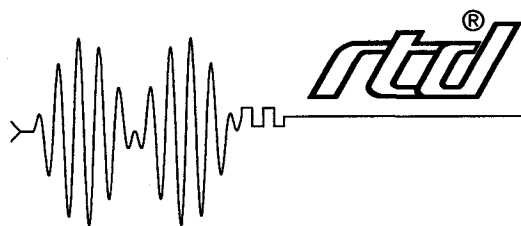


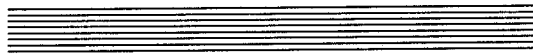
DIO24

User's Manual



Real Time Devices, Inc.

"Accessing the Analog World"™



DIO24



User's Manual



REAL TIME DEVICES, INC.
Post Office Box 906
State College, Pennsylvania 16804

Phone: (814) 234-8087
FAX: (814) 234-5218

Published by
Real Time Devices, Inc.
P.O. Box 906
State College, PA 16804

Copyright © 1992 by Real Time Devices, Inc.
All rights reserved

Printed in U.S.A.

Table of Contents

| | |
|---|------------|
| INTRODUCTION | i-1 |
| Digital I/O | i-3 |
| What Comes With Your Board | i-3 |
| Board Accessories | i-3 |
| Using This Manual | i-4 |
| When You Need Help..... | i-4 |
| CHAPTER 1 — BOARD SETTINGS | 1-1 |
| Factory-Configured Switch and Jumper Settings | 1-3 |
| P3 — Interrupt and Interrupt Channels (Factory Setting: GND Connected; Interrupt Channels Disabled) | 1-4 |
| P4 — Interrupt Source Select (Factory Setting: EXT) | 1-4 |
| P5 — 8254 Timer/Counter Clock Sources (Factory Settings: CLK0-XTAL, CLK1-OUT0, CLK2-OUT1) | 1-5 |
| S1 — Base Address (Factory Setting: 300 hex (768 decimal)) | 1-6 |
| S2 — Buffer Bypass Switch (Factory Setting: OPEN (Not Bypassed)) | 1-7 |
| F1 — External +5-volt Fuse | 1-7 |
| Pull-up/Pull-down Resistors on Digital I/O Lines | 1-8 |
| CHAPTER 2 — BOARD INSTALLATION | 2-1 |
| Board Installation | 2-3 |
| External I/O Connections | 2-4 |
| Connecting the Digital I/O | 2-4 |
| Connecting the Timer/Counter I/O..... | 2-4 |
| Connecting the External Interrupt | 2-4 |
| Running the 24DIAG Diagnostics Program | 2-4 |
| CHAPTER 3 — HARDWARE DESCRIPTION | 3-1 |
| Digital I/O, 8255 Programmable Peripheral Interface..... | 3-3 |
| Timer/Counters | 3-4 |
| Interrupts | 3-4 |
| CHAPTER 4 — BOARD OPERATION AND PROGRAMMING | 4-1 |
| Defining the I/O Map | 4-3 |
| BA + 0: PPI Port A — Digital I/O (Read/Write) | 4-3 |
| BA + 1: PPI Port B — Digital I/O (Read/Write) | 4-3 |
| BA + 2: PPI Port C — Digital I/O (Read/Write) | 4-3 |
| BA + 3: 8254 PPI Control Word (Write Only) | 4-4 |
| BA + 4: IRQ Enable (Write Only) | 4-5 |
| BA + 5: Interrupt Status/Clear (Read/Write) | 4-5 |
| BA + 6: Reserved | 4-6 |
| BA + 7: Reserved | 4-6 |
| BA + 8: 8254 Timer/Counter 0 (Read/Write) | 4-6 |
| BA + 9: 8254 Timer/Counter 1 (Read/Write) | 4-6 |
| BA + 10: 8254 Timer/Counter 2 (Read/Write) | 4-6 |
| BA + 11: 8254 Control Word (Write Only) | 4-6 |
| Programming the DIO24 | 4-7 |
| Clearing and Setting Bits in a Port | 4-8 |
| Initializing the 8255 PPI..... | 4-9 |

| | |
|---|------------|
| Digital I/O Operations | 4-9 |
| Timer/Counters | 4-9 |
| Interrupts | 4-11 |
| What Is an Interrupt? | 4-11 |
| Interrupt Request Lines | 4-11 |
| 8259 Programmable Interrupt Controller | 4-11 |
| Interrupt Mask Register (IMR) | 4-11 |
| End-of-Interrupt (EOI) Command | 4-11 |
| What Exactly Happens When an Interrupt Occurs? | 4-11 |
| Using Interrupts in Your Programs | 4-12 |
| Writing an Interrupt Service Routine (ISR) | 4-12 |
| Saving the Startup Interrupt Mask Register (IMR) and Interrupt Vector | 4-13 |
| Restoring the Startup IMR and Interrupt Vector | 4-14 |
| Common Interrupt Mistakes | 4-14 |
| Example Programs | 4-14 |
| C and Pascal Programs | 4-14 |
| BASIC Programs | 4-14 |
| APPENDIX A — DIO24 SPECIFICATIONS | A-1 |
| APPENDIX B — I/O CONNECTOR PIN ASSIGNMENTS | B-1 |
| APPENDIX C — COMPONENT DATA SHEETS | C-1 |
| APPENDIX D — WARRANTY | D-1 |

LIST OF ILLUSTRATIONS

| | | |
|------|--|------|
| 1-1 | Board Layout Showing Factory-Configured Settings | 1-3 |
| 1-2 | Interrupt and Interrupt Channel Jumper, P3 | 1-4 |
| 1-3 | Pulling Down the Interrupt Request Line | 1-4 |
| 1-4 | Interrupt Source Select Jumper, P4 | 1-5 |
| 1-5 | 8254 Timer/Counter Clock Source Jumpers, P5 | 1-5 |
| 1-6 | 8254 Timer/Counter Circuit Block Diagram | 1-6 |
| 1-7 | Base Address Switch, S1 | 1-6 |
| 1-8 | Port C Buffer Circuitry | 1-8 |
| 1-9 | Port A Pull-up/Pull-down Resistor Circuitry | 1-9 |
| 1-10 | Adding Pull-ups and Pull-downs to Some Digital I/O Lines | 1-9 |
| 2-1 | P2 and P6 I/O Connector Pin Assignments | 2-4 |
| 3-1 | DIO24 Block Diagram | 3-3 |
| 4-1 | 8254 Timer/Counter Circuit Block Diagram | 4-10 |

INTRODUCTION

The DIO24 series boards are general purpose digital I/O board for use in the IBM PC/XT/AT or compatible computer. The DIO24 series has two models: the DIO24-1 and the DIO24-2 with timer/counters. Installed within a single short or full-size expansion slot in the computer, the DIO24 features:

- 24 TTL/CMOS 8255-based programmable digital I/O lines,
- Direct connection to opto-22 I/O system modules,
- Buffered outputs for high driving capability,
- Three 16-bit timer/counters (DIO24-2 model only),
- Optional pull-up/pull-down resistors,
- Simple I/O or strobed I/O operation,
- Software enabled interrupts (IRQ2-IRQ7),
- Assembly, BASIC, Turbo Pascal, and Turbo C source code; diagnostics program.

The following paragraphs briefly describe the major function of the board. A more detailed discussion of board functions is included in Chapter 3, *Hardware Operation*, and Chapter 4, *Board Operation and Programming*. The board setup is described in Chapter 1, *Board Settings*.

Digital I/O

The DIO24 has 24 TTL/CMOS-compatible digital I/O lines which can be directly interfaced with external devices or signals to sense switch closures, trigger digital events, or activate solid-state relays. These lines are provided by the on-board 8255 programmable peripheral interface chip. The 8255 can be operated in one of two modes: Mode 0 or Mode 1. To ensure high driving capacity, CMOS buffers are installed. TTL buffers are available on request.

Pads for installing and activating pull-up or pull-down resistors are included on the board. Installation procedures are given at the end of Chapter 1, *Board Settings*.

8254 Timer/Counter (DIO24-2)

An 8254 programmable interval timer contains three 16-bit, 8 MHz timer/counters to support a wide range of timing and counting functions. The clock, gate, and output pins for each of the timer/counters are available at P6, a 20-pin on-board box header connector.

What Comes With Your Board

You receive the following items in your DIO24 package:

- DIO24-1 or DIO24-2 (with timer/counters) interface board
- Software and diagnostics diskette with Assembly, BASIC, Turbo Pascal, and Turbo C source code
- User's manual

If any item is missing or damaged, please call Real Time Devices' Customer Service Department at (814) 234-8087. If you require service outside the U.S., contact your local distributor.

Board Accessories

In addition to the items included in your DIO24 package, Real Time Devices offers a full line of accessories. Call your local distributor or our main office for more information about these accessories and for help in choosing the best items to support your board's application.

Accessories for the DIO24 include the TB50 terminal board and XB50 prototype/terminal board for prototype development and easy signal access, the XO50 cable assembly for direct connection to opto-22 systems, and the TW50 twisted pair wire flat ribbon cable assembly for external interfacing.

Using This Manual

This manual is intended to help you install your new board and get it running quickly, while also providing enough detail about the board and its functions so that you can enjoy maximum use of its features even in the most complex applications. We assume that you already have an understanding of data acquisition principles and that you can customize the example software or write your own applications programs.

When You Need Help

This manual and the example programs in the software package included with your board provide enough information to properly use all of the board's features. If you have any problems installing or using this board, contact our Technical Support Department, (814) 234-8087, during regular business hours, eastern standard time or eastern daylight time, or send a FAX requesting assistance to (814) 234-5218. When sending a FAX request, please include your company's name and address, your name, your telephone number, and a brief description of the problem.

CHAPTER 1

BOARD SETTINGS

The DIO24 board has jumper and switch settings you can change if necessary for your application. The board is factory-configured with the most often used settings. The factory settings are listed and shown on a diagram in the beginning of this chapter. Should you need to change these settings, use these easy-to-follow instructions before you install the board in your computer.

Note that DIP switch S2 has been provided to bypass the Port C buffers and allow Mode 1 operation of the 8255.

Also note that by installing resistor packs at three locations around the 8255 PPI and soldering jumpers in the desired locations in the associated pads, you can configure your digital I/O lines to be pulled up or pulled down. This procedure is explained at the end of this chapter.

Factory-Configured Switch and Jumper Settings

Table 1-1 lists the factory settings of the user-configurable jumper and switches on the DIO24 board. Figure 1-1 shows the board layout and the locations of the factory-set jumpers. The following paragraphs explain how to change the factory settings. Pay special attention to the setting of S1, the base address switch, to avoid address contention when you first use your board in your system.

| Table 1-1 — Factory Settings | | |
|------------------------------|---|---|
| Switch/Jumper | Function Controlled | Factory Settings (Jumpers Installed) |
| P3 | Connects 1 of 6 interrupt sources to an interrupt channel; pulls tri-state buffer to ground (GND) for multiple interrupt applications | GND (ground for buffer) connected; interrupt channels disabled |
| P4 | Selects the interrupt source | EXT (external interrupt) |
| P5 | Sets the clock sources for the three 8254 timer/counters (TC0-TC2) | CLK0-XTAL; CLK1-OUT0; CLK2-OUT1 (all three timer/counters are cascaded) |
| S1 | Sets the base address | 300 hex (768 decimal) |
| S2 | Bypasses Port C buffers for Mode 1 operation | Open (buffers not bypassed) |

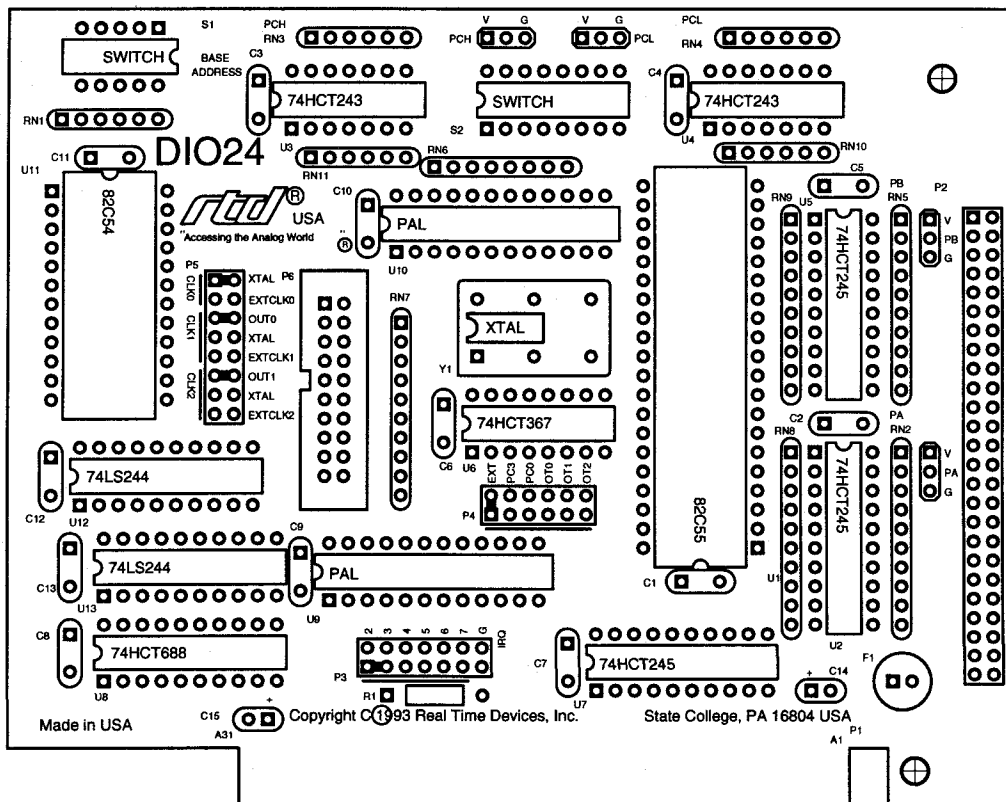


Fig. 1-1 — Board Layout Showing Factory-Configured Settings

P3 — Interrupt and Interrupt Channels (Factory Setting: GND Connected; Interrupt Channels Disabled)

This header connector, shown in Figure 1-2, lets you connect an interrupt source selected on P4 to an interrupt channel, IRQ2 through IRQ7. To connect the interrupt source to an interrupt channel, you must install a jumper across the desired IRQ channel.

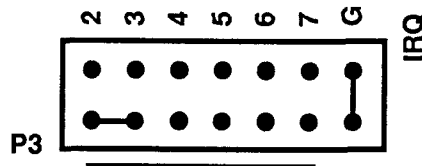


Fig. 1-2 — Interrupt and Interrupt Channel Jumper, P3

The rightmost pair of pins on P3, labeled G, are provided so that you can install a jumper which connects a 1 kilohm pull-down resistor to the output of a high-impedance tri-state driver which carries the interrupt request signal. This pull-down resistor drives the interrupt request line low whenever interrupts are not active. So, whenever an interrupt request is made, the tri-state buffer is enabled, forcing the output high and causing an interrupt. You can monitor the interrupt status through bit 0 in the status word (I/O address location BA + 5). After the interrupt has been serviced, the clear command returns the IRQ line low, disabling the tri-state buffers, and pulling the output low again. Figure 1-3 shows this circuit. Because the interrupt request line is driven low only by the pull-down resistor, you can have two or more boards which share the same IRQ channel. You can tell which board issued the interrupt request by monitoring each board's IRQ status bit.

NOTE: When you use multiple boards that share the same interrupt, only one board should have the G ground jumper installed. The rest should be disconnected. Whenever you operate a single board, the G jumper should be installed.

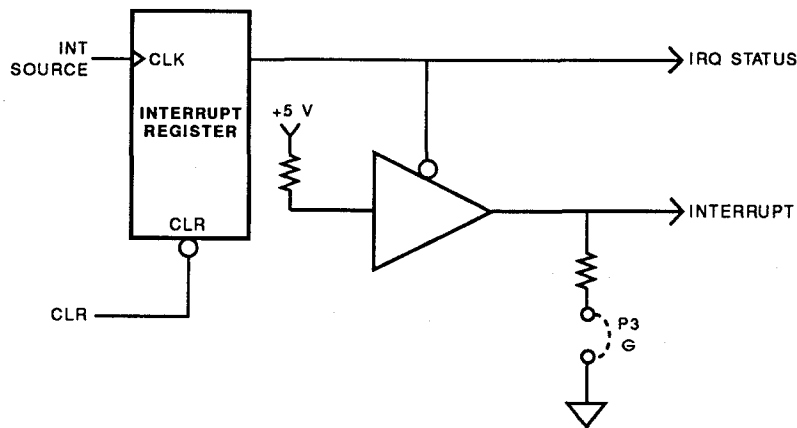


Fig. 1-3 — Pulling Down the Interrupt Request Line

P4 — Interrupt Source Select (Factory Setting: EXT)

This header connector, shown in Figure 1-4, lets you connect one of six interrupt sources for interrupt generation. These sources are: OT0, OT1, and OT2, which are the three 8254 timer/counter outputs available on the DIO24-2 model; PC3, which is the INTRA signal from the 8255 PPI; PC0, which is the INTRB signal from the 8255 PPI; and EXT, an external interrupt you can route onto the board through the P2 I/O connector. To connect an interrupt source, place the jumper across the desired set of pins. Note that only ONE interrupt source can be activated at a time.



Fig. 1-4 — Interrupt Source Select Jumper, P4

P5 — 8254 Timer/Counter Clock Sources (Factory Settings: CLK0-XTAL, CLK1-OUT0, CLK2-OUT1)

This header connector, shown in Figure 1-5, lets you select the clock sources for the 8254 timer/counters, TC0, TC1, and TC2 which are on the DIO24-2 model. The factory setting cascades all three timer/counters, with the clock source for TC0 being the on-board 8 MHz oscillator, the output of TC0 providing the clock for TC1, and the output of TC1 providing the clock for TC2. You can connect any or all of the sources to an external clock input through the P6 on-board I/O connector, or you can set TC1 and TC2 to be clocked by the 8 MHz oscillator. Figure 1-6 shows a block diagram of the timer/counter circuitry to help you with these connections.

NOTE: When installing jumpers on this header, make sure that only one jumper is installed in each group of two or three CLK pins.

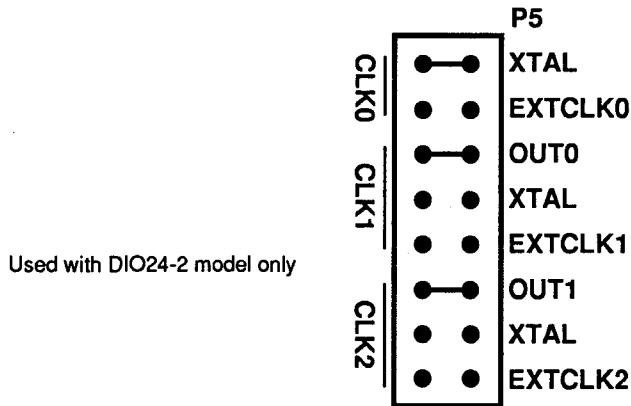


Fig. 1-5 — 8254 Timer/Counter Clock Source Jumpers, P5

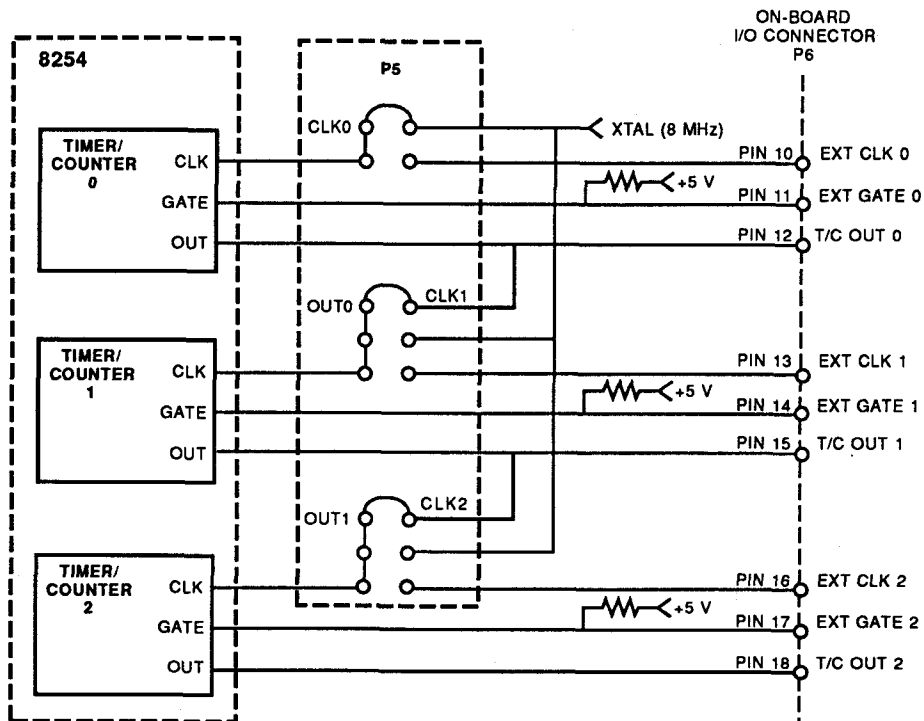


Fig. 1-6 — 8254 Timer/Counter Circuit Block Diagram

S1 — Base Address (Factory Setting: 300 hex (768 decimal))

One of the most common causes of failure when you are first trying your board is address contention. Some of your computer's I/O space is already occupied by internal I/O and other peripherals. When the DIO24 board attempts to use I/O address locations already used by another device, contention results and the board does not work.

To avoid this problem, the DIO24 has an easily accessible DIP switch, S1, which lets you select any one of 32 starting addresses in the computer's I/O. Should the factory setting of 300 hex (768 decimal) be unsuitable for your system, you can select a different base address simply by setting the switches to any value shown in Table 1-2. The table shows the switch settings and their corresponding decimal and hexadecimal (in parentheses) values. Make sure that you verify the order of the switch numbers on the switch (1 through 5) before setting them. When the switches are pulled forward, they are OPEN, or set to logic 1, as labeled on the DIP switch package. When you set the base address for your board, record the value in the table inside the back cover. Figure 1-7 shows the DIP switch set for a base address of 300 hex (768 decimal).

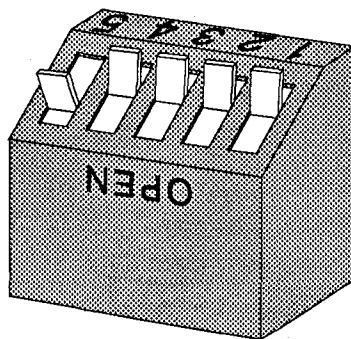


Fig. 1-7 — Base Address Switch, S1

| Table 1-2 — Base Address Switch Settings, S1 | | | |
|--|-----------------------------|---------------------------------|-----------------------------|
| Base Address Decimal / (Hex) | Switch Setting 5 4 3 2 1 | Base Address Decimal / (Hex) | Switch Setting 5 4 3 2 1 |
| 512 / (200) | 0 0 0 0 0 | 768 / (300) | 1 0 0 0 0 |
| 528 / (210) | 0 0 0 0 1 | 784 / (310) | 1 0 0 0 1 |
| 544 / (220) | 0 0 0 1 0 | 800 / (320) | 1 0 0 1 0 |
| 560 / (230) | 0 0 0 1 1 | 816 / (330) | 1 0 0 1 1 |
| 576 / (240) | 0 0 1 0 0 | 832 / (340) | 1 0 1 0 0 |
| 592 / (250) | 0 0 1 0 1 | 848 / (350) | 1 0 1 0 1 |
| 608 / (260) | 0 0 1 1 0 | 864 / (360) | 1 0 1 1 0 |
| 624 / (270) | 0 0 1 1 1 | 880 / (370) | 1 0 1 1 1 |
| 640 / (280) | 0 1 0 0 0 | 896 / (380) | 1 1 0 0 0 |
| 656 / (290) | 0 1 0 0 1 | 912 / (390) | 1 1 0 0 1 |
| 672 / (2A0) | 0 1 0 1 0 | 928 / (3A0) | 1 1 0 1 0 |
| 688 / (2B0) | 0 1 0 1 1 | 944 / (3B0) | 1 1 0 1 1 |
| 704 / (2C0) | 0 1 1 0 0 | 960 / (3C0) | 1 1 1 0 0 |
| 720 / (2D0) | 0 1 1 0 1 | 976 / (3D0) | 1 1 1 0 1 |
| 736 / (2E0) | 0 1 1 1 0 | 992 / (3E0) | 1 1 1 1 0 |
| 752 / (2F0) | 0 1 1 1 1 | 1008 / (3F0) | 1 1 1 1 1 |
| 0 = closed, 1 = open | | | |

S2 — Buffer Bypass Switch (Factory Setting: OPEN (Not Bypassed))

When operating the 8255 in Mode 1, the lines of Port C function as control lines, some as outputs and some as inputs. When using Mode 1, the Port C buffers must be removed and bypassed to allow the Port C lines to be individually set as inputs or outputs. Figure 1-8 shows the Port C buffers, and the following steps tell you how to configure the board for Mode 1 operation.

To remove buffering from Port C:

1. Close DIP switches 1 through 8 on S2.
2. Remove U3 from the board.
3. Remove U4 from the board.

CAUTION: Remember, whenever you close the switches, be sure to remove the buffers, U3 and U4, from the board. Failure to do so may damage the board.

F1 — External +5-volt Fuse

This 1 ampere fuse protects the +5 volt line available at I/O connector P2, pin 49 from drawing too much current and damaging system equipment.

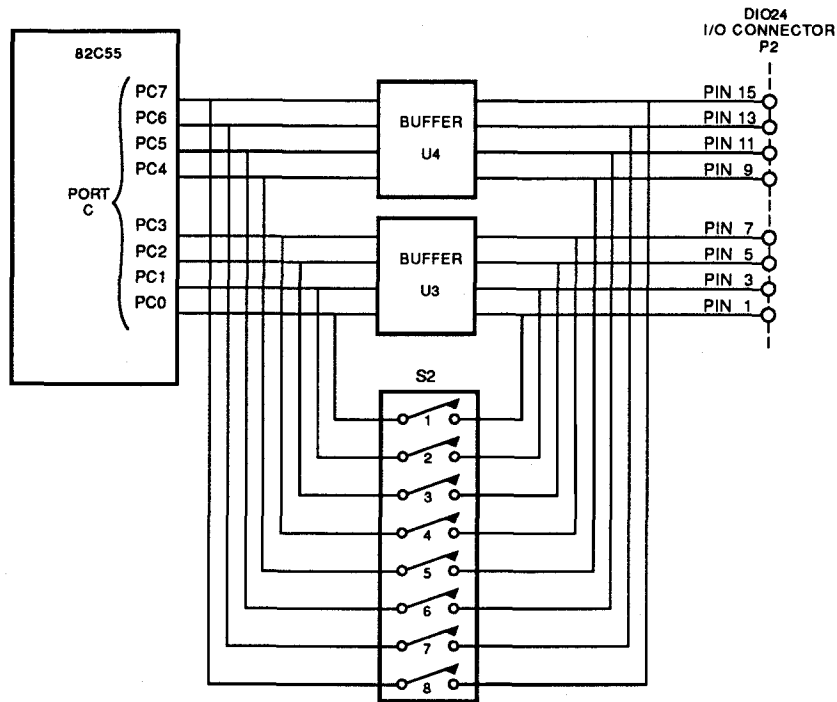


Fig. 1-8 — Port C Buffer Circuitry

Pull-up/Pull-down Resistors on Digital I/O Lines

The 8255 programmable peripheral interface provides 24 parallel TTL/CMOS compatible digital I/O lines which can be interfaced with external devices. The lines are divided into four groups: eight Port A lines, four Port C Lower lines, eight Port B lines, and four Port C Upper lines. You can install and connect pull-up or pull-down resistors for any or all of these four groups of lines. You may want to pull lines up for connection to switches. This will pull the line high when the switch is disconnected. Or, you may want to pull down lines connected to relays which control turning motors on and off. These motors turn on when the digital lines controlling them are high.

To use the pull-up/pull-down feature, you must first install 10 kilohm resistor packs in any or all of the four locations around the 8255, labeled PA, PB, PCL, and PCH. PA and PB take a 10-pin pack, and CL and CH take 6-pin packs. Figure 1-9 shows these locations.

After the resistor packs are installed, you must connect them into the circuit as pull-ups or pull-downs. Locate the three-hole pads on the board near the resistor packs. They are labeled G (for ground) on one end and V (for Vcc) on the other end. The middle hole is common. PA is for Port A, PB for Port B, CL is for Port C Lower, and CH is for Port C Upper. Figure 1-9 shows a blowup of the pads for Port A. To operate as pull-ups, solder a jumper wire between the common pin (middle pin of the three) and the V pin. For pull-downs, solder a jumper wire between the common pin (middle pin) and the G pin. For example, Figure 1-10 shows Port A lines with pull-ups, Port C Lower with pull-downs, and Port C Upper with no resistors.

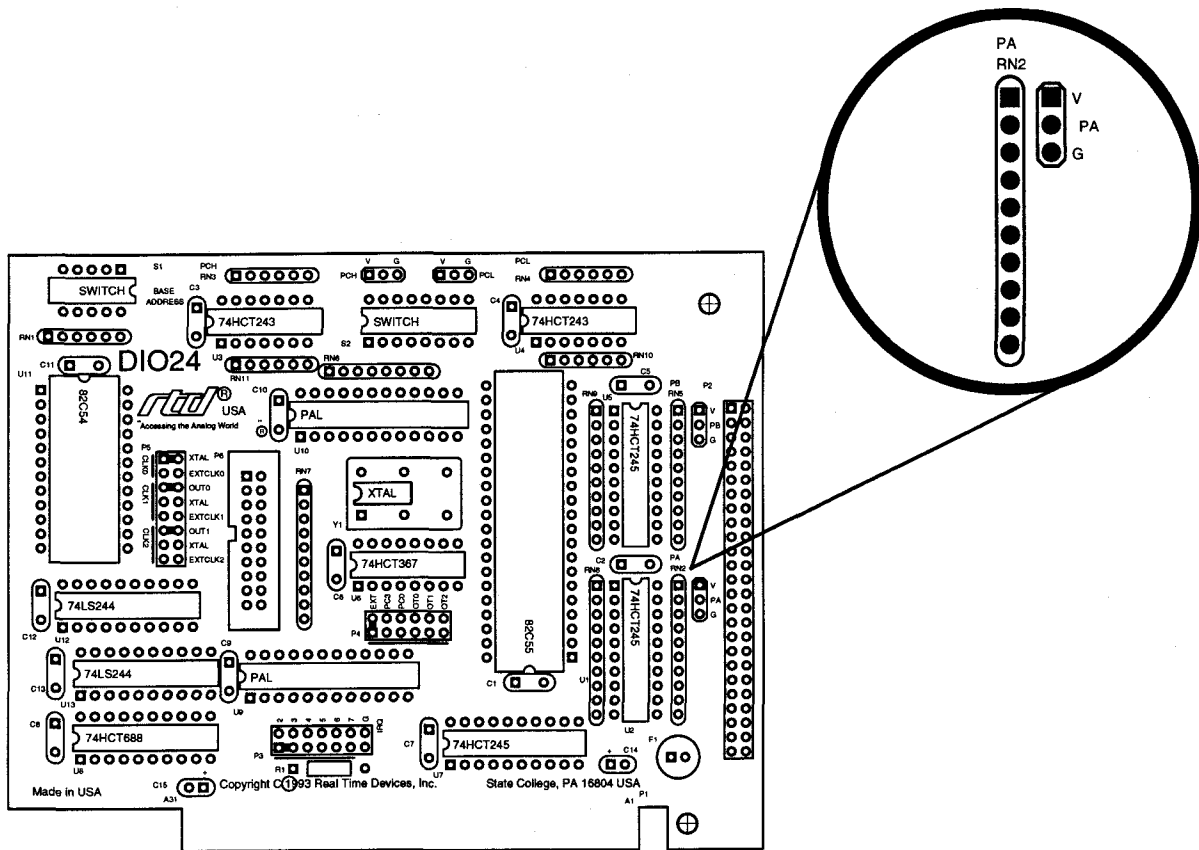


Fig. 1-9 — Port A Pull-up/Pull-down Resistor Circuitry

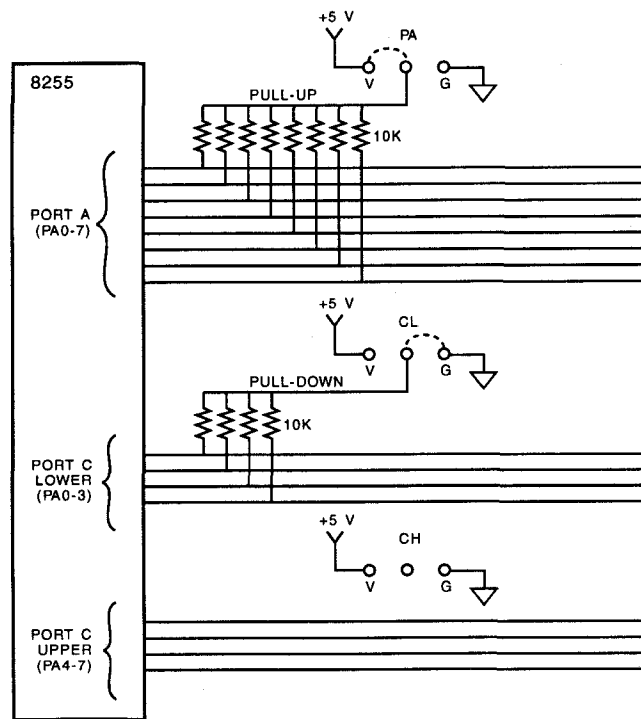


Fig. 1-10 — Adding Pull-ups and Pull-downs to Some Digital I/O Lines

CHAPTER 2

BOARD INSTALLATION

The DIO24 is easy to install in your IBM PC/XT/AT. It can be placed in any slot, short or full-size. This chapter tells you step-by-step how to install and connect the board.

After you have installed the board and made all of your connections, you can turn your system on and run the 24DIAG board diagnostics program included on your example software disk to verify that your board is working.

Board Installation

Keep the board in its antistatic bag until you are ready to install it in your computer. When removing it from the bag, hold the board at the edges and do not touch the components or connectors.

Before installing the board in your computer, check the jumper settings. Chapter 1 reviews the factory settings and how to change them. If you need to change any settings, refer to the appropriate instructions in Chapter 1. Note that incompatible jumper settings can result in unpredictable board operation and erratic response.

Also note that the P2 I/O connector mounting bracket has an oversized cutout to allow space for running the cable to 20-pin on-board connector P6 through the same I/O slot. If you want to run both cables through the same slot, you must make these connections before installing the board.

To install the board:

1. Turn OFF the power to your computer.
2. Remove the top cover of the computer housing (refer to your owner's manual if you do not already know how to do this).
3. Select any unused short or full-size expansion slot and remove the slot bracket.
4. Touch the metal housing of the computer to discharge any static buildup and then remove the board from its antistatic bag.
5. Holding the board by its edges, orient it so that its card edge (bus) connector lines up with the expansion slot connector in the bottom of the selected expansion slot.
6. After carefully positioning the board in the expansion slot so that the card edge connector is resting on the computer's bus connector, gently and evenly press down on the board until it is secured in the slot.

NOTE: Do not force the board into the slot. If the board does not slide into place, remove it and try again. Wiggling the board or exerting too much pressure can result in damage to the board or to the computer.

7. After the board is installed, secure the slot bracket back into place and put the cover back on your computer. The board is now ready to be connected via the external I/O connector at the rear panel of your computer.

External I/O Connections

Figure 2-1 shows the DIO24's P2 I/O connector pinout and P6 on-board I/O connector pinout. Refer to these diagrams as you make your I/O connections.

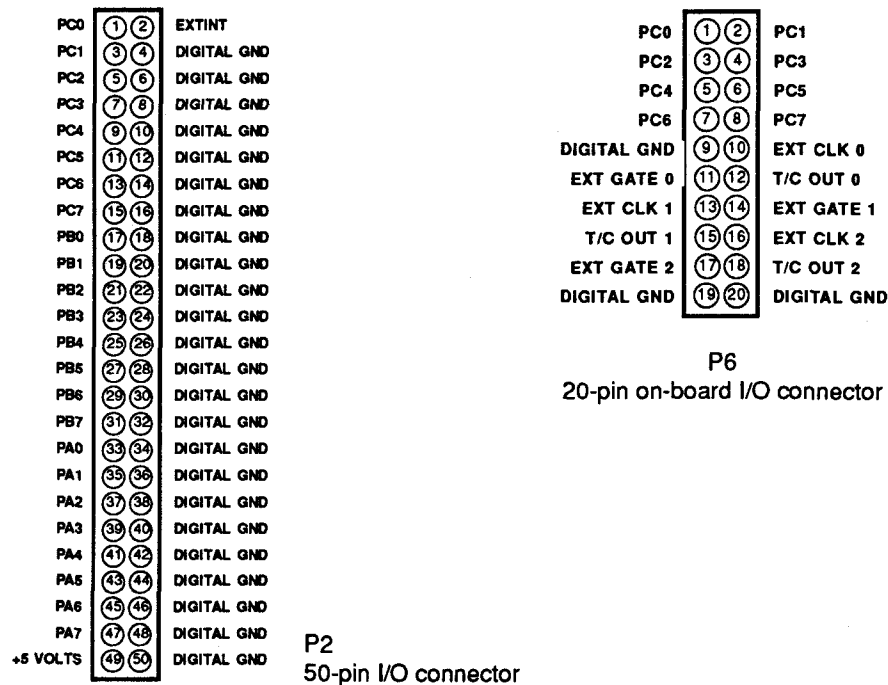


Fig. 2-1 — P2 and P6 I/O Connector Pin Assignments

Connecting the Digital I/O

The DIO24 is designed for direct connection to industry standard opto-22 isolated I/O racks and system modules. Each digital I/O line on P2 has a digital ground, as shown in Figure 2-1. For all digital I/O connections, the high side of an external signal source or destination device is connected to the appropriate signal pin on the I/O connector, and the low side is connected to the DIGITAL GND. A cable to provide direct connection to opto-22 systems is available as an accessory from RTD.

Connecting the Timer/Counter I/O

External connections to the timer/counters on the DIO24-2 model can be made by connecting the high side of the external device to the appropriate signal pin on on-board connector P6 and the low side to a P6 DIGITAL GND.

Connecting the External Interrupt

The DIO24 can receive an externally generated interrupt signal, EXTINT, through I/O connector P2, pin 2 and route it to an IRQ channel through on-board header connectors P3 and P4. Interrupt generation is enabled through software. When interrupts are enabled, a rising edge on the EXTINT line will cause the selected IRQ line to go high, and the IRQ status bit will change from 0 to 1. The pulse applied to the EXTINT pin should have a duration of at least 100 nanoseconds.

Running the 24DIAG Diagnostics Program

Now that your board is ready to use, you will want to try it out. An easy-to-use, menu-driven diagnostics program, 24DIAG, is included with your example software to help you verify your board's operation. You can also use this program to make sure that your current base address setting does not contend with another device.

CHAPTER 3

HARDWARE DESCRIPTION

This chapter describes the major features of the DIO24's 8255 based digital I/O and 8254 timer/counters. This chapter also describes the hardware-selectable interrupts.

The DIO24 provides buffered digital I/O lines, as shown Figure 3-1. In addition, the DIO24-2 model provides three 16-bit timer/counters. This chapter describes the hardware which makes up the digital I/O circuitry, timer/counters, and hardware-selectable interrupts.

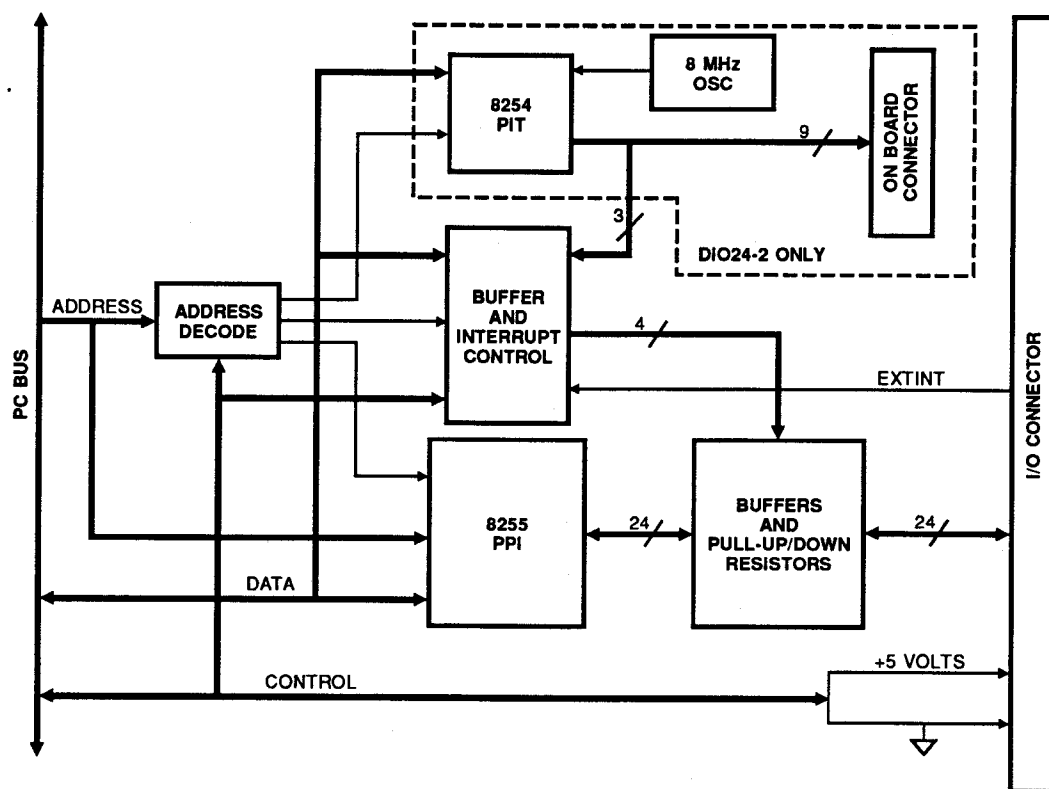


Fig. 3-1 — DIO24 Block Diagram

Digital I/O, 8255 Programmable Peripheral Interface

The 8255 programmable peripheral interface (PPI) can be easily configured to solve a wide range of digital real-world problems. This high-performance TTL/CMOS compatible chip has 24 parallel programmable digital I/O lines divided into two groups of 12 lines each:

- Group A — Port A (8 lines) and Port C Upper (4 lines);
- Group B — Port B (8 lines) and Port C Lower (4 lines).

Each group can be programmed for Mode 0 or Mode 1 operation. **Do not try to use Mode 2 operation!** The DIO24 does not support Mode 2. When operating in Mode 1, the on-board buffers must be removed from the Port C lines. This procedure is described in Chapter 1 in the S2 DIP switch discussion. The DIO24 operating modes are:

- Mode 0 — Basic input/output. Lets you use simple input and output operation for a port. Data is written to or read from the specified port.
- Mode 1 — Strobed input/output. Lets you transfer I/O data from Port A or Port B in conjunction with strobes or handshaking signals.

These modes are detailed in the 8255 Data Sheet, reprinted from Intel in Appendix C.

The bidirectional buffers on the 8255's I/O lines monitor the 8255 control word to automatically set their direction. Hardware changes to the buffer circuitry is required only when using Mode 1, where the Port C buffers must be removed as described in Chapter 1.

Timer/Counters

An 8254 programmable interval timer provides three 16-bit, 8 MHz timer/counters to support a wide range of timing and counting functions. These timer/counters can be cascaded or used individually for many applications.

Each timer/counter has two inputs, CLK in and GATE in, and one output, timer/counter OUT. The clock sources for the timer/counters can be selected using jumpers on header connector P5 (see Chapter 1). The timer/counters can be programmed as binary or BCD down counters by writing the appropriate data to the command word, as described in Chapter 4. The command word also lets you set up the mode of operation. The six programmable modes are:

| | |
|--------|---|
| Mode 0 | Event Counter (Interrupt on Terminal Count) |
| Mode 1 | Hardware-Retriggerable One-Shot |
| Mode 2 | Rate Generator |
| Mode 3 | Square Wave Mode |
| Mode 4 | Software-Triggered Strobe |
| Mode 5 | Hardware Triggered Strobe (Retriggerable) |

These modes are detailed in the 8254 Data Sheet, reprinted from Intel in Appendix C.

Interrupts

The DIO24 can use any one of six signal sources to generate interrupts. These sources are: OT0, OT1, and OT2, which are the three 8254 timer/counter outputs available on the DIO24-2 model; PC3, which is the INTRA signal from the 8255 PPI; PC0, which is the INTRB signal from the 8255 PPI; and EXT, an external interrupt you can route onto the board through the P2 I/O connector. Chapter 1 tells you how to set the jumpers on interrupt header connectors P3 and P4, and Chapter 4 provides some programming information.

CHAPTER 4

BOARD OPERATION AND PROGRAMMING

This chapter shows you how to program and use your DIO24 board. It provides a complete description of the I/O map and a detailed description of programming operations to aid you in programming. The example programs included on the disk in your board package are listed at the end of this chapter. These programs, written in Turbo C, Turbo Pascal, Assembly, and BASIC, include source code to simplify your applications programming.

Defining the I/O Map

The I/O map for the DIO24 is shown in Table 4-1 below. As shown, the board occupies 12 consecutive I/O port locations. The base address (designated as BA) can be selected using DIP switch S1 as described in Chapter 1, *Board Settings*. This switch can be accessed without removing the board from the computer. The following sections describe the register contents of each address used in the I/O map.

| Table 4-1 — DIO24 I/O Map | | | |
|---------------------------|---------------------------------|---|------------------------|
| Register Description | Read Function | Write Function | Address * (Decimal) |
| 8255 PPI Port A | Read Port A digital input lines | Program Port A digital output lines | BA + 0 |
| 8255 PPI Port B | Read Port B digital input lines | Program Port B digital output lines | BA + 1 |
| 8255 PPI Port C | Read Port C digital input lines | Program Port C digital output lines | BA + 2 |
| 8255 PPI Control Word | Not used | Program PPI configuration | BA + 3 |
| IRQ Enable | Not used | Enable and disable interrupt generation | BA + 4 |
| Interrupt Status/Clear | Read status of interrupt | Clear interrupt | BA + 5 |
| Reserved | | | BA + 6 |
| Reserved | | | BA + 7 |
| 8254 Timer/Counter 0 | Read TC0 count value | Load TC0 count register | BA + 8 |
| 8254 Timer/Counter 1 | Read TC1 count value | Load TC1 count register | BA + 9 |
| 8254 Timer/Counter 2 | Read TC2 count value | Load TC2 count register | BA + 10 |
| 8254 Control Word | Not used | Program control register | BA + 11 |
| * BA = Base Address | | | |

BA + 0: PPI Port A — Digital I/O (Read/Write)

Transfers the 8-bit Port A digital input and digital output data between the board and an external device. A read transfers data from the external device, through P2, and into PPI Port A; a write transfers the written data from Port A through P2 to an external device.

BA + 1: PPI Port B — Digital I/O (Read/Write)

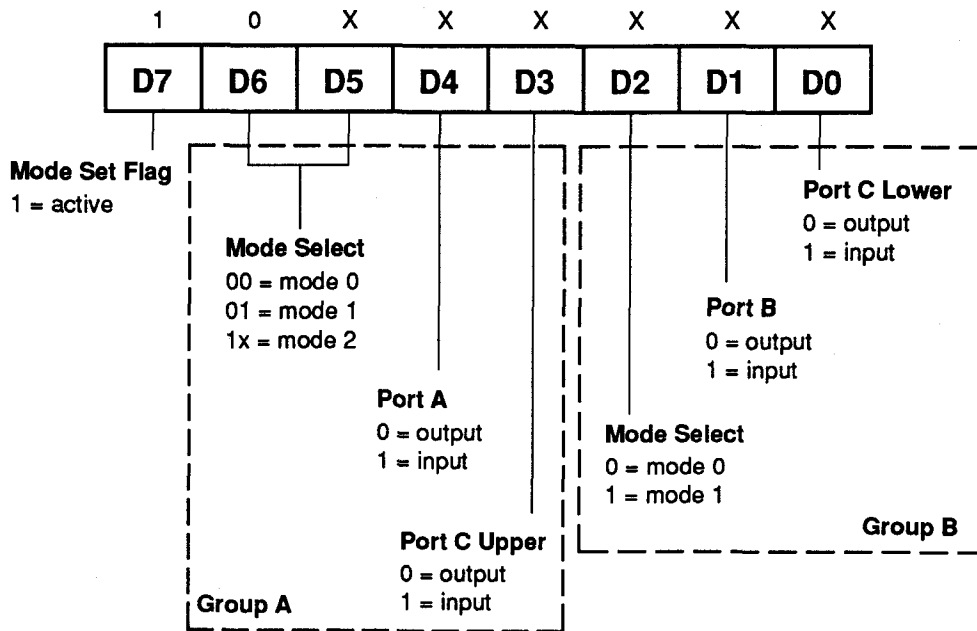
Transfers the 8-bit Port B digital input and digital output data between the board and an external device. A read transfers data from the external device, through P2, and into PPI Port B; a write transfers the written data from Port B through P2 to an external device.

BA + 2: PPI Port C — Digital I/O (Read/Write)

Transfers the two 4-bit Port C digital input and digital output data groups (Port C Upper and Port C Lower) between the board and an external device. A read transfers data from the external device, through P2, and into PPI Port C; a write transfers the written data from Port C through P2 to an external device.

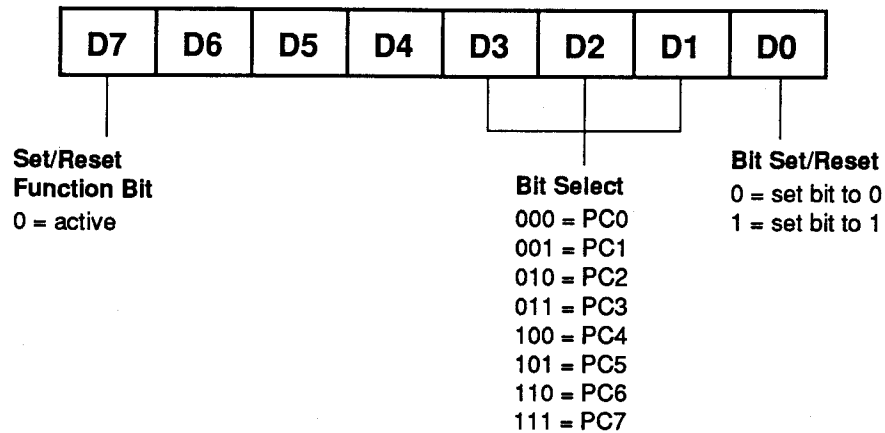
BA + 3: 8255 PPI Control Word (Write Only)

When bit 7 of this word is set to 1, a write programs the PPI configuration. Bit 6 must always be set to 0 (Mode 2 operation is not supported by the DIO24). The table below shows the control words for the 16 possible Mode 0 Port I/O combinations.

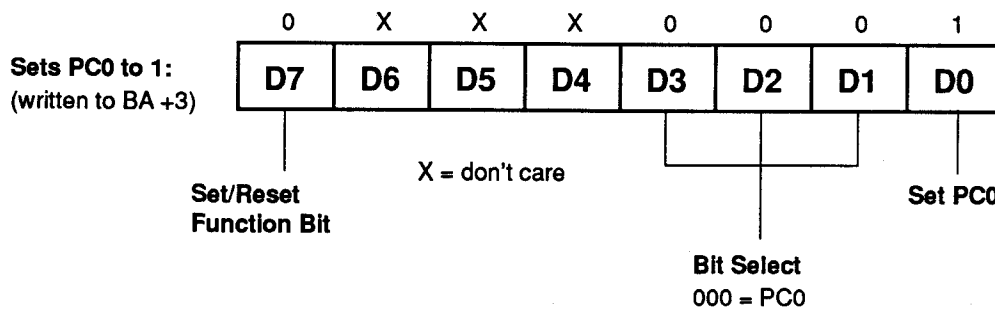


| 8255 Port I/O Flow Direction and Control Words, Mode 0 | | | | | | |
|--|--------------|---------|--------------|-----------------|---------|-----|
| Group A | | Group B | | Control Word | | |
| Port A | Port C Upper | Port B | Port C Lower | Binary | Decimal | Hex |
| Output | Output | Output | Output | 1 0 0 0 0 0 0 | 128 | 80 |
| Output | Output | Output | Input | 1 0 0 0 0 0 1 | 129 | 81 |
| Output | Output | Input | Output | 1 0 0 0 0 1 0 | 130 | 82 |
| Output | Output | Input | Input | 1 0 0 0 0 1 1 | 131 | 83 |
| Output | Input | Output | Output | 1 0 0 0 1 0 0 | 136 | 88 |
| Output | Input | Output | Input | 1 0 0 0 1 0 1 | 137 | 89 |
| Output | Input | Input | Output | 1 0 0 0 1 0 1 0 | 138 | 8A |
| Output | Input | Input | Input | 1 0 0 0 1 0 1 1 | 139 | 8B |
| Input | Output | Output | Output | 1 0 0 1 0 0 0 | 144 | 90 |
| Input | Output | Output | Input | 1 0 0 1 0 0 0 1 | 145 | 91 |
| Input | Output | Input | Output | 1 0 0 1 0 0 1 0 | 146 | 92 |
| Input | Output | Input | Input | 1 0 0 1 0 0 1 1 | 147 | 93 |
| Input | Input | Output | Output | 1 0 0 1 1 0 0 0 | 152 | 98 |
| Input | Input | Output | Input | 1 0 0 1 1 0 0 1 | 153 | 99 |
| Input | Input | Input | Output | 1 0 0 1 1 0 1 0 | 154 | 9A |
| Input | Input | Input | Input | 1 0 0 1 1 0 1 1 | 155 | 9B |

When bit 7 of this word is set to 0, a write can be used to individually program the Port C lines.

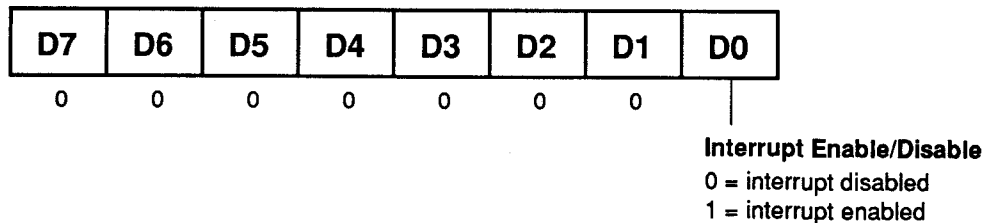


For example, if you want to set Port C bit 0 to 1, you would set up the control word so that bit 7 is 0; bits 1, 2, and 3 are 0 (this selects PC0); and bit 0 is 1 (this sets PC0 to 1). The control word is set up like this:



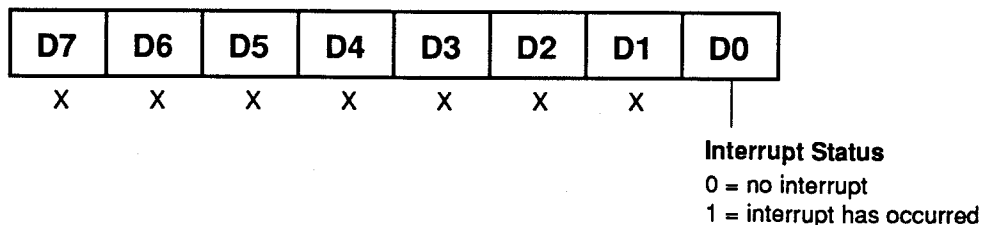
BA + 4: IRQ Enable (Write Only)

Enables and disables interrupt generation. Writing a "1" enables interrupt generation; writing a "0" disables interrupt generation.



BA + 5: Interrupt Status/Clear (Read/Write)

A read shows the status of the interrupt (bit 0 only) as defined below. A write clears the interrupt (data written is irrelevant). Each time the interrupt status bit goes high, a write should follow to clear the bit.



BA + 6: Reserved

BA + 7: Reserved

BA + 8: 8254 Timer/Counter 0 (Read/Write)

A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 9: 8254 Timer/Counter 1 (Read/Write)

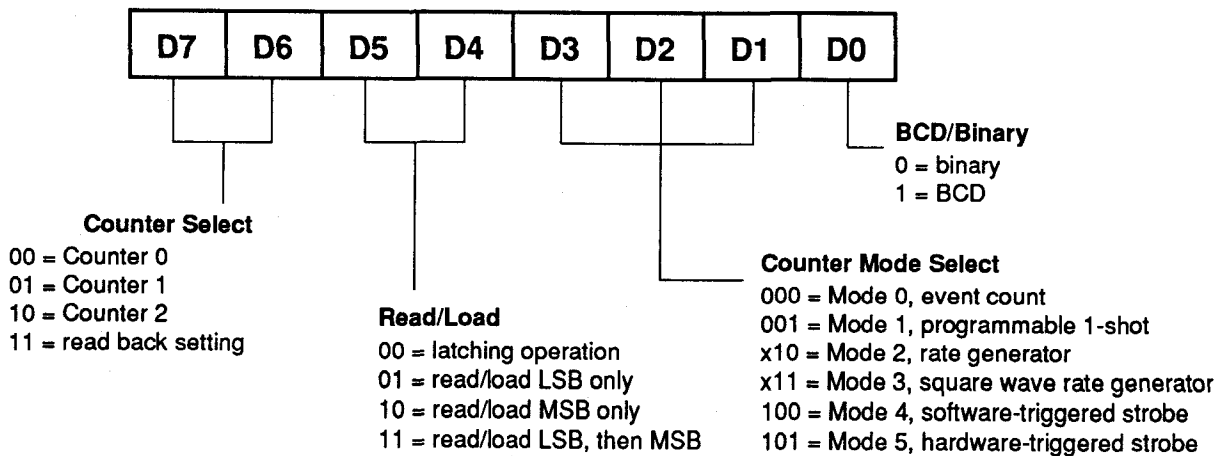
A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 10: 8254 Timer/Counter 2 (Read/Write)

A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 11: 8254 Control Word (Write Only)

Accesses the 8254 control register to directly control the three timer/counters.



Programming the DIO24

This section gives you some general information about programming and the DIO24 board, and then walks you through the major DIO24 programming functions. These descriptions will help you as you use the example programs included with the board. All of the program descriptions in this section use decimal values unless otherwise specified.

The DIO24 is programmed by writing to and reading from the correct I/O port locations on the board. These I/O ports were defined in the previous section. Most high-level languages such as BASIC, Pascal, C, and C++, and of course assembly language, make it very easy to read/write these ports. The table below shows you how to read from and write to I/O ports using some popular programming languages.

| Language | Read | Write |
|--------------|------------------------------|---|
| BASIC | Data = INP(Address) | OUT Address, Data |
| Turbo C | Data = inportb(Address) | outportb(Address, Data) |
| Turbo Pascal | Data := Port[Address] | Port[Address] := Data |
| Assembly | mov dx, Address in al, dx | mov dx, Address mov al, Data out dx, al |

In addition to being able to read/write the I/O ports on the DIO24, you must be able to perform a variety of operations that you might not normally use in your programming. The table below shows you some of the operators discussed in this section, with an example of how each is used with Pascal, C, and BASIC. Note that the modulus operator is used to retrieve the least significant byte (LSB) of a two-byte word, and the integer division operator is used to retrieve the most significant byte (MSB).

| Language | Modulus | Integer Division | AND | OR |
|----------|---------------------|----------------------------|---------------------|-------------------|
| C | % a = b % c | / a = b / c | & a = b & c | a = b c |
| Pascal | MOD a := b MOD c | DIV a := b DIV c | AND a := b AND c | OR a := b OR c |
| BASIC | MOD a = b MOD c | \ (backslash) a = b \ c | AND a = b AND c | OR a = b OR c |

Many compilers have functions that can read/write either 8 or 16 bits from/to an I/O port. For example, Turbo Pascal uses **Port** for 8-bit port operations and **PortW** for 16 bits, Turbo C uses **inportb** for an 8-bit read of a port and **inport** for a 16-bit read. **Be sure to use only 8-bit operations with the DIO24!**

Clearing and Setting Bits in a Port

When you clear or set one or more bits in a port, you must be careful that you do not change the status of the other bits. You can preserve the status of all bits you do not wish to change by proper use of the AND and OR binary operators. Using AND and OR, single or multiple bits can be easily cleared in one operation.

To clear a single bit in a port, AND the current value of the port with the value b, where $b = 255 - 2^{\text{bit}}$.

Example: Clear bit 5 in a port. Read in the current value of the port, AND it with 223 ($223 = 255 - 2^5$), and then write the resulting value to the port. In BASIC, this is programmed as:

```
V = INP (PortAddress)
V = V AND 223
OUT PortAddress, V
```

To set a single bit in a port, OR the current value of the port with the value b, where $b = 2^{\text{bit}}$.

Example: Set bit 3 in a port. Read in the current value of the port, OR it with 8 ($8 = 2^3$), and then write the resulting value to the port. In Pascal, this is programmed as:

```
V := Port [PortAddress];
V := V OR 8;
Port [PortAddress] := V;
```

Setting or clearing more than one bit at a time is accomplished just as easily. To clear multiple bits in a port, AND the current value of the port with the value b, where $b = 255 -$ (the sum of the values of the bits to be cleared). Note that the bits do not have to be consecutive.

Example: Clear bits 2, 4, and 6 in a port. Read in the current value of the port, AND it with 171 ($171 = 255 - 2^2 - 2^4 - 2^6$), and then write the resulting value to the port. In C, this is programmed as:

```
v = inportb(port_address);
v = v & 171;
outportb(port_address, v);
```

To set multiple bits in a port, OR the current value of the port with the value b, where b = the sum of the individual bits to be set. Note that the bits to be set do not have to be consecutive.

Example: Set bits 3, 5, and 7 in a port. Read in the current value of the port, OR it with 168 ($168 = 2^3 + 2^5 + 2^7$), and then write the resulting value back to the port. In assembly language, this is programmed as:

```
mov dx, PortAddress
in al, dx
or al, 168
out dx, al
```

Often, assigning a range of bits is a mixture of setting and clearing operations. You can set or clear each bit individually or use a faster method of first clearing all the bits in the range then setting only those bits that must be set using the method shown above for setting multiple bits in a port. The following example shows how this two-step operation is done.

Example: Assign bits 3, 4, and 5 in a port to 101 (bits 3 and 5 set, bit 4 cleared). First, read in the port and clear bits 3, 4, and 5 by ANDing them with 199. Then set bits 3 and 5 by ORing them with 40, and finally write the resulting value back to the port. In C, this is programmed as:

```

v = inportb(port_address);
v = v & 199;
v = v | 40;
outportb(port_address, v);

```

A final note: Don't be intimidated by the binary operators AND and OR and try to use operators for which you have a better intuition. For instance, if you are tempted to use addition and subtraction to set and clear bits in place of the methods shown above, DON'T! Addition and subtraction may seem logical, but they **will not work** if you try to clear a bit that is already clear or set a bit that is already set. For example, you might think that to set bit 5 of a port, you simply need to read in the port, add 32 (2^5) to that value, and then write the resulting value back to the port. This works fine if bit 5 is not already set. But, what happens when bit 5 *is* already set? Bits 0 to 4 will be unaffected and we can't say for sure what happens to bits 6 and 7, but we can say for sure that bit 5 ends up cleared instead of being set. A similar problem happens when you use subtraction to clear a bit in place of the method shown above.

Now that you know how to clear and set bits, we are ready to look at the programming steps for the DIO24 board functions.

Initializing the 8255 PPI

Before you can operate the DIO24, the 8255 must be initialized. This step must be executed every time you start up, reset, or reboot your computer.

The 8255 is initialized by writing the appropriate control word to I/O port BA + 3. The contents of your control word will vary, depending on how you want to configure your I/O lines. Use the control word description in the previous I/O map section to help you program the right value. Remember that the DIO24 cannot use Mode 2. In the example below, a decimal value of 128 sets up the 8255 so that all I/O lines are Mode 0 outputs.

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

Digital I/O Operations

Once the 8255 is initialized, you can use the digital I/O lines to control or monitor external devices.

Timer/Counters

An 8254 programmable interval timer provides three 16-bit, 8-MHz timer/counters for timing and counting functions such as frequency measurement, event counting, and interrupts. All three timer/counters are cascaded at the factory. Figure 4-1 shows the timer/counter circuitry.

Each timer/counter has two inputs, CLK in and GATE in, and one output, timer/counter OUT. They can be programmed as binary or BCD down counters by writing the appropriate data to the command word, as described in the I/O map section at the beginning of this chapter.

One of two clock sources, the on-board 8-MHz crystal or an external clock routed through on-board I/O connector P6 can be selected as the clock input to each timer/counter. In addition, the timer/counters can be cascaded by connecting TC0's output to TC1's clock input and TC1's output to TC2's clock input. The diagram shows how these clock sources are connected to the timer/counters.

An external gate source can be connected to each timer/counter through P6. When a gate is disconnected, an on-board pull-up resistor automatically pulls the gate high, enabling the timer/counter.

The output from each timer/counter is available at P6, where it can be used for interrupt generation or for counting functions.

The timer/counters can be programmed to operate in one of six modes, depending on your application. The following paragraphs briefly describe each mode.

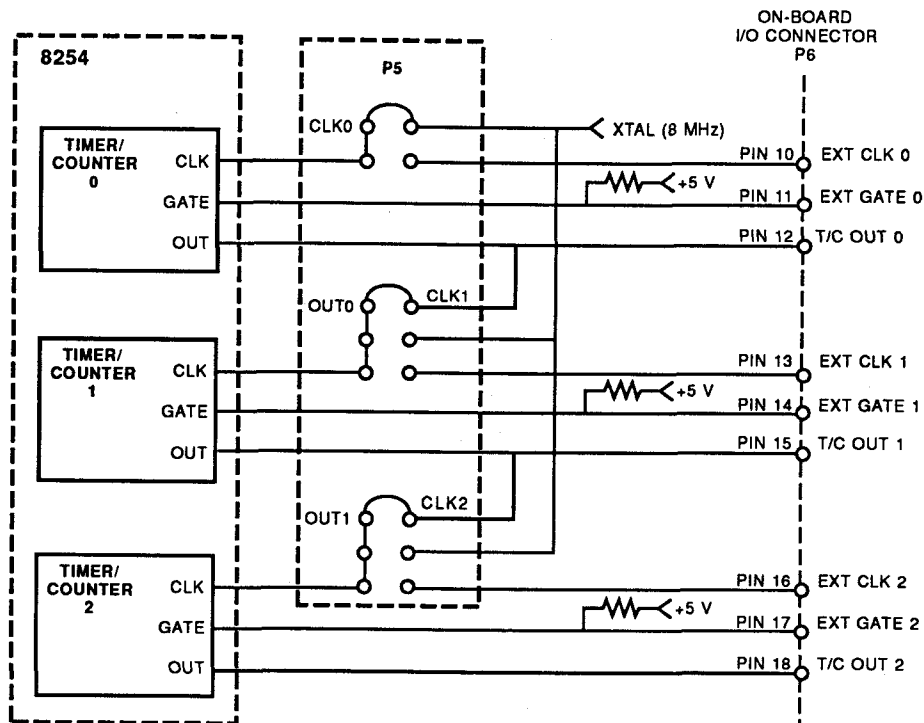


Fig. 4-1 — 8254 Timer/Counter Circuit Block Diagram

Mode 0, Event Counter (Interrupt on Terminal Count). This mode is typically used for event counting. While the timer/counter counts down, the output is low, and when the count is complete, it goes high. The output stays high until a new Mode 0 control word is written to the timer/counter.

Mode 1, Hardware-Retriggerable One-Shot. The output is initially high and goes low on the clock pulse following a trigger to begin the one-shot pulse. The output remains low until the count reaches 0, and then goes high and remains high until the clock pulse after the next trigger.

Mode 2, Rate Generator. This mode functions like a divide-by-N counter and is typically used to generate a real-time clock interrupt. The output is initially high, and when the count decrements to 1, the output goes low for one clock pulse. The output then goes high again, the timer/counter reloads the initial count, and the process is repeated. This sequence continues indefinitely.

Mode 3, Square Wave Mode. Similar to Mode 2 except for the duty cycle output, this mode is typically used for baud rate generation. The output is initially high, and when the count decrements to one-half its initial count, the output goes low for the remainder of the count. The timer/counter reloads and the output goes high again. This process repeats indefinitely.

Mode 4, Software-Triggered Strobe. The output is initially high. When the initial count expires, the output goes low for one clock pulse and then goes high again. Counting is "triggered" by writing the initial count.

Mode 5, Hardware Triggered Strobe (Retriggerable). The output is initially high. Counting is triggered by the rising edge of the gate input. When the initial count has expired, the output goes low for one clock pulse and then goes high again.

Interrupts

- What Is an Interrupt?

An interrupt is an event that causes the processor in your computer to temporarily halt its current process and execute another routine. Upon completion of the new routine, control is returned to the original routine at the point where its execution was interrupted.

Interrupts are very handy for dealing with asynchronous events (events that occur at less than regular intervals). Keyboard activity is a good example; your computer cannot predict when you might press a key and it would be a waste of processor time for it to do nothing while waiting for a keystroke to occur. Thus, the interrupt scheme is used and the processor proceeds with other tasks. Then, when a keystroke does occur, the keyboard 'interrupts' the processor, and the processor gets the keyboard data, places it in memory, and then returns to what it was doing before it was interrupted. Other common devices that use interrupts are modems, disk drives, and mice.

Your DIO24 board can interrupt the processor when one of the six interrupt sources is enabled. By using these interrupts, you can write software that effectively deals with real world events.

- Interrupt Request Lines

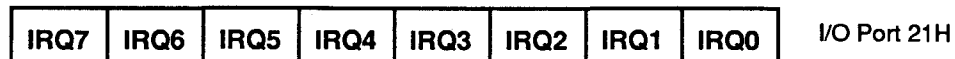
To allow different peripheral devices to generate interrupts on the same computer, the PC bus has eight different interrupt request (IRQ) lines. A transition from low to high on one of these lines generates an interrupt request which is handled by the PC's interrupt controller. The interrupt controller checks to see if interrupts are to be acknowledged from that IRQ and, if another interrupt is already in progress, it decides if the new request should supersede the one in progress or if it has to wait until the one in progress is done. This prioritizing allows an interrupt to be interrupted if the second request has a higher priority. The priority level is based on the number of the IRQ; IRQ0 has the highest priority, IRQ1 is second-highest, and so on through IRQ7, which has the lowest. Many of the IRQs are used by the standard system resources. IRQ0 is used by the system timer, IRQ1 is used by the keyboard, IRQ3 by COM2, IRQ4 by COM1, and IRQ6 by the disk drives. Therefore, it is important for you to know which IRQ lines are available in your system for use by the DIO24 board.

- 8259 Programmable Interrupt Controller

The chip responsible for handling interrupt requests in the PC is the 8259 Programmable Interrupt Controller. To use interrupts, you will need to know how to read and set the 8259's interrupt mask register (IMR) and how to send the end-of-interrupt (EOI) command to the 8259.

- Interrupt Mask Register (IMR)

Each bit in the interrupt mask register (IMR) contains the mask status of an IRQ line; bit 0 is for IRQ0, bit 1 is for IRQ1, and so on. If a bit is set (equal to 1), then the corresponding IRQ is masked and it will not generate an interrupt. If a bit is clear (equal to 0), then the corresponding IRQ is unmasked and can generate interrupts. The IMR is programmed through port 21H.



For all bits:

0 = IRQ unmasked (enabled)

1 = IRQ masked (disabled)

- End-of-Interrupt (EOI) Command

After an interrupt service routine is complete, the 8259 interrupt controller must be notified. This is done by writing the value 20H to I/O port 20H.

- What Exactly Happens When an Interrupt Occurs?

Understanding the sequence of events when an interrupt is triggered is necessary to properly write software interrupt handlers. When an interrupt request line is driven high by a peripheral device (such as the DIO24), the

interrupt controller checks to see if interrupts are enabled for that IRQ, and then checks to see if other interrupts are active or requested and determines which interrupt has priority. The interrupt controller then interrupts the processor. The current code segment (CS), instruction pointer (IP), and flags are pushed on the stack for storage, and a new CS and IP are loaded from a table that exists in the lowest 1024 bytes of memory. This table is referred to as the interrupt vector table and each entry is called an interrupt vector. Once the new CS and IP are loaded from the interrupt vector table, the processor begins executing the code located at CS:IP. When the interrupt routine is completed, the CS, IP, and flags that were pushed on the stack when the interrupt occurred are now popped from the stack and execution resumes from the point where it was interrupted.

- Using Interrupts in Your Programs

Adding interrupts to your software is not as difficult as it may seem, and what they add in terms of performance is often worth the effort. Note, however, that although it is not that hard to use interrupts, the smallest mistake will often lead to a system hang that requires a reboot. This can be both frustrating and time-consuming. But, after a few tries, you'll get the bugs worked out and enjoy the benefits of properly executed interrupts.

- Writing an Interrupt Service Routine (ISR)

The first step in adding interrupts to your software is to write the interrupt service routine (ISR). This is the routine that will automatically be executed each time an interrupt request occurs on the specified IRQ. An ISR is different than standard routines that you write. First, on entrance, the processor registers should be pushed onto the stack **BEFORE** you do anything else. Second, just before exiting your ISR, you must clear the interrupt status of the DIO24 and write an end-of-interrupt command to the 8259 controller. Finally, when exiting the ISR, in addition to popping all the registers you pushed on entrance, you must use the IRET instruction and **not** a plain RET. The IRET automatically pops the flags, CS, and IP that were pushed when the interrupt was called.

If you find yourself intimidated by interrupt programming, take heart. Most Pascal and C compilers allow you to identify a procedure (function) as an interrupt type and will automatically add these instructions to your ISR, with one important exception: most compilers **do not** automatically add the end-of-interrupt command to the procedure; you must do this yourself. Other than this and the few exceptions discussed below, you can write your ISR just like any other routine. It can call other functions and procedures in your program and it can access global data. If you are writing your first ISR, we recommend that you stick to the basics; just something that will convince you that it works, such as incrementing a global variable.

NOTE: If you are writing an ISR using assembly language, you are responsible for pushing and popping registers and using IRET instead of RET.

There are a few cautions you must consider when writing your ISR. The most important is, **do not use any DOS functions or routines that call DOS functions from within an ISR.** DOS is not reentrant; that is, a DOS function cannot call itself. In typical programming, this will not happen because of the way DOS is written. But what about when using interrupts? Then, you could have a situation such as this in your program. If DOS function X is being executed when an interrupt occurs and the interrupt routine makes a call to DOS function X, then function X is essentially being called while it is already active. Such a reentrancy attempt spells disaster because DOS functions are not written to support it. This is a complex concept and you do not need to understand it. Just make sure that you do not call any DOS functions from within your ISR. The one wrinkle is that, unfortunately, it is not obvious which library routines included with your compiler use DOS functions. A rule of thumb is that routines which write to the screen, or check the status of or read the keyboard, and any disk I/O routines use DOS and should be avoided in your ISR.

The same problem of reentrancy exists for many floating point emulators as well, meaning you may have to avoid floating point (real) math in your ISR.

Note that the problem of reentrancy exists, no matter what programming language you are using. Even if you are writing your ISR in assembly language, DOS and many floating point emulators are not reentrant. Of course, there are ways around this problem, such as those which involve checking to see if any DOS functions are currently active when your ISR is called, but such solutions are well beyond the scope of this discussion.

The second major concern when writing your ISR is to make it as short as possible in terms of execution time. Spending long periods of time in your ISR may mean that other important interrupts are being ignored. Also, if you

spend too long in your ISR, it may be called again before you have completed handling the first run. This often leads to a hang that requires a reboot.

Your ISR should have this structure:

- Push any processor registers used in your ISR. Most C and Pascal interrupt routines automatically do this for you.
- Put the body of your routine here.
- Clear the interrupt bit on the DIO24 by writing any value to BA + 5.
- Issue the EOI command to the 8259 interrupt controller by writing 20H to port 20H.
- Pop all registers pushed on entrance. Most C and Pascal interrupt routines automatically do this for you.

The following C and Pascal examples show what the shell of your ISR should be like:

In C:

```
void interrupt ISR(void)
{
    /* Your code goes here. Do not use any DOS functions! */
    outportb(BaseAddress + 5, 0);    /* Clear DIO24 interrupt */
    outportb(0x20, 0x20);           /* Send EOI command to 8259 */
}
```

In Pascal:

```
Procedure ISR; Interrupt;
begin
    { Your code goes here. Do not use any DOS functions! }
    Port[BaseAddress + 5] := 0;      { Clear DIO24 interrupt }
    Port[$20] := $20;               { Send EOI command to 8259 }
end;
```

- Saving the Startup Interrupt Mask Register (IMR) and Interrupt Vector

The next step after writing the ISR is to save the startup state of the interrupt mask register and the interrupt vector that you will be using. The IMR is located at I/O port 21H. The interrupt vector you will be using is located in the interrupt vector table which is simply an array of 256-bit (4-byte) pointers and is located in the first 1024 bytes of memory (Segment = 0, Offset = 0). You can read this value directly, but it is a better practice to use DOS function 35H (get interrupt vector). Most C and Pascal compilers provide a library routine for reading the value of a vector. The vectors for the hardware interrupts are vectors 8 through 15, where IRQ0 uses vector 8, IRQ1 uses vector 9, and so on. Thus, if the DIO24 will be using IRQ3, you should save the value of interrupt vector 11.

Before you install your ISR, temporarily mask out the IRQ you will be using. This prevents the IRQ from requesting an interrupt while you are installing and initializing your ISR. To mask the IRQ, read in the current IMR at I/O port 21H and set the bit that corresponds to your IRQ (remember, setting a bit disables interrupts on that IRQ while clearing a bit enables them). The IMR is arranged so that bit 0 is for IRQ0, bit 1 is for IRQ1, and so on. See the paragraph entitled *Interrupt Mask Register (IMR)* earlier in this chapter for help in determining your IRQ's bit. After setting the bit, write the new value to I/O port 21H.

With the startup IMR saved and the interrupts on your IRQ temporarily disabled, you can assign the interrupt vector to point to your ISR. Again, you can overwrite the appropriate entry in the vector table with a direct memory write, but this is a bad practice. Instead, use either DOS function 25H (set interrupt vector) or, if your compiler provides it, the library routine for setting an interrupt vector. Remember that vector 8 is for IRQ0, vector 9 is for IRQ1, and so on.

If you need to program the source of your interrupts, do that next. For example, if you are using the programmable interval timer to generate interrupts, you must program it to run in the proper mode and at the proper rate.

Finally, clear the bit in the IMR for the IRQ you are using. This enables interrupts on the IRQ.

- Restoring the Startup IMR and Interrupt Vector

Before exiting your program, you must restore the interrupt mask register and interrupt vectors to the state they were in when your program started. To restore the IMR, write the value that was saved when your program started to I/O port 21H. Restore the interrupt vector that was saved at startup with either DOS function 35H (get interrupt vector), or use the library routine supplied with your compiler. Performing these two steps will guarantee that the interrupt status of your computer is the same after running your program as it was before your program started running.

- Common Interrupt Mistakes

- Remember that hardware interrupts are numbered 8 through 15, even though the corresponding IRQs are numbered 0 through 7.
- Two of the most common mistakes when writing an ISR are forgetting to clear the interrupt status of the DIO24 and forgetting to issue the EOI command to the 8259 interrupt controller before exiting the ISR.

Example Programs

Included with the DIO24 is a set of example programs that demonstrate the use of many of the board's features. These examples are written in C, Pascal, Assembly, and BASIC. Also included is an easy-to-use menu-driven diagnostics program, 24DIAG, which is especially helpful when you are first checking out your board after installation.

Before using the software included with your board, make a backup copy of the disk. You may make as many backups as you need.

C and Pascal Programs

These programs are source code files so that you can easily develop your own custom software for your DIO24 board.

Digital I/O:

DIGITAL Simple program that shows how to read and write the digital I/O lines.

Timer/Counters:

TIMER A short program demonstrating how to program the 8254 for use as a timer.

BASIC Programs

These programs include both source code files and executable files so that you can run them on your DIO24. All of the executable programs are set up to look for the board at a base address (BA) of 300 hex (768 decimal). If you change the base address of the board, you must also change the BA in your programs.

Digital I/O:

DIGITAL Simple program that shows how to read and write the digital I/O lines.

Timer/Counters:

TIMER A short program demonstrating how to program the 8254 for use as a timer.

APPENDIX A

DIO24 SPECIFICATIONS

DIO24 Characteristics Typical @ 25° C

Interface

Switch-selectable base address, I/O mapped
Jumper-selectable interrupts

Digital I/O CMOS 82C55

Opto-22 compatible
Number of lines 24
Logic compatibility TTL/CMOS
(Configurable with optional I/O pull-up/pull-down resistors)
High-level output voltage 4.2V, min
Low-level output voltage 0.45V, max
High-level input voltage 2.2V, min; 5.5V, max
Low-level input voltage -0.3V, min; 0.8V, max
High-level output current, I_{source} CMOS buffer: -12 mA, max;
TTL buffer: -16 mA, max
Low-level output current, I_{sink} CMOS buffer: 24 mA, max;
TTL buffer: 64 mA, max
Input load current ±10 μA
Input capacitance 10 pF
Input capacitance,
C(IN)@F=1MHz 10 pF
Output capacitance,
C(OUT)@F=1MHz 20 pF

Timer/Counters CMOS 82C54

Three 16-bit down counters
6 programmable operating modes
Counter input source External clock (8 MHz, max) or
on-board 8 MHz clock
Counter outputs Available externally; used as PC interrupts
Counter gate source External gate or always enabled

Current Requirements

194 mA @ +5 volts

Connectors

P2 — 50-pin right angle shrouded box header
P6 — 20-pin box header

Size

Short slot — 3.875"H x 5.25"W (99mm x 134mm)

APPENDIX B

I/O CONNECTOR PIN ASSIGNMENTS

I/O Connector P2:

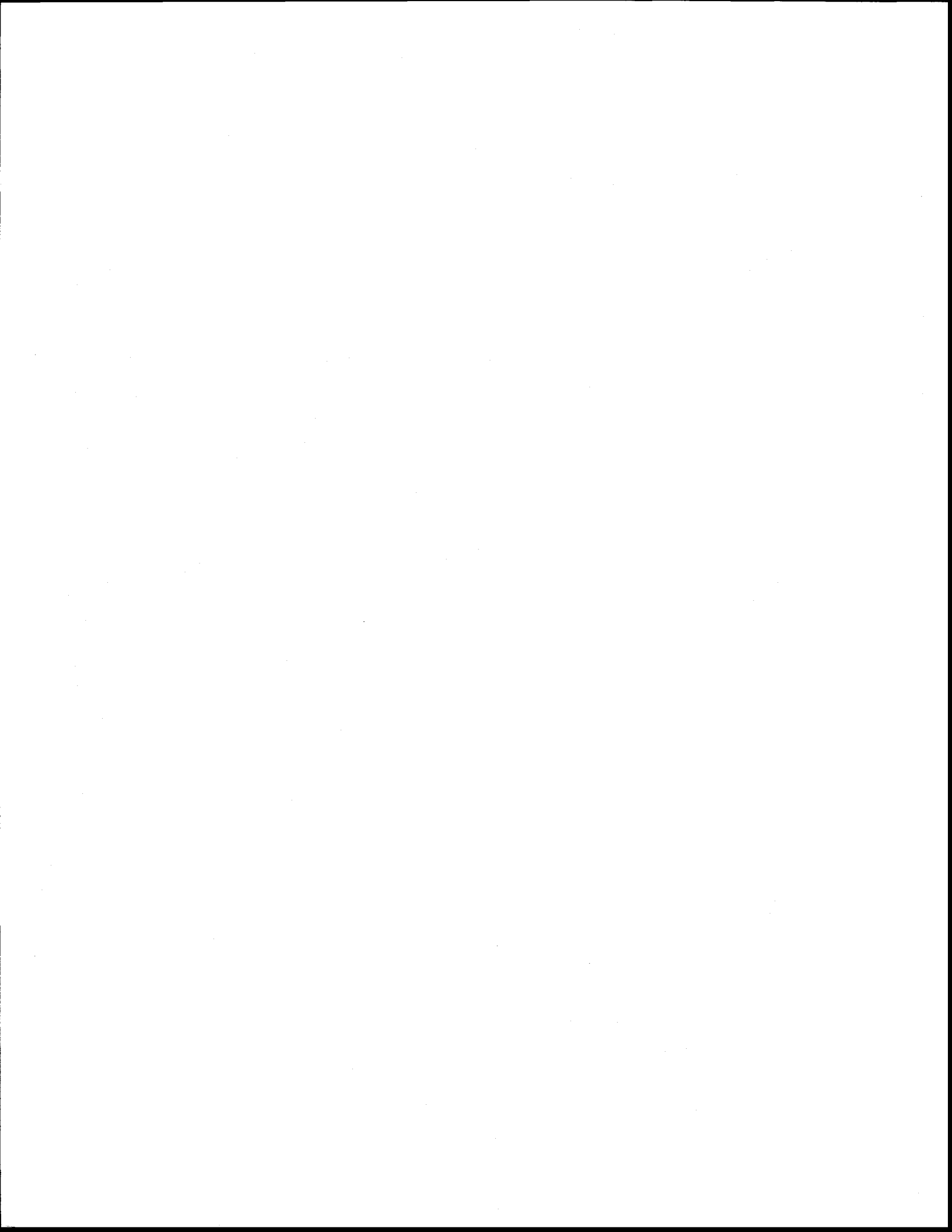
| | | | |
|----------|---|---|-------------|
| PC0 | ① | ② | EXTINT |
| PC1 | ③ | ④ | DIGITAL GND |
| PC2 | ⑤ | ⑥ | DIGITAL GND |
| PC3 | ⑦ | ⑧ | DIGITAL GND |
| PC4 | ⑨ | ⑩ | DIGITAL GND |
| PC5 | ⑪ | ⑫ | DIGITAL GND |
| PC6 | ⑬ | ⑭ | DIGITAL GND |
| PC7 | ⑮ | ⑯ | DIGITAL GND |
| PB0 | ⑰ | ⑱ | DIGITAL GND |
| PB1 | ⑲ | ⑳ | DIGITAL GND |
| PB2 | ㉑ | ㉒ | DIGITAL GND |
| PB3 | ㉓ | ㉔ | DIGITAL GND |
| PB4 | ㉕ | ㉖ | DIGITAL GND |
| PB5 | ㉗ | ㉘ | DIGITAL GND |
| PB6 | ㉙ | ㉚ | DIGITAL GND |
| PB7 | ㉛ | ㉜ | DIGITAL GND |
| PA0 | ㉝ | ㉞ | DIGITAL GND |
| PA1 | ㉟ | ㊱ | DIGITAL GND |
| PA2 | ㊳ | ㊴ | DIGITAL GND |
| PA3 | ㊵ | ㊶ | DIGITAL GND |
| PA4 | ㊷ | ㊸ | DIGITAL GND |
| PA5 | ㊹ | ㊺ | DIGITAL GND |
| PA6 | ㊻ | ㊼ | DIGITAL GND |
| PA7 | ㊽ | ㊾ | DIGITAL GND |
| +5 VOLTS | ㊿ | ① | DIGITAL GND |

On-board Connector P6:

| | | | |
|-------------|---|---|-------------|
| PC0 | ① | ② | PC1 |
| PC2 | ③ | ④ | PC3 |
| PC4 | ⑤ | ⑥ | PC5 |
| PC6 | ⑦ | ⑧ | PC7 |
| DIGITAL GND | ⑨ | ⑩ | EXT CLK 0 |
| EXT GATE 0 | ⑪ | ⑫ | T/C OUT 0 |
| EXT CLK 1 | ⑬ | ⑭ | EXT GATE 1 |
| T/C OUT 1 | ⑮ | ⑯ | EXT CLK 2 |
| EXT GATE 2 | ⑰ | ⑱ | T/C OUT 2 |
| DIGITAL GND | ⑲ | ⑳ | DIGITAL GND |

APPENDIX C

COMPONENT DATA SHEETS



**Intel 82C55A Programmable Peripheral Interface
Data Sheet Reprint**





82C55A CHMOS PROGRAMMABLE PERIPHERAL INTERFACE

- Compatible with all Intel and Most Other Microprocessors
- High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188
- 24 Programmable I/O Pins
- Low Power CHMOS
- Completely TTL Compatible
- Control Word Read-Back Capability
- Direct Bit Set/Reset Capability
- 2.5 mA DC Drive Capability on all I/O Port Outputs
- Available in 40-Pin DIP and 44-Pin PLCC
- Available in EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel 82C55A is a high-performance, CHMOS version of the industry standard 8255A general purpose programmable I/O device which is designed for use with all Intel and most other microprocessors. It provides 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The 82C55A is pin compatible with the NMOS 8255A and 8255A-5.

In MODE 0, each group of 12 I/O pins may be programmed in sets of 4 and 8 to be inputs or outputs. In MODE 1, each group may be programmed to have 8 lines of input or output. 3 of the remaining 4 pins are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

The 82C55A is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent NMOS product. The 82C55A is available in 40-pin DIP and 44-pin plastic leaded chip carrier (PLCC) packages.

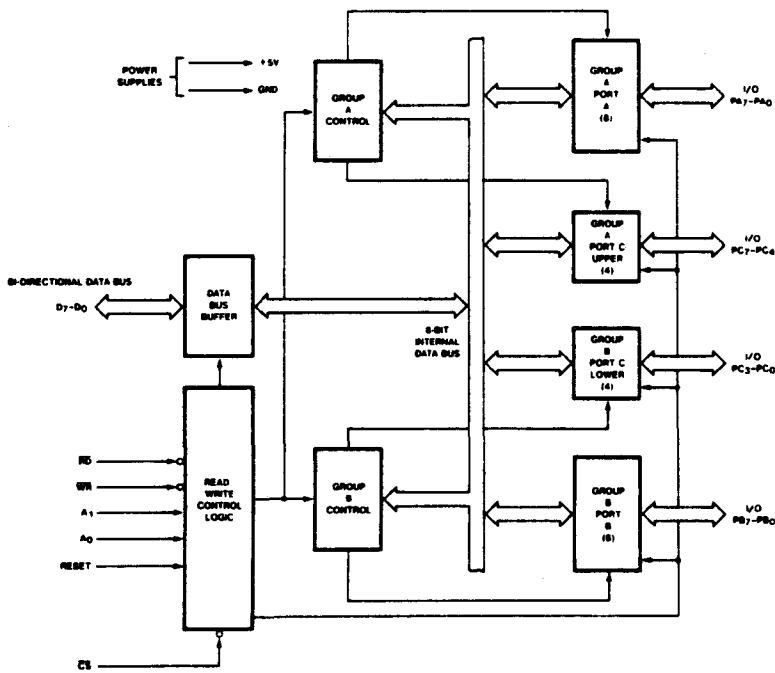
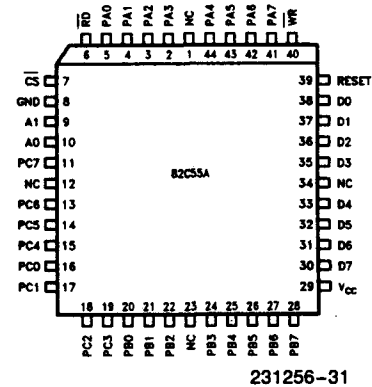
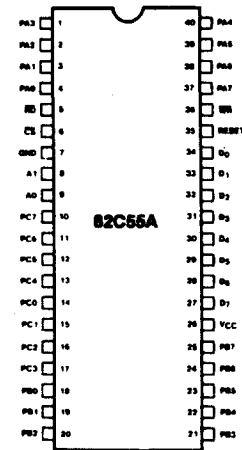


Figure 1. 82C55A Block Diagram

231256-1



231256-31



231256-2

Figure 2. 82C55A Pinout
Diagrams are for pin reference only. Package sizes are not to scale.

Table 1. Pin Description

| Symbol | Pin Number | | Type | Name and Function | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|------------|---------------|------|--|----------------------|-----------------|-----------------|-----------------|-------------------------|------------------------|---|---|---|---|---|-------------------|---|---|---|---|---|-------------------|---|---|---|---|---|-------------------|---|---|---|---|---|-------------------------|
| | Dip | PLCC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PA ₃₋₀ | 1-4 | 2-5 | I/O | PORT A, PINS 0-3: Lower nibble of an 8-bit data output latch/buffer and an 8-bit data input latch. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| \overline{RD} | 5 | 6 | I | READ CONTROL: This input is low during CPU read operations. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| \overline{CS} | 6 | 7 | I | CHIP SELECT: A low on this input enables the 82C55A to respond to \overline{RD} and \overline{WR} signals. \overline{RD} and \overline{WR} are ignored otherwise. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| GND | 7 | 8 | | System Ground | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A ₁₋₀ | 8-9 | 9-10 | I | ADDRESS: These input signals, in conjunction \overline{RD} and \overline{WR} , control the selection of one of the three ports or the control word registers. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | <table border="1"> <thead> <tr> <th>A₁</th> <th>A₀</th> <th>\overline{RD}</th> <th>\overline{WR}</th> <th>\overline{CS}</th> <th>Input Operation (Read)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>Port A - Data Bus</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>Port B - Data Bus</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>Port C - Data Bus</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>Control Word - Data Bus</td> </tr> </tbody> </table> | A ₁ | A ₀ | \overline{RD} | \overline{WR} | \overline{CS} | Input Operation (Read) | 0 | 0 | 0 | 1 | 0 | Port A - Data Bus | 0 | 1 | 0 | 1 | 0 | Port B - Data Bus | 1 | 0 | 0 | 1 | 0 | Port C - Data Bus | 1 | 1 | 0 | 1 | 0 | Control Word - Data Bus |
| | | | | A ₁ | A ₀ | \overline{RD} | \overline{WR} | \overline{CS} | Input Operation (Read) | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 0 | 0 | 0 | 1 | 0 | Port A - Data Bus | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 0 | 1 | 0 | 1 | 0 | Port B - Data Bus | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 1 | 0 | 0 | 1 | 0 | Port C - Data Bus | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | Control Word - Data Bus | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | Output Operation (Write) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 0 | 0 | 1 | 0 | 0 | Data Bus - Port A | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 0 | 1 | 1 | 0 | 0 | Data Bus - Port B | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 1 | 0 | 1 | 0 | 0 | Data Bus - Port C | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | 1 | 1 | 1 | 0 | 0 | Data Bus - Control | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | Disable Function | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| X | X | X | X | 1 | Data Bus - 3 - State | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| X | X | 1 | 1 | 0 | Data Bus - 3 - State | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PC ₇₋₄ | 10-13 | 11,13-15 | I/O | PORT C, PINS 4-7: Upper nibble of an 8-bit data output latch/buffer and an 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PC ₀₋₃ | 14-17 | 16-19 | I/O | PORT C, PINS 0-3: Lower nibble of Port C. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PB ₀₋₇ | 18-25 | 20-22, 24-28 | I/O | PORT B, PINS 0-7: An 8-bit data output latch/buffer and an 8-bit data input buffer. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| V _{CC} | 26 | 29 | | SYSTEM POWER: + 5V Power Supply. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| D ₇₋₀ | 27-34 | 30-33, 35-38 | I/O | DATA BUS: Bi-directional, tri-state data bus lines, connected to system data bus. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RESET | 35 | 39 | I | RESET: A high on this input clears the control register and all ports are set to the input mode. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| \overline{WR} | 36 | 40 | I | WRITE CONTROL: This input is low during CPU write operations. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PA ₇₋₄ | 37-40 | 41-44 | I/O | PORT A, PINS 4-7: Upper nibble of an 8-bit data output latch/buffer and an 8-bit data input latch. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| NC | | 1, 12, 23, 34 | | No Connect | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

82C55A FUNCTIONAL DESCRIPTION

General

The 82C55A is a programmable peripheral interface device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 82C55A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7-C4)
Control Group B - Port B and Port C lower (C3-C0)

The control word register can be both written and read as shown in the address decode table in the pin descriptions. Figure 6 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

Ports A, B, and C

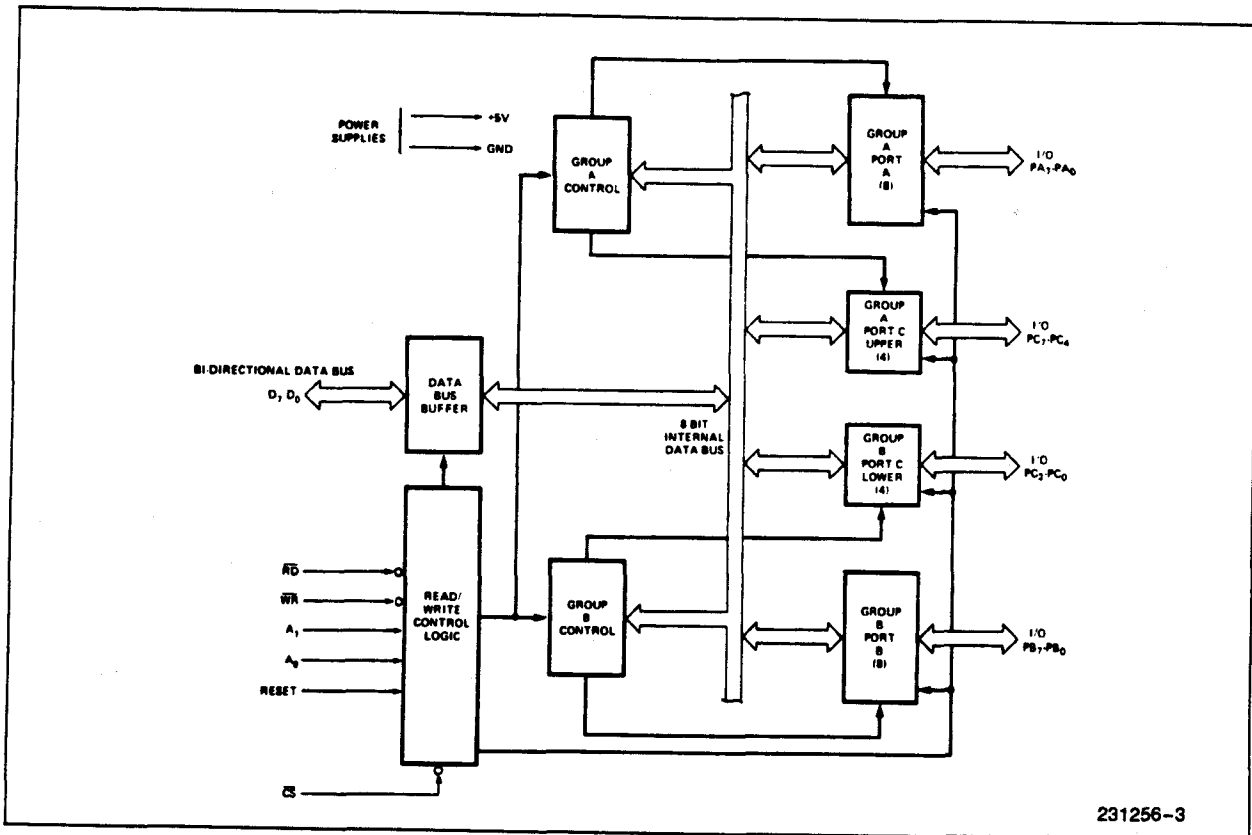
The 82C55A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

Port A. One 8-bit data output latch/buffer and one 8-bit input latch buffer. Both "pull-up" and "pull-down" bus hold devices are present on Port A.

Port B. One 8-bit data input/output latch/buffer. Only "pull-up" bus hold devices are present on Port B.

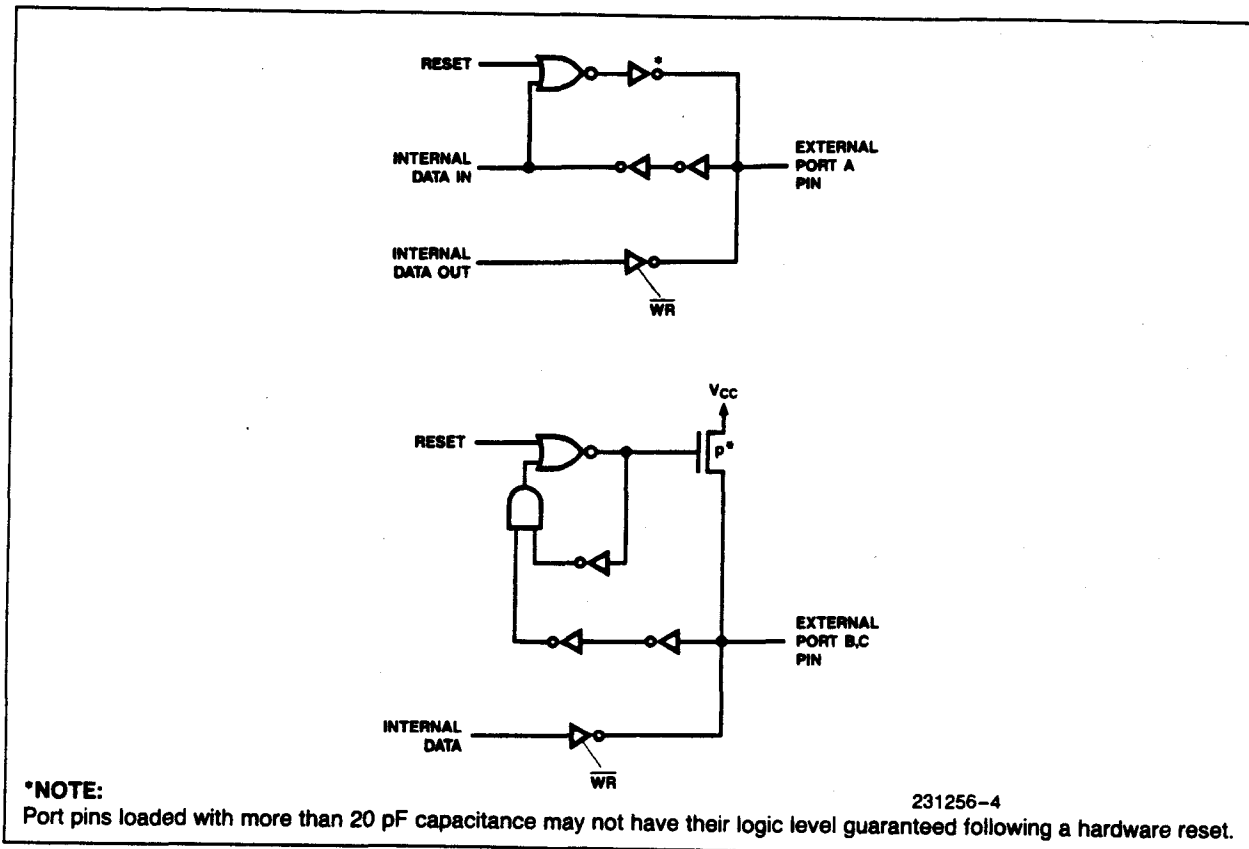
Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B. Only "pull-up" bus hold devices are present on Port C.

See Figure 4 for the bus-hold circuit configuration for Port A, B, and C.



231256-3

Figure 3. 82C55A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions



231256-4

Figure 4. Port A, B, C, Bus-hold Configuration

82C55A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

- Mode 0 — Basic input/output
- Mode 1 — Strobed Input/output
- Mode 2 — Bi-directional Bus

When the reset input goes "high" all ports will be set to the input mode with all 24 port lines held at a logic "one" level by the internal bus hold devices (see Figure 4 Note). After the reset is removed the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need for pullup or pulldown devices in "all CMOS" designs. During the execution of the system program, any of the other modes may be selected by using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

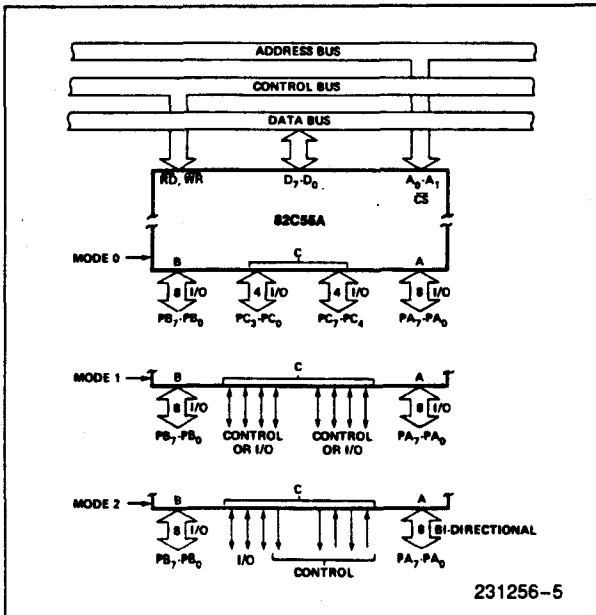


Figure 5. Basic Mode Definitions and Bus Interface

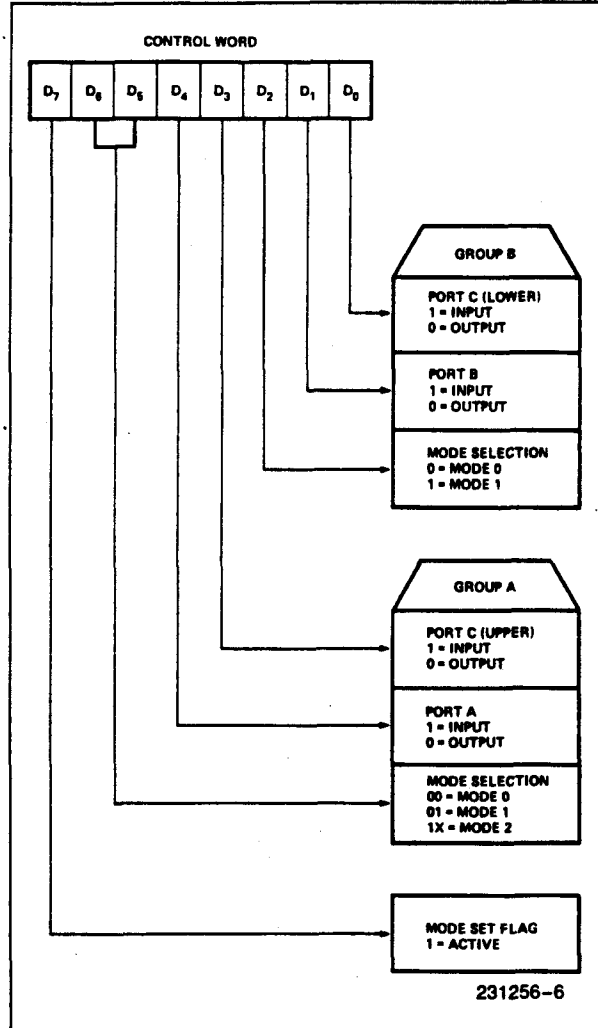


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTPUT instruction. This feature reduces software requirements in Control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

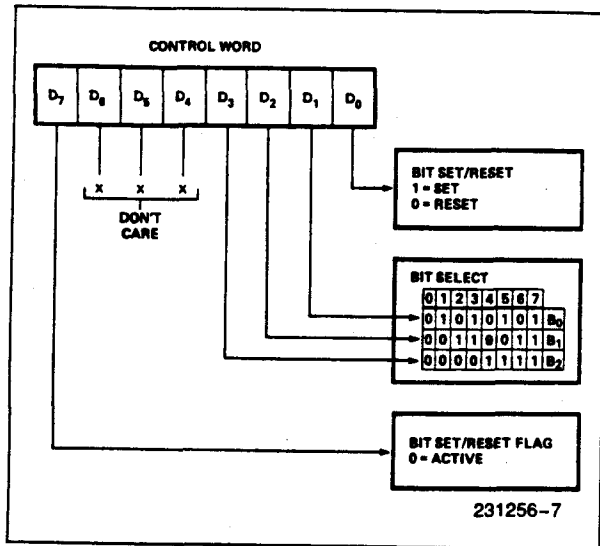


Figure 7. Bit Set/Reset Format

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

- (BIT-SET)—INTE is SET—Interrupt enable
- (BIT-RESET)—INTE is RESET—Interrupt disable

Note:

All Mask flip-flops are automatically reset during mode selection and device Reset.

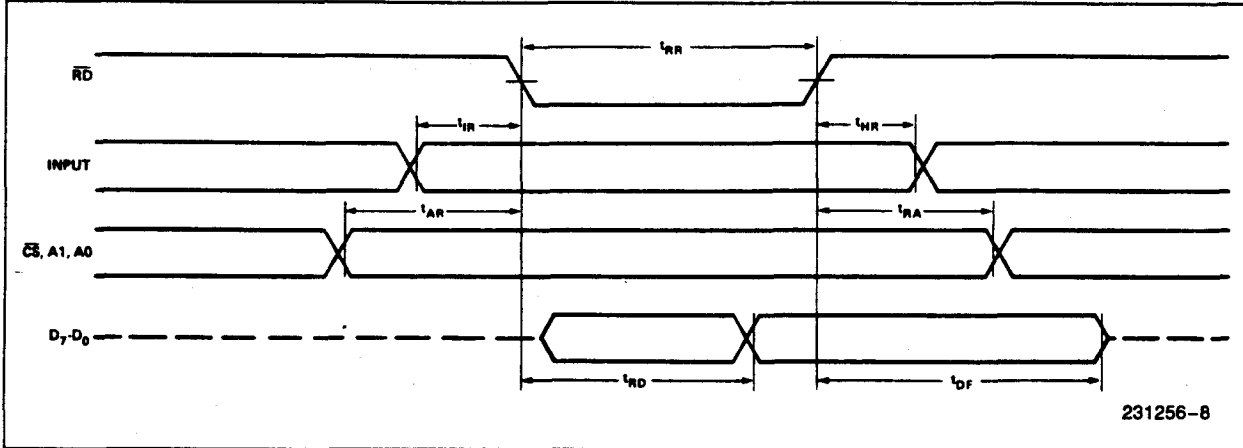
Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



MODE 0 (BASIC OUTPUT)

