ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Keithley PDMA-16* at our website: *Click HERE*

User Guide

for the

Keithley MetraByte

PDMA-16

Parallel Digital

Interface

Board

Revision B - January 1992
Copyright © Keithley Instruments, Inc. 1986
Part Number: 24885

Keithley Instruments, Inc. Data Acquisition Division

440 Myles Standish Blvd., Taunton, MA 02780
TEL. 508/880-3000, FAX 508/880-0179

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contents

5.12 MODE 10: Deallocate Memory Segment 5-14
5.13 MODE 11: Move Data From Source To Destination 5-14
5.14 MODE 12: Disable DMA 5-15

CHAPTER 6: PROGRAMMABLE INTERVAL TIMER

6.1 The 8254 Programmable Interval Timer 6-1
6.2 Reading & Loading The Counters 6-3

CHAPTER 7: APPLICATIONS

7.1 Typical Handshake Connection 7-1
7.2 Waveform Generation With a D/A Converter 7-2
7.3 High Speed A/D Conversion 7-3
7.4 Combined A/D & D/A Conversion Using Directional Controls 7-4
7.5 Commonly Encountered Problems 7-6

CHAPTER 8: MAINTENANCE & REPAIR

8.1 Service & Repair 8-1
8.2 Performing Your Own Repairs 8-1

APPENDICES

Appendix A Specifications
Appendix B Summary Of Error Codes
Appendix C Understanding DMA
Appendix D Modes 9 & 10: Allocate/Deallocate DMA Buffers
Appendix E Storage Of Integer Variables
Appendix F PDMA-16 PCF

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

The PDMA-16 high-speed, 16-bit, parallel-digital interface card with DMA (Direct Memory Access) plugs directly into an expansion slot of the IBM PC/XT/AT and compatible machines, including the AT&T 6300. All digital I/O feeds through a 37-pin, D-type, male connector that projects through the rear mounting plate.

Compared with standard digital I/O boards (such as the Keithley MetraByte PIO-12), the PDMA-16 with its internal DMA and interrupt-control hardware is capable of much higher data transfer rates than can be achieved with programmed I/O transfers through the main processor (at best limited to around 5-10KHz). Actual DMA data transfer rates are computer dependent, influenced by the clock rate of the computer's main processor and the clock supplied to the 8237 DMA controller. Continuous transfer rates of 200,000 bytes/sec or 120,000 words/sec are possible on a standard IBM PC with 4.77MHz clock. Note that the DMA controller may address only one page (64Kbytes) of memory at a time. It is possible to transfer any number of bytes up to this limit as a single-shot operation, or to constantly output or input to a block of memory using the DMA Controller's Auto-initialize Mode.

1.2 APPLICATIONS

Typical applications for the PDMA-16 are as follows:

1. Interface for high-speed digital peripherals.
2. Arbitrary waveform generation with an external D/A converter.
3. High-speed A/D conversion with an external A/D converter.
4. High-speed digital stimulus for testing and control.
5. Fast block transfer of data between computers.
6. Interrupt or DMA driven background data transfers.
7. High sink current TTL digital I/O (24mA sink current).
8. Data line monitoring.
9. Transient generation and monitoring.

1.3 FEATURES

The PDMA-16 includes the following features:

1. Two 8-bit I/O ports,A and B. Each port is settable as an input or output under software control. The ports are addressable as normal I/O locations using programmed transfer (IN and OUT I/O instructions), or by using the PC's internal 8237 DMA controller. Data may be directly transferred at high speed to/from the ports from/to memory. DMA transfers may be bytes (8-bit) through

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the A Port only, or words (16 bit) using both the A and B Ports. The A DIRECTION and B DIRECTION outputs provide information on the current direction of the ports to simplify interface to devices capable of bidirectional data transfer. In addition, three auxiliary outputs, AUX1 - AUX3 are available from the DMA and Interrupt Control Registers and can be further utilized in general control and handshake functions. In Byte DMA transfers, the B Port is free for use as a regular port for programmed I/O.

2. A DMA transfer may be initiated by an external signal (XFER REQUEST) or by the internal timer. The internal timer consists of a 10MHz crystal oscillator divided through two sections of an 8254 counter. This provides a clock rate ranging from 2.5MHz to 0.0023Hz (about 8 pulses/hr). The choice of external signal or internal clock is via software control. On receipt of a positive edge on the XFER REQUEST input, the XFER ACKNOWLEDGE output goes low. Completion of the transfer to/from memory is acknowledged by the XFER ACKNOWLEDGE output returning to the high state. The operating DMA level is selectable as either Level 1 or 3 under software control. The user is required to initialize the 8237 DMA controller on the PC system board before commencing transfers.
3. An interrupt channel is also provided. Software control allows you to select the active Interrupt Level (2-7) and choose between a positive- or negative-edge external interrupt on the INTERRUPT pin, a periodic interrupt from the PDMA-16's internal timer, or a terminal interrupt generated from the 8237 DMA controller.

1.4 ACCESSORIES

A set of software drivers for interpreted Microsoft BASIC, compiled BASIC, and QuickBASIC, and Assembly Language are included in the Distribution Software. Because of variety of possible applications, interrupt service routines, etc. the drivers perform basic commom-to-all-application functions such as setting up the DMA Controller and internal Timer. Fully commented source code for the driver (PDMA.ASM) is supplied and can serve as a starting point for more application-specific, user-developed drivers. Some simple BASIC test and example programs are also included.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 2

INSTALLATION

This chapter contains instructions for the installing the PDMA-16 in an IBM PC/XT/AT and compatible models. The chapter begins with procedures for unpacking and inspection. It then describes how to make a back-up copy of the Distribution Software. Descriptions of the Base Address Switch and the I/O Connector and procedures for installing the PDMA-16 are also given.

2.1 UNPACKING & INSPECTING

1. Unpack the board down to its anti-static packaging. If possible, retain the outer packing material in case the board must be returned to the factory for repair.
2. Check to be sure you have received every item on the packing list has been shipped. Report any missing items to the manufacturer.
3. Holding wrapped PDMA-16 Board in one hand, place your other hand firmly on a metal portion of the system chassis (which must be grounded). (This procedure drains any static electricity from your body, preventing damage from such a charge to the board.)
4. Carefully remove the board from its anti-static wrapping.
5. Inspect the board and any other components for shipping damage. If damage is detected, return the board to the manufacturer.

You are now ready to install your PDMA-16. Set the Base Address Switch if necessary (see Section 2.3). Then, install the board as described in section 2.5.

2.2 BACKING UP THE DISTRIBUTION SOFTWARE

As soon as possible, make a working copy of your Distribution Software. You may put the working copy on diskettes or on the PC Hard Drive. In either case, making a working copy allows you to store your original software in a safe place as a backup.

To make a working copy of your Distribution Software, you will use the DOS *COPY* or *DISKCOPY* function according to one of the instructions in the following two subsections.

To Copy Distribution Software To Another Diskette

In either of these instructions, the *source* diskette will be the diskette containing your Distribution Software; the *target* diskette will be the diskette you will copy to. Before you start, be sure to have one (or more,as needed) formatted diskettes on hand to serve as target diskettes.

First, place your Distribution Software diskette in your PC's A Drive and log to that drive by typing **A:** . Then, use one of the following instructions to copy the diskette files.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- If your PC has just one diskette drive (Drive A), type **COPY *.* B:** (in a single-drive PC, Drive A also serves as Drive B) and follow the instructions on the screen.

  If you prefer to use the DOS *DISKCOPY* function, instead of *COPY*, you will type **DISKCOPY A: A:** and follow instructions on the screen. This alternative is faster, but requires access to DISKCOPY.COM, in your DOS files.

- If your PC has two diskette drives (Drive A and Drive B), type **COPY *.* B:** (the same as above) and follow the instructions on the screen.

  If you prefer to use the DOS *DISKCOPY* function, instead of *COPY*, you will type **DISKCOPY A: B:** and follow instructions on the screen. This alternative is faster, but requires access to DISKCOPY.COM, in your DOS files.

To Copy Distribution Software To The PC Hard Drive

Before copying Distribution Software to a hard drive, make a directory on the hard drive to contain the files. While the directory name is your choice, the following instructions use *PDMA16*.

1. After making a directory named *PDMA16*, place your Distribution Software diskette in your PC's A Drive and log to that drive by typing **A:**.
2. Then, type **COPY *.*** *path***PDMA16**, where *path* is the drive designation and DOS path (if needed) to the PDMA16 directory.

When you finish copying your Distribution Software, store it in a safe place (away from heat, humidty, and dust) for possible future use as a backup.

2.3 BASE ADDRESS SWITCH

The PDMA-16 Base Address Switch (the only board switch) is a 6-position DIP switch set at the factory for 300h (see Figure 2-1). Normally, the preset address of 300h will be suitable. However, if 300h presents a conflict, select a new address on an 8-bit boundary anywhere in the in the PC-AT I/O space. Refer to the table below for acceptable areas of selection.

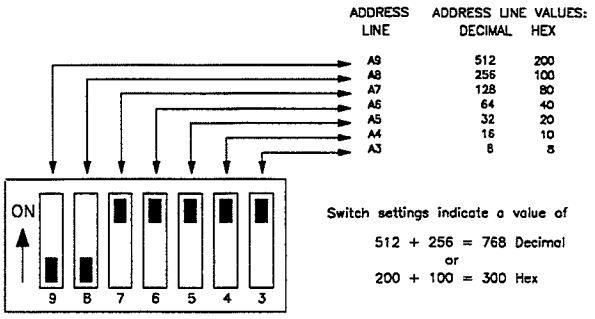


Figure 2-1. Base Address Switch

In selecting a Base Address, bear in mind that the PDMA-16 must be assigned a unique Base Address within the range of 200 to 3FFh (512 to 1023 Decimal). Use the following table as an aid to selecting this address.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PC-AT I/O Address Space

HEX RANGE	USAGE	HEX RANGE	USAGE
000-00F	DMA Controller #1	2D0-2DF	EGA
020-021	Interrupt Controller #1	2E0-2E7	GPIB
040-043	Timer	2E8-2EF	Serial Port
060-064	Controller (Keyboard)	2F8-2FF	Serial Port COM2:
070-071	Real-time Clock & NMI Mask Reg.	300-30F	Prototype Card
080-08F	DMA Page Register	310-31F	Prototype Card
0A0-0A1	Interrupt Controller #2	320-32F	Hard Disk
0C0-0DF	DMA Controller #2	378-37F	Parallel Printer LPT1:
0F0-0FF	Coprocessor	380-38F	SDLC
1F0-1FF	Hard Disk	3A0-3AF	SDLC
200-20F	Game I/O	3B0-3BB	Monochrome Display
238-23B	Reserved	3BC-3BF	Parallel Printer
23C-23F	Reserved	3C0-3CF	EGA
278-27F	Parallel Printer LPT2:	3D0-3DF	CGA
2B0-2BF	EGA	3E8-3EF	Serial Port
2C0-2CF	EGA	3F0-3F7	Floppy Disk Controller
		3F8-3FF	Serial Port COM1:

The PDMA-16 Distribution Software includes an *Install* program to assist you in setting the Base Address Switch. To run this program, proceed as follows:

1. Log to the directory containing the Distribution Software. Then, type **INSTALL** followed by < Enter > .
2. The program will respond with the prompt **DESIRED BASE ADDRESS ----> ?** . Type a Base Address in either decimal or hexadecimal form (a hexadecimal number must be preceded by &H, such as &H300). The computer will display the corresponding Base Address switch settings. If the entry is unacceptable, the computer will display an explanatory statement and a request for another entry.

2.4 OTHER SETTINGS

The only other settings of concern, those of which Interrupt Level and DMA Level to use, are software-selectable; there is no need to be concerned about them at installation. The following information provides a brief description of the PC architecture and its influence on the possible choices.

Standard PC architecture includes an Intel 8237 DMA Controller on the system board that provides 4 DMA channels. The highest priority channel, Level 0, is reserved for internal memory refresh and is not accessible on the expansion bus connectors. Levels 1, 2, and 3 are available on the expansion interface; however, Level 2 is reserved for the floppy disk controller, and (if a hard disk is installed) Level 3 may also be used. Some hard-disk controllers use hardware DMA to transfer data between DOS's disk buffers and the controller board; others use block moves, instead. The method used is controlled by the fixed-disk BIOS, which is located in a ROM on the Controller Board and is specific to the type of controller board used.

BIOS call operation is the same whether the controller uses hardware or software block transfers. This means that on a hard-disk-equipped machine, IBM PC XT or compatible, Level 1 is available for the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDMA-16; Level 3 may also be available. On floppy-disk-only machines (IBM PC or compatible), Levels 1 and 3 are both available. Also, the IBM PC/AT always has Levels 1 and 3 available even when a hard disk is included. Selection of the DMA Level 1 or 3 is under software control by Bit 6 of the PDMA-16 Control Register.

The PDMA-16 may also generate interrupts on any of the available expansion-connector Interrupt Levels (2 thru 7). Like the DMA level, the active Interrupt Level is selected by software control using Bits 4-6 of the PDMA-16's Interrupt Control Register.

It is possible to operate more than one PDMA-16 in a single computer. In this case boards should be placed at different I/O addresses. Since the interrupt and DMA channels employ tri-state drivers, the same interrupt and DMA channels may be shared among boards as long as they are individually enabled and disabled in sequence by the controlling software. If the boards are to perform simultaneous DMA and interrupts, they must operate on different levels.

2.5 I/O CONNECTOR

A standard 37-pin, D-type male connector is used for all I/O. The mating connector is a standard, 37-pin D-type female such as an ITT/Cannon #DC-37S for soldered connections. Insulation displacement (flat cable) types are readily available (for example, Amp #745242-1). Other manufacturers make equivalent parts. If you wish to access connections with screw connectors, use the manufacturer's STA-U Universal Screw Connector Board. The I/O Connector and its signal conductor functions are described in Figure 2-2 and the table of functions that follow.

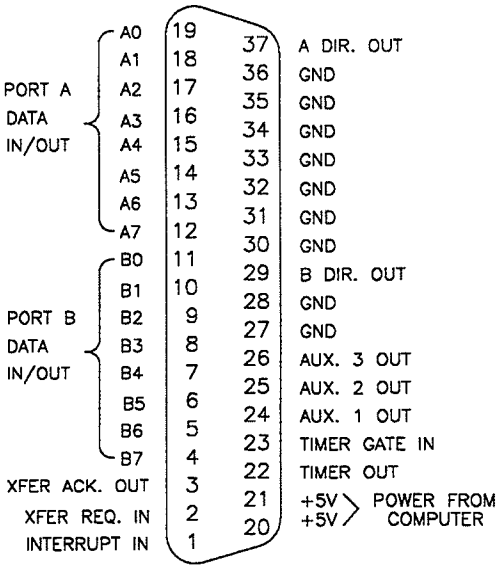


Figure 2-2. Main I/O Connector

I/O Signal Functions

SIGNAL NAME	FUNCTION
INTERRUPT IN	This is a positive or negative edge triggered interrupt input. The slope of the trigger edge is selected by bit D0 of the interrupt control register.
XFER. REQ. IN	A positive edge on this input initiates a DMA transfer with DMA enabled and D3 of the DMA control register = 0.
XFER. ACK. OUT	On receipt of a XFER. REQ. the XFER. ACK. goes low. After the 8237 DMA has finished the byte or word transfer, XFER. ACK. returns high indicating that valid data is on the port(s) in output mode, or that data has been transferred in input mode.
B0 - B7	Port B data (input/output)
A0 - A7	Port A data (input/output)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

+5V This is the +5V logic supply from the computer. Provided that you are sure you can avoid short circuits, overloads and application of external voltages (i.e. any type of abuse which might damage the computer), it may be used to power external peripherals. Be sure to observe available power limits, in any case do not exceed 2 amps due to connector limitations - see Technical Reference Manual.

TIMER OUT This is a positive pulse or square wave output from the timer.

TIMER GATE IN When low, this input inhibits (or holds) external and internal pulses from the timer. The TIMER GATE has an internal 10K pull-up resistor to +5v, if not used this input can be left disconnected.

AUX 1 - 3 These are general purpose control output lines corresponding to spare bits in the DMA and Interrupt control registers.

GND Logic and power return ground.

B DIR OUT Port B direction (output)
0 = Port B input
1 = Port B output

A DIR OUT Port A direction (output)
0 = Port A input
1 = Port A output

2.6 BOARD INSTALLATION

This section provides general instructions for installing the PDMA-16 Board. For more detailed information regarding installation of peripheral boards, consult the documentation provided with your computer.

WARNING
DO NOT ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH THE COMPUTER POWER ON! THIS COULD CAUSE DAMAGE TO YOUR COMPUTER!

To install the Board,

1. Turn off power to the PC and to all attached options.
2. Unplug the power cords of all attached options from the electrical outlets. Make a note of where all the cables and cords are attached to the rear of the system unit and disconnect.
3. Remove the cover of the PC. To do this, first remove the five cover mounting screws on the rear panel of the computer. Then, slide the cover of the computer about 3/4 of the way forward. Tilt the cover upwards to remove.
4. Choose an available option slot. Loosen and remove the screw at the top of the blank adapter plate. Then slide the plate up and out to remove.
5. Hold the PDMA-16 in one hand. With the other hand, touch any metallic part of the PC/AT cabinet. This will safely discharge any static electricity which has built-up in your body.
6. Set the Base Address Switch as described in section 2.3.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7. Align the gold edge connector with the edge socket and the back adapter place with the adapter plate screw. Gently press the board downward into the socket. Re-install the adapter plate screw.
8. Replace the computer's cover. Tilt the cover up and slide it onto the system's base, making sure the front of the cover is under the rail along the front of the frame. Install the mounting screws.
9. Plug in all cords and cables. Turn the power to the computer back on.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 3

REGISTER STRUCTURES

At the lowest level, the PDMA-16 is programmable via I/O (Input/Output) instructions. In BASIC, these are the INP(X) and OUT X,Y functions. Assembly Language and most other high- level languages have equivalent instructions (for example, IN AL,DX and OUT DX,AL in Assembly). Use of these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, this type of programming requires that you have a full understanding of the devices, data format, and architecture of the PDMA-16.

3.1 I/O MAP

PDMA-16 boards use 16 consecutive addresses starting at the Base Address in the computer's I/O space, as shown in the following table.

ADDRESS	FUNCTION	TYPE
Base Address+0	A Port	Read/Write
Base Address+1	B Port	Read/Write
Base Address+2	DMA Control	Read/Write
Base Address+3	Interrupt Control	Read/Write
Base Address+4	Counter 0	Read/Write
Base Address+5	Counter 1	Read/Write
Base Address+6	Counter 2	Read/Write
Base Address+7	Counter Control	Write
	Counter Status	Read

Note that addresses Base Address +4 thru +7 correspond to the 8254 timer.

3.2 PORTS A & B

Port A data corresponds one-for-one with the data bus. When transferring words by DMA, Port A transfers the Least Significant Byte and Port B transfers the Most Significant Byte. For DMA byte transfers, Port A becomes the default I/O port (you cannot use Port B for DMA byte transfers). Both ports A and B can always be written to and read as standard I/O ports by an I/O instruction whether or not they are involved in DMA transfers. Port B can be used as a standard I/O port while DMA byte transfers are taking place through port A.

The Port A data format is as follows:

BIT:	D7	D6	D5	D4	D3	D2	D1	D0
Base +0	A7	A6	A5	A4	A3	A2	A1	A0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Port B data format is as follows:

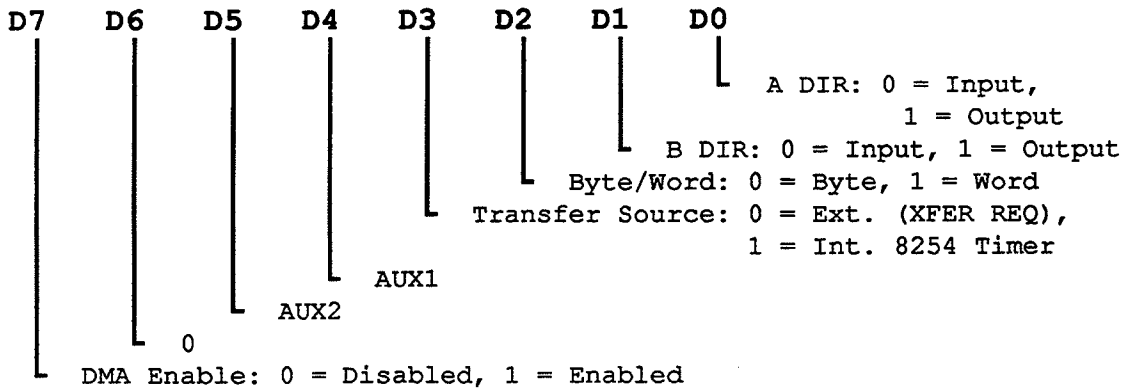
BIT:	D7	D6	D5	D4	D3	D2	D1	D0
Base +1	B7	B6	B5	B4	B3	B2	B1	B0

Notes:

1. Port B is automatically set up as an input on power up of the computer (when hardware reset bus line is active). It will not be reset by a soft boot (Ctrl-Alt-Del sequence).
2. Data direction is controlled by D0 (for Port A) or D1 (for Port B) of the DMA Control Register.
3. In input mode each line of port B presents 1 74LS load to the driving source. In output mode, outputs will sink 24mA and are 74LS and standard TTL compatible. For driving CMOS inputs, 330 ohm pull up resistors to +5V are recommended.
4. In output mode, data can be read back from Port B. This data corresponds to the actual data on the output pins and may not correspond to the data written if an output line is shorted or faulty.

3.3 DMA CONTROL REGISTER

DMA Control Register Bits have the following functions:



NOTES:

Notes:

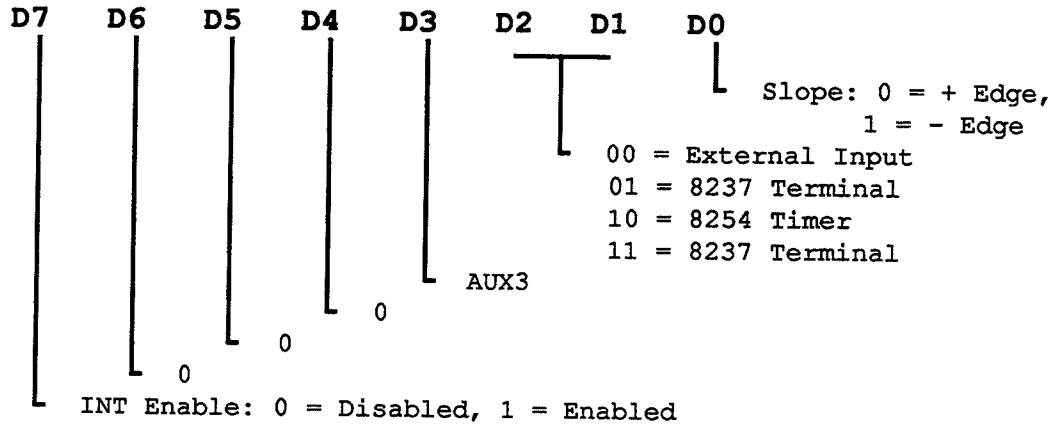
1. The DMA Control Register is cleared on power up (reset) of the computer, thus disabling DMA and setting Ports A and B as inputs.
2. The DMA ENABLE bit should be set before enabling the 8237 Mask Register to avoid spurious transfers.
3. When DMA ENABLE = 0, the associated DMA Level is tristated and may be shared by other devices.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4. The AUX1 and AUX2 bits correspond to unused bits. For convenience, these have been brought out to provide additional outputs on the rear connector.

3.4 INTERRUPT CONTROL REGISTER

The Interrupt Control Register bits have the following functions:



Notes:

1. The Interrupt Control Register is cleared on power-up (reset) of the computer, thus disabling interrupts.
2. The INT ENABLE bit should be set before enabling the 8259 Interrupt Controller Mask Register to avoid generation of spurious interrupts.
3. When INT ENABLE = 0, the associated Interrupt Level is tristated and available for use by other devices.
4. The AUX3 bit corresponds to an unused bit. For convenience, it has been brought out to provide an additional output on the rear connector.

3.5 8254 TIMER

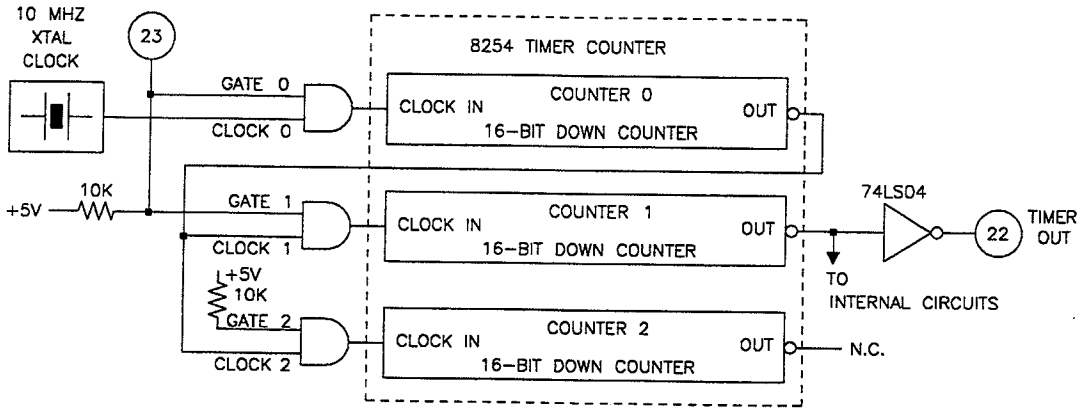


Diagram of 8254 connection in PDMA-16.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I/O locations BASE +4 thru +7 correspond to the 8254 Timer. A full description of the 8254 capabilities and programming is provided in the Intel 8254 data sheet and a partial description, adequate for programming, is supplied in Chapter 6. Its use in the PDMA-16 is purely for periodic timing pulse generation and its connection is shown below. Note that Counter 2 is not hardware accessible to you, although it may be loaded and read for timing purposes.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 4

PROGRAMMING FOR THE CALL MODES IN BASICA & QUICKBASIC

4.1 THE PDMA-16 CALL MODES

The PDMA16.BIN driver (Distribution Software) supports 13 CALL modes (numbered 0 - 12). Each mode performs a specific operation. This chapter describes considerations for using these modes in either BASICA or QuickBASIC.

For convenience, a list of the 13 CALL modes follows.

MODE	DESCRIPTION
MODE 0	Initialize the PDMA-16.
MODE 1	Setup and Perform DMA Transfer.
MODE 2	Return Status.
MODE 3	Set Timer.
MODE 4	Digital Output.
MODE 5	Digital Input.
MODE 6	Auxiliary Output.
MODE 7	Interrupt Enable.
MODE 8	Interrupt Disable.
MODE 9	Allocate Memory for DMA.
MODE 10	Deallocate Memory Segment.
MODE 11	Move Data from Source to Destination.
MODE 12	Disable DMA.

Each mode is fully described in Chapter 5.

4.2 PROGRAMMING IN BASICA

Loading The Machine Language CALL Routine PDMA.BIN

You may choose from two methods for loading the PDMA.BIN driver. The option you choose will depend on the amount of RAM memory installed in your machine. For most applications, Method 1 will suffice; this method requires a minimum of 256K RAM. If your machine does not have this much RAM, use Method 2.

Method 1

This method calls for loading the driver outside of the BASIC workspace using the **BLOAD** command. You must initially select a segment of memory in which at least 6 KBytes is clear at the beginning and does not conflict with any other program or data area. For example, you could choose &H2800 which is at 160K. Then proceed as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
xxx10   DEF SEG = &H6000      'Sets up load segment
xxx20   BLOAD "PDMA.BIN",0    'Loads at 2800:0000
xxx30   PDMA = 0
xxx40   DIM D%(16)
...
xxxxx   DEF SEG = &H6000
xxxyy   CALL PDMA (MD%, D%(0), FLAG%)
xxxzz   etc.
```

An example of this method in given in the DMA1.BAS file, in your Distribution Software.

Method 2

This method involves loading the driver within the BASIC workspace using the **BLOAD** command. You must initially select a segment of memory in which at least 6 KBytes is clear at the beginning and does not conflict with any other program or data area. (If you do interfere with another program's use of memory, the CALL routine will not work and your PC will most likely "hang-up". If this happens, reboot your computer.) To determine a safe loading area, proceed as follows.

1. Determine the size of the BASIC workspace. By nature of its design, the maximum memory segment that BASIC is able to use is 64K. To determine the size of the workspace, at the DOS prompt type **BASIC(A)** .

   The computer will respond with something like

   ```
   BASIC
   The IBM Personal Computer Basic
   Version D1.10 Copyright IBM Corp.  1981, 1982
   61807 Bytes Free
   ```

   The exact number of "Bytes Free" varies with the version of BASIC(A) and DOS but is usually greater than 60KB. A number less than 60K indicates that your PC's memory is already heavily used. If this is true, you will have to load the CALL routine by further contraction of the BASIC workspace and by loading the routine at the end of the newly defined workspace.

   You will need to 6K (6144 byte) space for the PDMA16.BIN driver. To do this, first determine how much memory BASIC is able to use. Then, load BASIC(A) from DOS with the command **BASIC(A)** .

   Note the number of Bytes Free in BASIC's greeting message. Now, use a **SYSTEM** command to return to DOS and reload BASIC(A) with the optional **/M** parameter, as follows:

   BASIC(A) /M:WS

   Try setting the WS (workspace) parameter to 30000 or 40000 and then note the number of Bytes Free. Continue this process, increasing the workspace parameter until the Bytes Free number is reduced by at least 6144 bytes. Then, you can either load BASIC(A) by specifying this workspace or include a **CLEAR** command right at the beginning of your program, as follows:

   xxx10 CLEAR, WS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2. Identify the segment BASIC occupies in memory. In all versions of Microsoft derived BASIC, you can determine the segment from the contents of absolute memory locations &H511 and &H510. These locations hold the current BASIC segment, which we can call SG. Determine SG as follows:

```
xxx20 DEF SEG = 0
'define current segment = 0000 before reading
'absolute addresses 0000:0510 & 0000:0511
xxx30  SG = 256*PEEK(&H511) + PEEK(&H510)
```

The segment address at which the CALL routine can be loaded will be at the end of the working space. For example,

```
xxx40 SG = WS/16 + SG       'remember, segment addresses
                            'are on 16-byte boundaries
```

Load the routine as follows:

```
xxx50 DEF SEG = SG
xxx60 BLOAD "PDMA.BIN",0      'loads routine at SG:0000
```

A **BLOAD** must be used as you are loading a binary (machine language) program. Once loaded, the CALL can be entered as many times as needed in the program after initializing the call parameters MD%, D%, FLAG%. Enter these parameters prior to the CALL sequence as follows:

```
xxx70 DEF SEG = SG
xxx80 PDMA = 0
xxx90 CALL PDMA (MD%, D%(0), FLAG%)
```

Note that *PDMA* is a variable that specifies the memory offset of the starting address of the CALL routine from the current segment, as defined in the most recent preceding *DEF SEG* statement.

Notes

a. *PDMA* is the offset (actually zero) from the current segment, as defined by the last *DEF SEG* statement (*DEF SEG* tells your BASIC interpreter where the CALL routine is located). Avoid inadvertently redefining the current segment somewhere in a program before entering the CALL. It is good practice to immediately precede the CALL statement by the appropriate *DEF SEG* statement (the same one you preceded your *BLOAD* with).

b. *CLEAR* sets working space from the bottom of the BASIC working area up, whereas you must set aside space for your subroutine from the top of memory down. Any attempt to *CLEAR* more space than is actually available will load your routine over the end of the BASIC program, data space, and stack and will hang up the computer. Be especially careful this does not happen if you are memory-limited and later load BASIC with DEBUG or some coresident program without declaring a corresponding reduction in workspace (WS) in your *CLEAR* statement.

Format Of The Call Statement

If you are inexperienced with CALL statements, this section will help you to understand their use. Prior to entering the CALL, the **DEF SEG = SG** statement sets the segment address where the CALL subroutine is located. The CALL statement for the PDMA16 driver is of the format

```
xxx  CALL PDMA (MD%, D%(0), FLAG%)
```

where

PDMA = The address offset from the current segment of memory as defined in the last **DEF SEG**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

statement.

MD% = Call parameter representing the Mode Number.

D% = Call parameter(s) representing the Data Variable(s).

FLAG% = Call parameter representing Errors.

In executing the CALL, the addresses of the variables (pointers) are passed in the sequence written to BASIC's stack. The CALL routine unloads these pointers from the stack and uses them to locate the variables in BASIC's data space so data can be exchanged with them. There are several important rules to remember when using the CALL statement:

1. The CALL parameters must always be written in the correct order. The subroutine does not recognize the names of the variables, just their locations . For example, if the line *xxxxx CALL PDMA (D%(0), MD%, FLAG%)* is used, the CALL routine would interpret *D%(0)* as the MODE number, *MD%* as the data, etc.
2. All parameters must be defined as integer type variables. The CALL does not perform any error checking on the variable type. If you use the wrong variable type, the CALL function will not perform correctly.
3. Do not perform any arithmetic functions within the parameter list brackets of the CALL statement. For example, **`CALL PDMA (MD% + 2, D%(0) * 8, FLAG%)`** is illegal and will produce a syntax error.
4. Do not use constants for any of the parameters. For example, this is illegal: **`CALL PDMA (7, 2, FLAG%)`** .
5. You may assign any name you wish to the variables.
6. Declare all variables before executing the CALL. If you do not, the simple variables will be declared by default on execution but array variables cannot be dimensioned by default and must be dimensioned before the CALL to pass data correctly if used as a CALL parameter. Most MODEs of the CALL routine require multiple items of data to be passed in an array. For this reason, *D%(0)* is specified as the data variable so that the CALL routine can locate the whole array from the position of its initial element.

   Likewise, any of the other CALL parameters may be integer array variables if required, and you can name any number of different integer data arrays for output and input. It is permissible to dimension arrays with more elements than will be used by the CALL. Unused elements will be unchanged and for example could be used for tagging data with time, date, or other information.

Execution Times - Compiled BASIC

The execution times of most modes of the PDMA-16 are limited by the software. Additionally, other operations that process data in your program may also delay your overall throughput. One solution which would improve the speed of your program is to use compiled BASIC or QuickBASIC.

4.3 PROGRAMMING IN QUICKBASIC

This section contains information for users wishing to write data acquisition programs in QuickBASIC (QB). In addition to the information provided in this section, you may want to consult the QuickBASIC example programs in the Distribution Software.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Loading The Program

A QuickBASIC program will have to make calls to an external driver/library. Your Distribution Software contains the following linkable driver/libraries:

PDMAQB45.QLB Load this Quick Library into your QuickBASIC Integrated Environment Version 4.5 or lower.

PDMAQBX.QLB Load this Quick Library into your QuickBASIC Extended Environment Version 7.0 .

PDMAQB45.LIB LINK this library to your stand-alone QuickBASIC program.

Load the PDMAQB45.QLB Quick Library into the QB environment from the DOS command line using the **/L** switch, as follows: **QB /L PDMAQB45** .

To load an application program (such as EXG3.BAS) along with the Quick Library, you load the Quick Library and EXG3.BAS together by typing **QB /L PDMAQB45 EXG3** .

Use PDMAQBX.QLB in an identical manner when using the QuickBASIC Extended Enviroment (QBX) Version 7.0.

Declaring The Driver

Before you use the driver/library, you must declare the CALL label to make it known to your application. Make this declaration by inserting the following at the beginning of your program:

```
DECLARE SUB QBPDMA (MODE%, BYVAL dummy%, FLAG%)
```

where *QBPDMA* is the common entry point to the driver/library for driver modes.

NOTE: All subroutine DECLAREs in your program MUST be before any $DYNAMIC arrays are allocated. $DYNAMIC data is allocated space in the FAR heap, outside the default data segment. All arrays used for data acquisition must be declared as $DYNAMIC; QuickBASIC assumes $STATIC data (Default data segment) unless otherwise specified.

Format Of The Call Statement

Unlike BASICA, the first and third parameters in QB are passed as variables while the second parameter is passed as a pointer. This arrangement is necessary because the second parameter must represent the offset of the command integer array. To pass the actual offset, use the *VARPTR* function as follows:

```
CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)
```

where

MD% is the Mode number.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D%(N) is the parameter array, and FLAG% returns detected errors.

The VARPTR function returns the address of D%(0), which you pass as a value to the driver. The driver uses that value as a pointer to the first element of our command integer array D%.

D% is declared as a $STATIC array since it must reside in the default data segemnet in order to be relative to the QB data segment, as required by the driver. Since D% is passed to and returned by the external driver, declare this parameter as an inter-module global variable, as follows:

```
...
DIM D%(16)
COMMON SHARED D%()
...
```

The *COMMON* makes this variable visible between modules, and the *SHARED* statement at the module level makes it known globally in this module. After declaring all your $STATIC variable, you may declare any large $DYNAMIC arrays for DMA data acquisition.

```
...
REM $DYNAMIC
DIM DMA%(10000)
...
```

NOTE: All $DYNAMIC data declaration must occur AFTER all COMMON and DECLARE statements in your program. If you get the QB error *COMMON and DECLARE must preceed all executable statements* , double check the order of DECLAREs COMMONs and $DYNAMIC declarations.

See Section 2.5 ($STATIC and $DYNAMIC Arrays) and (COMMON statement) in the QuickBASIC Language Reference Manual for a detailed discussion of the factors that determine array types.

To summarize, your program header should look as follows:

```
...
DECLARE SUB QBPDMA (MODE%, BYVAL dummy%, FLAG%)
...
DIM D%(16)
COMMON SHARED D%()          ' Parameter Array for mode CALLs
...
REM $DYNAMIC                '
DIM DMA%(10000)             ' DMA array in FAR Heap
...
```

Refer to the QB example programs (Distribution Software) for more detail. Note that the address of the array *DMA%()* is a "FAR" address (32-bits)and may be determined by using the built-in operators VARSEG and VARPTR. Your PDMA-16 driver/library (.LIB or .QLB) is designed to accept 32-bit addresses, allowing you to avoid specifying absolute addresses as in BASIC.

```
D%(1) = 5000H
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Instead, when passing FAR pointers to QB, specify -1 where you normally specify the "memory segment" and supply the FAR pointer's segment and offset in unused *D%()* parameter array elements. The following is an example of how this is accomplished for the DMA Mode 1. Assume that the array **DMA%(10000)** is previously declared in the $DYNAMIC area, as described above.

```
...
D%(0) = 10000              'Sample count
D%(1) = 0                  'flag to look for Seg:ofs below!
D%(2) = 1                  'internal clock
D%(3) = 0                  'single cycle
D%(4) = VARPTR(DMA%(0))    'Offset of DMA%(0)
D%(5) = VARSEG(DMA%(0))    'Segment of DMA%(0)
...
```

A final note on arrays. If you wish to erase and redimension an array during program execution, it should be declared as **$DYNAMIC**. The **REDIM** statement can now change the size but not the dimensional structure. The **ERASE** statement is not necessary.

Making Executable Programs

There are two ways to create a stand-alone QuickBasic program that is executable from the DOS command line:

1. From within the QB (QBX) environment.
2. From the DOS command line.

When making a stand-alone executable, you need the P32QB45.LIB driver/library.

From within the QB Environment

To make an executable from the QB Environment, use the following sequence:

1. Invoke the environment, as follows:

```
QB /L PDMAQB45 QBDEMO          ' for QB up Ver 4.5
```

or

```
QBX /L PDMAQBX QBDEMO          ' for QB Ver 7.0
```

2. Select the Run menu item.
3. Select the Make EXE File... from the Run menu.
4. Select Produce: (*D) Stand-Alone EXE File.
5. Select the < Make EXE and Exit > option.

This sequence produces a standalone QBDEMO.EXE that does not require run-time support. As an alternative, you may use the following procedure:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1. Select the Run menu item.
2. Select the Make EXE File... from the Run menu.
3. Select Produce: (*D) EXE Requiring BRUN45.EXE.
4. Select the <Make EXE and Exit> option.

This sequence produces a program QBDEMO.EXE that requires the run-time support program BRUN45.EXE. To run either type of executable program, type: **QBDEMO** .

From the DOS Command Line

To compile QB programs from the DOS command line, use the following command sequence from DOS:

```
BC /e /o QBDEMO.BAS;
LINK QBDEMO,,,PDMAQB45.LIB;
```

The */o* option causes references to the BCOM45.LIB library to be placed in the object module, so the library response need not be given in the LINK line. This sequence will produce a standalone executable.

As an alternative, type

```
BC QBDEMO.BAS;
LINK QBDEMO,,,PDMAQB45.LIB;
```

The absence of the */o* option in the compiler line causes references to the BRUN45.LIB library to be placed in the object module, so the library response need not be given in the LINK line. This sequence produces an executable program that requires BRUN45.EXE to be in a subdirectory named \BIN or in the current directory at the time the program is executed.

Note that the Compiler and Linker expect to find the necessary executables (BC.EXE, LINK.EXE, etc..) in a subdirectory named \BIN, and to find Libraries in the directory named by the environment variable *LIB* (*SET LIB=* in your AUTOEXEC.BAT.)

To run either type of executable program, type **QBDEMO** .

The Software Driver CALL Label

You must declare the CALL label to make it known to your application; make this declaration by inserting the following at the beginning of your program:

```
DECLARE SUB QBPDMA (MD%, BYVAL PARAMS%, FLAG%)
```

Note that all subroutine DECLAREs in your program MUST be made before any $DYNAMIC arrays are allocated. $DYNAMIC data is data that is allocated space in the FAR heap, outside the default data segment. All arrays used for data acquisition must be declared as $DYNAMIC; QuickBasic assumes $STATIC data (Default data segment) unless otherwise specified.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Call Parameters

Declare the mode call parameter array D%(10) as follows:

```
DIM D%(9)
COMMON SHARED D%()
```

By making the array *COMMON SHARED* , other modules and subroutines can use it.

For example, to initialize your PDMA-16 board, use MODE 0 as follows:

```
    :
180 MD% = 0                          'initialize mode
190 FLAG% = 0                        'declare error variable
200 D%(0) = &H300                    'Card BASE ADDRESS
210 D%(1) = 3                        'DMA LEVEL
220 D%(2) = 7                        'INTERRUPT LEVEL
230 D%(3) = 1                        'WORD MODE
240 CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)
250 IF FLAG% <> 0 THEN PRINT "MODE 0 Error # "; FLAG% : STOP
    :
```

Linking To The Driver Interface Module

The QuickBASIC Interface consists three separate Modules:

PDMAQB45.QLB Use when you load the QuickBASIC Enviroment Version 4.5 and you plan to run your program from within the Environment (no EXE envolved here). Use the /L switch to load this Quick Library into QuickBASIC, as follows:

QB /L PDMAQB45 ***your-program***

PDMAQBX.QLB This is identical to PDMAQB45.QLB except that it is designed for QuickBASIC Extended Environment Version 7.0 (QBX). Use the /L switch to load this Quick Library into QuickBASIC, as follows:

QBX /L PDMAQBX ***your-program***

PDMAQB45.LIB Link to this library when you want to make a standalone EXE program from your QuickBASIC (4.5) source. To create such a program, use BC and LINK the QuickBASIC compiler and linker as follows:

```
BC your-program.bas /o;
LINK your-program,,,PDMAQB45.LIB;
```

PDMAQBX.LIB Link to this library when you want to make a stand-alone EXE program from your QuickBASIC (7.0) source. To create such a program, use BC and LINK the QuickBASIC compiler and linker as follows:

```
BC your-program.bas /o;
LINK your-program,,,PDMAQBX.LIB;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE: All $DYNAMIC data declaration must occur after all COMMON and DECLARE statements in your program. If you get the QB error *COMMON and DECLARE must preceed all executable statements* , double check the order of all DELCARE, COMMON, and $DYNAMIC declarations.

4.4 EXAMPLE BASIC PROGRAMS

The Distribution Software contains several example BASICA programs that are commented for listing and examination. For example, *DMA1.BAS* shows how to load PDMA.BIN, initialize the PDMA-16, and perform DMA transfers. *DMA2.BAS* performs the same functions as DMA1.BAS but does not use the driver; it shows what is involved with handling the I/O directly through BASIC, and it shows how much the driver simplifies programming. *BASDEMO.BAS* is an example program that you may compile then load with *PDMABAS.LIB* ; it uses Mode 9 to allocate memory and Mode 10 to deallocate memory. BASDEMO.BAS is not executable from within the BASICA environment.

The remaining BASIC programs are as follows:

- *TIMER.BAS* sets the timer to any rate.
- *STATUS.BAS* returns the status.
- *IN.BAS* reads ports.
- *OUT.BAS* writes to the ports.
- *INT.BAS* enables and disables interrupts.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 5

THE MODE CALLS

5.1 OVERVIEW

This chapter details each of the 13 PDMA-16 MODE Calls. The PDMA.BIN driver (Distribution Software) supports these modes (numbered 0 - 12), and each performs a specific operation.

The particulars for each mode include a description, data required for entry, data to be expected upon exit, and a typical example of a program entry.

A list of the 13 MODE Calls is as follows:

MODE	DESCRIPTION
MODE 0	Initialize the PDMA-16 Driver & Check Hardware.
MODE 1	Setup and Perform DMA Transfer.
MODE 2	Return Status of DMA Transfer.
MODE 3	Set Timer Rate.
MODE 4	Digital Output.
MODE 5	Digital Input.
MODE 6	Auxiliary Output.
MODE 7	Set Up and Enable Interrupt.
MODE 8	Disable Interrupt.
MODE 9	Allocate Memory for DMA.
MODE 10	Deallocate Memory Segment.
MODE 11	Move Data from Source to Destination.
MODE 12	Disable DMA.

5.2 MODE 0: INITIALIZE THE PDMA-16 DRIVER & CHECK HARDWARE

Description:

MODE 0 initializes the driver and PDMA-16 hardware **and must be executed before using any of the other MODEs**. Initializing checks and stores the PDMA-16's Base I/O Address, the desired Interrupt Level for interrupts, and the desired DMA Level. This MODE also determines whether the PDMA-16's A and B Ports will appear as a 16- or 8-bit peripheral for normal I/O.

The Base I/O Address is checked to be in the legal range of 256 - 1008 (100h - 3F0h) for the PC/AT. If not an Error Exit #3 will occur. If OK, the Base I/O Address is stored for use by other MODEs on re-entry to the CALL. A short read/write test is made to some of the PDMA-16 internal register; it is sufficient to detect the presence or absence of the board at the specified I/O Address. If no board is detected (absent board, wrong Base I/O Address), Error #4 (Hardware Error) is returned.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If initialization is successful, any other MODE may be selected on subsequent CALLs. **Trying to select any other MODE before performing initializing MODE 0 will give rise to error # 1.** If an invalid DMA Level number or interrupt level is selected, Errors #5 or #6 are returned.

Entry Data:

D%(0) =	Base I/O Address of PDMA-16, range 100-3F0h.
D%(1) =	Selected DMA Level; 1 and 3 are valid.
D%(2) =	Selected Interrupt Level; 2 - 7 are valid.
D%(3) =	Selects port MODE; 0 = Byte I/O, 1 = Word I/O.
D%(4) thru D%(6) =	Not used, value irrelevant.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12. 3 if Base Address < 100h or > 3F0h. 4 if hardware error, no board, incorrect I/O address. 5 if DMA Level illegal. 8 if Interrupt Level illegal. 12 if Byte/Word MODE specifier D%(3) not 0 or 1.

Programming Example:

A typical start for a program initializing sequence would be as follows:

```
100 DEF SEG = &H4000                        'segment to load driver
110 BLOAD "PDMA.BIN",0                    'load it at zero offset
120 PDMA = 0                                'declare variables
130 MD% = 0                                 'select initialization
140 DIM D%(6)                               'declare data array
150 FLAG% = 0
160 D%(0) = &H300                           'Base Address = 300 hex.
170 D%(1) = 3                               'DMA Level 3
180 D%(2) = 7                               'Interrupt Level 7
190 D%(3) = 1                             'word MODE - 16 bit I/O
200 CALL PDMA (MD%, D%(0), FLAG%)       'initialize
210 IF FLAG%<>0 THEN PRINT "Error in initializing # ";FLAG%:STOP
220 'continue program
```

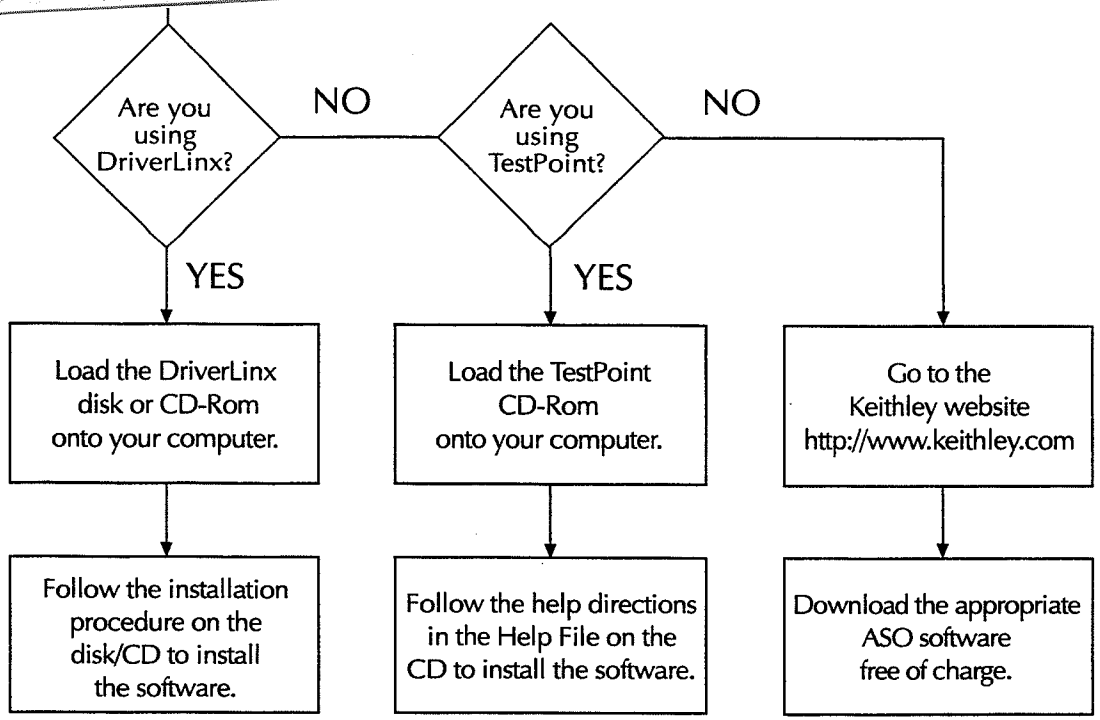
5.3 MODE 1: SET UP & PERFORM DMA TRANSFER

Description:

MODE 1 sets up both the 8237 DMA Controller and the PDMA-16 for an input or output DMA transfer using Bytes or Words. Using Bytes, the DMA hardware works through Port A, while Port B remains open for programmed I/O, and MODE 1 will not disturb its configuration. In Word Mode, DMA uses both Ports A and B; Port A transfers the Less Significant Byte and Port B the More Significant.

The DMA setup, which is performed by the driver, involves the following steps:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



PA-604 Rev. A / 7-98

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

KEITHLEY

Software Quick Start Guide

Please use the following instructions to install software for your board.

DriverLinx Software

If you are using DriverLinx software, load the DriverLinx disk/CD and follow the installation procedure found on the disk/CD.

TestPoint Software

If you are using TestPoint software, load the TestPoint CD and follow the help directions in the Help File found on the CD.

ASO Software

If you want to use ASO software, the software and documentation can be downloaded from our website at http://www.keithley.com free of charge.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1. Disable any DMA transfer already in progress.
2. Check data for valid ranges.
3. Check for a page wrap-around problem, which can occur if for example you want to transfer 40,000 bytes starting at an offset of 32,000 in a page. The 8-bit DMA Page Register selects 1 of 16 pages of 64 Kbytes out of the memory. The PC/AT system board hardware is not capable of incrementing the Page Register from the 8237 DMA Controllers when the end of a page is reached. Instead, DMA will continue at the beginning of the same page and may overwrite program or data memory. This condition of page wrap-around is checked before enabling the DMA operation.
4. The next step is to work out the correct control byte for the PDMA-16 DMA Control Register.
5. The MODE, Initial Address, and Byte Count Registers of the appropriate DMA level of the 8237s are then loaded.
6. The timer is halted prior to enabling the DMA channel so that the first transfer always occurs a fixed delay (corresponding to the timer pulse rate) after setup.
7. The 8237's Mask Register is enabled, opening the DMA channel.
8. The timer is brought up to its correct speed.

The setup procedure is followed by a return to your program. The DMA transfer may continue as the rest of the following program executes. If you wish to determine the progress of a transfer, use MODE 2.

Entry Data:

D%(0) =	Number of transfers in bytes or words.
D%(1) =	Value irrelevant.
D%(2) =	Direction; 0 = input 1 = output.
D%(3) =	Autorecycle On/Off; 0 = Off, 1 = On.
D%(4) =	Transfer Clock Source; 0 = External, 1 = Timer.
D%(5) =	Transfer segment for DMA buffer.
D%(6) =	Transfer offset for DMA buffer.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12. 6 DMA Page Wrap-around Error. 7 DMA direction, D%(2), not 0 or 1. 11 Autorecycle, D%(3), not 0 or 1. 17 Transfer Clock Source, D%(4), not 0 or 1.

Programming Example:

A typical program entry preceding MODE 1 might read as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
210 MD% = 1                                 'select MODE Number
220 D%(0) = 1000                            'transfer 1000 words
230 D%(1) = 1                               'word transfer MODE
240 D%(2) = 1                               'output data on ports A & B
250 D%(3) = 1                               'autorecycle on continuous output
260 D%(4) = 1                               'clock data out with timer
270 D%(5) = &H5000                          'DMA buffer memory segment
280 D%(6) = 0                               'DMA buffer memory offset
290 CALL PDMA (MD%, D%(0), FLAG%)     'execute MODE 1
300 IF FLAG%<>0 THEN PRINT "Error in MODE 1, # ";FLAG%:STOP
310 'continue program
```

A few points of explanation on the preceding Programming Example:

D%(0) always sets the number of transfers whether they are Bytes or Words as selected by D%(1). If D%(0) = 0, then a full page (65,536) transfer will be performed.

D%(2) sets the port(s) direction for input or output. If the board has been set up (MODE 0 initialization) to operate on DMA Levels 1 - 3.

D%(3) controls whether the DMA Controller is set in the Auto-Initialize MODE.

D%(4) controls selection of the DMA transfer request pulse source. If D%(4) = 0, the external XFER. REQ. on Pin 2 of the rear connector is used. If D%(4) = 1, then transfer requests are generated by the internal timer. The timer rate can be set using MODE 3.

D%(5) controls selection of the DMA buffer segment in memory. If D%(5) = &HFFFF, then BASIC's data segment will be used (see caution below).

D%(6) controls the offset of the buffer area within the data segment set by D%(5).

DMA DIRECTLY TO/FROM A BASIC INTEGER ARRAY

If you wish to transfer data directly to/from an integer array, proceed as follows (performing DMA directly to/from an array is a potentially dangerous procedure, read warning at end):

```
210 MD% = 1                                 'select MODE Number
220 N = 360                                 'number of elements in array
230 DIM ARRAY%(N)
240 D%(0) = N+1                             'transfer N+1 words
250 D%(1) = 1                               'must be word transfer MODE
260 D%(2) = 1                               'output data on ports A & B
270 D%(3) = 1                               'auto-recycle on
280 D%(4) = 1                               'clock data out with timer
290 D%(5) = &HFFFF                          'array segment = BASIC's
300 D%(6) = VARPTR(ARRAY%(0))               'array offset
310 CALL PDMA (MD%, D%(0), FLAG%)     'execute MODE 1
320 IF FLAG%<>0 THEN PRINT "Error in MODE 1, # ";FLAG%:STOP
330 'continue program
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Warning !

Since BASIC dynamically allocates variable storage and locates arrays above simple variables, adding a new simple variable after line 310 will unavoidably relocate ARRAY%(*). The DMA hardware will not be aware of this and will carry on outputting data from the locations it got in lines 290 & 300 leading to strange effects especially on input of data. You must use a structured programming method of pre-declaring all variables at the start of your program to safely perform DMA directly in/out of BASIC arrays. If in doubt, use a buffer area external to BASIC.

5.4 MODE 2: RETURN STATUS

Description:

MODE 2 returns information about the status of DMA and interrupt operations. For DMA, D%(3) indicates whether a DMA transfer set up by MODE 1 is still active. It also returns the number of transfers (Bytes or Words) requested in MODE 1 in D%(1) and the number transferred so far in D%(0). If the DMA Controller is operating in nonauto-initialize mode and the transfer is complete (number transferred = number requested), D%(3) returns 0 to indicate DMA inactive. It is also possible for the hardware to generate a terminal interrupt to indicate the end of a DMA transfer, although this MODE does not utilize this capability. MODE 2 also returns the current data directions of Ports A and B and the current interrupt status whether active or done.

Entry Data:

D%(0) thru D%(6) = Value Irrelevant.

Exit Data:

D%(0) =	Number of Bytes/Words transferred.
D%(1) =	Number of Byte/Word transfer requested.
D%(2) =	Auto-Initialize On/Off; 1 = On, 0 = Off.
D%(3) =	DMA Active/Finished; 1 = Active, 0 = Finished.
D%(4) =	Port A Data Direction; 0 = Input, 1 = Output.
D%(5) =	Port B Data Direction; 0 = Input, 1 = Output.
D%(6) =	Interrupt Status; 0 = Disabled, 1 = Enabled.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12.

Programming Example:

A typical programming sequence using MODE 2 is as follows:

```
310 MD% = 2                                    'select MODE Number
320 CALL PDMA (MD%, D%(0), FLAG%)            'execute MODE 2
330 IF FLAG%<>0 THEN PRINT "Error in MODE 2, # ";FLAG%:STOP
340 IF D%(3) = 1 THEN GOTO 320                 'wait till DMA finished
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.5 MODE 3: SET TIMER RATE

Description:

MODE 3 sets the division ratio for the Timer. The Timer output consists of two 16-bit Down Counters driven in cascade from a 10.000MHz 0.01% precision crystal clock. Initializing (MODE 0) sets these counters to operate in 8254 MODE 2 (see Chapter 6) or Rate Generator Mode, which produces a single pulse on terminal count of Counter 1. MODE 0 also initially loads these counters to 65,535 (FFFFh) so that the initialized output rate is about 1 pulse every seven minutes. Immediately after initialization, the counter output is guaranteed to be low for seven minutes, allowing plenty of time for set up of an external device.

This MODE reloads the counters to any other desired rate up to a maximum of 2.5MHz. The output pulse rate is:

Rate = (10,000,000)/(D%(0) * D%(1)) pulses/sec.

Permissible values for D%(0) and/or D%(1) are 2 thru 65,535. Values of 0 and 1 are not legal.

It is also possible to operate the counters in square-wave mode (8254 MODE 3), but this requires direct programming to the 8254 control port, as this driver does not support changing the initialized configuration. Counting may also be enabled and disabled with the TIMER GATE Input (Pin 3). When held low, this input inhibits counting; when open or held high, counting is allowed to proceed.

Entry Data:

D%(0) =	Counter 0 Divider.
D%(1) =	Counter 1 Divider.
D%(2) thru D%(6) =	Not used.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12.

Programming Example:

A typical timer setup to output 1 KHz would be as follows:

```
340 MD% = 3                                  'select MODE Number
350 D%(0) = 1000                             'counter 0 divisor
360 D%(1) = 10                               'counter 1 divisor
360 CALL PDMA (MD%, D%(0), FLAG%)      'execute MODE 3
370 IF FLAG%<>0 THEN PRINT "Error in MODE 3, # ";FLAG%:STOP
380 'continue program
```

Note: The output pulse width from Counter 1 depends on the period of the clock it receives from Counter 0. In this example, Counter 0 outputs a frequency of 10,000/1000 = 10KHz for an output pulse width will be 100 microseconds. Since Counter 1 divides by 10 the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ultimate timer rate is 1KHz. If D%(0) = 100 and D%(1) = 100, the Timer Rate would still be 1KHz, but the pulse width would be 10 microseconds.

5.6 MODE 4: DIGITAL OUTPUT

Description:

MODE 4 performs digital output as a programmed transfer either through the A Port or the B Port or in the case of Word transfers, through both the A and B Ports. Note this MODE does not perform DMA; use MODE 1 instead. The operation performed by this MODE is equivalent to a direct write to the port address using OUT addr, data in most languages. Once written, data is latched on the output port and can be read back using MODE 5 or INP(addr). The operating MODE of the ports (Word/Byte) is set by variable D%(3) in initializing MODE 0. If Word MODE has been selected, then the PDMA-16 appears as a 16-bit peripheral for digital I/O and data is written in a single instruction from Ports A and B combined. If Byte I/O Mode has been selected, Ports A and B are written separately as Byte-wide ports, and the PDMA-16 appears as a Byte-oriented peripheral to the PC/AT bus. Note it is possible to be doing DMA in Word Mode and still perform Byte-wide I/O operations on the A and B ports.

The output port(s) are selected by D%(1). Byte (8 bit) transfers may be made either through the A Port or the B Port. If the PDMA-16 is operating in Word I/O Mode, the value of D%(1) is irrelevant. Word (16 bit) transfers are made through both the A and B Ports with the A Port outputting the Least Significant Byte and the B port outputting the Most Significant.

The data direction of the ports is checked initially by reading the PDMA-16' DMA Control Register. If it differs from the desired output arrangement, the data directions are set accordingly. On power-up, all ports are reset to the input condition.

A check on valid data is performed on Byte transfers only and Error Code #13 is returned if out of range and the output operation abandoned.

Entry Data:

D%(0) =	Output Data; Rrange 0 to 255 if Byte, -32768 to 32767 if Word.
D%(1) =	Output Port(s); Value Irrelevant in Word I/O Mode; 0 = Port A (Byte), 1 = Port B (Byte).
D%(2) thru D%(6) =	Not used.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12. 13 if data D%(0) is out of range, < 0 or > 255 for byte xfers. 14 if port configuration D%(1) data < 0 or > 1.

Programming Example:

The following example outputs all 1's (65,535) as a word on Ports A and B. Note that we wish to output unsigned data, whereas integers store signed data (see Appendix E for further explanation).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The examples assume we have performed initialization , MODE 0, with D%(3) = 1 (word MODE). Initially we start with the data in a real variable D and sign correct it for output in an integer in Line 400:

```
380 MD% = 4                                  'select MODE Number
390 D = 65535                                'data to output
400 IF D > 32767 THEN D = D - 65536          'sign correct
410 D%(0) = D                                'output data
420 CALL PDMA (MD%, D%(0), FLAG%)          'execute MODE 4
430 IF FLAG%<>0 THEN PRINT "Error in MODE 4, # ";FLAG%:STOP
440 'continue program
```

The following code would output a travelling pattern of 1 on Port B assuming Byte I/O Mode had been selected in initializing MODE 0 (D%(3) = 0):

```
380 MD% = 4                                  'select MODE Number
390 D%(1) = 1                                'select port B
400 FOR I% = 0 TO 7
410 D%(0) = 2^I%                             'output data
430 CALL PDMA (MD%, D%(0), FLAG%)          'execute MODE 4
440 IF FLAG%<>0 THEN PRINT "Error in MODE 4, # ";FLAG%:STOP
450 NEXT I%
460 'continue program
```

5.7 MODE 5: DIGITAL INPUT

Description:

MODE 5 reads Byte (8 bit) data from Port A or Port B, as specified by D%(1) or Word (16 bit) data from Ports A and B. In the case of Word data reads, Port A provides the Least Significant Byte, and Port B the Most Significant. Data is always returned in D%(0). The operating MODE of the ports (Word/Byte) is set by variable D%(3) in initializing MODE 0. If Word MODE has been selected, then the PDMA-16 appears as a 16-bit peripheral for digital I/O, and data is read in a single instruction from Ports A and B combined. If Byte I/O MODE has been selected then Ports A and B are read separately as Byte-wide ports, and the PDMA-16 appears as a Byte-oriented peripheral to the PC/AT bus. Note it is possible to be doing DMA in Word MODE and still perform Byte-wide I/O operations on the A and B ports. If Word MODE has been selected on initialization, the value of D%(1) is irrelevant.

Note that the ports may be in input or output mode on entry to MODE 5. If D%(2) is 0, MODE 5 will not interfere with the port configuration i.e. will read back data from a port set in output MODE by MODE 4 and leave it set in output MODE. If however you want to alter the configuration and make the port an input port only, D%(2) should be set to 1. An addressed port will then the be set in input MODE prior to reading it and will remain in input MODE afterwards until changed by an output operation using MODE 4. Ports are automatically configured as inputs by the computer on power up reset.

Entry Data:

D%(0) = Value irrelevant.
D%(1) = Input Port(s); Value irrelevant if in Word I/O Mode; 0 = Port A (Byte)
1 = Port B (Byte).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D%(2) = Change Configuration; 0 = No Change, 1 = Change Addressed Port to Input Mode.
D%(3) thru D%(6) = Not Used.

Exit Data:

D%(0) = Data from Port.
D%(1) thru D%(6) = Unchanged.
FLAG% = 0 if OK; otherwise:
1 if MODE Number < 0 or > 2
14 if port configuration (D1) data <0 or >1.
15 if configuration change data (D2) not 0 or 1.

Programming Example:

Set Port A to input and read (assumes PDMA-16 initialized in Byte I/O MODE), as follows:

```
460 MD% = 5                                  'select MODE Number
470 D%(1) = 0                                'select Port A
480 D%(2) = 1                                'change Port A to input config.
490 CALL PDMA (MD%, D%(0), FLAG%)     'execute MODE 5
500 IF FLAG%<>0 THEN PRINT "Error in MODE 5, # ";FLAG%:STOP
510 PRINT "Data from Port A = ";D%(0)
520 'continue program
```

Set Port B to output, output data, and read it back, as follows:

```
380 MD% = 4                                   'select digital output MODE
390 D%(0) = 99 : X% = D%(0)                   'output data and save in X%
400 D%(1) = 1                                 'set Port B to output Byte
410 CALL PDMA (MD%, D%(0), FLAG%)      'execute MODE 4
420 IF FLAG%<>0 THEN PRINT "Error in MODE 4, # ";FLAG%:STOP
430 MD% = 5                                   'select digital input MODE
440 D%(1) = 1                                 'select Port B
450 D%(2) = 0                                 'don't change Port B config.
460 CALL PDMA (MD%, D%(0), FLAG%)      'execute MODE 5
470 IF FLAG%<>0 THEN PRINT "Error in MODE 5, # ";FLAG%:STOP
480 IF X% <> D%(0) THEN PRINT "Error on Port B"
490 'continue program
```

5.8 MODE 6: AUXILIARY OUTPUT

Description:

MODE 6 provides a means of writing data to any of the auxiliary data bits of the DMA or Interrupt Control Registers. Data is OR'd into the registers so that other control bits are not altered.

If any of the data in D%(0) thru D%(2) is not 0 or 1, Error Code 16 is returned.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Entry Data:

D%(0) =	AUX 1 Data (0 or 1 permitted).
D%(1) =	AUX 2 Data (0 or 1 permitted).
D%(2) =	AUX 3 Data (0 or 1 permitted).
D%(3) thru D%(6) =	Not used.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if MODE Number < 0 or > 12. 16 if any auxiliary data bit is not 0 or 1.

Programming Example:

```
520 MD% = 6                                    'select MODE Number
530 D%(0) = 0                                  'AUX 0 = 0
540 D%(1) = 1                                  'AUX 1 = 1
550 D%(2) = 1                                  'AUX 2 = 1
560 CALL PDMA (MD%, D%(0), FLAG%)          'execute MODE 6
570 IF FLAG%<>0 THEN PRINT "Error in MODE 6, # ";FLAG%:STOP
580 'continue program
```

Note: There is no MODE provided in the driver for reading back the state of the AUXiliary control bits, although since both the DMA and Interrupt Control Registers are R/W registers, it is possible to do this. The following program will perform this function.

```
590 BASE = &H300                          'provide Base I/O Address
600 IF (INP(BASE + 2) AND &H10)=&H10 THEN AUX1=1 ELSE AUX1=0
610 IF (INP(BASE + 2) AND &H20)=&H20 THEN AUX2=1 ELSE AUX2=0
600 IF (INP(BASE + 3) AND &H8)=&H8 THEN AUX3=1 ELSE AUX3=0
```

5.9 MODE 7: SETUP & INTERRUPT ENABLE

Description:

MODE 7 installs an interrupt handling routine and enables a hardware interrupt on the level previously selected in Initialization MODE 0. An interrupt may be generated from one of three sources: an external interrupt on rear connector Pin 1 (the internal PDMA-16 timer) or a terminal interrupt from the selected DMA level. The source is selected by D%(0). In the case of external interrupts, an interrupt will be generated on the positive edge of the signal if D%(0) = 0 or the negative edge if D%(0) = 1. After the edge-initiated interrupt, it is a requirement of the 8259 interrupt controllers in the PC/AT that the interrupt remain asserted until the interrupt service routine starts execution. The PDMA-16 hardware includes an internal interrupt latch which should be cleared at some point in your interrupt handler by reading the Interrupt Status Register at I/O address BASE + 0Ah.

The various steps performed by MODE 7 are:

1. Check for valid source data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

2. Disable any active interrupt on the selected level.
3. Install interrupt vectors using DOS function call 25h of INT 21h.
4. Write correct control Byte to PDMA-16 Interrupt Control Register.
5. Enable the 8259 Interrupt Controllers by clearing the appropriate Mask Register Bit for the level selected. The Interrupt Level is selected during Initialization MODE 0 and is stored in the Interrupt Level Select Register.

A sample "beep the bell" interrupt service routine is supplied in the PDMA.ASM source listing (labelled INTh:) and assembled into the PDMA.BIN driver. You can modify this interrupt service routine to do whatever you wish. To develop your own routine, use any text editor to expand and modify the code and re-assemble with the Macro Assembler following the instructions in the file HOWTOBIN.DOC to generate a BASIC callable .BIN file.

The PDMA-16 can generate a hardware interrupt on any of the 11 expansion bus Levels 3 - 7, 9 - 12, and 14 - 15. The level is software-selectable through the Interrupt Level Select Register, which is set during MODE 0 initialization. The PC/AT's 8259 interrupt controllers can prioritize 15 different hardware interrupts. Level 0 is the highest priority and is used by the internal timer, which generates an interrupt about 18 times/sec. This is used by the BIOS and DOS to provide the system time and date. Level 1 is used by the keyboard to signal that a key has been pressed and invoke a keyboard handling routine. Level 2 is used to cascade the second 8259 controller. Levels 0, 1, 2, 8, and 13 are internal to the PC/AT and not available on the expansion bus. The interrupt levels have been assigned by IBM to the standard peripherals as follows:

Level 0	:	Used for interval timer, time of day (not on bus)
Level 1	:	Used by keyboard (not on bus)
Level 2	:	Used for cascade input on PC/AT only (not on bus)
Level 3	:	Used by COM2: serial port if installed
Level 4	:	Used by COM1: serial port if installed
Level 5	:	Used by LPT2: or hard disk if installed
Level 6	:	Used by floppy disk drive adapter
Level 7	:	Used by LPT1:
Level 8	:	Used internally by real time clock (not on bus)
Level 9	:	Unassigned
Level 10	:	Unassigned
Level 11	:	Unassigned
Level 12	:	Unassigned
Level 13	:	80287 coprocessor error (not on bus)
Level 14	:	Fixed disk drive controller
Level 15	:	Unassigned

Generally the unassigned Levels 9 - 12 and 15 are the best choices, but you may use any other level if you know that the corresponding peripheral device is not installed; for example, if you have no COM2: serial port, Level 3 would be free. It is best to avoid multiplexing one Interrupt Level between two or more adapters, although this can be done with the restriction that only one adapter's interrupt can be active at a time. Do not use Levels 6 and 14, which are usually in use by the disk drives.

Since the upper Levels 8 - 15 are cascaded through Interrupt Level 2, they actually have a higher priority than Levels 3 - 7. If no lower level interrupt is pending and interrupts are enabled, an interrupt will normally be serviced within a few microseconds of its generation. If a lower level interrupt collides with a simultaneous one from the PDMA-16, it can delay servicing of the PDMA-16 interrupt. The usual culprit here is the Timer Interrupt, on Level 0, it can occasionally delay the PDMA-16 interrupt by 30-40 microseconds, which in most cases is negligible but in some cases may be

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

a problem. If it is a nuisance, the delay variation can be eliminated by disabling the timer interrupt through the 8259 Mask Register (located at I/O port &H21), and also by refraining from using the keyboard or COM: ports.

Once a PDMA-16 hardware interrupt has been enabled by MODE 7, it will remain active until disabled by MODE 8 or stopped by a "self extinguishing" interrupt service routine (one that does the equivalent of MODE 8 after a certain number of operations).

Entry Data:

D%(0) =	Specifies interrupt source: 0 = External Input, positive slope. 1 = External Input, negative slope. 2 = DMA Terminal Interrupt. 3 = Timer Interrupt.
D%(1) thru D%(6) =	Not used.

Exit Data:

D%(0) thru D%(6) =	Unchanged.
FLAG% =	0 if OK; otherwise: 1 if Mode Number < 0 or > 12. 18 if Interrupt Source Data (D0) is out of range: < 0 or > 3.

Programming Example:

The following program will beep the bell ad nauseam every two seconds:

```
520 MD% = 3                                   'set timer
530 D%(0) = 20000                             'counter 0 divisor
540 D%(1) = 1000                              'counter 1 divisor - 0.5Hz
550 CALL PDMA (MD%, D%(0), FLAG%)           'execute MODE 3
560 IF FLAG%<>0 THEN PRINT "Error in MODE 3, # ";FLAG%:STOP
570 MD% = 7                                   'enable interrupt
580 D%(0) = 3                                 'from timer
590 CALL PDMA (MD%, D%(0), FLAG%)           'execute MODE 7
600 IF FLAG%<>0 THEN PRINT "Error in MODE 7, # ";FLAG%:STOP
610 'continue program, try MODE 8 to shut it off!
```

Note: MODE 2 provides information on whether the interrupt is active if there is any doubt.

5.10 MODE 8: DISABLE INTERRUPT

Description:

MODE 8 sets the appropriate bit in the 8259 Interrupt Controllers Mask Register to disable further interrupts for the level selected in MODE 0. This driver does not restore previous interrupt routine vectors (possibly required if sharing an Interrupt Level with another device). MODEs 7 and 8 can be expanded to do this, if required.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Entry Data:

D%(0) thru D%(6) = Value irrelevant.

Exit Data:

D%(0) thru D%(6) = Unchanged.
FLAG% = 0 if OK; otherwise: 1 if Mode Number < 0 or > 12.

Programming Example:

To turn off the interrupt started in MODE 7's Programming Example:

```
610 MD% = 8                                          'select MODE
620 CALL PDMA (MD%, D%(0), FLAG%)                   'execute MODE 8
630 IF FLAG%<>0 THEN PRINT "Error in MODE 8, # ";FLAG%:STOP
640 'continue program
```

5.11 MODE 9: ALLOCATE MEMORY FOR DMA

Description:

MODE 9 allocates a memory buffer for DMA, using the DOS Memory Allocate Function 48. This mode returns the following: a segment in D%(5), which will be the segment for DMA; the DMA segment offset in D%(6), which will always be 0; and the value of the actual segment allocated by DOS in D%(7), which may be different than the DMA segment in D%(5) and should be passed to MODE 10 when Deallocation of the DMA segment is necessary. MODE 9 is not usable in BASICA, but it may be used in QuickBASIC, Quick C, etc. if enough memory is available, or it may be used in compiled BASIC, C, PASCAL, or FORTRAN.

For more information on MODE 9, refer to Appendix D.

Entry Data:

D%(0) = Number of Words or Bytes to Allocate.
D%(1) = 0 = Bytes, 1 = Words.

Exit Data:

D%(5) = DMA Segment.
D%(6) = Offset (Always 0).
D%(7) = Actual Allocated Segment.
Flag% = 0 If OK; otherwise:
19 Memory Allocate/Deallocate Error.

Programming Example:

A typical program entry preceding MODE 9 might read as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
210 MD% = 9                    'Select mode number.
220 D%(0) = 5000               'Allocate 5000 Words.
230 D%(1) = 1                  'Words (should be the same as MODE 1 D%(1)).
240 'Execute MODE 9
250 CALL PDMA(MD%,D%(0),FLAG%)
260 DMASEG=D%(5)               'Pass to MODE 1.
270 DMAOFF=D%(6)               'Pass to MODE 1.
280 ACTSEG= D%(7)              'Pass to MODE 10.
290 IF FLAG% < > 0 THEN PRINT "MODE 9 ERROR #"; FLAG%
300 'CONTINUE PROGRAM
```

5.12 MODE 10: DEALLOCATE MEMORY SEGMENT

Description:

MODE 10 releases memory allocated by MODE 9, using DOS Deallocate Memory Function 49. This mode requires the Actual segment to be passed in D%(0), whose value is returned by MODE 9 in D%(7). MODE 10 should be called when the program is finished using the DMA buffer allocated by MODE 9.

Entry Data:

D%(0) = Actual Segment (value returned by MODE 9 in D%(7)).

Exit Data:

Flag% = 0 if OK; otherwise:
19 Memory Allocate/Deallocate Error.

Programming Example:

A typical program entry preceding MODE 10 might read as follows:

```
210 MD% = 10                   'Select mode number.
220 D%(0) = ACTSEG             'Segment to release from MODE 9.

230 'Execute MODE 10.
240 CALL PDMA(MD%,D%(0),FLAG%)
250 IF FLAG% < > 0 THEN PRINT "MODE 10 ERROR #";FLAG%
260 'CONTINUE PROGRAM.
```

5.13 MODE 11: MOVE DATA FROM SOURCE TO DESTINATION

Description:

MODE 11 moves data from array-to-array, array-to-memory, memory-to-array, or memory-to-memory. The data format can be Words or Bytes, and the strating element can be specified.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Entry Data:

D%(0) = Number of Words or Bytes.
D%(1) = Source Packing; 0 = Bytes, 1 = Words.
D%(2) = Source Segment.
D%(3) = Source Offset.
D%(4) = Source Index.
D%(5) = Destination Packing; 0 = Bytes, 1 = Words.
D%(6) = Destination Segment.
D%(7) = Destination Offset.
D%(8) = Destination Index.

Exit Data:

Flag% = 0 if OK; otherwise
20 Byte/Word Count is 0 or Negative.
21 D%(1) and D%(5) Source and Destination Packing should be 0 for Bytes or 1 for Words.

Programming Example:

A typical program entry preceding MODE 11 might read as follows:

```
210 MD% = 11                    'Select mode number.
220 D%(0) = 1000                'Number of Words.
230 D%(1) = 1                   'Copy Words.
240 D%(2) = &h8000              'From Memory Segment 8000h.
250 D%(3) = 0                   'Offset 0.
260 D%(4) = 0                   'Copy from first Element.
270 D%(5) = 1                   'Copy Words.
280 D%(6) = -1                  'Use basics segment.
290 D%(7) = VARPTR(A%(0))       'Offset of our array.
300 D%(8) = 0                   'Copy to first element.

310 'Execute MODE 11.
320 CALL PDMA(MD%,D%(0),FLAG%)
330 IF FLAG% < > 0 THEN PRINT "MODE 11 ERROR #";FLAG%
340 'CONTINUE PROGRAM
```

5.14 MODE 12: DISABLE DMA

Description:

MODE 12 terminates all current DMA operations.

Entry Data:

None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Exit Data:

None.

Programming Example:

A typical program entry preceding MODE 12 might read as follows:

```
210 MD% = 12                    'Select mode number.
220 D%(0) - D%(8)               'Don't care.

230 'Execute MODE 12.
240 CALL PDMA(MD%,D%(0),FLAG%)
250 IF FLAG% < > 0 THEN PRINT "MODE 12 ERROR #";FLAG%
260 CONTINUE PROGRAM
```

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 6

PROGRAMMBLE INTERVAL TIMER

6.1 THE 8254 PROGRAMMABLE INTERVAL TIMER

Intel's 8254 is the Programmable Interval Timer used in the PDMA-16. This is a flexible, but complex device consisting of three independent 16-bit presettable down counters. Each counter may be programmed to divide by any integer in the range 2 - 65,536. As configured in the PDMA-16, the input of Counter 0 connects to an 0.01% precision 10MHz crystal oscillator. The output of Counter 0 connects to to both the inputs of Counters 1 and 2. Counter 2 output is buffered and inverted and available to the user as the TIMER OUT (Pin 22). The other counter, Counter 3, has no external connections but can be loaded and read if required. The primary purpose of the 8254 is to serve as a periodic source for interrupts and DMA transfers; no attempt has been made to use its other modes of operation. The following brief description provides information for programming, the Intel 8254 data sheet provides more comprehensive details. A block diagram of the PDMA-16 counter arrangement is shown in Figure 6-1.

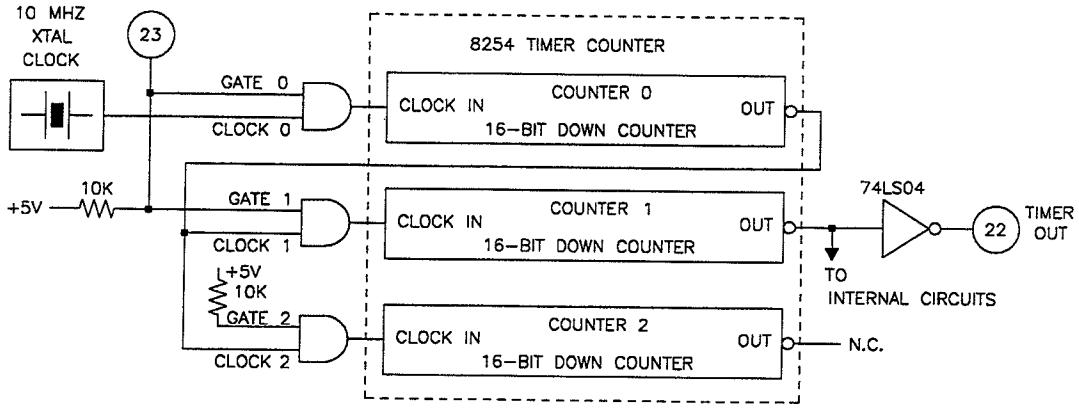


Figure 6-1. Programmable timer configuration.

Each counter has a clock input, a gate input that controls counting and triggering, and an output. All the gate inputs are tied toigether and are available externally on Pin 23 (TIMER GATE IN). There are six possible operating configurations for each counter, as follows:

CONFIGURATION	DESCRIPTION
0	PULSE ON TERMINAL COUNT. The output is initially low for this configuration. After the count loads and the counter decrements thru zero, the output goes high and remains high until the counter is reloaded. The counter continues to decrement after passing thru zero and counting can be inhibited by a low gate input. This mode produces a single positive going output transition such as may be required in a time delay initiated by the program.
1	PROGRAMMABLE ONE SHOT. The output goes low after a rising edge of the gate input and goes high when the counter passes thru zero. The period that the output is low is set by the loaded count. If the gate input goes high again before the one shot has timed out, a new timing cycle is initiated (the one shot is re-triggerable and, if a new

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

count is loaded, it will not become effective until any cycle in progress has terminated). This provides a hardware triggered delay or one-shot.

2 RATE GENERATOR (or Divide-By-N counter). The output goes low for one input clock period every N counts, where N is the count loaded. The gate input when low, forces the output high, and on going high, reloads the counter. Thus, the gate input can be used to synchronize the counter. This configuration is useful for generating periodic interrupts to trigger A/D conversions.

3 SQUARE-WAVE GENERATOR. Similar to Configuration 2 except that the output is high for half of the count and low for the other half. If N is even, a symmetrical square wave output is obtained. If N is odd, the output is high for (N+1)/2 counts and low for (N-1)/2 counts (has a 1-count assymmetry). This configuration can be used in the same way as Configuration 2 for periodic triggering or for frequency synthesis.

4 SOFTWARE TRIGGERED STROBE. After the mode is set the output is high. When a count of N is loaded the counter begins counting and the output will go low for one input clock period as it passes thru zero. The cycle is repeated on loading another count. The gate input may be used to inhibit counting.

5 HARDWARE TRIGGERED STROBE. This is essentially the same as Configuration 1, except that the output will go low for one clock period at the end of the cycle and return high again. The start of the cycle is triggered by the rising edge of the gate input, and as in Configuration 1, is retriggerable.

The 8254 programmable interval counter occupies four I/O address locations in the PDMA-16 I/O address map:

ADDRESS	REGISTER TYPE	DESCRIPTION
Base Address +4	Read/write	Counter 0
Base Address +5	Read/write	Counter 1
Base Address +6	Read/write	Counter 2
Base Address +7	Write only	Control

Before loading or reading any of the individual counters, the 8254 conrol register must be loaded with data setting the counter operating configuration as above, the type of read or write operation that will be performed (see following), and the modulus or binary (0-65,535) or BCD (Binary Coded Decimal 0-9,999). The format of the control byte is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

SC1-0 - Control which counter is selected.

SC1	SC0	COUNTER
0	0	0
0	1	1
1	0	2
1	1	Readback Command (see next section)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RL1-0 - Control the type of Read/Load operation.

RL1	RL0	OPERATION
0	0	Counter latch (see following)
0	1	Read/load most significant byte
1	0	Read/load least significant byte
1	1	Read/load least significant byte, followed by most significant byte (MSB).

M2-0 - Control counter configuration as above.

M2	M1	M0	CONFIGURATION
0	0	0	0 - Pulse on terminal count.
0	0	1	1 - Programmable one shot.
X	1	0	2 - Rate generator.
X	1	1	3 - Square wave generator.
1	0	0	4 - Software triggered strobe.
1	0	1	5 - Hardware triggered strobe.

BCD - Controls binary/decimal counting.

BCD	COUNTER TYPE
0	Binary 16 bits
1	Decimal 4 decades

The counters may be programmed to count in binary (modulus 2) or binary coded decimal (modulus 10) modes by the BCD bit. The binary mode with a full count of 65,535 has the obvious advantage of providing a greater count range than the BCD mode which has a 9,999 full scale and is also more compatible with internal storage of binary integers used by the PC.

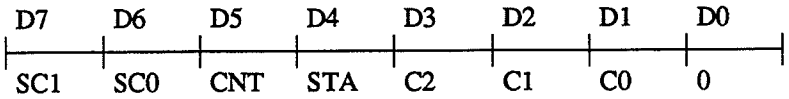
6.2 READING & LOADING THE COUNTERS

If you attempt to read the counters on the fly with a high-input frequency, you will most likely obtain erroneous data. This is caused partly by the rippling of the counter during the read operation and also by the fact that the low and high bytes are read sequentially rather than simultaneously, making it highly probable that carries will be propagated from the low to high byte during the read cycle. To circumvent these problems, you can perform a counter latch operation in advance of the read cycle. To do this you load the RL 1/0 bits of the control byte with "0 0" which instantaneously latches the count of the selected counter in a 16 bit hold register. An alternative method of latching counter(s) which has an additional advantage of operating simultaneously on several counters is by use of the readback command - see below. A subsequent read operation on the selected counter returns the held value. Latching is the only way of correctly reading a counter "on the fly" without disturbing the counting process. If you do not specify a latching operation, then the counter itself will be read. You can only rely on directly read counter data if the counting operation is suspended while reading e.g by removing the clock input or taking the TIMER GATE low.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For each counter you are required to specify in advance the type of read or load operation that you intend to perform. You have a choice of loading/reading the high byte of the count or the low byte of the count, or the low byte followed by the high byte. This last mode is of the most general use and is selected for each counter by setting the RL 1/0 bits to "1 1". Subsequent read/load operations must be performed in pairs in this sequence, otherwise the internal sequencing flip-flop of the 8254 will get out of step.

If the SC0 and SC1 bits are both set to 1, you can perform two types of operations. When CNT=0 (see below) counters selected by C0 thru C2 are latched **simultaneously**. When STA=0, the counter status byte will be read on accessing the counter I/O location. The status byte provides information on the current output state of the counter, and its operating configuration. The readback command byte format is as follows:



SC0 and SC1 = 1: Read-back Command

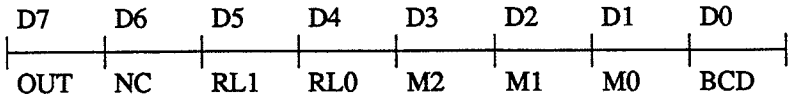
CNT: When 0, latches counters selected by C0 - C2.

STA: When 0, returns status byte of counters selected by C0 - C2.

C0 - C2: When high, select a particular counter for a readback operation:

C0 = 1 selects Counter 0
C1 = 1 selects Counter 1
C2 = 1 selects Counter 2

The status byte returned if STA = 0, consists of:



OUT: Current state of counter output

NC: Null count. This indicates when the last count loaded into the counter register has actually been loaded into the counter itself. The exact time of load depends on the configuration selected. Until the count is loaded into the counter itself, it cannot be read from the counter.

If both the STA and CNT bits are set low and RL0 and RL1 have both been previously set high in the counter control byte (selecting 2 byte reads), then reading a selected counter location will yield as follows:

1st. read : Status byte
2nd. read : Low byte of latched data
3rd. read : High byte of latched data
4th. read : Low byte of counter direct

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5th. read : High byte of counter direct

Programming examples in BASIC are as follows:

```
xxx10  OUT BASE + 7, &H36        'set Counter 1 to squarewave mode
xxx20  OUT BASE + 7, &H76        'set Counter 2 to squarewave mode
xxx30  X = 1000                  'number to load both counters
xxx40  XH% = INT(X/256)          'high byte
xxx50  XL% = X - 256 * XH%       'low byte
xxx60  OUT BASE + 4, XL%
xxx70  OUT BASE + 4, XH%         'load Counter 0
xxx80  OUT BASE + 5, XL%
xxx90  OUT BASE + 5, XH%         'load Counter 1
```

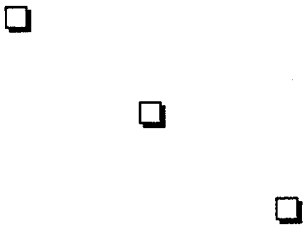
The two counters will now be dividing the 1 or 10 MHz xtal clock by 1,000,000 (1,000 x 1,000) so that COUNTER 2 OUT will now be outputting a 1 or 10Hz squarewave (depending on crystal input).

The following lines would determine the status of Counter 1:

```
yyy10  OUT BASE +7, &HE4         'control byte to read status
yyy20  PRINT HEX$(INP(BASE + 5))     'read status
```

■■■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 7

APPLICATIONS

7.1 TYPICAL HANDSHAKE CONNECTION

This chapter describes methods for connecting and operating the PDMA-16 and then expands to specific examples using A/D and D/A converters.

Whether you are transferring bytes or words into or out of the PDMA-16, the transfer control signals *XFER REQ* and *XFER ACK* work the same way. The following diagrams show their timing.

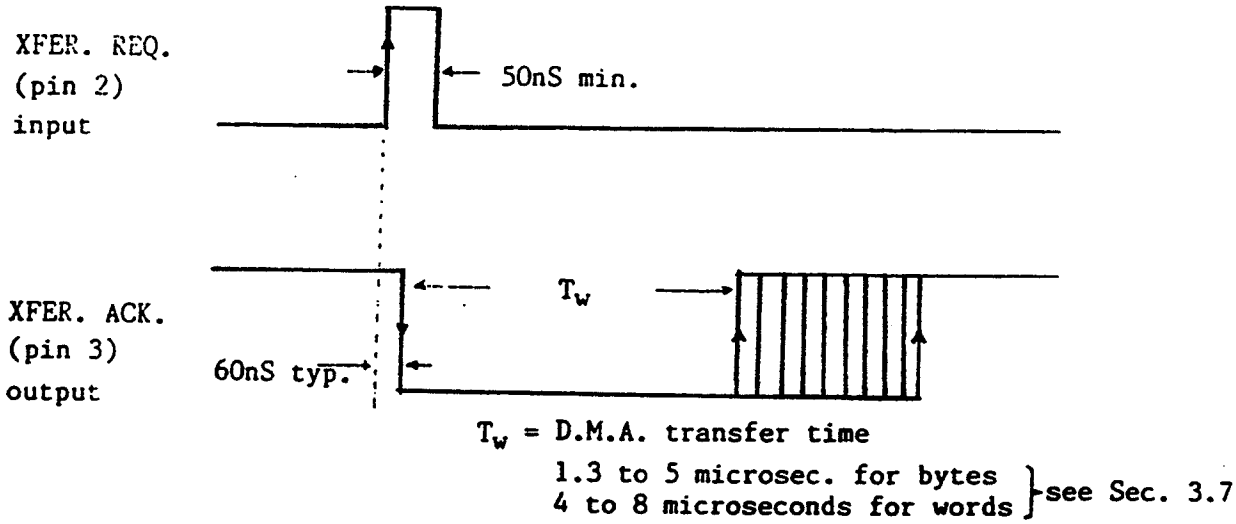


Figure 7-1. Timing diagrams for XFER REQ and XFER ACK.

For inputs to the PDMA-16, present data to the port(s). When XFER REQ is stable, the input should be go high. About 60 ns later, XFER ACK goes low, indicating that the PDMA-16 has initiated a DMA request cycle. Input data must remain stable until XFER ACK returns high, indicating completion of the transfer cycle. The PDMA-16 can then accept the next byte or word of data, and the cycle can repeat to store it in the next memory location. This handshake is the same whether you are transferring bytes or words even though in the case of word transfers, the PDMA-16 is actually making two internal byte transfers. When transferring words, the XFER ACK goes low upon receipt of the XFER REQ and remains low until the whole word transfers.

For outputs from the PDMA-16, when the next byte or word of data is required, the XFER REQ input should go high. XFER ACK will go low upon receipt of the transfer request and remain low until the byte or word completes its transfer and the new data is valid. The data then remains latched on the output ports until the next transfer request. If you are operating in word mode, the PDMA-16 will actually make 2-byte transfers internally. The first, least significant byte, will appear on Port A a few microseconds ahead of the more significant byte arriving on Port B. If the output must change instantaneously from one word to the next, additional buffering will be required as shown in the next section. This avoids a byte by byte change and prevents intermediate steps when driving output devices such as D/A converters.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you want to clock data out at a steady rate, the internal 8254 timer may be substituted for the external XFER REQ by appropriate programming of the DMA control register. The timer can be set to a suitable rate. Note that the timer pulse that generates the transfer request does not appear on the XFER REQ input (this input is inactive), but the same timer pulse that generates the internal transfer request is always present on the TIMER OUT. This is sometimes useful as a reference signal or in double buffering the output data and reclocking to eliminate jitter due to DMA latency. Timer operation can be suspended by taking the TIMER GATE input low and will continue from where it left off on taking the TIMER GATE input high again.

The A DIR and B DIR port direction outputs provide information on the signal direction of the ports and if required can be used to activate bidirectional drivers for rebuffering the output data. This can let you perform a series of DMA outputs to one device connected to the ports followed by a series of DMA inputs from another on the same ports. The direction signals may be used to enable the right devices as appropriate.

When enabled through the interrupt control register, the INTERRUPT INPUT feeds an external interrupt signal through to any of the PC expansion bus interrupt levels. The external interrupt pulse may be as short as 100ns, as it triggers a monostable in the PDMA-16 which generates a longer pulse (about 300 microseconds) adequate for correct interrupt service operation. Interrupts commence on negative or positive edges of the external signal, the slope being selected through the interrupt control register. It is also possible to select the timer output or the DMA controller terminal count pulse as interrupt sources through software control via the interrupt control register.

The auxiliary outputs (AUX1 - AUX3) provide three extra output bits that are user programmable and may be useful in selecting and controlling external devices. Since these bits are part of the DMA and interrupt control registers, it is necessary to first read these registers and OR the auxiliary data in, so that none of the other register bits are affected. The PDMA.BIN driver does this operation automatically.

Power from the PC's +5 V supply is available from the PDMA-16 connector. This power can support external buffering logic, small peripheral devices, and DC/DC converters as well as provide a connection point for pull-up and terminator resistors (if used). When using the internal computer power, be careful not to apply external power sources such as line voltage to the connections as this could cause very considerable damage to your computer. Although the +5 V supply is a convenient feature, it should be used with caution. Use an external power supply if there is any possibility of abuse. The +5 V supply is well protected against short circuits which will result in a shut down of the whole computer. To restore power, switch off the computer, remove the short circuit and switch the computer back on again. The amount of power available depends on the other peripheral devices in the computer, but in any case it is recommended that you limit current drawn to less than 2 amps to avoid copper trace and connector limitations on the PDMA-16.

7.2 WAVEFORM GENERATION WITH A D/A CONVERTER

Note the additional buffering of the PDMA-16 output to obtain simultaneous update of all bits and minimize glitches. There are two ways of clocking the latches: on the rising edge of the XFER ACK signal, when new data is available, or on the rising edge of the timer pulse, which initiates the next DMA transfer, loading data from the previous DMA transfer. The first method suffers from the DMA-latency delay, which is variable and will introduce a few microseconds of "jitter" in the sample outputs. The second method resynchronizes data to the clock. Although data is 1-word delayed, it is perfectly timed at equal intervals. Data can output single-shot or continuously using Mode 1 of the PDMA.BIN driver. The schematic below is a working example; other components (16-bit D/As) will

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

also work.

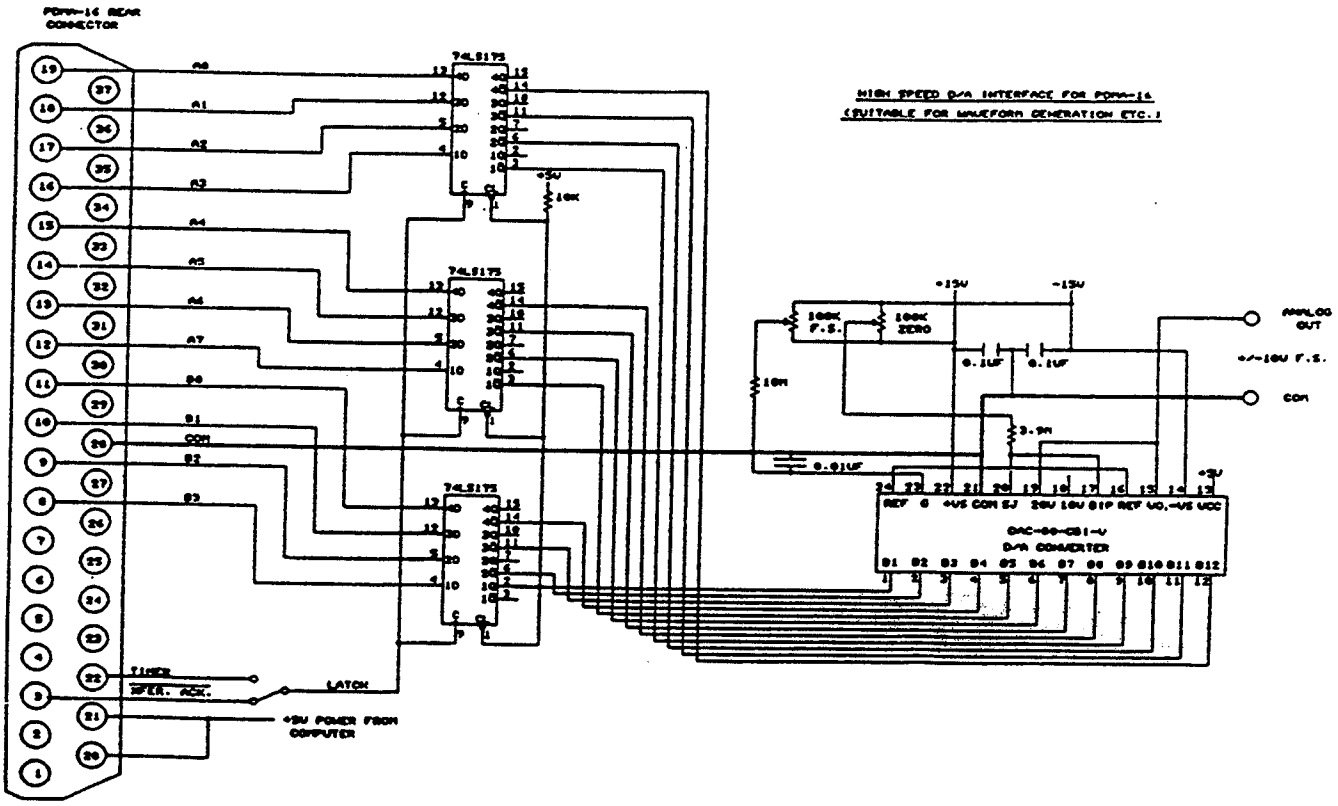


Figure 7-2. High-speed D/A interface for PDMA-16.

The PDMA-16 delivers +5 V power from the computer on Pins 20 and 21 of the connector. This power supports the logic circuits of the external interface. A small DC/DC converter (such as the manufacturer's PG-408) could also generate the ±15 V analog supplies for the D/A from the +5V power. An external supply is also usable.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7.3 HIGH SPEED A/D CONVERSION

The following schematic shows connections for a 12-bit A/D converter (16-bit A/Ds could also be used). The PDMA-16 timer triggers the A/D, and the A/D's EOC (end of conversion) signal generates the DMA XFER REQ after each conversion. Rates in excess of 100KHz are obtainable using Mode 1 of the PDMA.BIN driver. The AUX outputs are used to select 1 of 8 channels using an input multiplexer.

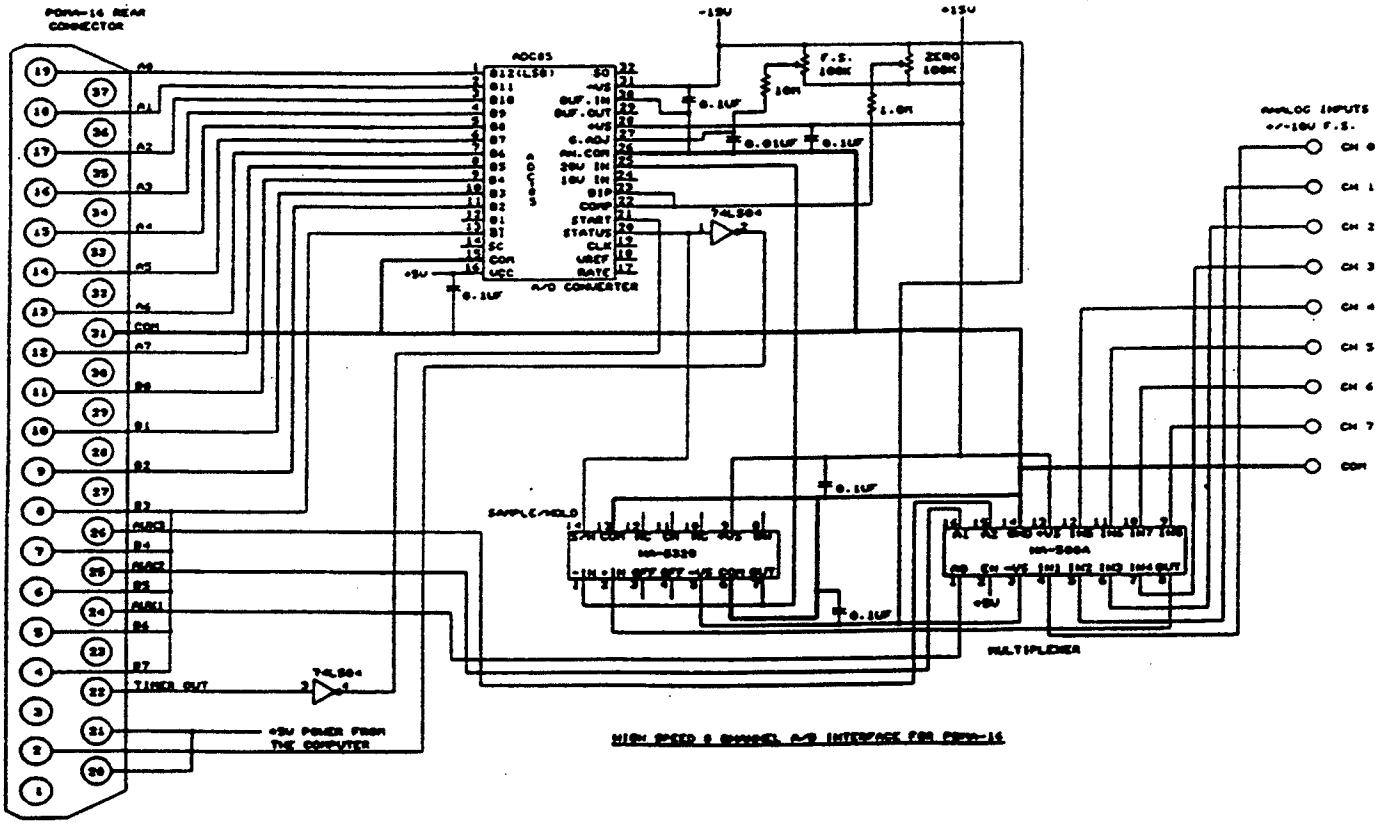


Figure 7-3. High-speed 8-channel interface for the PDMA-16.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7.4 COMBINED A/D & D/A CONVERSION USING DIRECTIONAL CONTROLS

This arrangement combines the previous two examples and uses the A DIR and B DIR direction output signals to control the enabling of the A/D or D/A converter. This circuit permits the digitization of an event at high speed with the A/D and reconstruction of this event back through the D/A either at the same speed or a reduced speed – to drive a plotter for instance. The circuit requires a minimum of external circuitry.

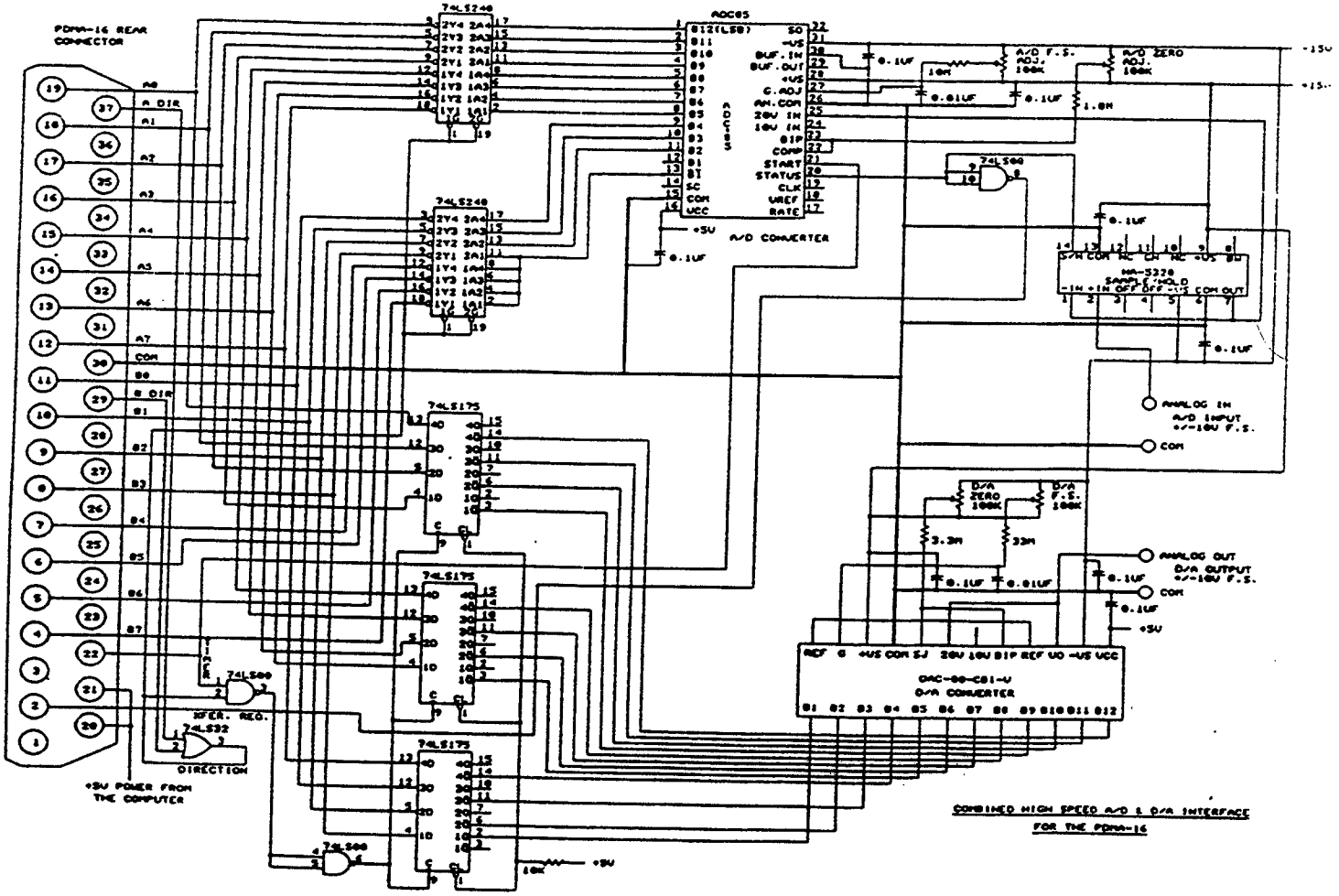
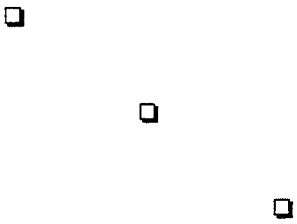


Figure 7-4. Combined high-speed A/d and D/A interface for the PDMA-16.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 8

SERVICE & REPAIR

8.1 SERVICE & REPAIR

The PDMA-16 requires no periodic calibration. If a digital output or input appears damaged and faulty, you have two options. One, all critical integrated circuits that communicate with the external world are in sockets and are thus easily replaceable: Section 8.2 contains the replacement procedure. Two, you may return the board to the manufacturer for repair; include a brief description of the problem.

8.2 PERFORMING YOUR OWN REPAIRS

This series of tests and fixes involves a voltmeter, an oscilloscope, and possible replacement of some standard 74LS series logic available from many manufacturers.

The first step is to diagnose the fault. The most likely fault is damage to an output or input from static electricity, shorts to power supplies, or other overvoltages. Check each output port using BASIC or DEBUG, as follows:

```
OUT BASE + 2, 3          'set ports A & B to output mode
OUT BASE, 0              'check all lines port A low
OUT BASE + 1, 0          'check all lines port B low
```

Check Outputs A0 - A7 and B0 - B7 with a digital voltmeter. Every output should be between 0 and +0.4v . Next set all the outputs high, as follows:

```
OUT BASE, 255            'set port A outputs high
OUT BASE + 1, 255        'set port B outputs high
```

Use the voltmeter to check that every output is between +2.4 V and +5 V. If any output is faulty, replace U17 (74LS373) for any of the bits in Port A or U19 (74LS373) for any of the bits in Port B. If there is no fault, check the inputs, as follows:

```
PRINT INP(BASE)          'read port A - should return 255
PRINT INP(BASE + 1)      'read port B - should return 255
```

Next, output zero again:

```
OUT BASE, 0
OUT BASE + 1, 0
```

and read back result, as follows:

```
PRINT INP(BASE)          'read port A - should return 0
PRINT INP(BASE + 1)      'read port B - should return 0
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If there is a fault in the returned data, replace U18 (74LS244) for any of the Port A bits or U20 (74LS244) for any of the Port B bits.

Check the other inputs and outputs as follows:

1. A DIR and B DIR - write 0 and 3 alternately to the DMA Control Register at BASE +2. If either output fails to change state, replace U15 (74LS04). The fault may also be due to a damaged DMA Control Register chip (see Step 2, below).
2. AUX 1 and AUX 2 - write 0 and 48 alternately to the DMA Control Register at BASE +2. If either output fails to change state, replace DMA Control Register chip U7 (74LS273).
3. AUX 3 - write 0 and 8 alternately to the interrupt control register at BASE + 3. If the output fails to change state, replace U5 (74LS273).
4. If you cannot read back the contents of the DMA Control Register correctly and have replaced U7, then replace U6 (74LS244).
5. If you cannot read back the contents of the Interrupt Control Register correctly and have replaced U5, then replace U4 (74LS244).
6. Run TIMER.BAS and if you cannot see a 1 KHz repetition rate pulse on the TIMER OUT, replace U14 (74LS14N). Any other problem with the timer. For example, TIMER GATE inoperative, no pulse with U14 replaced, or wrong output frequency. Try replacing U16 (8254).
7. Run DMA1.BAS using an auto-initialize input mode and selecting the internal timer as source. The XFER ACK should go low for several microseconds at the timer rate, TIMER OUT can be used to synchronize a scope. If XFER ACK is stuck low or does not appear to respond, replace U10 (74LS74).

This series of tests will detect the majority of faults caused by damage to the inputs or outputs. All other faults will require return of the board to the manufacturer.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX A

SPECIFICATIONS

+5V Power	850mA typical / 1A max.
-5V, +12V, -12V Power	Not used.
I/O Ports	Ports A & B - each of 8 bits. Direction software-selectable. Auxiliary - 3 outputs.
Logic Output Levels	Ports A & B - TTL/DTL: 0.4V max low at Isink = 24mA; 2.4V min high at Isource = -2.6mA. All other outputs TTL/DTL: 0.4V max low at Isink = 8mA; 2.4V min high at Isource = -400uA.
Logic Input Levels	All inputs TTL/DTL compatible: 0.8V max low level at -400uA; 2.0V min high level at 20uA.
Timer	Internal 8254 timer & 10MHz crystal clock. Pulse rates from 2.5MHz to 0.002Hz.
I/O Address	Can be set on any 8-bit boundary from 0h to 3F8h (200h - 3F8h useable in PC).
PC Bus Internal Loading	One 74LS TTL load on all inputs.
Interrupts	Software-selectable on Levels 2 - 7. Active when interrupt enabled, tri-state otherwise.
Interrupt Source	1 of 3 software selectable: External input ±slope; PDMA-16 timer; DMA terminal pulse.
DMA Level	Software selectable Level 1 or 3 (active only in DMA transfer, tri-state otherwise).
DMA Transfer Source	1 of 2 software selectable: External request; PDMA-16 timer.
DMA Transfer Rate	350,000 bytes/200,000 words/sec asynchronous. 200,000 bytes/120,000 words/sec clocked synchronous.
Maximum Number Of PDMA-16s In One Computer	Limited only by availability of expansion slots and/or DMA or Interrupt Levels for concurrent use.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector	On rear plate, 37-pin D-type male.
User Adjustments/Calibration	None required.
Board Size	9 in. (22.9 cm) long x 3.9 in. (9.9 cm) high full-size PC/AT connector.
Weight	6 oz. (170.1 g).
Operating Temperature Range	0 to 60° C.
Storage Temperature Range	-40 to 100° C.
Humidity	0 - 90% noncondensing.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX B

SUMMARY OF ERROR CODES

The following list contains Error Code definitions and suggested actions.

Error 0: No Error, OK.

Meaning:	Function executed without errors.
Action:	None.

Error 1: Driver NOT Initialized.

Meaning:	Mode 0 was not called to perform initialization.
Action:	Make a call to Mode 0.

Error 2: Mode Number < 0 or > 12 .

Meaning:	Illegal Mode number detected.
Action:	Specify Mode Numbers 0-12.

Error 3: Invalid Base Address, < 512 or > 1016 (< 200h or > 3F8h).

Meaning:	Board Base Address out of range.
Action:	Specify Address in the range of 512 to 1016 (200h to 3F8h).

Error 4: Board Hardware Error.

Meaning:	An attempt to write and read from the PDMA-16 failed.
Action:	Check the board's I/O address DIP switch. Consult manufacturer if problem persists.

Error 5: DMA Level Incorrect.

Meaning:	Incorrect DMA level specified.
Action:	Specify DMA Level as 0 0r 3 for Byte, 5-7 for Word.

Error 6: DMA Page Wrap-around Error.

Meaning:	Mode 1 detected a DMA page-wrap.
Action:	Use Mode 9 to allocate a Good DMA buffer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error 7: DMA Data Direction Not 0 or 1.

Meaning:	Mode 1 reports selected DMA direction to be invalid.
Action:	Direction should be (0) for Input or (1) for Output.

Error 8: Interrupt Level Out Of Range.

Meaning:	Mode 0 reports Interrupt Level out of range.
Action:	Select Interrupt Level 1 0r 3.

Error 9: Interrupt Source Out Of Range, < 0 or > 3 .

Meaning:	Mode 7 reports Interrupt Source out of range.
Action:	Select Interrupt source 2, 3, 4, 5, 6, or 7.

Error 10: Interrupt Slope Not 0 or 1.

Meaning:	Mode 7 reports incorrect interrupt slope.
Action:	Specify *0* for positive slope or *1* for negative slope.

Error 11: Auto-recycle Not 0 or 1.

Meaning:	Mode 1 reports Auto-recycle error.
Action:	Specify 1 for Auto-recycle ON, 0 for Auto-recycle OFF.

Error 12: Byte Or Word Specifier Not 0 or 1.

Meaning:	Mode 1 reports invalid Byte/Word specifier.
Action:	Specify 0 for DMA Byte transfers or 1 for DMA Word transfers.

Error 13: Digital Output Data Out Of Range, < 0 or > 255.

Meaning:	Mode 4 reports data out of range.
Action:	Specify Digital Output Data 0-255.

Error 14: Port Configuration Code Out Of Range, < 0 or > 2.

Meaning:	Mode 4 or 5 reports invalid port selection.
Action:	Specify 0 for Port A, 1 for Port B, or 2 for Ports A + B.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error 15: Configuration Change Data Not 0 Or 1.

Meaning:	Mode 5 detects port configuration change error.
Action:	Specify 0 for no change or 1 for change port configuration.

Error 16: Auxiliary Output Data Out Of Range.

Meaning:	Mode 6 reports invalid data.
Action:	Specified auxiliary data bit must be 0 or 1.

Error 17: Transfer Clock Source Data Illegal, Not 0 Or 1.

Meaning:	Mode 1 reports illegal clock source.
Action:	Specify 0 for external clock, 1 for internal timer.

Error 18: Word Transfers On ODD Boundary.

Meaning:	Mode 1 reports illegal Word Boundary.
Action:	Specify valid Word Count..

Error 19: Memory Allocation Error.

Meaning:	Mode 9 reports it cannot allocate a DMA buffer.
Action:	See Appendix D for more information.

Error 20: Word Count 0 or Negative From Mode 11.

Meaning:	Mode 11 reports a count of 0 or < 0.
Action:	Specify a count >0

Error 21: Packing Should Be 0 For Bytes or 1 For Words.

Meaning:	Mode 11 reports illegal packing selected.
Action:	Specify 0 for Byte or 1 for Word packing.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX C

UNDERSTANDING DMA

C.1 WHAT IS DMA?

Consider the problem of moving a large amount of data to or from an I/O device. This requirement is commonly encountered in the operation of many peripheral devices (for example disk drives) that are constantly moving large amounts of data into and out of memory. An obvious way of making the transfer would be a short program which for an input operation might read a byte from the I/O port into the accumulator (AX register) of the 8088 processor and then move the data from the AX register to a memory location to store it. In addition, we have to keep track of the memory locations where the data is going. By far the simplest way of handling this is to lay the data down in contiguous locations within a block of memory, using one of the processor's index registers to control the address. Each time a byte is transferred, the index register (usually the DI or Destination Index register) is incremented or decremented to point to the next location. A typical example assembly language program to do this might be as follows:

We could read each byte of data from memory and send it out to an I/O port using the main processor and a program loop similar to the example below.

```
SETUP:  MOV AX,SEGMENT      ;setup segment of memory for transfers
        MOV DS,AX
        MOV DI,OFFSET       ;setup start address within segment
        MOV CX,COUNT        ;setup number of bytes to transfer
        MOV DX,IOPORT       ;DX = I/O port address

READ:   IN AL,DX            ;read byte from I/O port      (8)
        MOV [DI],AL         ;store data                   (10)
        INC DI              ;increment index              (2)
        LOOP READ           ;continue reading until CX=0  (17)

CONT:   ........            ;yes, continue with program
```

The opposite of transferring data from memory to an I/O port is essentially similar. The numbers in parentheses following the READ: label are the number of processor clock cycles required to execute each instruction. On the original IBM PC, which has an 8088 processor operating at a clock frequency of 4.77MHz, you would find the loop takes 8 + 10 + 2 + 17 = 37 cycles or 7.8 microseconds to transfer each byte. If you were operating with a faster processor, such as a 12 MHz 80286 or 16 MHz 80386, and using exactly the code shown above, you would find transfer times are roughly inversely proportional to the clock rate (for example, a 16 MHz 80386 would take only 1.8 microseconds/byte). The 80286 and 80386 also include complex string I/O instructions (OUTSB & INSB) which are not available on the 8088. Using these instructions would be even more efficient. Note also the following:

1. The processor is tied up 100% of the time in transferring data; it cannot be used for any other function while the transfer is in progress without interrupting the transfer.
2. The rate of data transfer is controlled by the processor clock and may not correspond to the rate at which the I/O device wants to handle the data. This problem may be resolved by polling the I/O device to see if it is ready, or by having the I/O device generate a hardware interrupt. But both of these solutions add further code to the I/O routine, slowing down transfer rates even further.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3. If the processor has to handle a hardware interrupt from one device (the keyboard, COM: port, or system clock) while it is involved in handling a data transfer to another device, the delays involved may cause it to miss data, or at least will necessitate a discontinuity in the data flow.

It would be nice to have a way of transferring data without involving the processor, so as to free up as much as possible to attend to execution of the program. It would be an added advantage if we could speed up the transfer rate compared with the example programmed transfer above and also be able to control the rate easily. Since all we want to do is to move a byte or word directly to/from an I/O port from/to memory without any kind of intermediate processing on the way, it is better not to use the processor at all, but to provide specialized hardware that will accomplish this commonly required task and tackle it faster than the processor could do it. The process of passing data to/from an I/O device directly from/to memory is known as DMA (Direct Memory Access), and the hardware that controls this process is known as the DMA Controller, which in the case of the IBM PC/AT is handled by two 8237 DMA Controller chips on the system board.

C.2 THE MECHANICS OF A DMA TRANSFER

What happens when a device wants to transfer data to/from memory? The first step is for the device to send a signal known as the DMA REQUEST (DREQ for short) to the DMA Controller. The processor normally controls the computer's address and data buses as well as control signals such as the memory read/write (MEMR & MEWW) and I/O read/write (IOR & IOW) lines. To accomplish a DMA transfer, control of these lines must be temporarily relinquished to the DMA Controller. On receipt of the DREQ, the DMA Controller in turn issues a HOLD REQUEST to the processor. As soon as it can and when it has completed any part of an instruction in process that involves a bus cycle (I/O or memory access), the processor issues a HOLD ACKNOWLEDGE signal to the DMA Controller, and simultaneously disconnects itself from the address, data, and control buses. This process is "tri-stating," as the connections to the processor assume a third open-circuit state compared to their usual binary states of 1s and 0s or highs and lows. Although the processor is in a HOLD state, it has not necessarily stopped. During the hold state, the processor continues executing parts of instructions or instructions that do not involve any external I/O action (bus cycles); when it can no longer continue, it will insert wait states until the DMA Controller gives the bus back.

On receipt of the HOLD ACKNOWLEDGE, the DMA Controller begins its work. It releases its own connections to the address and control buses from their tri-state condition, asserting a valid memory address from an internal counter and then issuing a DMA ACKNOWLEDGE (DACK) signal to the I/O device followed by a simultaneous IOW and MEMR for a data output, or IOR and MEMW for input. The peripheral in turn responds to the DACK and IOR or IOW signals by placing or receiving data on the data bus, effectuating a transfer directly to/from memory. On completion of the MEMR/IOW or MEMW/IOR from the DMA Controller, the controller removes DACK, releases HOLD REQUEST, tri-states its own address and control lines, and increments or decrements its internal address counter to be ready for the next transfer. The processor in turn regains control of the buses, continuing execution of the next instruction. From the assertion of DREQ to completion of the cycle takes about 2.5 - 5 microseconds, depending on the length of the instruction that the processor happens to be engaged in on receipt of the DREQ. The actual amount of time between instructions that the processor loses the bus to the DMA Controller is even less, about 1.7 microseconds. The effect on program execution is minimal even when transferring data at very high rates which can exceed 500,000 bytes/sec on the PC/AT. To prevent the DMA Controller from "hogging" the buses if the DREQ is held constantly high, the Controller always allows the processor to perform at least part of an instruction between each DMA transfer, so that even operating "flat-out," DMA cannot grab much more than 30% of the bus bandwidth.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In order to perform DMA operations the peripheral must include hardware that generates the DREQ and responds to the DACK. The PDMA-16 includes this special hardware, and this is what distinguishes it from a simple digital I/O interface that does not, such as the manufacturer's Model PIO-12 (Note: The PDMA-16's DMA capability does not preclude it from being used as a standard I/O port for programmed I/O using IN's and OUT's). The PDMA-16 works in conjunction with the 8237 DMA Controller(s) which is a system component that is a part of the PC and PC/AT architecture and is essential to the operation of the PDMA-16.

It is important to appreciate that the DMA Controller sets the dynamics of the DMA transfer; nothing in the peripheral I/O device can alter the maximum data- handling speed of the controller. This fact leads to surprising side effects, in particular the IBM PC/AT which is generally three times faster than a standard PC or PC/XT is actually slower on DMA transfers because its DMA Controller clocks operate at 3MHz instead of the 4.77MHz on the PC. On the other hand it can also perform word (16 bit) transfers on its extended data bus as well as byte (8 bit) transfers on the PC-compatible section of its data bus. Since word transfers amount to two bytes of data, the overall transfer rate in bytes-per-second for word transfers is higher on the PC/AT than the PC despite the slower controller clock rate but correspondingly, the single byte transfer rate is slower. The PDMA-16 is designed to take advantage of the PC/AT's word transfer capabilities as well as operate in byte transfer mode when needed.

C.3 DMA STRUCTURE OF THE PC/AT

Although we have discussed the operation of a single device using DMA, it is customary to cater to the needs of several devices by providing several DMA channels, each one dedicated to a particular device. The 8237 provides four separate DMA channels known as Levels 0 thru 3. Correspondingly, there are four DMA request lines, DREQ0 - DREQ3, and four corresponding acknowledge lines, DACK0 - DACK3. The priorities of these lines are set according to two possible protocols set by a bit in the controller command register, either Fixed Priority (where lower DMA levels have higher priority than higher levels) or Rotating Priority (where each level takes a turn at having the highest priority). The PC BIOS sets the 8237 to operate in Fixed Priority mode upon power up, and it is generally inadvisable to change this as it may interfere with the correct operation of the computer or other peripheral devices that use DMA

The original PC and PC/XT design provided a single 8237 DMA Controller on the system board and the four DMA levels were allocated to system resources as follows:

DMA LEVEL	FUNCTION
0	Memory refresh
1	Unassigned, general I/O use
2	Floppy disk controller
3	Hard disk controller (if installed)

Apart from its uses for high-speed data transfer, the DMA Controller includes counter hardware that cycles through the memory addresses. Thus, as a byproduct of its design, the Controller can also be used to refresh dynamic memory, saving the cost of a separate memory refresh controller. This is what IBM chose to do in the original PC design using Level 0 to perform this function with its DREQ

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

being driven from Counter 1 of the internal 8253 timer at a 15 microsecond interval. On PC/AT memory refresh has been handled by additional hardware and does not involve the DMA Controller.

On PC/XTs, only one DMA channel is usually available for expansion use, Level 1. All levels are only capable of byte (8 bit) transfers, and the controller can make up to 65,536 transfers in one operation. The PC/AT design expanded the number of DMA channels by using two 8237 DMA Controllers (see Figure 3-1). Since one channel (Level 4) is used to cascade one controller into the other, and Level 0 is no longer used for memory-refresh functions, the AT I/O bus provides a total of seven channels, a considerable increase compared to the original PC bus, especially as six of them are uncommitted. The two controllers are hardware-wired so that Controller #1 accesses Address Lines A0 - A15 and can perform up to 65,536 byte transfers on Levels 0 - 3, and Controller #2 accesses Address Lines A1 - A16 (A0 is permanently set to zero) and can perform up to 65,536 word transfers (128 Kbytes) on even memory addresses on Levels 5 - 7. The DMA level assignments on the PC/AT are as follows:

DMA LEVEL	TYPE	FUNCTION
0	Byte	Unassigned, general I/O use
1	Byte	Unassigned
2	Byte	Floppy disk controller
3	Byte	Unassigned
4	-	Not available (internal cascade)
5	Word	Unassigned
6	Word	Unassigned
7	Word	Unassigned

The PC/AT's internal hardware structure limits byte transfers to Levels 0 - 3 and word transfers to Levels 5 - 7. It is important to understand that this minor restriction is imposed by the design of the PC/AT, not by the PDMA-16. If you wish, you may use DMA Level 2 both on the PC and PC/AT by "sharing" it with the floppy-disk controller. This entails either sequentially disabling the floppy controller and enabling the PDMA-16 or vice versa. Although possible, the additional programming involved, the requirement that transfers may only be made sequentially on a shared level, and the availability of so many other free levels seldom make the programming effort involved in using Level 2 worthwhile.

Note also that the PC/AT does not generally use DMA for servicing the hard disk controller. This is an evolutionary fluke and has resulted from maintaining downward compatibility of the PC/AT's DMA Controllers structure and speed with the original PC and PC/XT. In the PC the controller is clocked with the same 4.77 MHz clock used for the main processor. The PC/AT design which started out with a 6MHz 80286 processor, and presumably for design convenience, the controllers were driven at half the main processor clock rate or 3 MHz. This meant that transfers would be a little slower than on a PC and insured backward compatibility with all the earlier PC adapters and peripheral devices. As processor clock speeds have risen, 8 - 10 -12 - 16 MHz and now 20 - 25 MHz and higher speeds, most manufacturers have played safe by maintaining the original convention of clocking the DMA Controllers with a separate 3 MHz clock, although there are a few exceptions to this practice on some clones (consult your technical manual). The net result of this is that a transfer takes about 60% longer on a PC/AT than a PC. If you are moving a word on each transfer, the throughput

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

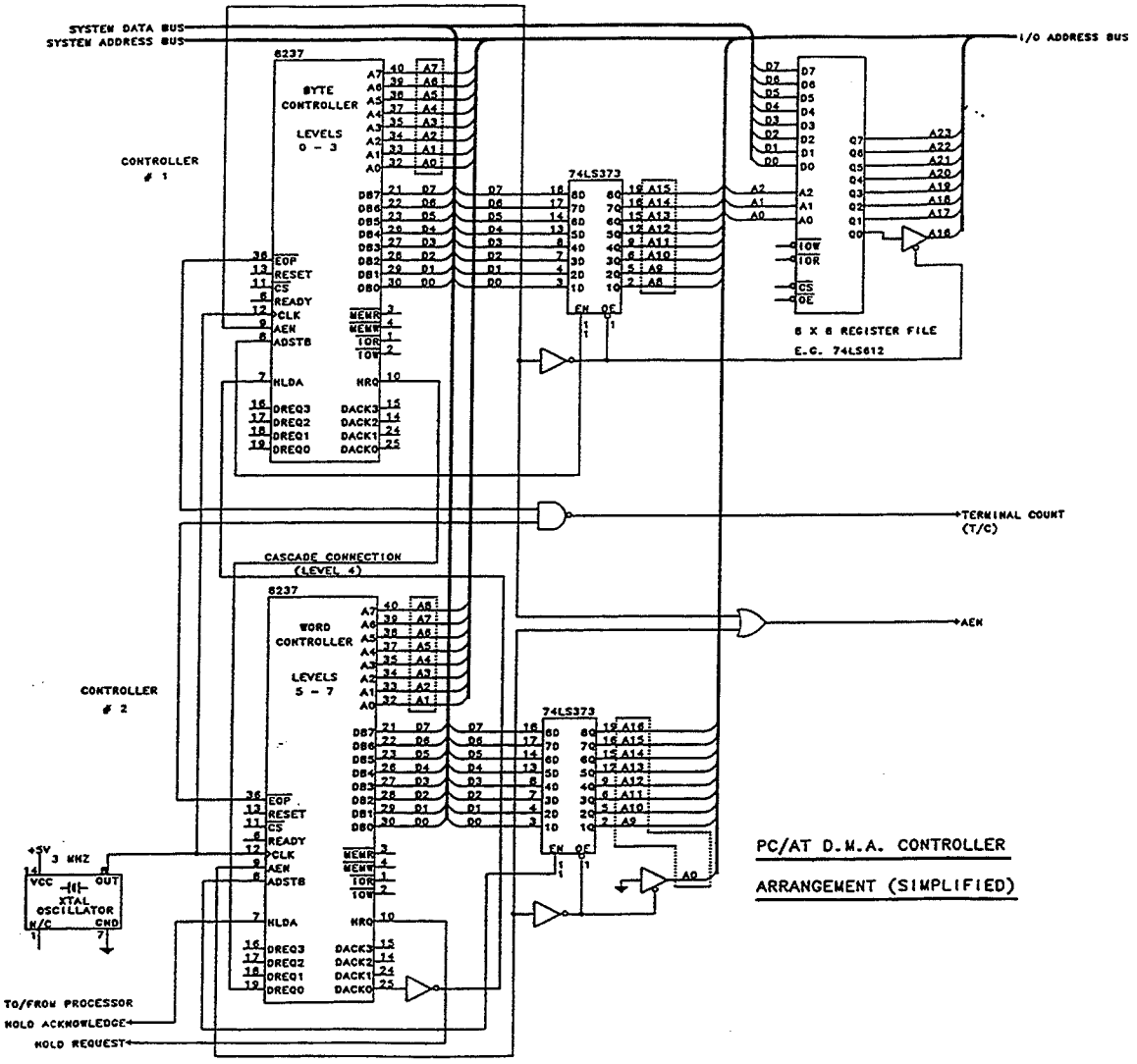


Figure 3-1. PC/AT DMA Controller Arrangement

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

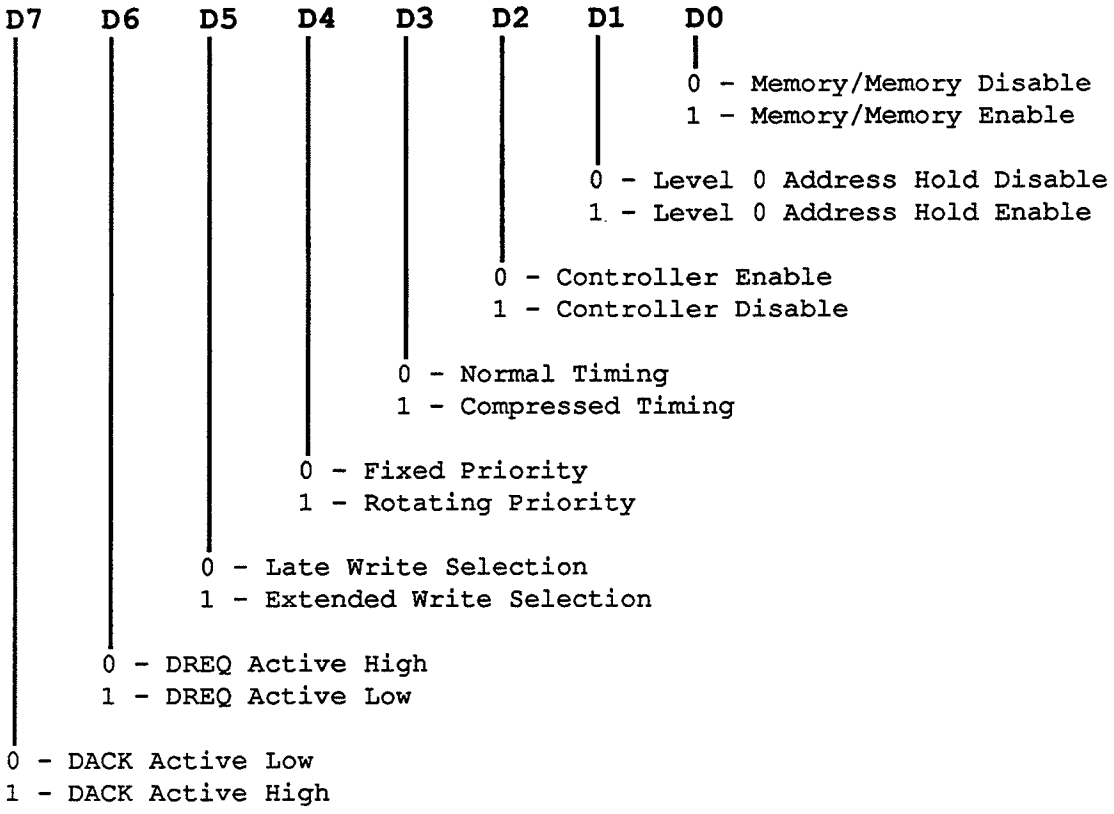
C.4 THE 8237 DMA CONTROLLER

This section provides a more detailed description of the Intel 8237 DMA Controller, used on the PC family. A full description is contained on the 8237 component data sheet available from Intel (Intel Corp., Lit. Dept., 3065 Bowers Avenue, Santa Clara, Ca. 95051).

Apart from the command register, the 8237 includes several other registers using a total of 16 I/O addresses. The 8237 interfaces internally with the PC/AT data bus as an 8-bit device, even though some of its internal registers are 16 bits wide. This means that all I/O must be made to the controllers using 8-bit I/O instructions. On the PC/AT, the word controller, Controller 2, is connected across address lines A1 - A16 (A0 is not connected) and its I/O registers appear at every other address on even boundaries (in effect, it uses 32 I/O addresses). The I/O addresses of the Byte Controller (#1) extend from 0 - Fh and the word controller (#2) from C0 - DFh. A description of the registers (identical in both controllers) is as follows:

COMMAND REGISTER - I/O Location 8H or D0H

This is an 8-bit write-only register that controls overall operation of the 8237 as follows:



BIOS initializes the command registers of both controllers with byte 00H (zero). Since many of the bits control signal polarity and timing features that must be observed for correct operation of the internal hardware, it is inadvisable to alter BIOS's initialization of the Command Register Byte after power up (unless you really know what you are doing!).

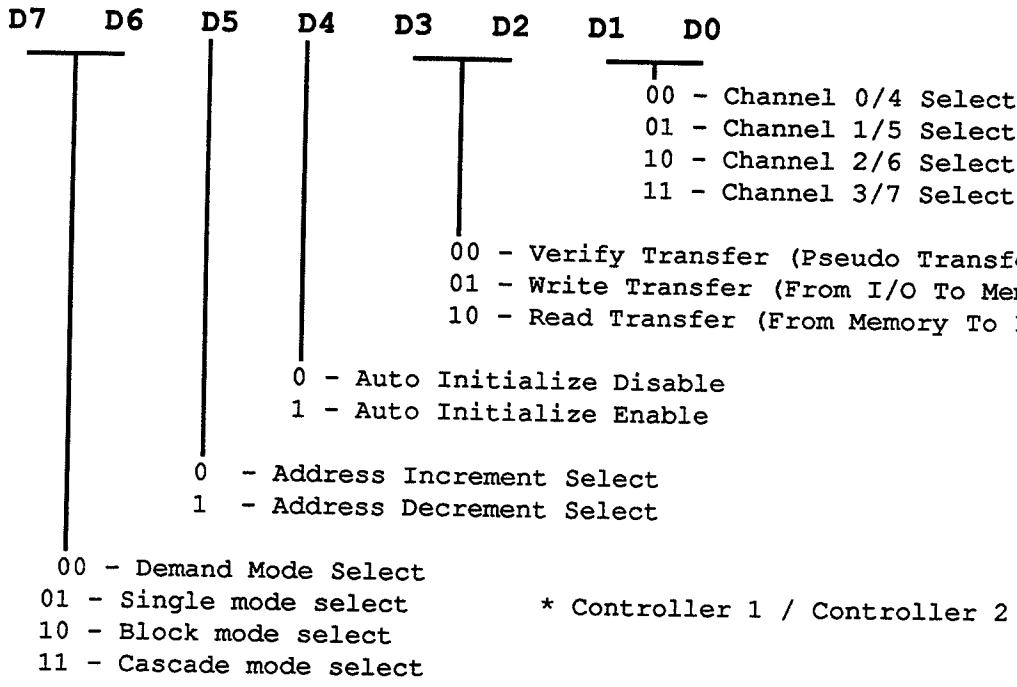
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Key features initialized by the BIOS are

1. Normal Timing, Late Write, and Fixed Priority are selected.
2. The DREQs are set active high, and the DACK's are set active low.

MODE REGISTER - Location 0BH or D6H

This is a group of four write-only, 8-bit registers, each one controlling the characteristics of each DMA channel and selected by the lower two bits of the byte. The bit functions are as follows:



A Mode Register is associated with each channel and unlike the Command Register, it can be safely programmed by the user at least for the unassigned levels. Some explanation of the bit functions follows:

D0 & D1 - The two lowest bits select the Mode Register for the channels being programmed. Make sure these are correct so that you do not inadvertently operate on the wrong channel.

D2 & D3 - Control the direction of transfer.

D4 - Controls auto-initialization (see below).

D5 - The Address Counter in the 8237 can count up or down from its starting value. This bit sets the count direction.

D6 & D7 - Control the type of DMA transfer (see below).

- 00 = Demand mode select
- 01 = Single mode select
- 10 = Block mode select
- 11 = Cascade mode select

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

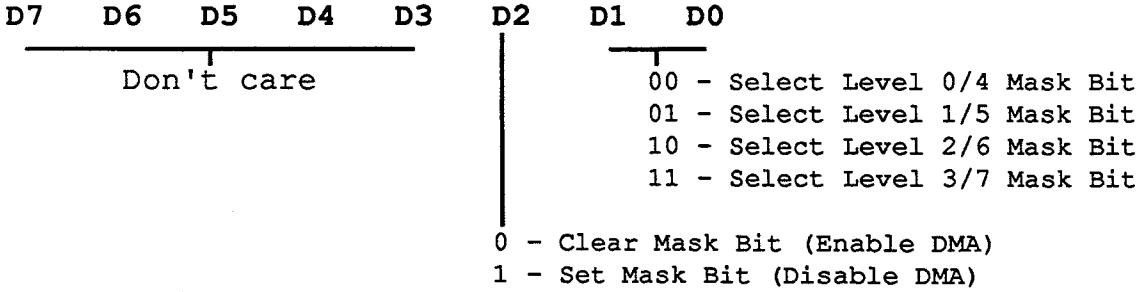
If Bit D4 = 0, a DMA transfer starts at the memory address specified in the 8237 Address Register and proceeds for the number of bytes specified in the Word Count Register. On reaching the specified word count, the Controller automatically sets the channel mask and any further DREQ's are ignored, terminating the DMA transfer. If D4 = 1, Auto-Initialize Mode is enabled. On reaching the final word count, the Controller re-initializes the Address Counter with its starting address, so that DMA proceeds continuously in this mode to a memory buffer area of location specified by the starting address, and length specified by the word count. On reaching the end of the buffer, transfers continue at the beginning of the buffer. In effect the buffer is circular in Auto-Initialize Mode. This can be useful when outputting data of a cyclic nature such as driving a D/A Converter to generate a periodic waveform or when streaming data to/from disk.

D6 and D7 control the DMA transfer modes. BIOS initializes D6 and D7 on every channel to select the Single Transfer Mode. In this mode, each DREQ will result in a single byte transfer of data followed by return of bus control to the main processor. The PDMA-16 has been designed for operation of the DMA Controller in this mode and this is the only valid selection. The Block and Demand Modes are almost similar to each other. In Block Mode, a single DREQ will make the DMA Controller perform multiple transfers, one after another until the final word count is reached. Once the DMA process becomes active, the DREQ can be taken low without affecting continuation of the DMA block transfer (that is, the DREQ is ignored once transfers start). Demand Mode is similar except that the transfers can be halted at any time during the block transfer by taking DREQ low. Transfers will continue from the point where they stopped on taking DREQ high. Both of these transfer modes can block the processor's use of the bus for extended times and may hold off service of other DMA channels while they are active. Since the use of Block and Demand Modes was not allowed for in the original PC/AT design, it is inadvisable to try using them as they may produce system problems. In any case the PDMA-16 has not been designed to support them. Cascade Mode allows connection of another 8237 Hold Request and Hold Acknowledge signals to the DREQ and DACK of the first DMA Controller. On the PC/AT, Level 4 of the primary controller is used for cascade expansion in this way and is programmed by the BIOS accordingly.

MASK REGISTER - I/O locations 0Ah/0Fh or D4h/DEh

Each channel has associated with it a Mask Bit that may be set to disable the incoming DREQ. A Mask Bit is also set when not in the Auto-Initialize Mode by the terminal count, T/C, produced at the end of the DMA transfer. The Mask Register, including all the Mask Bits for all the channels, may be written to as a whole through the port at I/O location 0Fh (Controller #1) or DEh (Controller #2). Apart from initializing the DMA Controller, which the BIOS performs on power up, it is best to avoid this method of setting and resetting Mask Bits, since you may disturb the Mask Bit on another channel inadvertently as the Mask Register is write-only and its current contents are impossible to read.

The I/O locations at address 0Ah (Controller #1) or D4h (Controller #2) provide a better way of setting and clearing a specific channel mask register bit without disturbing the mask bits of the other DMA channels. The protocol for doing this is as follows:



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Before performing any setup of the DMA Controller, set the Mask Bit of the level being programmed to disable any spurious DMA transfer that might otherwise occur.

ADDRESS REGISTERS - Location (2 x Level #) or C0h + (4 x Level (# - 4))

There are four 16-bit read/write address registers in each controller, one associated with each DMA level. The address register should be loaded with the offset of the starting address of the memory buffer area reserved for DMA. Since these are 16-bit registers, they are loaded by two sequential outputs to the same I/O port. An example (in BASIC) for Level 3 is as follows:

```
OUT 6, Low byte of address
OUT 6, High byte of address
```

If you are programming in Assembly Language, ***you must insert a delay between back to back I/O instructions when using 80286/386 processors***. This is to allow sufficient recovery time for the I/O bus (see IBM Technical Reference Manual). Failure to insert a delay will lead to erroneous reads and writes and erratic results. A short jump provides an adequate delay. Also, since the 8237 uses an internal byte pointer flip-flop to sequence the low byte / high byte reads and writes, it is important this is cleared prior to a read or write operation and is not altered by an interrupt during execution of your code. The byte pointer flip-flop is cleared by writing any data to I/O address 0Ch for Byte Controller #1 or D8h for Word Controller #2. The following Assembly Language example shows loading the Address Register of Level 5 including all the correct precautions and steps:

```
MOV AX,DATA          ;get address data to load in AX
CLI                  ;disable interrupts
OUT 0D8H,AL          ;clear byte pointer flip-flop
JMP $+2              ;I/O delay (recommended)
OUT 0C4H,AL          ;write low byte
MOV AL,AH            ;transfer high byte to AL
JMP $+2              ;I/O delay (required)
OUT 0C4H,AL          ;write high byte
STI                  ;re-enable interrupts
```

The address read back from the same location is the current address (the starting address plus or minus the number of DMA transfers, depending on whether increment or decrement is selected by the mode register).

TRANSFER COUNT REGISTERS - Location (2 x Level#)+1 or C0h+(4 x Level (#-4)+2)

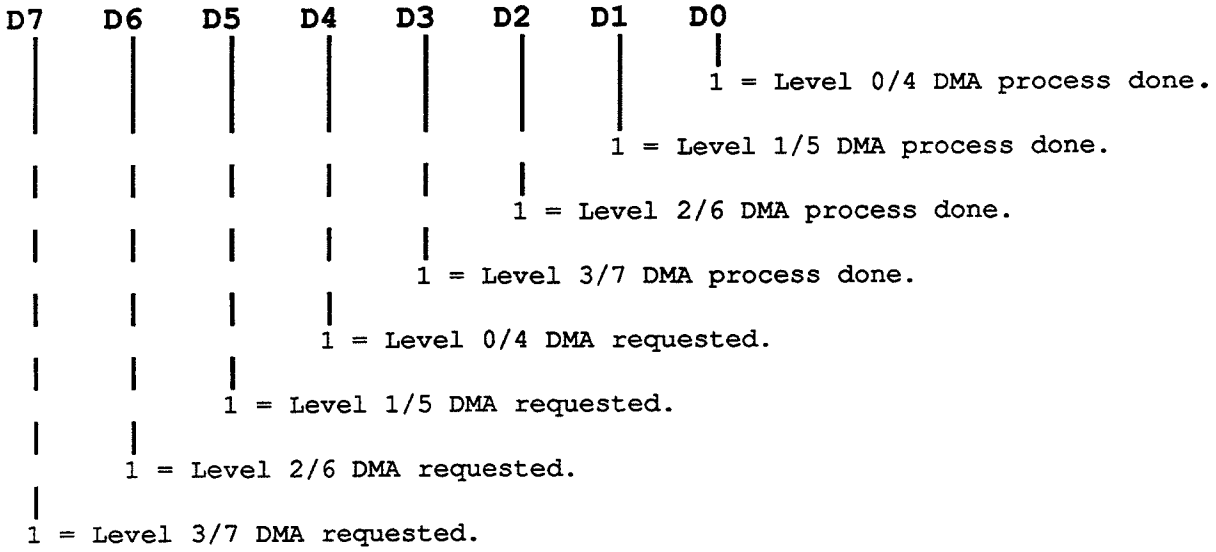
There are four 16-bit Transfer Count Registers (called *Word Count* on the Intel data sheet) that control the number of DMA transfers. These registers are loaded with the number of transfers required in an operation minus one. The registers decrement through zero and on reaching FFFFh generate a terminal count, T/C, on the expansion bus. In the non-auto initialize mode further DMA automatically masks off and further transfers cease, whereas in the Auto-Initialize Mode both the Transfer Count and Address Registers are automatically reloaded with their initial values and DMA proceeds continuously. The combination of the T/C (terminal count) I/O bus signal and the DACK for the selected DMA level can provide an interrupt to signal the end of the DMA transfer. The

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDMA-16 can generate this type of interrupt on any level if required. Other methods of determining if the DMA transfer has ended is to poll (read) the Controller Status Register (I/O address 8h Controller #1 or D0h Controller #2) or read the Transfer Count Register until it has reached FFFFh. This latter method also provides data on the number of DMA transfers that have taken place and is the method used by Mode 2 of the PDMA.BIN driver.

STATUS REGISTER - Location 8h or D0h

The DMA Controller Status Register provides information on whether a channel is active and whether a DMA transfer has finished, as follows:



BYTE POINTER FLIP-FLOP - Location 0Ch or D8h

When reading or writing any of the 16-bit internal registers of the 8237 (the Address or Transfer Count Registers) that involve double 8-bit sequential I/O operations to the same I/O address, the 8237 uses an internal sequencing flip-flop to select the low- and high-byte portions of the register. It is advisable to clear this flip-flop before one of these double-byte read or write operations so that data is sequenced in the correct order. Writing any data to I/O addresses 0Ch (Controller #1) or D8h (Controller #2) clears the byte pointer flip-flop.

C.5 THE DMA PAGE REGISTERS

The 8237 DMA Controller is an older peripheral device that was designed to work with a 16-bit address bus which these days is adequate only for controlling a small portion of the total addressable memory. The original PC used a 20-bit address bus for the 1 Mbyte of RAM, and for ROM memory allowed in the PC architecture, the PC/AT provides a 24-bit address bus to address a physical maximum of 16 Mbytes of memory. It is therefore only possible to perform DMA within a 16-bit addressable area of memory at a time. This area is termed a *Page* and for the Byte Controller amounts to 64Kbytes of memory; for the Word Controller, it is 128 Kbytes as A0, the least significant address byte, is always zero. The Page Address is provided by a set of 8-bit registers (4-bit on the original PC) external to the 8237 DMA Controllers. There is a register for each DMA level except Internal Level 4. The Byte Controller #1 provides bits A0 - A15 of the address and the Page Register provides the remaining address bits, A16 - A23. The Word Controller #2 provides bits A1 - A16 of the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

address, the Page Register provides bits A17 - A23; the Least Significant Bit of the Word Page Registers is ignored and the Least Significant Bit of the address bus (A0) is set to zero so that word transfers for Levels 5 - 7 may be made only on even address boundaries.

It is important to understand that although the Page Register supplements the DMA Controller to provide the full 24-bit address, there is no way that the address generating counter in the 8237 can propagate a carry/borrow into the Page Registers. This leads to an important programming precaution: always be sure that your transfer count plus your base memory address will not take you across a page boundary. If this situation occurs, since the page address does not increment, you will get a wrap-around to the beginning of the page which more often than not will overwrite some essential program memory and crash the computer!

The 8-bit Page Registers may be read/write or write only registers, depending on the type and design of your computer. The Page Register locations and functions are as follows:

I/O LOCATION	FUNCTION
87h	Level 0 Page Register
83h	Level 1 Page Register
81h	Level 2 Page Register
82h	Level 3 Page Register
8Bh	Level 5 Page Register
89h	Level 6 Page Register
8Ah	Level 7 Page Register

Note that there is no simple relationship between the Page Register I/O address and its associated DMA level; this just makes programming a little more of a challenge!

C.6 SETTING UP A DMA TRANSFER

Setup of a DMA transfer entails several steps which must be performed in the correct sequence. The process also requires a good understanding of the action of the hardware. Mode 1 of the PDMA.BIN driver performs this setup and avoids a lot of programming, the source listing of Mode 1 (see PDMA.ASM) provides an example of the Controller setup. If you do not want to use the driver and wish to write your own setup program you should proceed in the following sequence:

1. Disable any active DMA on the chosen level by setting the appropriate Mask Register Bit or alternatively wait for the transfer to finish.
2. Write to the appropriate mode register of the appropriate 8237 DMA Controller to set up the transfer characteristics.
3. Clear the byte pointer flip-flop and disable interrupts (clearing interrupts in a high level language may not be possible - you can take your chances or use the driver).
4. Load the Address and Byte Count Registers with appropriate data.
5. Re-enable interrupts.
6. Write the Page Register to select the DMA page.
7. If you intend to make use of a Terminal Count Interrupt to signal the end of DMA, then install your Interrupt Handler (set Interrupt Vector and 8259 Controller(s)).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8. Set the PDMA-16 to operate on the DMA and interrupt levels selected by writing the level numbers to the PDMA-16 DMA and Interrupt Level Select Registers and then enable DMA and interrupts from the PDMA-16 using its DMA and Interrupt Control Registers.
9. If you are using the PDMA-16's on-board timer to generate DMA transfer requests, it is a good idea to reload it so that the first DMA transfer will occur at a fixed delay from this point. An excellent technique is to load the timer with the largest division ratio possible which will produce a pulse every 7 minutes - in effect disabling the timer. The timer can later be set to its desired speed after all the DMA and interrupt setup is completed.
10. Enable 8259 Interrupt Controller(s) Mask Register (if using interrupts).
11. Last step: enable DMA Controller Mask Register. If you are using the on-board timer to pace DMA transfers, now is the point to bring it up to correct speed (see Step 9).

The main pitfall when doing the DMA setup is enabling the 8237 DMA Controller before DMA is enabled on the PDMA-16 through bit 7 of the DMA Control Register. Until the PDMA-16 is enabled, the DREQ line is tri-state or high impedance and picks up a lot of crosstalk from other lines on the expansion bus. If the 8237 Mask Register Bit is enabled, many spurious rapid DMA transfers will be generated, and it is not unusual to find that the 8237 signals that transfers are ended before the PDMA-16 is later enabled! Always enable the PDMA-16 first followed by enabling the 8237 controller.

Another detail not very obvious in the 8237 data sheet is that while a channel is masked off, it will store a DREQ. As soon as the Channel Mask Bit is cleared (enabled), this stored request gets serviced and you often get an undesired transfer immediately on enabling the controller. Since you usually do not know whether there is going to be a stored request pending, in some applications, you may have to enable the controller and reload the address and Transfer Count Registers with the desired values and then start up your DMA transfers. Generally if the DREQ bus line input has been left in an open-circuit or tristate condition, bus crosstalk will have induced enough noise into the line to generate a pending DMA request. This is usually the condition on making the first set of transfers with the PDMA-16.

C.7 DMA TRANSFER TIMING

On receipt of a DMA request (DREQ) from a peripheral, the 8237 DMA Controller immediately sends a HOLD request to the 80286/386 processor. The bus interface unit responds to the HOLD as soon as possible according to the following rules:

1. The response is close to immediate if no bus cycle (transfer to or from memory or I/O) is in progress. This time can be as little as 0.1 microseconds.
2. If a bus cycle is in progress, it will be completed before Hold Acknowledge is enabled. In this case, a processor-clock-speed-dependent delay of 200 - 600ns (nanoseconds) may take place. Memory wait states will add to this delay.
3. If a bus cycle involving a word (2 byte) transfer to an odd byte memory boundary is taking place, both bus cycles will be completed before Hold Acknowledge is enabled. In this case the delay can extend beyond a microsecond.
4. If the Hold Request occurs at the beginning of an Interrupt Acknowledge Sequence, then Hold Acknowledge can be delayed somewhat in excess of 13 processor clock periods, 1 - 3 microseconds.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5. If an instruction is executed with a LOCK prefix, the bus interface unit will insure that the whole following instruction will be executed before issuing a Hold Acknowledge. In the case of instructions that require several memory references during execution (ADD [BX],CX) this could add many microseconds of delay.

In general, Condition 5 is unlikely to arise since the LOCK prefix is used by programmers to insure total instruction execution in multiprocessor systems, and the PC/AT is a single-processor system. There is therefore no reason for a programmer to use the LOCK prefix in programs for the PC, though of course there is nothing to prevent its use.

Once the DMA transfer is under way, the DMA Controller will take five clock cycles or 1.7 microseconds (3MHz clock) to effect the transfer. Assuming that instructions are not LOCKed, the worst-case transfer timing would appear to be Condition 4 plus the DMA cycle time, around 4 microseconds. This neglects the effect of higher-level DMA channels which, if active and requesting service, can extend the delay. The technical term for this service delay is *DMA latency* .

A practical method of measuring the latency is to set the PDMA-16 up in Auto-initialize Mode with the DMA requests driven by the timer, which should be run at a pulse rate of a few KHz. You may synchronize a scope from the timer pulse output (Pin 22) the positive edge of which generates the DREQ. If you examine the XFER. ACK. (Pin 3), it goes low on the start or positive edge of the DREQ and rises back to a high state on completion of the DMA transfer at the end of DACK. The trailing edge will appear as a blur on the scope, and the blur extends anywhere from 2.5 to somewhat less than 5 microseconds from the start of the pulse. This corresponds to the best- and worst-case DMA transfer times. You will notice that the majority of transfers occur in 2 to 3 microseconds.

There are a number of conclusions that can be drawn from these numbers. If you are performing clocked DMA transfers (regulated by the timer), it will limit transfer speeds to the worst-case delay time corresponding to about 200,000 transfers-per-second. Note this applies whether you are transferring bytes (Levels 0 - 3) or words (Levels 5 - 7) so the corresponding data rates are 200,000 bytes/sec or 200,000 words/sec (400,000 bytes/sec.). If you are able to use the XFER. REQ. and XFER. ACK. to handshake with the peripheral and the peripheral can provide data as fast as it can be transferred, you will no longer be limited by the worst-case transfer time and you can expect speeds of up to 350,000 transfers/sec.

C.8 USING INTERRUPTS WITH DMA

The PDMA-16 can generate a hardware interrupt on any of the 11 PC/AT levels available on the bus. These are levels from 2-7. The interrupt can be generated from one of the following three sources:

1. An external positive or negative edge input on the INTERRUPT IN (Pin 1).
2. A periodic interrupt from the timer.
3. A terminal count (T/C) from the DMA Controller.

This provides several options in programming. The DMA Terminal Count Interrupt can be used to signal the end of a DMA transfer and start up another operation (another DMA transfer) or move data from the DMA buffer to be processed, etc. The external interrupt can be supplied by an external device that requires DMA service, or to signal that it has transferred all its data, etc. The periodic interrupt can be used to perform some repetitive operation like sending a block of data to a peripheral.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Since there are many possible actions that a user may require from an interrupt service routine, it is impossible to provide a ready-made, all-purpose routine. The PDMA.BIN driver does include a sample interrupt service handler (labelled INTH:) that can be installed and enabled using Mode 7 and disabled with Mode 8. This particular routine generates an audible "beep" every time it is invoked (run INT.BAS for a sample). A programmer can readily modify the handler to his own needs and reassemble the PDMA.ASM source. Note that most high-level languages, including BASIC, do not provide a means of writing an interrupt service routine, and it has to be done through the driver as described. Modes 7 and 8 provide examples of performing the set up of the PC/AT's 8259 interrupt controllers on any level which is fairly complex. One further precaution, when you write interrupt handlers: avoid using DOS calls within them, since most DOS calls are not re-entrant. Your program can be in the middle of using a DOS call when it is interrupted, and if your handler happens to use the same or similar DOS call, things get altered in strange ways and programs crash! This mistake can lead to difficult to diagnose, randomly appearing bugs, so be aware of it.

C.9 DETERMINING STATUS OF A DMA TRANSFER

Once a DMA transfer is under way, it is often necessary to find out how many transfers have taken place or whether the DMA transfer has completed. This can be determined by reading the appropriate byte count register of the 8237 DMA Controller. For example,

```
10 XL% = INP(7)    'low byte for level 3
20 XH% = INP(7)    'high byte for level 3
30 B = 256 * XH% + XL%  'B = number of transfers left to complete
```

Alternatively, Mode 2 of the PDMA.BIN driver will perform a similar function, returning the number of bytes or words transferred according to the setup of Mode 1.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX D

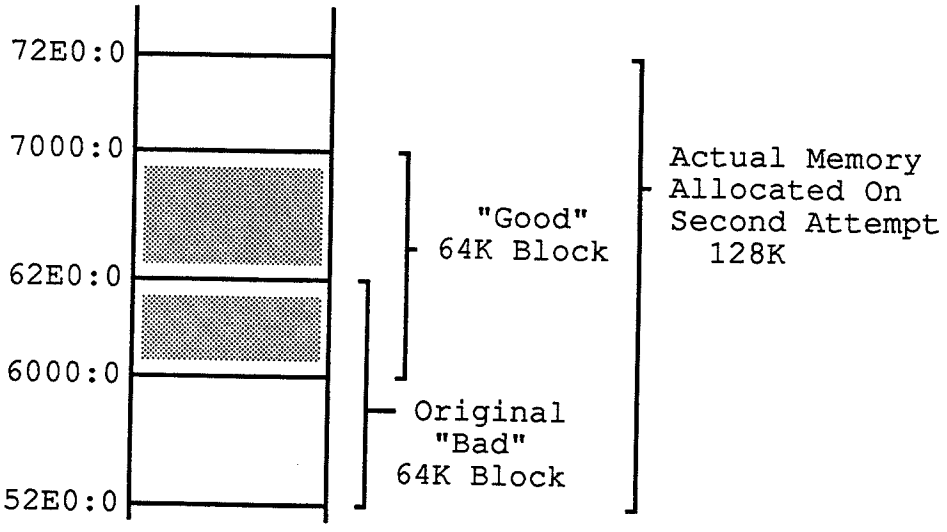
MODES 9 & 10: ALLOCATE/DEALLOCATE DMA BUFFERS

D.1 OVERVIEW

This appendix explains the method to be used to allocate a suitable block of memory for DMA transfers. The "memory allocation" modes (Modes 9 and 10) supplied with the software driver will work in the majority of programming languages including C, PASCAL, FORTRAN, and QuickBASIC that are compiled from the DOS command line. The same method used in the driver may work for other languages according to the following guidelines.

The first step for allocating memory suitable for DMA transfer is to allocate the number of bytes that are actually needed. In a DOS environment, all allocation and deallocation of memory is handled through the INTERRUPT 21 FUNCTION 48 (Allocate Memory Block) and FUNCTION 49 (Release Memory Block). After allocating this initial block of memory, a check is performed to see if the entire block falls in the same DMA PAGE. If the block is completely in one PAGE (i.e, the sum of the BYTE COUNT and the DMA OFFSET is less than or equal to FFFFh), then no further action is necessary and this block can be passed to the DMA MODE of the driver. If, however, the block of memory is not all on the same DMA PAGE, extra steps are necessary. The first of these steps is to release the previously allocated memory. Once this memory is released, twice as much memory as is actually needed is allocated. Upon successful completion of this second allocation a suitable block of memory has been obtained and all that is left to do is determine the OFFSET within this block to be sent to the DMA MODE.

NOTE: Although the SEGMENT Address of the allocated memory may not be the ADDRESS sent to the DMA MODE, it should be saved as it is needed when DMA is complete and the block is released back to DOS.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The preceding diagram illustrates the memory usage for this allocation scheme, using a desired byte count of 64 KB. The original allocated block (from 52E0:0 to 62E0:0) crossed a page boundary at 6000:0. After .the original block was released, twice as much was allocated, resulting in the actual block that starts at 52E0:0 and ends at 72E0:0. The actual memory block crosses two page boundaries (6000:0 and 7000:0) and there is now a complete page of memory accessible by the DMA controller. After storing the SEGMENT 52E0 for the purpose of releasing the memory when it is no longer needed, the segment 6000 can be sent to the DMA mode of the driver and 64KB can be transferred.

Using this method to allocate DMA buffers works for buffer sizes up to 64KB as long as there is a sufficiently large block of memory available in your system.

The following sections demonstrate how to set up for and make the calls to the appropriate memory allocation modes from different programming languages. When programming language specifics make other modifications necessary, they are included in the example code and/or description. Some programming languages are not compatible with the memory allocation modes for various reasons; alternative solutions are provided for these cases.

D.2 LANGUAGE SPECIFIC SOLUTIONS & EXAMPLES

The examples in this section use PDMA-16 MODE CALLS MODE 1, MODE 9, and MODE 10. These modes are described in Chapter 5.

BASIC(A)

From inside the interpreted versions of BASIC and BASICA, it is necessary to choose a memory location that is above the BASIC(A) workspace as a starting SEGMENT for DMA transfers. Using a debugger or a memory-mapping utility can help when deciding on a memory location. If you do not have one of these utilities and you do not have a large number of resident programs, use an address of &H5000 or above. Remember to choose a segment value that will allow the required number of bytes to be transferred in the same DMA PAGE (for example, &H5000, &H6000).

Example:

```
430  DMASEG = &H6000

440  '

450  '---- Setup and Perform DMA Transfer (Mode 1) ----------------
460  MD% = 1                              'Select mode
470  D%(0) = 5000                         'number of transfers
480  D%(1) = 0                            'don't care
490  D%(2) = 0                            'direction
500  D%(3) = 0                            'auto-recycle
510  D%(4) = 1                            'transfer source
520  D%(5) = DMASEG                       'SEGMENT address to transfer to
530  D%(6) = 0                            'DMA OFFSET
560  '
580  '

590  CALL PDMA (MD%, D%(0), FLAG%)        'set DMA running
600  IF FLAG% <> 0 THEN PRINT "Error in DMA setup #"; FLAG%: stop
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D.3 MICROSOFT QUICKBASIC *

When executing a QuickBASIC application from the command line, use Mode 9 to allocate memory for DMA. Use Mode 10 to deallocate the memory when DMA is finished and the memory is no longer needed.

When running an application from within the QuickBASIC environment, any call to Mode 9 will return an *INSUFFICIENT MEMORY* error because the QuickBASIC environment uses all available memory. The subroutine address in the second example allocates a suitable memory location for DMA transfer and returns the address in `aptr%`.

Example 1

For QuickBASIC stand-alone executable applications executed from the DOS command line:

```
'-----Allocate memory for DMA (Mode 9)------------------
' *** NOTE:  This mode can only be used from an executable file.
'
D%(0) = 5000
'Number of Transfers:
MD% = 9

CALL QBPDMA (MD%, VARPTR(D%(0)), FLAG%)  'Allocate memory
IF FLAG% <> 0 THEN PRINT "Error in Memory Allocate #"; FLAG% : STOP
ASEG = D%(7)
'***  This value must be used with
'*** Mode 10 to deallocate memory...
'-----Setup DMA transfer Mode 1 ----------------------
MD% = 1      'Select mode
D%(0) = 5000                  'Number of transfers (must be the same
                              'as allocated in Mode 9):
D%(1) = 0    'Don't care
D%(2) = 0    'Direction
D%(3) = 0    'Auto-Recycle
D%(4) = 1    'Transfer source
CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)  'Set DMA running
IF FLAG% <> 0 THEN PRINT "Error in DMA setup #"; FLAG% : STOP
.
.
.
'------Deallocate Memory Back to DOS (Mode 10) -------

D%(0) = ASEG          ; actual segment allocated using Mode 9
MD% = 10

CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)    'set timer
IF FLAG% <> THEN PRINT "Error in releasing Memory #" : FLAG% : STOP
```

Example 2

For QuickBASIC applications executed from within the QuickBASIC environment:

NOTE: This application may be done only if using Byte Transfers DMA Levels 0-3.

```
'Declare the Subroutine to Calculate the useable address for DMA
'NOTE:  All subroutine DECLAREs must be before any $DYNAMIC arrays!...
'DECLARE SUB ADDRESS (DMA() AS LONG, samp AS LONG, addr%)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
REM $DYNAMIC

DIM DMA(32766) AS LONG

DIM samp AS LONG
SAMP = 32766

CALL ADDRESS (DMA(), samp, aptr%)  'CALL SUB TO CALC. DMA ADDRESS
addr = &H300                       'PDMA-16 I/O address
'------Setup DMA transfer (Mode 1)-----------
MD% = 1                            'select mode
D%(0) = samp                       'Number of transfers
D%(1) = 1                          'Don't Care
D%(2) = 0                          'Direction
D%(3) = 0                          'Auto-Recycle
D%(4) = 1                          'Transfer Source
D%(5) = aptr%                      'DMA Offset
D%(6) = 0


CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)   'set DMA running
IF FLAG% <> 0 THEN PRINT "Error in DMA setup #"; FLAG% : STOP


SUB ADDRESS (DMA() AS LONG, samp AS LONG, addr%)
'
'   SUBROUTINE TO ALLOCATE SPACE FOR DMA TRANSFERS
'
DIM a AS LONG
DIM b AS LONG
   CLS
   s = VARSEG(DMA(0))              'GET SEGMENT ADDRESS
   o = VARPTR(DMA(0))              'GET SEGMENT OFFSET

   IF o < 0 THEN o = o + 65536
   IF s < 0 THEN s = s + 65536

   a = s * 16 + o
   PAGE = INT(a / 65536)           'See if there will be a DMA "WRAP-AROUND"
   b = a - (PAGE * 65536)
   b = b + samp * 2

   IF b < 65535 THEN
      a = (PAGE + 1) * 4096:       'here if PAGE WRAP would occur
      IF a > 32767 THEN
            addr% = a - 65536
      ELSE
            addr% = a
      ENDIF
   ELSE
      a = INT (a / 16)
      IF a > 32767 THEN
            addr% = a - 65536
      ELSE
            addr% = a
      ENDIF
   ENDIF
END SUB
```

D.4 MICROSOFT QUICK C OR BORLAND TURBO C

When executing a program from within Borland's TURBO C or Microsoft's Quick C, the environment uses a large portion of your PC's free memory. This memory usage can limit the amount of memory

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

available for DMA buffer space. If a "Memory Allocation Error" is returned after a call to the "Memory Allocation for DMA" Mode, you must take alternative action.

One option for limited memory is reducing the number of samples, thus reducing the size of the required memory block. If the number of samples is already at the minimum and there is still insufficient memory for allocation, another option may be necessary.

A second option is compiling and linking the program into a stand-alone executable file and executing it from the DOS command line. If you normally program within an environment and find that your application must be compiled and executed outside of the environment, the following may be useful:

1. Test the program from within the environment by taking fewer samples so as not to exhaust your PC's memory.
2. After successfully testing the program from within the environment, increase the number of samples as necessary, then compile and link the application into an executable file.

The next two sections provide examples for each of these languages.

Microsoft C Example

Using Version 4.x, 5.x, or 6.0 Microsoft C Compiler, use Mode 9 to allocate memory for DMA and Mode 10 to deallocate memory after DMA is finished and the memory is no longer available. For example,

```
/* Allocate Suitable Block of Memory for DMA */
mode = 9
/* allocate memory for DMA */
params[0] = 40000
/* allocate 40000 bytes/words */
mscm_pdma(&mode,params,&flag)
/* Microsoft C medium model Mode call */
if (flag != 0)
/* Error if flag is not zero */
{
   printf("\n\nMode 9 Error Flag = %d/n", flag);
   exit(1);
}

actual_seg = params[7]
/*Save actual segment for later deallocation with mode 10 */
dma_seg = params[5]
/* "good" segment for DMA Mode 1 */

/*Set Up for Call to Mode 1:  Set Up and Perform DMA Transfer */

mode = 1
/* Set up and Perform DMA */
params[0] = 40000
/* number of bytes */
params[1] = 0
/* value does not matter */
params[2] = 0
/* input */
params[3] = 0
/* auto-recycle off */
params[4] = 1
/*transfer clock source = timer */
params[5] = dma_seg
/*Transfer segment already set if Mode 1 called immediately after Mode 9 */
params[6] = 0
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/*DMA Offset: also set by Mode 9*/
mscm_pdma(&mode, params,&flag)
/* Microsoft C medium mode Mode call */
if (flag! = 0)
/* Error if flag is not is not zero */
{
  printf("\n\nMode 1 Error Flag = %d/n", flag);
  exit(1);
}
.
.
.
/* Release Memory Back to DOS when Finished */
mode = 10
/* Deallocate memory */
params[0] = actual_seg
/* segment to release */
mscm_pdma(&mode, params, &flag)
/* call driver */
if (flag !=0)
/* Error if flag is not zero */
{
  printf("\n\nMode 10 Error Flag = %d/n", flag);
  exit(1);
}
```

BORLAND TURBO C Example*

```
/* Allocate Suitable Block of Memory for DMA */
mode = 9
/* allocate memory for DMA */
params[0] = 40000
/* allocate 40000 bytes/words */
tcm_pdma(&mode,params,&flag)
/* TURBO C medium model Mode call */
if (flag != 0)
/* Error if flag is not zero */
{
  printf("\n\nMode 9 Error Flag = %d/n", flag);
  exit(1);
}

actual_seg = params[7]
/*Save actual segment for later deallocation with mode 10 */
dma_seg = params[5]
/* "good" segment for DMA Mode 1 */

/*Set Up for Call to Mode 1:  Set Up and Perform DMA Transfer */

mode = 1
/* Set up and Perform DMA */
params[0] = 40000
/* number of bytes */
params[1] = 0
/* value does not matter */
params[2] = 0
/* input */
params[3] = 0
/* auto-recycle off */
params[4] = 1
/*transfer clock source = timer */
params[5] = dma_seg
/*Transfer segment already set if Mode 1 called immediately after Mode 9 */
params[6] = 0
/*DMA Offset: also set by Mode 9*/
tcm_pdma(&mode, params,&flag)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/* TURBO C medium mode Mode call */
if (flag! = 0)
/* Error if flag is not is not zero */
{
   printf("\n\nMode 1 Error Flag = %d/n", flag);
   exit(1);
}
.
.
.
/* Release Memory Back to DOS when Finished */
mode = 10
/* Deallocate memory */
params[0] = actual_seg
/* segment to release */
tcm_pdma(&mode, params, &flag)
/* call driver */
if (flag !=0)
/* Error if flag is not zero */
{
   printf("\n\nMode 10 Error Flag = %d/n", flag);
   exit(1);
}
```

D.5 TURBO PASCAL

Using TURBO PASCAL Compiler Versions up to and including 6.0, work with Mode 9 to allocate memory for DMA and Mode 10 to deallocate the memory, when DMA is finished and the memory is no longer needed.

TURBO PASCAL allows the user to specify the memory requirements for a program with the "Memory Allocation Sizes" parameter directive $M. The $M directive allows specification of the stacksize and the minimum-and-maximum heap size using the syntax `{$M stacksize, heapmin, heapmax}`.

When specifying a minimum and maximum heapsize for a program that calls the Allocation Mode within the software driver, it must be insured that TURBO PASCAL does not use all available memory. If the maximum heapsize is not set by the user, TURBO PASCAL uses all of the remaining memory for its internal heap. The Memory Allocation mode of the software driver can acquire memory only from DOS, if the $M directive is used to force TURBO PASCAL not to use up all available memory.

The limits for stacksize, heapmin, and heapmax are as follows:

PARAMETER	MINIMUM	MAXIMUM
stacksize	1024	65520
heapmin	0	655360
heapmax	heapmin	655360

For further information regarding the "Memory Allocation Sizes" Parameter Directive, refer to the TURBO PASCAL manual.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

```
{$M 16384,0,131072}
.
.
.
BEGIN
  (* Allocate DMA Buffer Using Mode 9 *)
  mode := 9;
(* allocate memory for DMA *)
  params[0] := 40000;
(* allocate 40000 bytes/words *)
  tp_pdma(&mode, params,&flag);
(* TURBO PASCAL mode call *)
  if (flag <> 0) then
(* Error if flag is not zero *)
  begin
   write("Mode 9 Error Flag = ');
   writeln(flag);
   halt;



(* Halt program execution *)
  end;
  dma_seg := params[6];
  actual_seg := params[7];

(* save DMA "good" segment *)
(* save actual segment for later *)
  (* Set up and perform DMA transfer using Mode 1 *)
  mode := 1;
(* Set Up and Perform DMA *)
  params[0] := 40000;
(* number of bytes *)
  params[1] := 0;
(* value does not matter *)
  params[2] := 0;
(* input *)
  params[3] := 0;
(* auto-recycle off *)
  params[4] := 1;
(* transfer clock source = timer *)
  params[5] := dma_seg;
(* Transfer segment: already set *)
(* if Mode 1 called immediately  *)
(* after Mode 9 *)
  params[6] := 0;
(* DMA Offset: also set by Mode 9 *)

  tp_pdma(&mode,params,&flag);   (* TURBO PASCAL mode call *)
  if (flag <> 0) then
  begin
   write ('Mode 1 Error Flag = ');
   writeln(flag);
   halt           (* Exit to DOS *)
  end;
.
.
.
  (* Releases Memory back to DOS using Mode 10 *)
  mode := 10;
(* Deallocate memory *)
  params[0] := actual_seg;
(* Segment to release *)
  tp_pdma(&mode, params, &flag);
(* Call driver *)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
    if (flag <> 0) then
    begin
     write ("Mode 10 Error Flag = ');
     writeln(flag);
     halt           (* Exit to DOS *)
    end;
    .
    .
    .
END.
```

D.6 MICROSOFT PASCAL

Using Microsoft PASCAL Compiler Versions up to and including 4.0, work with Mode 9 to allocate memory for DMA and with Mode 10 to deallocate the memory, when DMA is finished and the memory is no longer needed.

Example

```
  mode := 9;
(* allocate memory for DMA *)
  params[0] := 40000;
(* allocate 40000 bytes/words *)
  msp_pdma(&mode, params,&flag);
(* MICROSOFT PASCAL mode call *)
  if (flag <> 0) then
(* Error if flag is not zero *)
  begin
   write("Mode 9 Error Flag = ');
   writeln(flag);
   abort(0)


(* Exit to DOS *)
  end;
  dma_seg := params[6];
  actual_seg := params[7];

(* save DMA "good" segment *)
(* save actual segment for later *)
(* deallocation with Mode 10 *)
  (* Set up for call to Mode 1:  Setup and Perform DMA transfer *)
  mode := 1;
(* Set Up and Perform DMA *)
  params[0] := 40000;
(* number of bytes *)
  params[1] := 0;
(* value does not matter *)
  params[2] := 0;
(* input *)
  params[3] := 0;
(* auto-recycle off *)
  params[4] := 1;
(* transfer clock source = timer *)
  params[5] := dma_seg;
(* Transfer segment: already set *)
(* if Mode 1 called immediately  *)
(* after Mode 9 *)
  params[6] := 0;
(* DMA Offset: also set by Mode 9 *)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
    msp_pdma(&mode,params,&flag);   (* MICROSOFT PASCAL mode call *)
    begin
     write ('Mode 1 Error Flag = ');
     writeln(flag);
     abort(0)           (* Exit to DOS *)
    end;
  .
  .
  .
    (* Releases Memory back to DOS using Mode 10 *)
    mode := 10;
  (* Deallocate memory *)
    params[0] := actual_seg;
  (* Segment to release *)
    msp_pdma(&mode, params, &flag);
  (* Call driver *)
    begin
     write ("Mode 10 Error Flag = ');
     writeln(flag);
     abort(0)         (* Exit to DOS *)
    end;
    .
    .
    .
  END.
```

D.7 MICROSOFT FORTRAN

Using Microsoft FORTRAN Compiler Versions up to and including 5.0, work with Mode 9 to allocate memory for DMA and with Mode 10 to deallocate the memory, when DMA is finished and the memory is no longer needed.

Example

```
        mode = 9
        i(1) = 40000
C       Allocate DMA buffer using mode 9

        call msf_pdma(mode, i(1), Flag)
        if (flag .NE. 0) then
        print *, 'Mode=',mode,'    Error#',flag
        goto 35
        endif
        dmaseg = i(7)
        actseg = i(8)

C       Set Up and Perform DMA Transfers
        mode = 1
        i(1) = 40000
        i(2) = 0
        i(3) = 0
        i(4) = 0
        i(5) = 1
        i(6) = dmaseg
        i(7) = 0
        flag = 0

        call msf_pdma(mode, i(1), Flag)

        if (flag .NE. 0) then
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
      print *,'Mode = ',mode, '     Error #', flag
      goto 35
      endif

C     Free DMA buffer using Mode 10
      mode = 10
      flag = 0
      i(1) = actseg

      call msf_pdma(mode, i(1), Flag)

      if (flag .NE. 0) then
      print *, 'Mode = ', mode,'     Error #', flag
      goto 35
      endif

35    end
```

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STORAGE OF INTEGER VARIABLES

STORAGE OF INTEGER VARIABLES

Data is stored in integer variables (% type) in 2's complement form. Each integer variable uses 16 bits or 2 bytes of memory. 16 bits of data is equivalent to values from 0 to 65,535 decimal, but the 2's complement convention interprets the most significant bit as a sign bit so the actual range becomes -32,768 to +32,767 (a span of 65,535) as shown below:

	HIGH BYTE								LOW BYTE							
	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
+32,768	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
+10,000	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0
+1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-10,000	1	1	0	1	1	0	0	0	1	1	1	1	0	0	0	0
-32,767	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

^

Sign bit (1 if negative, 0 if positive)

Integer variables are the most compact form of storage for data from the 12-bit A/D converter and 16-bit data from the 8253 interval timer. To conserve memory and disk space as well as to optimize execution speed, all data exchange via the CALL is through integer-type variables. This may pose a programming problem when handling unsigned numbers in the range 32,768 to 65,535.

Unsigned integers greater than 32,767 require a signed 2's-compliment format. For example, assume we want to load a 16-bit counter with 50,000 decimal. An easy way of turning this to binary is to enter BASIC and execute PRINT HEX$(50000). This returns C350 or binary 1100 0011 0101 0000. Since the most significant bit is 1, it would be stored as a negative integer and, in fact, the correct integer variable value would be 50,000 - 65,536 = -15,536. The programming steps for switching between integer and real variables for representation of unsigned numbers between 0 and 65,535 is therefore:

From real variable N (0 <= N <= 65,535) to integer variable N%:

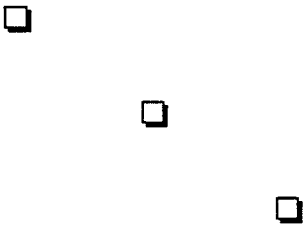
```
xxx10  IF N<=32767 THEN N% = N ELSE N% = N - 65536
```

From integer variable N% to real variable N:

```
xxx20  IF N% >= 0 THEN N=N% ELSE N = N% + 65536
```

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDMA-16 PCF

Instructions

for the

PDMA-16 PCF

Callable Driver

Contents

SECTION F1 INTRODUCTION

F1.1 Overview F-3
F1.2 Supported Languages F-3
F1.3 Copying Distribution Software F-3
F1.4 Writing Your Program F-4
F1.5 This Manual F-5

SECTION F2 DRIVER INFORMATION

F2.1 Overview F-7
F2.2 Driver Source Modules F-7
F2.3 Drivers F-7
F2.4 Mode Calls F-8
F2.5 Calling The Driver F-8
F2.6 Creating New Drivers F-10

SECTION F3 DRIVER USAGE

F3.1 Overview F-13
f3.2 Microsoft C/Turbo C F-13
F3.3 Microsoft PASCAL F-15
F3.4 Borland TURBO PASCAL F-17
F3.5 Microsoft FORTRAN F-19
F3.6 Microsoft QuickBASIC F-20

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SECTION **F1**

INTRODUCTION

F1.1 OVERVIEW

The PDMA-16 is a software package for programmers using Pascal, C, FORTRAN, and QuickBASIC to write data acquisition and control routines (referred to herein as *Application Code*) for the PDMA-16. The Distribution Software for this package is normally supplied on 5.25" low-density diskettes but is also available (upon request) on 3.5" diskette(s). Contents of the package include the following:

- PDMA-16 Drivers for each of the supported languages
- Driver Source Modules for creating new Drivers
- Miscellaneous documentation (.DOC) files
- Example program files in all supported languages

F1.2 SUPPORTED LANGUAGES

The PDMA-16 supports all memory modules of the following languages:

- Microsoft C (V4.0 - 6.0)
- Microsoft Quick C (V1.0 - 2.0)
- Microsoft Pascal (V3.0 - 4.0)
- Microsoft FORTRAN (V4.0, 4.1)
- Microsoft QuickBASIC (V4.0 and higher)
- Borland Turbo Pascal (V3.0 - 5.0)
- Borland Turbo C (V1.0 - 2.0)
- GW, COMPAQ, and IBM BASIC (V2.0 and higher)

F1.3 COPYING DISTRIBUTION SOFTWARE

As soon as possible, make a working copy of your Distribution Software. You may put the working copy on diskettes or on the PC Hard Drive. In either case, making a working copy allows you to store your original software in a safe place as a backup.

To make a working copy of your Distribution Software, you will use the DOS COPY or DISKCOPY function according to one of the instructions in the following two subsections.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To Copy Distribution Software To Another Diskette

Note that the *source* diskette is the diskette containing your Distribution Software; the *target* diskette is the diskette you copy to. Before you start, be sure to have one (or more,as needed) formatted diskettes on hand to serve as target diskettes.

First, place your Distribution Software diskette in your PC's A Drive and log to that drive by typing **A:** . Then, use one of the following instructions to copy the diskette files.

- If your PC has just one diskette drive (Drive A), type **COPY *.* B:** (in a single-drive PC, Drive A also serves as Drive B) and follow the instructions on the screen.

  If you prefer to use the DOS *DISKCOPY* function, instead of *COPY* , you will type **DISKCOPY A: A:** and follow instructions on the screen. This alternative is faster, but requires access to DISKCOPY.COM, in your DOS files.

- If your PC has two diskette drives (Drive A and Drive B), type **COPY *.* B:** (the same as above) and follow the instructions on the screen.

  If you prefer to use the DOS *DISKCOPY* function, instead of *COPY* , you will type **DISKCOPY A: B:** and follow instructions on the screen. This alternative is faster, but requires access to DISKCOPY.COM, in your DOS files.

To Copy Distribution Software To The PC Hard Drive

Before copying Distribution Software to a hard drive, make a directory on the hard drive to contain the files. While the directory name is your choice, the following instructions use *PDMA16* .

1. After making a directory named *PDMA16* , place your Distribution Software diskette in your PC's A Drive and log to that drive by typing **A:** .
2. Then, type **COPY *.*** *path***PDMA16** , where *path* is the drive designation and DOS path (if needed) to the PDMA16 directory.

When you finish copying your Distribution Software, store it in a safe place (away from heat, humidty, and dust) for possible future use as a backup.

F1.4 GENERATING AN APPLICATION PROGRAM

In the Distribution Software, the example program for the language you are using provides most of the information you need to start your own PDMA-16-based Application Program. The overall procedure for a typical executable program, however, is as follows:

1. Write your Application Code using a text editor or the language environment.
2. Compile your program.
3. Link the compiled program to a Driver (from the Distribution Software) suited to the language of your Application Code.

This procedure gives you an executable Application Program, ready to test. Repeat all three steps as you modify/fix this program.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

F1.5 THIS MANUAL

Chapter 1 of this manual is introductory material.

Chapter 2 presents information on the PDMA-16 Drivers required for the supported languages. Since the Drivers support the full series of PDMA-16 Mode Calls, Chapter 2 also lists and briefly describes the Mode Calls. And since the Drivers may not be perfectly suited to your particular applications, Chapter 2 discusses the Driver Source Modules, which are the source-code files you may use for creating new Drivers. Finally, the chapter includes instructions for creating new Drivers.

Chapter 3 presents brief instructions and examples for using the Drivers with your Application Programs.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SECTION F2

DRIVER INFORMATION

F2.1 OVERVIEW

When you write a program for your own PDMA-16 application, your program is referred to herein as the *Application Code*. You have a choice of writing this Code in BASIC, QuickBASIC, PASCAL, Turbo PASCAL, C, or FORTRAN. You then compile your Application Code and link the resulting program with a *Driver*. The linking process develops the *Application Program*, which is the program giving you software control of your hardware.

The Driver you link with your Application Code must be suited to the language used for the Code. For example, if you write your Application Code in C, your must link it with a Driver suited to C.

The Distribution Software contains Drivers for BASIC, QuickBASIC, PASCAL, Turbo PASCAL, C, and FORTRAN. The Distribution Software also contains the *Driver Source Modules*, which are the Assembly Language source files provided for the purpose of allowing you to create new Drivers customized to your particular needs.

Section 2.2 of this chapter lists and describes the Driver Source Modules, with which you may create new Drivers. Section 2.3 lists and describes the Drivers available in the Distribution Software. Section 2.4 lists the Mode Calls supported by the Drivers. Section 2.5 instructs you on how to make calls from your Application Code. The final section (Section 2.6) instructs you on how to use the Driver Source Modules to create new Drivers.

F2.2 DRIVER SOURCE MODULES

The following two Driver Source Modules are the essential building blocks for creating a PDMA-16 Driver in any language:

PDMA.ASM	Core of the driver.
PDMAIFC.ASM	Driver interface module for PASCAL, C, FORTRAN, and QuickBASIC.

As mentioned earlier, these two modules are available in your Distribution Software.

For instructions on using these modules to create Drivers, refer to Section 2.6.

F2.3 DRIVERS

As a convenience, your Distribution Software contains Drivers for PASCAL, Turbo PASCAL, C, FORTRAN, BASIC, and QuickBASIC. You must link the appropriate Driver with your Application

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Code; choose the Driver that matches the language used for your Application Code. Available Drivers are as follows:

PDMAPCF.LIB:	Driver for Pascal, C, FORTRAN, and stand-alone QuickBASIC programs.
PDMA.BIN:	Driver for BASIC(A).
PDMAQB45.QLB:	Driver for the QuickBASIC Integrated Development Environment (Ver. 4.0-4.5).
PDMAQB45.LIB:	Driver for the QuickBASIC Integrated Development Environment (Ver. 4.0-4.5) stand-alone programs.
PDMAQBX.QLB:	Driver for the QuickBASIC Extended Environment (Ver. 7.0).
PDMAQBX.LIB:	Driver for the QuickBASIC Extended Environment (Ver. 7.0) stand-alone programs.
TPPDMA.OBJ:	Driver for TURBO Pascal.

F2.4 MODE CALLS

This list briefly describes the Mode Calls supported by the PDMA-16 driver software. More detailed explanations of each Mode are available in the main text of the PDMA-16 User Guide.

MODE 0	Initialize Driver and Test Hardware.
MODE 1	Setup and Perform DMA Transfer.
MODE 2	Return Status of DMA Transfer.
MODE 3	Set Timer Rate.
MODE 4	Digital Output.
MODE 5	Digital Input.
MODE 6	Auxiliary Output.
MODE 7	Set-up and Enable Interrupt.
MODE 8	Disable Interrupt.
MODE 9	Allocate Memory for DMA.
MODE 10	Deallocate Memory Segment.
MODE 11	Move Data from Source to Destination.
MODE 12	Disable DMA.

Refer to the PDMA-16 User Guide for details of each Mode. It is essential that you perform a channel initialization (MODE 1) on each channel separately before selecting any other Mode (0-14).

F2.5 CALLING THE DRIVER

In your Application Code, you write a call the PDMA-16 driver through a single label that corresponds to the language used for your Code and to the memory model used for compiling. These labels are the *Call Labels* . PDMA-16 Call Labels and their corresponding Drivers are as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDMAPCF.LIB:

mscs_pdma	For Calls from Microsoft C, Small Model
mscm_pdma	For Calls from Microsoft C, Medium Model
mscl_pdma	For Calls from Microsoft C, Large Model
tcs_pdma	For Calls from TURBO C, Small Model
tcm_pdma	For Calls from TURBO C, Medium Model
tcl_pdma	For Calls from TURBO C, Large Model
msp_pdma	For Calls from Microsoft Pascal
msf_pdma	For Calls from Microsoft FORTRAN
qbpdma	For Calls from Microsoft QuickBASIC

TPPDMA.OBJ:

tp_pdma	For Calls from TURBO Pascal

PDMA.BIN:

pdma	For Calls from BASIC(A)

Regardless of the language/model you are using, with each call to a label you must specify three input parameters, as follows:

MODE — A 16-bit integer containing the number of the mode to be executed by the PDMA-16 driver.

PARAM — An array of 16-bit integers containing a variable number of mode-dependent arguments required for the successful execution of the mode.

FLAG — A 16-bit integer quantity that contains a number representing any error code reported by the PDMA-16 driver. (See Chapter 4 for error-code definitions.)

The following is code fragment (in C) on how to declare and use the call parameters.

```
int     Mode;
int     Flag;
int     Params[9];
   :
Mode = 0;
Flag = 0;
Params[0] = 0x300;        /* Card Base Address */
Params[1] = 1;            /* Selected DMA Level */
Params[2] = 7;            /* Selected Interrupt Level */

mscl_pdma(&Mode, Params, &Flag);
if (Flag != 0)
  printf ("**** Error %d detected in mode 0", Flag);
   :
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to Chapter 3 for additional details on how to declare and use these variables in other languages.

F2.6 CREATING NEW DRIVERS

General

While the Drivers available to you in the Distribution Software (see Section 2.3) support all the Call Modes described in Section 2.4, they may not suit your particular application. You may remedy this problem by creating a new version of the desired Driver. This section provides the information necessary to create a new Driver for BASIC, QuickBASIC, PASCAL, Turbo PASCAL, C, and FORTRAN.

Note that to create a new version of a Driver, your working directory (generally, the directory containing the Distribution Software) must contain the Driver Source Modules (Section 2.2) and the following development tools:

MASM.EXE	Microsoft Assembler
LINK.EXE	Microsoft Linker
LIB.EXE	Microsoft Librarian

Other utilities will be specified as needed in the instructions of the subsections that follow.

Also, note that in the MASM compile commands you use to create a new Driver, you must define the two symbols *BIN* and *QBASIC* . These definitions use the */D* option for BASIC, QuickBASIC, PASCAL, C, and FORTRAN. For Turbo PASCAL, only the symbol */DTURBOPAS* requires definition. These symbol definitions are as follows:

BIN = 1: Compile for BASIC(A) Driver. Usage example: **`/DBIN=1`** .

BIN = 0: Compile for non-BASIC(A) Driver (PASCAL, C, FORTRAN, and QuickBASIC). Usage example: **`/DBIN=0`** .

WARNING

The manufacturer does not provide technical support for user modications of the driver source code.

The PDMA.BIN Driver For BASIC(A)

To create a PDMA.BIN Driver, you must have access to the following utilities:

EXE2BIN.EXE	A Microsoft .EXE-to-.COM file conversion utility (generally available in DOS files).
MAKEBIN.EXE	A .COM-to-.BIN file-conversion utility (supplied in the PDMA-16 Distribution Software).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Then, use the following commands:

```
MASM /DBIN=1 /DTURBOPAS=0 PDMA.ASM;
MASM /DBIN=1 /DQBASIC=0 PDMAIFC.ASM;
LINK PDMAIFC+PDMA,PDMA,,;
EXE2BIN PDMA.EXE PDMA.COM
MAKEBIN PDMA.COM
```

NOTE: If needed, use the batch file *MAKEBAS.BAT* (in the Distribution Software) to build this file.

All five steps must be successful. Note that the linking operation generates the warning:

LINK : Warning L4021 : no stack segment

Disregard this warning; it is irrelevent.

The TPPDMA.OBJ Driver For Turbo PASCAL

To create a TPPDMA.OBJ Driver, you must have access to the following utility:

TASM.EXE TURBO Assembler

Then, use the command

```
TASM /DBIN=0 /DTURBOPAS=1 PDMA.ASM TPPDMA.OBJ.
```

NOTE: If needed, use the batch file *MAKETP.BAT* (in the Distribution Software) to build this file.

The PDMAQB45.QLB Driver For The QuickBASIC Integrated Environment (V4.5)

Make the interface for QuickBASIC Integrated Environment (up to Ver 4.5) using the Quick Library file PDMAQB45.QLB. Specify this file on the command line with the load, **/L** switch. For example, **QB /L PDMAQB45.QLB**.

To create the PDMAQB45.QLB file you must have access to the utility *BQLB45.LIB*, which is the QuickBASIC Integrated Environment Library. Use the following entries:

```
MASM /DBIN=0 /DTURBOPAS=0 PDMA.ASM;
MASM /DBIN=0 /DQBASIC=1 PDMAIFC.ASM;
LINK /q PDMA+PDMAIFC,PDMAQB45,,BQLB45;
```

The PDMAQB45.LIB Driver For A Stand-alone QuickBASIC (V4.5) Program

To create the PDMAQB45.LIB file, you must have access to *MASM* (the Microsoft Assembler) and *LIB.EXE* (the Microsft Library Manager). Then, use the following commands:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
MASM /DBIN=0 /DTURBOPAS=0 PDMA.ASM;
MASM /DBIN=0 /DQBASIC=1 PDMAIFC.ASM;
LIB PDMAQB45-+PDMA;
LIB PDMAQB45-+PDMAIFC;
```

NOTE: If needed, use the batch file *MAKEQB4.BAT* (in the Distribution Software) to build this file.

The PDMAQBX.QLB Driver For The QuickBASIC Extended Environment (V7.0)

To create a QLB library compatible with QuickBASIC Version 7.0, follow the procedure described for QuickBASIC Version 4.5. However, link with *QBXQLB.LIB* , instead of *BQLB45.LIB* , as follows:

```
LINK /q PDMA+PDMAIFC,PDMAQBX,,QBXQLB;
```

Note that the output file (from the linker) is renamed *PDMAQBX.QLB* to avoid incompatibilities with QuickBASIC 4.5.

The PDMAQBX.LIB Driver For A Stand-alone QuickBASIC (V7.0) Program

To create the PDMAQBX.LIB file, you must have access to *MASM* and *LIB.EXE* . Use the following entries:

```
MASM /DBIN=0 /DTURBOPAS=0 PDMA.ASM;
MASM /DBIN=0 /DQBASIC=1 PDMAIFC.ASM;
LIB PDMAQBX-+PDMA;
LIB PDMAQBX-+PDMAIFC;
```

NOTE: If needed, use the batch file *MAKEQBX.BAT* (in the Distribution Software) to build this file.

The PDMAPCF.LIB Driver For PASCAL, C, & FORTRAN

When your Application Code is in PASCAL, C, or FORTRAN, use the PDMAPCF.LIB Driver to compile your Application Program.

To create the PDMAPCF.LIB file, you must have access to *MASM* (the Microsoft Assembler) and *LIB.EXE* (the Microsft Library Manager). Use the following commands:

```
MASM /DBIN=0 /DTURBOPAS=0 PDMA;
MASM /DBIN=0 /DQBASIC=0 PDMAIFC;
LIB PDMAPCF-+PDMA;
LIB PDMAPCF-+PDMAIFC;
```

NOTE: If needed, use the batch file *MAKEPCF.BAT* (in the Distribution Software) to build this file.

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SECTION F3

DRIVER USAGE

F3.1 OVERVIEW

Although your PDMA-16 drivers perform similarly for all supported languages, there are differences from language-to-language in how they pass parameters and parameter values. Items causing confusion are as follows:

- Memory allocation for DMA buffers.
- Separating a FAR (32-bit) pointer into its Segment and Offset values (two 16-bit values).

This chapter discusses these items and any others of concern in the separate treatment of each supported language. Refer to the appropriate section below for details on performing the Mode Calls from the language you are using. The language sections contain brief code fragments for illustration. More information is also available in the example programs (Distribution Software).

F3.2 MICROSOFT C/TURBO C

The C Language, with its large run-time libraries and full pointer-manipulation support, provides the most flexible environment for writing Application Code that fully utilizes your PDMA-16 product.

Function Prototypes

In your Application Code, declare one of the following function prototypes, depending on the Memory Model you will use:

```
mscs_pdma(int *, unsigned *, long *, int *);  /* MS C Small Model */
mscm_pdma(int *, unsigned *, long *, int *);  /* MS C Medium Model */
mscl_pdma(int *, unsigned *, long *, int *);  /* MS C Large Model */
tcs_pdma(int *, unsigned *, long *, int *);   /* Turbo C Small Model */
tcm_pdma(int *, unsigned *, long *, int *);   /* Turbo C Medium Model*/
tcl_pdma(int *, unsigned *, long *, int *);   /* Turbo C Large Model */
```

You have the option of preceding these function prototypes with the C keyword *extern*. Note that each prototype contains a Call Label that corresponds to the Memory Model to be used during compilation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Call Parameters

Declare the Mode Call parameters as follows:

```
int Mode;
unsigned Params[9];
int Flag;
```

The Param[ ] array index values are 0 thru 8, inclusive.

An Example

To call MODE 0 of the PDMA-16 Driver from an MS C Medium Model program, your commands would be

```
    :
Mode=0;
Flag=0;
Params[0]=0X300;        /* BOARD ADDRESS */
Params[1]=3;            /* DMA LEVEL */
Params[2]=7;            /* INTERRUPT LEVEL */

mscm_pdma(&Mode, Params, &Flag);
if (Flag !=0)
   {
        printf("Mode %d Error Flag = &d\n", Mode, Flag);
        exit (1);
   }
     :
     :
```

Note that specifying *Params* in the Call statment is the same as *&Params[0]* .

Linking To The Driver

After compiling your C Application Code, link it to the PDMAPCF.LIB Driver (for Call Label *mscm_pdma*) as follows:

```
LINK your-program,,,PDMAPCF.LIB;
```

If no error reports occur, you will obtain your Application Program *your-program.EXE* , ready to test. If the Linker reports errors such as Unresolved External(s), determine whether you linked to PDMAPCF.LIB correctly.

NOTE: Be sure to use the correct Call Label for the Memory Model you are using.

DMA Memory Buffer Allocation

MODE 1 requires memory buffer represented by a special DMA buffer address in form of one 16-bit value (the segment; to complete the address, the Driver assumes an Offset of 0). This special DMA address is available by calling MODE 9 (for detail, refer to the PDMA-16 User Guide).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Far Pointer Manipulation

MODE 1 allows FAR pointers to be passed in the user Param[] integer array. The Segment and Offset of all FAR pointers(32 bits) in C may be retrieved using C macro: FP_OFF and FP_SEG. Refer to your C Run-time library manual for more detail.

For example,

```
int Mode;
int Flag;
int Params[10];
int far* Buffer;

        .

  Mode = 1 ;
  Flag = 0 ;
  Params[1] =5000;
  Params[2] =0;
  Params[3] =0;
  Params[4] =0;
  Params[5] =1;
  Params[6] =FP_SEG(Buffer);
  Params[7] =FP_OFF(Buffer);

mscm_pdma(&Mode,Params,&Flag);
  if(result = 0)
    ReportError;
    Return;
```

F3.3 MICROSOFT PASCAL

The Software Driver Mode Call Labels

In your program, declare the following function prototype:

```
FUNCTION MSP_PDMA(VAR Mode:integer;VAR Params:PArray;VAR Flag:integer):integer; external;
```

The Call Parameters

Declare the mode call parameters as follows:

```
TYPE
  PArray = array [1..9] of word ;

VAR
  Params          : PArray;       (* MODE PARAM ARRAY *)
  Mode,Flag       : integer;      (* MODE CALL VARIABLES *)
  Result          : integer;      (* MODE CALL RETURN VALUE *)
```

The Params[ ] array index values are 1 thru 9, inclusive. Note that if PArray TYPE is declare as [0..9], the index value starts at 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example

To call MODE 0 of the PDMA-16 driver from MS Pascal program,

```
    :
Mode := 0;
Params[1] := 768;          (* BOARD ADDRESS *)
Params[2] := 3;            (* DMA LEVEL *)
Params[3] := 7;            (* INTERRUPT LEVEL *)

Result := MSP_PDMA (Mode, Params, Flag);
if (Result <> 0) then ReportError;
    :
```

where *ReportError* is a previously declared procedure that displays an error message and terminates the program. Refer to the Microsoft PASCAL example program (in the Distribution Software) for more detail.

Linking To The Driver/Interface Module

Once you have written your MS Pascal program, you must compile and LINK it to the Interface Module, *PDMAPCF.LIB* . PDMAPCF.LIB is where the label *MSP_PDMA* label resides.

For example,

```
PL   your-program ;
LINK  your-program ,,,PDMAPCF.LIB;
```

If no errors occur, you have the executable file *your-program.EXE* that is ready to test. If the linker reports errors such as Unresolved External(s), you must determine whether you linked to PDMAPCF.LIB correctly.

DMA Memory Buffer Allocation

MODE 1 requires memory buffer represented by a special DMA buffer address in form of two 16-bit address, the Segment and Offset Adresses. This special DMA address is available by calling MODE 23 (for detail, refer to the DAS-16 User Guide).

Far Pointer Manipulation

MODE 1 allows FAR pointers to be passed in the user Params[] integer array. The Segment and Offset of all FAR pointers (32 bits) in MS PASCAL may be retrieved using the built-in operator ADS and the *.S* and *.R* sub-operators. Refer to your MS Pascal Run-time library manual for more detail.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For example,

```
Type
 DArray = array [1..5000] of integer;
var
 Buffer      : DArray;

        .

 Mode := 9 ;
 Flag := 0 ;
 Params[1] :=5000;
 Params[2] :=01;
 Params[3] :=0;
 Params[4] :=0;
 Params[5] :=1;
 Params[6] :=ORD((ADS Buffer).S);
 Params[7] :=ORD((ADS Buffer).R);
 Params[8] :=0;
 Params[9] :=dest_seg;
 Result := msp_pdma(Mode,Params,Flag) ;
 if(result <> 0) then
 Begin
    ReportError;
    Return;
 End;
```

F3.4 BORLAND TURBO PASCAL

The Call Label

The Call Label *TP_PDMA* is usable from any Turbo Pascal program; declare this label in your Application Code as follows:

```
FUNCTION TP_PDMA(VAR Mode:integer;VAR Params:PArray;VAR Flag:integer):integer; external;
```

The Call Parameters

Declare the Mode Call parameters as follows:

```
TYPE
     PArray = array [1..9] of word;
VAR
     Param           : PArray;       (* MODE PARAM ARRAY *)
     Mode,Flag       : integer;      (* MODE CALL VARIABLES *)
     Result          : integer;      (* MODE CALL RETURN VALUE *)
```

The Param[ ] array index values are 10 thru 9, inclusive. The index values start at 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

An Example:

To call Mode 15 of the PDMA-16 Driver from Turbo Pascal Application Code:

```
     :
Mode := 0;
Params[1] := 768;                    (* BOARD NUMBER 0 *)
Params[2] := 3;                      (* DMA LEVEL *)
Params[3] := 7;                      (* INTERRUPT LEVEL *)

Result := TP_PDMA (Mode, Params, Flag);
if (Result <> 0) then ReportError;
     :
```

Where *ReportError* is previously declared procedure that displays an error message and terminates the program. Refer to the Turbo Pascal example program provided for more detail.

Linking To The Driver

The Turbo Pascal Driver is *TPPDMA.OBJ* . This file is linked into your program using the $L Compiler Directive. Include this command at the beginning of your progam as follows:

```
{$L TURBOPAS}
```

Once included, you are ready to compile your program with the command

```
TPC your-program
```

DMA Memory Buffer Allocation

MODE 1 requires memory buffer represented by a special DMA buffer address in form of one 16-bit value (the segment; to complete the address, the Driver assumes an Offset of 0). This special DMA address is available by calling MODE 9 (for detail, refer to the PDMA-16 User Guide).

Far Pointer Manipulation

MODE 1 allows FAR pointers to be passed in the user Param[] integer array. The Segment and Offset of all FAR pointers (32 bits) in Turbo Pascal may be retrieved using built-in function *Ofs* and *Seg* . Refer to your Turbo Pascal Run-time library manual for more detail.

For example,

```
Type
 PArray = array [1..9]of integer ;
 DArray = array [1..5000] of integer;
var
 Params              : PArray ;
 Buffer              : DArray;
 Mode,Flag            : integer;

        .
        .
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
Mode := 1 ;
Flag := 0 ;
Param[1] :=5000;
Param[2] :=0;
Param[3] :=0;
Param[4] :=0;
Param[5] :=1;
Param[6] :=SEG(Buffer);
Param[7] :=OFS(Buffer);
Result := tp_pdma(Mode,Param,Flag) ;
```

F3.5 MICROSOFT FORTRAN

The Software Driver Call Label

The call label *msf_pdma* is usable from any MS FORTRAN Application Code; no prototype declaration of the label is required.

The Mode Call Parameters

Declare the Mode Call parameters as follows:

```
integer*2 i(15)              !Parameter Array
integer*2 mode               !Mode number
integer*2 flag               !Return error flag
```

Note that by default, FORTRAN array index values begin at 1. The latest versions of FORTRAN, however, allow you override this default to start at Index Value 0. Refer to your FORTRAN Manuals for more detail.

An Example

To call MODE 0 of the Driver from Microsoft FORTRAN Application Code,

```
mode=0
i(1)=768                     ! Board Address
i(2)=3                       ! DMA Level
i(3)=7                       ! Interrupt Level

call msf_pdma(mode, i(1), Flag)
if (flag .NE. 0) then
    print *,'Mode = ',mode,'     Error # ',flag
endif
```

Linking To The Driver

After compiling your FORTRAN Application Code, link it to the PDMAPCF.LIB Driver (for the Call Label *msf_pdma*) as follows:

```
LINK your-program,,,PDMAPCF.LIB;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If no error reports occur, you will obtain your Application Program *your-program.EXE* , ready to test. If the Linker reports errors such as Unresolved External(s), determine whether you linked to PDMAPCF.LIB correctly.

DMA Memory Buffer Allocation

MODE 1 requires memory buffer represented by a special DMA buffer address in form of one 16-bit value (the segment; to complete the address, the Driver assumes an Offset of 0). This special DMA address is available by calling MODE 9 (for detail, refer to the PDMA-16 User Guide).

Far Pointer Manipulation

MODE 1 allows FAR pointers to be passed in the user Param() integer array. The Segment and Offset of all FAR pointers that are to represent a memory buffer in FORTRAN may be retrieved using the FORTRAN intrinsic function *LOCFAR()* and some simple calculation. Refer to your FORTRAN Run-time library manual and our FORTRAN example programs for more detail.

For example,

```
      integer*2 Buffer(5000),params(16)
      integer*2 mode,flag
      integer*2 Buffer_off,Buffer_seg
      integer*4 Buffer_addr

              .
              .
              .
C     Get segment and offset address of DMA Buffer
      Buffer_addr=LOCFAR(Buffer(1))
      Buffer_seg=Buffer_addr/#10000
      Buffer_off=Buffer_addr-(Buffer_seg*#10000)

      mode=2
      flag=0
      params(1)=5000
      params(2)=0
      params(3)=0
      params(4)=0
      params(5)=1
      params(6)=Buffer_seg
      params(7)=Buffer_off
      call msf_pdma(mode, params(1), flag)
```

F3.6 MICROSOFT QUICKBASIC

The Call Label

You must declare the Call Label in your Application Code. Make this declaration by inserting the following command at the beginning of your Code:

```
DECLARE SUB QBPDMA (MD%, BYVAL PARAMS%, FLAG%)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note that all subroutine DECLAREs in your program MUST be made before any $DYNAMIC arrays are allocated. $DYNAMIC data is data that is allocated space in the FAR heap, outside the default data segment. All arrays used for data acquisition must be declared as $DYNAMIC; QuickBasic assumes $STATIC data (Default data segment) unless otherwise specified.

The Call Parameters

Declare the Mode Call parameter array D%(15) as follows:

```
DIM D%(9)
COMMON SHARED D%()
```

The term *COMMON SHARED* allows the use other modules and subroutines in this array.

An Example

To initialize your PDMA-16 board, use MODE 0 as follows:

```
    :
180 MD% = 0                              'initialize mode
190 FLAG% = 0                            'declare error variable
200 D%(0) = &H300                        'Card BASE ADDRESS
210 D%(1) = 3                            'DMA LEVEL
220 D%(2) = 7                            'INTERRUPT LEVEL
240 CALL QBPDMA(MD%, VARPTR(D%(0)), FLAG%)
250 IF FLAG% <> 0 THEN PRINT "MODE 0 Error # "; FLAG% : STOP
    :
```

Linking To The Driver

The QuickBASIC interface consists three separate Drivers:

PDMAQB45.QLB — Use when you load the QuickBASIC Enviroment Version 4.5 and you plan to run your program from within the Environment (no EXE envolved here). Use the /L switch to load this Quick Library into QuickBASIC:

```
QB /L PDMAQB45  your-program
```

PDMAQBX.QLB — This is identical to PDMAQB45.QLB except that it is designed for QuickBASIC Extended Environment Version 7.0 (QBX). Use the /L switch to load this Quick Library into QuickBASIC:

```
QBX /L PDMAQBX  your-program
```

PDMAQB45.LIB — Use when you want to make a stand-alone EXE program from your QuickBASIC (4.5) source. To create such a program, use *BC* and *LINK*, the QuickBASIC compiler and linker as follows:

```
BC  your-program  /o;
LINK  your-program,,,PDMAQB45.LIB
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDMAQBX.LIB — Use when you want to make a stand-alone EXE program from your QuickBASIC (7.0) source. To create such a program, use *BC* and *LINK*, the QuickBASIC compiler and linker as follows:

```
BC  your-program  /o;
LINK  your-program,,,PDMAQBX.LIB
```

NOTE: All $DYNAMIC data declaration must occur after all COMMON and DECLARE statements in your program. If you get the QB error, COMMON and DECLARE must precede all executable statements; double check the order of all DECLARE, COMMON, and $DYNAMIC declarations.

DMA Memory Buffer Allocation

MODE 1 requires memory buffer represented by a special DMA buffer address in form of one 16-bit value (the segment; to complete the address, the Driver assumes an Offset of 0). This special DMA address is available by calling MODE 9 (for detail, refer to the PDMA-16 User Guide).

Far Pointer Manipulation

QuickBASIC provides the built-in functions VARPTR and VARSEG for obtaining the Offset and Segment of a given variable. If the variable is declared in the $STATIC area (by default), VARSEG returns the default data segment. If the variable is declared as $DYNAMIC, then it is placed in the FAR heap and VARSEG for such a variable returns a unique Segment value outside the default data segment.

For example,

```
DIM BUFFER%(1000)           ' Data array used by MODE 9

    .
    .
    .

MD% = 2
PARAM%(0) = 1000
PARAM%(1) = 0
PARAM%(2) = 0
PARAM%(3) = 0
PARAM%(4) = 1
PARAM%(5) = VARSEG(BUFFER%(0))
PARAM%(6) = VARPTR(BUFFER%(0))

CALL QBPDMA(MD%, VARPTR(PARAM%(0)), FLAG%)  'make transfer
IF FLAG% <> 0 THEN PRINT "Mode 1 error # "; FLAG%: STOP
```

■ ■ ■

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP