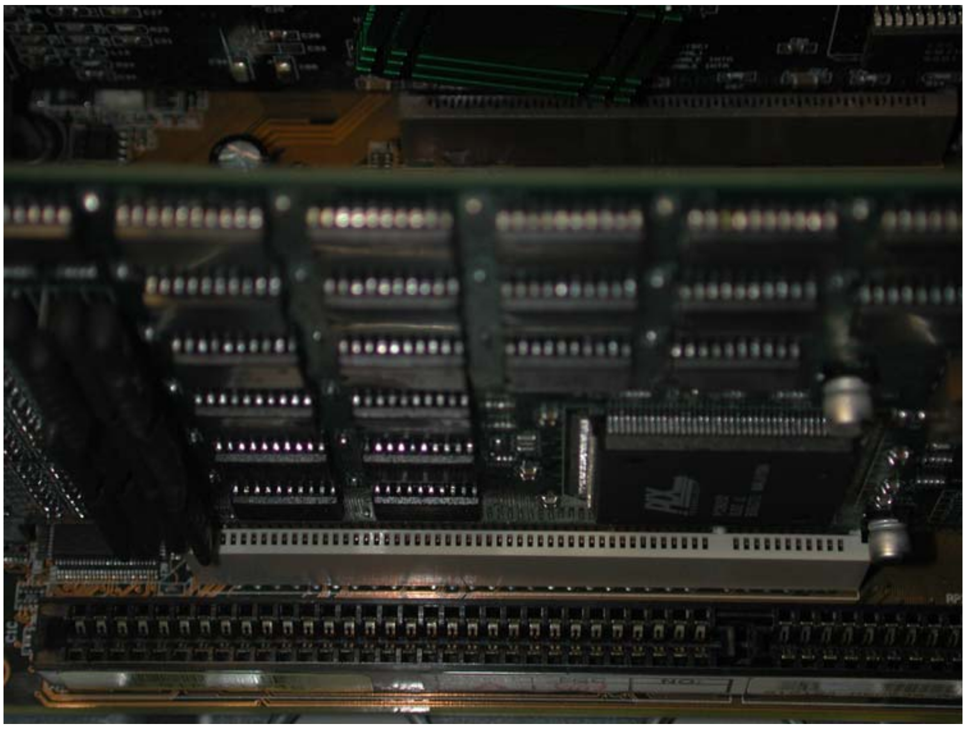


Figure 5-1-3 Installing PCI-IO diagram

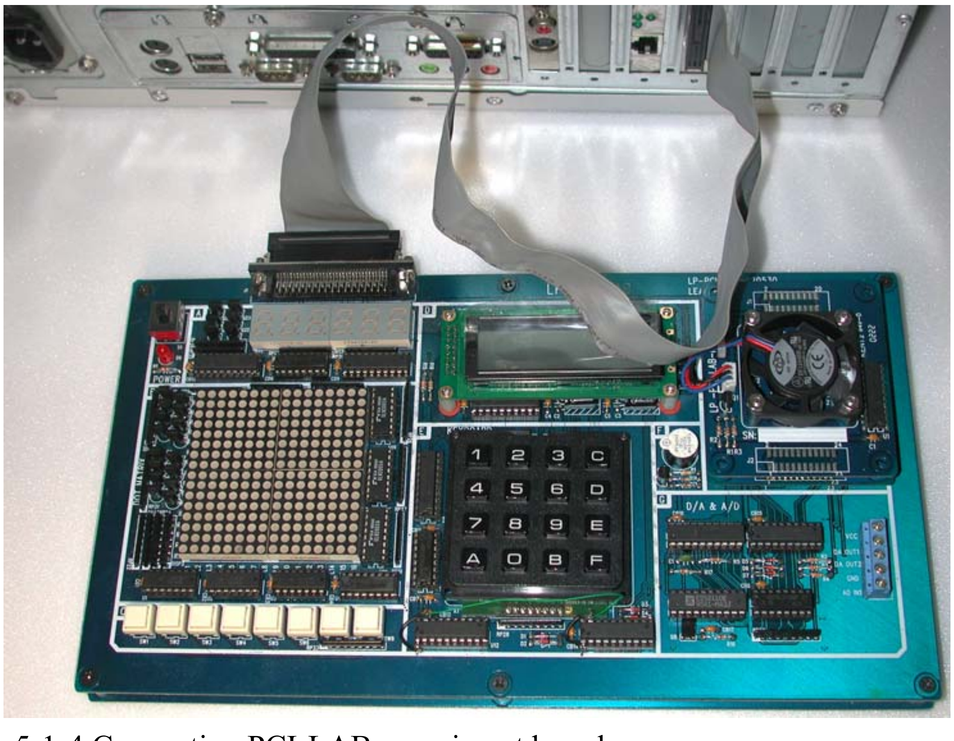


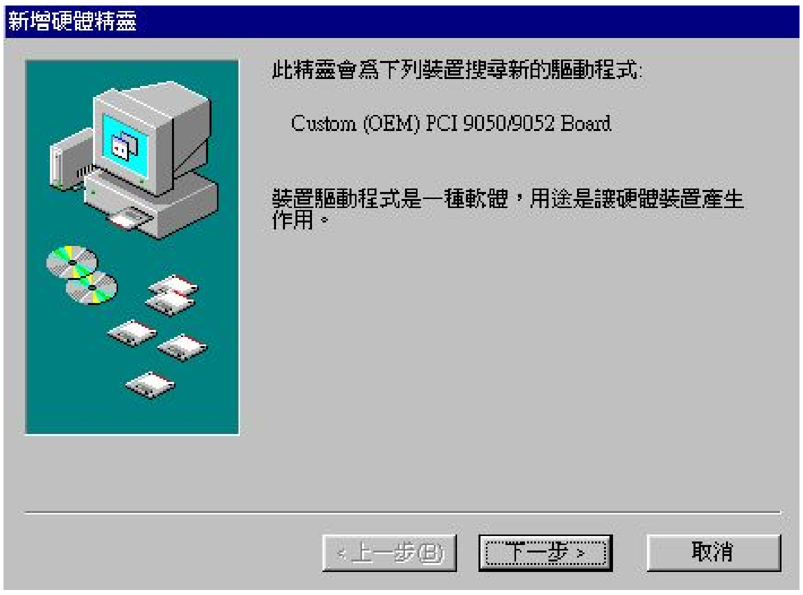
Figure 5-1-4 Connecting PCI-LAB experiment board

5.2 PCI-IO drive program setting

This experiment board is PCI-IO and PCI interface board, to install it into the PC, the drive program must be installed. Currently, the most commonly seen operating system is Windows 98/ME and Windows 2000/XP, this section will describe the process of loading the drive program in two parts:

The computer operating system for installing this experiment is Windows 98, with drive program settings as follows: after the computer is started, the system will search new devices (Custon(OEM) PCI 9050/9052 Board)as shown in Figure 5-2-1. When you are unable to see the Add New Hardware Wizard window, you can go to the control panel to select Add New hardware. If this hardware can not be detected, please shut off the computer and check whether the PCI-IO interface is plugged securely in the PCI slot, or change the slot of PCI-IO interface card, move PCI-IO interface card to another empty PCI slot, then start the computer to set the drive program. Click Next in Figure 5-2-1, then the window is shown as Figure 5-2-2, click the item suggested to be used, and click Next, then the window in Figure 5-2-3 will appear. Click the designated position and browse…\Win32\Driver\Wdm in the drive program, click Next, then the window shown in Figure 5-2-4 appears, indicating the drive program setting of PCI-IO interface board on Win 98 system has been completed.

The setting of the drive program installed in Windows 2000 operating system environment for this experiment module is as follows: when installing PCI-IO interface in computer system and after the computer is started, window shown in Figure 5-2-5 will be displayed, informing you that a new device has been detected (Custon(OEM) PCI 9050/9052 Board), meanwhile "New Add Hardware Wizard" window shown in Figure 5-2-6 will be displayed. Like the drive program loading methods under Win98 system. Select recommended options in Figure 5-2-7 window, and select "Designated Position" in Figure 5-2-8 window, browse "…\Win32\Driver\Wdm" in the drive program disk in Figure 5-2-9, while Figure 5-2-10 is the window that the installation of drive program has been completed.

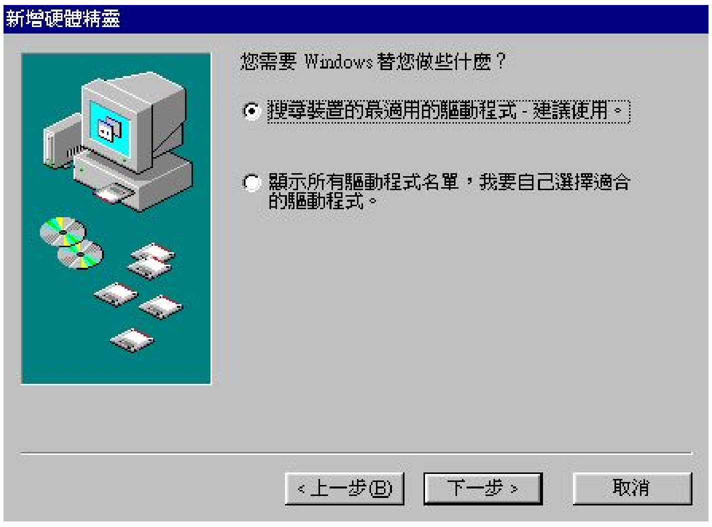

新增硬體精靈

此精靈會為下列裝置搜尋新的驅動程式:

Custom (OEM) PCI 9050/9052 Board

裝置驅動程式是一種軟體，用途是讓硬體裝置產生作用。

Figure 5-2-1 Win98 drive program setting step one

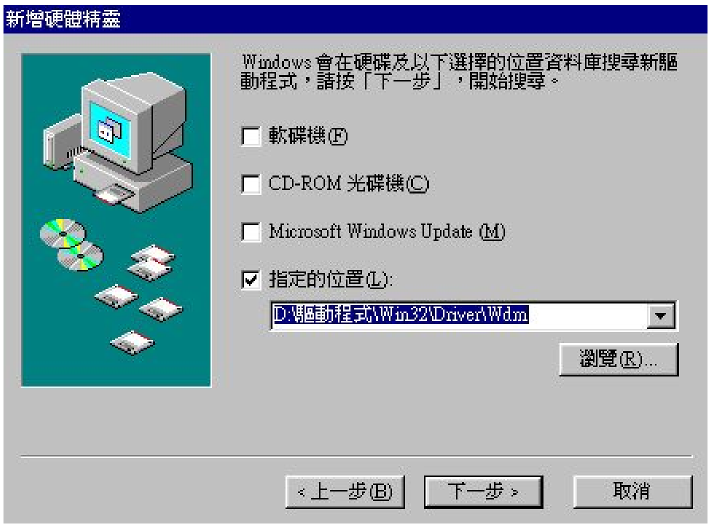

新增硬體精靈

您需要 Windows 替您做些什麼？

[✓ 搜尋裝置的最適用的驅動程式 - 建議使用。]

顯示所有驅動程式名單，我要自己選擇適合的驅動程式。

Figure 5-2-2 Win98 drive program setting step two


新增硬體精靈

Windows 會在硬碟及以下選擇的位置資料庫搜尋新驅動程式，請按「下一步」，開始搜尋。

☐ 軟碟機(F)

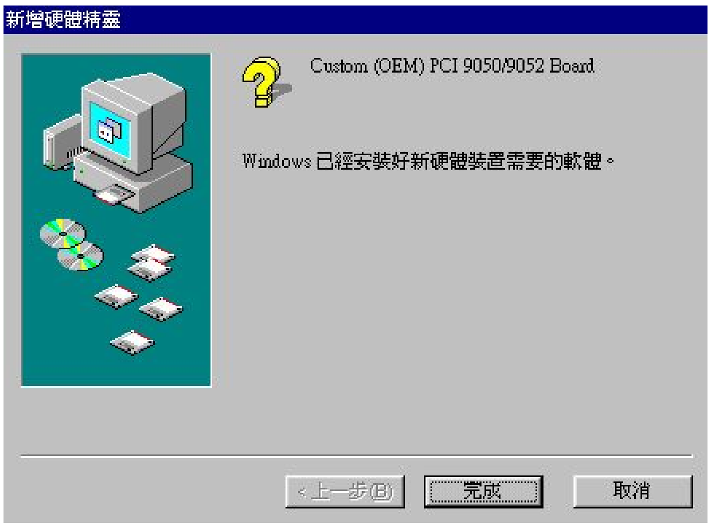
☐ CD-ROM 光碟機(C)

☐ Microsoft Windows Update (M)

☑ 指定的位置(L):

D:\驅動程式\Win32\Driver\Wdm

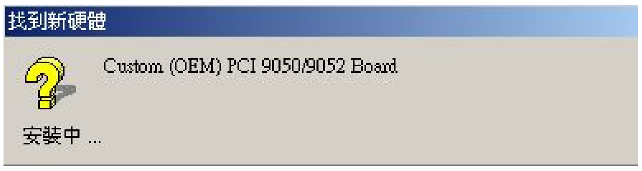
Figure 5-2-3 Win98 drive program setting step three


新增硬體精靈

Custom (OEM) PCI 9050/9052 Board

Windows 已經安裝好新硬體裝置需要的軟體。

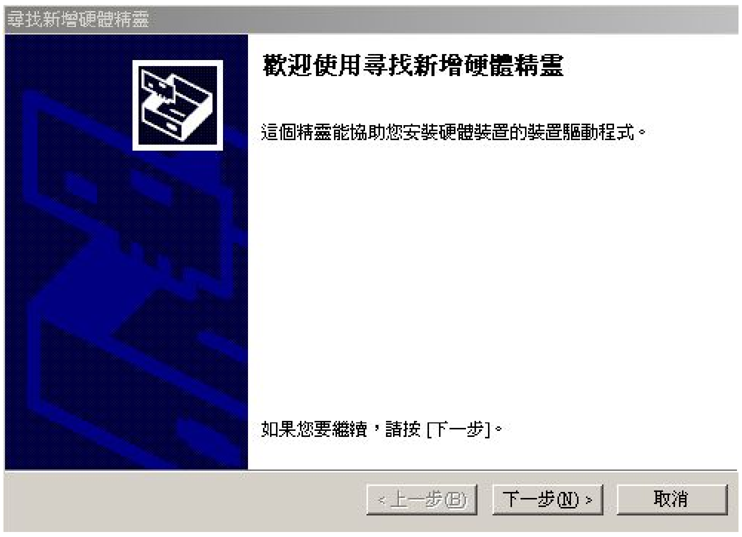
Figure 5-2-4 Win98 drive program setting step four


找到新硬體

Custom (OEM) PCI 9050/9052 Board

安裝中 ...

Figure 5-2-5 Win2k drive program setting step one

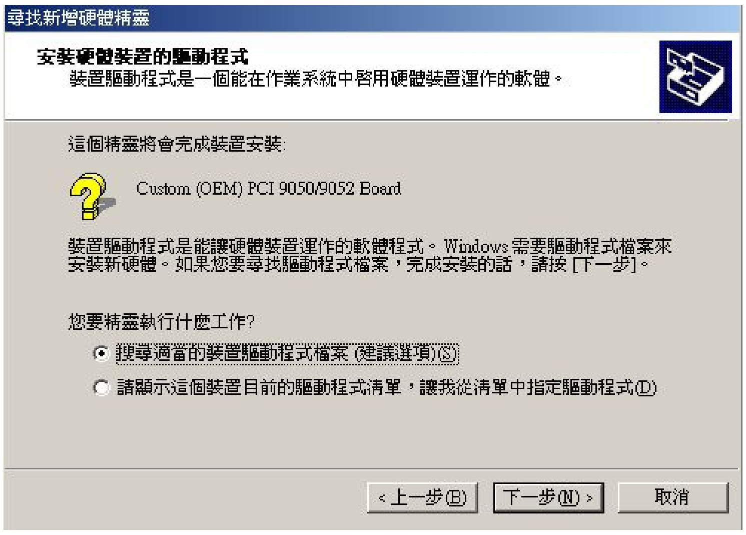

尋找新增硬體精靈

歡迎使用尋找新增硬體精靈

這個精靈能協助您安裝硬體裝置的裝置驅動程式。

如果您要繼續，請按 [下一步]。

Figure 5-2-6 Win2k drive program setting step two


尋找新增硬體精靈

安裝硬體裝置的驅動程式
裝置驅動程式是一個能在作業系統中啓用硬體裝置運作的軟體。

這個精靈將會完成裝置安裝:

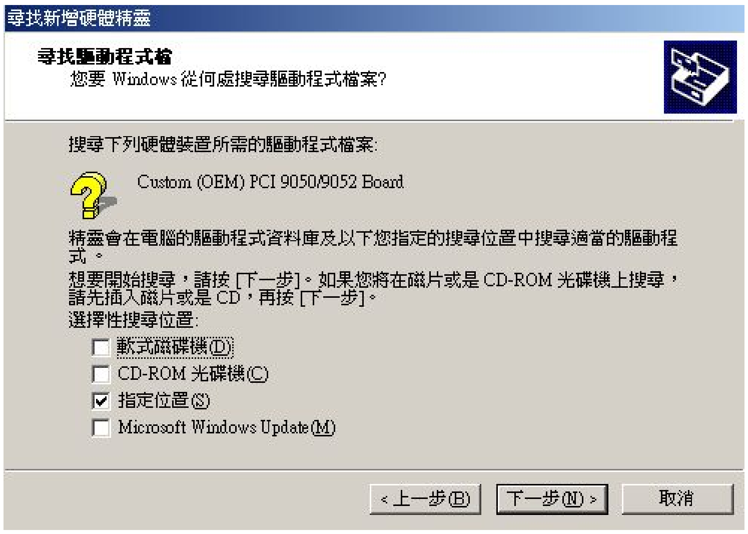
Custom (OEM) PCI 9050/9052 Board

裝置驅動程式是能讓硬體裝置運作的軟體程式。Windows 需要驅動程式檔案來安裝新硬體。如果您要尋找驅動程式檔案，完成安裝的話，請按 [下一步]。

您要精靈執行什麼工作?

[✓ 搜尋適當的裝置驅動程式檔案 (建議選項)(S)]
請顯示這個裝置目前的驅動程式清單，讓我從清單中指定驅動程式(D)

Figure 5-2-7 Win2k drive program setting step three


尋找新增硬體精靈

尋找驅動程式檔
您要 Windows 從何處搜尋驅動程式檔案?

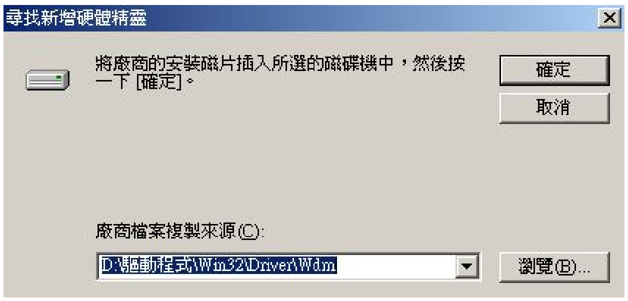
搜尋下列硬體裝置所需的驅動程式檔案:

Custom (OEM) PCI 9050/9052 Board

精靈會在電腦的驅動程式資料庫及以下您指定的搜尋位置中搜尋適當的驅動程式。
想要開始搜尋，請按 [下一步]。如果您將在磁片或是 CD-ROM 光碟機上搜尋，請先插入磁片或是 CD，再按 [下一步]。
選擇性搜尋位置:

- ☐ 軟式磁碟機(D)
- ☐ CD-ROM 光碟機(C)
- ☑ 指定位置(S)
- ☐ Microsoft Windows Update(M)

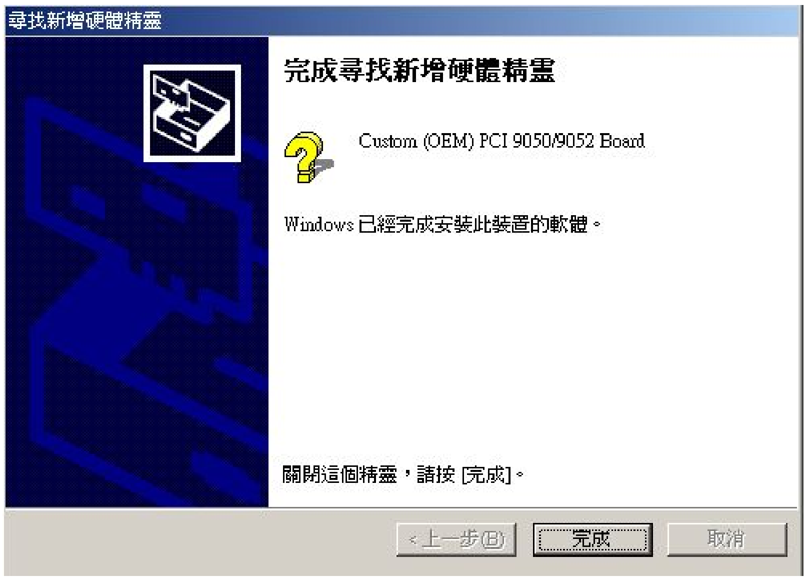
Figure 5-2-8 Win2k drive program setting step four


尋找新增硬體精靈

將廠商的安裝磁片插入所選的磁碟機中，然後按一下 [確定]。

廠商檔案複製來源(C):
D:\驅動程式\Win32\Driver\Wdm

Figure 5-2-9 Win2k drive program setting step five


尋找新增硬體精靈

完成尋找新增硬體精靈

Custom (OEM) PCI 9050/9052 Board

Windows 已經完成安裝此裝置的軟體。

關閉這個精靈，請按 [完成]。

Figure 5-2-10 Win2k drive program setting step six

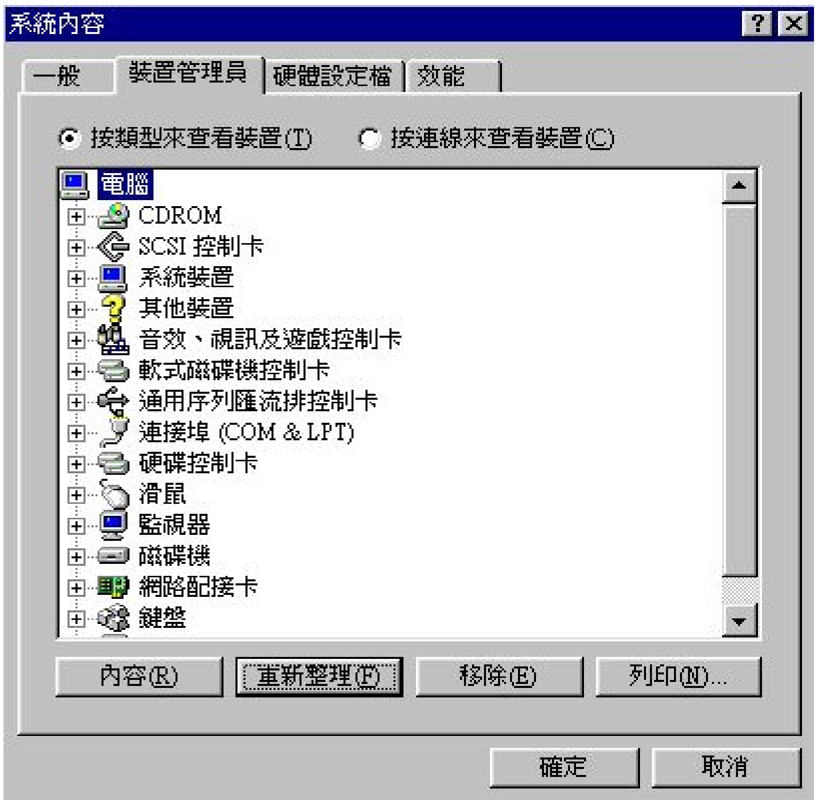
Note: After installing the drive program, the computer must be restarted. After restarting the computer, the read should confirm under Win98 operating system whether there is PLXAPI.DLL dynamic linking document in WINDOWS\SYSTEM and WINDOWS\SYSTEM32 folder and confirm under Win2k system whether there is PLXAPI.DLL dynamic linking document in WINNT\SYSTEM and WINNT\SYSTEM32 folder. Lack of this document will cause VC/C++ program to be unable to execute the program and output the results on PCI-IO interface card and PCI-LAB. In this case, the reader himself can copy this file into the above position from the disk attached with this drive program.

5.3 IO address of PCI-IO interface card

This PCI-IO interface card, after load the drive program, can restart the system. After the system is restarted, you can observe the IO base position of this interface card, because this interface card and experiment board both use IO port method. Debug mode and Assembly programs can be written only after obtaining the IO address of this interface card, so this part of the information is very important. Also, this section is divided into two parts: Win98 and Win2k, describing how to observe the IO address of this PCI-IO interface card.

To observe the IO port of this interface card in Win98 operating system environment, by Start\Setting\Console\system or clicking “My computer” in the desktop, window shown in Figure 5-3-1 appears; by clicking sub-page device administrator in the window, a other devices can be found. After clicking this device, window shown in Figure 5-3-2 appears ,and click sub-page resource in the window, window shown in Figure 5-3-3 will appear; in which the input/output scope shown ranges from 1000h~10FFh, 1488h~148Bh and 1880h~18FFh. These 3 groups output scope listed are the IO ports of this interface.

To observe the IO port of this interface card in Win2k operating system environment, by Start\Setting\Console\system or clicking “My computer” in the desktop, window shown in Figure 5-3-4 appears. Like the steps in Win98, select Device administrator Other devices as shown in Figure 5-3-5, 5-3-6 and 5-3-7 in sequence, and I/O scope between C000h~C07Fh, C400h~C403h and C800h~C8FFh may be found in sub-page Resource. The three groups of output scopes listed are the IO port of this interface card, the address of IO scope from 00h to FFh is the IO port addressed used by this experiment board.



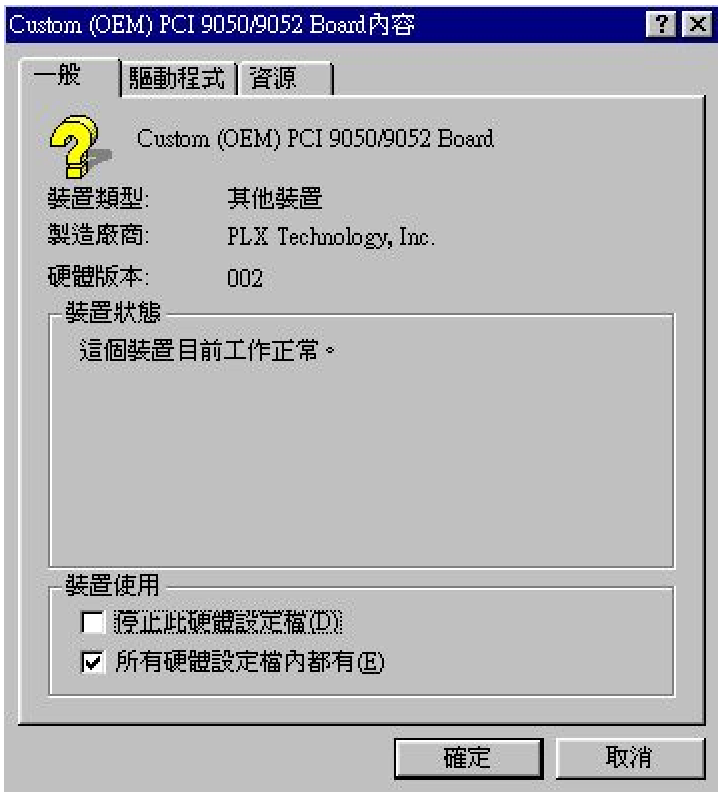
系統內容

一般 | 裝置管理員 | 硬體設定檔 | 效能

[✓ 按類型來查看裝置(T)] 按連線來查看裝置(C)

- 電腦
  - CDROM
  - SCSI 控制卡
  - 系統裝置
  - 其他裝置
  - 音效、視訊及遊戲控制卡
  - 軟式磁碟機控制卡
  - 通用序列匯流排控制卡
  - 連接埠 (COM & LPT)
  - 硬碟控制卡
  - 滑鼠
  - 監視器
  - 磁碟機
  - 網路配接卡
  - 鍵盤

Figure 5-3-1 IO address read one of PCI-IO (Win98)



Custom (OEM) PCI 9050/9052 Board內容

一般 | 驅動程式 | 資源

Custom (OEM) PCI 9050/9052 Board

裝置類型:	其他裝置
製造廠商:	PLX Technology, Inc.
硬體版本:	002

裝置狀態

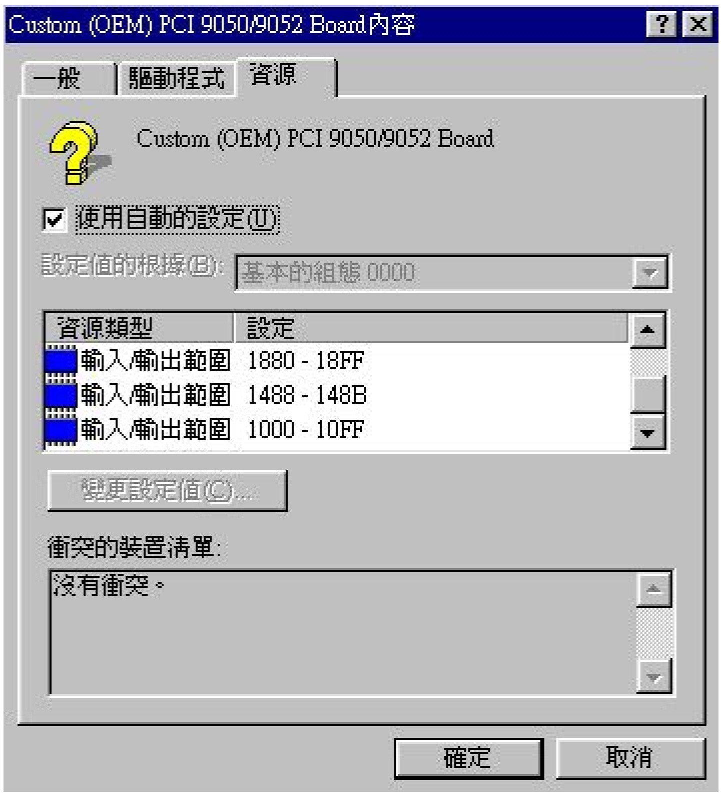
這個裝置目前工作正常。

裝置使用

☐ 停止此硬體設定檔(D)

☑ 所有硬體設定檔內都有(E)

Figure 5-3-2 IO address read two of PCI-IO (Win98)



Custom (OEM) PCI 9050/9052 Board內容

Custom (OEM) PCI 9050/9052 Board

☑ 使用自動的設定(U)

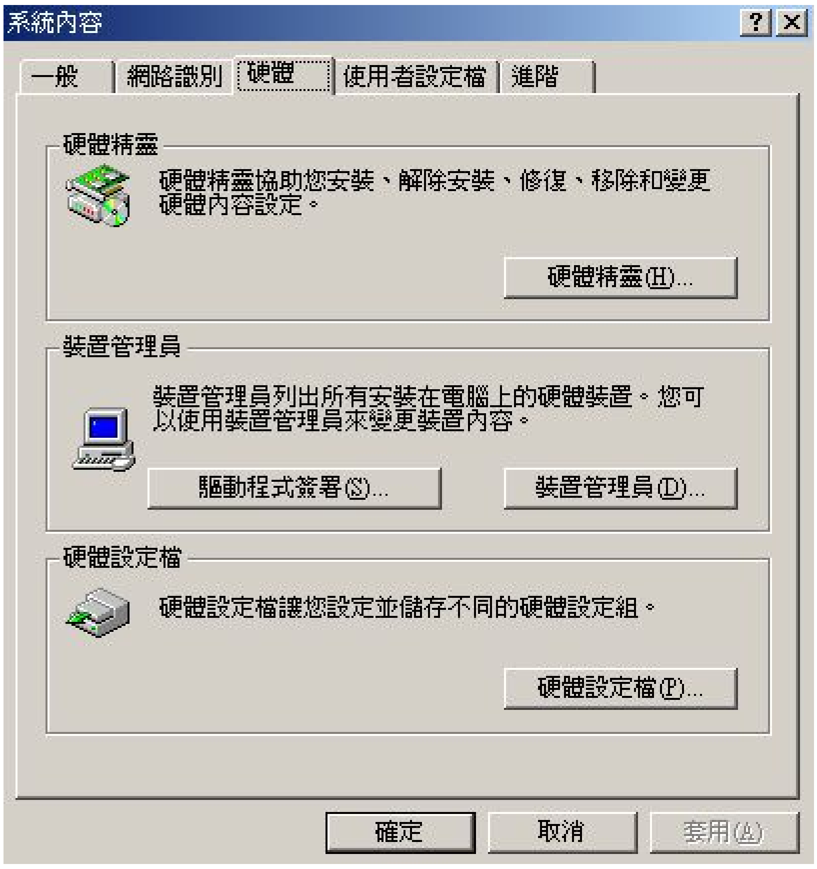
設定值的根據(B): 基本的組態 0000

資源類型	設定
輸入/輸出範圍	1880 - 18FF
輸入/輸出範圍	1488 - 148B
輸入/輸出範圍	1000 - 10FF

衝突的裝置清單:

沒有衝突。

Figure 5-3-3　IO address read three of PCI-IO (Win98)



系統內容

硬體精靈

硬體精靈協助您安裝、解除安裝、修復、移除和變更硬體內容設定。

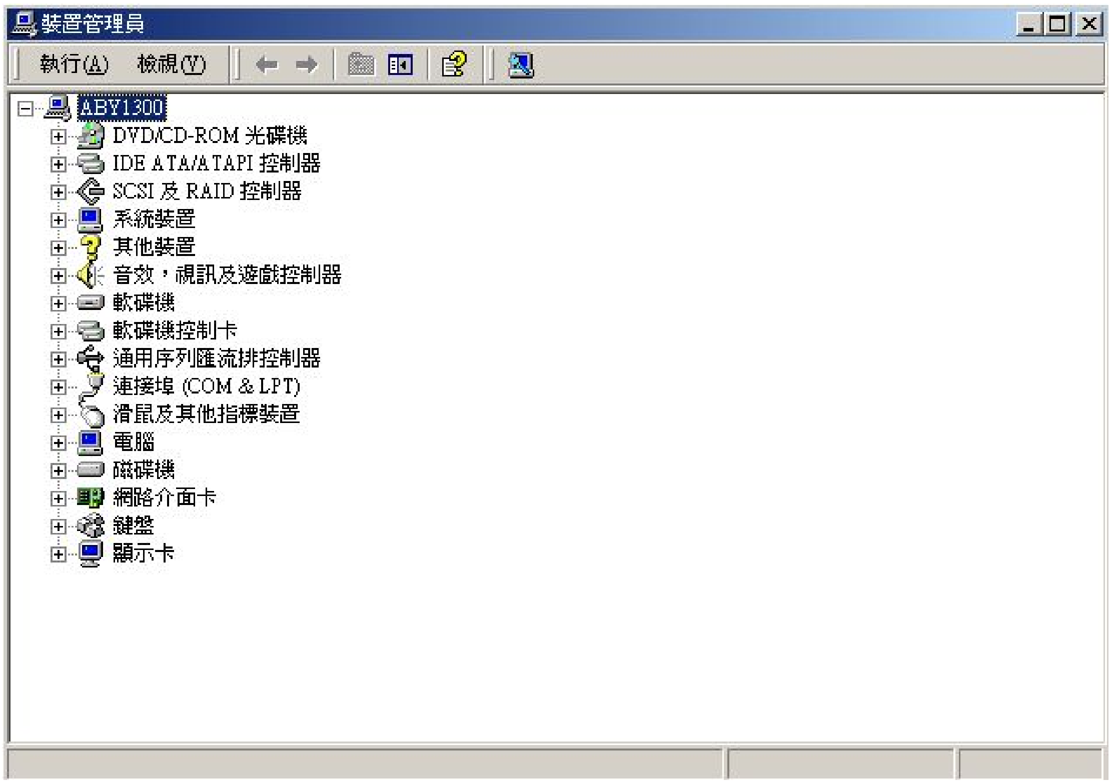
裝置管理員

裝置管理員列出所有安裝在電腦上的硬體裝置。您可以使用裝置管理員來變更裝置內容。

硬體設定檔

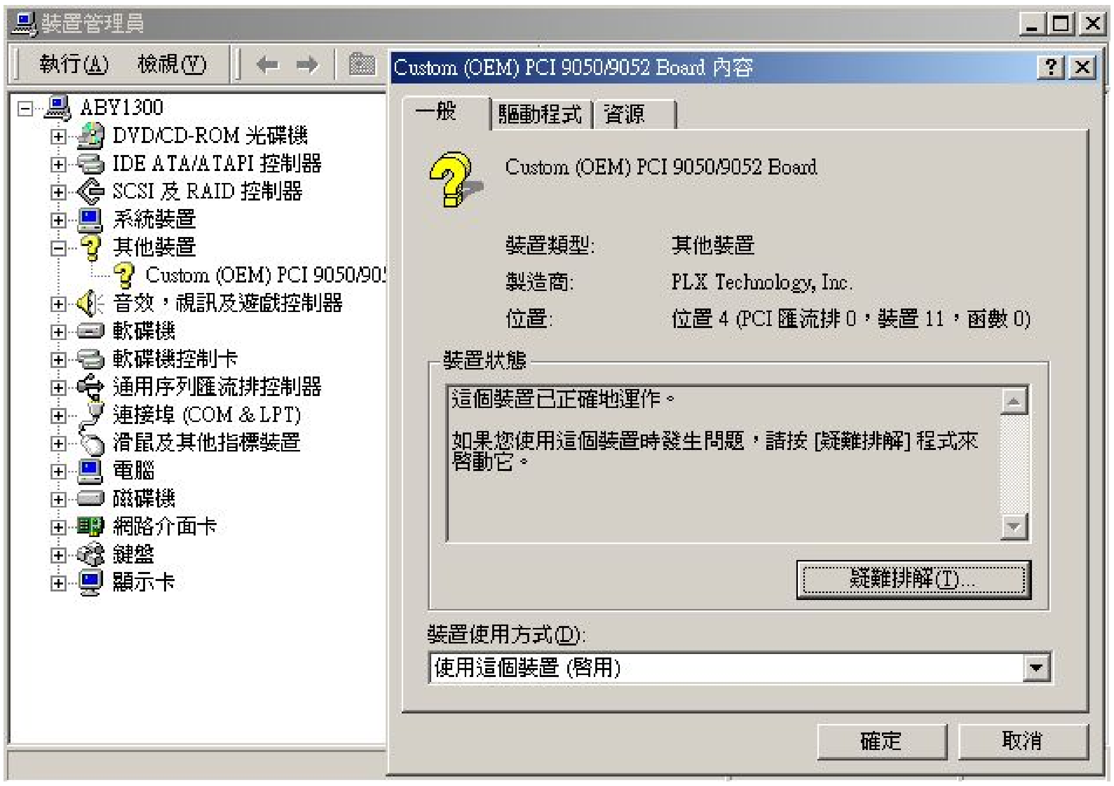
硬體設定檔讓您設定並儲存不同的硬體設定組。

Figure 5-3-4　IO address read one of PCI-IO (Win2k)



ABY1300
- DVD/CD-ROM 光碟機
- IDE ATA/ATAPI 控制器
- SCSI 及 RAID 控制器
- 系統裝置
- 其他裝置
- 音效，視訊及遊戲控制器
- 軟碟機
- 軟碟機控制卡
- 通用序列匯流排控制器
- 連接埠 (COM & LPT)
- 滑鼠及其他指標裝置
- 電腦
- 磁碟機
- 網路介面卡
- 鍵盤
- 顯示卡

Figure 5-3-5　IO address read two of PCI-IO (Win2k)



ABY1300
- DVD/CD-ROM 光碟機
- IDE ATA/ATAPI 控制器
- SCSI 及 RAID 控制器
- 系統裝置
- 其他裝置
  - Custom (OEM) PCI 9050/905
- 音效，視訊及遊戲控制器
- 軟碟機
- 軟碟機控制卡
- 通用序列匯流排控制器
- 連接埠 (COM & LPT)
- 滑鼠及其他指標裝置
- 電腦
- 磁碟機
- 網路介面卡
- 鍵盤
- 顯示卡

Custom (OEM) PCI 9050/9052 Board

裝置類型:	其他裝置
製造商:	PLX Technology, Inc.
位置:	位置 4 (PCI 匯流排 0，裝置 11，函數 0)

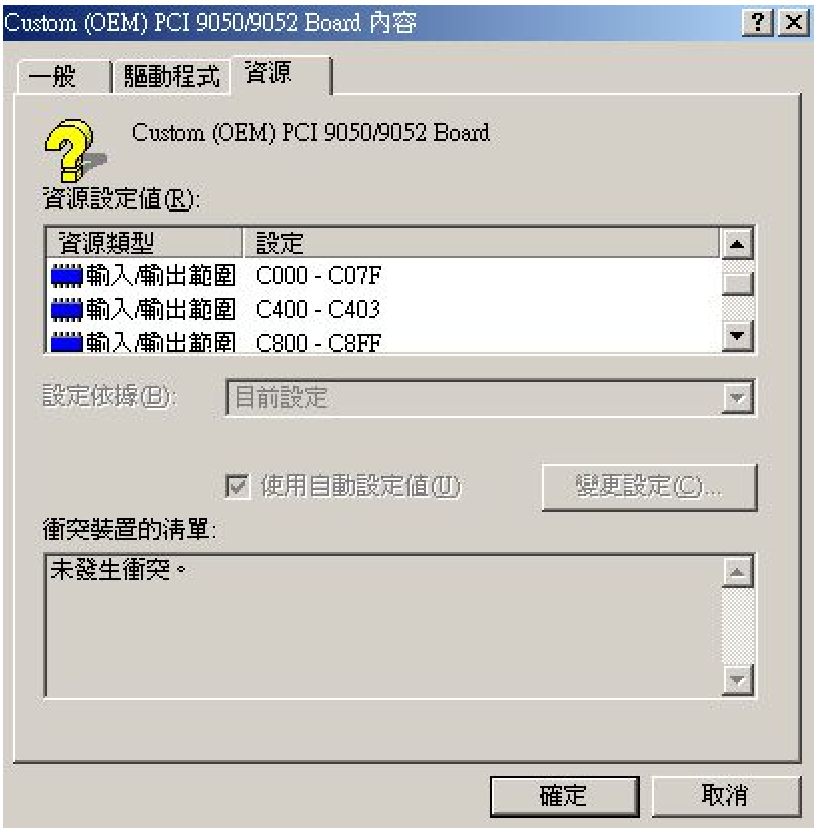
裝置狀態

這個裝置已正確地運作。

如果您使用這個裝置時發生問題，請按 [疑難排解] 程式來啓動它。

裝置使用方式(D):
使用這個裝置 (啓用)

Figure 5-3-6　IO address read three of PCI-IO (Win2k)



Custom (OEM) PCI 9050/9052 Board 內容

一般 | 驅動程式 | 資源

Custom (OEM) PCI 9050/9052 Board

資源設定值(R):

資源類型	設定
輸入/輸出範圍	C000 - C07F
輸入/輸出範圍	C400 - C403
輸入/輸出範圍	C800 - C8FF

設定依據(B): 目前設定

☑ 使用自動設定值(U)

變更設定(C)...

衝突裝置的清單:

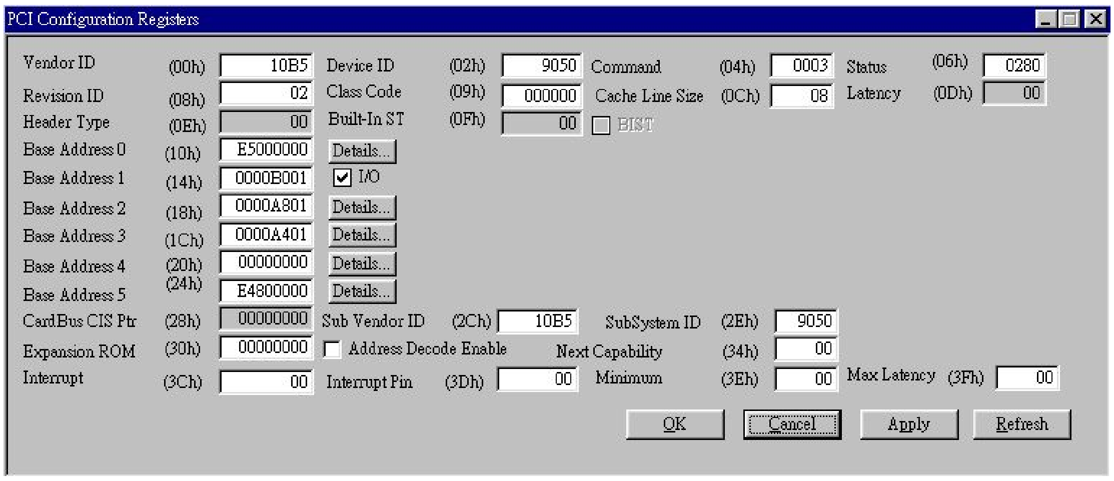
未發生衝突。

確定 取消

Figure 5-3-7 IO address read four of PCI-IO (Win2k)

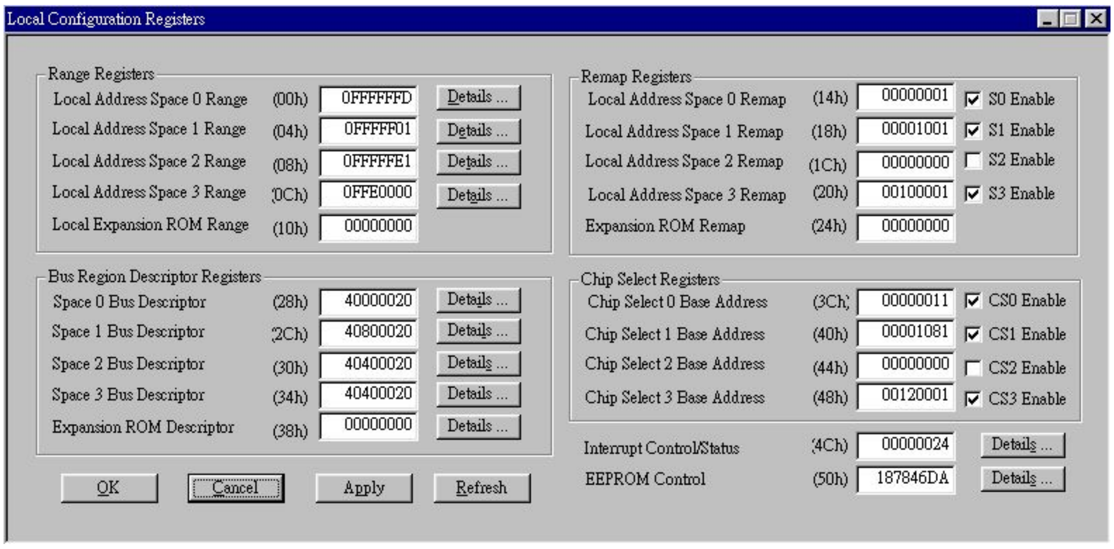
5.4 Configuration cache

As with the interface card PCI-IO of this experiment, software PLXMON of PLX Company can be used to configure read/write action of cache. This book does not provide this software, only fetches its cache values for the reader's references. Figure 5-4-1 is PCI configuration register. And Figure 5-4-2 is local configuration register. Readers are specially cautioned not to change the internal values inside the configuration cache willfully, otherwise it may cause the computer system or interface card to be unable to act properly. However, we will not describe this in detail.



Vendor ID	(00h)	10B5	Device ID	(02h)	9050	Command	(04h)	0003	Status	(06h)	0280
Revision ID	(08h)	02	Class Code	(09h)	000000	Cache Line Size	(0Ch)	08	Latency	(0Dh)	00
Header Type	(0Eh)	00	Built-In ST	(0Fh)	00	☐ BIST					
Base Address 0	(10h)	E5000000	Details...								
Base Address 1	(14h)	0000B001	☑ I/O								
Base Address 2	(18h)	0000A801	Details...								
Base Address 3	(1Ch)	0000A401	Details...								
Base Address 4	(20h)	00000000	Details...								
Base Address 5	(24h)	E4800000	Details...								
CardBus CIS Ptr	(28h)	00000000	Sub Vendor ID	(2Ch)	10B5	SubSystem ID	(2Eh)	9050			
Expansion ROM	(30h)	00000000	☐ Address Decode Enable			Next Capability	(34h)	00			
Interrupt	(3Ch)	00	Interrupt Pin	(3Dh)	00	Minimum	(3Eh)	00	Max Latency	(3Fh)	00

Figure 5-4-1 PCI configuration cache (PCI-IO interface card)



Range Registers			
Local Address Space 0 Range	(00h)	0FFFFFFD	Details ...
Local Address Space 1 Range	(04h)	0FFFFF01	Details ...
Local Address Space 2 Range	(08h)	0FFFFFE1	Details ...
Local Address Space 3 Range	(0Ch)	0FFE0000	Details ...
Local Expansion ROM Range	(10h)	00000000	

Remap Registers			
Local Address Space 0 Remap	(14h)	00000001	☑ S0 Enable
Local Address Space 1 Remap	(18h)	00001001	☑ S1 Enable
Local Address Space 2 Remap	(1Ch)	00000000	☐ S2 Enable
Local Address Space 3 Remap	(20h)	00100001	☑ S3 Enable
Expansion ROM Remap	(24h)	00000000	

Bus Region Descriptor Registers			
Space 0 Bus Descriptor	(28h)	40000020	Details ...
Space 1 Bus Descriptor	(2Ch)	40800020	Details ...
Space 2 Bus Descriptor	(30h)	40400020	Details ...
Space 3 Bus Descriptor	(34h)	40400020	Details ...
Expansion ROM Descriptor	(38h)	00000000	Details ...

Chip Select Registers			
Chip Select 0 Base Address	(3Ch)	00000011	☑ CS0 Enable
Chip Select 1 Base Address	(40h)	00001081	☑ CS1 Enable
Chip Select 2 Base Address	(44h)	00000000	☐ CS2 Enable
Chip Select 3 Base Address	(48h)	00120001	☑ CS3 Enable
Interrupt Control/Status	(4Ch)	00000024	Details ...
EEPROM Control	(50h)	187846DA	Details ...

Figure 5-4-2 Local configuration register (PCI-IO interface card)

Description:

(00h)Vendor ID = 10B5 h = 0001 0000 1011 0101 b PLX Technology

(02h)Device ID = 9050 h = 1001 0000 0101 0000 b 9050/9052 OEM board

(04h)Command = 0003 h = 0000 0000 0000 0011 b with Memory and IO space

(06h)Status = 0280 h = 0000 0010 0100 0000 b with Fast Back-to-Back ability

(08h)Revision = 02 h = 0000 0010 b Revision No of vendor product

(09h)Class Code = 000000 h All non-VGA device before class code definition

(0Ch)Cache Line = 08 h 32 bytes(8 dword)cache lines

(0Eh)Latency = 00 h PCI Target device need not be set

(0Dh)Header Type = 00 h = 0000 0000 b Single function device

(0Fh)BIST = 00 h = 0000 0000 b disable BIST

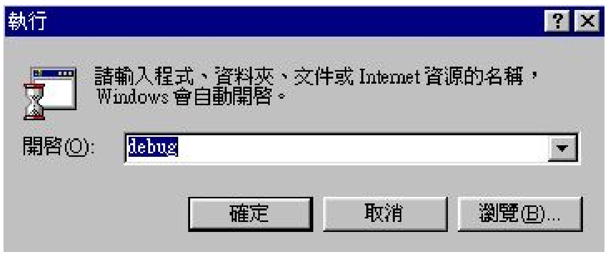
(10h)Base Address 0 = E500 0000 h

= 1110 0101 0000 0000 0000 0000 0000 0000 b Memory Base

(14h)Base Address 1 = 0000 B001 h
= 0000 0000 0000 0000 1011 0000 0000 0001 b IO Base
(18h)Base Address 2 = 0000 A801 h
= 0000 1010 1000 0000 0001 b IO Base
(1Ch)Base Address 3 = 0000 A401 h
= 0000 1010 0100 0000 0001 b IO Base
(20h)Base Address 4 = 0000 h
= 0000 b Memory Base
(24h)Base Address 5 = E480 0000 h
= 1110 0100 1000 0000 b Memory Base
(28h)Card Bus CIS = 0000 h undefined
(2Ch)Sub Vendor ID = 10B5 h PLX Technology
(2Eh)Sub System ID = 9050 h 9050/9052 Chip
(30h)Expansion Rom = 0000 0000 h no definition
(34h)Next Capability = 00 h without new functions
(3Ch)Interrupt = 00 h No hardware interrupt(engineering board)
(3Dh)Interrupt Pin = 00 h No correspondence(engineering board)
(3Eh)Min Latency = 00 h time slice request
(3Fh)Max Latency = 00 h Priority level request

5.5 Use PCI-IO in Debug mode

To use PCI-IO interface cards in Debug mode, no additional setting is required, as long as the basic hardware settings are in the above 4 sections, Debug mode program can be written. The process to open Debug mode is: click Start\execute, as is shown in Figure 5-5-1. Click Yes after entering Debug, the window of Debug mode will appear, as is shown in Figure 5-5-2; the program can be written in the window. Debug mode instructions are discussed in chapter four.


執行

請輸入程式、資料夾、文件或 Internet 資源的名稱，Windows 會自動開啓。

開啓(O): debug

Figure 5-5-1 Open Debug mode

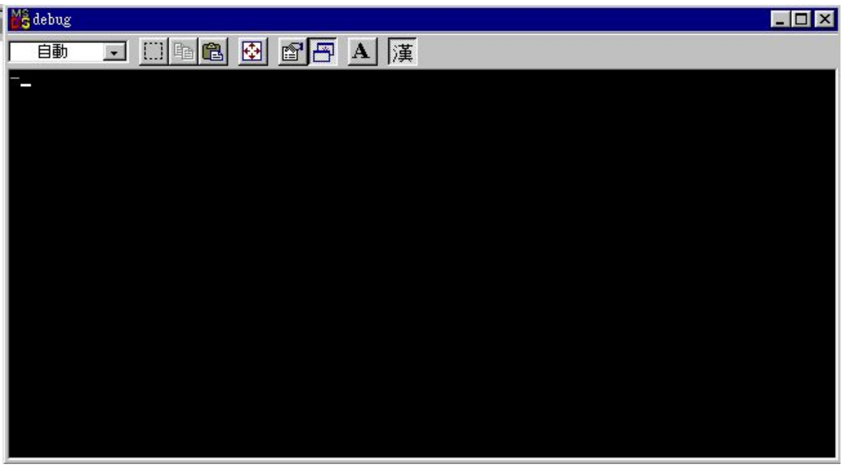


Figure 5-5-2 Debug mode working window

5.6 MASM assembly language setting

MASM is the X86 assembly language designed by Microsoft, which is shortened from Macro Assembly; the editions that can be used for PCI-IO interface cards include: MASM 6.11, MASM 6.14, MASM 6.15, etc. Form the disk attached with this book, assembly language compiler folder, or downloading MASM compiler online, they can be used by unzipping or copying into a fixed folder.

This folder is put in BIN folder in C disk, the assembly language can be compiled by using Notebook; the storage format can be .txt file, if the file name of this program is C:\testasm\test.txt, then it should be in Testasm folder; to assembly this language, use "Start", "Program", "MS-DOS mode", MS-DOS mode displays "C:\WINDOWS>", key in "cd\bin", after changing to MASM working folder, assembly language can be started to be assembled. Figure 5-6-1 shows MASM folder. Using masm C:\testasm\test.txt instruction, "test.obj" will be generated in the folder. ".obj" is an Object file, namely mechanical language file. "link C:\testasm\test.txt" needs to be used to produce a "test.exe" execute file. Usually, "test.obj" and "test.exe" file can be generated by using "ml C:\testasm\test" File.

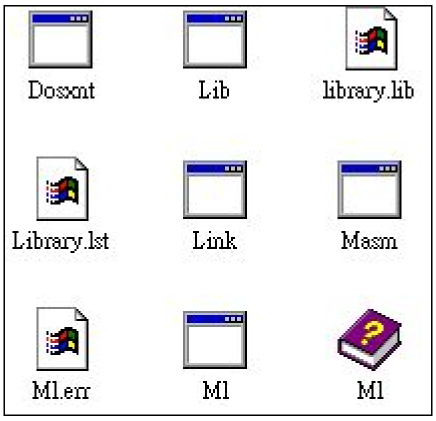
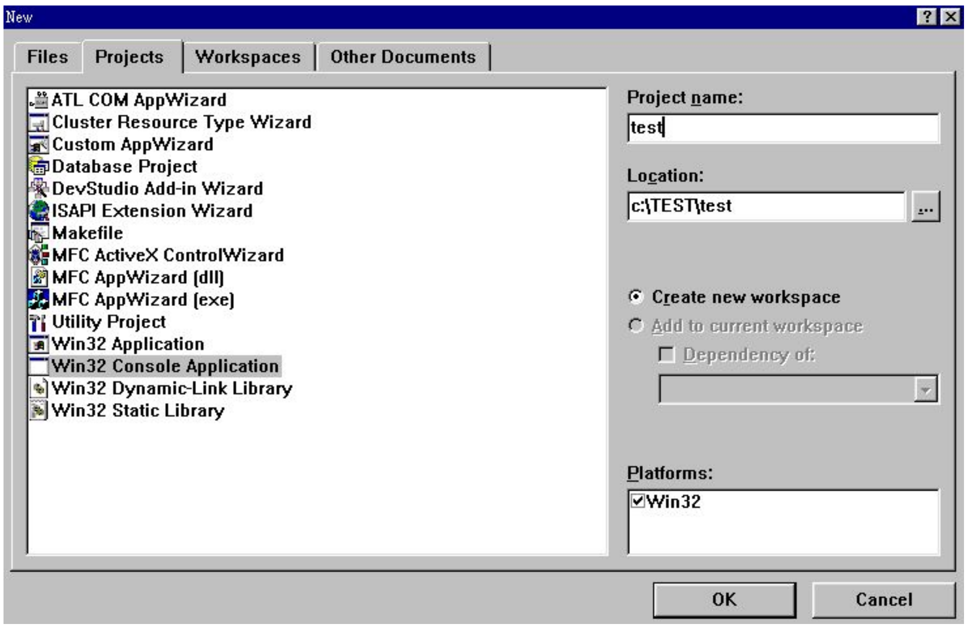


Figure 5-6-1 MASM folders

5.7 Visual C/C++ standard original setting

Visual C/C++ program language is the C/C++ program language issued by Microsoft, there are the following basic setting for using this program language to write programs and using PCI-IO interface card, and notebook can be used to edit in programming, or to write during the process of compiling programs. First copy Include and Win32folder into Drive program folder in drive program disk, then proceed to set.

The environment setting used in this compiling program is as follows, first a new "Projects" must be opened, as is shown in Figure 5-7-1, and define it as test here. This Project is in C:\TEST folder, and select Win32 Console Application, following the windows in Figures 5-7-2 and 5-7-3; file window will appear, as shown in Figure 5-7-4. Figure 5-7-5 is adding the required program edit file; Figure 5-7-6 is adding PlxInit.c to Source File. Figure 5-7-7 adds PlxApi.Lib to Test File, and sets PCI_CODE and LITTLE_ENDIAN in the setting of Project. Figure 5-7-8 and 5-7-9 show adding Include File and Library file to sub-page Directories in Option window in Tool respectively. You can start to compile your own VC/C++ program after completing the above settings.



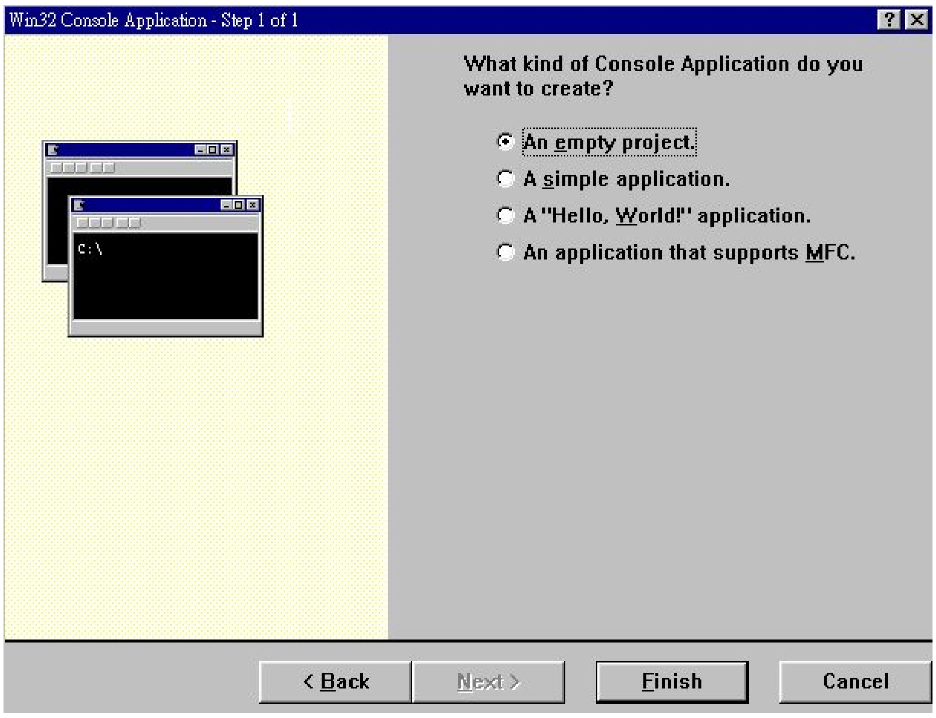
Project name: test

Location: c:\TEST\test

(● Create new workspace)

Platforms: ☑Win32

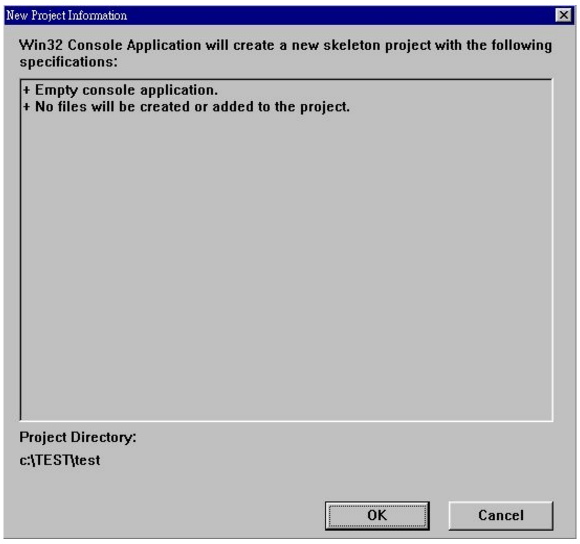
Figure 5-7-1 Open new Projects



What kind of Console Application do you want to create?

[✓ An empty project.]
A simple application.
A "Hello, World!" application.
An application that supports MFC.

Figure 5-7-2 Select empty Project



Win32 Console Application will create a new skeleton project with the following specifications:

+ Empty console application.
+ No files will be created or added to the project.

Project Directory:
c:\TEST\test

Figure 5-7-3 Results after selecting

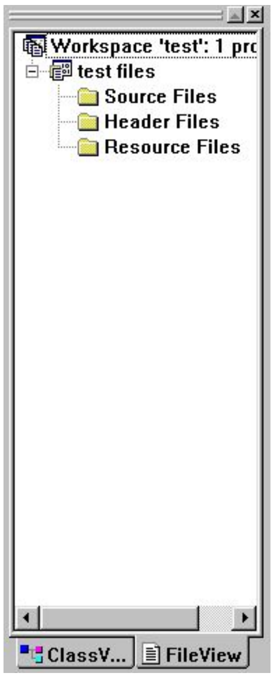
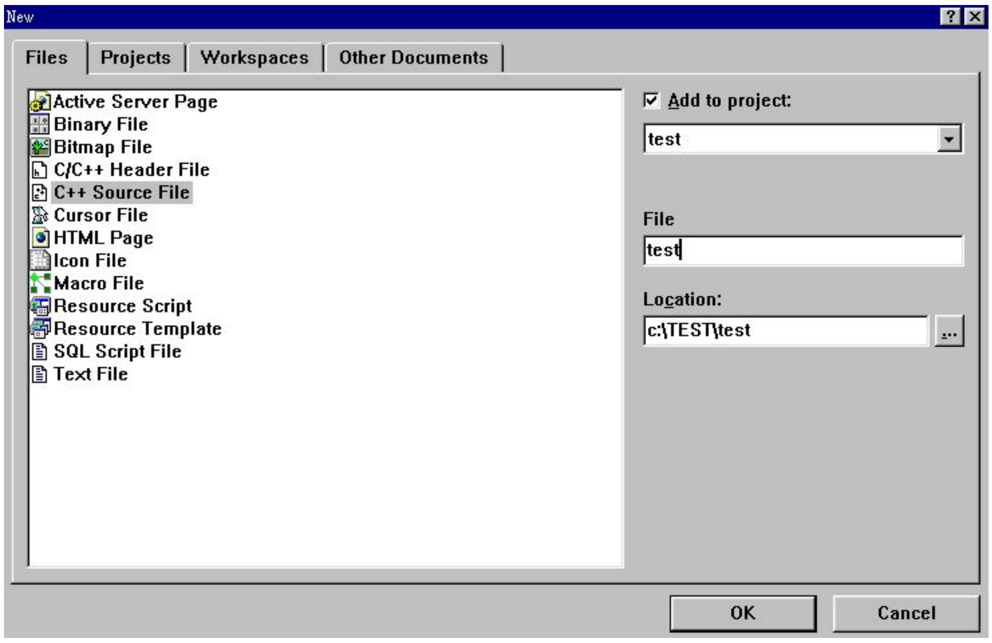


Figure 5-7-4 File window of Project

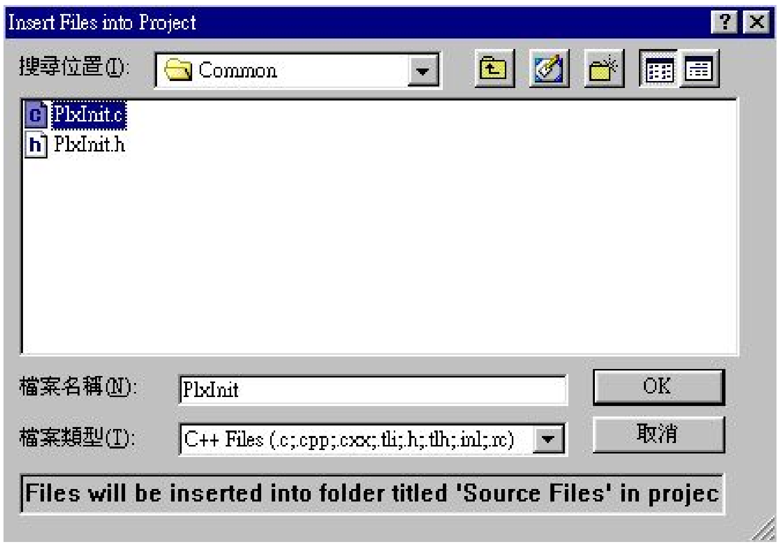


☑ Add to project: test

File: test

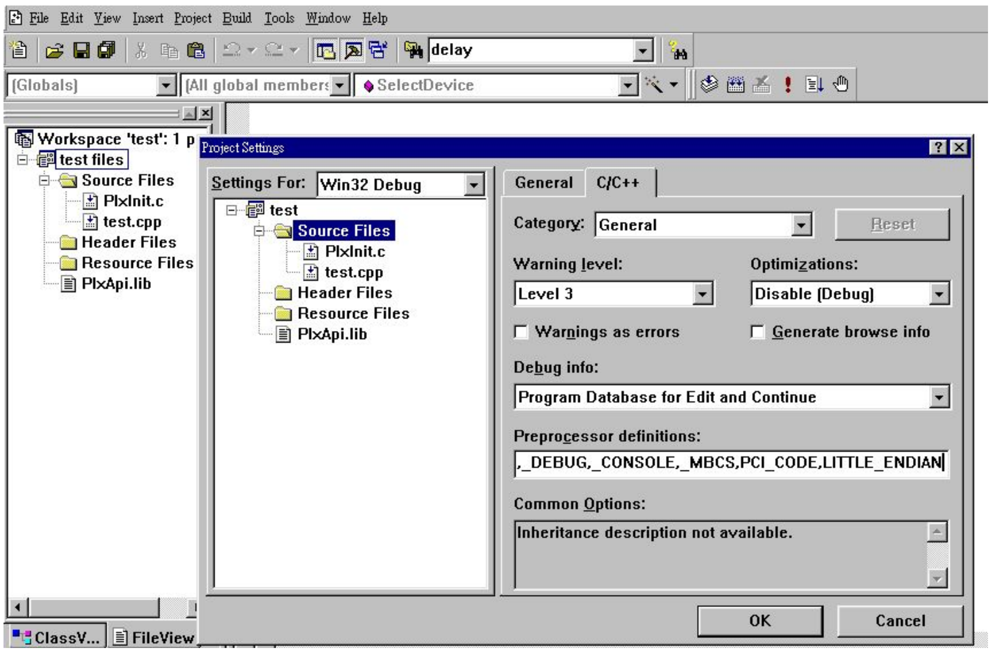
Location: c:\TEST\test

Figure 5-7-5 Add the program file (.c file) compiled



Files will be inserted into folder titled 'Source Files' in projec

Figure 5-7-6 Add PlxInit.c file



Settings For: Win32 Debug

Category: General

Warning level: Level 3

Optimizations: Disable (Debug)

Warnings as errors

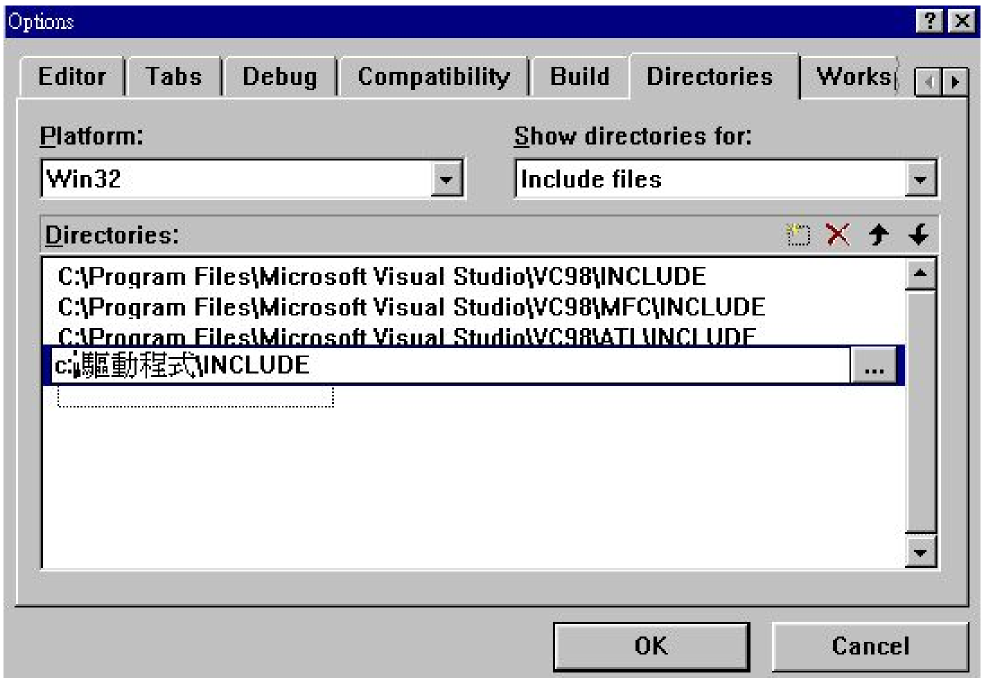
Generate browse info

Debug info: Program Database for Edit and Continue

Preprocessor definitions: ,_DEBUG,_CONSOLE,_MBCS,PCI_CODE,LITTLE_ENDIAN

Common Options: Inheritance description not available.

Figure 5-7-7 Add PlxApi.lib, PCI_CODE and LITTLE_ENDIAN setting



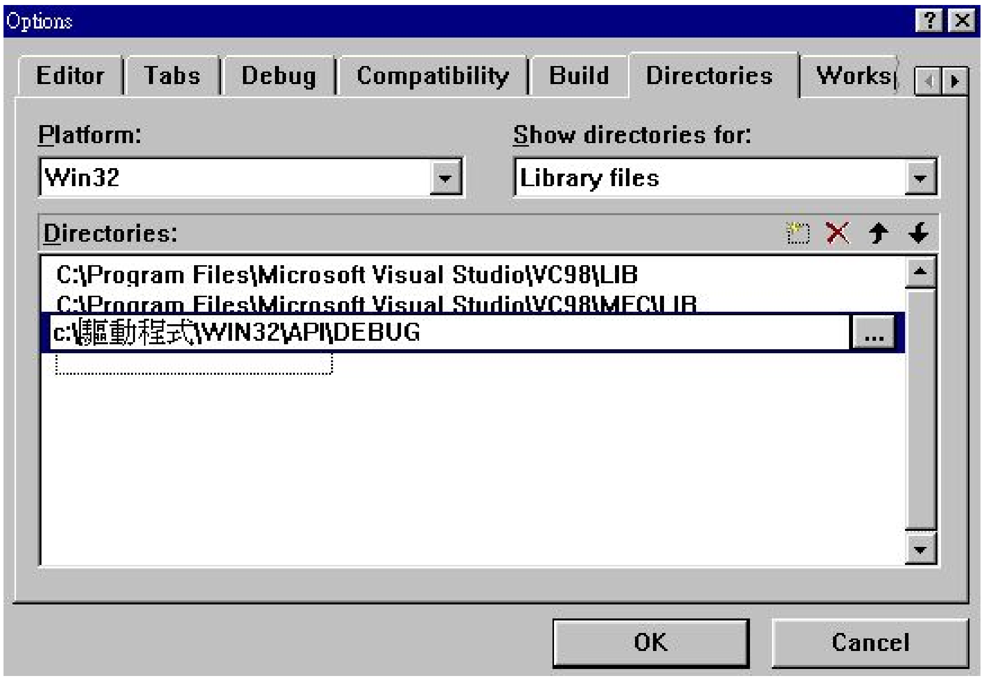
Platform: Win32

Show directories for: Include files

Directories:

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
c:\驅動程式\INCLUDE

Figure 5-7-8 Designating INCLUDE file



Platform: Win32

Show directories for: Library files

Directories:

C:\Program Files\Microsoft Visual Studio\VC98\LIB
C:\Program Files\Microsoft Visual Studio\VC98\MFC\LIB
c:\驅動程式\WIN32\API\DEBUG

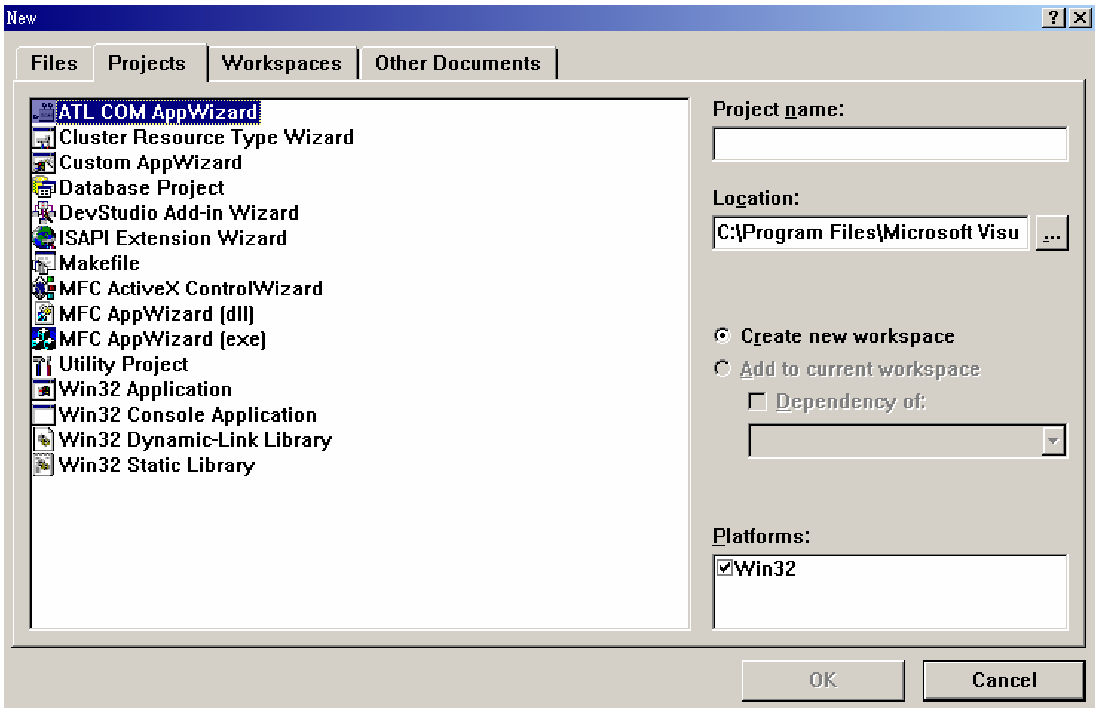
Figure 5-7-9 Designating Library file

This chapter only describes the setting of the PCI interface card, which mainly uses PLX-SDK original files to set. The next chapter is the simplified setting methods after sorting.

5.8 Visual C/C++ 6.0 initial environment setting STEP by STEP

PCI_IO interface card Visual C/C++ 6.0 initial environment setting STEP by STEP
PCI_IO interface card Visual C/C++ 6.0 initial environment setting STEP by STEP

1) From Start→Program sets→Microsoft Visual C++ 6.0→Microsoft Visual C++ 6.0 open Visual C++ 6.0
2) To open and complete Visual C++ 6.0 from File→New, the window has to be opened then can complete Visual C++ 6.0. It is necessary to build a PROJECT to compile the program, FIGURE as shown in Figure 5-8-1 appears.


New

Files | Projects | Workspaces | Other Documents

- ATL COM AppWizard
- Cluster Resource Type Wizard
- Custom AppWizard
- Database Project
- DevStudio Add-in Wizard
- ISAPI Extension Wizard
- Makefile
- MFC ActiveX ControlWizard
- MFC AppWizard (dll)
- MFC AppWizard (exe)
- Utility Project
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Static Library

Project name:

Location: C:\Program Files\Microsoft Visu

[✓ Create new workspace] Add to current workspace

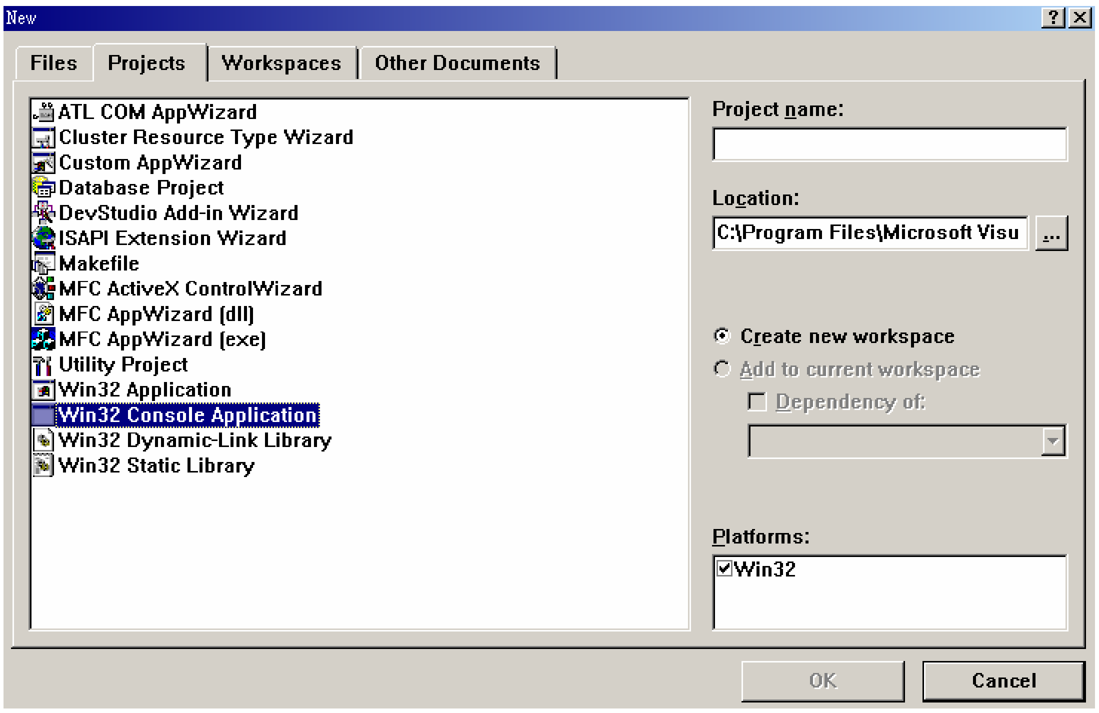
Dependency of:

Platforms: ✓ Win32

OK | Cancel

Figure 5-8-1

3) Win32 Console Application using the mouse in the window as shown in Figure 5-8-1 to return it to blue as shown in Figure 5-8-2.

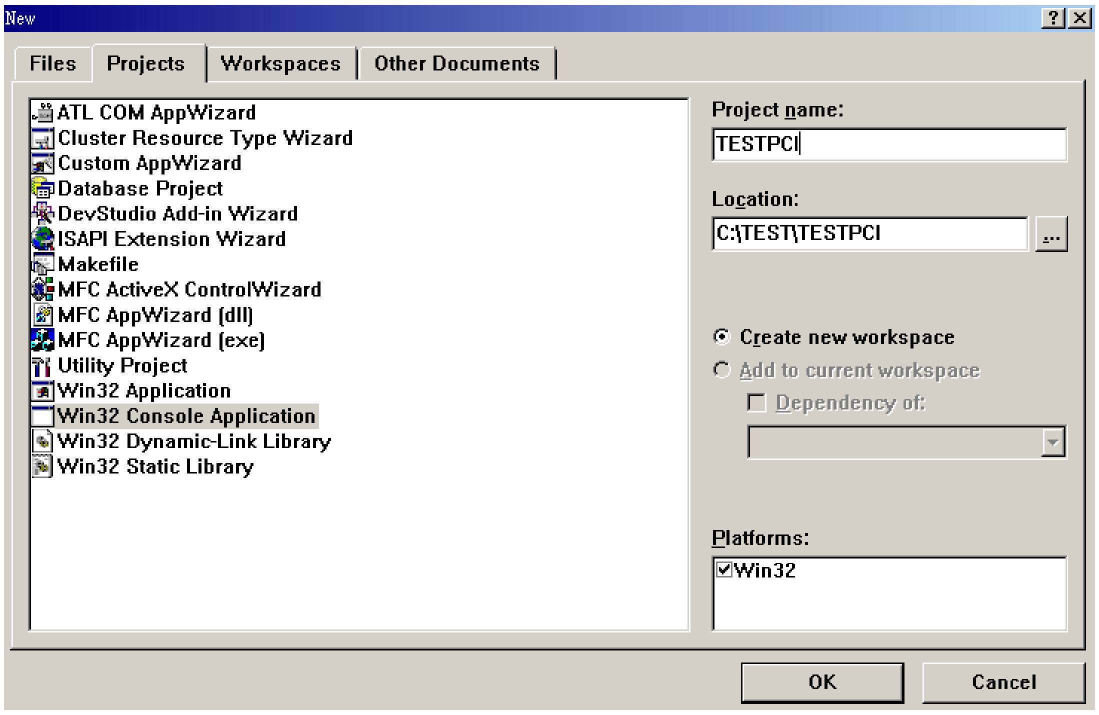


Location: C:\Program Files\Microsoft Visu

Platforms: [✓ Win32]

Figure 5-8-2

4) Project name (its name) and Location (its storage location) in the window may be changed, as shown in Figure 5-8-3. This example places the file whose Project name is TESTPCI in the location of C:\TEST\TESTPCI. Then finally click OK button using the mouse.



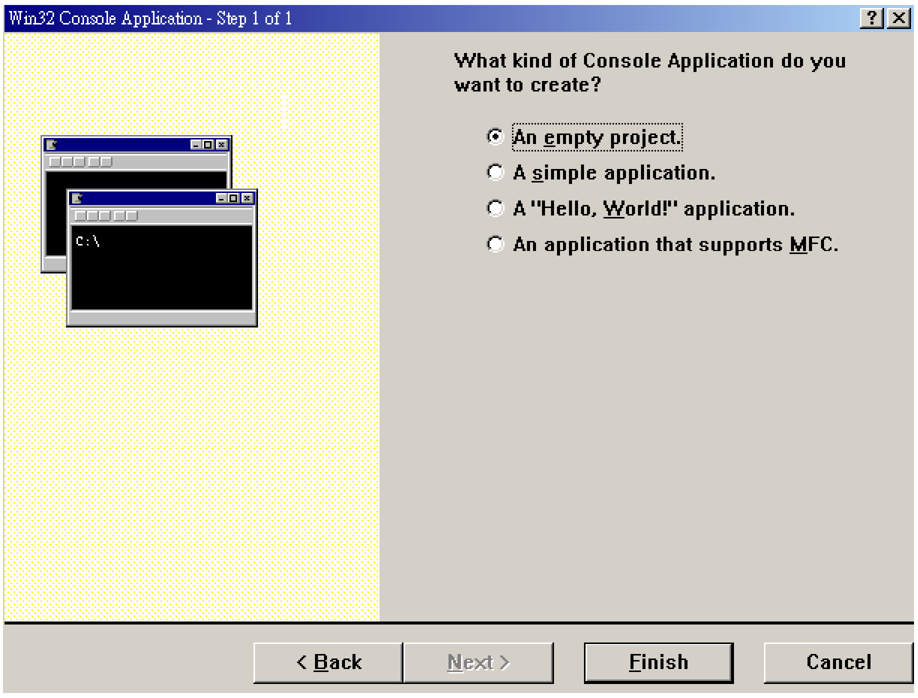
Project name: TESTPCI

Location: C:\TEST\TESTPCI

Platforms: [✓ Win32]

Figure 5-8-3

5) After clicking OK pushbutton, the window shown in Figure 5-8-4 appears. Click An empty project, and click Finish button using the mouse.

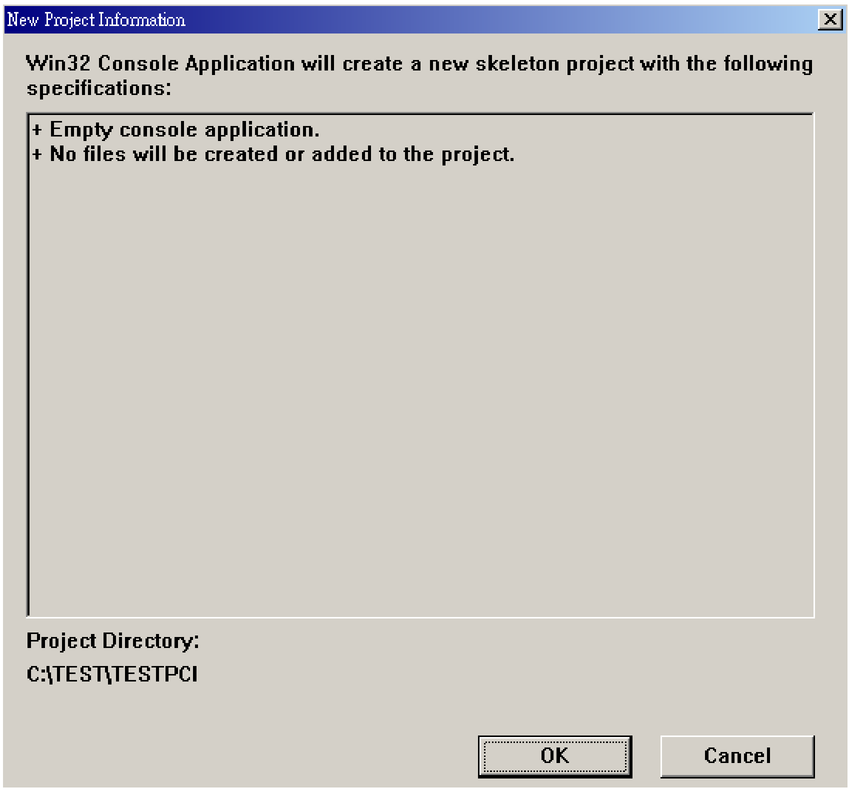

Win32 Console Application - Step 1 of 1

What kind of Console Application do you want to create?

[✓ An empty project.]
A simple application.
A "Hello, World!" application.
An application that supports MFC.

Figure 5-8-4

6) New Project Information window as shown in Figure 5-8-5 appears, with content description such as Empty console application and No files will be created or added to the project. In case of different information, please close the PROJECT built previously, and rebuild a new PROJECT again, click OK with the mouse to end this window. Following the above steps, the Test folder will appear in the hard disk, in which there will be a PROJECT folder of all TESTPCI built.


New Project Information

Win32 Console Application will create a new skeleton project with the following specifications:

+ Empty console application.
+ No files will be created or added to the project.

Project Directory:
C:\TEST\TESTPCI

Figure 5-8-5

7) Build and complete new PROJECT, start to set the program environment. Shown in Figure 5-8-6 is the content displayed in the sub-window on the right of the main window after building and completing new project, click TESTPCI Files, the cross pattern in the left will be unfolded, displaying the content shown in Figure 5-8-8, then it can be learned that TESTPCI Files consists of three folders: Source Files, Header Files, and Resource Files.

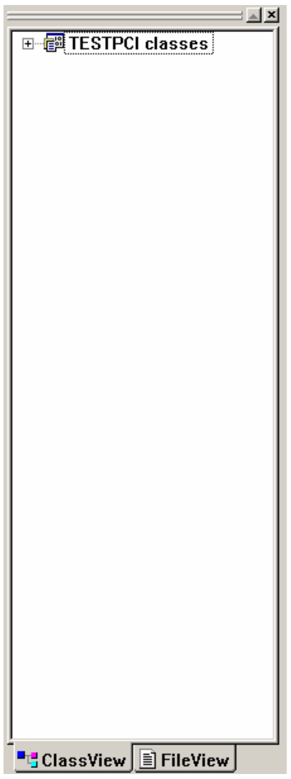


Figure 5-8-6

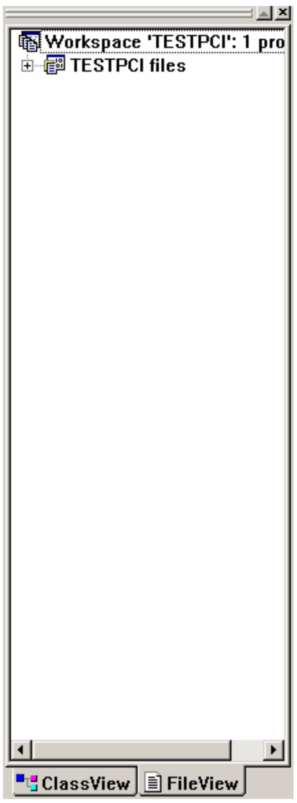


Figure 5-8-7

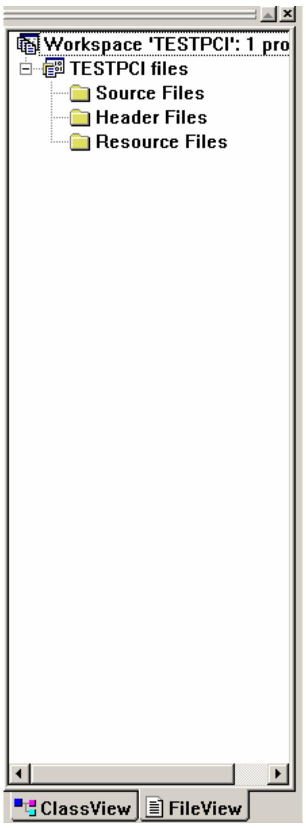


Figure 5-8-8

8) Click Source Files to return it blue, as shown in Figure 5-8-9, then the window shown in Figure 5-8-10 will appear from File→New, click C++ Source File to return it to blue, as shown in Figure 5-8-11. Fill in the name of this file in the box below the file, and fill in PCI as example.

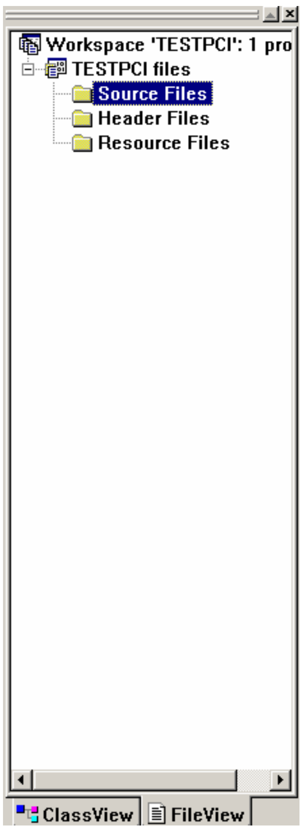
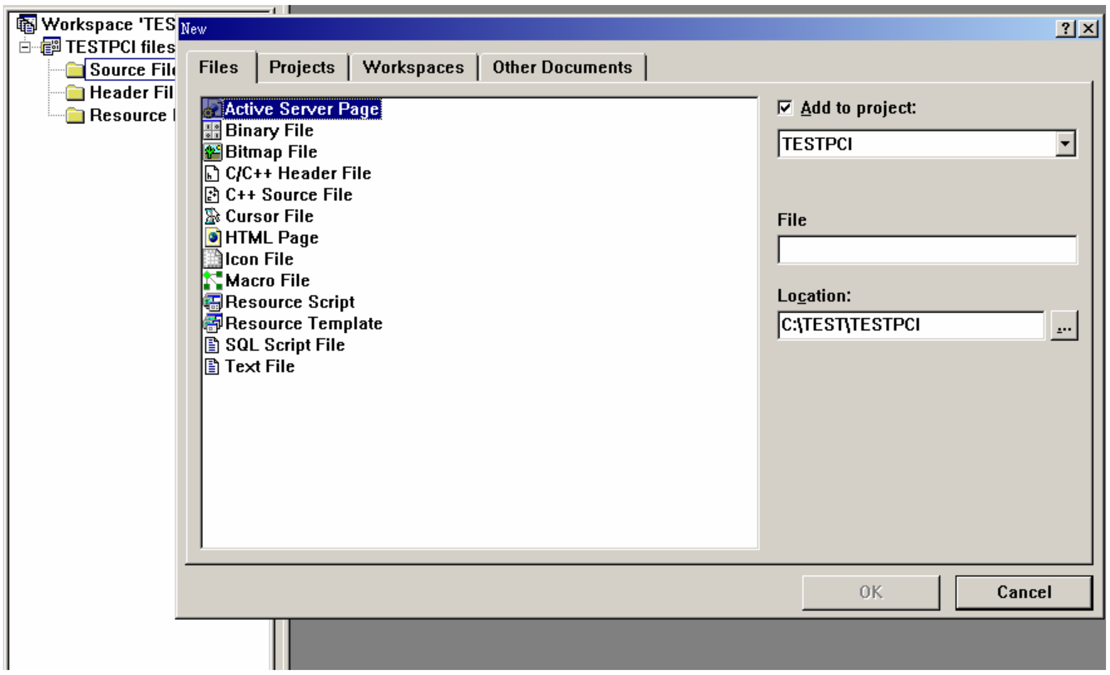


Figure 5-8-9

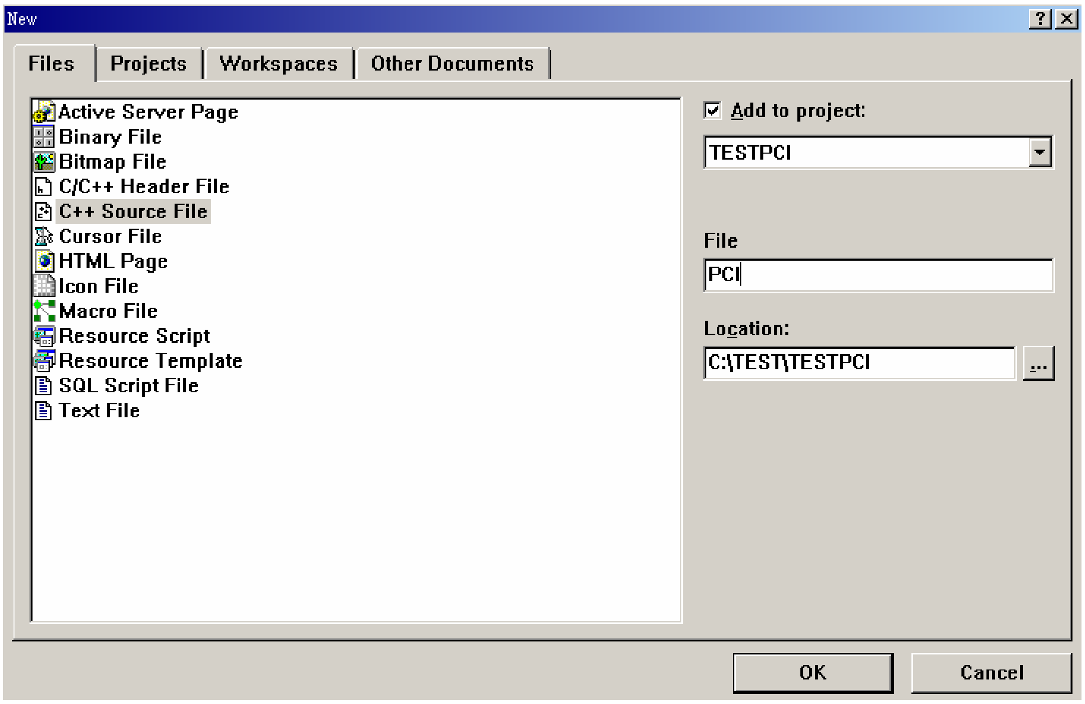


☑ Add to project: TESTPCI

File

Location: C:\TEST\TESTPCI

Figure 5-8-10


New

Files | Projects | Workspaces | Other Documents

Active Server Page
Binary File
Bitmap File
C/C++ Header File
C++ Source File
Cursor File
HTML Page
Icon File
Macro File
Resource Script
Resource Template
SQL Script File
Text File

☑ Add to project: TESTPCI

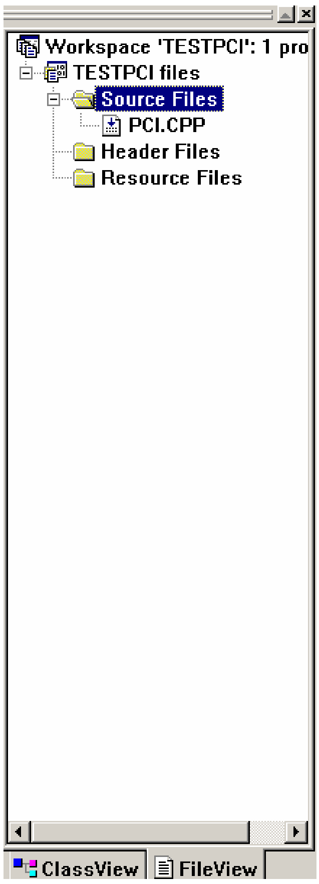
File: PCI

Location: C:\TEST\TESTPCI

OK | Cancel

Figure 5-8-11

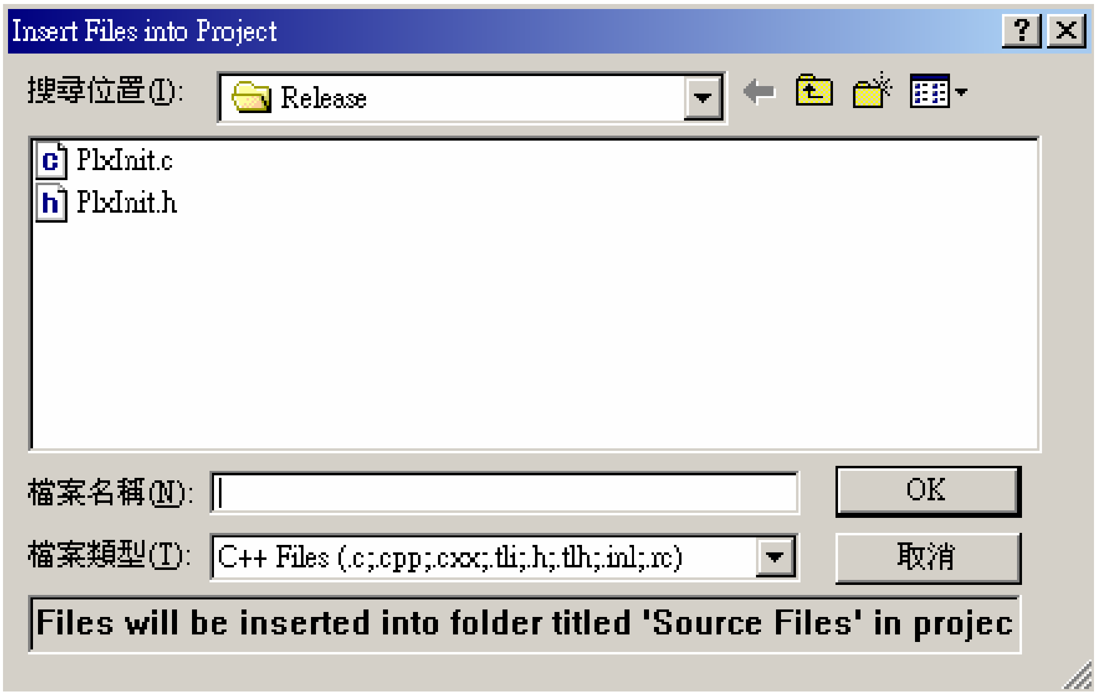
9) After clicking OK, PCI.CPP sample program logic file will be generated in C++ Source File folder, unfold C++ Source File folder, the results shown in Figure 5-8-12 will appear.


Workspace 'TESTPCI': 1 pro
TESTPCI files
Source Files
PCI.CPP
Header Files
Resource Files

ClassView | FileView

Figure 5-8-12

10) Move the mouse cursor onto C++ Source File folder. After clicking mouse right button and selecting Add Files to Folder, window shown in Figure 5-8-13 appears. Look for PlxInit.c and add this file by selecting OK, with the result shown in Figure 5-8-14.


Insert Files into Project

搜尋位置(I): Release

PlxInit.c
PlxInit.h

檔案名稱(N):

檔案類型(T): C++ Files (.c;.cpp;.cxx;.tli;.h;.tlh;.inl;.rc)

Files will be inserted into folder titled 'Source Files' in projec

Figure 5-8-13

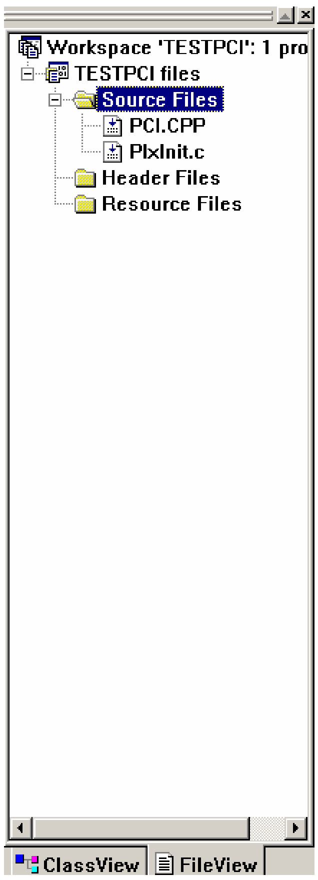


Figure 5-8-14

11) Click TESTPCI to return it to blue, click mouse right button and after clicking Add Files to Folder, select Library Files(.lib)in file types , as is shown in Figure 5-8-15. Look for PlxApi.lib and click OK to add it to PROJECT, with the result shown in Figure 5-8-16.

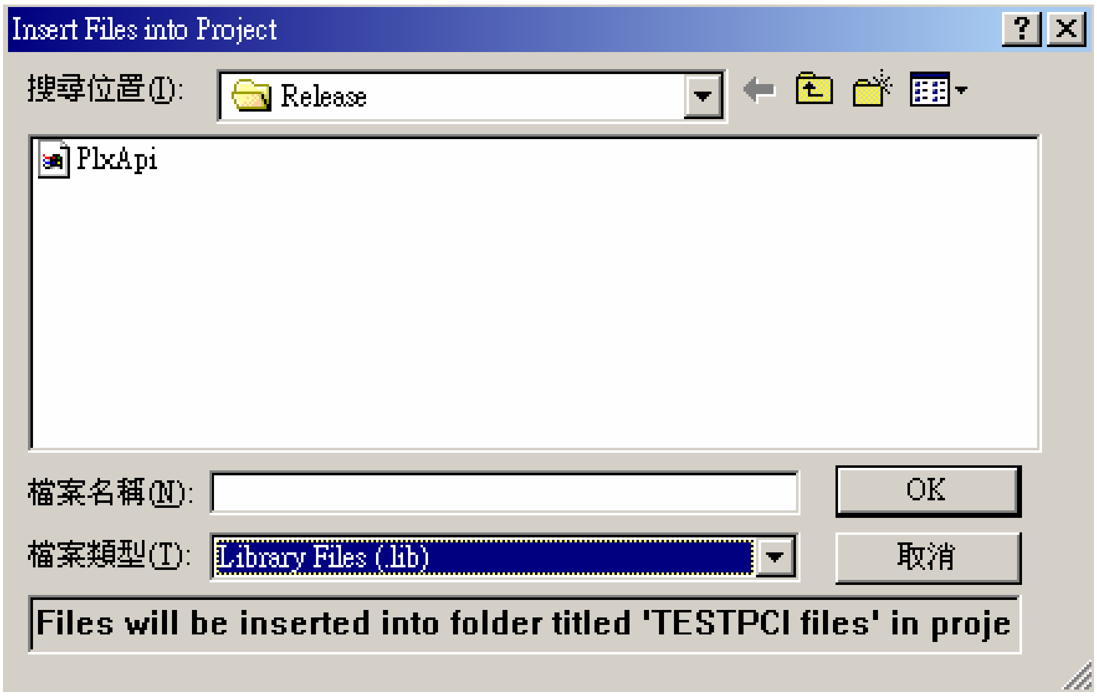

Insert Files into Project
搜尋位置(I):
Release
PlxApi
檔案名稱(N):
OK
檔案類型(T):
Library Files (.lib)
取消
Files will be inserted into folder titled 'TESTPCI files' in proje

Figure 5-8-15

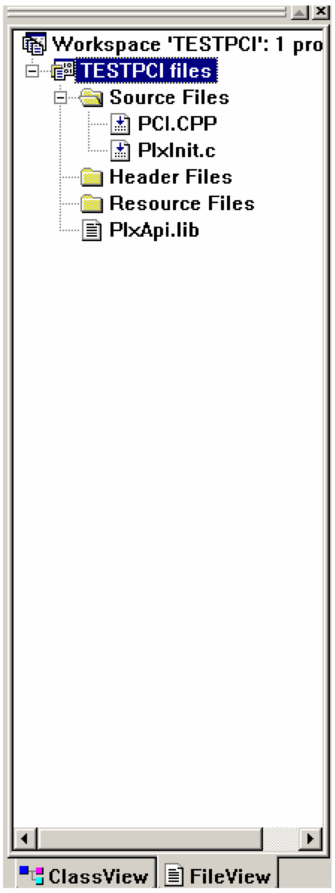
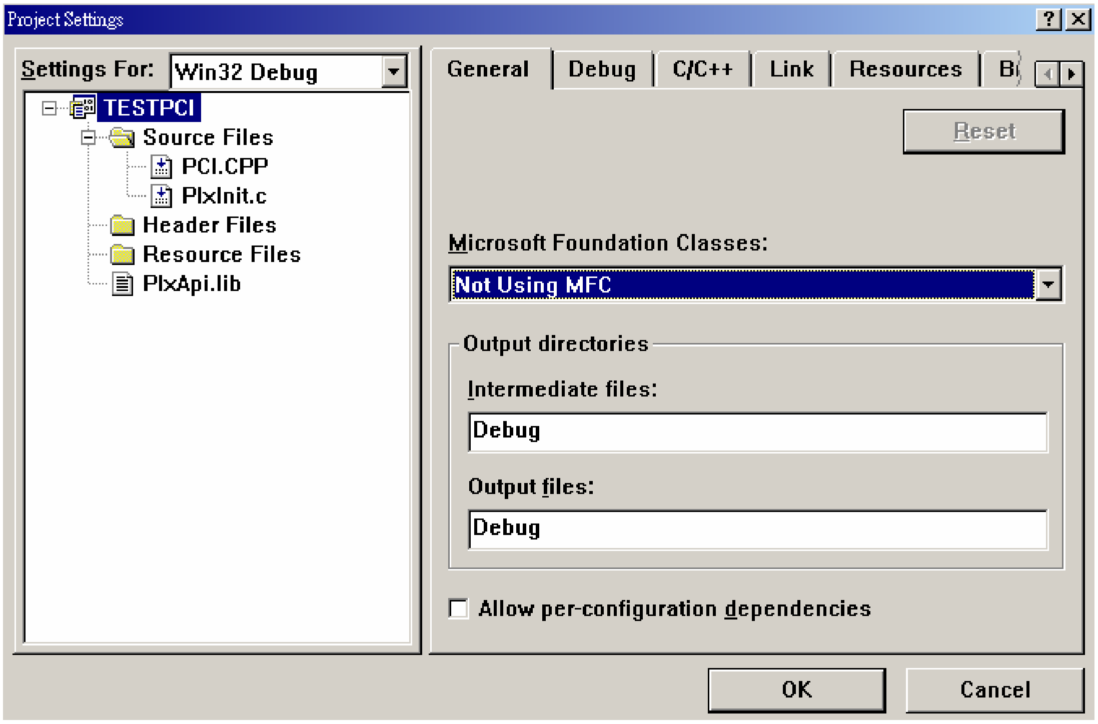


Figure 5-8-16

12) Open the Window as shown in Figure 5-8-17 from Project→Settings. Pay attention to set the whole TESTPCI PROJECT here. After clicking C/C++, the result is shown in Figure 5-8-18. Enter PCI_CODE and LITTLE_ENDIAN in Preprocessor definitions, as is shown in Figure 5-8-19, click OK to complete its setting. Here, attention should be paid to the cases of the characters entered.



Settings For: Win32 Debug

Microsoft Foundation Classes: Not Using MFC

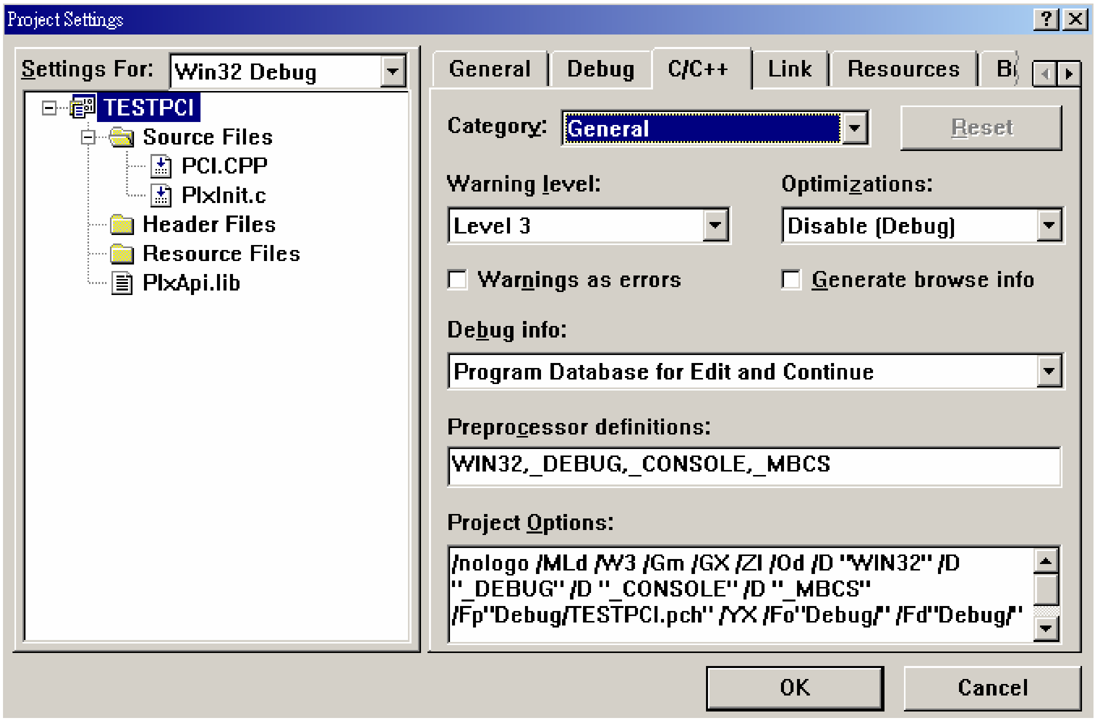
Output directories

Intermediate files: Debug

Output files: Debug

☐ Allow per-configuration dependencies

Figure 5-8-17



Settings For: Win32 Debug

Category: General

Warning level: Level 3

Optimizations: Disable (Debug)

☐ Warnings as errors

☐ Generate browse info

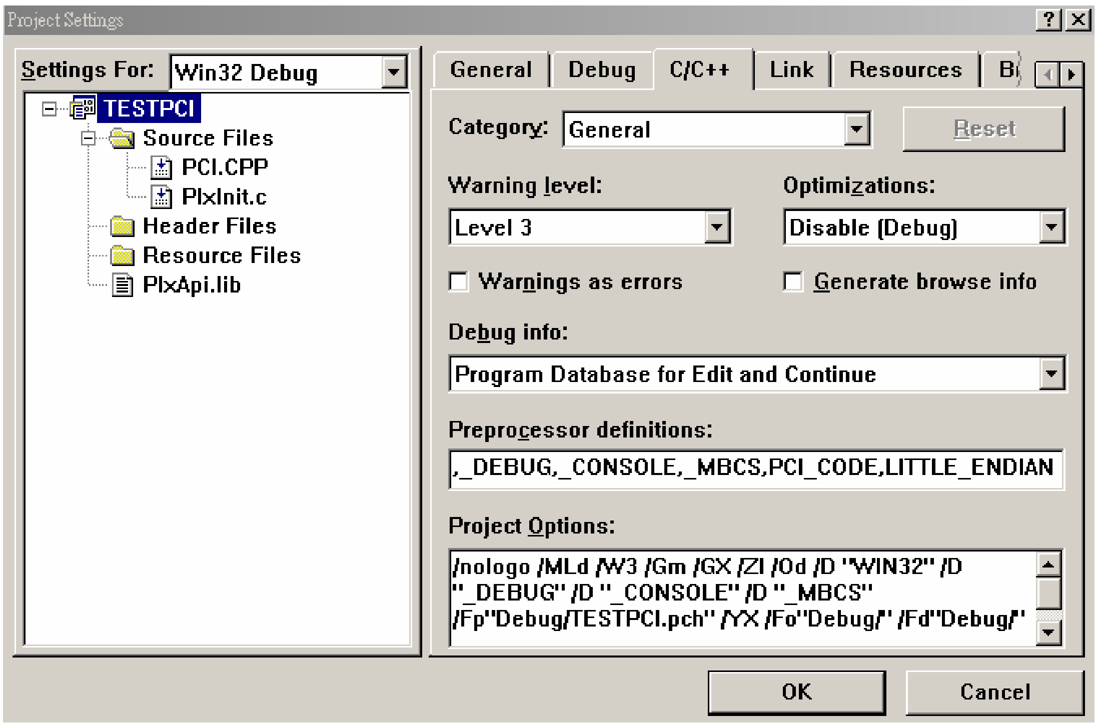
Debug info: Program Database for Edit and Continue

Preprocessor definitions: WIN32,_DEBUG,_CONSOLE,_MBCS

Project Options:

```
/nologo /MLd /W3 /Gm /GX /ZI /Od /D "WIN32" /D
"_DEBUG" /D "_CONSOLE" /D "_MBCS"
/Fp"Debug/TESTPCI.pch" /YX /Fo"Debug/" /Fd"Debug/"
```

Figure 5-8-18



Category: General

Warning level: Level 3

Optimizations: Disable (Debug)

☐ Warnings as errors

☐ Generate browse info

Debug info: Program Database for Edit and Continue

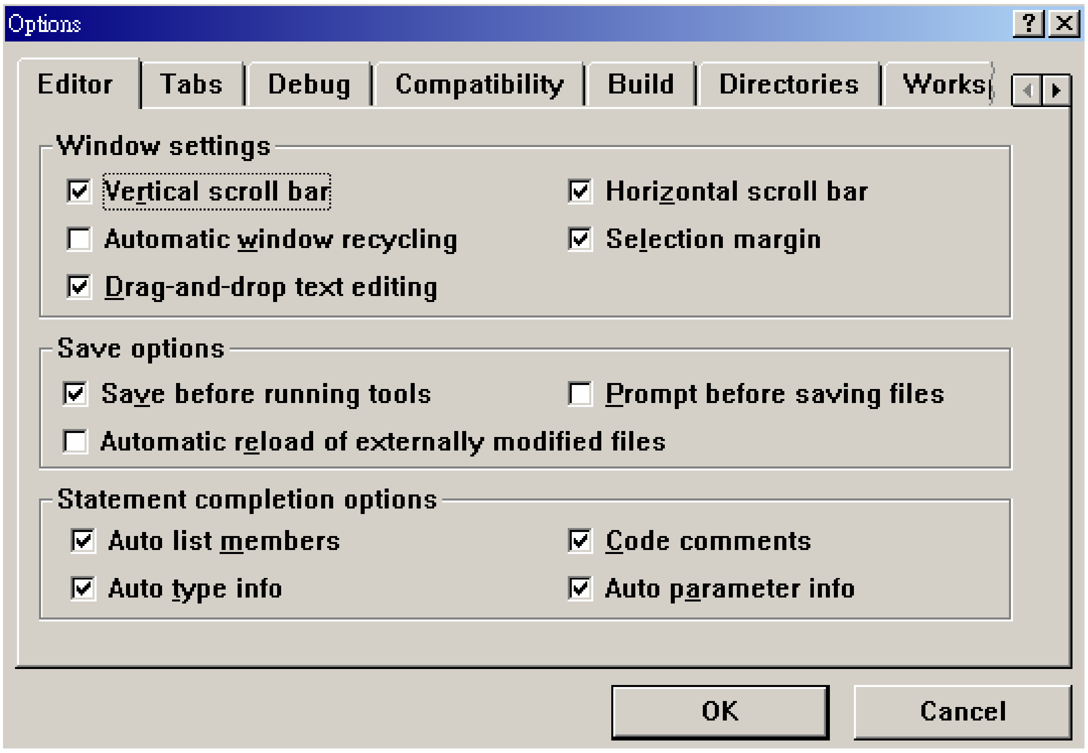
Preprocessor definitions: ,_DEBUG,_CONSOLE,_MBCS,PCI_CODE,LITTLE_ENDIAN

Project Options:

```
/nologo /MLd /W3 /Gm /GX /ZI /Od /D "WIN32" /D
"_DEBUG" /D "_CONSOLE" /D "_MBCS"
/Fp"Debug/TESTPCI.pch" /YX /Fo"Debug/" /Fd"Debug/"
```

Figure 5-8-19

13) Open the window as shown in Figure 3-8-20 form Tools→Options, select Directories, and its result is shown in Figure 3-8-21, select Include Files in Show directories for, and move the mouse cursor to the box under Directories, click mouse left key on it, as is shown in Figure 3-8-22, click and the status shown in Figure 3-8-22 will be displayed, and click the "…" Pushbutton on its right to look for the HEADER FILES as shown in Figure 3-8-24. The result is shown in Figure 3-8-25. Add the ".h" document corresponding to the PLX chip of this device here.


Options

Editor | Tabs | Debug | Compatibility | Build | Directories | Works

Window settings

- [✓] Vertical scroll bar
- [✓] Horizontal scroll bar
- [ ] Automatic window recycling
- [✓] Selection margin
- [✓] Drag-and-drop text editing

Save options

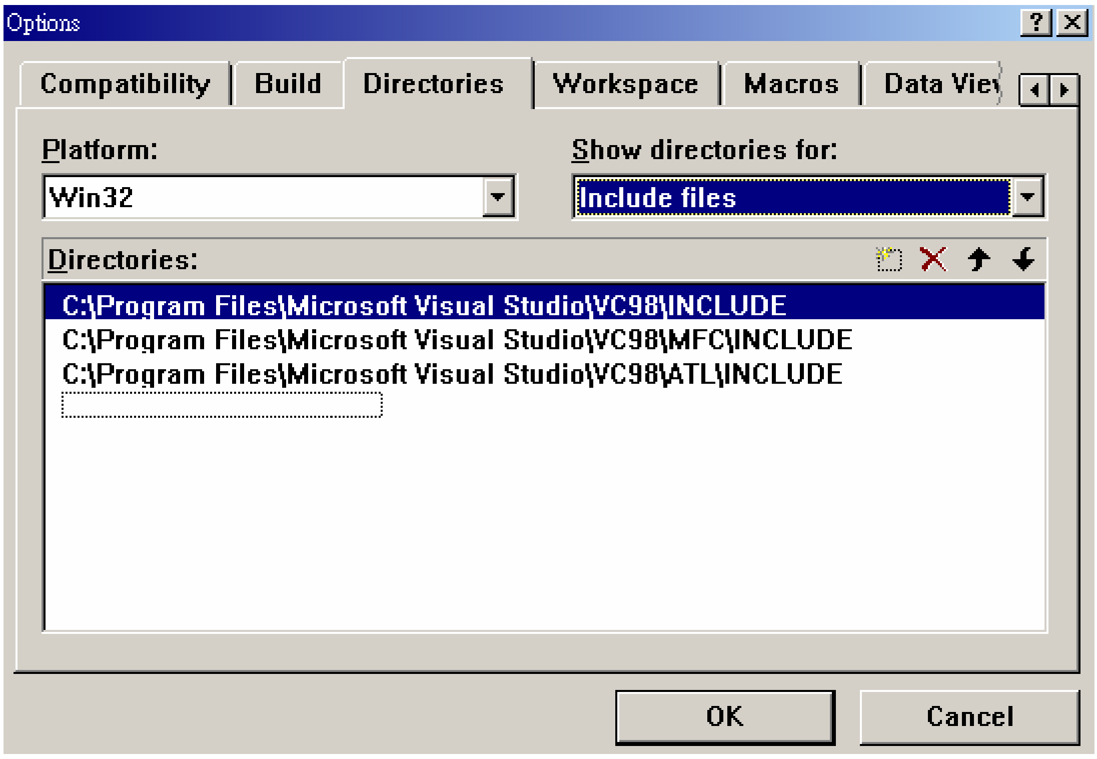
- [✓] Save before running tools
- [ ] Prompt before saving files
- [ ] Automatic reload of externally modified files

Statement completion options

- [✓] Auto list members
- [✓] Code comments
- [✓] Auto type info
- [✓] Auto parameter info

OK | Cancel

Figure 5-8-20


Options

Compatibility | Build | Directories | Workspace | Macros | Data View

Platform: Win32

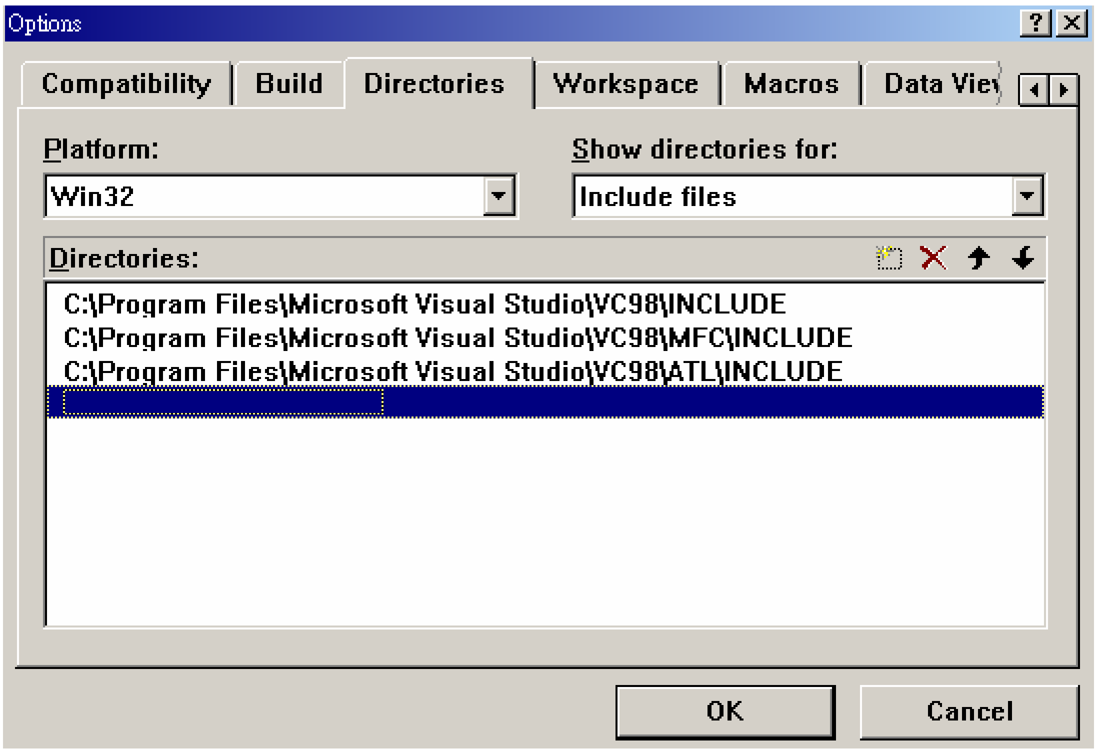
Show directories for: Include files

Directories:

```
C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
```

OK | Cancel

Figure 5-8-21


Options

Compatibility | Build | Directories | Workspace | Macros | Data Viev

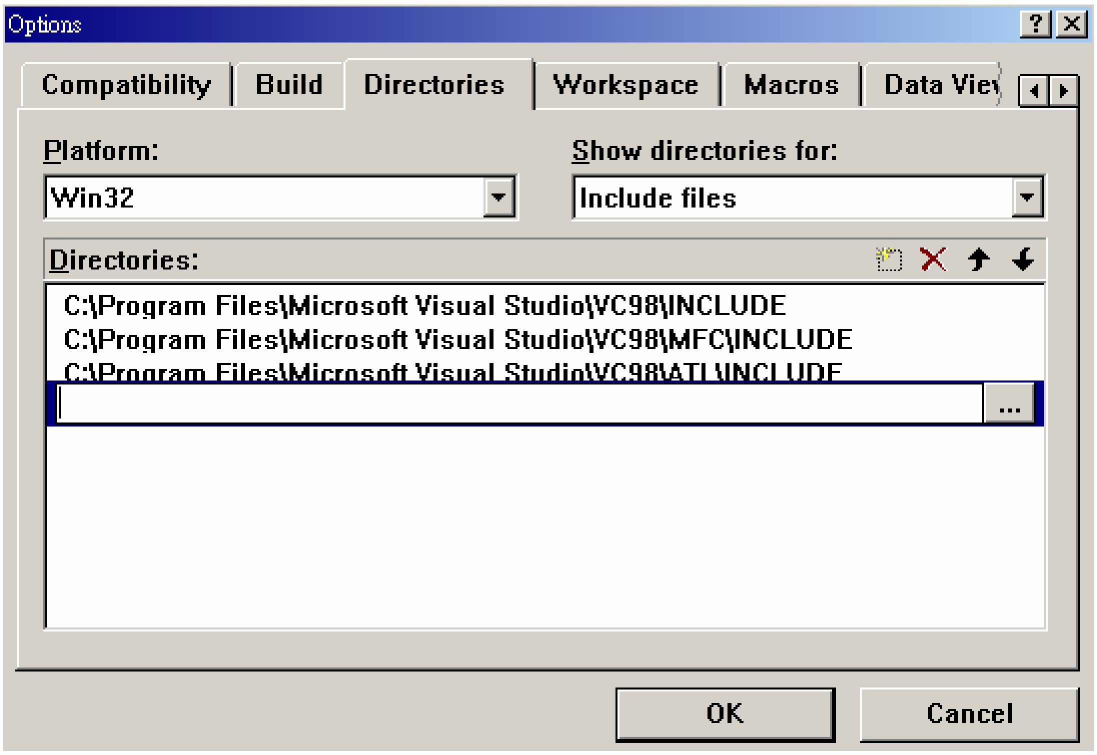
Platform: Win32

Show directories for: Include files

Directories:

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE

Figure 5-8-22


Options

Compatibility | Build | Directories | Workspace | Macros | Data Viev

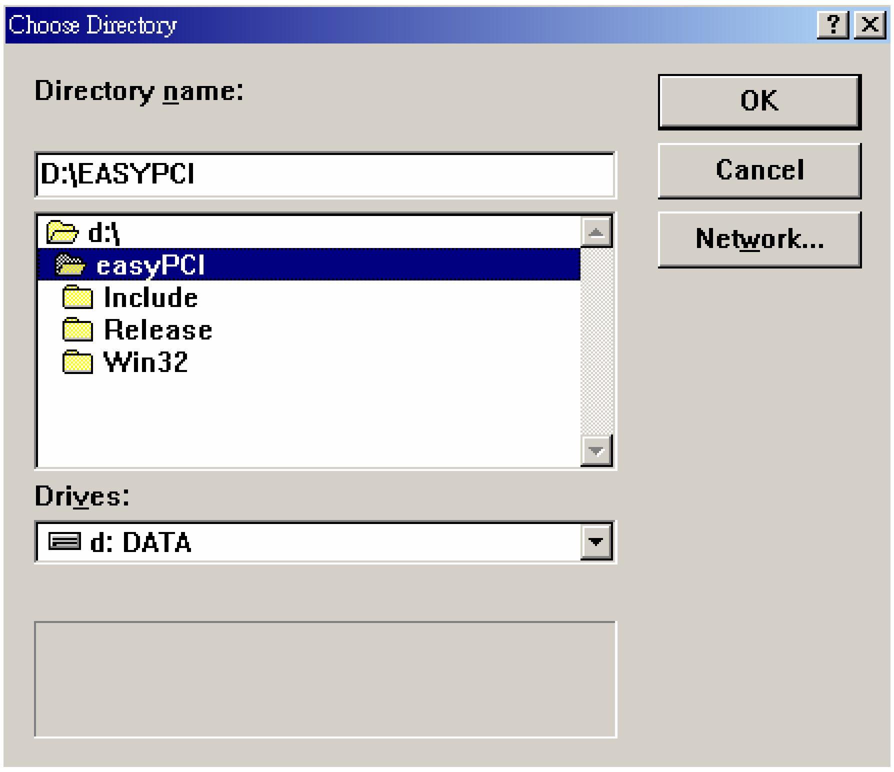
Platform: Win32

Show directories for: Include files

Directories:

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE

Figure 5-8-23

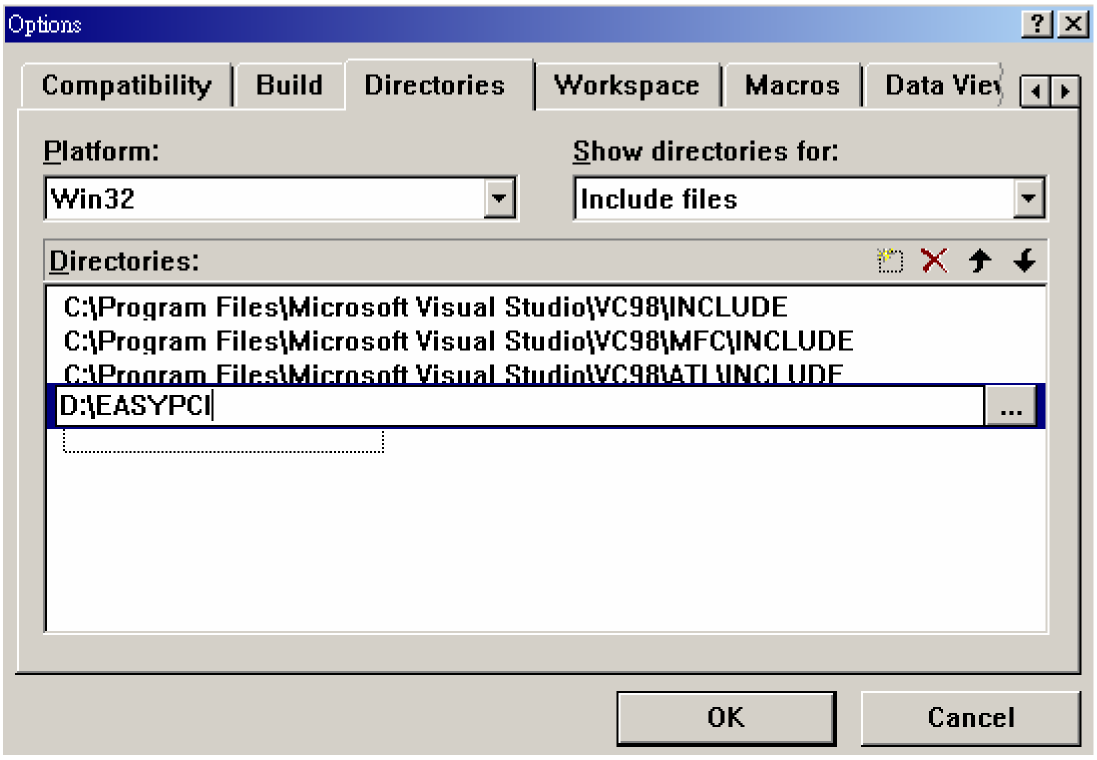


D:\EASYPCI

d:\
easyPCI
Include
Release
Win32

d: DATA

Figure 5-8-24



Platform: Win32

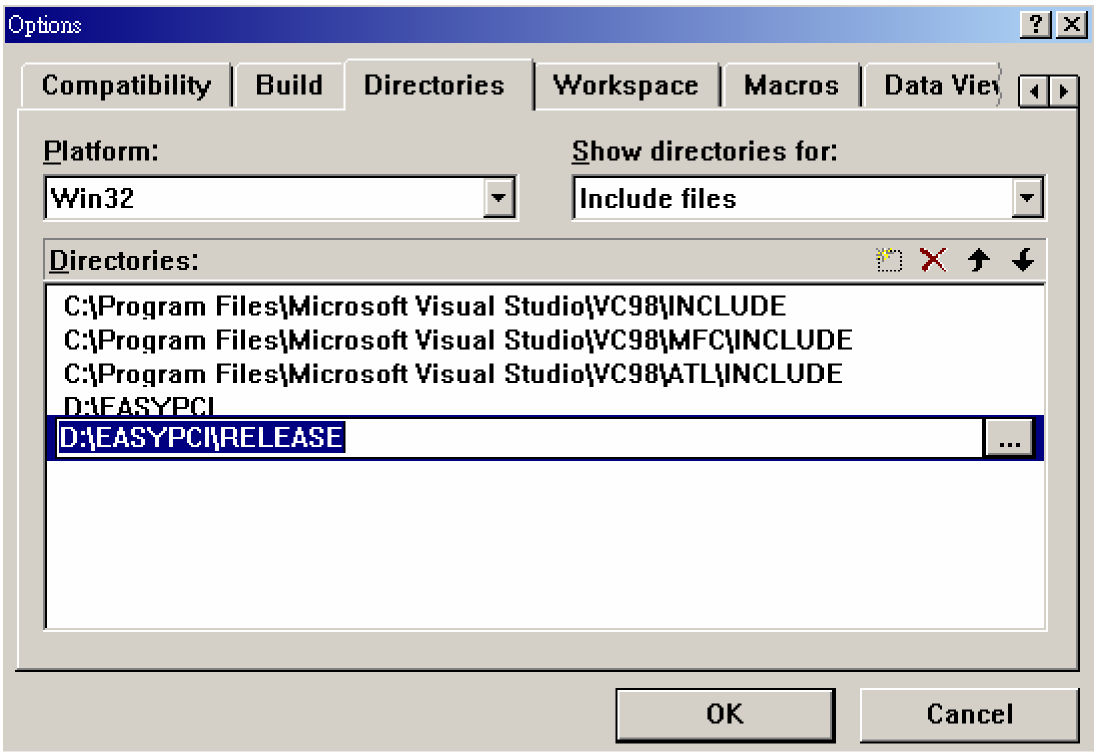
Show directories for: Include files

Directories:

C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
D:\EASYPCI

Figure 5-8-25

14) Add folder which contains PlxInit.h in the same sequence as step 13, as is shown in Figure 5-8-26.



Platform: Win32

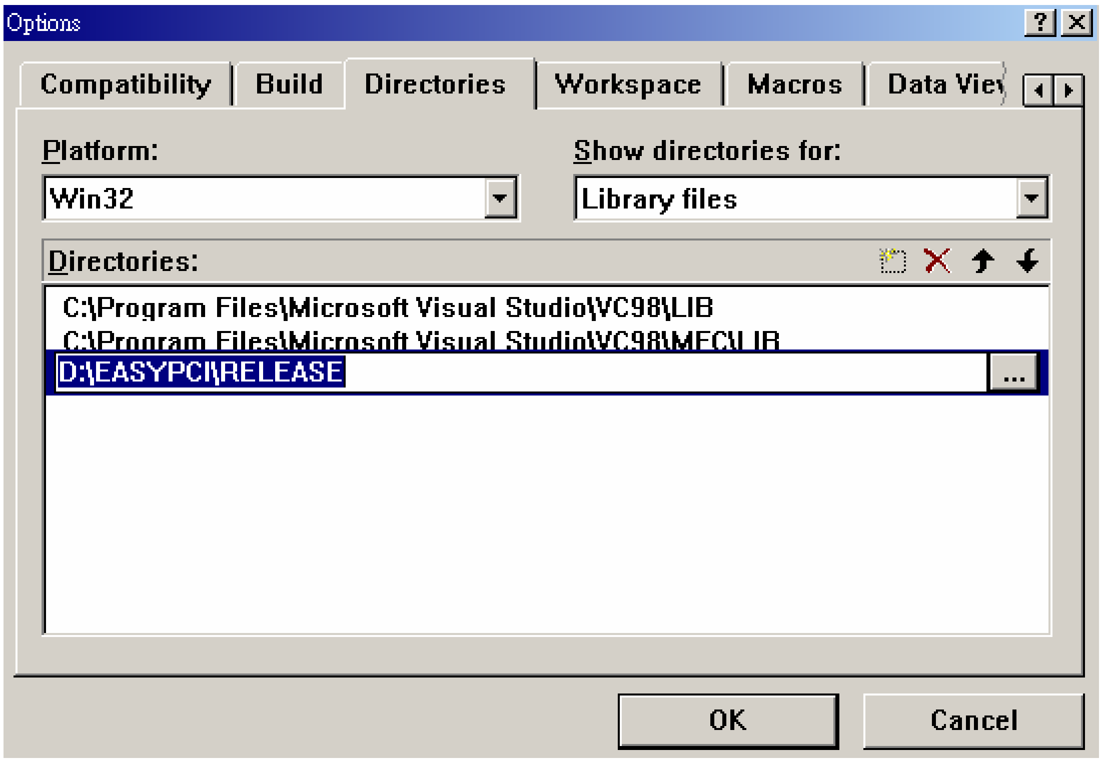
Show directories for: Include files

Directories:

```
C:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
D:\EASYPCI
D:\EASYPCI\RELEASE
```

Figure 5-8-26

15) Select Library Files in Show directories for, add it to the folder where PlxApi.lib as in step 13, as shown in Figure 5-8-27, then click OK to complete the setting.



Platform: Win32

Show directories for: Library files

Directories:

```
C:\Program Files\Microsoft Visual Studio\VC98\LIB
C:\Program Files\Microsoft Visual Studio\VC98\MFC\LIB
D:\EASYPCI\RELEASE
```

Figure 5-8-27

16) Click PCI.cpp using the mouse, and you can proceed to write the program code.

Chapter 6 Program language description

From what is described in chapter five, IO port address of PCI-IO card can be read. The reader records this address in table 6-0 below, in which reference IO address is the IO address used in the test in this book. The reader must compile the programs using the setting of the computer host machine. Described below are the instructions needed by Debug Mode, Masm, and vc/c++.

Table 6-0 PCI-IO interface card IO address reference values and readers experiment value record

Input/output scope	Reference values in this book	Scope	Reader's practice value (self-record)
Lowest IO scope	A400~A4FF	_00~_FF	
Second lowest IO scope	A888~A88B	__8~__B	
Highest IO scope	AC80~ACFF	_80~_FF	

6.1 Debug mode instructions

As is shown in Figure 6-1-1 below is the Debug mode instructions for the readers' references.

Table 6-1-1 Debug Mode commands

Command	Use	Format
A	Assemble	A [address]
C	Compare	C range address
D	Dump (Dump memory)	D [address] or D[range]
E	Enter Memory	E address [list]
F	Fill memory block	F range list
G	GO (execute)	G [= add exec][[addr]..]
H	Hexadecimal addition/subtraction	H Value1 Value2
I	Input from port	I port address
L	Load file	L [address]
L	Load sector	L address drive sector n
M	Move memory block	M range address
N	Name file for Load and Write	N file spec [filespec]
O	Output byte to port	O port address byte
Q	Quit to DOS	Q
R	Display and or change Register/flags	R [register]
S	Search list or string	S range list
T	Trace	T [=address][value]
U	Unassembled	U [address]
W	Write file	W [address](Write length to center of CX)
W	Write sector	W address drive sector n

6.2 MASM description

Generally speaking, register of 80X86 series CPU has the following types: common register: AX, BX, CX, DX; point register: IP, SP, BP, SI, DI; segment register: CS, DS, ES, SS; flag register: flag etc. While a float-point register (R7~R0) and a 16 bits status register are added to 80486 portion, these are roughly commonly seen registers.

Common register: AX (Accumulator store the result of operation), BX (base registration), CX (counter), DX (date), 32 bits are four registers: EAX, EBX, ECX, EDX. While AX, BX, CX, DX can be further divided into high low bytes. Figure 6-2-1 below shows its common bit format sketch.

EAX																															
AX																															
AL								AH																							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 6-2-1 Common register format

Pointer register is divided into IP(Instruction Pointer), program pointer, SP, ESP (Stack Pointer), stack pointer, BP, EBP(Stack Pointer), stack pointer, SI, ESI (Index Pointer), index pointer and DI, EDI (index pointer) index pointer. The size of a segment is 64KB, the address pointed by segment register is the lowest address, and the four registers do not affect each other, which can fully or partly point to the same memory address. The Flag register mainly displays CPU status and operation results.

Figure 6-2-2 below shows the bit definition of flag register; while for the rest of register figures, please refer to books related with combined languages.

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved				OF	DF	IF	TF	SF	ZF	Reserved	AF	Reserved	PF	Reserved	CF
				Overrun	Direction	Interrupt	Single step	Symbol	Zero value		Auxiliary carry		Odd/even		Carry

Figure 6-2-2 Bit definition of flag register

6.3 MASM instruction

Table 6-3-1 below shows the instructions of MASM assembly language; the reader himself can refer to books related to MASM for more detailed instruction explanations.

Table 6-3-1 MASM instructions

Instruction group	Detailed division	Instruction
Data transmission	Data transmission	mov, movsx, movzx, xchg
	Pile address	push, pop, pushf, popf, pusha, popa, pushfd, popfd
	Address access	lea, lds, les, lss, lfs, lgs
	Form access	Xlat
	Flag access	lahf, sahf
Arithmetic operation	Addition	add, adc, inc
	Subtraction	sub, sbb, dec, neg
	Multiply	mul, imul
	Divide	div, idiv, cbw, cwq
	BCD	aaa, aas, aam, aad, daa, das
Bit operation	Logic operation	and, or, xor, not
	Bit test	bt, btc, btr, bts, bsf, bsr
	Flag setting	Setxx
	Offset rotation	shl, shr, sal, sar, ror, rol, rcl, rcr, shrd, shld
Flow control	Jump	jmp, jxx
	Test compare	cmp, test
	Loop	loop, loopxx
	Auxiliary program call	call, ret, retn, retf
	Auxiliary program enter and leave	enter, leave
	Interrupted call	int, into, iret
String processing	String processing	movs, scas, cmps, lods, stos
	Each string processing instruction has changes such ad xxs,ssxb,ssxw,ssxd	
	Stub code	rep, repe, repz, repne, repnz
I/O	Data I/O	in, out

	String I/O	ins, insb, insw, insd, outs, outsb, outsw, outsd
CPU control	Direct mode	lock, wait, esc, hit
	Memory scope	Bound
	Address	nap
	Protection mode	lar, lsl, lgdt, sgdt, lidt, sidt, lldt, sldt, ltr, str, lmsw, smsw, arpl, clts, verr, verw
	80386 exclusive control instruction	
	Control instruction	cr0, cr2, cr3
	Error detecting instruction	dr0, dr1, dr2, dr3, dr6, dr7
	Test instructions	tr6, tr7

6.4 PCI-IOin C/C++ program language instructions

API needed to be used by this PCI-IO interface card are:

DeviceSelected
PlxChipTypeGet
PlxPciDeviceOpen
PlxPciDeviceClose
PlxPciConfigRegisterRead
PlxIoPortWrite
PlxIoPortRead

For API of other functions, please refer to PLX SDK Programmer's Reference Manual document in the folder of drive program in the disk attached with this book. The above API can use this interface card in a simple way.

As with the actions of this interface card, select the interface card device with DeviceSelected and PlxChipTypeGet, use PlxPciDeviceOpen to open the interface card, then use PlxPciConfigRegisterRead" to read IO base address. Then IO data can be outputted with PlxIoPortWrite or inputted with PlxIoPortRead. In the end, the program executes the action of closing this interface card and completes this action with PlxPciDeviceClose, which may not be used. However, each VC/C++ must have a start and return-to-zero action. About the samples of each program, the user can observe its use, now we will proceed to briefly describe the use of PlxPciDeviceOpen, PlxPciDeviceClose, PlxIoPortWrite, PlxIoPortRead and PlxPciConfigRegisterRead.

◎PlxPciDeviceOpen:

Format:

```
RETURN_CODE
PlxPciDeviceOpen(
DEVICE_LOCATION *pDevice,
HANDLE *pDrvHandle
);
```

Sample:

```
HANDLE hDevice;
RETURN_CODE rc;
DEVICE_LOCATION Device;

rc = PlxPciDeviceOpen(
&Device,
&hDevice
);

if (rc != ApiSuccess)
{
// ERROR – Unable to open a PLX device
}
```

◎PlxPciDeviceClose:

Format:

```
RETURN_CODE
PlxPciDeviceClose(
HANDLE hDevice
);
```

Sample:

```
HANDLE hDevice;
RETURN_CODE rc;
// Release the open PLX device
rc = PlxPciDeviceClose(
hDevice
);

if (rc != ApiSuccess)
{
// ERROR – Unable to release PLX device
}
```

◎ PlxIoPortWrite(IO port write))):

Format:

```
RETURN_CODE
PlxIoPortWrite(
HANDLE hDevice,
U32 address,
ACCESS_TYPE bits,
VOID *pValue
);
```

Sample:

```
U32 port;
U32 RegValue;
HANDLE hDevice;
RETURN_CODE rc;

port = PlxPciConfigRegisterRead(
Device.bus,
Device.slot,
CFG_BAR1,
&rc
);
```

```
port = port & ~(1 << 0);
RegValue = 0x00300024;
rc = PlxIoPortWrite(
hDevice,
port + 0x34, // Write to local register 34h
BitSize32,
&RegValue
);
if (rc != ApiSuccess)
{
// ERROR - Unable to read I/O port
}
```

◎ PlxIoPortRead(IO port read):

Format:

```
RETURN_CODE
PlxIoPortRead(
HANDLE hDevice,
U32 address,
ACCESS_TYPE bits,
VOID *pOutData
);
```

Sample:

```
U32 port;
U32 RegValue;
HANDLE hDevice;
RETURN_CODE rc;

port = PlxPciConfigRegisterRead(
Device.bus,
Device.slot,
CFG_BAR1,
&rc
);

port = port & ~(1 << 0);
rc = PlxIoPortRead(
hDevice,
port + 0x34,          // Read local register 34h
BitSize32,
&RegValue
);
if (rc != ApiSuccess)
{
// ERROR - Unable to read I/O port
}
```

◎ PlxPciConfigRegisterRead read:

“Subsystem Device/Vendor ID”

Format

```
U32
PlxPciConfigRegisterRead(
U32 bus,
U32 slot,
U32 registerNumber,
RETURN_CODE *pReturnCode
);
```

sample:

```
U8 bus;
U8 slot;
U32 RegValue;
RETURN_CODE rc;
DEVICE_LOCATION Device;

RegValue =
PlxPciConfigRegisterRead(
Device.BusNumber,
Device.SlotNumber,
CFG_SUB_VENDOR_ID,
&rc
);
```

6.5 IO definition port of LEAP PCI-IO/LAB

IO definition port of LEAP PCI-IO/LAB is shown in Figure 6-5-1 below. LEAP PCI-IO card shares four IO-BANKs, each of which has forty-eight IO outputs, and eight IO outputs form an IO-PORT. As is shown in table 4-0, IO-BANK is built-in IO-BANK (the user is unable to change it), which is used by built-in 8254. IO-BANK1~IO-BANK4 can be used by the reader himself. The IO address of IO-BANK1 IO-PORT0 is A410h, corresponding to I00~I07 of IO-BANK1. The IO address of IO-BANK2 IO-PORT0 is A420h, corresponding to IO0~IO7 of IO-BANK2, while the IO address of built-in 8254 is A400~A403 on IO-BANK0.

Table 6-5-1 IO definition port of PCI-IO/LAB

Unit / I/O MAP		7-segment code LED display &LCD display & 4x4 keyboard & buzzer & A/D&D/A		Dot matrix LED display & Logic state input key
		IO_BANK_1		IO_BANK_2
I/O0-IO7	Data BUS	LCD display D/A unit A/D unit	LCD_DB0-LCD_DB7 DA_DB0-DA_DB7 AD_DB0-AD_DB7	Dot matrix LED LED_COL1 | LED-COL16 (OUT)
I/O8-IO15	OUT	7 segment LED	LED a-LED g & LED p	
	IN			
I/O16-IO19	OUT	4×4 keyboard KEY_IN0-KEY_IN3		
I/O20-IO23	IN	4×4 keyboard KEY_SEL0-KEY_SEL3		
I/O24-IO31	OUT			
	IN			Logic state input key SW_1-SW_8

I/O32	OUT	7 segment LED LCD unit /D/A unit A/D unit	IO33-32=0 enable LED7_EN IO33-32=01 enable LCD_EN IO33-32=10 enable /DA_CS IO33-32=11 enable /DA_CS	Dot matrix LED	ROW_SEL0
I/O33	OUT			Dot matrix LED	ROW_SEL1
I/O34	OUT	Unit read write control	IO35-34=10 enable /IO_RD IO35-34=01 enable /IO_WR	Dot matrix LED	ROW_SEL2
I/O35	OUT			Dot matrix LED	ROW_SEL3
I/O36	OUT	LCD display	LCD_RS		
I/O37	OUT	D/A unit	DA_A/B		
I/O38	OUT	Buzzer	BUZZER_CTRL		
I/O39	OUT	Fan	FAN_ON		
I/O40	OUT	7 segment code LED	Common cathode SEL0	Step motor control STEP_DR1	
I/O41	OUT	7 segment code LED	Common cathode SEL1	Step motor control STEP_DR2	
I/O42	OUT	7 segment LED	Common cathode SEL2	Step motor control STEP_DR3	
I/O43	OUT			Step motor control STEP_DR4	
I/O44	IN	SENSOR control			
I/O45	IN				
I/O46	IN	Fan unit	FAN_OUTPUT		
I/O47	IN	A/D unit	/AD_INTR		

IO-BANK4 IO-PORT5 is A444, corresponding to IO32~IO39 of IO-BANK4. Another IO PORT is the ninth IO PORT of each IO-BANK, which defines the output or input mode of each IO-PORT in this IO-BANK, and is described in the programs.

IO	IO Port	IO BANK	Sample value	IO BANK	Sample value	IO (16 bits)	IO (24 bits)	(32 bits)
00	1	1	A410 h	2	A420 h			
01								
02								
03								
04								
05								
06								
07								
08	2		A411 h		A421 h			
09								
10								
11								
12								
13								
14								
15								
16	3		A412 h		A422 h			
17								
18								
19								
20								
21								
22								
23								
24	4		A413 h		A423 h			
25								
26								
27								
28								
29								
30								
31								
32	5		A414 h		A424 h			
33								
34								
35								
36								
37								
38								
39								
40	6		A415 h		A425 h			
41								
42								
43								
44								
45								
46								
47								

6.6 Visual C/C++ simple program compiling description

For programs that use PCI-IO interface card to compile Visual C\C++, the following program shows that the user can go straight to execute its result as long as he changes the position of the program compiling zone with program code.

VC/C++ program code:

```
// start of the program//
#include <stdio.h>
#include "PlxApi.h"
#include "PciRegs.h"
#include "PlxInit.h"    // Need to add different Include File to individual program//

int main()       //main program//
{
     U8                  Revision;
     U16                 j;
     U32                 ChipType;
     U32                 LocalAddress=0;
     S8                  DeviceSelected;
     HANDLE              hDevice;
     RETURN_CODE         rc;
     DEVICE_LOCATION Device;
     IOP_SPACE           IopSpace;
     U32                 port,RegValue;
     U32                 buffer[64];        // define the parameters, add by yourself
upon lack of parameters //

     DeviceSelected = SelectDevice( &Device );      // Select interface card//
     rc = PlxPciDeviceOpen(&Device, &hDevice );    //PCI-IO card open action //

     port=PlxPciConfigRegisterRead(
          Device.BusNumber,
          Device.SlotNumber,
          CFG_BAR3,
          &rc);                        // read base address//

     buffer[0] = 0x00000000;
```

```
        port = port & ~(1<<0);      // return-to-zero and reset action//

        PlxChipTypeGet( hDevice, &ChipType, &Revision );    //IO Port setting //
              IopSpace = IopSpace0;
              IopSpace = IopSpace1;

< program writing zone > add program code here

}
        return 1;
}
        // end of the program//
```

Article 4 Basic PCI-LAB experiment examples

Basic I/O experiment conducts related interface test and experiment using the most basic and simplest element of interface and electronics. This article is divided into two parts: chapter seven and chapter eight. The pushbutton method is the most basic input elements in the chapter, while the derived keyboard is a little difficult, and LED is the principal basic output element. This chapter will focus on LED light and seven-segment display screen, while LED dot matrix and LED display will be described in detail in Chapter 8---advanced output experiment. Each chapter or section will first briefly introduce the characteristics of elements used for basic I/O experiment, and then describe the experiment and its application. Complete sample programs can be found in the sample program folder in the attached disk.

Chapter 7 Simple I/O experiment

LED is the most commonly seen electronic elements for display, which can usually be used to display binary data. Whereby LED ON stands for 1, OFF stands for 0. When different LEDs are arranged together in different manners, more information will be displayed, for instance, when seven LEDs are arranged in the manner shown in Figure 7-0-1, they can display Arabic numerals symbols. This method is widely used in electronic watches, electronic instrument, acoustic, which is commonly known as seven-segment display.

Usually, commercially available seven-segment displays have many specifications, which can mainly be classified as:

1. Distinguished by the overall dimension of LED display.
2. Distinguished by the colors emitted by the display, usually red, yellow, green, etc (blue LED is rarely seem). In addition, there are LED displays consisting of several different kinds of colors.
3. Distinguished by the display unit number contained in the display.
4. Distinguished by the connection methods of LED, which is divided into 2 types: common anode and common cathode.

In addition to specifications such as dimension, colors, brightness, common anode and common cathode are more important in selecting LED display. This characteristic affects the design of the LED display drive circuit. Figure 7-0-2 and 7-0-3 show the connection methods of common anode and common cathode LED displays: the anodes of all LEDs in the seven-segment display are connected together called the Common Anode. The cathode of all LEDs in the seven-segment display, connected together is called Common Cathode. Figure 7-0-4 shows the corresponding seven-segment LED display and Pins for PCI-LAB experiment plate, the format of bit is ABCDEFGH.

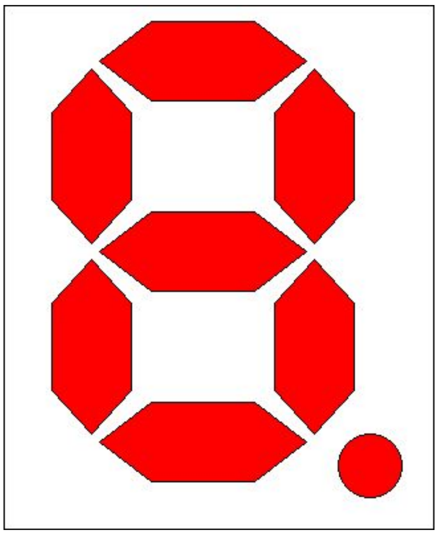


Figure 7-0-1 Seven-Segment LED display

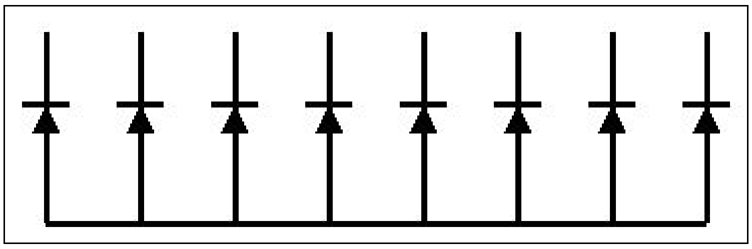


Figure 7-0-2 Common anode

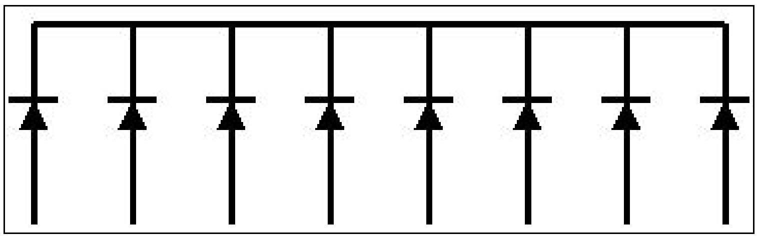


Figure 7-0-3 Common cathode

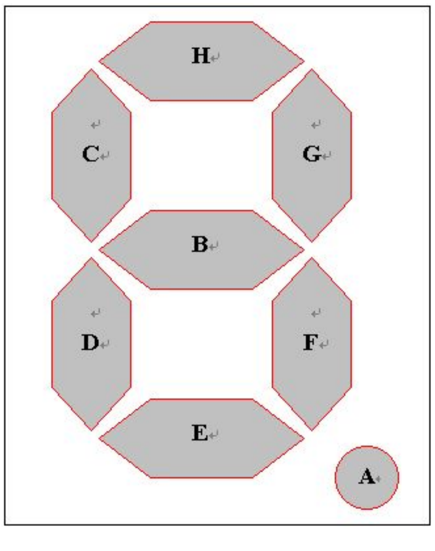


Figure 7-0-4 Seven-segment LED display and bits correspondence

To facilitate identification, English letter codes a~g are used to represent the seven LED. At the lower right corner of the seven-segment LED, there is another LED point, which is used to display decimal points, and is expressed as symbol dp. Due to the different Pins for seven-segment LED display manufactured by different manufacturers, confirming with triple use ammeter is the most reliable method. For digital ammeter confirming methods, if there is a specialized LED test file, it can directly use this test file, namely judge the positions relationships between its Pins and various displays form whether the LED is shining, and find out common contacts and its polarity. If there is not specialized an LED test file, it can be confirmed using common diode testing methods. As with pointer ammeter identifying method, transfer the ammeter to x1 files specially for testing resistance, connect the black testing rod to LED anode, and the red testing rod to LED cathode, then LED display Pins can be determined by whether the LED is shining.

Pushbutton switch is the most widely used signal input element, and the switches take the form of a mechanical switch, electronic switch and photoelectric switch, etc. These experiments use the relatively simple mechanical switch. Generally speaking, mechanical switch has two basic types: the first type is normally a closed switch, which is generally in a closed-circuit (short-circuited) status; it will not become an open circuit before the switch is pressed down. The other type is normally an open switch, which is usually in an open-circuited status and becomes a closed circuit when the switch is pressed down. Since the switch has two statuses: short-circuit and open circuit, it is easy to match with a digital circuit. When the switch is an open circuit (namely the switch is not pressed down), we can obtain the potential of H from the input end by means of a 4.7KΩ lifting resistance connected to ＋5V. Conversely, when the switch is connected to the grounding end due to closing (namely the switch is pressed down), then the potential of L will be obtained at the input/output end. Common pushbutton switches can be used as the simplest logic input device.

7.1 Seven-segment display experiment (DEBUG Mode)

Experiment purpose: use DEBUG Mode to enable the seven-segment display to show numbers 0~9.
Experiment module: single or six seven-segment LED displays (as is shown in Figure 7-1-1)

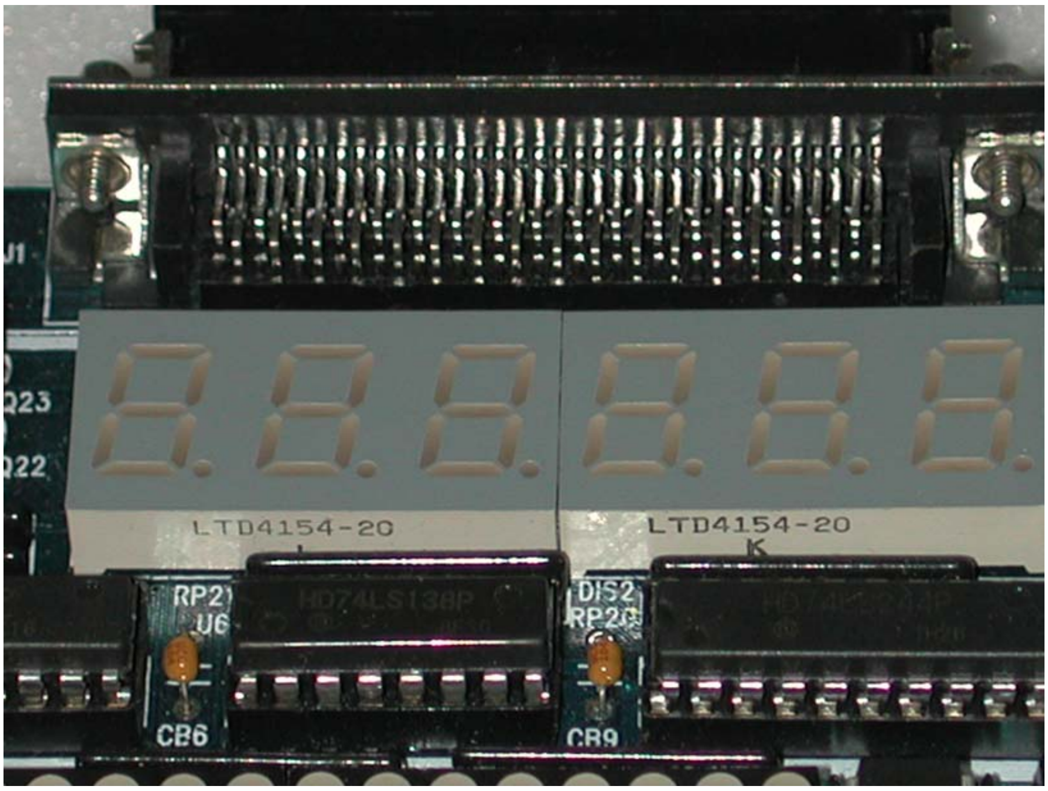


Figure 7-1-1 6 seven-segment LED display module

Part list:

- digital IC:
  - one 74LS138
  - one 74LS240
  - one 74LS244
- NPN-BJT:
  - six 2SC945
- Analog elements:
  - eight R 330Ω
  - six R 1KΩ
  - three C 0.1 uF
- Seven-segment LED display:
  - 2 three sets seven-segment LED display

Circuit diagram: (as is shown in Figure 7-1-2)

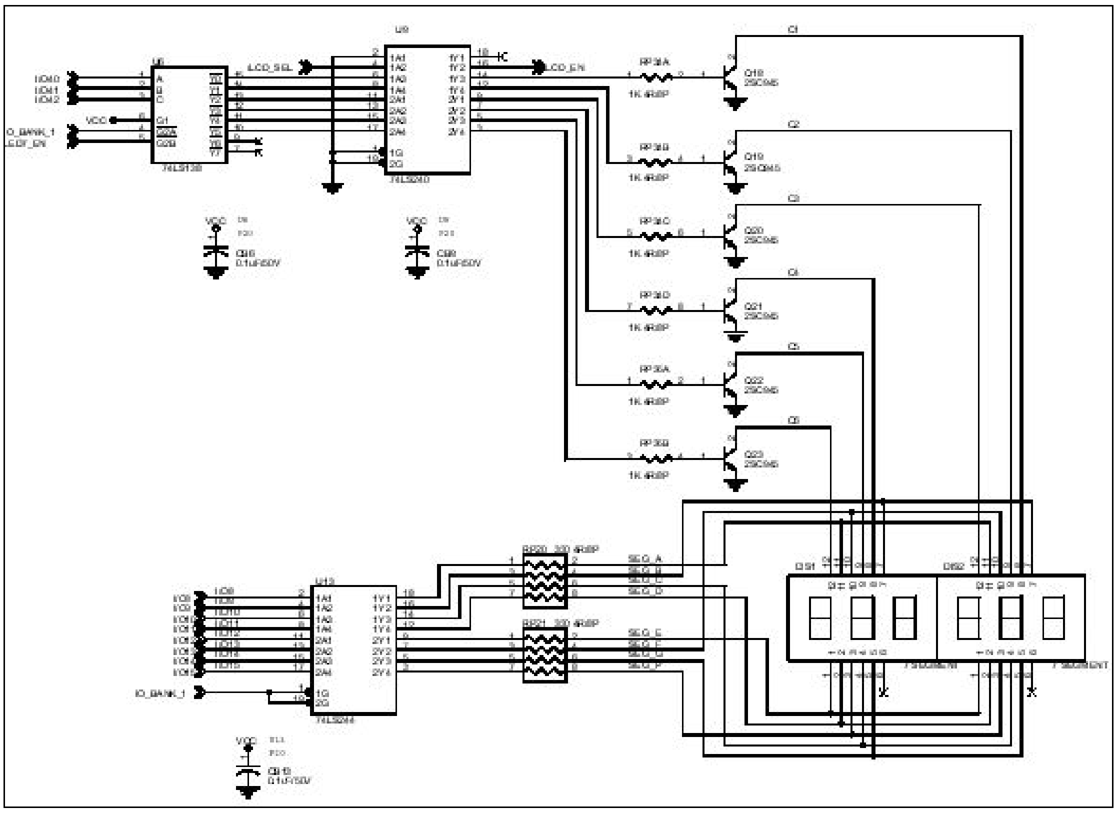


Figure 7-1-2 seven-segment LED display circuit diagram

Principle of experiment: 6 NPN-BJT control the display of 6 seven-segment LED display respectively, whose display bits are controlled by 74LS244 end. The selected is common cathode seven-segment LED display, 74LS138 and 74LS240 are responsible for enabling 3 bit interpretation controls of IO42~IO40 to select 7 segment display.

Seven-segment LED display selection

IO [42：40] = 000b=0h	The first seven-segment LED display
IO [42：40] = 001b=1h	The second seven-segment LED display
IO [42：40] = 010b=2h	The third seven-segment LED display
IO [42：40] = 011b=3h	The fourth seven-segment LED display
IO [42：40] = 100b=4h	The fifth seven-segment LED display
IO [42：40] = 101b=5h	The sixth seven-segment LED display

The numbers displayed：

IO [15：8] = 00111111b=3Fh display 0

IO [15：8] = 00000110b=06h display 1

IO [15：8] = 01011011b=5Bh display 2

IO [15：8] = 01001111b=4Fh display 3

IO [15：8] = 01100110b=66h display 4
IO [15：8] = 01101101b=6Dh display 5
IO [15：8] = 01111101b=7Dh display 6
IO [15：8] = 00000111b=07h display 7
IO [15：8] = 01111111b=7Fh display 8
IO [15：8] = 01101111b=6Fh display 9
IO [15：8] = 01110111b=77h display A
IO [15：8] = 01111100b=7Ch display B
IO [15：8] = 01011000b=58h display C
IO [15：8] = 01011110b=5Eh display D
IO [15：8] = 01111001b=79h display E
IO [15：8] = 01110001b=71h display F

Experiment procedures:

(1)Read PCI-IO interface card IO base address
(2)This function is in IO_BANK 1, so base address adds 10
(3)IO IO[42：40] is in the sixth byte, IO [15：8] is in the second byte
(4)Open Debug Mode to write directly
(5)Use instruction "-O (address) (data)" output
(6)First set all IO to be output port , then transmit data.

Example:

Suppose that base IO address is A400h, which is the first IO byte of IO_BANK 1, then the IO address of IO [42：40] is A415h, IO address of IO [15：8] is A411h, displaying 0~9 numbers as follows:

```
-O A418 00   \all IO_BANK 1 are output ports
-O A415 00   \ the first seven-segment LED display
-O A411 3F   \ display 0
-O A411 06   \ display 1
-O A411 5B   \ display 2
-O A411 4F   \ display 3
-O A411 66   \ display 4
-O A411 6D   \ display 5
-O A411 7D   \ display 6
-O A411 07   \ display 7
-O A411 7F   \ display 8
-O A411 6F   \display 9
```

When the experiment is in output port mode, the outputting of input setting -o A418 00 can be omitted.

7.2 Seven-segment display experiment (MASM)

Experiment code:

Continue 7.1 seven-segment display experiment (Debug Mode), use MASA to enable single seven-segment LED display to display numbers 0~9.
MASM program code:

```
. MODEL SMALL
.386
. STACK

. DATA

IO_PORT0             EQU     0A411H; io_bank_1 io-8~15
IO_PORT1             EQU     0A415H; io_bank_1 io-40~47
OUT_DISABLED         EQU     0A408H; io_bank_0
IO_PORT0D            EQU     0A400H; io_bank_0
IO_PORT1D            EQU     0A404H; io_bank_0

. CODE
BEGIN:
         PUSH     DS
         MOV      AX, 0
         PUSH     AX
         MOV      AX,@DATA
         MOV      DS,AX

DISP_BEGIN:


         MOV      DX,IO_PORT1              ; The first seven-segment code on
         MOV      AX,0000H                 ;0000_0000
         OUT      DX,AX
```

```
          CALL      COUNT_7SEG
          CALL      WAIT_1S

          MOV        AH,0BH                          ;press any key to end the program
          INT       21H                            ;interrupt vector 21h
          CMP        AL,0FFH
          JNZ       DISP_BEGIN

          JMP       EXIT                            ;end of the program

COUNT_7SEG:                                         ; display 0~9 numbers
          MOV        DX,IO_PORT0                     ; display O
          MOV        AX,003FH                       ;0011_1111
          OUT       DX,AX

          CALL      WAIT_1S

          MOV        DX,IO_PORT0                     ; display 1
          MOV        AX,0006H                       ;0000_0110
          OUT       DX,AX

          CALL      WAIT_1S

          MOV        DX,IO_PORT0                     ; display 2
          MOV        AX,005BH                       ;0101_1011
          OUT       DX,AX

          CALL      WAIT_1S

          MOV        DX,IO_PORT0                     ;display 3
          MOV        AX,004FH                       ;0100_1111
          OUT       DX,AX

          CALL      WAIT_1S

          MOV        DX,IO_PORT0                     ; display 4
          MOV        AX,0066H                       ;0110_0110
          OUT       DX,AX
```

```
        CALL      WAIT_1S

        MOV       DX,IO_PORT0                    ; display 5
        MOV       AX,006DH                       ;0110_1101
        OUT       DX,AX

        CALL      WAIT_1S

        MOV       DX,IO_PORT0                    ; display 6
        MOV       AX,007DH                       ;0111_1101
        OUT       DX,AX

        CALL      WAIT_1S

        MOV       DX,IO_PORT0                    ; display 7
        MOV       AX,0007H                       ;0000_0111
        OUT       DX,AX

        CALL      WAIT_1S

        MOV       DX,IO_PORT0                    ; display 8
        MOV       AX,007FH                       ;0111_1111
        OUT       DX,AX

        CALL      WAIT_1S

        MOV       DX,IO_PORT0                    ;display 9
        MOV       AX,006FH                       ;0110_1111
        OUT       DX,AX

        CALL      WAIT_1S

        RET

WAIT_1S:
        MOV       BX,0007FH
WAIT_LOOP:
```

```
            CALL       WAIT_1MS
            DEC        BX
            CMP        BX,0000H
            JBE        WAIT_1S_EXIT
            LOOP       WAIT_LOOP
WAIT_1S_EXIT:
            RET

WAIT_1MS:
            MOV        CX,03FFFH
WAIT_LOOP1:
            MOV        BX,BX
            MOV        BX,BX
            MOV        BX,BX
            MOV        BX,BX
            MOV        BX,BX
            MOV        BX,BX
            MOV        BX,BX
            LOOP       WAIT_LOOP1
            RET

EXIT:
            MOV        DX,OUT_DISABLED                ;clear all settings
            MOV        AX,0000H
            OUT        DX,AX
            MOV        DX,IO_PORT0D
            MOV        EAX,00000000H
            OUT        DX,EAX
            MOV        DX,IO_PORT1D
            MOV        AX,0000H
            OUT        DX,AX
            MOV        AH,4CH
            INT        21H
            END        BEGIN
```

Due to the high executing speed of assembly languages, if it is found that the numbers are displayed too fast, you yourself can add more waiting time (WAIT_1S).

7.3 Seven-segment display experiment (VC/C++)

Experiment purpose: Continue 7.1 seven-segment display experiment (Debug Mode),use VC/C++ to enable sing seven-segment LED display to display numbers 0~9.

VC/C++ program code

```
// Set all IO BANK 1 to be output //
        RegValue = 0x0000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x18,
                BitSize32,
                &RegValue);

// Set the seven-segment LED on the extreme right //
        RegValue = 0x00000000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
// display"0"//
        RegValue = 0x00003F00;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
/ display"1"//
        RegValue = 0x00000600;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
// display"2"//
```

```
          RegValue = 0x00005B00;
          rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x10,
                    BitSize32,
                    &RegValue);
//display "3"//
          RegValue = 0x00004F00;
          rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x10,
                    BitSize32,
                    &RegValue);
//display "4"//
          RegValue = 0x00006600;
          rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x10,
                    BitSize32,
                    &RegValue);
//display "5"//
          RegValue = 0x00006D00;
          rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x10,
                    BitSize32,
                    &RegValue);
//display "6"//
          RegValue = 0x00007D00;
          rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x10,
                    BitSize32,
                    &RegValue);
//display "7"//
          RegValue = 0x00000700;
          rc=PlxIoPortWrite(
                    hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//display "8"//
        RegValue = 0x00007F00;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//display "9"//
        RegValue = 0x00006F00;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
```

7.4 Drive multi-sets seven-segment LED display experiment (MASM):

Experiment purpose: Enable multi-sets seven-segment LED display to display 0~9 numbers by writing MASA program.

MASM program codes

```
.MODEL   SMALL
.386
.STACK

.DATA

IO_PORT0            EQU     0A411H  ;  io_bank_1 io-8~15
IO_PORT1            EQU     0A415H  ;  io_bank_1 io-40~47
OUT_DISABLED        EQU     0A408H  ;  io_bank_0
IO_PORT0D           EQU     0A400H  ;  io_bank_0
IO_PORT1D           EQU     0A404H  ;  io_bank_0
```

```
.CODE
BEGIN:
            PUSH      DS
            MOV        AX,0
            PUSH      AX
            MOV        AX,@DATA
            MOV        DS,AX

DISP_BEGIN:



            MOV        DX,IO_PORT1                    ; the first seven-segment code on
            MOV        AX,0000H                      ;0000_0000
            OUT       DX,AX
            CALL      COUNT_7SEG
            CALL      WAIT_1S

            MOV        DX,IO_PORT1                    ; the second seven-segment code on
            MOV        AX,0001H                      ;0000_0001
            OUT       DX,AX
            CALL      COUNT_7SEG
            CALL      WAIT_1S

            MOV        DX,IO_PORT1                    ; the third seven-segment code on
            MOV        AX,0002H                      ;0000_0010
            OUT       DX,AX
            CALL      COUNT_7SEG
            CALL      WAIT_1S

            MOV        DX,IO_PORT1                    ; the fourth seven-segment code on
            MOV        AX,0003H                      ;0000_0011
            OUT       DX,AX
            CALL      COUNT_7SEG
            CALL      WAIT_1S

            MOV        DX, IO_PORT1                   ; the fifth 7-segment code on
            MOV        AX, 0004H                     ;0000_0100
```

```
          OUT      DX, AX
          CALL     COUNT_7SEG
          CALL     WAIT_1S

          MOV       DX,IO_PORT1                  ; The sixth seven-segment code on
          MOV       AX,0005H                    ;0000_0101
          OUT      DX,AX
          CALL     COUNT_7SEG
          CALL     WAIT_1S



          MOV       AH,0BH                       ;Press any key to end the program
          INT      21H                        ; Interrupt vector 21h
          CMP       AL,0FFH
          JNZ      DISP_BEGIN

          JMP      EXIT                        ;end of the program

COUNT_7SEG:                                     ;display 0~9 number
          MOV       DX,IO_PORT0                  ;display O
          MOV       AX,003FH                    ;0011_1111
          OUT      DX,AX
          CALL     WAIT_1S


          MOV       DX,IO_PORT0                  ;display 1
          MOV       AX,0006H                    ;0000_0110
          OUT      DX,AX
          CALL     WAIT_1S

          MOV       DX,IO_PORT0                  ;display 2
          MOV       AX,005BH                    ;0101_1011
          OUT      DX,AX
          CALL     WAIT_1S

          MOV       DX,IO_PORT0                 display 3
          MOV       AX,004FH                    ;0100_1111
```

```
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0                ;display   4
        MOV      AX,0066H                  ;0110_0110
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0                ;display 5
        MOV      AX,006DH                   ;0110_1101
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0                ;display 6
        MOV      AX,007DH                   ;0111_1101
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0                ;display 7
        MOV      AX,0007H                  ;0000_0111
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0               display 8
        MOV      AX,007FH                  ;0111_1111
        OUT     DX,AX
        CALL    WAIT_1S

        MOV      DX,IO_PORT0                ;display   9
        MOV      AX,006FH                  ;0110_1111
        OUT     DX,AX
        CALL    WAIT_1S

        RET

WAIT_1S:
        MOV      BX,0007FH
WAIT_LOOP:
```

```
          CALL      WAIT_1MS
          DEC       BX
          CMP       BX,0000H
          JBE       WAIT_1S_EXIT
          LOOP      WAIT_LOOP
WAIT_1S_EXIT:
          RET

WAIT_1MS:
          MOV       CX,03FFFH
WAIT_LOOP1:
          MOV       BX,BX
          MOV       BX,BX
          MOV       BX,BX
          MOV       BX,BX
          MOV       BX,BX
          MOV       BX,BX
          MOV       BX,BX
          LOOP      WAIT_LOOP1
          RET

EXIT:
          MOV       DX,OUT_DISABLED                   ;clear all settings
          MOV       AX,0000H
          OUT       DX,AX
          MOV       DX,IO_PORT0D
          MOV       EAX,00000000H
          OUT       DX,EAX
          MOV       DX,IO_PORT1D
          MOV       AX,0000H
          OUT       DX,AX

          MOV       AH,4CH
          INT       21H
          END       BEGIN
```

7.5 Drive multi-sets seven-segment LED display experiment (VC/C++)

Experiment purpose: Write VC/C++ program to enable several group of seven-segment LED displays to display 0~9 numbers.

VC/C++ program code

```
// Set the first seven-segment LED display on the right //
    RegValue = 0x00000000;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x14,
        BitSize32,
        &RegValue);

// Set the second seven-segment LED display on the right //
    RegValue = 0x00000001;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x14,
        BitSize32,
        &RegValue);

// Set the third seven-segment LED display on the right //
    RegValue = 0x00000002;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x14,
        BitSize32,
        &RegValue);

// Set the fourth seven-segment LED display on the right//
    RegValue = 0x00000003;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x14,
        BitSize32,
        &RegValue);
```

```
// Set the fifth seven-segment LED display on the right //

        RegValue = 0x00000004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);

// Set the sixth seven-segment LED display on the right //
        RegValue = 0x00000005;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
```

7-6 Buzzer experiment (Debug Mode & MASM)

Experiment purpose: Let the buzzer emit a sound with Debug Mode and MASM. Experiment module: As is shown in Figure 7-1-6 is the buzzer and drive circuit on PCI-LAB.

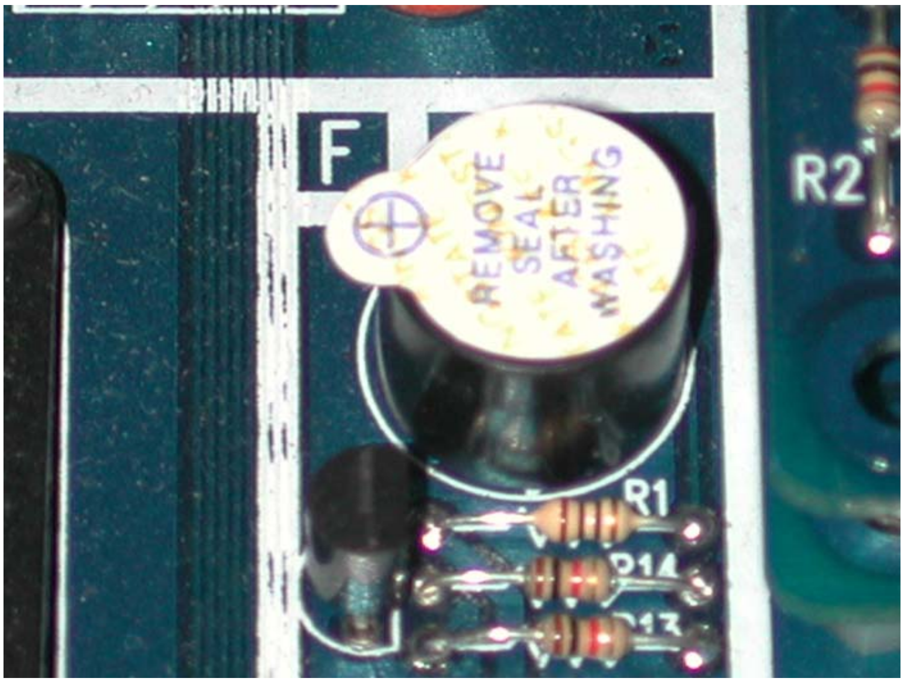


Figure 7-6-1 experiment modules

Part list:

Buzzer:

one KSS_1206

NPN-BJT：

one 8050

Resistance:

one 10Ω

two 1kΩ

Principle of experiment: The starting output signal is low, drive this signal to high, then to low, so that a sound can be emitted.

Circuit diagram： (as is shown in Figure 7-6-2)

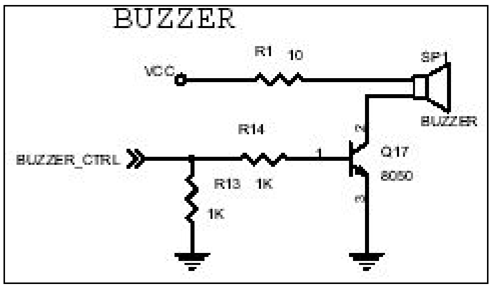


Figure 7-6-2 Circuit diagram

Experiment procedures: (1) Read PCI-IO interface card IO base address.

(2) This function is in IO_BANK 1, so base address adds one.

(3) IO [38] is in the fifth byte.

(4) Open Debug Mode to write directly.

(5) Use instruction "-O (address) (data)" output.

(6) First set IO used to be output port, then transmit data.

Example:

```
-o A418 00
-o A414 00
-o A414 40
-o A414 00
```

MASM program code

```
.MODEL SMALL
```

```
.386
.STACK

.DATA


IO_PORT1                EQU     0A414H ;   io_bank_1   io-32~39
OUT_DISABLED             EQU     0A408H ;   io_bank_0
IO_PORT0D                EQU     0A400H ;   io_bank_0
IO_PORT1D                EQU     0A404H ;   io_bank_0

.CODE
BEGIN:
        PUSH    DS
        MOV      AX,0
        PUSH    AX
        MOV      AX,@DATA
        MOV      DS,AX

DISP_BEGIN:


        MOV      DX,IO_PORT1
        MOV      AX,00H
        OUT     DX,AX
        CALL    WAIT_1MS
        MOV      DX,IO_PORT1
        MOV      AX,40H
        OUT     DX,AX
        CALL    WAIT_1MS

        MOV      AH,0BH                  press any key to end the program
        INT     21H
        CMP      AL,0FFH
        JNZ     DISP_BEGIN

        JMP     EXIT                     end of the program

WAIT_1MS:
        MOV     CX,03FFFH
WAIT_LOOP1:
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        LOOP    WAIT_LOOP1
        RET

EXIT:
        MOV     DX,OUT_DISABLED                 ;clear all settings
        MOV     AX,0000H
        OUT     DX,AX
        MOV     DX,IO_PORT0D
        MOV     EAX,00000000H
        OUT     DX,EAX
        MOV     DX,IO_PORT1D
        MOV     AX,0000H
        OUT     DX,AX

        MOV     AH,4CH
        INT     21H
        END     BEGIN
```

7.7 Buzzer experiment (VC/C++)

Experiment purpose: Using VC/C++ to enable the buzzer to emit a sound.

VC/C++ program code

```
//low//
     RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);
//high//
    RegValue = 0x0040;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);
```

//It is required to output low-high-low in sequence continuously to have sound//

7.8 Pushbutton switches input experiment (Debug Mode & MASM)

Experiment purpose: Practising pressing pushbutton switch and output its result with LED display.

Experiment module: as is shown in Figure 7-8-1 below.

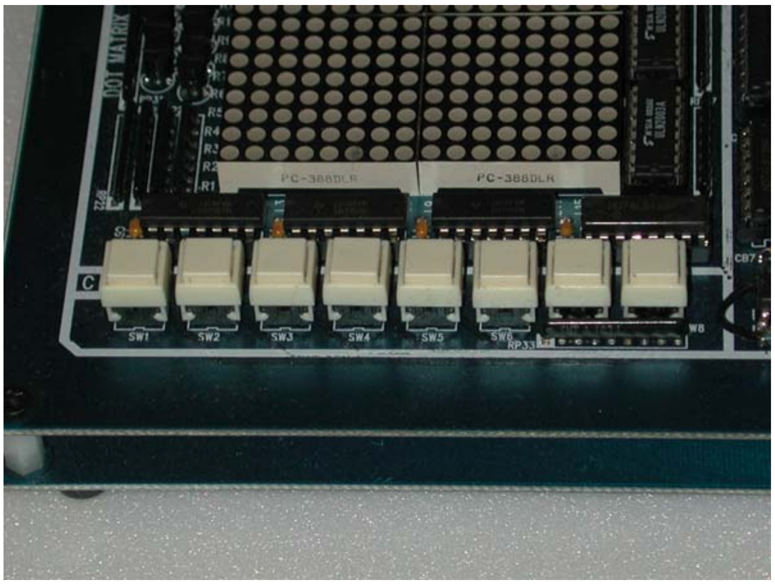


Figure 7-8-1 Pushbutton switches module

Part list:

eight pushbutton switches

Resistance

one 10kΩ resistance network

logic IC：

one 74LS244

Capacitance:

one 0.1uF

Circuit diagram: as is shown in Figures 7-8-2 and 7-8-3 below.

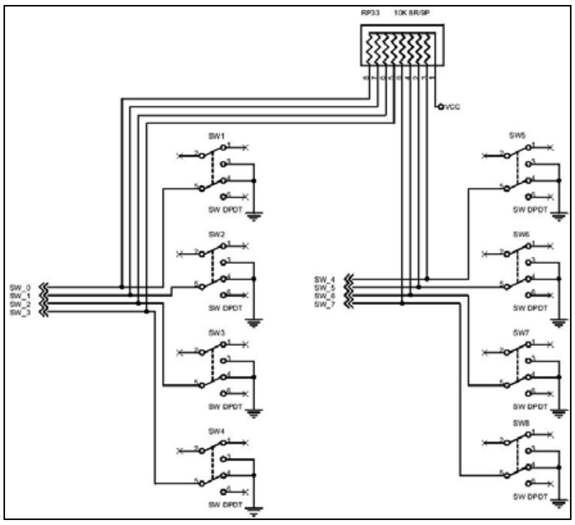


Figure 7-8-2 Pushbutton input switch connection diagram

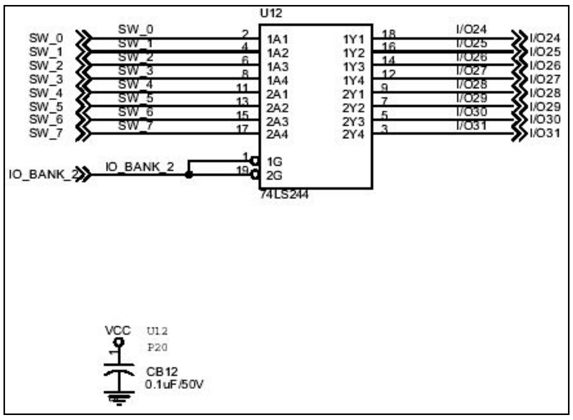


Figure 7-8-3 Data latch circuit

Principle of experiment: Read the data of IO port

SW_1	0000_0001	01h
SW_2	0000_0010	02h
SW_3	0000_0100	04h
SW_4	0000_1000	08h
SW_5	0001_0000	10h

SW_6	0010_0000	20h
SW_7	0100_0000	40h
SW_8	1000_0000	80h

Experiment procedure: (1)Set IO port to output mode.
(2)read the data of IO_BANK2 IO_PORT4

Sample:

```
-o A428 D0          Set to be output mode
-i   A423
<display result>
-o A428 00
```

MASM program code:

```
.MODEL    SMALL
.386
.STACK

.DATA
OUT_DISABLE              EQU       0A418H
IO_PORT0               EQU       0A410H
IO_PORT1               EQU       0A414H
IO_PORT2               EQU       0A412H
IO_PORT3               EQU       0A413H
OUT_DISABLEA             EQU       0A428H
IO_PORT0A              EQU       0A420H
IO_PORT1A              EQU       0A424H
IO_PORT2A              EQU       0A423H
OUT_DISABLED             EQU       0A408H
IO_PORT0D              EQU       0A400H
IO_PORT1D              EQU       0A404H
IO_PORT2D              EQU       0A402H
TEST_UNIT              DB        10H
MAT_ROW                  DB        01H
MAT_COL                 DB        00H
MAT_COUNT                DB        00H
```

```
TEMP                    DW      0H
TEMP_LOOP               DW      0H
LCD_TEMP                DW      0H
TEMP_LCD1               DW      0H
TEMP_LCD2               DD      0H

.CODE
BEGIN:
        PUSH    DS
        MOV     AX,0
        PUSH    AX
        MOV     AX,@DATA
        MOV     DS,AX
PUSH_BOTTOM:
        MOV     DX, OUT_DISABLEA        ; Set I/O 16~31 to INPUT
        MOV     AX, 00D0H
        OUT     DX,AX

        MOV     TEMP,0FFH
BUT_BEGIN:
        MOV     DX,IO_PORT1A            ; Set the F column of MATRIX TO to display
        MOV     AX,000FH
        OUT     DX,AX

        MOV     DX,IO_PORT2A
TEST_LOOP:
        IN      AX,DX           ;Read back PUSH_BOTTON value
        XOR     AH,AH

        MOV   DX,IO_PORT0A    ;Display PUSH_BOTTON in the bottom column of Matrix
        OUT     DX,AX

        CALL    WAIT_1MS

        CMP     AX, 00FFH                       ; ended when all one.
        JZ      PUSH_EXIT

        MOV     AH, 0BH                         ; press any key to end the program
```

```
          INT       21H
          CMP        AL, 0FFH
          JNZ       BUT_BEGIN
          JMP       EXIT
PUSH_EXIT:
          CALL       WAIT_3S                              ; PUSH_BOTTOM end of program
MAT_COLA                                                   : MATRIX COMMAND
          MOV        DX, IO_PORT0A
          MOV        EAX, 0000FFFFH
          OUT       DX, EAX
          CALL       WAIT_1S
          RET

MAT_ROWA:
          MOV        DX, IO_PORT1A
          OUT       DX, AX
          CALL       WAIT_1S
          RET

MAT_COU:
          MOV        DX, IO_PORT0A
          MOV        EAX, 0000FFFFH
          OUT       DX, EAX
MAT_A:
          MOV        DX, IO_PORT1A
          MOV        AL, MAT_COUNT
          OUT       DX, AX
          CALL       WAIT_1MS
          INC       MAT_COUNT
          CMP        MAT_COUNT, 10H
          JE        MAT_AEXIT
          JMP       MAT_A

MAT_AEXIT:
          RET
          JMP       EXIT
WAIT_3S:
          MOV        TEMP_LOOP, 07H
```

```
WAIT_3S_LOOP:
            CALL       WAIT_2S
            DEC        TEMP_LOOP
            CMP        TEMP_LOOP, 00H
            JBE        WAIT_3S_EXIT
            JMP        WAIT_3S_LOOP
WAIT_3S_EXIT:
            RET

WAIT_2S:
            CALL       WAIT_1S
            CALL       WAIT_1S
            CALL       WAIT_1S
            CALL       WAIT_1S
            CALL       WAIT_1S
            CALL       WAIT_1S
            RET
WAIT_1S:
            MOV        BX, 0007FH
WAIT_LOOP:
            CALL       WAIT_1MS
            DEC        BX
            CMP        BX, 0000H
            JBE        WAIT_1S_EXIT
            LOOP       WAIT_LOOP
WAIT_1S_EXIT:
            RET

WAIT_1MS:
            MOV        CX, 03FFFH
WAIT_LOOP1:
            MOV        BX, BX
            MOV        BX, BX
            MOV        BX, BX
            MOV        BX, BX
            MOV        BX, BX
            MOV        BX, BX
            MOV        BX, BX
            LOOP        WAIT_LOOP1
            RET

WAIT_2MS:
            MOV         CX, 07FFH
WAIT_LOOP2:
            MOV         BX, BX
            MOV         BX, BX
            MOV         BX, BX
            MOV         BX, BX
            MOV         BX, BX
            MOV         BX, BX
            MOV         BX, BX
            LOOP        WAIT_LOOP2
            RET
EXIT:
            MOV         DX, OUT_DISABLED                ; clear all settings
            MOV         AX, 0000H
            OUT         DX, AX
            MOV         DX, IO_PORT0D
            MOV         EAX, 00000000H
            OUT         DX, EAX
            MOV         DX, IO_PORT1D
            MOV         AX, 0000H
            OUT         DX, AX

            MOV         AH, 4CH
            INT         21H
            END         BEGIN
```

7.9 Pushbutton switch input experiment (VC/C++)

Experiment purpose: Complete the section 7.8 experiment by means of VC/C++.

VC/C++ program code

```
//set IO needed to be input //
       RegValue = 0x00D0;
```

```
        Arc=PlxIoPortWrite (
                hDevice,          Port + 0x28,
                BitSize32,
                &RegValue);

//set 8 LED to return to 0//
        RegValue = 0x0000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x20,
                BitSize32,
                &RegValue);

//Read input value//
        rc=PlxIoPortRead(
                hDevice,
                port + 0x23,
                BitSize32,
                &RegValue);

//output to LED//
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x20,
                BitSize32,
                &RegValue);
```

Chapter 8 Motor and resistance heater

Table 8-0-1 contains the comparison between the step motor and DC motor, regarding the basic principles of step motor and DC motor. It can be found in relevant literature related to motor mechanics, so we do not describe it in detail in this manual.

Table 8-0-1 Comparison between step motor and DC motor

	Step motor (open circuit control)	DC motor (closed-circuit control)
Step angle	Depends on the motor	Vary with rotary coder and circuit
Maximum speed	About 3000 rpm in case of 200 step motor	6000 rpm
Start, stop and Repeat speed	Variable, maximum repeating speed is proportional to inertia, friction and the ambient temperature of load. Resonance frequency is not stable in cases of low torque.	Variable, maximum repeating speed is Proportional to inertia, friction and the ambient temperature of the load. Resonance gains of the system are limited.
Precision	Depends on the design of motor without accumulated errors.	Depends on servo circuit.
Cost	Open circuit control is cheaper.	High prices
Still toque	Large torque	Depends on the feedback circuit method.
System reliability	Depends on the life of bearing.	Depends on the life of brush.
Load inertia	Smaller inertia is better.	Smaller inertia is better.
Single step response	Prone to vibrate	Less prone to vibrate

8.1 DC motor experiment (Debug Mode & MASM)

Experiment purpose: Testing the operating and stopping of DC fan.

Experiment module: Figure 8-1-1 below shows DC fan module for the experiment.

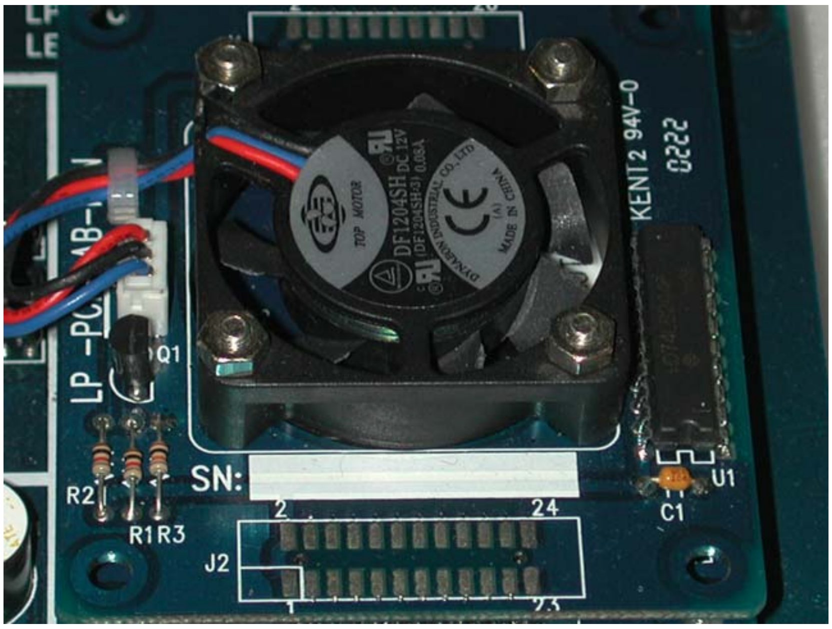


Figure 8-1-1 DC fan modules

Part list:

- one DC fan
- NPN-BJT：
  - one 8050
- Capacitance:
  - one 1kΩ
  - one 10kΩ
- Digital logic IC：
  - one 74LS244
- Capacitance:
  - one 0.1uF

Circuit diagram: consisting the two Figures below: 8-1-2 and 8-1-3

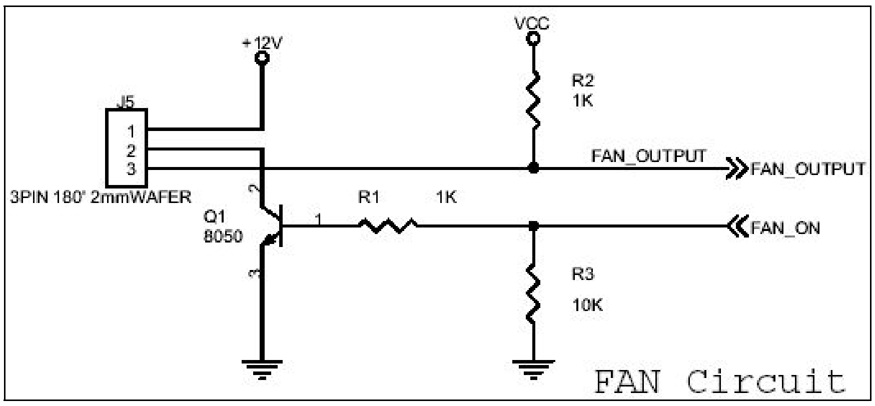


Figure 8-1-2 Fan switch circuit

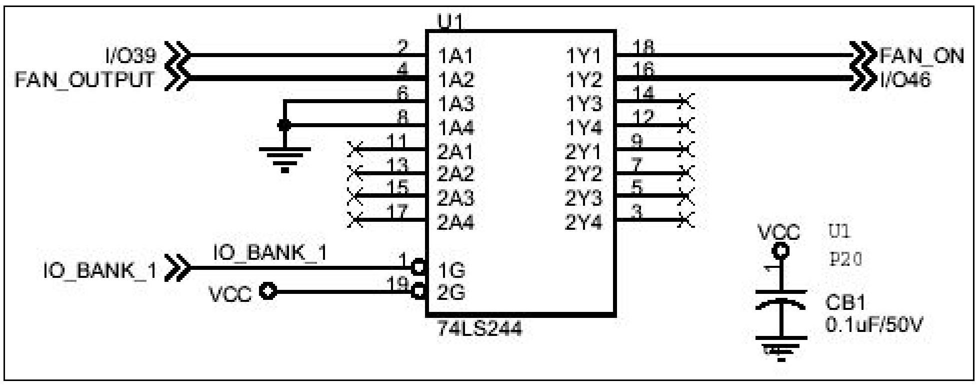


Figure 8-1-3 DC fan moduleIO latch circuit

Principle of experiment: FAN_ON high, the fan works, FAN_ON low, the fan stops working

Experiment procedures:

(1) Read PCI-IO interface IO base address.
(2) This function is in IO_BANK 1, so base address adds 1.
(3) I [39] is the fifth byte.
(4) Open Debug Mode to write directly
(5) Use instruction "-O (address) (data)"
(6) First set the IO used to output port, then transmit data.

Sample:

```
-o A414 80     1000_0000    The fan works
-o A414 00     0000_0000    The fan stops
```

MASM program code:

```
.MODEL    SMALL
.386
.STACK

. DATA

IO_PORT1                    EQU      0A414H; io_bank_1 io-32~39
OUT_DISABLED                  EQU      0A408H; io_bank_0
IO_PORT0D                    EQU      0A400H; io_bank_0
IO_PORT1D                    EQU      0A404H; io_bank_0

.CODE
BEGIN:
          PUSH      DS
          MOV       AX,0
          PUSH      AX
          MOV       AX,@DATA
          MOV       DS,AX

DISP_BEGIN:

          MOV       DX,IO_PORT1               ;fan on
          MOV       AX,80H                    ;1000_0000
          OUT       DX,AX

          MOV       AH,0BH                    ;Press any key to end the program
          INT       21H                      ;interrupt vector 21h
          CMP       AL,0FFH
          JNZ       DISP_BEGIN

          JMP       EXIT                      ;end of the program

EXIT:
          MOV       DX,OUT_DISABLED                     ;clear all settings
          MOV       AX,0000H
          OUT       DX,AX
          MOV       DX,IO_PORT0D
          MOV       EAX,00000000H
          OUT        DX,EAX
          MOV         DX,IO_PORT1D
          MOV         AX,0000H
          OUT        DX,AX
          MOV         AH,4CH
          INT        21H
          END        BEGIN
```

8.2 DC motor experiment (VC/C++)

Experiment purpose: Write programs with VC/C++ to drive DC fan motor.

VC/C++ program code:

```
//turn on the fan//
     RegValue = 0x0080;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);
//turn off the fan//
     RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);

//set fan input//
     RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x18,
            BitSize32,
            &RegValue);

//read fan input//
```

```
rc=PlxIoPortRead(
        hDevice,
        port + 0x15,
        BitSize32,
        &RegValue);
```

8.3 Step motor experiment (Debug Mode)

Experiment purpose: Use Debug Mode to drive step motor.

Experiment module: Figure 8-3-1 below shows step motor module for the experiment.

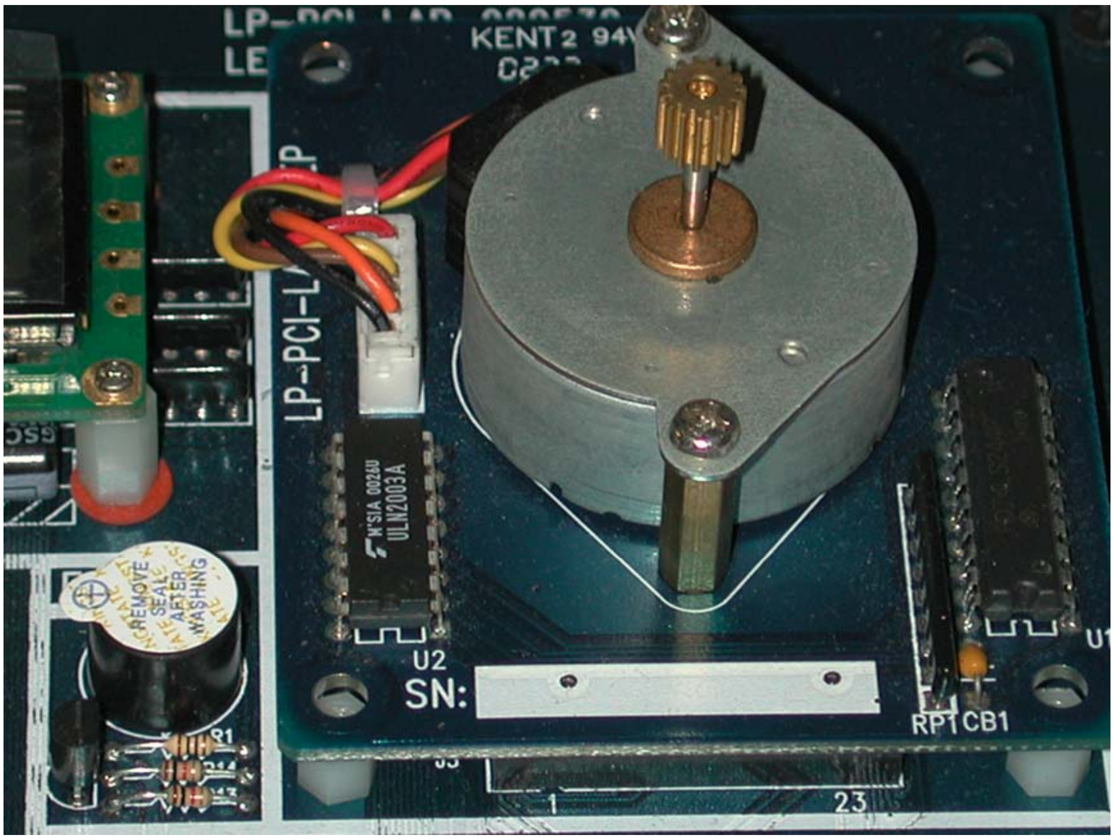


Figure 8-3-1 step motor modules

Part list

- a four phase step motor
- Digital logic IC:
    - one 74LS244
- Current drive IC:
    - one ULN2003

Resistance:

one 2.2kΩ

Capacitance:

1 0.1uF

Circuit diagram: Figure 8-3-2and 8-3-3 are step motor drive circuit.
According to the excitation phase of different step motors, drive with 1-2-3-4 phase respectively to achieve forward and reverse.

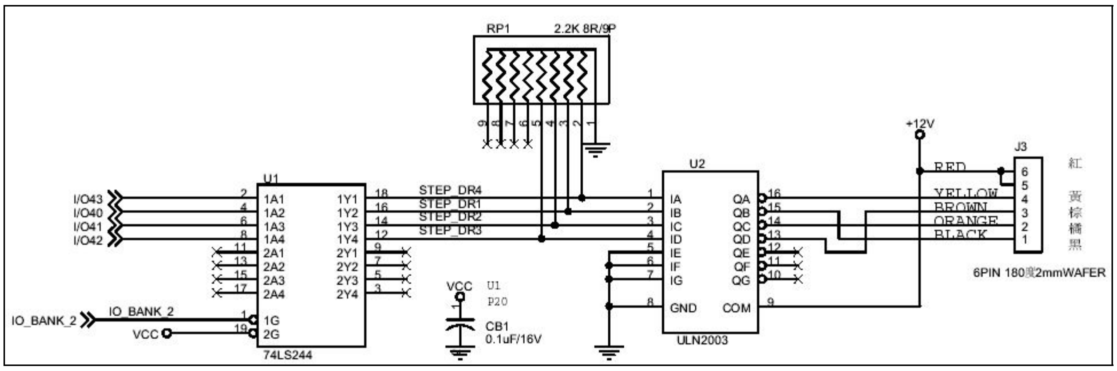


Figure 8-3-2 Step motor drive circuits 1

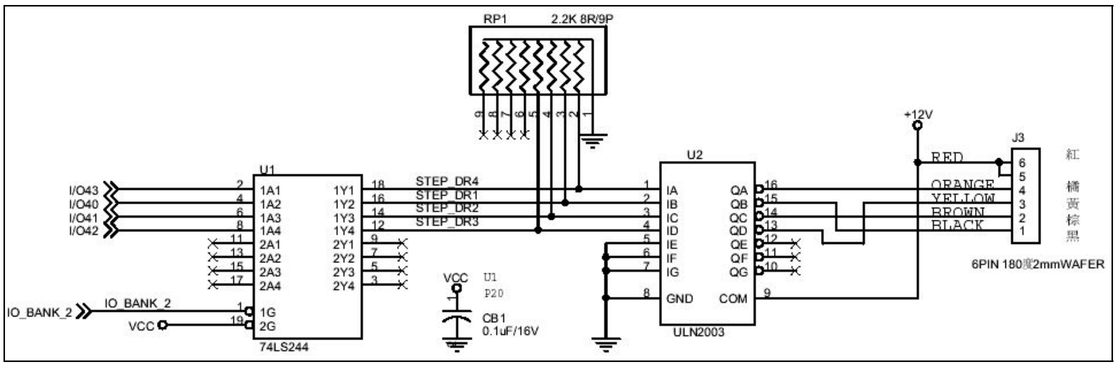


Figure 8-3-2 Step motor drive circuits 2

Principal of experiment: Output signal from A425h port

IO40－IO43 are the first through the fourth phases of the step motor respectively.

0001 the first phase of step motor

0010 the second phase of step motor

0100 the third phase of step motor

1000 the fourth phase of step motor

Example:

-O A425 01

```
-O A425 02
-O A425 04
-O A425 08
```

8.4 Step motor experiment (MASM)

Experiment purpose: Write programs with MASA to make the step motor to forward or reverse.
Forward:

```
.MODEL   SMALL
.386
.STACK

.DATA


IO_PORT1                    EQU       0A425H ;   io_bank_2   io-40~47
OUT_DISABLED                  EQU       0A408H ;   io_bank_0
IO_PORT0D                    EQU       0A400H ;   io_bank_0
IO_PORT1D                    EQU       0A404H ;   io_bank_0

.CODE
BEGIN:
          PUSH      DS
          MOV        AX,0
          PUSH      AX
          MOV        AX,@DATA
          MOV        DS,AX

DISP_BEGIN:



          MOV        DX,IO_PORT1                  ;1
          MOV        AX,01H                     ;0000_0001
          OUT       DX,AX
          CALL      WAIT_1S
```

```
            CALL      WAIT_1S
            MOV       DX,IO_PORT1               ;4
            MOV       AX,08H                ;0000_1000
            OUT       DX,AX
            CALL      WAIT_1S
            CALL      WAIT_1S
            MOV       DX,IO_PORT1               ;3
            MOV       AX,04H                ;0000_0100
            OUT       DX,AX
            CALL      WAIT_1S
            CALL      WAIT_1S
            MOV       DX,IO_PORT1               ;2
            MOV       AX,02H                ;0000_0010
            OUT       DX,AX
            CALL      WAIT_1S
            CALL      WAIT_1S

            MOV       AH,0BH                    ;Press Any key to end the program
            INT       21H
            CMP       AL,0FFH
            JNZ       DISP_BEGIN

            JMP       EXIT                   ;end of the program

WAIT_1S:
            MOV       BX,0007FH
WAIT_LOOP:
            CALL      WAIT_1MS
            DEC       BX
            CMP       BX,0000H
            JBE       WAIT_1S_EXIT
            LOOP      WAIT_LOOP
WAIT_1S_EXIT:
            RET



WAIT_1MS:
            MOV       CX,03FFFH
```

```
WAIT_LOOP1:
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            LOOP      WAIT_LOOP1
            RET

EXIT:
            MOV       DX,OUT_DISABLED                    ;clear all settings
            MOV       AX,0000H
            OUT       DX,AX
            MOV       DX,IO_PORT0D
            MOV       EAX,00000000H
            OUT       DX,EAX
            MOV       DX,IO_PORT1D
            MOV       AX,0000H
            OUT       DX,AX

            MOV       AH,4CH
            INT       21H
            END       BEGIN
```

Reverse:

```
.MODEL   SMALL
.386
.STACK

.DATA


IO_PORT1                EQU       0A425H ;   io_bank_2   io-40~47
OUT_DISABLED            EQU       0A408H ;   io_bank_0
IO_PORT0D               EQU       0A400H ;   io_bank_0
```

```
IO_PORT1D                  EQU        0A404H ;   io_bank_0

.CODE
BEGIN:
          PUSH     DS
          MOV      AX,0
          PUSH     AX
          MOV      AX,@DATA
          MOV      DS,AX

DISP_BEGIN:




          MOV      DX,IO_PORT1                  ;1
          MOV      AX,01H                    ;0000_0001
          OUT      DX,AX
          CALL     WAIT_1S
          MOV      DX,IO_PORT1                  ;2
          MOV      AX,02H                    ;0000_0010
          OUT      DX,AX
          CALL     WAIT_1S
          MOV      DX,IO_PORT1                  ;3
          MOV      AX,04H                    ;0000_0100
          OUT      DX,AX
          CALL     WAIT_1S
          MOV      DX,IO_PORT1                  ;4
          MOV      AX,08H                    ;0000_1000
          OUT      DX,AX
          CALL     WAIT_1S

          MOV      AH,0BH                       ;Press Any key to end the program
          INT      21H
          CMP      AL,0FFH
          JNZ      DISP_BEGIN

          JMP      EXIT                      ;end of the program
```

```
WAIT_1S:
        MOV     BX,0007FH
WAIT_LOOP:
        CALL    WAIT_1MS
        DEC     BX
        CMP     BX,0000H
        JBE     WAIT_1S_EXIT
        LOOP    WAIT_LOOP
WAIT_1S_EXIT:
        RET


WAIT_1MS:
        MOV     CX,03FFFH
WAIT_LOOP1:
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        MOV     BX,BX
        LOOP    WAIT_LOOP1
        RET

EXIT:
        MOV     DX,OUT_DISABLED                 ;clear all settings
        MOV     AX,0000H
        OUT     DX,AX
        MOV     DX,IO_PORT0D
        MOV     EAX,00000000H
        OUT     DX,EAX
        MOV     DX,IO_PORT1D
        MOV     AX,0000H
        OUT     DX,AX

        MOV     AH,4CH
        INT     21H
```

```
END     BEGIN
```

8.5 Step motor experiment (VC/C++)

Experiment purpose: Program with VC/C++ program languages to make the step motor to forward or reverse.

Forward:

```
//Set all IO Port to be output//
      RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x11,
            BitSize32,
            &RegValue);
// Reverse action ( engineering plate) //
     RegValue = 0x0001;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0008;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0004;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0002;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
```

Reverse:

```
/Reverse action ( engineering plate)//
     RegValue = 0x0001;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0002;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0004;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
     RegValue = 0x0008;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x25,
            BitSize32,
            &RegValue);
```

8.6 Resistance heater experiment

Experiment purpose: enable resistance heater to heat

Note: please do not heat for more than 5 minutes or cooperate with DC fan to provide coolant.

Experiment module: As is shown in Figure 8-6-1.

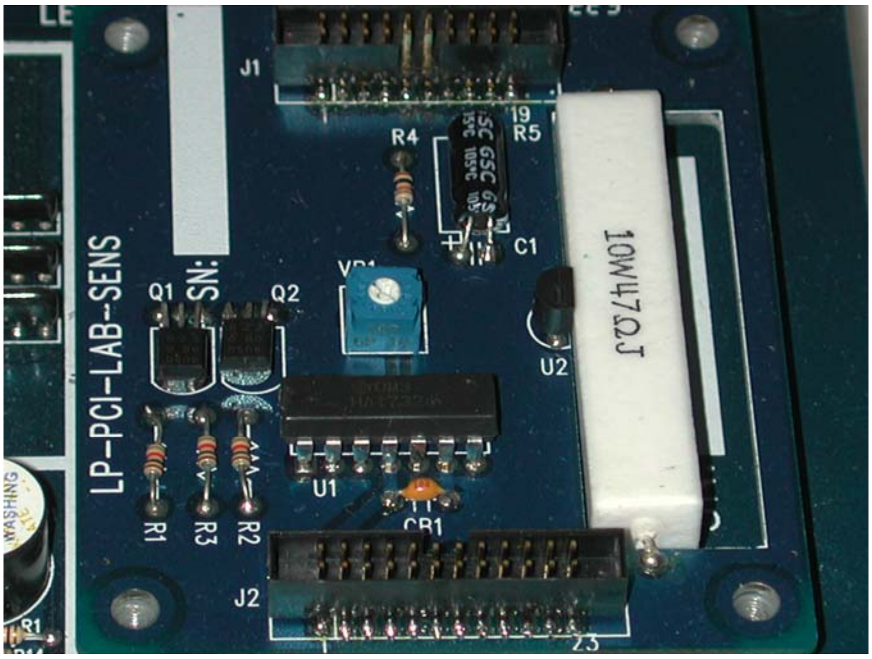


Figure 8-6-1 Resistance heaters

Part list:

Resistance:

47Ω10W cement resistance (heater)

three 1kΩ

NPN_BJT:

two 8050

Circuit diagram:

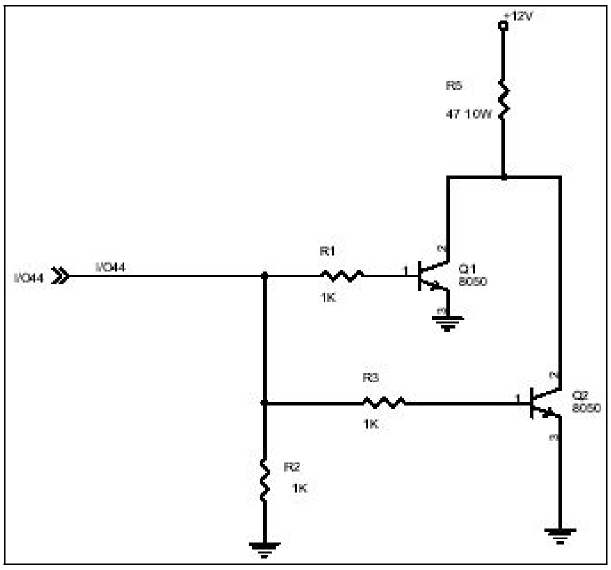


Figure 8-6-2 Circuit Diagram

Principal of experiment: Provide current to resistance heater so that it can produce heat energy.

Samples:

```
-O A415 10      Heater ON
-O A415 00      Heater OFF
```

MASM program code:

```
. MODEL SMALL
.386
. STACK

. DATA
OUT_DISABLE              EQU        0A418H
IO_PORT1                EQU        0A415H
OUT_DISABLED             EQU        0A408H
IO_PORT0D               EQU        0A400H
IO_PORT1D               EQU        0A404H

. CODE
BEGIN:
        PUSH      DS
        MOV     AX, 0
        PUSH    AX
        MOV     AX, @DATA
        MOV     DS, AX

DISP_BEGIN:
        MOV     DX, OUT_DISABLE
        MOV     AX, 0010H
        OUT     DX, AX
        MOV     DX, IO_PORT1                ; heater on
        MOV     AX, 10H                     ; 0001_0000
        OUT     DX, AX

        MOV     AH, 0BH                     ; Press any key to end the program
        INT     21H
        CMP     AL, 0FFH
        JNZ     DISP_BEGIN

        JMP     EXIT                        ; end of the program
EXIT:
        MOV     DX, OUT_DISABLED                        ; clear all settings
        MOV     AX, 0000H
        OUT     DX, AX
        MOV     DX, IO_PORT0D
        MOV     EAX, 00000000H
        OUT     DX, EAX
        MOV     DX, IO_PORT1D
        MOV     AX, 0000H
        OUT     DX, AX

        MOV     AH, 4CH
        INT     21H
        END     BEGIN
```

VC/C++ program code:

```
//heating//
```

```
        RegValue = 0x0010;
         arc=PlxIoPortWrite(
                hDevice,
                port + 0x15,
                BitSize32,
                &RegValue);

//no heating//
        RegValue = 0x0000;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x15,
                BitSize32,
                &RegValue);
```

8.7 Temperature sensor and DC motor upstream signal

The temperature sensor works with resistance heater in 8.6, and the values for the temperature sensed must be converted into digital signals through A/D converter, which is a little complicated. Figure 8-7-1 below shows its circuit diagram. DC motor fan circuit diagram is shown in Figure 8-7-3 below. It can be seen that one set of output line is FAN_OUTPUT signal, the reader can analyze the fan speed with this part of signal.

The temperature sensing part uses LM335 as temperature sensing element, while variable resistance mainly adjusts this element double bias voltage to improve its accuracy, and amplifies output to AD converter circuit using LM324 current, the unused OP AMP is pin-grounded as shown in Figure 8-7-2. When AD converter is separately used, this module must be removed to avoid the interferences of this circuit.

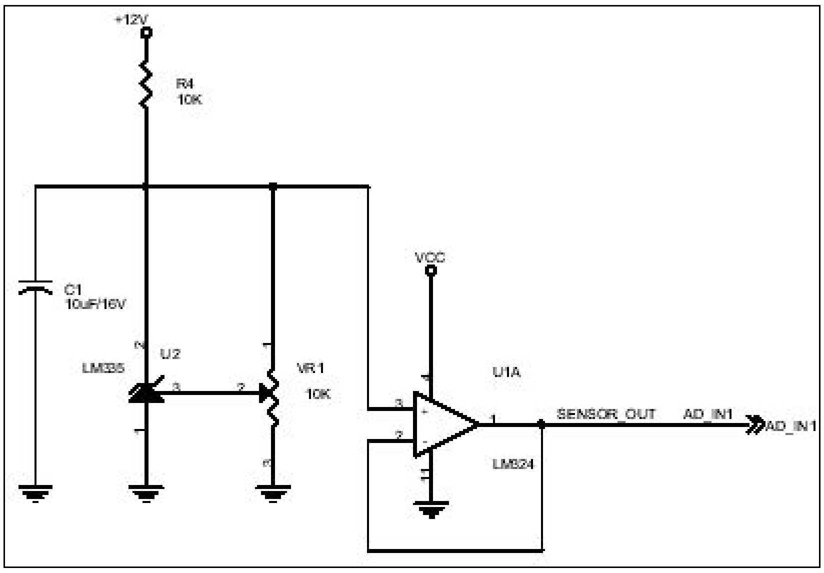


Figure 8-7-1 Temperature sensing circuit

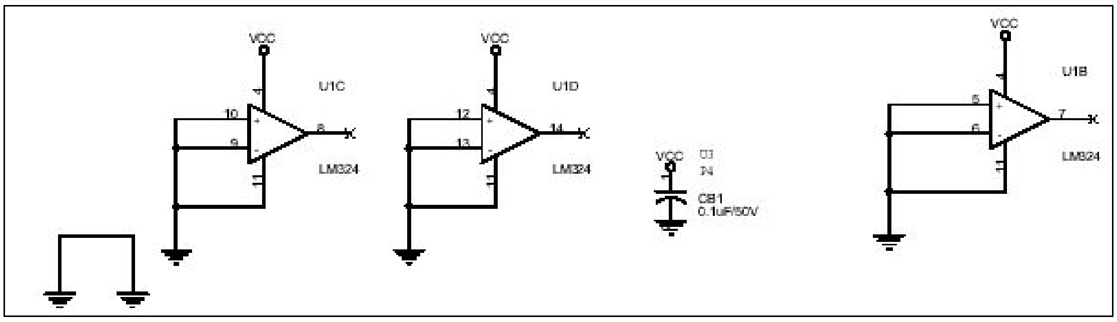


Figure 8-7-2   LM324 OP AMP

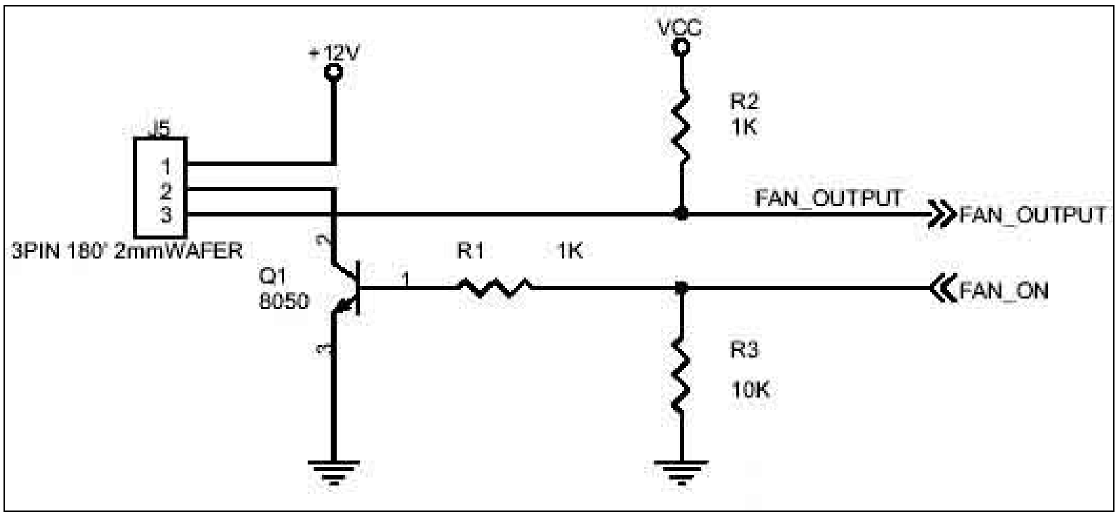


Figure 8-7-3 DC fan speed

Chapter 9 Dot matrix, keyboard and LCD

Dot matrix, keyboard scan and LCD are more difficult experiment modules, dot matrix is 16X16 red LED module, which is capable of carrying out different dot matrix experiments such as 4X4、8X8 and 16X16. The keyboard input is the commonly seen 4X4 scan input keyboard model, LCD module is a double row 16 bits character LCD module. The following is the experiment of the three modules.

9.1 Dot matrix output experiment (Debug Mode & MASM)

Experiment purpose: Make the rows and columns of dot matrix shine respectively.

Experiment module: As is shown in 9-1-1 below.

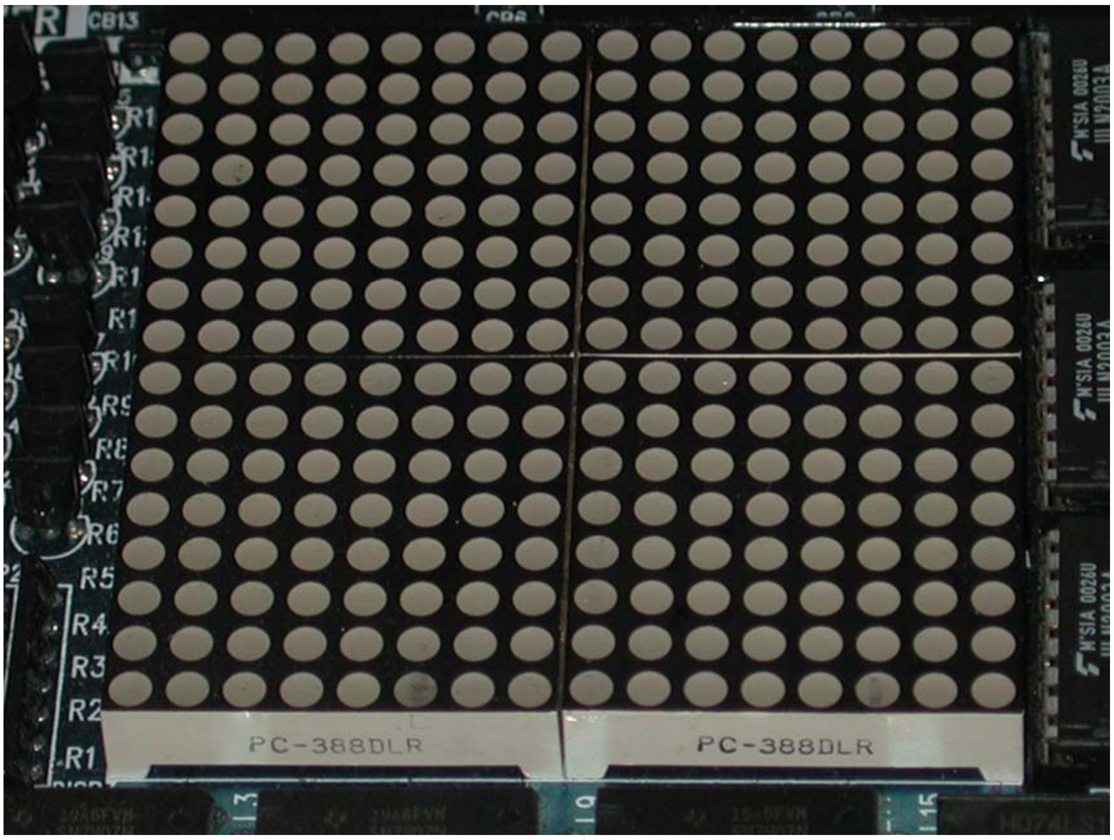


Figure 9-1-1   16X16 dot matrix module

Circuit Diagram: Figure 9-1-2 and 9-1-3 are row selection circuits, 16 columns can be selected from 4 bits. Figure 9-1-4 controls 16 rows with 2 sets of IO_PORT, forming this experiment circuit.

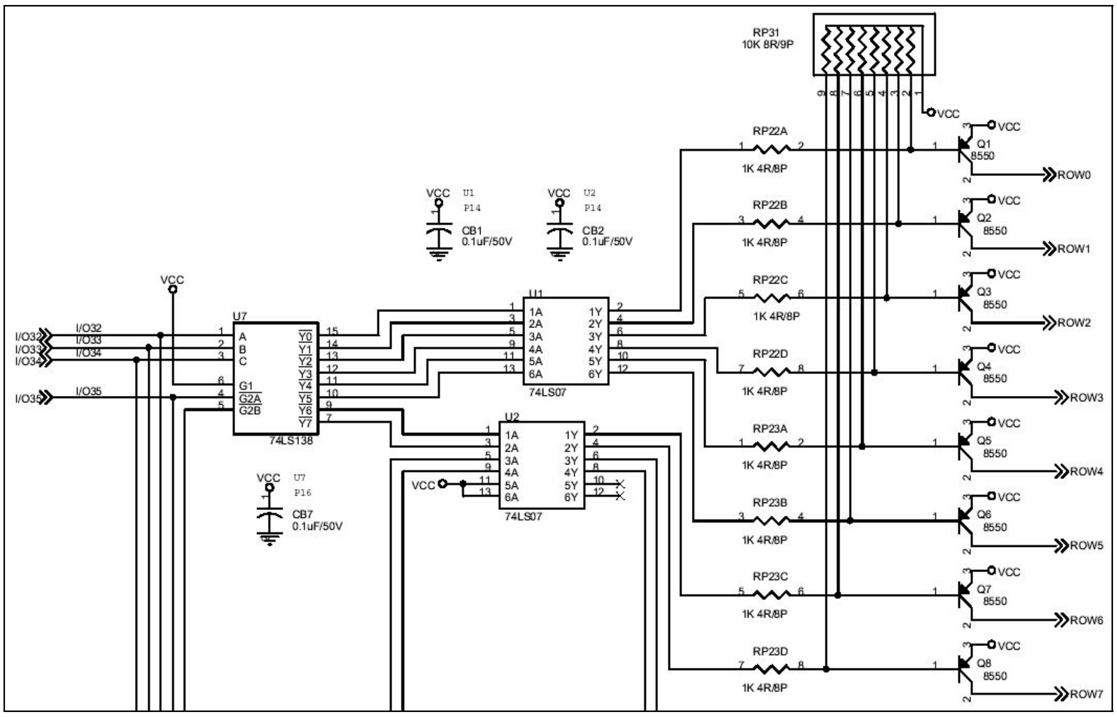


Figure 9-1-2 Column circuit diagram

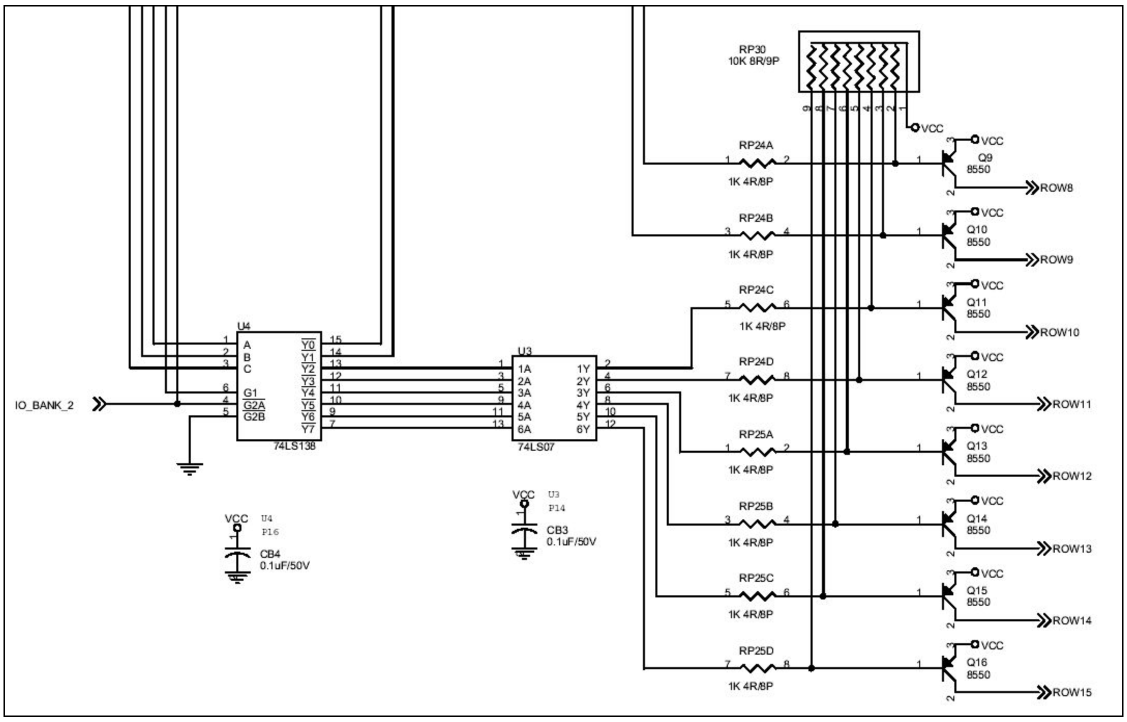


Figure 9-1-3 Column circuit diagram

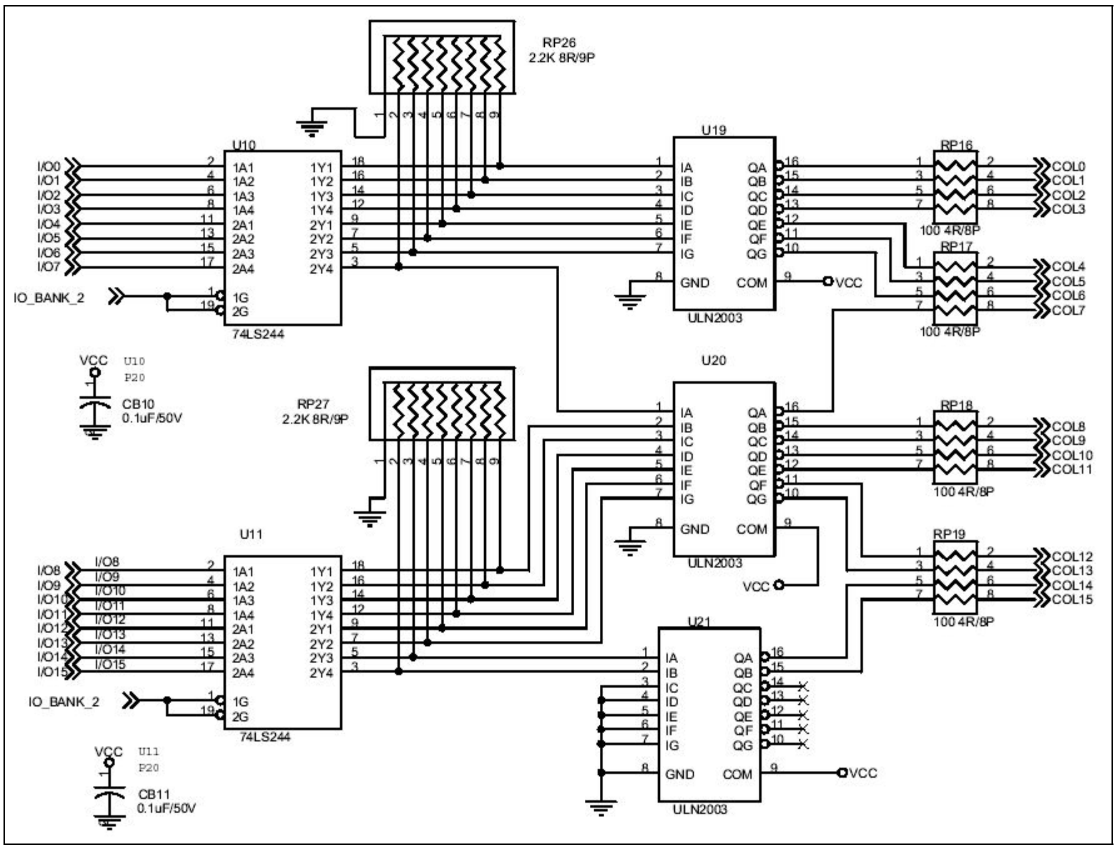


Figure 9-1-4 Row circuit diagram

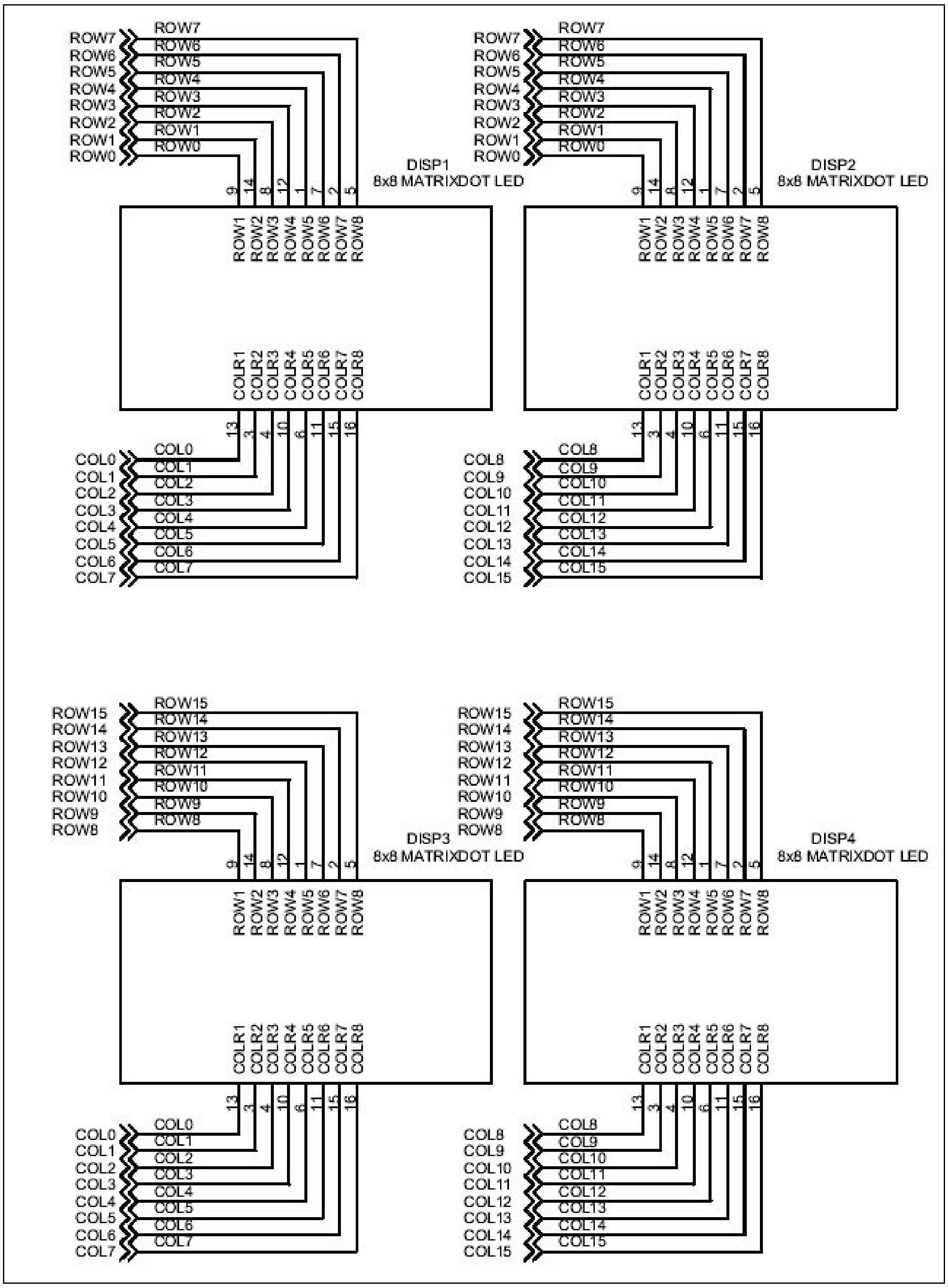


Figure 9-1-5 Dot matrix connection diagram

Principle of experiment: Column data port IO_00~IO15 Bank_2
Row data port IO_32~IO_35 Bank_2

Experiment procedure: First open IO port to output mode.

Define the row and column of output.
Output respectively and close IO port upon completion.

MASM Program code:

```
. MODEL SMALL
.386
. STACK

. DATA
OUT_DISABLE          EQU     0A418H
IO_PORT0             EQU     0A410H
IO_PORT1             EQU     0A414H
IO_PORT2             EQU     0A412H
IO_PORT3             EQU     0A413H
OUT_DISABLEA         EQU     0A428H
IO_PORT0A            EQU     0A420H
IO_PORT1A            EQU     0A424H
IO_PORT2A            EQU     0A423H
OUT_DISABLED         EQU     0A408H
IO_PORT0D            EQU     0A400H
IO_PORT1D            EQU     0A404H
IO_PORT2D            EQU     0A402H
TEST_UNIT            DB      10H
MAT_ROW              DB      01H
MAT_COL              DB      00H
MAT_COUNT            DB      00H
TEMP                 DW      0H
TEMP_LOOP            DW      0H
LCD_TEMP             DW      0H
TEMP_LCD1            DW      0H
TEMP_LCD2            DD      0H
; ***********************************************************
. CODE
BEGIN:
        PUSH    DS
        MOV     AX, 0
        PUSH    AX
```

```
            MOV         AX, @DATA
            MOV         DS, AX
MARTIX:                                                 ; Matrix test program
            MOV         DX, OUT_DISABLEA
            MOV           AX,0010H                          ;Set  I/O16~I/O19&  I/O0~I/O7  to  be  high
impedance.
            OUT         DX, AX

            CALL        MAT_COU                         ; MATRIX scan all on
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU
            CALL        MAT_COU

            MOV         TEST_UNIT, 10H
            MOV         EAX, 00000001H
MARTIX_A:
            MOV         DX, IO_PORT0A                       ; Control COL display
            OUT         DX, EAX
            CALL        WAIT_1S
            PUSH        EAX

ROW_LOOP:
            MOV         AL, MAT_ROW

            MOV         DX, IO_PORT1A
            OUT         DX, AX
            CALL        WAIT_1MS

            INC MAT_ROW
            CMP         MAT_ROW, 10H
            JE          MATRIX_B

            JMP         ROW_LOOP
```

```
MATRIX_B:
            POP         EAX
            SHL         EAX, 1
            DEC         TEST_UNIT
            CMP         TEST_UNIT, 0000H
            JBE         MAT_EXIT
            JMP         MATRIX_A

MAT_EXIT:
            MOV         AL, MAT_COL
            CALL        MAT_ROWA

            CALL        MAT_COLA

            INC MAT_COL
            CMP         MAT_COL, 10H
            JE          MATRIX_END

            JMP         MAT_EXIT

MATRIX_END:                                         ;End of matrix display
            MOV         DX, IO_PORT0A
            MOV         EAX, 00000000H
            OUT         DX, EAX
            MOV         DX, IO_PORT1A
            MOV         AX, 0000H
            OUT         DX, AX
MAT_COLA:                                             ;MATRIX COMMAND
            MOV         DX, IO_PORT0A
            MOV         EAX, 0000FFFFH
            OUT         DX, EAX
            CALL        WAIT_1S
            RET

MAT_ROWA:
            MOV         DX, IO_PORT1A
            OUT         DX, AX
            CALL        WAIT_1S
            RET

MAT_COU:
            MOV         DX, IO_PORT0A
            MOV         EAX, 0000FFFFH
            OUT         DX, EAX
MAT_A:
            MOV         DX, IO_PORT1A
            MOV         AL, MAT_COUNT
            OUT         DX, AX
            CALL        WAIT_1MS
            INC         MAT_COUNT
            CMP         MAT_COUNT, 10H
            JE          MAT_AEXIT
            JMP         MAT_A

MAT_AEXIT:
            RET
            JMP         EXIT
WAIT_3S:
            MOV         TEMP_LOOP, 07H
WAIT_3S_LOOP:
            CALL        WAIT_2S
            DEC         TEMP_LOOP
            CMP         TEMP_LOOP, 00H
            JBE         WAIT_3S_EXIT
            JMP         WAIT_3S_LOOP
WAIT_3S_EXIT:
            RET

WAIT_2S:
            CALL        WAIT_1S
            CALL        WAIT_1S
            CALL        WAIT_1S
            CALL        WAIT_1S
            CALL        WAIT_1S
            CALL        WAIT_1S
            RET
```

```
WAIT_1S:
            MOV       BX,0007FH
WAIT_LOOP:
            CALL      WAIT_1MS
            DEC       BX
            CMP       BX,0000H
            JBE       WAIT_1S_EXIT
            LOOP      WAIT_LOOP
WAIT_1S_EXIT:
            RET

WAIT_1MS:
            MOV       CX, 03FFFH
WAIT_LOOP1:
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            LOOP      WAIT_LOOP1
            RET

WAIT_2MS:
            MOV       CX, 07FFH
WAIT_LOOP2:
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            MOV       BX, BX
            LOOP      WAIT_LOOP2
            RET
EXIT:
            MOV       DX, OUT_DISABLED                ; clear all settings
            MOV         AX, 0000H
            OUT         DX, AX
            MOV         DX, IO_PORT0D
            MOV         EAX, 00000000H
            OUT         DX, EAX
            MOV         DX, IO_PORT1D
            MOV         AX, 0000H
            OUT         DX, AX

            MOV         AH, 4CH
            INT         21H
            END         BEGIN
```

9.2 Dot matrix output experiment (VC/C++)

Experiment purpose: Drive 16X16 dot matrix with VC/C++

VC/C++ Program code:

```
//Set all IO Port to output
      RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x28,
            BitSize32,
            &RegValue);
//Select the first row
     RegValue = 0x0000;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x24,
            BitSize32,
            &RegValue);
//The eighth one on the left on
    RegValue = 0x00FF;    //0000_0000_1111_1111
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x20,
            BitSize32,
            &RegValue);
 //the eighth one on the right on
    RegValue = 0xFF00;    //1111_1111_0000_0000
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x20,
            BitSize32,
            &RegValue);
//the eighth one on the left on
    RegValue = 0x00FF;    //1111_1111
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x21, //Change output port
            BitSize32,
            &RegValue);
//All on
    RegValue = 0xFFFF;    //1111_1111_1111_1111
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x20,
            BitSize32,
            &RegValue);
//Select the second row
 RegValue = 0x0001;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x24,
            BitSize32,
            &RegValue);
//Select the third row
    RegValue = 0x0002;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x24,
            BitSize32,
            &RegValue);
//Select the fourth row
```

```
        RegValue = 0x0003;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x24,
                BitSize32,
                &RegValue);
//Select the fifth row
        RegValue = 0x0004;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x24,
                BitSize32,
                &RegValue);
//Select the sixth row
        RegValue = 0x0005;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x24,
                BitSize32,
                &RegValue);
// Select the seventh row
        RegValue = 0x0006;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x24,
                BitSize32,
                &RegValue);
// Select the 8th row
        RegValue = 0x0007;
         rc=PlxIoPortWrite(
                hDevice,
                port + 0x24,
                BitSize32,
                &RegValue);
// Select the 9th row
        RegValue = 0x0008;
         rc=PlxIoPortWrite(
                hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
//Select the 10th row
        RegValue = 0x0009;
         rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
//Select the 11th row
        RegValue = 0x000a;
         rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
//Select the 12th row
        RegValue = 0x000b;
         rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
//Select the 13th row
        RegValue = 0x000c;
         rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
//Select the 14th row
        RegValue = 0x000d;
         rc=PlxIoPortWrite(
                    hDevice,
                    port + 0x24,
                    BitSize32,
                    &RegValue);
```

```
//Select the 15th row
        RegValue = 0x000e;
         rc=PlxIoPortWrite(
                  hDevice,
                  port + 0x24,
                  BitSize32,
                  &RegValue);
//Select the 16th row
        RegValue = 0x000f;
         rc=PlxIoPortWrite(
                  hDevice,
                  port + 0x24,
                  BitSize32,
                  &RegValue);
```

9.3 4 X 4 Keyboard input experiment (Debug Mode & MASM)

Experiment purpose: Pushbutton switches input experiment, practicing scan input.

Experiment module: As is shown in Figure 9-3-1 below.

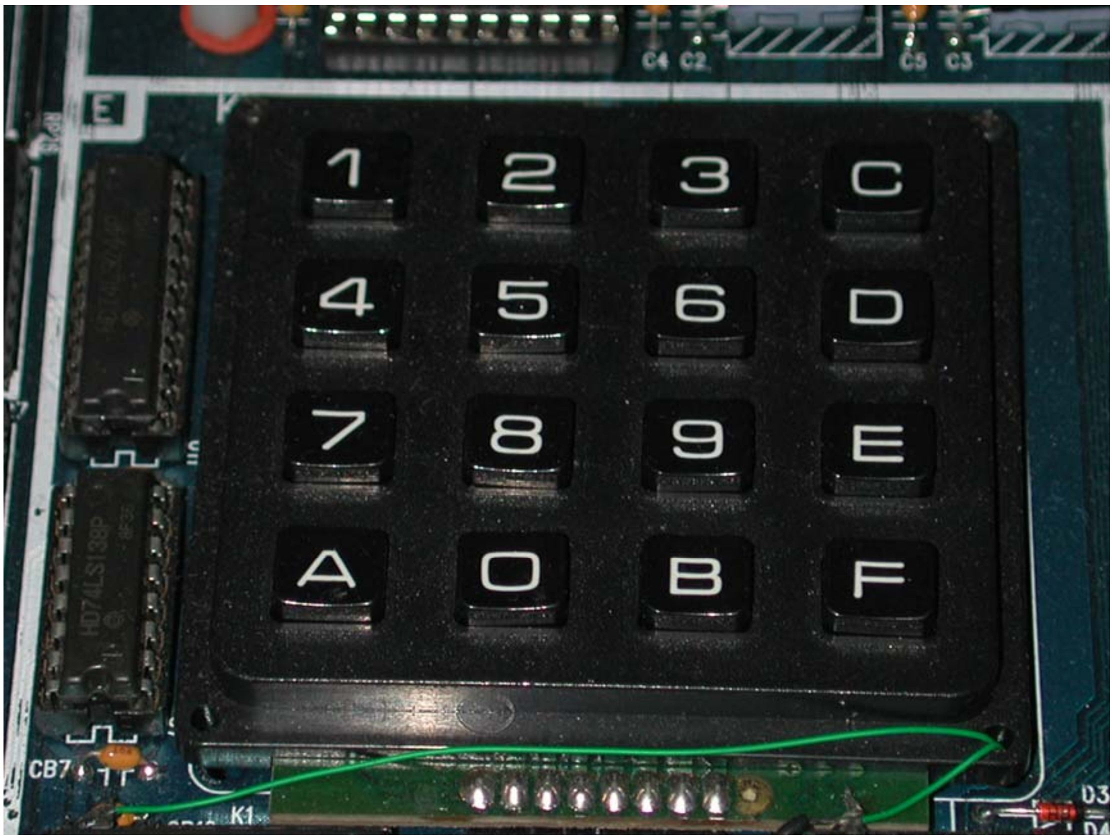


Figure 9-3-1 Keyboard experiment module

Part list:

- Logic IC 74LS244 X 1
- Resistor network 10KΩ X 1
- Diode 1N4148 X 4
- Capacitance 0.1u X 1

Circuit diagram: As is shown in the Figures 9-3-2 and 9-3-3 below.

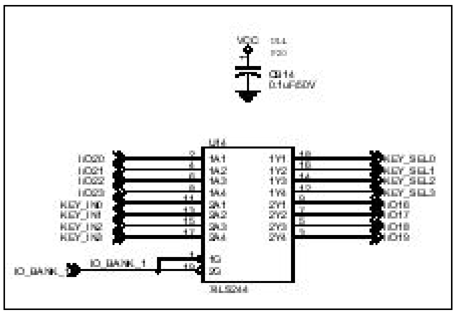


Figure 9-3-2 74LS244 latch circuit diagram

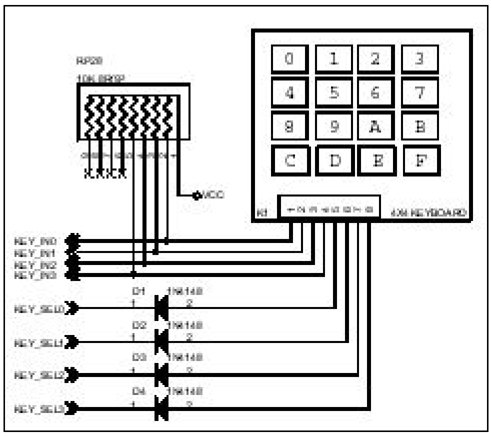


Figure 9-3-3 Keyboard circuit diagram

Principle of experiment:

In common circuit designs, many pushbuttons are often used, especially when it is required to input plenty of data, keyboards are the best choice. When we want to form a keyboard circuit, as is shown in the above Figures 9-3-2 and 9-3-3, sixteen I/O are required to connect each switch to output ports directly. Arrange all pushbuttons in the form of matrix, therefore 16 pushbutton can form 4X4 matrix circuit.

To facilitate identification, fix the coordinate of keyboard positions with rows and columns, the traverse keys are marked with the first row, the second row, etc, while the longitudinal keys are marked with the first column, the second column, etc.

Scan signal output IO_16~IO19 Bank_1

Read signal input IO_20~IO23 Bank_1

Experiment procedures: Open IO port with -o A418 c0 to enable it to have an input function.

Output scan signal to IO_16~IO_19

Receive signals input by IO_20~IO_23

end the program to close IO port by means of -o A418 00.

MASM program code:

```
.MODEL    SMALL
.386
.STACK

.DATA
OUT_DISABLE                     EQU        0A418H
IO_PORT0                      EQU        0A410H
IO_PORT1                      EQU        0A414H
IO_PORT2                      EQU        0A412H
IO_PORT3                      EQU        0A413H
OUT_DISABLEA                    EQU        0A428H
IO_PORT0A                     EQU        0A420H
IO_PORT1A                     EQU        0A424H
IO_PORT2A                     EQU        0A423H
OUT_DISABLED                    EQU        0A408H
IO_PORT0D                     EQU        0A400H
IO_PORT1D                     EQU        0A404H
IO_PORT2D                     EQU        0A402H
TEST_UNIT                     DB         10H
MAT_ROW                         DB         01H
MAT_COL                        DB         00H
MAT_COUNT                       DB         00H
TEMP                          DW         0H
TEMP_LOOP                      DW         0H
LCD_TEMP                       DW         0H
TEMP_LCD1                      DW         0H
TEMP_LCD2                      DD         0H
;************************************************************
.CODE
BEGIN:
            PUSH      DS
            MOV        AX,0
            PUSH      AX
            MOV        AX,@DATA
            MOV        DS,AX
KEY_SCAN:
            MOV        DX,OUT_DISABLE
```

```
        MOV     AX,0010H                ;set  I/O16~I/O19&  I/O0~I/O7  to  high
impedance.
        OUT     DX,AX
        MOV     DX,IO_PORT1
        MOV     AX,0000H
        OUT     DX,AX
KEY_BEGIN:

        CALL    KEY_START1              ; Keyboard scan    0~3

        CALL    KEY_START2              ; Keyboard scan    4~7

        CALL    KEY_START3              ; Keyboard scan    8~B

        CALL    KEY_START4              ;Keyboard scan C~F

        MOV     AH,0BH                  ;press any key to stop the program.
        INT     21H
        CMP     AL,0FFH
        JNZ     KEY_BEGIN

        JMP     EXIT            ;end of main program
KEY_DB0:
        MOV     AX,00003F00H            ;Display numbers
        CALL    KEY_DISPLAY

KEY_DB1:
        MOV     AX,00000600H
        CALL    KEY_DISPLAY

KEY_DB2:
        MOV     AX,00005B00H
        CALL    KEY_DISPLAY

KEY_DB3:
        MOV     AX,00004F00H
        CALL    KEY_DISPLAY
```

```
KEY_DB4:
        MOV      AX,00006600H
        CALL     KEY_DISPLAY

KEY_DB5:
        MOV      AX,00006D00H
        CALL     KEY_DISPLAY

KEY_DB6:
        MOV      AX, 00007D00H
        CALL     KEY_DISPLAY

KEY_DB7:
        MOV      AX, 00000700H
        CALL     KEY_DISPLAY

KEY_DB8:
        MOV      AX, 00007F00H
        CALL     KEY_DISPLAY

KEY_DB9:
        MOV      AX, 00006F00H
        CALL     KEY_DISPLAY

KEY_DBA:
        MOV      AX, 00007700H
        CALL     KEY_DISPLAY

KEY_DBB:
        MOV      AX, 00007C00H
        CALL     KEY_DISPLAY

KEY_DBC:
        MOV      AX, 00005800H
        CALL     KEY_DISPLAY

KEY_DBD:
        MOV      AX, 00005E00H
```

```
          CALL      KEY_DISPLAY

KEY_DBE:
          MOV        AX, 00007900H
          CALL      KEY_DISPLAY

KEY_DBF:
          MOV        AX, 00007100H
          CALL      KEY_DISPLAY
          RET
; ******************************
KEY_DISPLAY: KEY COMMAND
          MOV        DX, IO_PORT0
          OUT       DX, AX
          CALL      WAIT_1S
          JMP       KEY_BEGIN
          RET

KEY_START1:
          MOV        DX, IO_PORT0
          MOV        EAX, 00E00000H
          OUT       DX, EAX

          MOV        DX, IO_PORT2
          IN        AX, DX

          CMP       AX,0EEH                  press "1 ",  "1 "is displayed seven-segment code
          JZ        KEY_DB1
          CALL      WAIT_2MS

          CMP        AX,0EDH                        press "2 ",  "2 "is displayed seven-segment
code
          JZ        KEY_DB2
          CALL      WAIT_2MS

          CMP       AX,0EBH                    press "3 ",  "3 "is displayed seven-segment code
          JZ        KEY_DB3
          CALL      WAIT_2MS
```

```
        CMP     AX,0E7H             ; press “C “,   “C “is displayed seven-segment
code
        JZ      KEY_DBC
        CALL    WAIT_2MS
        RET

KEY_START2:
        MOV     DX, IO_PORT0
        MOV     EAX, 00D00000H
        OUT     DX, EAX

        MOV     DX, IO_PORT2
        IN      AX, DX

        CMP     AX,0DEH               press “4 “,   “4 “is displayed seven-segment
code
        JZ      KEY_DB4
        CALL    WAIT_2MS

        CMP     AX,0DDH             ; press “5 “,   “5 “is displayed seven-segment
code
        JZ      KEY_DB5
        CALL    WAIT_2MS

        CMP     AX,0DBH             ; press “6 “,   “6 “is displayed seven-segment
code
        JZ      KEY_DB6
        CALL    WAIT_2MS

        CMP     AX,0D7H             ; press “D “,   “D “is displayed seven-segment
code
        JZ      KEY_DBD
        CALL    WAIT_2MS
        RET

KEY_START3:
        MOV     DX, IO_PORT0
        MOV     EAX, 00B00000H
        OUT     DX, EAX

        MOV     DX, IO_PORT2
        IN      AX, DX

        CMP     AX,0BEH                ; press “7 “,  “7 “is displayed seven-segment
code
        JZ      KEY_DB7
        CALL    WAIT_2MS

        CMP     AX,0BDH                ; press “8 “,  “8 “is displayed seven-segment
code
        JZ      KEY_DB8
        CALL    WAIT_2MS

        CMP     AX,0BBH                ; press “9 “,  “9 “is displayed seven-segment
code
        JZ      KEY_DB9
        CALL    WAIT_2MS

        CMP     AX,0B7H                  press “E “,  “E “is displayed seven-segment
code
        JZ      KEY_DBE
        CALL    WAIT_2MS
        RET

KEY_START4:
        MOV     DX,IO_PORT0
        MOV     EAX, 00700000H
        OUT     DX, EAX

        MOV     DX, IO_PORT2
        IN      AX, DX

          CMP     AX,07EH              ; press “A “,  “A “is displayed seven-segment
  code
        JZ      KEY_DBA
```

```
          CALL      WAIT_2MS

          CMP       AX,07DH                   ;press "0 ",   "0 "is displayed seven-segment
code
          JZ        KEY_DB0
          CALL      WAIT_2MS

          CMP       AX,07BH                   ;press "B ",   "B "is displayed seven-segment
code
          JZ        KEY_DBB
          CALL      WAIT_2MS

          CMP       AX,077H                   ;press "F ",   "F " is displayed on seven-segment
code
          JZ        KEY_DBF
          CALL      WAIT_2MS
          RET
WAIT_3S:
          MOV       TEMP_LOOP, 07H
WAIT_3S_LOOP:
          CALL      WAIT_2S
          DEC       TEMP_LOOP
          CMP       TEMP_LOOP, 00H
          JBE       WAIT_3S_EXIT
          JMP       WAIT_3S_LOOP
WAIT_3S_EXIT:
          RET

WAIT_2S:
          CALL      WAIT_1S
          CALL      WAIT_1S
          CALL      WAIT_1S
          CALL      WAIT_1S
          CALL      WAIT_1S
          CALL      WAIT_1S
          RET
WAIT_1S:
          MOV       BX, 0007FH
```

```
WAIT_LOOP:
          CALL      WAIT_1MS
          DEC       BX
          CMP       BX, 0000H
          JBE       WAIT_1S_EXIT
          LOOP      WAIT_LOOP
WAIT_1S_EXIT:
          RET

WAIT_1MS:
          MOV       CX, 03FFFH
WAIT_LOOP1:
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          LOOP      WAIT_LOOP1
          RET

WAIT_2MS:
          MOV       CX, 07FFH
WAIT_LOOP2:
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          MOV       BX, BX
          LOOP      WAIT_LOOP2
          RET
EXIT:
          MOV       DX, OUT_DISABLED; clear all settings.
          MOV       AX, 0000H
          OUT       DX, AX
```

```
MOV      DX, IO_PORT0D
MOV      EAX, 00000000H
OUT      DX, EAX
MOV      DX, IO_PORT1D
MOV      AX, 0000H
OUT      DX, AX

MOV      AH, 4CH
INT      21H
END      BEGIN
```

9.4 4 X 4 keyboard input experiment (VC/C++)

Experiment purpose: To enable keyboard input by writing programs with VC/C++ and outputs to the seven-segment LED.

VC/C++ program code:

```
//set IO Port
    RegValue = 0x0010;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x18,
        BitSize32,
        &RegValue);
//Scan the first row.
    RegValue = 0x0010;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
// Fetch the upstream value
    rc=PlxIoPortRead(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
//Scan the second row
    RegValue = 0x0020;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
// Fetch the upstream value
    rc=PlxIoPortRead(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
//Scan the third row
    RegValue = 0x0040;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
//Fetch the upstream value
    rc=PlxIoPortRead(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
//Scan the fourth row.
    RegValue = 0x0080;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
//Fetch the upstream value
    rc=PlxIoPortRead(
        hDevice,
        port + 0x12,
        BitSize32,
        &RegValue);
```

9.5 LCD

Exercises module: Shown in Figure 9-5-1 is the LCD display board used in this exercise.

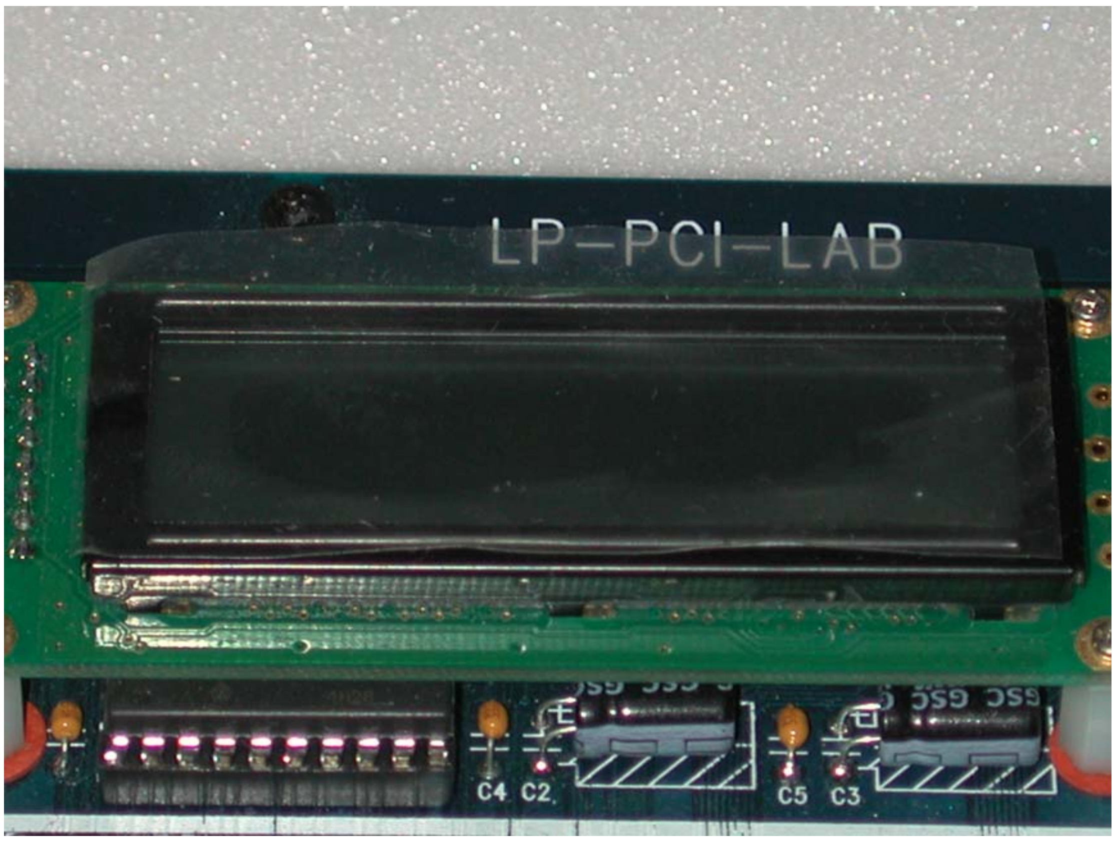


Figure 9-5-1 LCD practice module

Circuit diagram: The user himself should refer to PCI_LAB.pdf file in circuit diagram folder in the disk.

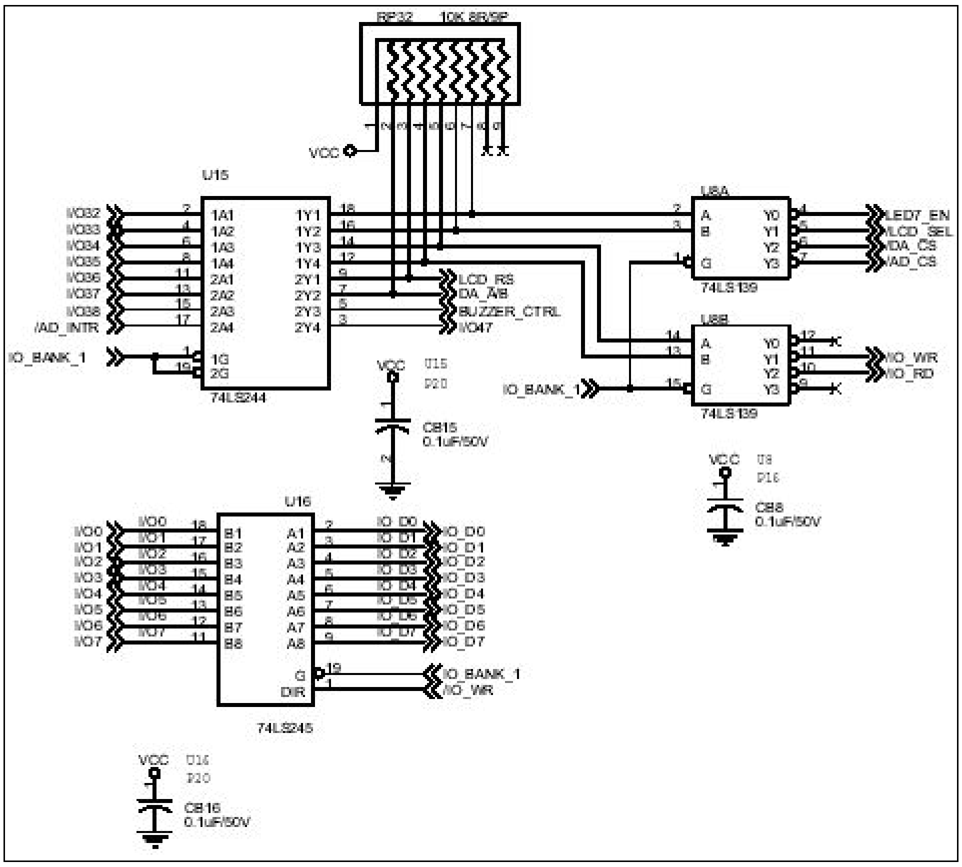


Figure 9-5-2 module switching circuit

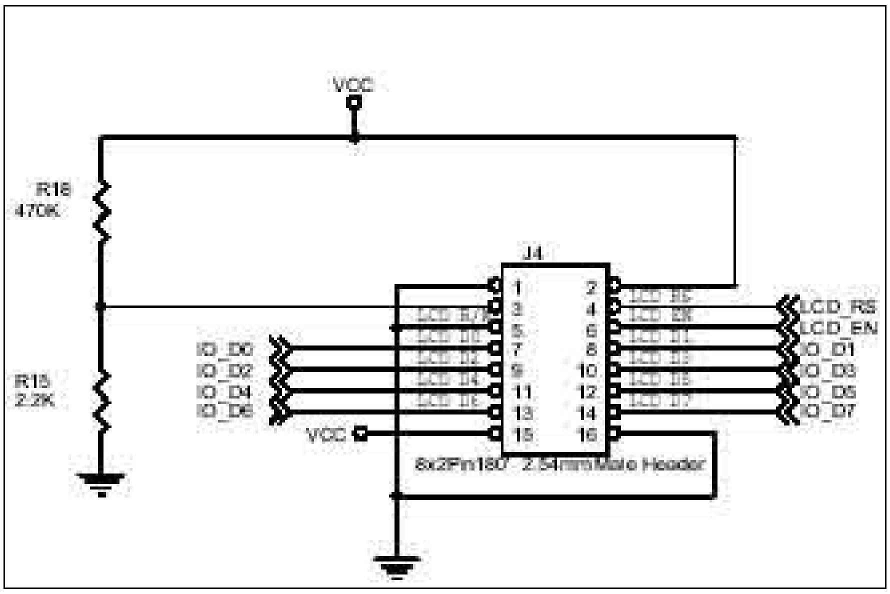


Figure 9-5-3 LCD enable and read/write end

There are two types of LCDs: fonts and drawing. Different character fonts are burned inside the fonts LCD (arithmetic symbols, Arabic numerals, capital lowercase English letters, Japanese), the user writes the control codes (ASCII CODE) of the characters to be displayed into LCD, the fonts will be displayed in the display screen. LCD on common fax machine is font LCD. All the dots on drawing LCD use ON or OFF control to display data or graphs, LCD used on common notebooks or laptops are drawing LCD. The control of the former is relatively simple, so font LCD (CM 1602222) is selected, now we will proceed to describe this type of LCD, as follows:

LCD outside drawing and Pins

This LCD is double column 16 character LCD with 14 Pins, which is divided into data signal lines (DB0 ~ DB7) and control signal lines (RS, R/W, E), while the other 3 three Pins are power control lines (Vss, Vdd, Vo). The functions of the Pins are described as follows:

DB3 ~ DB0 (Low-order Data Bus)

These four lines are low 4 bits data lines, used for transmitting data. When the LCD is connected to a 4 bits CPU, these four signal lines are disconnected.

DB7 ~ DB4 (High-order Data Bus)

These 4 signal lines are high 4 bits data lines, used for transmitting data. When LCD is connected to 4 bits CPU, these four signal lines must be connected to a data line (D3 ~ D0) of the controller

RS (Register Select)

RS is a signal line that selects instruction register or data register. When RS=1, DR is selected. Conversely, when RS=0, IR is selected. After selecting register, write into or read from, so when R/$\overline{W}$=1, data is read from LCD. R/$\overline{W}$ signal line is usually used in conjunction with the RS signal line.

E (Enable)

This Enable signal line is used to enable LCD, whether data can be written into/read from LCD depends on Pin E. That Pin E is high or low will not enable data to be written into or read from LCD, which can only be enabled when level signals from low→high→low are generated.

V_{DD} (Power Supply)

V_{DD} is the power cord connected to +5V.

V_{SS} (Power Supply)

V_{SS} is the power cord that is grounded

V_o (Power Supply)

V_o is the power cord that adjusts voltage, controlling the brightness of the LCD.

Internal structure diagram

There are more then 10 block diagrams inside LCD, as follows:

1. Registers
2. Busy Flag
3. Address Counter
4. Display Data RAM
5. Code Generator RAM
6. Code Generator ROM
7. Timing Generator
8. Cursor/Blink Converter
9. Parallel to Serial Converter
10. Bias Voltage Generator
11. LCD Driver
12. LCD Panel

Register (R)

There are 2 registers inside LCD module: instruction register (IR for short) and data register (DR for short). Instruction register is used to store instructions that control LED, such as Display Clear and Cursor Shift, etc. And a data register is used to store display data to be written into/read from DD RAM or CG RAM. After writing external data into the data register, the LCD will automatically write the data of the DR into DD RAM or CG RAM. To choose IR or DR depends on RS signal level, when RS=0, select DR, and when RS =1, IR is selected.

Busy flag (BF)

This flag is used to indicate whether LCD is working during internal operation, namely whether it is ready to receive external data. When BF=1, it means LCD is during internal operation; now LCD will not accept any instructions from the outside until BF=0. After executing current instructions, LCD will automatically clear busy flag to be 0, that is, BF=0.

Address counter (AC)

Address counter is used to generate the addresses required by display data memory and character generator memory. When setting the address instructions of DD RAM or CG RAM to be written into instructions register, select the addresses of DD RAM or CG RAM to be stored in AC so that write into/read out data can be stored in/read from DD RAMO or CG RAM that AC points to. When data is written into/read from DD RAM or CG RAM, AC may automatically add or subtract one. The contents of AC corresponds to data line DB0 ~ DB6.

Display data memory (DD RAM)

Display data memory is used to store the display data of LCD, with capacity of 80*8 bit, it can store 80 8-bit character codes. The address of DD RAM is the contents of AC, usually expressed in Hexadecimal.

Character generator only memory (CG ROM)

CG ROM inside LCD can generate 160 different kinds of 5*7 dot matrix fonts. Shown in table 9-5-1 are the dot matrix fonts, and it can be seen that the control codes for these fonts are nearly the same as ASCII CODE. To display the fonts in CG ROM, it is only necessary to write the control codes of the fonts into the display memory (DD RAM).

Table 9-5-1 font codes

Lower 4 Bit \ Upper 4 Bit	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	β	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	█

Character generator memory (CG RAM)

CG RAM IN inside LCD is used to store the user-defined 5*7 dot matrix fonts; at most 8 fonts can be stored. To display the fonts in CG RAM, the control codes displayed in the row in the extreme left of table 9-5-1 must be written into display memory (DD RAM).

Timing generator (TG)

Timing generator generates the timing signals required by DD RAM, CG RAM and CG ROM. With proper timing signal control, no interference may occur when externally accessing DD RAM data and reading data display.

Parallel to Serial Converter

This converting circuit can convert parallel data read from CG ROM or CG RAM into serial data for the use of a display driver.

Cursor/Blink Controller

This circuit controls the character blink on DD RAM address and whether the cursor appears

Bias Voltage Generator

Bias voltage circuit is used to provide the voltage necessary to drive the LCD.

LCD Driver

This circuit produces signals necessary to drive 5*7 dot matrix after receiving the display data, time signal and bias voltage.

LCD Panel

LCD front panel is a dot matrix display screen, which can be divided into 6 types of specifications such as a single/double column 16 characters, single/double column 20 characters, single/double column 40 characters, etc.

LCD instruction set

Table 9-5-2 is the instruction set that controls LCD, including 11 types. We can learn from the table of the functions of the instructions and the time needed to execute them. Some of the instructions are operating modes that set the LED. Some are addressing internal DD RAM or CG RAM, while the remainders are used to write into/read from data from DD RAM or CG RAM. For the external circuit to input control LED, whether LCD is in a busy status should be check first, if BF=1, it means that LCD is in a busy status, until BF=0.

Display Clear

This command can clear all the contents of DD RAM to be 20H and the contents of address counter to be 0.

Display/Cursor Home

This command can clear the contents of AC to be 0 without affecting the contents of DD RAM.

Enter Mode Set

I/D: When this bit is used to display data write into/read from DD RAM or CG RAM, the contents of AC is +1 or –1.

S: When this bit can only control data write into DD RAM rather than CG RAM, the entire display is whether shift is necessary.

Display ON/OFF

D: This bit can control the ON/OFF function of the display.

C: This bit can control whether the cursor will display on LCD display screen.

B: This bit can control whether the display character blinks.

Display/Cursor Shift

This command can control individual shift of the cursor or the simultaneous shift of the cursor and display.

Function Set

Is used to set data length and display format. DL bit in command filed is used to set data length, when DL=1, the data length is 8 bit; when DL=0, the data length is 4 bits. Bit N in the field is used to select single column or double column display, when N=1, it is double column display; when N=0, it is a single column display.

CG RAM Address Set

This command can set CG RAM Address.

DD RAM Address Set

This instruction is used to set DD RAM Address.

Busy Flag/Address Counter Read

This read instruction can be used to judge whether the LCD is in a busy status or not and to read the contents of the address counter.

This instruction is used to write display data into the CG RAM or DD RAM.

This instruction is used to read data in CG RAM or DD RAM.

Table 9-5-2 LCD instruction set

Instruction	Instruction Code										DESCRIPTION	Executed Time(fosc =270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address to "00H " from AC	1.53mS
Cursor At Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original Position if shifted. The contents of DDRAM are not changed.	1.53mS
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39μS
Display On/Off Control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor(C), and Blinking of cursor(B) ON/OFF control bit.	39μS
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shifts cursor bit, and the direction, without changing of DDRAM data.	39μS
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL:8-BIT/4-BIT), number of display lines(N:2-line/1-line) and, display font type (F:5x11dots/5x8 dots).	39μS
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39μS
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39μS
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0μS
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM / CGRAM)	43μS
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Reads data from internal RAM (DDRAM / CGRAM).	43μS

9.6 LCD(MASM)

Experiment purpose: Write programs with MASM to enable LCD to display English characters like A~\.
MASM program code:

```
. MODEL SMALL
.386
. STACK

. DATA
OUT_DISABLE              EQU      0A418H
IO_PORT0               EQU      0A410H
IO_PORT1               EQU      0A414H
IO_PORT2               EQU      0A412H
```

```
IO_PORT3                EQU      0A413H
OUT_DISABLED              EQU      0A408H
IO_PORT0D                EQU      0A400H
IO_PORT1D                EQU      0A404H
IO_PORT2D                EQU      0A402H
TEST_UNIT                DB       10H
TEMP                     DW       0H
TEMP_LOOP                 DW       0H
LCD_TEMP                  DW       0H
TEMP_LCD1                 DW       0H
TEMP_LCD2                 DD       0H
; ************************************************************
. CODE
BEGIN:
        PUSH     DS
        MOV      AX, 0
        PUSH     AX
        MOV      AX, @DATA
        MOV      DS, AX

        MOV      DX, OUT_DISABLE
        MOV        AX,0010H                    ;  set   I/O16~I/O19&  I/O0~I/O7  to  high
impedance
        OUT      DX, AX

LCD_DISPLAY:                                   ;LCD_DISPLAY testing program
        MOV      EBX, 00000038H                 ; clear LCD screen
        CALL     COMMAND

        MOV      EBX, 00000001H
        CALL     COMMAND

        MOV      EBX, 0000000EH
        CALL     COMMAND
        MOV      DX, OUT_DISABLE
        MOV      AX,0010H                   ;set I/O16~I/O19 to high impedance
        OUT      DX, AX
LCD_DISPLAY_1:
```

```
          MOV       DX,IO_PORT1                ;WR   is LOW。
          MOV       AX, 0004H
          OUT       DX, AX

          MOV       EBX, 00000038H                  ; SET mode
          CALL      COMMAND

          MOV       EBX, 00000001H                    ; clear the screen
          CALL      COMMAND

          MOV       EBX, 0000000FH                     ; move the cursor to the first place
          CALL      COMMAND

          MOV       TEMP_LCD1, 0010H
          MOV       TEMP_LCD2, 0041H
LCD_DISPLAY_2:
          MOV       EBX,  TEMP_LCD2                    ; display  the  characters  of  A~P.  in
sequence

          CALL      WRITE_COMMAND
          INC TEMP_LCD2
          DEC       TEMP_LCD1
          CMP       TEMP_LCD1, 0000H
          JBE       LCD_DISPLAY_3
          JMP       LCD_DISPLAY_2
LCD_DISPLAY_3:
          MOV       EBX, 000000C0H                     ; new line
          CALL      COMMAND

          MOV       TEMP_LCD1, 0010H
          MOV       TEMP_LCD2, 0051H
LCD_DISPLAY_4:
          MOV        EBX,  TEMP_LCD2                    ;display  the  characters  of  Q~\  in
sequence
          CALL      WRITE_COMMAND
          INC TEMP_LCD2
          DEC       TEMP_LCD1
          CMP       TEMP_LCD1, 0000H
```

```
            JBE         EXIT
            JMP        LCD_DISPLAY_4
COMMAND:                                            ;LCD_DISPLAY COMMAND
            MOV         DX,IO_PORT1                 ;WR is LOW, EN is HIGH
            MOV         AX, 0005H
            OUT        DX, AX

            MOV         DX,IO_PORT0                 ;feed into 01H
            MOV         EAX, EBX
            OUT        DX, EAX

            MOV         DX, IO_PORT1                 ;EN is LOW
            MOV         AX,0004H
            OUT        DX,AX

            MOV         DX,IO_PORT1                 ;RS and WR are HIGH
            MOV         AX,0010H
            OUT        DX,AX
            CALL       WAIT_1S
            CALL       WAIT_1S
            RET

WRITE_COMMAND:                                        ;LCD write characters
            MOV         DX,IO_PORT1
            MOV         AX,0000H
            OUT        DX,AX

            MOV         DX,IO_PORT1
            MOV         AX,0014H
            OUT        DX,AX

            MOV         DX,IO_PORT1
            MOV         AX,0015H
            OUT        DX,AX

            MOV         DX,IO_PORT0
            MOV         EAX,EBX
            OUT        DX,EAX
```

```
        MOV     DX,IO_PORT1
        MOV     AX,0014H
        OUT     DX,AX

        MOV     DX,IO_PORT1
        MOV     AX,0000H
        OUT     DX,AX
        CALL    WAIT_1MS

        MOV     DX,IO_PORT1
        MOV     AX,0010H
        OUT     DX,AX
        CALL    WAIT_1S
        RET
        JMP     EXIT
WAIT_3S:
        MOV     TEMP_LOOP,07H
WAIT_3S_LOOP:
        CALL    WAIT_2S
        DEC     TEMP_LOOP
        CMP     TEMP_LOOP,00H
        JBE     WAIT_3S_EXIT
        JMP     WAIT_3S_LOOP
WAIT_3S_EXIT:
        RET

WAIT_2S:
        CALL    WAIT_1S
        CALL    WAIT_1S
        CALL    WAIT_1S
        CALL    WAIT_1S
        CALL    WAIT_1S
        CALL    WAIT_1S
        RET
WAIT_1S:
        MOV     BX,0007FH
WAIT_LOOP:
```

```
            CALL      WAIT_1MS
            DEC       BX
            CMP       BX,0000H
            JBE       WAIT_1S_EXIT
            LOOP      WAIT_LOOP
WAIT_1S_EXIT:
            RET

WAIT_1MS:
            MOV       CX,03FFFH
WAIT_LOOP1:
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            LOOP      WAIT_LOOP1
            RET

WAIT_2MS:
            MOV       CX,07FFH
WAIT_LOOP2:
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            MOV       BX,BX
            LOOP      WAIT_LOOP2
            RET
EXIT:
            MOV       DX,OUT_DISABLED                 ;clear all settings
            MOV       AX,0000H
            OUT       DX,AX
            MOV       DX,IO_PORT0D
```

```
        MOV        EAX,00000000H
        OUT        DX,EAX
        MOV        DX,IO_PORT1D
        MOV        AX,0000H
        OUT        DX,AX

        MOV        AH,4CH
        INT        21H
        END        BEGIN
```

9.7 LCD (VC/C++)

VC/C++ program code:
 (LCD returns to zero)

```
//Set all IO Port
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x18,
                BitSize32,
                &RegValue);
//Set output to LCD
        RegValue = 0x0005;      0000_0101
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//output information to LCD
        RegValue = 0x0038;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
// disable LCD unit
        RegValue = 0x0004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//LCD_RS is high
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//Set output to LCD
        RegValue = 0x0005;      0000_0101
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//output information to LCD
        RegValue = 0x0001;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//Disable LCD unit
        RegValue = 0x0004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//LCD_RS is high
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//Set output to LCD
        RegValue = 0x0005;     0000_0101
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//output information to LCD
        RegValue = 0x0000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//disable LCD unit
        RegValue = 0x0004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//LCD_Rs is high
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//set all IO Port
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x18,
                BitSize32,
                &RegValue);
```

(Move the cursor to the first place)

```
//set the output to LCD
        RegValue = 0x0005; 0000_0101
        arc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//output information to LCD
        RegValue = 0x000f;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//disable LCD unit.
        RegValue = 0x0004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//LCD_RS is high
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
```

(newline)

```
//set the output to LCD
        RegValue = 0x0005;     0000_0101
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
```

```
//output information to LCD
        RegValue = 0x00c0;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
//Disable LCD unit
        RegValue = 0x0004;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
//LCD_RS is high
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
```

(Write into instructions)

```
        RegValue = 0x0000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
        RegValue = 0x0014;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
        RegValue = 0x0015;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
        RegValue = 0x0000;    Value to be outputted
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x10,
                BitSize32,
                &RegValue);
        RegValue = 0x0014;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
        RegValue = 0x0000;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
        RegValue = 0x0010;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x14,
                BitSize32,
                &RegValue);
```

9.8 8254 timer and counter

Experiment purpose: Write programs to enable 8254 to produce square wave output in mode 3.

MASM program codes:

```
. MODEL SMALL
.386
```

```
. STACK

. DATA
P54C0     EQU        0A400H              ;count 0 bit address of 82C54
P54CR      EQU        0A403H              ; control port bit address of 82C54
M054       EQU        16H                  ; count 0 is mode=3, only load LSB, binary counter
                                             ;0001_0110
. CODE
BEGIN:
            PUSH      DS
            MOV        AX, 0
            PUSH      AX
            MOV        AX, @DATA
            MOV        DS, AX

            MOV        DX, P54CR
            MOV        AL, M054
            OUT        DX, AL                    ;write into the control port.
            MOV        DX,P54C0
            MOV        AX,250
            OUT        DX,AL                     ;Load counter value 250, and the output is 10MHz/250
;           MOV        AL,AH
;           OUT        DX,AL
EXIT:
            MOV        AH,4CH
            INT        21H
            END        BEGIN
```

VC/C++ Program code:

```
//Load mode setting//
     RegValue = 0x0016;
      rc=PlxIoPortWrite(
            hDevice,
            port + 0x03,
            BitSize32,
            &RegValue);
```

```
//Set counter values //
        RegValue = 0x0250;
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x00,
                BitSize32,
                &RegValue);
```

Article 5 Advanced combined languages and C/C++ program language samples

D/A converter is an important device for computer experiments. Because the programming for this type of device is not easy, the readers can study the specifications of chips in the disc in this book on his/her own to write programs. The related circuit diagram is shown in Figure 5-0-1 below.

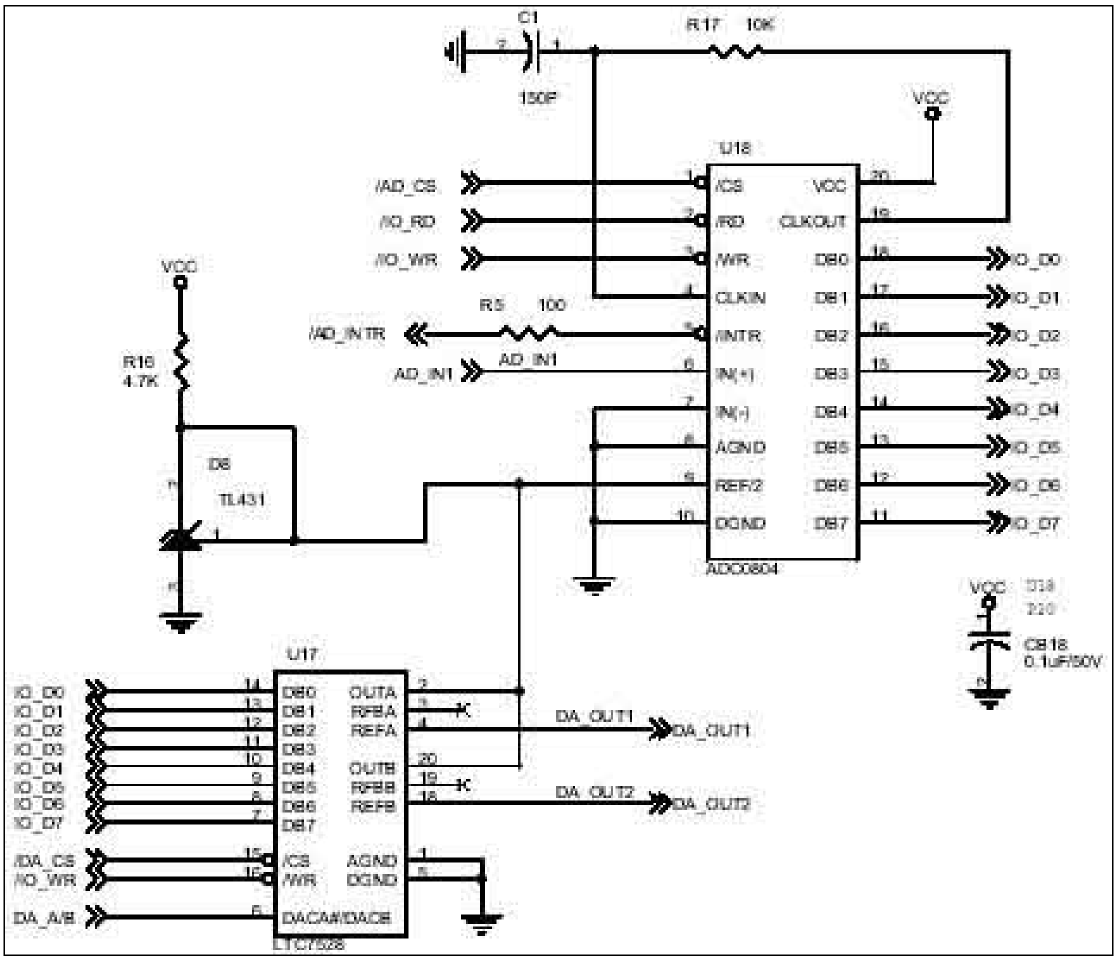


Figure 5-0-1 PCI_LAB experiment board A/D and D/A module circuit diagram

Chapter 10 Digital/analog converter

D/A converter is an essential element to the control and application of the PC. The four characteristics that require attention for common D/A converters are as follows:

(1) Revolution
(2) Linearity
(3) Setting time
(4) Accuracy

The advantages and disadvantages of D/A converters are related to the above four characteristics. Generally speaking, the higher the resolution is, the better the linearity, and the ones that have faster setting time and higher accuracy have better functions. The following is the description of the above characteristics:

(1) Revolution

The resolution of a D/A converter depends on the number of binary bits and the relationships between the two can be expressed by the following equation. In which n is the bit numbers of the converter, the number of ladders that can be produced by a n bit D/A converter is 2^n-1.

D/A converter DAC0808 is an 8 bit digital-to-analog converter, whose resolution is 1/256, which can produce 256 ladder waves; each ladder is equal to the increment of one LSB, namely 1/256. The larger the number of binary bits for the converter, the smaller the conversion error is; conversely, the higher the resolution is, the larger the conversion error. Currently the bit numbers of D/A converters are divided into four types: 8, 10, 12, 16, due to the higher resolution of 12, 16 bits DAC, their prices are higher; therefore, they are often used in more accurate control and experiments.

(2) Linearity

The second characteristic of D/A converter is Linearity. Linearity usually refer s to the same amount of changes by analog output signals when a convert starts from entering bits from low potential (0000_0000) to gradually changing to “all the input bits are high potential (1111_1111). As long as the maximum error of the D/A converter does not exceed ± 1/2 of the value of lowest sub-bit, namely ± 1/2 LSB, it is normal. The specifications of D/A converters currently available on the market are mostly equal to or more than ± 1/2 LSB.

(3) Setting time

The third characteristic of D/A converter is setting time. Setting time is the time the digital information starts to convert to obtain a stable output value (final value ± 1/2 LSB) after it is input. Generally speaking, the shorter the setting, the better, indicating that its response is good and the switching speed is fast.
The length of setting time is related to the changes of bits. When the input bit is switched from low potential status to high potential status, the setting time required is longer. Contrastingly, when the input bit is switched from high potential status to low potential status, the required setting time is relatively shorter.

(4) Accuracy

The accuracy of a D/A converter is determined by the difference between the actual output and ideal output. The smaller the difference is, the higher the accuracy. Accuracy can be divided into two types: absolute accuracy and relative accuracy. Absolute accuracy refers to the extent to which the actual output value is close to the ideal output value, while relative accuracy refers to the extent to which the actual output value is close to the ideal full-scale output value. Full-scale output value is the corresponding output value when the input bits of the D/A converter are all of high potential. Usually accuracy is expressed in percentages.

VC/C++ Sample program
（D/A output selection）

```
    RegValue = 0x0026;   A     0010_0110
    rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);
    RegValue = 0x0006;   B     0000_0110
    rc=PlxIoPortWrite(
            hDevice,
            port + 0x14,
            BitSize32,
            &RegValue);
```

（Writ-in information ）

```
    RegValue = 0x0000; Encountering written data
    rc=PlxIoPortWrite(
            hDevice,
            port + 0x10,
            BitSize32,
            &RegValue);
```
（IO_PORT setting）
```
    RegValue = 0x0010;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x18,
        BitSize32,
        &RegValue);
```

P.S. Program code should be aligned with references to D/A chips, just like VC/C++ program codes of liquid crystal display so that output form D/A converter can be effective.

Chapter 11 Analog/digital converter.

The device that converts analog signals into digital signal is called A/D converter, and there are many converting methods, such as integration method, successive approximation method, parallel method, integration methods and counter methods.

The characteristics of A/D converter are similar to that of D/A converter. However, there are still some differences, generally its characteristics can be roughly divided into four types. Before using A/D converter, the user should first refer to its characteristic so as to use it in interface designs effectively.

(1) Analog input voltage
(2) Resolution
(3) Switching time
(4) Digital output format

The above characteristics are described as follows.

Analog input voltage

Usually A/D converter can accept limited analog input voltage, so, before using, you have to know the limitations of input to A/D converter analog voltage. Otherwise, the converter may be damaged. Part of the A/D converters allows only univocal input voltage, namely the input voltage is either positive voltage or negative voltage. Some A/D converter can allow dual polar input voltage, namely the input voltage can be either positive or negative. Due to the different specifications of A/D converters, the proper values should be obtained from the information handbooks provided by the manufacturers. The common typical voltage values are: 0~+10V, 0~-10V, +5V~-5V etc.

Resolution

The resolution factor of A/D converter is similar to that of D/A converter, determining the binary bits outputted by converter, which is $1/2^n$ of resolution, of which n is the number of bit number. Usually the larger the number of bits the converter is, the better the resolution. In case of different maximum output voltage, the output level is also different.

Converting time

The time from AD convert's starting to convert analog input voltage to producing stable digital data is called conversion time. A shorter conversion time usually means a faster converting speed. Generally, the higher the resolution is, the slower the converting speed. To make the converting speed faster, the A/D converter price may be higher.

Digital output format

To be used in various different systems, A/D converter gas output code of the following formats; unipolar binary code, unipolar BCD code, offset binary code, one's complement and two's complement are available for choosing. Which kind of input code will be selected depends on the needs of actual lines. When the input ranges of A/D converters are all positive values or negative values, then binary code and reverse binary code are usually selected. When the input ranges of the A/D converter can be either positive or negative(dual-polar), offset binary code is usually selected.

VC/C++ sample program:

```
（A/D input）
    RegValue = 0x000b;    0000_1011
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x14,
        BitSize32,
        &RegValue);
（read out information）          The value of RegValue is the data fetched
    rc=PlxIoPortRead(
        hDevice,
        port + 0x10,
        BitSize32,
        &RegValue);
（IO_PORT setting）
    RegValue = 0x0010;
    rc=PlxIoPortWrite(
        hDevice,
        port + 0x18,
        BitSize32,
        &RegValue);
```

P.S. Program code should be aligned with references to A/D chips, just like VC/C++ program codes of liquid crystal display so that input form A/D converter can be effective.

Chapter 12 Project and IO-Port setting program

This PCI-IO interface panel can use 192 IO, which is presented by each 48 IO of four IO_BANK. Planned with eight IO as an IO_PORT, you can go straight to the self-made IO experiment board. With the experiment outward-pull module self-developed by the lab of our university as shown in Figure 12-0 below. The IO of PCI-IO board can be externally connected to the self-designed circuit, and LEE-PU electronics also have similar products.

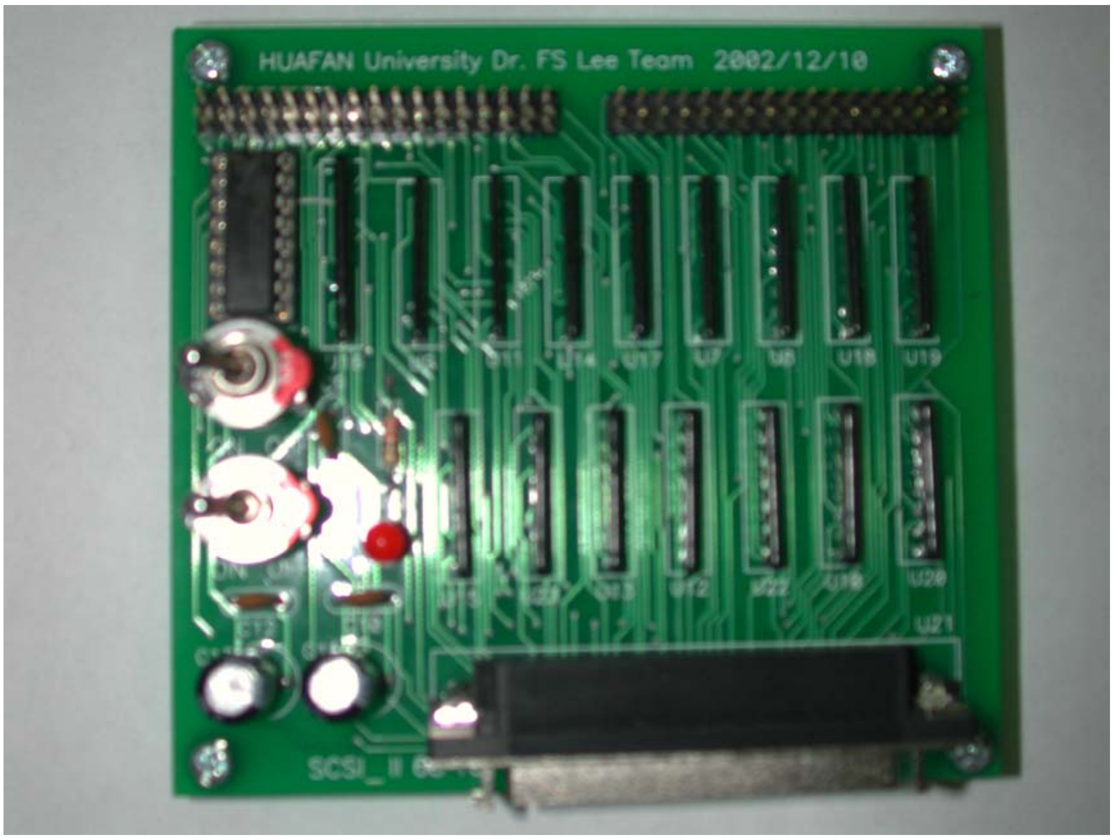


Figure 12-0 Externally connected module self-manufactured by this lab.

As for special case manufacturing, since it is necessary to set the output or input of IO_PORT, program codes are provided here for the reader to use. Related output /input information can be obtained by opening this test file with notebook and the program codes are as follows.

```
#include <stdio.h>
#include "PlxApi.h"
#include "PciRegs.h"
#include "PlxInit.h"  //add different Include File based on the needs of an individual program)
```

```
int main()       //main program //
{
      U8                        Revision,code;
      FILE                      *f;
      U16                       i;
      U32                       ChipType;
      U32                       LocalAddress=0;
      S8                        DeviceSelected;
      HANDLE                      hDevice;
      RETURN_CODE          rc;
      DEVICE_LOCATION Device;
      IOP_SPACE              IopSpace;
      U32                       port,pp,RegValue;
      U32                       buffer[64];          // define the parameter, increase upon
lack of parameters by yourself//

      DeviceSelected = SelectDevice( &Device );      //Select interface card//
      rc = PlxPciDeviceOpen( &Device, &hDevice );   //PCI-IO opening card
action //

      port=PlxPciConfigRegisterRead(
            Device.BusNumber,
            Device.SlotNumber,
            CFG_BAR3,
            &rc);                              //read base address//
      buffer[0] = 0x00000000;
      port = port & ~(1<<0);      //return-to-zero and reset action //

      PlxChipTypeGet( hDevice, &ChipType, &Revision );    //IO Port setting //
            IopSpace = IopSpace0;
            IopSpace = IopSpace1;
      RegValue = 0x00;
      pp=0x00;
      f=fopen("test","w");
      i=0x00;

for(i=0x00;i<0x49;)
```

```
{
        printf("\n BASE %x 8",i);
        fprintf(f,"\n BASE %x 8",i);
for(RegValue=0x00;RegValue<0x100;)
{
        printf("\n setting0 %x",RegValue);
        fprintf(f,"\n setting %x",RegValue);
        rc=PlxIoPortWrite(
                hDevice,
                port + 0x08 +i,
                BitSize32,
                &RegValue);
        for(pp=0x00;pp<0x06;pp=pp+0x01)
        {
              rc=PlxIoPortRead(
                hDevice,
                port + pp,
                BitSize32,
                &code);
              printf("\n bank0 code %x",code);
              fprintf(f,"\n bank0 code %x",code);
        }
        pp=0x10;
        for(pp=0x10;pp<0x16;pp=pp+0x01)
        {
              rc=PlxIoPortRead(
                hDevice,
                port + pp,
                BitSize32,
                &code);
              printf("\n bank1 code %x",code);
              fprintf(f,"\n bank1 code %x",code);
        }
        pp=0x20;
        for(pp=0x20;pp<0x26;pp=pp+0x01)
        {
              rc=PlxIoPortRead(
                hDevice,
                port + pp,
                BitSize32,
                &code);
              printf("\n bank2 code %x",code);
              fprintf(f,"\n bank2 code %x",code);
        }
        pp=0x30;
        for(pp=0x30;pp<0x36;pp=pp+0x01)
        {
              rc=PlxIoPortRead(
                hDevice,
                port + pp,
                BitSize32,
                &code);
              printf("\n bank3 code %x",code);
              fprintf(f,"\n bank3 code %x",code);
        }
        pp=0x40;
        for(pp=0x40;pp<0x46;pp=pp+0x01)
        {
              rc=PlxIoPortRead(
                hDevice,
                port + pp,
                BitSize32,
                &code);
              printf("\n bank4 code %x",code);
              fprintf(f,"\n bank4 code %x",code);
        }
        RegValue=RegValue+0x01;
}
i=i+0x10;
}
fclose(f);
        return 1;
}
```

This program first sets the setting of IO_BANK, then executes the setting of 0x08, fills the values from 0x00h to 0xFFh into the setting port, fetches the output or input status of IO_PORT in various IO_BANK, then executes the setting of 0x18, 0x28, 0x38, 0x48. In executing this program, IO combinations meeting various demands can be found, allowing the use of the 32 bit output or input status.