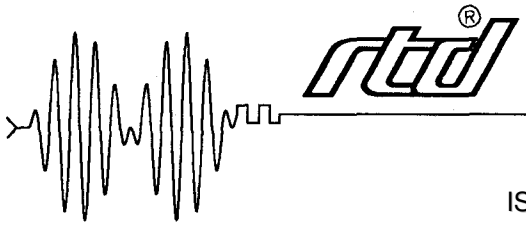
AD3700
User's Manual



Real Time Devices, Inc.

"Accessing the Analog World"™

ISO9001 and AS9100 Certified

AD3700

User's Manual

REAL TIME DEVICES, INC.
820 North University Drive
Post Office Box 906
State College, Pennsylvania 16804 USA
Phone: (814) 234-8087
FAX: (814) 234-5218

Published by
Real Time Devices, Inc.
820 N. University Dr.
P.O. Box 906
State College, PA 16804 USA

Printed in U.S.A.

Rev. C 9239

Table of Contents

LIST OF ILLUSTRATIONS

INTRODUCTION

The AD3700 DataMaster™ board turns your IBM PC XT/AT or compatible computer into a high-speed, high-performance data acquisition and control system. Installed within a single expansion slot in the computer, the AD3700 features:

- Eight single-ended analog input channels,
- 12-bit, 5 microsecond analog-to-digital converter with 200 kHz throughput,
- ±5, ±10, or 0 to +10 volt input range,
- Resistor-configurable input gain,
- Four conversion modes and programmable channel scan option,
- On-board FIFO interface and DMA transfer,
- Trigger in and trigger out for external triggering or cascading boards,
- Eight digital input and eight digital output lines,
- Four user-configurable 16-bit timer/counters which can be used to generate interrupts, or as an event counter, a frequency counter, a programmable one-shot, a rate generator, or for other special functions,
- BASIC, Turbo Pascal, and Turbo C source code; diagnostics program.

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

Analog-to-Digital Conversion

The analog-to-digital (A/D) circuitry receives up to eight single-ended analog inputs and converts these inputs into 12-bit digital data words which can then be read and/or transferred to PC memory.

The input voltage range is jumper-selectable for bipolar ranges of -5 to +5 volts or -10 to +10 volts, or a unipolar range of 0 to +10 volts. It is not necessary to recalibrate after changing the input range or polarity. The board is factory set for -5 to +5 volts. Overvoltage protection to ±35 volts is provided at the inputs.

A user-configurable gain, Gx, is provided so that you can customize a gain for a specific application. Gx is set as described in Chapter 1.

A/D conversions are performed in 5 microseconds, with a maximum throughput rate of 200 kHz. Conversions are controlled through software, by an on-board pacer clock, or by an external trigger brought onto the board through the I/O connector. A first in, first out (FIFO) interface helps your computer manage the high throughput rate of the A/D converter by acting as an elastic storage bin for the converted data. Even if the computer does not read the data as fast as conversions are performed, conversions can continue until the FIFO is full.

The converted data can be transferred to PC memory in one of two ways: by using the microprocessor or by using direct memory access (DMA). The mode of transfer and DMA channel are chosen through software. The PC data bus is used to read and/or transfer data, one byte at a time, to PC memory. In the DMA transfer mode, you can transfer a selected block of data in a single data dump, or you can make continuous transfers directly to PC memory without going through the processor.

8254 Timer/Counters

Two 8254 programmable interval timers, TC1 and TC2, each contain three 16-bit, 8-MHz timer/counters to support a wide range of timing and counting functions. Two of the timer/counters in TC1 are cascaded and used internally for the pacer clock. The third is available for counting applications. The three timer/counters in TC2 are cascaded for timing applications.

Digital I/O

The AD3700 has eight input and eight output TTL/CMOS-compatible digital lines which can be directly interfaced with external devices or signals to sense switch closures, trigger digital events, or activate solid-state relays. The input lines have on-board pull-up resistors.

What Comes With Your Board

You receive the following items in your AD3700 package:

- AD3700 interface board
- Software and diagnostics diskette with example programs in BASIC, Turbo Pascal, and Turbo C; source code
- User's manual

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

Board Accessories

In addition to the items included in your AD3700 package, Real Time Devices offers a full line of software and hardware accessories. Call your local distributor or our main office for more information about these accessories and for help in choosing the best items to support your board's application.

Application Software and Drivers

Our custom application software packages provide excellent data acquisition and analysis support. Use SIGNAL*MATH for integrated data acquisition and sophisticated digital signal processing and analysis, or ATLANTIS for real-time monitoring and data acquisition. rtdLINX and labLINX drivers provide full-featured high level interfaces between the AD3700 and custom or third party software, including LABTECH NOTEBOOK, NOTEBOOK/XE, and LT/CONTROL. rtdLINX source code is available for a one-time fee. Our Pascal and C Programmer's Toolkit provides routines with documented source code for custom programming.

Hardware Accessories

Hardware accessories for the AD3700 include the MX32 analog input expansion board which can expand a single input channel on your AD3700 to 16 differential or 32 single-ended input channels, SSH4/SSH8 four- and eight-channel simultaneous sample-and-hold boards, MR series mechanical relay output boards, OP series optoisolated digital input boards, the OR16 mechanical relay/optoisolated digital I/O board, the TS16 thermocouple sensor board, the TB50 terminal board and XB50 prototype/terminal board for prototype development and easy signal access, EX-XT and EX-AT extender boards for simplified testing and debugging of prototype circuitry, and the XT50 twisted pair flat ribbon cable assembly for external interfacing.

Using This Manual

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

When You Need Help

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

CHAPTER 1

BOARD SETTINGS

The AD3700 board has jumper and switch settings you can change if necessary for your application. The board is factory-configured with the most often used settings. The factory settings are listed and shown on a diagram in the beginning of this chapter. Should you need to change these settings, use these easy-to-follow instructions before you install the board in your computer.

Also note that by installing two resistors and a trimpot on the board, you can define the user-configurable gain, Gx, to be whatever value your application may require. A pad for installing a capacitor, C51, is also included in the gain circuitry for creating a low-pass filter. The procedure for customizing Gx is included at the end of this chapter.

Factory-Configured Switch and Jumper Settings

Table 1-1 lists the factory settings of the user-configurable jumpers and switches on the AD3700 board. Figure 1-1, on the next page, shows the board layout and the locations of the factory-set jumpers. The following paragraphs explain how to change the factory settings. Pay special attention to the setting of S1, the base address switch, to avoid address contention when you first use your board in your system.

Table 1-1 — Factory Settings

Switch/ Jumper	Function Controlled	Factory Setting
P3	Sets the A/D input voltage range	10 volts
P4	Sets the FIFO full/FIFO half-full flag to halt A/D conversions when full or half-full	FIFO full
P5	Sets the analog input for unipolar or bipolar	Bipolar
P6	Sets 8254 TC2's clock and gate sources and TIMER output	XTAL (top), +5V, and OUT0
P7	Sets the pacer clock source	XTAL
P8	Sets 8254 TC1, Counter 2's clock and gate sources	+5V, OUT1
P9	Selects the external trigger in or external gate signal to be available for monitoring	TRIGIN
P10	Jumper setting A sets the 3700 to be fully compatible with earlier 3700 boards (scan functions limited); jumper setting B provides full board capability	Jumper installed on Group B (not compatible with earlier boards)
P11	Configures the 3700 for normal use or for use with RTD's SSH series simultaneous sample- and-hold boards	NOR
S1	Sets the base address	300 hex (768 decimal)

P3 — Input Voltage Range (Factory Setting: 10V)

This header connector, located in the upper right area of the board, sets the input voltage range at 10 or 20 volts. The 10V setting is for the ±5 volts and 0 to +10 volts ranges; the 20V setting is for the ±10 volt range. Figure 1-2 shows P3 with the jumper installed at 10V. You do not have to recalibrate the board when you change voltage ranges.

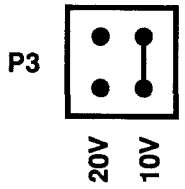


Fig. 1-2 — Input Voltage Range Jumper, P3

P4 — FIFO Full/Half-Full Flag (Factory Setting: FIFO Full)

This header connector, located above the FIFO at the top of the board, is used to halt A/D conversions when the FIFO is full (FF) or half-full (HF). The advantage of setting the FIFO to stop conversions when it is half-full is the assurance that there is room in the FIFO to store both bytes of the current conversion before shut-off. It is possible to lose the LSB of a conversion when the jumper is set to FIFO full, since the FF flag signals that only one 8-bit slot remains in the FIFO to be filled and each 12-bit conversion requires two 8-bit slots, one for the MSB and one for the LSB. Figure 1-3 shows P4 with the jumper installed so that conversions are halted when the FIFO is full.

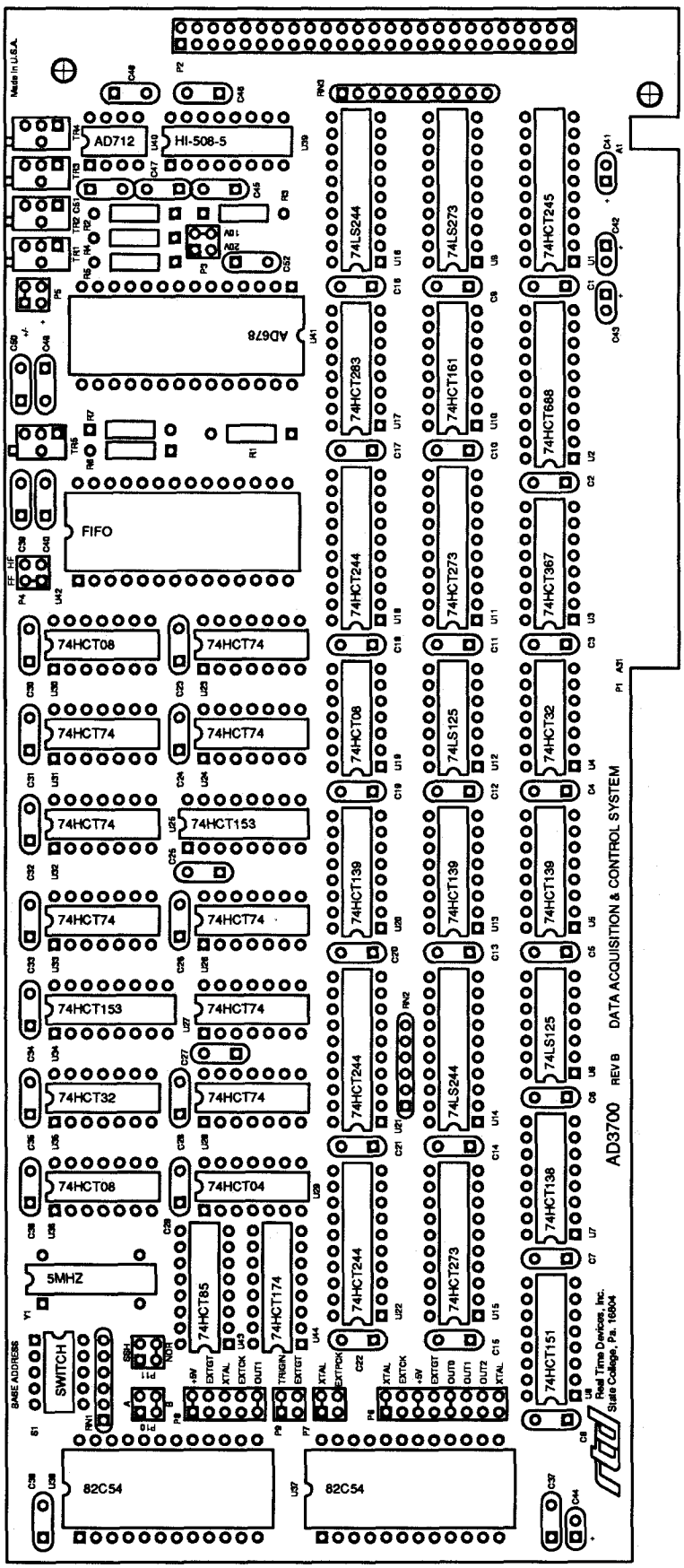


Fig. 1-1 — Board Layout Showing Factory-Configured Settings

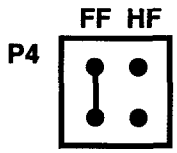


Fig. 1-3 — FIFO Full/Half-Full Flag Jumper, P4

P5 — Unipolar/Bipolar Analog Input (Factory Setting: Bipolar)

This header connector, shown in Figure 1-4, configures the analog input for unipolar (0 to +10 volts) or bipolar (±5 or ±10 volts) operation. You do not have to recalibrate the board when you change polarity.

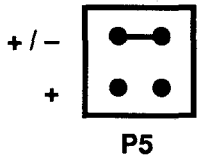


Fig. 1-4 — Analog Input Polarity Jumper, P5

P6 — Timer/Counter 2 Source and OUT Select (Factory Settings: XTAL (top), +5V, OUT0)

This header connector, shown in Figure 1-5, configures timer/counter 2's clock and gate sources and the selected TIMER output to the I/O connector (P2-42). The top two pairs of pins, XTAL and EXTCK, set the clock source for the three cascaded counters in TC2. XTAL connects the counters to the on-board 5-MHz clock, and EXTCK connects them to an external clock source brought onto the board through the I/O connector. The +5V and EXTGT pins connect the counters' gate input to +5 volts or to an external gate brought onto the board through the I/O connector. The bottom four pins, OUT0, OUT1, OUT2, and XTAL, let you select any one of the three counter outputs or the on-board 5-MHz clock to be available at the TIMER output on the I/O connector. The timer/counters are further described in Chapters 3 and 4.

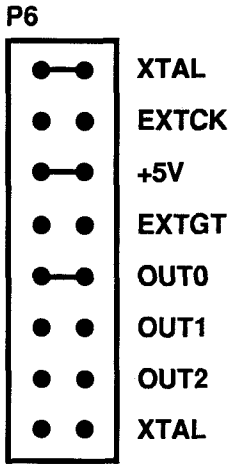


Fig. 1-5 — TC2 Source and OUT Select Jumper, P6

P7 — Pacer Clock Source Select (Factory Setting: XTAL)

This header connector, shown in Figure 1-6, connects the pacer clock's clock source to the on-board 5 MHz (XTAL) clock or to an external clock applied through I/O connector P2.

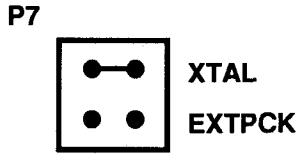


Fig. 1-6 — Pacer Clock Source Select Jumper, P7

P8 — TC1, Counter 2 Sources (Factory Settings: +5V, XTAL)

This header connector, shown in Figure 1-7, configures the clock and gate sources for Counter 2 in TC1. The top two pairs of pins set the gate input for +5 volts or the external gate source. The bottom three pairs of pins set the clock source for the on-board 5-MHz clock (XTAL), the external clock source (EXTCK), or the output of the pacer clock (OUT1). Note that the external gate and clock sources are the same ones connected to P6 for TC2.

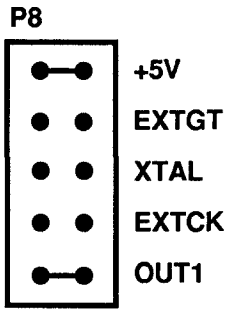


Fig. 1-7 — TC1, Counter 2 Sources Jumper, P8

P9 — External Trigger/External Gate Monitor (Factory Setting: External Trigger)

This header connector, shown in Figure 1-8, lets you select either the external trigger input (P2-39) or the external gate input (P2-46) to be available for monitoring at bit 4 of the status word (BA +3).

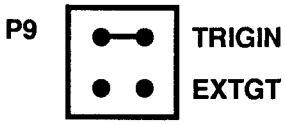


Fig. 1-8 — External Trigger/External Gate Monitor Jumper, P9

P10 — Board Compatibility Select (Factory Setting: Jumper on B)

This header connector, shown in Figure 1-9, allows you to maintain software and hardware compatibility with earlier AD3700 boards (board serial numbers 64XXXX). By installing a jumpers on the A pins (top) your new AD3700 will be fully compatible in data acquisition and control systems using the earlier board. However, the new AD3700's expanded features such as programmable channel scan cannot be used. When the jumper is installed across the B pins (factory setting), all new AD3700 functions are activated, but compatibility with previous boards is lost.

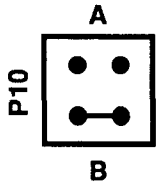


Fig. 1-9 — Board Compatibility Select Jumper, P10

P11 — Simultaneous Sample-and-Hold Select (Factory Setting: NOR)

This header connector, shown in Figure 1-10, configures the AD3700 to operate normally, or with Real Time Devices' SSH4 or SSH8 simultaneous sample-and-hold board. The SSH setting adapts the triggering for optimal use on the SSH boards.

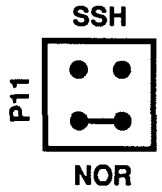


Fig. 1-10 — Simultaneous Sample-and-Hold/Normal Operation Jumper, P11

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

One of the most common causes of failure when you are first trying your board is address contention. Some of your computer's I/O space is already occupied by internal I/O and other peripherals. When the AD3700 board attempts to use I/O address locations already used by another device, contention results and the board does not work.

To avoid this problem, the AD3700 has an easily accessible DIP switch, S1, which lets you select any one of 32 starting addresses in the computer's I/O. Should the factory setting of 300 hex (768 decimal) be unsuitable for your system, you can select a different base address simply by setting the switches to any value shown in Table 1-2. The table shows the switch settings and their corresponding decimal and hexadecimal (in parentheses) values. Note that switch 5 is the leftmost switch and switch 1 is the rightmost switch when looking at the component side of the board. When the switches are pulled forward, they are OPEN, or set to logic 1, as labeled on the DIP switch

Table 1-2 — Base Address Switch Settings, S1

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

package. When you set the base address for your board, record the value in the table inside the back cover. Figure 1-11 shows the DIP switch set for a base address of 300 (decimal 768) (switch 5 OPEN).

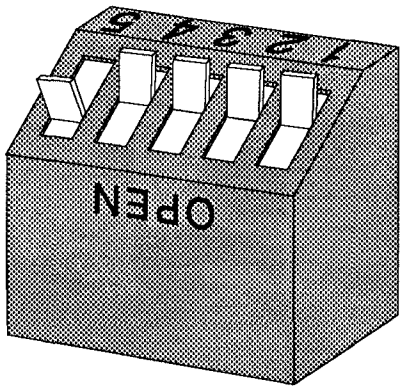


Fig. 1-11 — Base Address Switch, S1

Gx, User-Configurable Gain

Gx is provided so that you can easily configure a special gain setting for a specific application. Note that when you use this feature and set up the board for a gain of other than 1, all of the input channels will operate only at your custom gain setting. Gx is derived by adding resistors R2 and R3, trimpot TR4, and capacitor C51, all located in the upper right area of the board. The resistors and trimpot combine to set the gain, as shown in the formula in Figure 1-12. Capacitor C51 is provided so that you can add low-pass filtering in the gain circuit. If your input signal is a slowly changing one and you do not need to measure it at a higher rate, you may want to add a capacitor at C51 in order to reduce the input frequency range and in turn reduce the noise on your input signal. The formula for setting the frequency is given in the diagram below. If you install a custom gain circuit, a small trace on the bottom (non-component) side of the board must be cut to activate the circuit. Figure 1-12 shows how the Gx circuitry is configured.

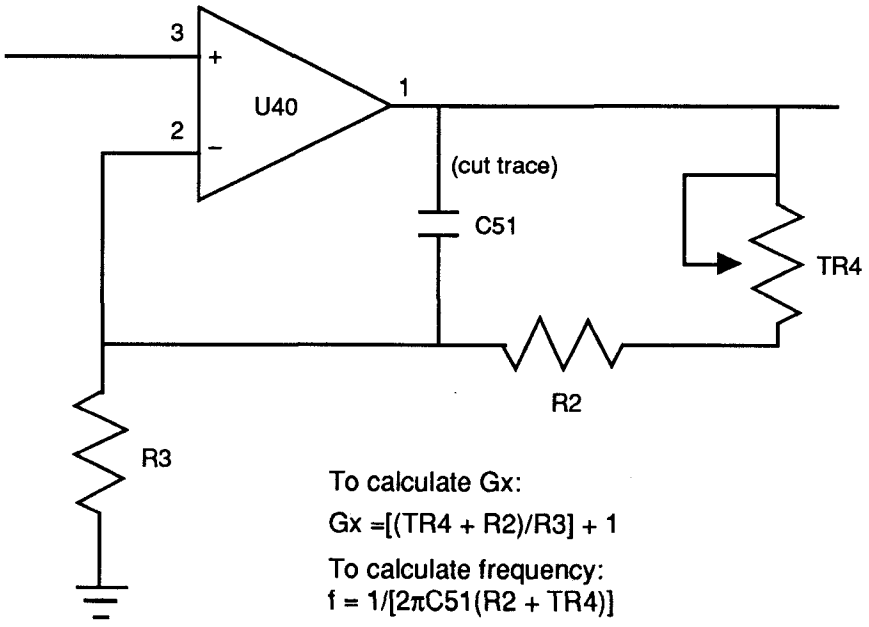


Fig. 1-12 — Gain Circuitry and Formulas for Calculating Gx and f

CHAPTER 2

BOARD INSTALLATION

The AD3700 board is easy to install in your IBM PC/XT/AT or compatible computer. It can be placed in any full-sized slot. This chapter tells you step-by-step how to install and connect the board.

After you have installed the board and made all of your connections, you can turn your system on and run the 3700DIAG board diagnostics program included on your example software disk to verify that your board is working.

Board Installation

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

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

To install the board:

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

   NOTE: Do not force the board into the slot. If the board does not slide into place, remove it and try again. Wiggling the board or exerting too much pressure can result in damage to the board or to the computer.
7. After the board is installed, secure the slot bracket back into place and put the cover back on your computer. The board is now ready to be connected via the external I/O connector at the rear panel of your computer. Be sure to observe the keying when connecting your external cable to the I/O connector.

External I/O Connections

Figure 2-1 shows the AD3700's P2 I/O connector pinout. Refer to this diagram as you make your I/O connections.

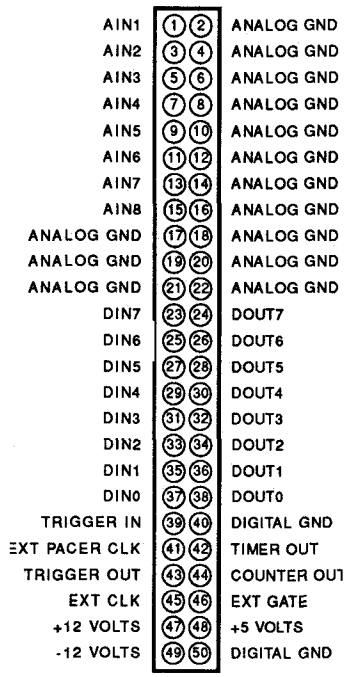


Fig. 2-1 — P2 I/O Connector Pin Assignments

Connecting the Analog Inputs

Connect the high side of the analog input to one of the analog input channels, AIN1 through AIN8, and connect the low side to the selected channel's dedicated ANALOG GND. Figure 2-2 shows how these connections are made.

NOTE: It is good practice to connect all unused channels to ground, as shown with channel 8 in the following diagrams. Failure to do so may affect the accuracy of your results.

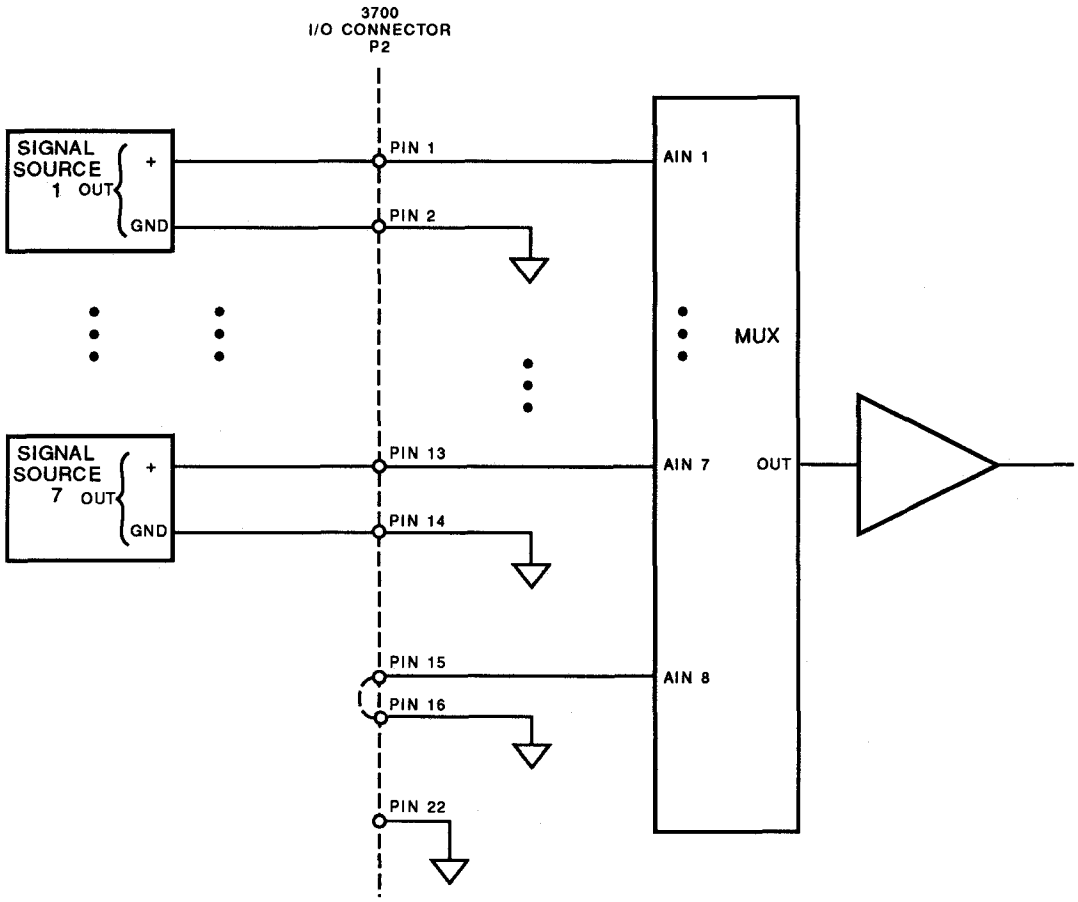


Fig. 2-2 — Analog Input Connections

Connecting the Trigger In and Trigger Out Pins, Cascading Boards

The AD3700 board has an external trigger input (P2-39) and output (P2-43) so that conversions can be started based on external events, or so that two or more boards can be cascaded and run synchronously in a "master/slave" configuration. By cascading two (or more) boards as shown in Figure 2-3, they can be triggered to start an A/D conversion at the same time (sampling uncertainty is less than 50 nanoseconds). When you cascade boards, be sure to set each board for a different base address (see Chapter 1), or system contention will result.

NOTE: The only delay you must take into account when cascading boards is the time it takes for the trigger signal to propagate through the boards. Because the sampling uncertainty is less than 50 nanoseconds, this should not affect boards operating at lower conversion rates. However, it may cause timing problems when you operate at higher speeds. If you want to make sure of precise, simultaneous triggering at higher speeds, then connect the trigger signal to the trigger input of each board, or use RTD's SSH4 or SSH8 four- or eight-channel simultaneous sample-and-hold board.

If you apply an external trigger to the board's trigger in pin, note that the board is triggered on the positive edge of the pulse. The pulse duration should be at least 50 nanoseconds.

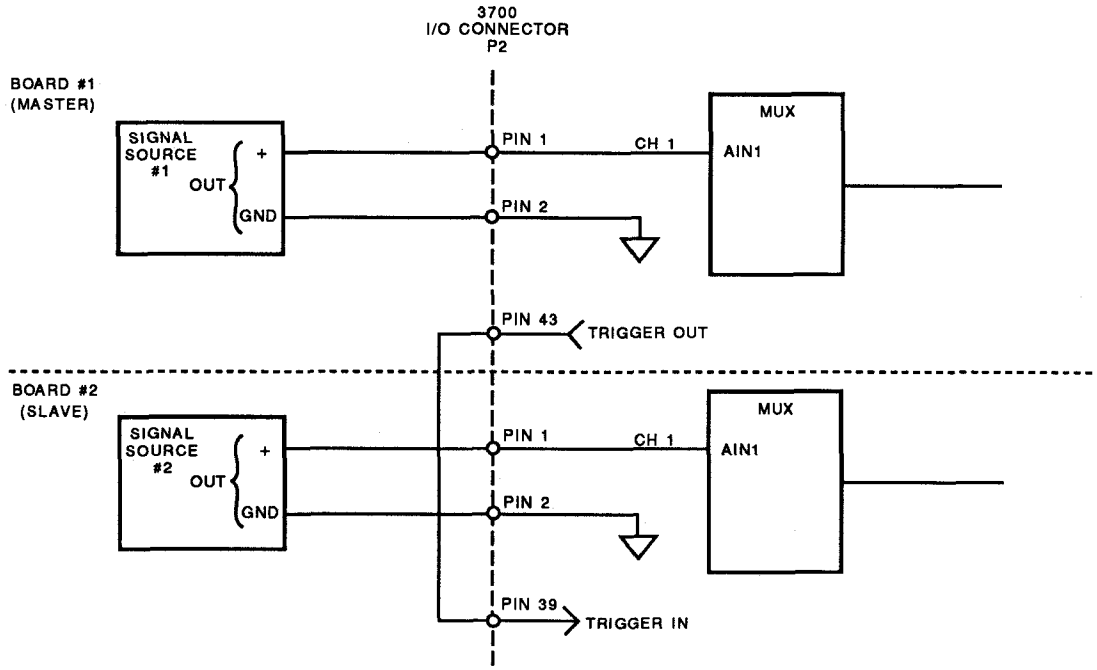


Fig. 2-3 — Cascading Two Boards for Simultaneous Sampling

Connecting the Timer/Counters and Digital I/O

For all of these connections, the high side of an external signal source or destination device is connected to the appropriate signal pin on the I/O connector, and the low side is connected to any DIGITAL GND.

Running the 3700DIAG Diagnostics Program

Now that your board is ready to use, you will want to try it out. An easy-to-use, menu-driven diagnostics program, 3700DIAG, is included with your example software to help you verify your board's operation. You can also use this program to make sure that your current base address setting does not contend with another device.

CHAPTER 3

HARDWARE DESCRIPTION

This chapter describes the features of the AD3700 hardware. The major circuits are the A/D, the timer/counters, and the digital I/O lines.

The AD3700 board has three major circuits, the A/D, the timer/counters, and the digital I/O lines. Figure 3-1 shows the block diagram of the board. This chapter describes the hardware which makes up the major circuits.

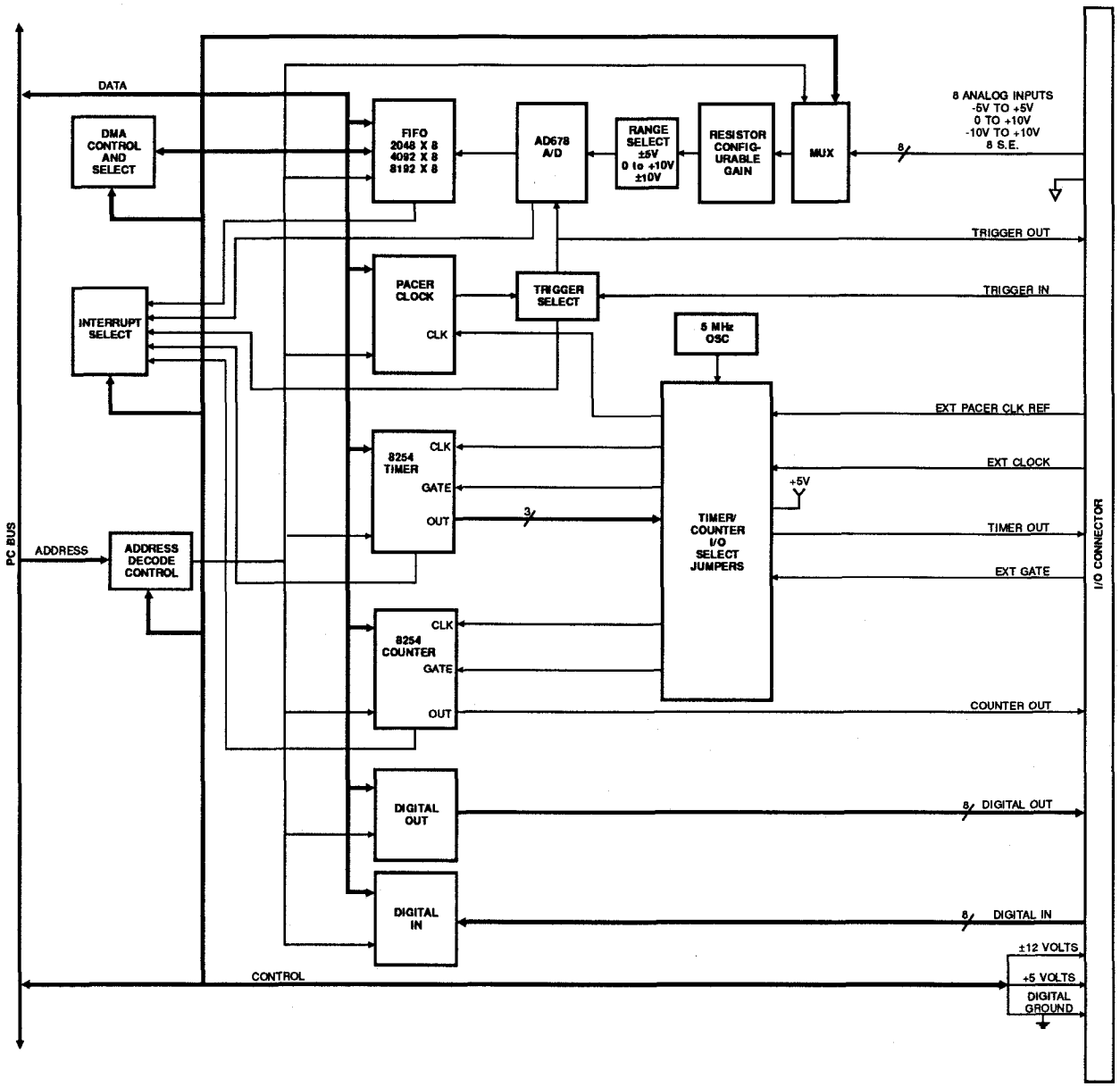


Fig. 3-1 — AD3700 Block Diagram

A/D Conversion Circuitry

The AD3700 board performs analog-to-digital conversions on up to eight software-selectable analog input channels. The following paragraphs describe the A/D circuitry.

Analog Inputs

The input voltage range is jumper-selectable for -5 to +5 volts, -10 to +10 volts, or 0 to +10 volts. A user-configurable gain, Gx, lets you amplify lower level signals to more closely match the board's input ranges. When

you increase the gain, the effective input range decreases by the input range divided by the gain. You can customize this gain setting by following the instructions at the end of Chapter 1. Overvoltage protection to ±35 volts is provided at the inputs.

A/D Converter

The AD678 12-bit successive approximation A/D converter accurately digitizes dynamic input voltages in 5 microseconds, for a maximum throughput rate of 200 kHz. The AD678 contains a sample-and-hold amplifier, a 12-bit A/D converter, a 5-volt reference, a clock, and a digital interface to provide a complete A/D conversion function on a single chip. Its low-power CMOS logic combined with a high-precision, low-noise design give you accurate results.

Conversions are controlled through software (internally triggered) or by an external trigger brought onto the board through the I/O connector. An on-board pacer clock can be used to control the conversion rate. Conversion modes and channel select options are described in Chapter 4, *Board Operation and Programming*.

FIFO Interface

A first in, first out (FIFO) interface helps your computer manage the high throughput rate of the A/D converter by providing an elastic storage bin for the converted data. Even if the computer does not read the data as fast as conversions are performed, conversions will continue until a FIFO full flag (or half-full flag, depending on the setting of the jumper at P4) is sent to stop the converter. The size of the FIFO was specified as 2K, 4K, or 8K when you placed your board order.

The FIFO does not need to be addressed when you are writing to or reading from it; internal addressing makes sure that the data is properly stored and retrieved. All data accumulated in the FIFO is stored intact until the PC is able to complete the data transfer. Its asynchronous operation means that data can be written to or read from it at any time, at any rate. When a transfer does begin, the data first placed in the FIFO is the first data out.

The converted data can be transferred to PC memory in one of two ways: through the PC data bus or by using direct memory access (DMA). Data bus transfers take more processor time to execute. They use polling and interrupts to determine when data has been acquired and is ready for transfer. DMA places data directly into the PC's memory, one byte at a time, with minimal use of processor time. DMA transfers are managed by the DMA controller as a background function of the PC, letting you operate at higher throughput rates.

Timer/Counters

Two 8254 programmable interval timers, TC1 and TC2, provide six 16-bit, 8-MHz timer/counters to support a wide range of timing and counting functions. Two of the timer/counters in TC1 are cascaded and used for the pacer clock. The pacer clock is described in Chapter 4. You can use the remaining four timer/counters – one from TC1 for counting applications and three cascaded on TC2 for timing applications. Figure 3-2 shows the timer/counter circuitry.

Each timer/counter has two inputs, CLK in and GATE in, and one output, timer/counter OUT. They can be programmed as binary or BCD down counters by writing the appropriate data to the command word, as described in Chapter 4. The command word also lets you set up the mode of operation. The six programmable modes are:

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

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

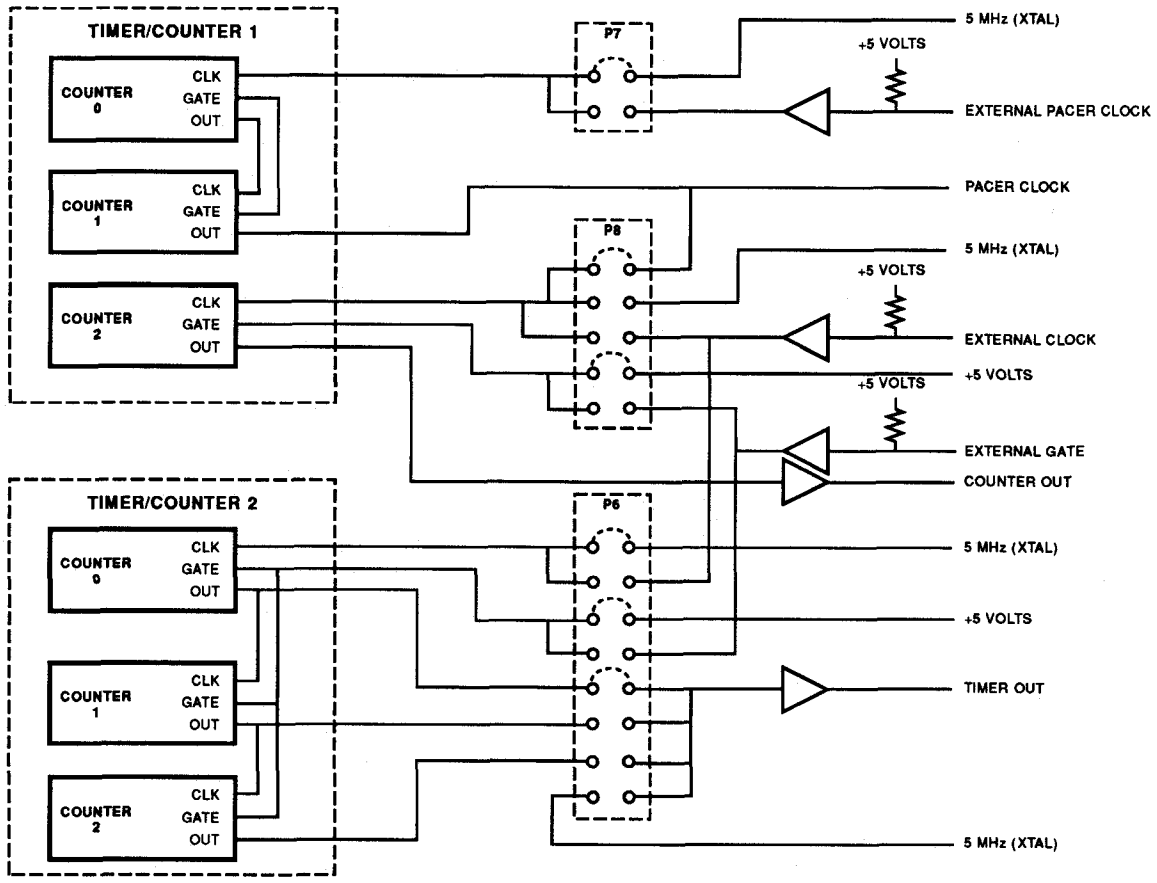


Fig. 3-2 — 8254 Programmable Interval Timer Circuits Block Diagram

Digital I/O

Eight digital input and eight digital output lines can be used to transfer data between the computer and external devices. Data transfers through the digital I/O lines are independent of other board functions. The input lines have pull-up resistors. All 16 lines are available at the external I/O connector.

CHAPTER 4

BOARD OPERATION AND PROGRAMMING

This chapter shows you how to program and use your AD3700 board. It provides a complete description of the I/O map, a detailed description of programming operations and operating modes, and flow diagrams to aid you in programming. The example programs included on the disk in your board package are listed at the end of this chapter. These programs, written in Turbo C, Turbo Pascal, and BASIC, include source code to simplify your applications programming.

Defining the I/O Map

The I/O map for the AD3700 is shown in Table 4-1 below. As shown, the board occupies 16 consecutive I/O port locations. The base address (designated as BA) can be selected using DIP switch S1, located on the top edge at the rear of the board (furthest from I/O connector P2), as described in Chapter 1, *Board Settings*. This switch can be accessed without removing the board from the computer. The following sections describe the register contents of each address used in the I/O map.

Table 4-1 — AD3700 I/O Map

Register Description	Read Function	Write Function	Address * (Decimal)
Digital I/O	Read 8 digital input lines	Program 8 digital output lines	BA + 0
Channel/Conversion Mode Select	Read A/D channel & conversion mode settings	Program A/D channel & conversion mode	BA + 1
Scan Channel Range Select	Read number of channels to be active	Program number of channels in scan cycle	BA + 2
Status/Clear FIFO	Read status word	Clear FIFO	BA + 3
Read Data/Start Convert	Read FIFO data, MSB & LSB	Start A/D conversion	BA + 4
Clear DMA Done	Reserved	Clear DMA done bit	BA + 5
IRQ/DMA Select	Read interrupt & DMA settings	Program interrupt source & channel select; DMA select	BA + 6
Clear Board	Reserved	Clear (reset) board	BA + 7
TC1 Counter 0 (Used for pacer clock)	Read count value	Load count register	BA + 8
TC1 Counter 1 (Used for pacer clock)	Read count value	Load count register	BA + 9
TC1 Counter 2 (Available for external use)	Read count value	Load count register	BA + 10
TC1 Control Word	Reserved	Program counter mode	BA + 11
TC2 Counter 0	Read count value	Load count register	BA + 12
TC2 Counter 1	Read count value	Load count register	BA + 13
TC2 Counter 2	Read count value	Load count register	BA + 14
TC2 Control Word	Reserved	Program counter mode	BA + 15
* BA = Base Address			

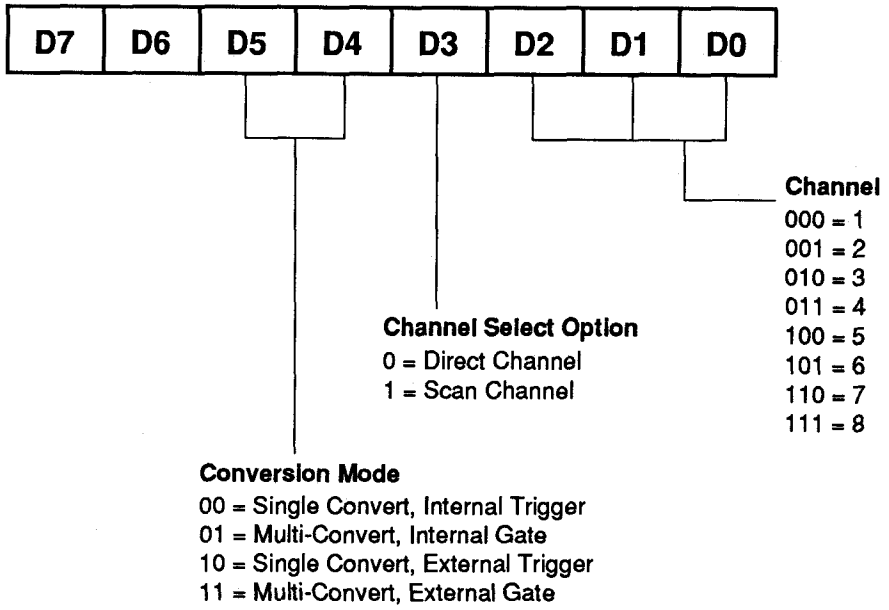
BA + 0: Digital I/O (Read/Write)

Transfers the 8-bit digital input and digital output data between the board and an external device. A read transfers data from the external device through P2 onto the board where it can be placed in user memory; a write transfers data from the board to an external device.

In7	In6	In5	In4	In3	In2	In1	In0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0

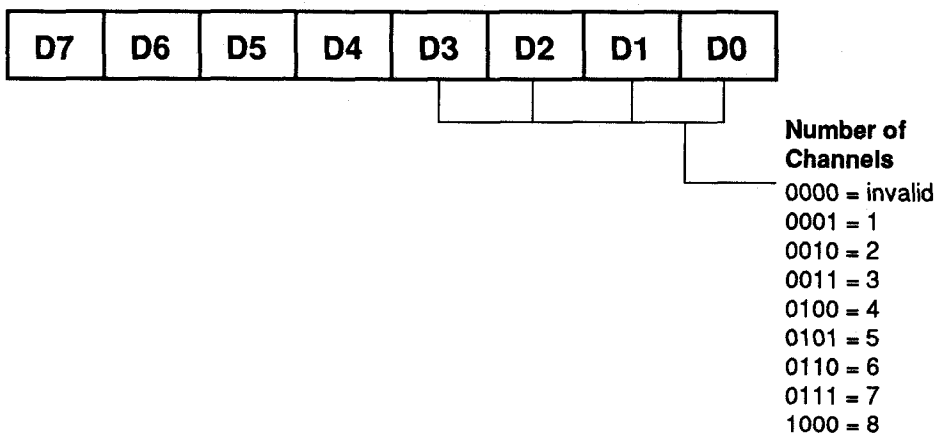
BA + 1: Channel/Conversion Mode Select (Read/Write)

Programs the analog input channel, A/D conversion mode, and the channel select option. The conversion modes and channel select options are detailed later in this chapter under *Programming the AD3700*. D6 and D7 are not used. Reading this register shows you the current settings.



BA + 2: Scan Channel Range Select (Read/Write)

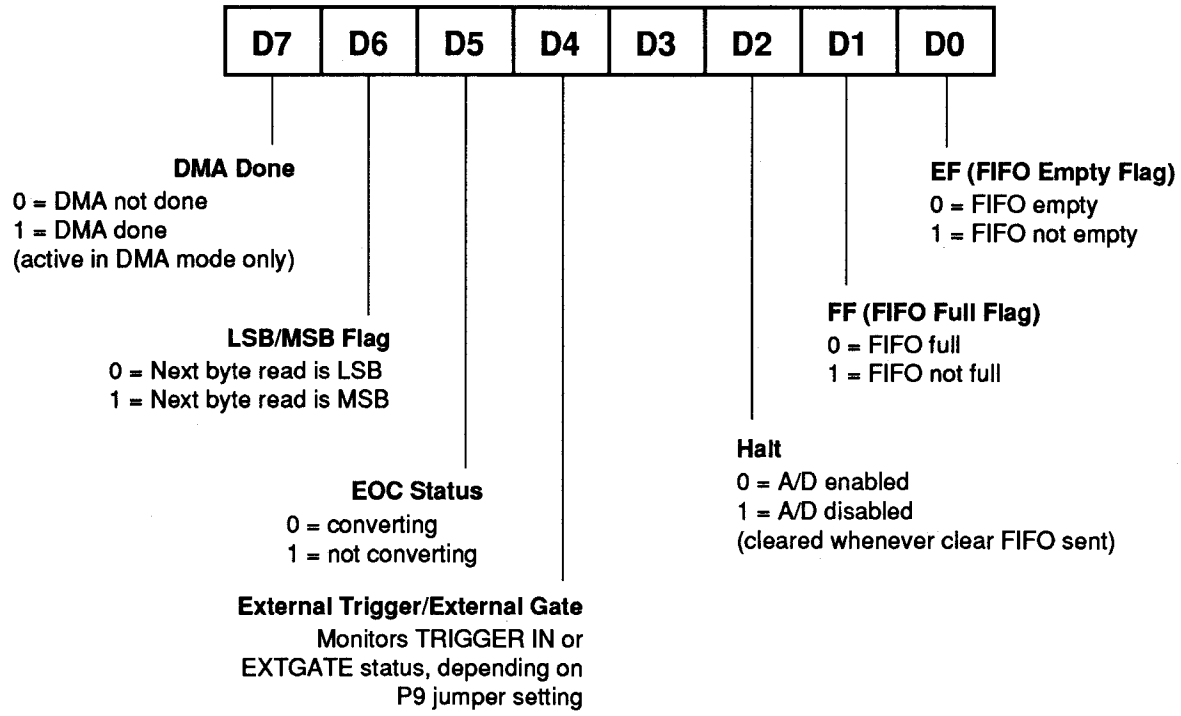
Programs the number of channels to be activated for a scan cycle. This number, coupled with the analog input channel select programmed at BA + 1, establishes the sequence for the channel scan. For example, if you want to do a scan of three channels starting with channel 3 (analog input channel select), one cycle will convert the input voltages at channels 3, 4, and 5.



BA + 3: Read Status/Clear FIFO (Read/Write)

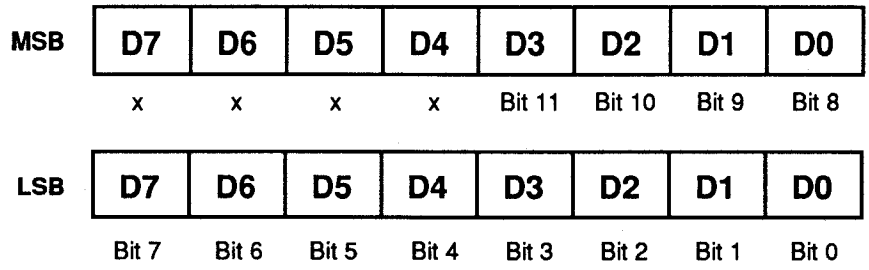
A read provides the eight-bit status word defined below. The A/D converter HALT bit, D2, is set to 1, stopping A/D conversions whenever the FIFO is full or half-full, depending on the setting of the jumper on P4. This is the only way conversions can be stopped in the Multi-Convert modes. D1 is the FIFO full flag. This flag is set to 0 whenever the FIFO is full. D4 shows the status of either the external trigger in signal (P2-39) or the external gate signal (P2-46), depending on the setting of jumper P6.

A write clears the FIFO (data written is irrelevant). When the FIFO is cleared using BA + 3, the FIFO empties out all data, sets the FIFO empty flag, EF, low, and sets the FIFO full flag high. Clearing the FIFO also sets the LSB/MSB flag to 1 so that the next byte of data read is the MSB, and clears the HALT bit, enabling A/D conversions.



BA + 4: Read FIFO Data/Start Conversion (Read/Write)

Two successive reads provide the MSB and LSB of the A/D conversion, as defined below. A write starts a conversion (data written is irrelevant). Note that the MSB line and LSB line toggle with each read. Bit 6 in the Status word (BA + 3) shows which byte is next.

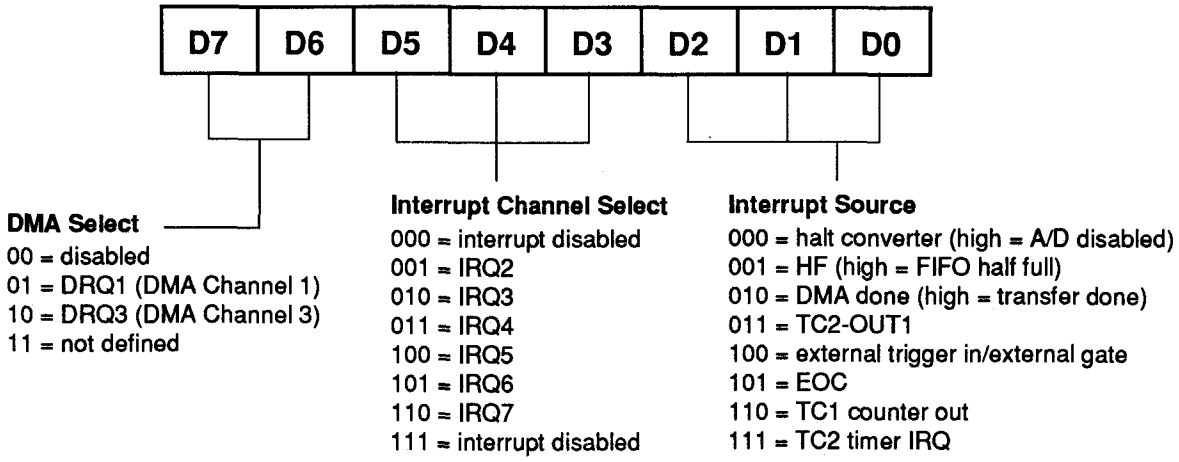


BA + 5: Clear DMA Done Bit (Write Only)

Writing to this address clears the DMA done bit at BA + 3, bit D7 (data written is irrelevant). This command lets you perform continuous DMA dumps of 64K from the FIFO into PC memory without losing any data while conversions are in progress.

BA + 6: IRQ/DMA Select (Read/Write)

Programs the interrupt source and channel, and the DMA transfer mode. Reading this register shows you the current settings.



BA + 7: Clear (Reset) Board (Write only)

A write to this location clears, or resets, the board (data written is irrelevant). This command resets all of the on-board registers to 0. It also initializes the A/D converter after power-up.

BA + 8: TC1 Counter 0 (Read/Write)

A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded. This counter is part of the 32-bit on-board pacer clock (TC1 counters 0 and 1).

BA + 9: TC1 Counter 1 (Read/Write)

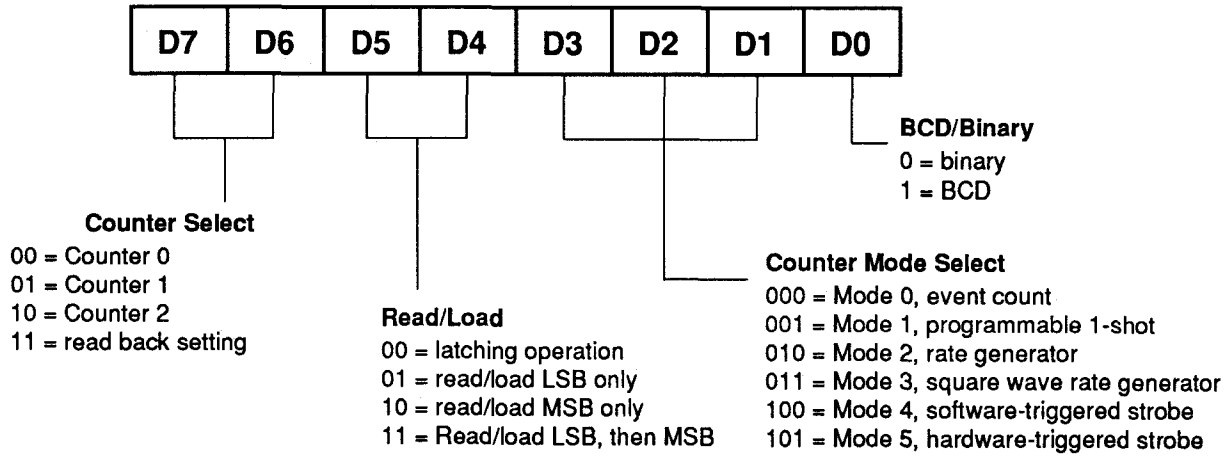
A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded. This counter is part of the 32-bit on-board pacer clock (TC1 counters 0 and 1).

BA + 10: TC1 Counter 2 (Read/Write)

A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded. This counter is user-configurable for counter applications.

BA + 11: TC1 Control Word (Write Only)

Accesses the TC1 control register to directly control the three TC1 counters.



BA + 12: TC2 Counter 0 (Read/Write)

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

BA + 13: TC2 Counter 1 (Read/Write)

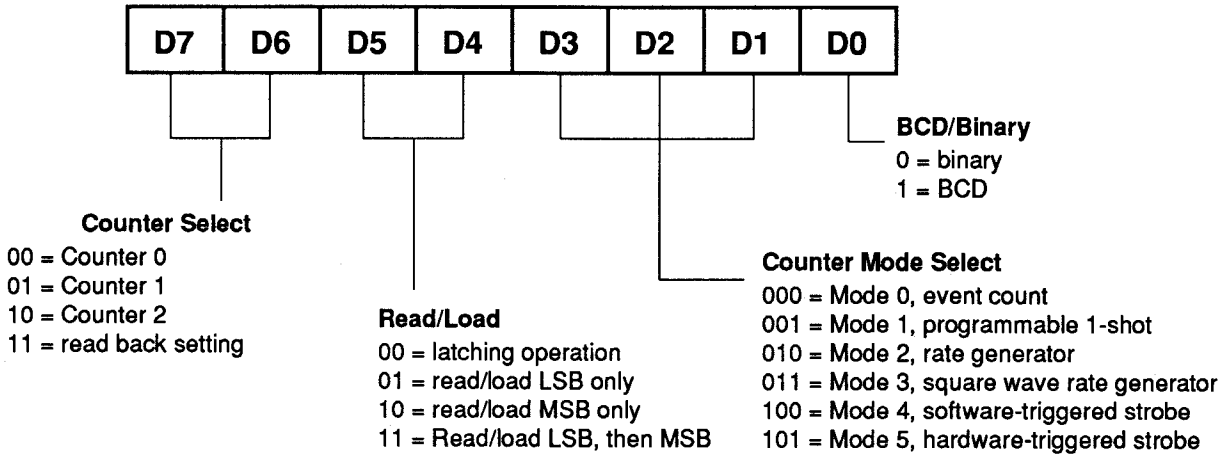
A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded. This counter is used for timer operations.

BA + 14: TC2 Counter 2 (Read/Write)

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

BA + 15: TC2 Control Word (Write Only)

Accesses the TC2 control register to directly control the three TC2 counters.



Programming the AD3700

This section gives you some general information about programming and the AD3700 board, and then walks you through the major AD3700 programming functions. These descriptions will help you as you use the example programs included with the board and the programming flow diagrams at the end of this chapter. All of the program descriptions in this section use decimal values unless otherwise specified.

The AD3700 is programmed by writing to and reading from the correct I/O port locations on the board. These I/O ports were defined in the previous section. Most high-level languages such as BASIC, Pascal, C, and C++, and of course assembly language, make it very easy to read/write these ports. The table below shows you how to read from and write to I/O ports using some popular programming languages.

Language	Read	Write
BASIC	Data = INP(Address)	OUT Address, Data
Turbo C	Data = inportb(Address)	outportb(Address, Data)
Turbo Pascal	Data := Port[Address]	Port[Address] := Data

Assembly	mov dx, Address
	in al, dx
		mov dx, Address
		mov al, Data
		out dx, al

In addition to being able to read/write the I/O ports on the AD3700, you must be able to perform a variety of operations that you might not normally use in your programming. The table below shows you some of the operators discussed in this section, with an example of how each is used with Pascal, C, and BASIC. Note that the modulus operator is used to retrieve the least significant byte (LSB) of a two-byte word, and the integer division operator is used to retrieve the most significant byte (MSB).

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

Many compilers have functions that can read/write either 8 or 16 bits from/to an I/O port. For example, Turbo Pascal uses **Port** for 8-bit port operations and **PortW** for 16 bits, Turbo C uses **inportb** for an 8-bit read of a port and **inport** for a 16-bit read. **Be sure to use only 8-bit operations with the AD3700!**

Clearing and Setting Bits in a Port

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Now that you know how to clear and set bits, we are ready to look at the programming steps for the AD3700 board functions.

A/D Conversions

The following paragraphs walk you through the programming steps for performing A/D conversions. Detailed information about the conversion modes and channel select options is presented in this section. You can follow these steps on the flow diagrams at the end of this chapter and in our example programs included with the board. In this discussion, BA refers to the base address.

• Clearing the Board

It is good practice to start your program by resetting the AD3700 board. You can do this by writing to the CLEAR BOARD port located at BA + 7. The actual value you write to this port is irrelevant. After writing to this port, you should pause several milliseconds and then clear the FIFO to remove any data placed there by the reset process.

• Clearing the FIFO

To clear the FIFO, write any value to the CLEAR FIFO port, located at BA + 3. Any data in the FIFO when this port is written to is lost.

• Selecting a Channel

To select a conversion channel or a starting channel for a scan of channels, you must assign values to bits 0 through 2 in the CHANNEL/CONVERSION MODE SELECT port at BA + 1. The table below shows you how to determine the bit settings. Note that if you do not want to change other settings also programmed through BA + 1, you must preserve them when you set the channel.

x	x	x	x	x	CH2	CH1	CH0

BA + 1

Channel	CH2	CH1	CH0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

• **Conversion Modes and Channel Select Options**

The AD3700 provides several triggering (conversion) modes and scan (channel select) options. Four conversion modes and two channel select options give you a variety of combinations of triggering and channel selection to meet just about any sampling requirement. This section describes the modes and options and includes a series of timing diagrams at the end so that you can see how they are implemented. The conversion mode and channel select option are set at port BA + 1.

- Conversion Modes/Triggering

Internal vs. external triggering. With internal triggering (also called software triggering), conversions are initiated by writing a value to the START CONVERT port at BA + 4 on the board. With external triggering, conversions are initiated by applying a high TTL signal to the external TRIGGER IN pin (P2-39). Any TTL signal can be used as a trigger source. In fact, you can use the TIMER OUT (P2-42) or COUNTER OUT (P2-44) as a trigger source.

Single convert, internal trigger. In this mode, a single specified channel is sampled whenever a value is written to the START CONVERT port, BA + 4. The active channel is the one specified in the CHANNEL/CONVERSION MODE SELECT port.

x	x	0	0	x	x	x	x	BA + 1

This is the easiest of all triggering modes. It can be used in a wide variety of applications, such as sample every time a key is pressed on the keyboard, sample with each iteration of a loop, or watch the system clock and sample every five seconds. See the SOFTTRIG sample program in C and Pascal and the SINGLE sample program in BASIC.

Multi-convert, internal gate. In this mode, conversions are continuously performed at the pacer clock rate. Sampling is initiated from software. To use this mode, you must program the pacer clock to run at the desired rate (see the pacer clock discussion later in this chapter).

x	x	0	1	x	x	x	x	BA + 1

This is the ideal mode for filling an array with data. Triggering is automatic, so your program is spared the chore of monitoring the pacer clock to determine when to sample. See the MULTI sample program in C and Pascal.

Single convert, external trigger. In this mode, a single conversion is initiated by the rising edge of an external trigger pulse.

x	x	1	0	x	x	x	x	BA + 1

This mode is implemented when an external device is used to determine when to sample. See the EXTTRIG sample program in C and Pascal.

Multi-convert, external gate. In this mode, channels are sampled at the pacer clock rate. The pacer clock is gated on and off by the external trigger line. When the external trigger line is held high, sampling occurs at the pacer clock rate. When the line is low, sampling is halted.

x	x	1	1	x	x	x	x	BA + 1

This is an ideal mode when you want to acquire data for only as long as an external device holds the trigger high. See the MULTGATE sample program in C and Pascal.

- Channel Select Options/Scans

Direct channel. In this option, the channel specified in the CHANNEL/CONVERSION MODE SELECT port is sampled each time a trigger is applied.

x	x	x	x	0	x	x	x	BA + 1

Use the direct channel option when you only need to sample from one channel or if the order of channels to be sampled is unknown or not consecutive.

Scan channel. In this option, the channel from which to sample is automatically incremented after a conversion is complete. The scan starts at the channel specified in the CHANNEL/CONVERSION MODE SELECT port. After converting channel 8, the AD3700 returns to channel 1.

x	x	x	x	1	x	x	x	BA + 1

Use the scan channel option when you want to sample from all eight channels in consecutive order. Since the channel counter is automatically incremented, it is faster (and easier) than using the direct scan option and setting the channel for each conversion from software.

- Timing Diagrams

The following timing diagrams show how each of the eight possible conversion mode/channel select option combinations are implemented by the A/D converter and associated circuitry. Figures 4-1 and 4-2 show you the *Single Convert, Internal Trigger* mode timing; Figures 4-3 and 4-4 show you the *Multi-Convert, Internal Gate* mode timing; Figures 4-5 and 4-6 show you the *Single Convert, External Trigger* mode timing; and Figures 4-7 and 4-8 show you the *Multi-Convert, External Gate* mode timing.

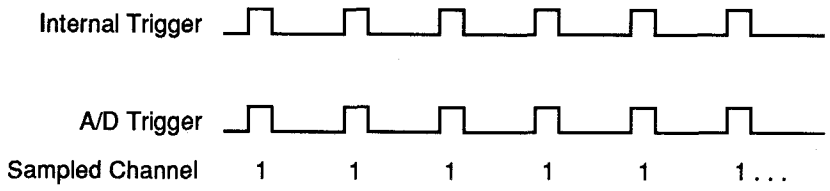


Fig. 4-1 — Timing Diagram, Single Convert, Internal Trigger/Direct Channel

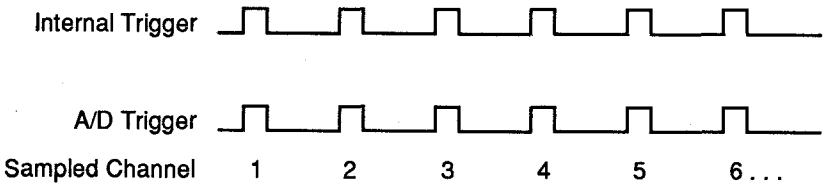


Fig. 4-2 — Timing Diagram, Single Convert, Internal Trigger/Scan Channel

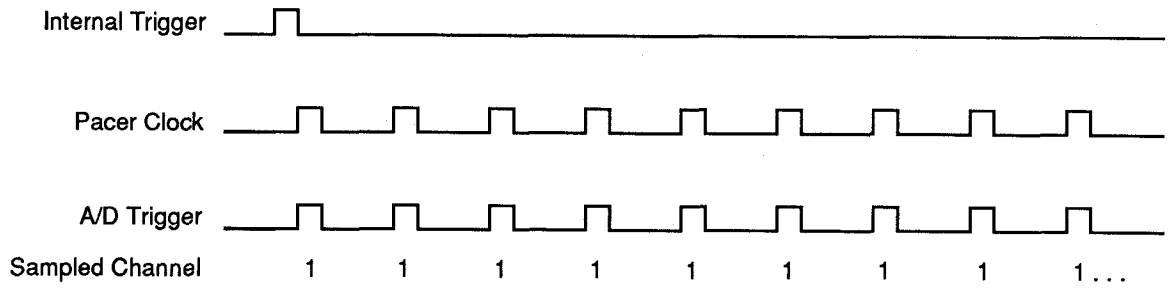


Fig. 4-3 — Timing Diagram, Multi-Convert, Internal Gate/Direct Channel

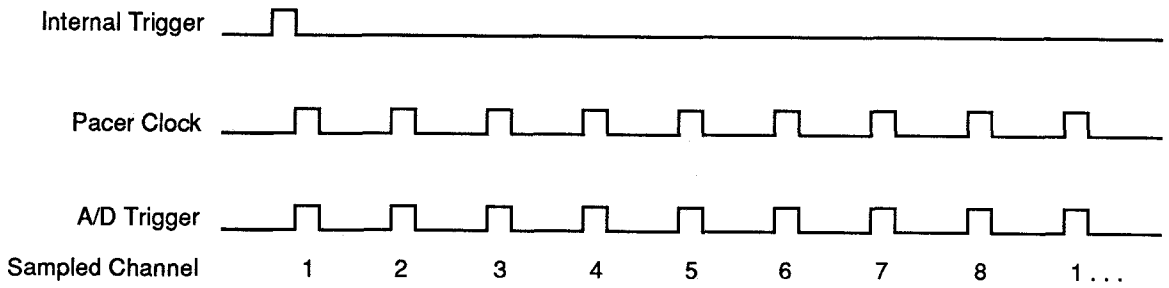


Fig. 4-4 — Timing Diagram, Multi-Convert, Internal Gate/Scan Channel

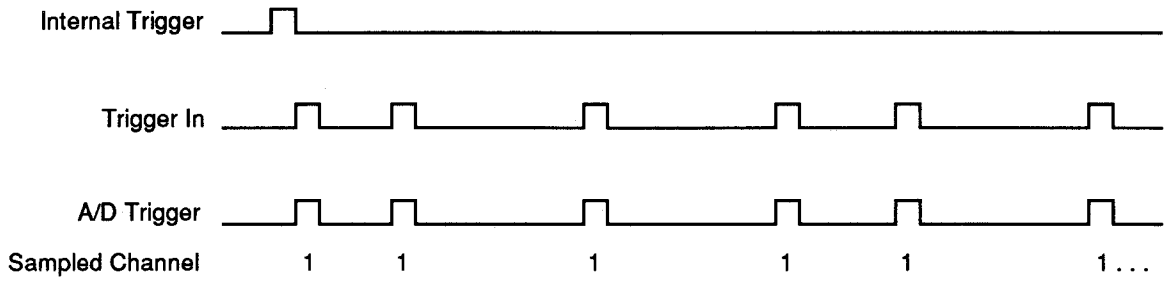


Fig. 4-5 — Timing Diagram, Single Convert, External Trigger/Direct Channel

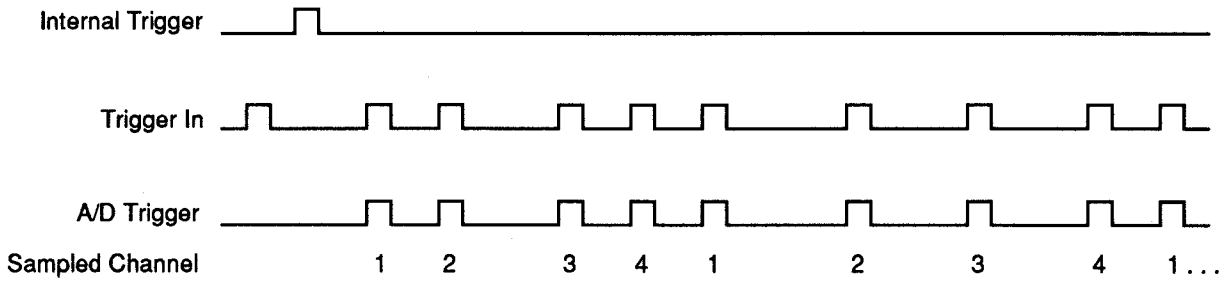


Fig. 4-6 — Timing Diagram, Single Convert, External Trigger/Scan Channel

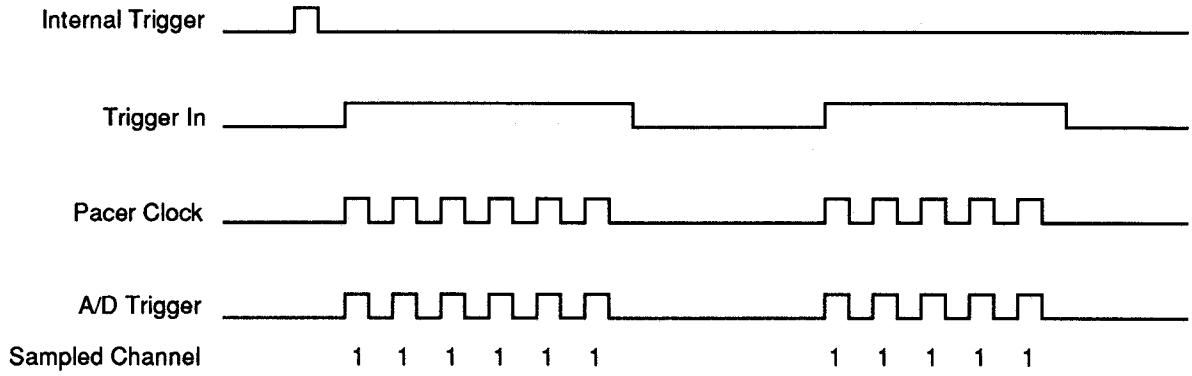


Fig. 4-7 — Timing Diagram, Multi-Convert, External Gate/Direct Channel

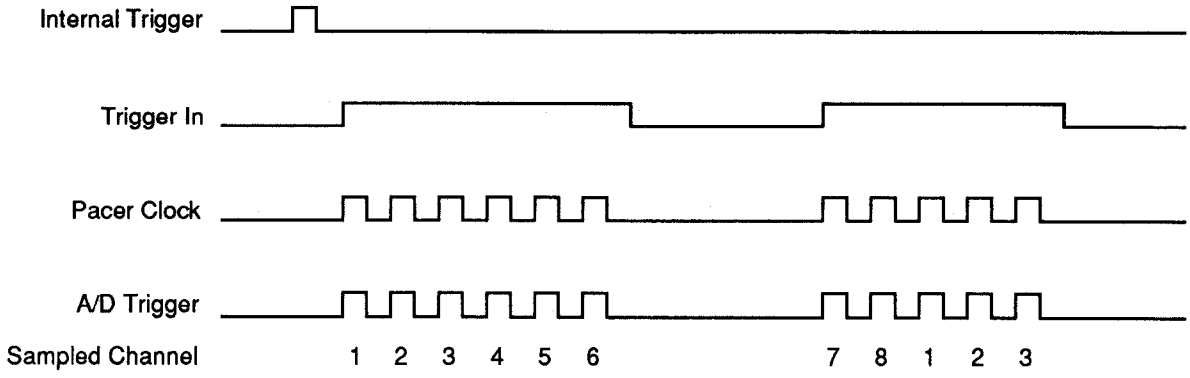


Fig. 4-8 — Timing Diagram, Multi-Convert, External Gate/Scan 8 Channels

• Starting an A/D Conversion

Whether you are using internal triggers, external triggers, single convert or multi-convert, you must start the conversion process by writing to the START CONVERT port at BA + 4. The value you write is irrelevant. For single conversion scan options, you must write to this port to initiate *every* conversion. In the multi-conversion modes, you need to write to this port only once to start the conversion cycle.

• Monitoring Conversion Status (EF Flag or End-of-Convert)

The A/D conversion status can be monitored through the FIFO empty (EF) flag or through the end-of-convert (EOC) bit in the STATUS port at BA + 3. Typically, you will want to monitor the EF flag for a transition from low to high. This tells you that a conversion is complete and data has been placed in the FIFO. The EOC line is available for monitoring conversion status in special applications.

• Halting Conversions

In the single convert modes, a single conversion is performed and the board waits for another START CONVERT command. In the multi-convert modes, conversions are halted when the FIFO is full. The HALT bit, bit 2 of the Status word (BA + 3), is set when the FIFO is full, disabling the A/D converter. If you want to stop execution in the middle of a run, you can send a CLEAR BOARD command by writing to BA + 7. However, if you do this, note that the contents of the FIFO will be lost.

• Reading the Converted Data

Two successive reads of port BA + 4 provide the MSB and LSB of the 12-bit A/D conversion in the format defined in the I/O map section at the beginning of this chapter. The MSB line and LSB line toggle with each read. The MSB must always be read first, followed by the LSB. Bit 6 of the Status word (BA + 3) shows which byte is next. This bit is set whenever a FIFO CLEAR command is issued so that the first byte read is the MSB.

The output code and the resolution of the conversion vary, depending on the input voltage range selected. Bipolar conversions are in twos complement form, and unipolar conversions are straight binary. When a bipolar value is read, you must first convert the result to straight binary and then calculate the voltage. The conversion formula is simple: for values greater than 2047, you must subtract 4096 from the value to get the sign of the voltage. For example, if your output is 2048, you subtract 4096: 2048 - 4096 = -2048. This result corresponds to -5 volts or -10 volts, depending on your binary range. For values of 2047 or less, you simply convert the result. The key digital codes and their input voltage values are given for each range in the following three tables.

A/D Bipolar Code Table (±5V; twos complement)

Input Voltage	Output Code
+4.998 volts	MSB 0111 1111 1111 LSB
+2.500 volts	0100 0000 0000
0 volts	0000 0000 0000
-.00244 volts	1111 1111 1111
-5.000 volts	1000 0000 0000
1 LSB = 2.44 millivolts	

A/D Bipolar Code Table (±10V; twos complement)	
Input Voltage	**Output Code**
+9.995 volts	MSB 0111 1111 1111 LSB
+5.000 volts	0100 0000 0000
0 volts	0000 0000 0000
-.00488 volts	1111 1111 1111
-10.000 volts	1000 0000 0000
1 LSB = 4.88 millivolts	

A/D Unipolar Code Table (0 to +10V; straight binary)	
Input Voltage	**Output Code**
+9.99756 volts	MSB 1111 1111 1111 LSB
+5.00000 volts	1000 0000 0000
0 volts	0000 0000 0000
1 LSB = 2.44 millivolts	

• Programming the Pacer Clock

Two 16-bit timer/counters in the 8254 Timer/Counter TC1 are cascaded to form the on-board pacer clock, shown in Figure 4-9. When you want to use the pacer clock for continuous A/D conversions, you must program the clock rate. To find the value you must load into the clock to produce the desired rate, you first have to calculate the value of Divider 1 (TC1 Counter 0) and Divider 2 (TC1 Counter 1) shown in the diagram. The formulas for making this calculation are as follows:

Pacer clock frequency = Clock Source Frequency/(Divider 1 x Divider 2)
Divider 1 x Divider 2 = Clock Source Frequency/Pacer Clock Frequency

To set the pacer clock frequency at 200 kHz using the on-board 5-MHz clock source, this equation becomes:

Divider 1 x Divider 2 = 5 MHz/200 kHz ---> 25 = 5 MHz/200 kHz

After you determine the value of Divider 1 x Divider 2, you then divide the result by the least common denominator. The least common denominator is the value that is loaded into Divider 1, and the result of the division, the quotient, is loaded into Divider 2. In our example above, the least common denominator is 5, so Divider 1 equals 5, and Divider 2 equals 25/5, or 5 also. The table with the diagram lists some common pacer clock frequencies and the counter settings (using the on-board 5-MHz clock source).

After you calculate the decimal value of each divider, you can convert the result to a hex value if it is easier for you when loading the count into the 16-bit counter.

To set up the pacer clock on the AD3700, follow these steps:

1. Select a clock source (the 5-MHz on-board clock or and external clock source).
2. Program TC1, Counter 0 for Mode 2 operation.
3. Program TC1, Counter 1 for Mode 2 operation.
4. Load Divider 1 LSB.
5. Load Divider 1 MSB.
6. Load Divider 2 LSB.
7. Load Divider 2 MSB.

Depending on your conversion mode, the counters start their countdown and the pacer clock starts running when a trigger occurs.

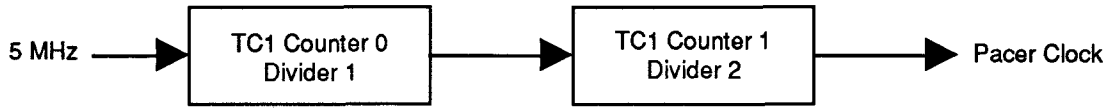


Fig. 4-9 — Pacer Clock Block Diagram

Pacer Clock	Divider 1 decimal / (hex)	Divider 2 decimal / (hex)
200 kHz	5 / (0005)	5 / (0005)
100 kHz	2 / (0002)	25 / (0019)
50 kHz	2 / (0002)	50 / (0032)
10 kHz	2 / (0002)	250 / (00FA)
1 kHz	2 / (0002)	2500 / (09C4)
100 Hz	2 / (0002)	25000 / (61A8)

• **Interrupts**

- What Is an Interrupt?

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

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

Your AD3700 board can interrupt the processor when a variety of conditions are met, such as FIFO not empty, timer countdown finished, and others. By using these interrupts, you can write software that effectively deals with real world events.

- Interrupt Request Lines

To allow different peripheral devices to generate interrupts on the same computer, the PC bus has eight different interrupt request (IRQ) lines. A transition from low to high on one of these lines generates an interrupt request which is handled by the PC's interrupt controller. The interrupt controller checks to see if interrupts are to be acknowledged from that IRQ and, if another interrupt is already in progress, it decides if the new request should supersede the one in progress or if it has to wait until the one in progress is done. This prioritizing allows an interrupt to be interrupted if the second request has a higher priority. The priority level is based on the number of the IRQ; IRQ0 has the highest priority, IRQ1 is second-highest, and so on through IRQ7, which has the lowest. Many of the IRQs are used by the standard system resources. IRQ0 is used by the system timer, IRQ1 is used by the keyboard, IRQ3 by COM2, IRQ4 by COM1, and IRQ6 by the disk drives. Therefore, it is important for you to know which IRQ lines are available in your system for use by the AD3700 board.

- 8259 Programmable Interrupt Controller

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

- Interrupt Mask Register (IMR)

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

I/O Port 21H

For all bits:
0 = IRQ unmasked (enabled)
1 = IRQ masked (disabled)

- End-of-Interrupt (EOI) Command

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

- What Exactly Happens When an Interrupt Occurs?

Understanding the sequence of events when an interrupt is triggered is necessary to properly write software interrupt handlers. When an interrupt request line is driven high by a peripheral device (such as the AD3700), the interrupt controller checks to see if interrupts are enabled for that IRQ, and then checks to see if other interrupts are active or requested and determines which interrupt has priority. The interrupt controller then interrupts the processor. The current code segment (CS), instruction pointer (IP), and flags are pushed on the stack for storage, and a new CS and IP are loaded from a table that exists in the lowest 1024 bytes of memory. This table is referred to as the interrupt vector table and each entry is called an interrupt vector. Once the new CS and IP are loaded from the interrupt vector table, the processor begins executing the code located at CS:IP. When the interrupt routine is completed, the CS, IP, and flags that were pushed on the stack when the interrupt occurred are now popped from the stack and execution resumes from the point where it was interrupted.

- Using Interrupts in Your Programs

Adding interrupts to your software is not as difficult as it may seem, and what they add in terms of performance is often worth the effort. Note, however, that although it is not that hard to use interrupts, the smallest mistake will often lead to a system hang that requires a reboot. This can be both frustrating and time-consuming. But, after a few tries, you'll get the bugs worked out and enjoy the benefits of properly executed interrupts. In addition to reading the following paragraphs, study the INTRPTS source code included on your AD3700 program disk for a better understanding of interrupt program development.

- Writing an Interrupt Service Routine (ISR)

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

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

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

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

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

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

The second major concern when writing your ISR is to make it as short as possible in terms of execution time. Spending long periods of time in your ISR may mean that other important interrupts are being ignored. Also, if you spend too long in your ISR, it may be called again before you have completed handling the first run. This often leads to a hang that requires a reboot.

Your ISR should have this structure:

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

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

In C:

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

In Pascal:

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

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

The next step after writing the ISR is to save the startup state of the interrupt mask register and the interrupt vector that you will be using. The IMR is located at I/O port 21H. The interrupt vector you will be using is located in the interrupt vector table which is simply an array of 256-bit (4-byte) pointers and is located in the first 1024 bytes of memory (Segment = 0, Offset = 0). You can read this value directly, but it is a better practice to use DOS function 35H (get interrupt vector). Most C and Pascal compilers provide a library routine for reading the value of a vector. The vectors for the hardware interrupts are vectors 8 through 15, where IRQ0 uses vector 8, IRQ1 uses vector 9, and so on. Thus, if the AD3700 will be using IRQ3, you should save the value of interrupt vector 11.

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

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

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

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

– Restoring the Startup IMR and Interrupt Vector

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

- Common Interrupt Mistakes

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

• Data Transfers Using DMA

Direct Memory Access (DMA) transfers data between a peripheral device and PC memory without using the processor as an intermediate. Bypassing the processor in this way allows very fast transfer rates. All PCs contain the necessary hardware components for accomplishing DMA. However, software support for DMA is not included as part of the BIOS or DOS, leaving you with the task of programming the DMA controller yourself. With a little care, such programming can be successfully and efficiently achieved.

The following discussion is based on using the DMA controller to get data from a peripheral device and write it to memory. The opposite can also be done; the DMA controller can read data from memory and pass it to a peripheral device. There are a few minor differences, mostly concerning programming the DMA controller, but in general the process is the same.

The following steps are required when using DMA:

1. Choose a DMA channel.
2. Allocate a buffer.
3. Calculate the page and offset of the buffer.
4. Set the DMA page register.
5. Program the DMA controller.
6. Program the device generating data (AD3700).
7. Wait until DMA is complete.
8. Disable DMA.

Each step is detailed in the following paragraphs.

- Choosing a DMA Channel

There are a number of DMA channels available on the PC for use by peripheral devices. The AD3700 can use either DMA channel 1 or DMA channel 3. You can arbitrarily choose one or the other; in most cases either choice is fine. Occasionally though, you will have another peripheral device (for example, a tape backup or Bernoulli drive) that also uses the DMA channel you have selected. This will certainly cause erratic results and can be hard to detect. The best approach to pinpoint this problem is to read the documentation for the other peripheral devices in your computer and try to determine which DMA channel each uses.

- Allocating a DMA Buffer

When using DMA, you must have a location in memory where the DMA controller will place data from the AD3700 board. This buffer can be either static or dynamically allocated. Just be sure that its location will not change while DMA is in progress. The following code examples show how to allocate buffers for use with DMA.

In Pascal:

```
Var Buffer : Array[1..10000] of Byte;   { static allocation }
-or-
Var Buffer : ^Byte;                     {dynamic allocation }
. . .
Buffer := GetMem(10000);
```

In C:

```
char Buffer[10000];                     /* static allocation */
-or-
char *Buffer;                           /* dynamic allocation */
. . .
Buffer = calloc(10000, 0);
```

In BASIC:

```
DIM BUFFER%(5000)
```

- Calculating the Page and Offset of a Buffer

Once you have a buffer into which to place your data, you must inform the DMA controller of the location of this buffer. This is a little more complex than it sounds because the DMA controller uses a **page**:offset memory scheme, while you are probably used to thinking about your computer's memory in terms of a **segment**:offset scheme. Paged memory is simply memory that occupies contiguous, **non-overlapping** blocks of memory, with each block being 64K (one page) in length. The first page (page 0) starts at the first byte of memory, the second page (page 1) starts at byte 65536, the third page (page 2) at byte 131072, and so on. A computer with 640K of memory has 10 pages of memory.

The DMA controller can write to (or read from) only one page without being reprogrammed. This means that the DMA controller has access to only 64K of memory at a time. If you program it to use page 3, it cannot use any other page until you reprogram it to do so.

When DMA is started, the DMA controller is programmed to place data at a specified offset into a specified page (for example, start writing at byte 512 of page 3). Each time a byte of data is written by the controller, the offset is automatically incremented so the next byte will be placed in the next memory location. The problem for you when programming these values is figuring out what the corresponding page and offset are for your buffer. Most compilers contain macros or functions that allow you to directly determine the segment and offset of a data structure, but not the page and offset. Therefore, you must calculate the page number and offset yourself. Probably the most intuitive way of doing this is to convert the segment:offset address of your buffer to a linear address and then convert that linear address to a page:offset address. The table below shows functions/macros for determining the segment and offset of a buffer.

Language	Segment	Offset
C	FP_SEG s = FP_SEG(&Buffer)	FP_OFF o = FP_OFF(&Buffer)
Pascal	Seg S := Seg(Buffer)	Ofs O := Ofs(Buffer)
BASIC	VARSEG S = VARSEG(BUFFER)	VARPTR O = VARPTR(BUFFER)

Once you've determined the segment and offset, multiply the segment by 16 and add the offset to give you the linear address. (Make sure you store this result in a long integer, or DWORD, or the results will be meaningless.) The page number is the quotient of the division of the linear address by 65536 and the offset into the page is the remainder of that division. Below are some programming examples for Pascal, C, and BASIC.

In Pascal:

```
Segment := SEG(Buffer);                  { get segment of buffer }
Offset := OFS(Buffer);                   { get offset of buffer }
Linear Address := Segment * 16 + Offset; { calculate a linear address }
Page := LinearAddress DIV 65536;         { determine page corresponding to this linear
                                           address }
PageOffset := LinearAddress MOD 65536;   { determine offset into the page }
```

In C:

```
segment = FP_SEG(&Buffer);               /* get segment of buffer */
offset = FP_OFS(&Buffer);                /* get offset of buffer */
linear_address = segment * 16 + offset;  /* calculate a linear address */
page = linear_address / 65536;           /* determine page corresponding to this linear
                                            address */
page_offset = linear_address % 65536;    /* determine offset into the page */
```

In BASIC:

```
S = VARSEG(BUFFER)
O = VARPTR(BUFFER)
LA= S * 16 + O
PAGE = INT(LA / 65536)
POFF = LA - (PAGE * 65536)
```

Beware! There is one big catch when using page-based addresses. The DMA controller cannot write properly to a buffer that 'straddles' a page boundary. A buffer straddles a page boundary if one part of the buffer resides in one page of memory while another part resides in the following page. The DMA controller cannot properly write to such a buffer because the DMA controller can only write to one page without reprogramming. When it reaches the end of the current page, it does not start writing to the next page. Instead, it starts writing back at the first byte of the current page. This can be disastrous if the beginning of the page does not correspond to your buffer. More often than not, this location is being used by the code portion of your program or the operating system, and writing data to it almost always causes bizarre behavior and an eventual system crash.

You must check to see if your buffer straddles a page boundary and, if it does, take action to prevent the DMA controller from trying to write to the portion that continues on the next page You can reduce the size of the buffer or try to reposition the buffer. However, this can be difficult when using large static data structures, and often, the only solution is to use dynamically allocated memory.

- Setting the DMA Page Register

Oddly enough, you do not inform the DMA controller directly of the page to be used. Instead, you put the page to be used into the DMA page register which is separate from the DMA controller, as shown in the table below. The location of this register depends on the DMA channel being used.

DMA Channel	Location of Page Register
1	83/(131)
3	82/(130)

- The DMA Controller

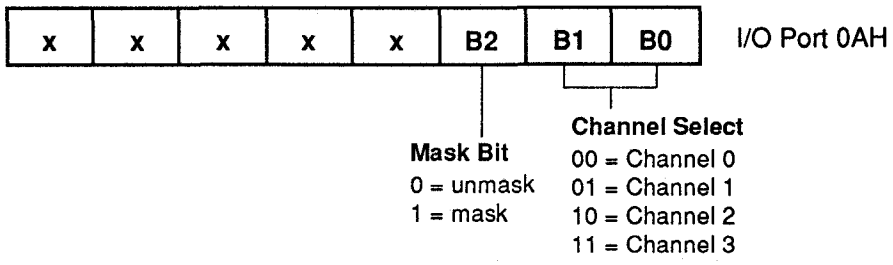
The DMA controller is a complex chip that occupies the first 16 bytes of the PC's I/O port space. A complete discussion on how it operates is beyond the scope of this manual; only relevant information is included here. The DMA controller is programmed by writing to the DMA registers in your PC. The table below lists these registers. Note that when you write 16-bit values to any of these registers (such as to the Count registers), you must write the LSB first, followed by the MSB.

Address hex/(decimal)	Register Description
02/(02)	Channel 1 Page Offset (write 2 bytes, LSB first)
03/(03)	Channel 1 Count (write 2 bytes, LSB first)
06/(06)	Channel 3 Page Offset (write 2 bytes, LSB first)
07/(07)	Channel 3 Count (write 2 bytes, LSB first)
0A/(10)	Single Mask Register
0B/(11)	Mode Register (write only)
0C/(12)	Clear Byte Pointer Flip-Flop (write only)

If you are using DMA channel 1, write your page offset and count to ports 02H and 03H; if you are using channel 3, write your page offset and count to ports 06H and 07H. The page offset is simply the offset that you calculated for your buffer (see discussion above). Count indicates the number of bytes that you want the DMA controller to transfer. Remember that each digitized sample from the AD3700 consists of 2 bytes, so the count that you write to the DMA controller should be equal to (the number of samples x 2) - 1. The single mask register and mode register are described below. The clear byte pointer sets an internal flip-flop on the DMA controller that keeps track of whether the LSB or MSB will be sent next to registers that accept both LSB and MSB. Ordinarily, you never **need** to write to this port, but it is a good habit to do so before programming the DMA controller. Writing any value to this port clears the flip-flop.

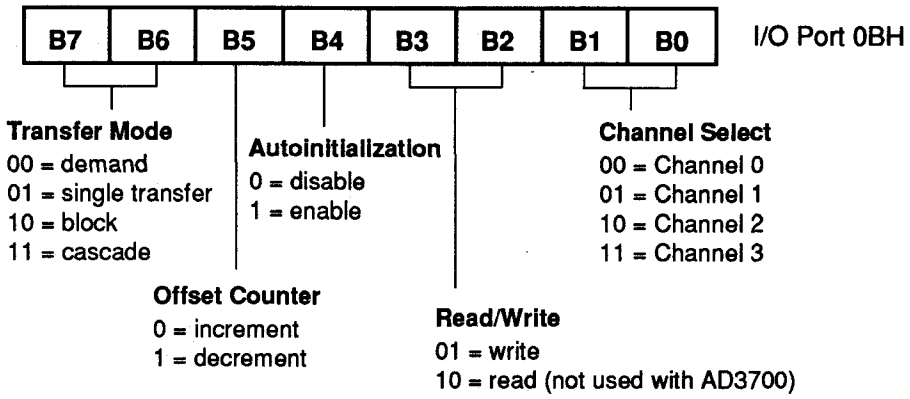
- DMA Single Mask Register

The DMA single mask register is used to enable or disable DMA on a specified DMA channel. You should mask (disable) DMA on the DMA channel you will be using while programming the DMA controller. After the DMA controller has been programmed and the AD3700 has been programmed to sample data, you can enable DMA by clearing the mask bit for the DMA channel you are using. You should manually disable DMA by setting the mask bit before exiting your program or, if for some reason, sampling is halted before the DMA controller has transferred all the data it was programmed to transfer. If you leave DMA enabled and it has not transferred all the data it was programmed to transfer, it will resume transfers the next time data appears in the AD3700 FIFO. This can spell disaster if your program has ended and the buffer has been reallocated to another application.



- DMA Mode Register

The DMA mode register is used to set parameters for the DMA channel you will be using. The read/write bits are self explanatory; the read mode cannot be used with the AD3700. Autoinitialization allows the DMA controller to automatically start over once it has transferred the requested number of bytes. Decrement means the DMA controller should decrement its offset counter after each transfer; the default is increment. We recommend that you use either the demand or single transfer mode when transferring data. The demand mode transfers data to the PC on demand. The single transfer mode forces the DMA controller to relinquish every other cycle so that the processor can take care of other tasks. We recommend that you do not use the block mode since it can tie up the processor and interfere with system operation.



- Programming the DMA Controller

To program the DMA controller, follow these steps:

1. Clear the byte pointer flip-flop.
2. Disable DMA on the channel you are using.
3. Write the DMA mode register to choose the DMA parameters.
4. Write the LSB of the page offset of your buffer.
5. Write the MSB of the page offset of your buffer.
6. Write the LSB of the number of bytes to transfer.
7. Write the MSB of the number of bytes to transfer.
8. Enable DMA on the channel you are using.

- Programming the AD3700 for DMA

Once you have set up the DMA controller, you must program the AD3700 for DMA. The following steps list this procedure:

1. Set the DMA channel bits in the IRQ DMA register.
2. Set the channel scan mode.
3. Set the triggering mode.
4. Program the pacer clock (if appropriate).
5. Start conversions.
6. Monitor the DMA done bit.

NOTE: If the DMA is set up in the single transfer mode, each DMA transfer will take two read cycles to complete. Therefore, when you run the AD3700 at 200 kHz in this mode, the DMA transfer rate cannot keep up with the board's conversion rate. Single transfers will run with the board up to about 120 kHz. Above 120 kHz, the FIFO can be used as a storage bin for the converted data until the DMA can transfer it to PC memory or the demand mode can be used.

- Monitoring for DMA Done

There are two ways to monitor for DMA done. The easiest is to poll the DMA done bit in the AD3700 status register (BA + 3). While DMA is in progress, the bit is clear (0). When DMA is complete, the bit is set (1). The second way to check is to use the DMA done signal to generate an interrupt. An interrupt can immediately notify your program that DMA is done and any actions can be taken as needed. Both methods are demonstrated in the sample C and Pascal programs, the polling method in the program named DMA and the interrupt method in DMASTR.

- Common DMA Problems

- Make sure that your buffer is large enough to hold all of the data you program the DMA controller to transfer.
- Check to be sure that your buffer does not straddle a page boundary.
- Remember that the number of bytes for the DMA controller to transfer is equal to twice the number of samples. This is because each sample is two bytes in size.
- If you terminate sampling before the DMA controller has transferred the number of bytes it was programmed for, be sure to disable DMA by setting the mask bit in the single mask register.

Timer/Counters

Two 8254 programmable interval timers, TC1 and TC2, each provide three 16-bit, 8-MHz timer/counters for timing and counting functions such as frequency measurement, event counting, and interrupts. Two of the timer/counters in TC1 are cascaded and used for the pacer clock, discussed earlier in this chapter. The remaining four timer/counters, Counter 2 in TC1 and Counters 0, 1, and 2, cascaded on TC2, are available for your use. Figure 4-10 shows the timer/counter circuitry.

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

One of two clock sources, the on-board 5-MHz crystal or the external clock (P2-45), can be jumpered as the clock input to TC1, Counter 2 and/or TC2's timer/counters. The clock source for the pacer clock is jumper-selectable for 5 MHz or the external pacer clock (P2-41). The diagram shows how these clock sources are connected to the timer/counters.

Two gate sources are available for enabling the timer/counters: a +5 volt source and an external gate source (P2-46). The same external gate source is connected to TC1, Counter 2 and the timer/counters in TC2.

The output from TC1, Counter 2 is available at the COUNTER OUT pin (P2-44) on the I/O connector where it can be used for interrupt generation, as an A/D trigger, or for counting functions. Any one of the three TC2 timer/counter outputs or the 5-MHz clock can be connected to the TIMER OUT pin (P2-42) on the I/O connector where it can be used for interrupt generation, as an A/D trigger, or for timing functions. These connections are jumper-selectable.

The timer/counters can be programmed to operate in one of six modes, depending on your application. For example, when measuring frequencies, the timer/counters in TC2 are set up for Mode 3 and TC1, Counter 2 is set up for Mode 0; when using it as an event counter, it is set up for Mode 0; and the pacer clock is set up for Mode 2. The following paragraphs briefly describe each mode.

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

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

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

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

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

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

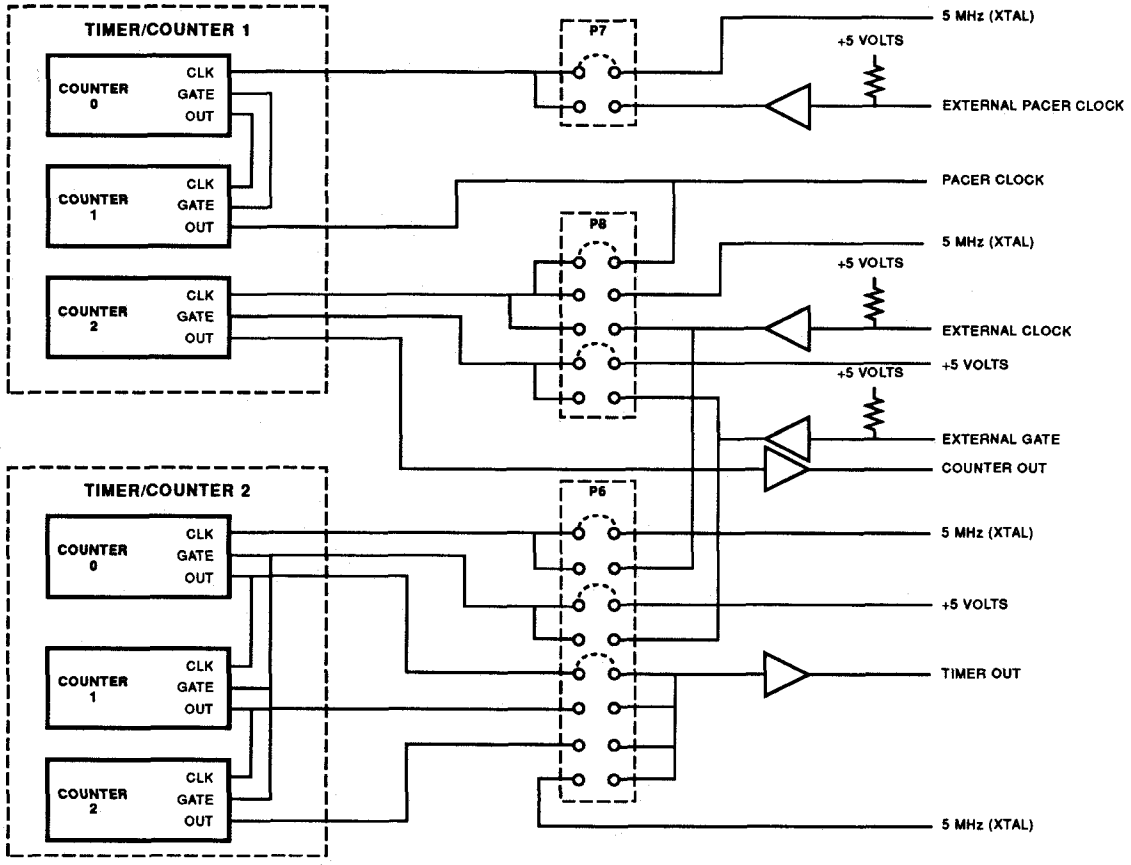


Fig. 4-10 — 8254 Programmable Interval Timer Circuits Block Diagram

Digital I/O

The eight digital input and eight digital output lines can be used to transfer data between the computer and external devices. The digital input lines have pull-up resistors as shown in Figure 4-11 so that they will be pulled high when the input source is disconnected. This is ideal to support switching applications.

The digital input data can be read at I/O port BA + 0 and transferred into PC memory. To output data, the desired value is written to I/O port BA + 0 and sent out to the external device connected to the digital output pins on external I/O connector P2.

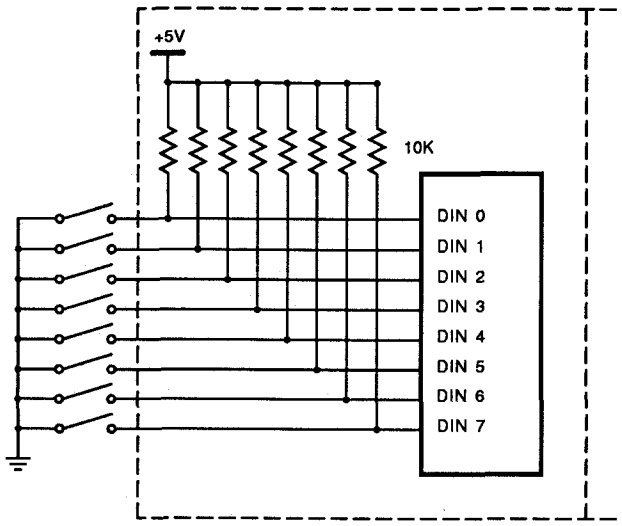


Fig. 4-11 — Digital Input Pull-up Resistors

Example Programs and Flow Diagrams

Included with the AD3700 is a set of example programs that demonstrate the use of many of the board's features. These examples are in written in C, Pascal, and BASIC. Also included is an easy-to-use menu-driven diagnostics program, 3700DIAG, which is especially helpful when you are first checking out your board after installation and when calibrating the board (Chapter 5).

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

C and Pascal Programs

These programs are source code files so that you can easily develop your own custom software for your AD3700 board. In the C directory, AD3700.H and AD3700.INC contain all the functions needed to implement the main C programs. H defines the addresses and INC contains the routines called by the main programs. In the Pascal directory, AD3700.PNC contains all of the procedures needed to implement the main Pascal programs.

Analog-to-Digital:

SOFTTRIG	Demonstrates how to use the software trigger mode for acquiring data.
EXTTRIG	Similar to SOFTTRIG except that an external trigger is used.
MULTI	Shows how to fill an array with data using a software trigger.
MULTGATE	Shows how to use the external trigger to gate multiple conversions.
SCANN	Demonstrates channel scanning of five channels

Timer/Counters:

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

Digital I/O:

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

Interrupts:

INTRPTS	Shows the bare essentials required for using interrupts.
INTSTR	A complete program showing interrupt-based streaming to disk.

DMA:

DMA	Demonstrates how to use DMA to acquire data to a memory buffer. Buffer can be written to disk and viewed with the included VIEWDAT program.
DMASTR	Demonstrates how to use DMA for disk streaming. Very high continuous acquisition rates can be obtained.

BASIC Programs

These programs are source code files so that you can easily develop your own custom software for your AD3700 board.

Analog-to-Digital:

SINGLE	Demonstrates how to use the single convert, internal trigger mode for acquiring data.
SCAN	Shows how to scan channels.

FIFO:

FIFO	Shows how to run the pacer clock and use the on-board FIFO.

DMA:

DMA	Shows how to take samples and transfer them to PC memory using DMA.

Flow Diagrams

The following paragraphs provide descriptions and flow diagrams for some of the AD3700's A/D conversion functions. These diagrams will help you to build your own custom applications programs.

• Single Convert Flow Diagram (Figure 4-12)

This flow diagram shows you the steps for taking a single sample on a selected channel. A sample is taken each time you send the Start Convert command. All of the samples will be taken on the same channel until you change the value in the CHANNEL/CONVERSION MODE SELECT register (BA + 1). Changing this value before each Start Convert command is issued lets you take the next reading from a different channel.

By changing the value in the CHANNEL/CONVERSION MODE SELECT register, you can change your program so that a sample is taken each time an external trigger occurs.

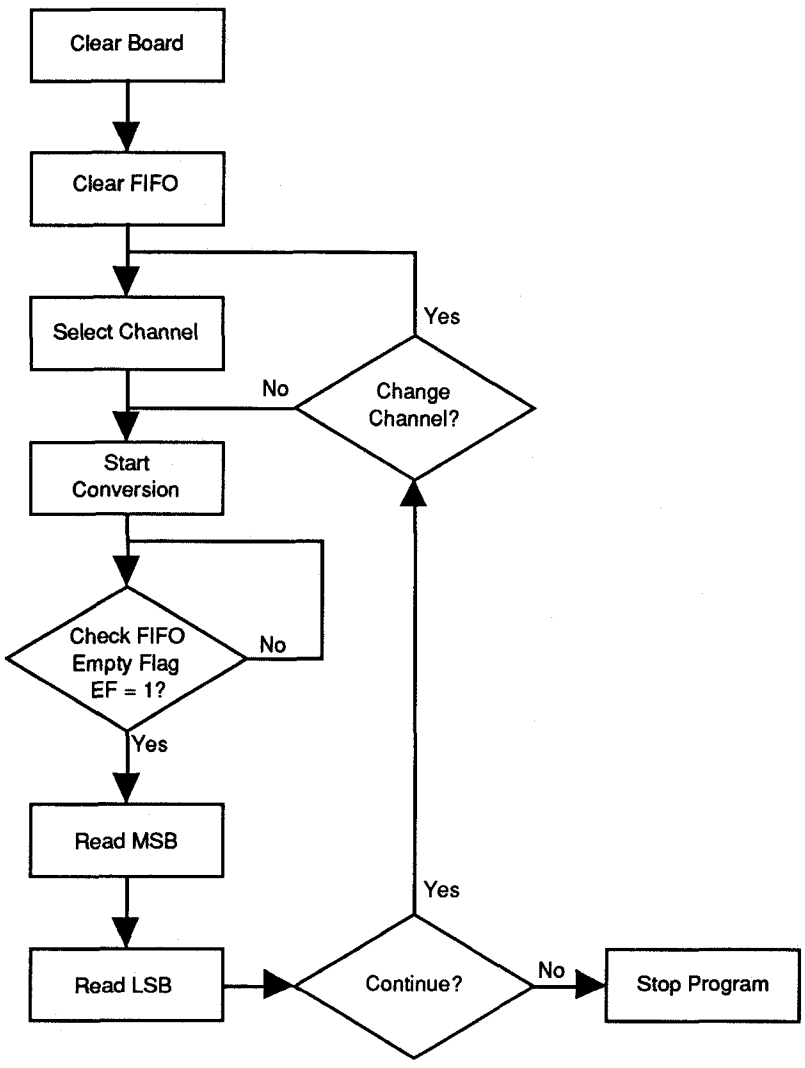


Fig. 4-12 — Single Convert Flow Diagram

• **FIFO Flow Diagram (Figure 4-13)**

This flow diagram shows you how to run the AD3700 from the pacer clock and use the on-board FIFO interface to store the converted data. You program the clock rate and take samples until the FIFO is full (FIFO full flag = 0). The samples are then read from the FIFO and displayed. A sample is taken each time the pacer clock generates a pulse. By using the pacer clock, the time interval between samples can be precisely set. The total number of samples taken depends on the size of the FIFO on your board. Each sample is sent to the FIFO in two 8-bit words, the MSB and the LSB. A 2K FIFO can hold 1024 samples, a 4K FIFO can hold 2048 samples, and an 8K FIFO can hold 4096 samples. The samples are taken on the channel specified in the bottom three bits of the CHANNEL/CONVERSION MODE SELECT register (BA + 1). By setting the channel select option bit in this register to *Scan Channel*, the converter will incrementally scan through all eight channels and store the data.

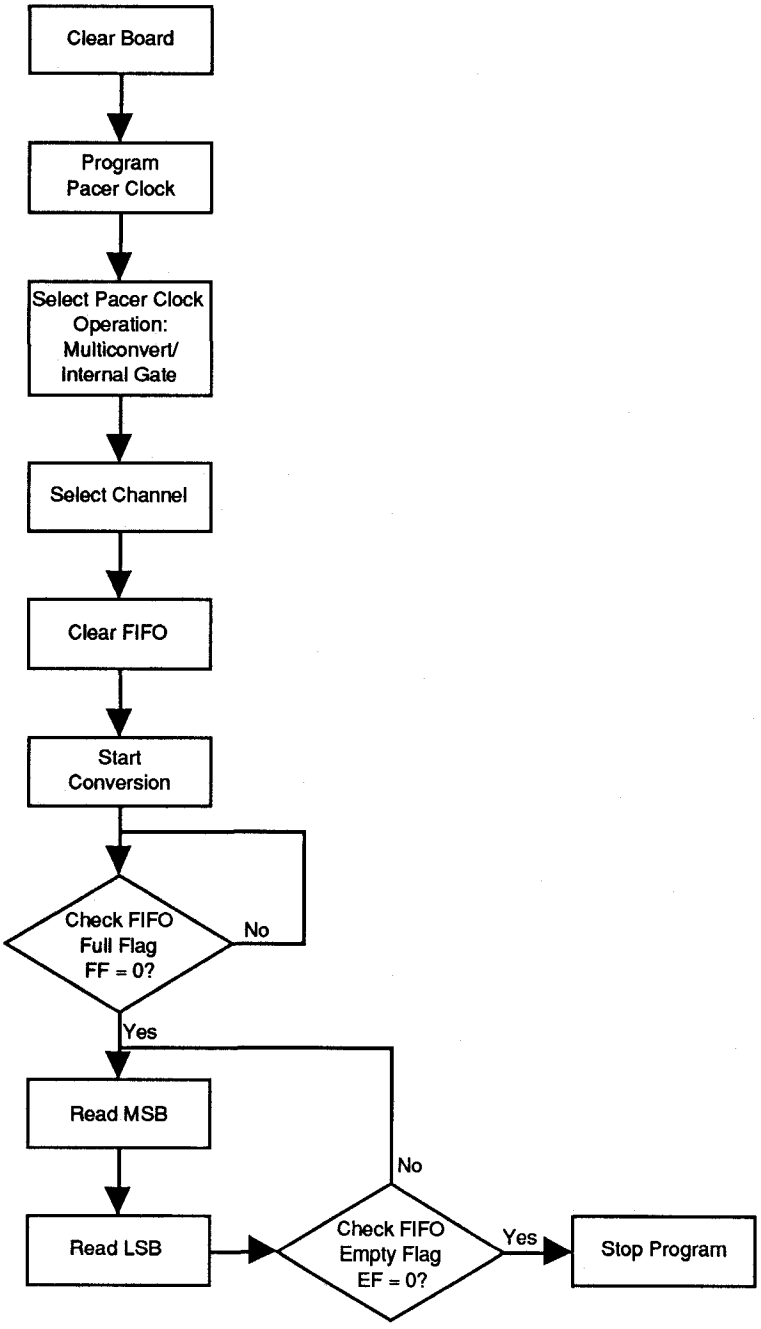


Fig. 4-13 — FIFO Flow Diagram

• **DMA Flow Diagram (Figure 4-14)**

This flow diagram shows you how to take samples and transfer the data directly into the computer's memory. You can use DMA channel 1 or 3 to transfer 1024 samples (2048 bytes) to the computer's memory.

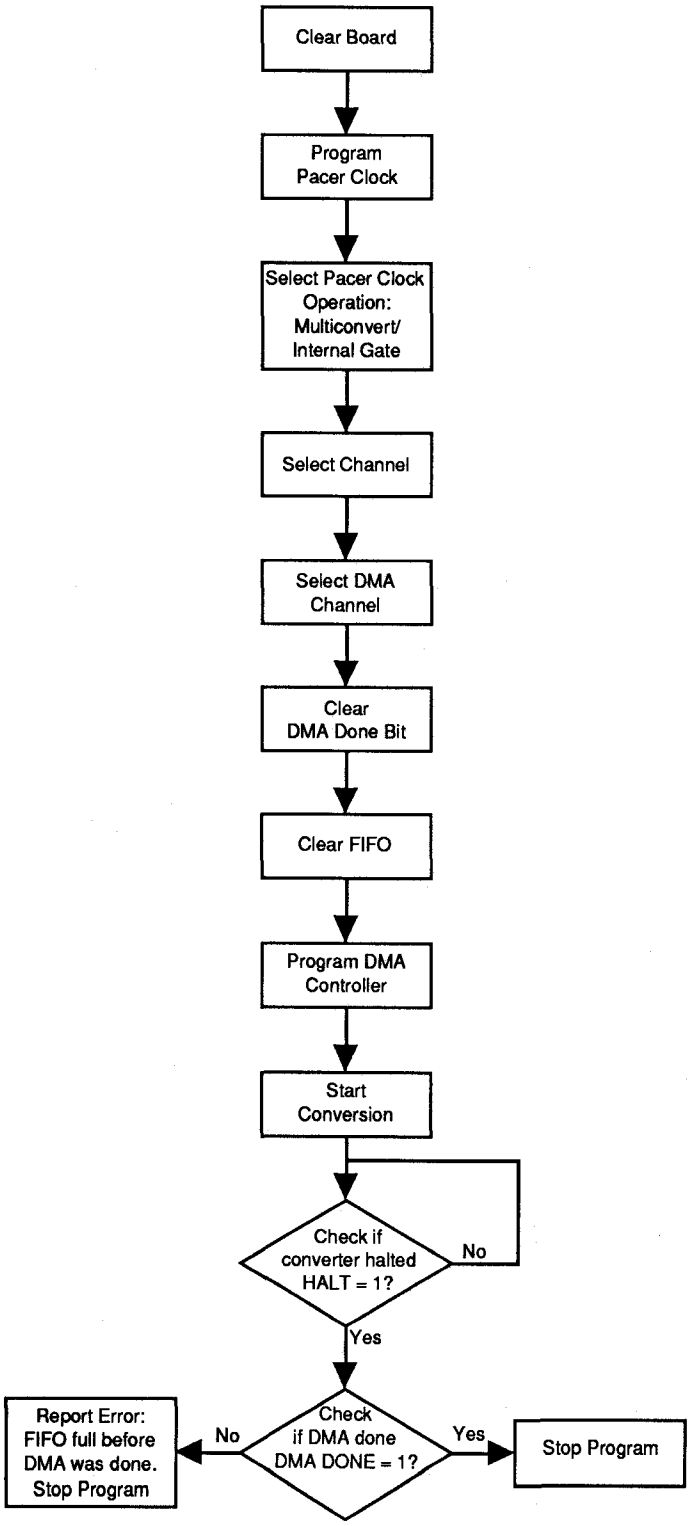


Fig. 4-14 — DMA Flow Diagram

• **Scan Flow Diagram (Figure 4-15)**

This flow diagram shows you how to take samples from a sequence of channels without selecting the channel each time a conversion is started.

By setting the channel select option bit in the CHANNEL/CONVERSION MODE SELECT register (BA + 1) to *Scan Channel* and setting the number of channels to be scanned at BA + 2, the converter will automatically increment the channel each time the Start Convert command is sent. The first channel sampled is the channel that is specified in the bottom three bits of the CHANNEL/CONVERSION MODE SELECT register. When the board increments through the number of channels programmed at BA + 2, it automatically starts over at the first channel in the sequence.

By changing the value in the CHANNEL/CONVERSION MODE SELECT register, you can change your program so that a sample is taken each time an external trigger occurs.

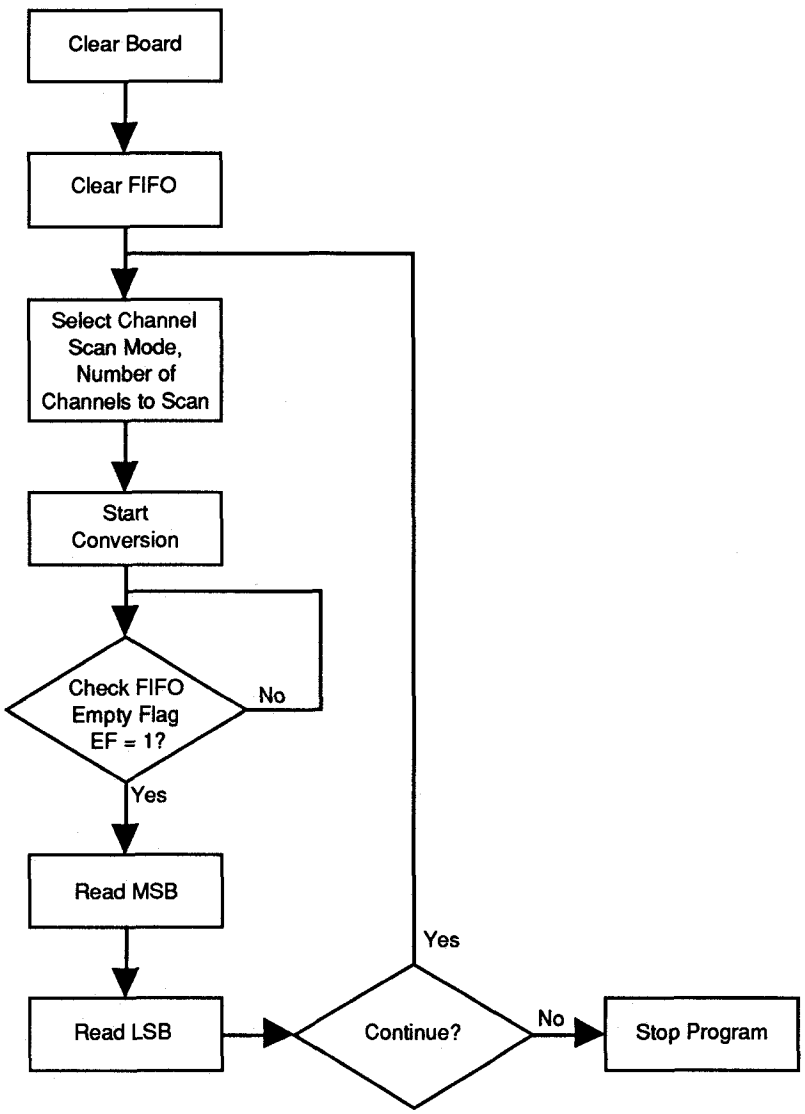


Fig. 4-15 — Scan Flow Diagram

• **Interrupts Flow Diagram (Figure 4-16)**

This flow diagram shows you how to program an interrupt routine for your AD3700. The diagram parallels the interrupts discussion included in the chapter. You can use this diagram in conjunction with the detailed text in this chapter to develop an interrupt program for your AD3700.

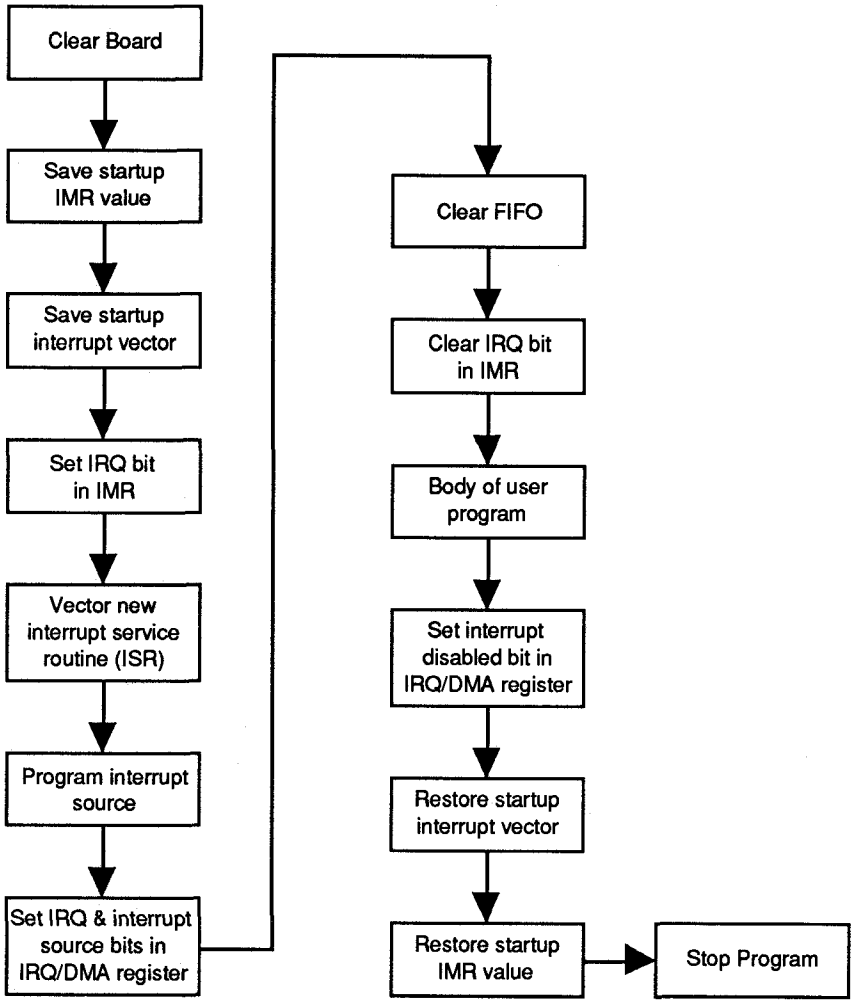


Fig. 4-16 — Interrupts Flow Diagram

CHAPTER 5

CALIBRATION

This chapter tells you how to calibrate the AD3700 using the 3700DIAG calibration program included in the example software package and the four trimpots (TR1 through TR3 and TR5) on the board. These trimpots calibrate the A/D converter gain and offset.

This chapter tells you how to calibrate the A/D converter gain and offset. The offset and full-scale performance of the board's A/D converter is factory-calibrated. Any time you suspect inaccurate readings, you can check the accuracy of your conversions using the procedure below, and make adjusts as necessary. Using the 3700DIAG diagnostics program is a convenient way to monitor conversions while you calibrate the board.

Calibration is done with the board installed in your PC. You can access the trimpots with the computer's cover removed. Power up the computer and let the board circuitry stabilize for 15 minutes before you start calibrating.

Required Equipment

The following equipment is required for calibration:

- Precision Voltage Source: -10 to +10 volts
- Digital Voltmeter: 5-1/2 digits
- Small Screwdriver (for trimpot adjustment)

While not required, the 3700DIAG diagnostics program (included with example software) is helpful when performing calibrations. Figure 5-1 shows the board layout. The four trimpots used for calibration are located in the upper left area of the board.

A/D Calibration

Two procedures are used to calibrate the A/D converter for all input voltage ranges. The first procedure calibrates the converter for the unipolar range (0 to +10 volts), and the second procedure calibrates the bipolar ranges (±5, ±10 volts). Table 5-1 shows the ideal input voltage for each bit weight for the unipolar, straight binary range, and Table 5-2 shows the ideal voltage for each bit weight for the bipolar, twos complement ranges.

Table 5-1 — A/D Converter Bit Weights, Unipolar, Straight Binary

	Ideal Input Voltage (millivolts)
A/D Bit Weight	0 to +10 Volts
1111 1111 1111	+9997.6
1000 0000 0000	+5000.0
0100 0000 0000	+2500.0
0010 0000 0000	+1250.0
0001 0000 0000	+625.00
0000 1000 0000	+312.50
0000 0100 0000	+156.250
0000 0010 0000	+78.125
0000 0001 0000	+39.063
0000 0000 1000	+19.5313
0000 0000 0100	+9.7656
0000 0000 0010	+4.8828
0000 0000 0001	+2.4414
0000 0000 0000	+0.0000

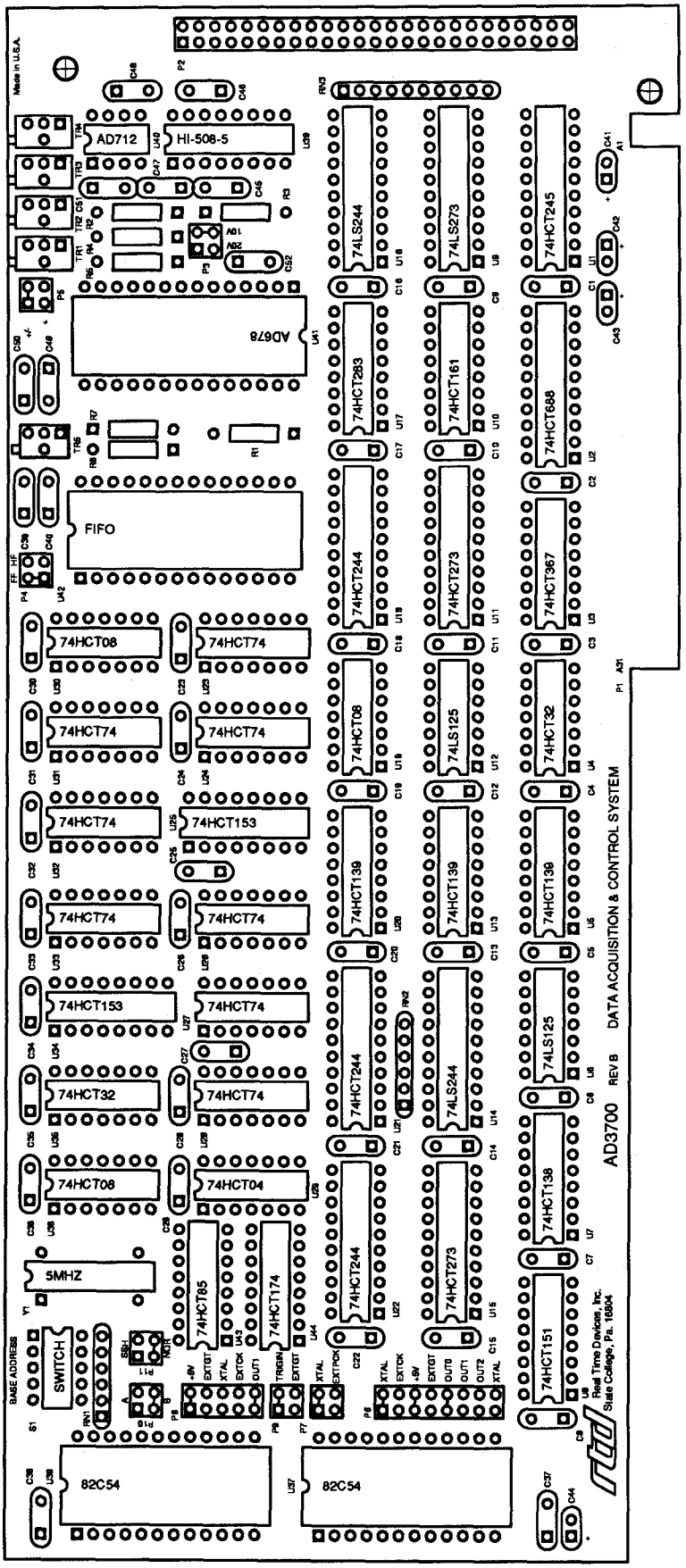


Fig. 5-1 — Board Layout Showing Factory-Configured Settings

Table 5-2 — A/D Converter Bit Weights, Bipolar, Twos Complement		
A/D Bit Weight	Ideal Input Voltage (millivolts)	
	-5 to +5 Volts	-10 to +10 Volts
1111 1111 1111	-2.44	-4.88
1000 0000 0000	-5000.00	-10000.00
0100 0000 0000	+2500.00	+5000.00
0010 0000 0000	+1250.00	+2500.00
0001 0000 0000	+625.00	+1250.00
0000 1000 0000	+312.50	+625.00
0000 0100 0000	+156.25	+312.50
0000 0010 0000	+78.13	+156.25
0000 0001 0000	+39.06	+78.13
0000 0000 1000	+19.53	+39.06
0000 0000 0100	+9.77	+19.53
0000 0000 0010	+4.88	+9.77
0000 0000 0001	+2.44	+4.88
0000 0000 0000	0.00	0.00

Unipolar Calibration

Two adjustments are made to calibrate the A/D converter for the unipolar range of 0 to +10 volts. One is the offset adjustment, and the other is the full scale, or gain, adjustment. Trimpot TR5 is used to make the offset adjustment, and trimpot TR1 is used for gain adjustment. This calibration procedure is performed with the board set up for a 0 to +10 volt input range. Before making these adjustments, make sure that the jumper on P3 is set for 10V and the jumper on P5 is set for +.

Use analog input channel 1 while calibrating the board. Connect your precision voltage source to channel 1. Set the voltage source to +1.22070 millivolts, start a conversion, and read the resulting data. Adjust trimpot TR5 until it flickers between the values listed in the table below. Next, set the voltage to +9.49829 volts, and repeat the procedure, this time adjusting TR1 until the data flickers between the values in the table. Note that the value used to adjust the full scale voltage is not the ideal full scale value for a 0 to +10 volt input range. This value is used because it is the maximum value at which the A/D converter is guaranteed to be linear, and ensures accurate calibration results.

Data Values for Calibrating Unipolar 10 Volt Range (0 to +10 volts)		
	Offset (TR5) Input Voltage = +1.22070 mV	Converter Gain (TR1) Input Voltage = +9.49829 V
A/D Converted Data	0000 0000 0000 0000 0000 0001	1111 0011 0010 1111 0011 0011

Bipolar Calibration

• Bipolar Range Adjustments: -5 to +5 Volts

Two adjustments are made to calibrate the A/D converter for the bipolar range of -5 to +5 volts. One is the offset adjustment, and the other is the full scale, or gain, adjustment. Trimpot TR2 is used to make the offset adjustment, and trimpot TR1 is used for gain adjustment. Before making these adjustments, make sure that the jumper on P3 is set for 10V and the jumper on P5 is set for +/-.

Use analog input channel 1 and set it for a gain of 1 while calibrating the board. Connect your precision voltage source to channel 1. Set the voltage source to -4.99878 volts, start a conversion, and read the resulting data. Adjust trimpot TR2 until it flickers between the values listed in the table below. Next, set the voltage to +4.99634 volts, and repeat the procedure, this time adjusting TR1 until the data flickers between the values in the table.

Data Values for Calibrating Bipolar 10 Volt Range (-5 to +5 volts)		
	Offset (TR2) **Input Voltage = -4.99878V**	**Converter Gain (TR1)** **Input Voltage = +4.99634V**
A/D Converted Data	1000 0000 0000 1000 0000 0001	0111 1111 1110 0111 1111 1111

• Bipolar Range Adjustments: -10 to +10 Volts

To adjust the bipolar 20-volt range (-10 to +10 volts), change the jumper on P3 so that it is installed across the 20V pins. Leave the P5 jumper at +/-. Then, set the input voltage to +5.0000 volts and adjust TR3 until the output matches the data in the table below.

Data Value for Calibrating Bipolar 20 Volt Range (-10 to +10 volts)	
	TR3 **Input Voltage = +5.0000V**
A/D Converted Data	0100 0000 0000

APPENDIX A

AD3700 SPECIFICATIONS

AD3700 Characteristics Typical @ 25° C

Interface

IBM PC/XT/AT compatible
Switch-selectable base address, I/O mapped
Jumper-selectable interrupts
Software-selectable DMA channel

Analog Input

8 single-ended inputs
Input impedance, each channel ..... >10 megohms
Input ranges ..... ±5, ±10, or 0 to +10 volts
Overvoltage protection ..... ±35 Vdc
Settling time ..... 5 µsec, max

A/D Converter AD678

Type ..... Successive approximation
Resolution ..... 12 bits (2.44 mV/4.88 mV)
Linearity ..... ±1 LSB, typ
Conversion speed ..... 5 µsec, typ
Throughput ..... 200 kHz

Pacer Clock

Range (using on-board clock) ..... 14 minutes to 5 µsec

FIFO ..... 2K, 4K, or 8K

IDT7203 ..... 2048 bytes, 1024 samples
IDT7204 ..... 4096 bytes, 2048 samples
IDT7205 ..... 8192 bytes, 4096 samples

Digital I/O

Number of lines ..... 8 input, 8 output

Timer/Counters ..... CMOS 82C54 (Optional NMOS 8254)

Six 16-bit down counters (3 per IC)
Binary or BCD counting
Programmable operating modes (6) ..... Interrupt on terminal count; programmable one-shot; rate generator; square wave rate generator; software-triggered strobe; hardware-triggered strobe
Counter input source ..... External clock (8 MHz, max) or on-board 5-MHz clock
Counter outputs ..... Available externally; used as PC interrupts
Counter gate source ..... External gate or always enabled

Miscellaneous Inputs/Outputs (PC bus-sourced)

±5 volts
±12 volts
Ground

Current Requirements

+5 volts ..... 80 mA
+12 volts ..... 36 mA
–12 volts ..... 34 mA

Connector

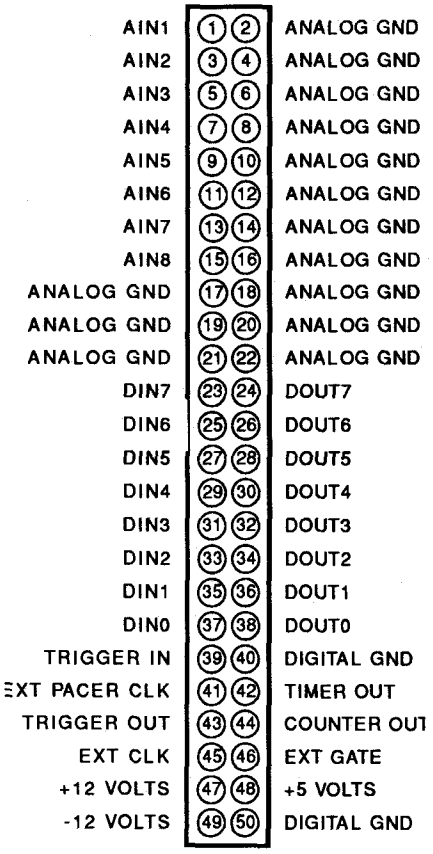
50-pin, right angle, shrouded box header

Size

3.875"H x 8.7"W (99mm x 221mm)

APPENDIX B

P2 CONNECTOR PIN ASSIGNMENTS


AIN1 1 2 ANALOG GND
AIN2 3 4 ANALOG GND
AIN3 5 6 ANALOG GND
AIN4 7 8 ANALOG GND
AIN5 9 10 ANALOG GND
AIN6 11 12 ANALOG GND
AIN7 13 14 ANALOG GND
AIN8 15 16 ANALOG GND
ANALOG GND 17 18 ANALOG GND
ANALOG GND 19 20 ANALOG GND
ANALOG GND 21 22 ANALOG GND
DIN7 23 24 DOUT7
DIN6 25 26 DOUT6
DIN5 27 28 DOUT5
DIN4 29 30 DOUT4
DIN3 31 32 DOUT3
DIN2 33 34 DOUT2
DIN1 35 36 DOUT1
DIN0 37 38 DOUT0
TRIGGER IN 39 40 DIGITAL GND
EXT PACER CLK 41 42 TIMER OUT
TRIGGER OUT 43 44 COUNTER OUT
EXT CLK 45 46 EXT GATE
+12 VOLTS 47 48 +5 VOLTS
-12 VOLTS 49 50 DIGITAL GND

APPENDIX C

COMPONENT DATA SHEETS

Intel 82C54 Programmable Interval Timer
Data Sheet Reprint

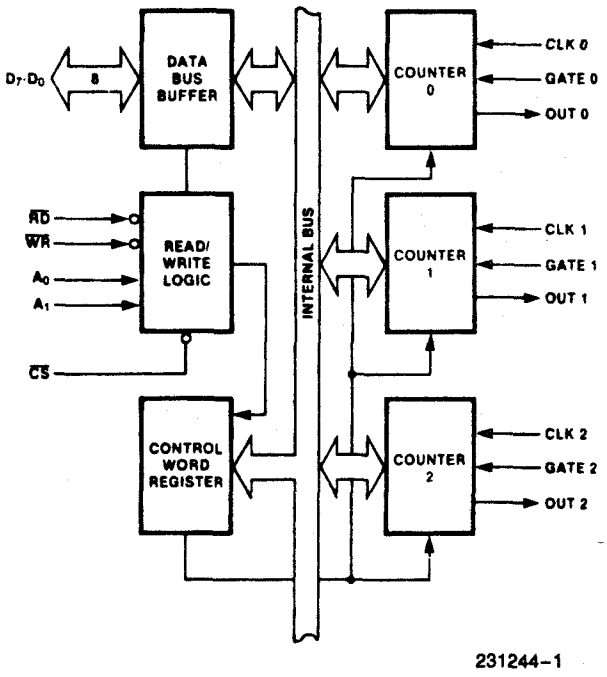
82C54
CHMOS PROGRAMMABLE INTERVAL TIMER

- **Compatible with all Intel and most other microprocessors**
- **High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188**
- **Handles Inputs from DC to 8 MHz — 10 MHz for 82C54-2**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**
- **Three Independent 16-bit counters**
- **Low Power CHMOS**
  - **I_{CC} = 10 mA @ 8 MHz Count frequency**
- **Completely TTL Compatible**
- **Six Programmable Counter Modes**
- **Binary or BCD counting**
- **Status Read Back Command**
- **Available in 24-Pin DIP and 28-Pin PLCC**

The Intel 82C54 is a high-performance, CHMOS version of the industry standard 8254 counter/timer which is designed to solve the timing control problems common in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 82C54 is pin compatible with the HMOS 8254, and is a superset of the 8253.

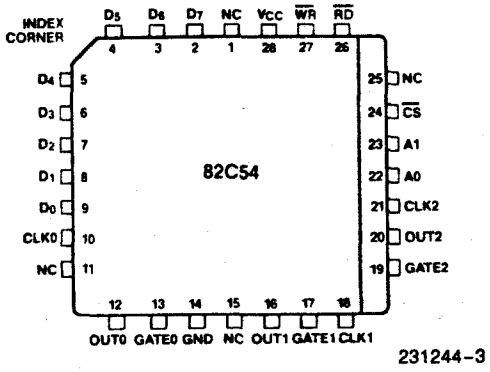
Six programmable timer modes allow the 82C54 to be used as an event counter, elapsed time indicator, programmable one-shot, and in many other applications.

The 82C54 is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent HMOS product. The 82C54 is available in 24-pin DIP and 28-pin plastic leaded chip carrier (PLCC) packages.



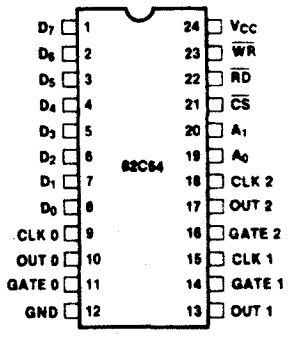
231244-1

Figure 1. 82C54 Block Diagram



231244-3

PLASTIC LEADED CHIP CARRIER



231244-2

Diagrams are for pin reference only. Package sizes are not to scale.

Figure 2. 82C54 Pinout

September 1989
Order Number: 231244-005

Table 1. Pin Description

Symbol	Pin Number		Type	Function
	DIP	PLCC		
D_7-D_0	1-8	2-9	I/O	Data: Bidirectional tri-state data bus lines, connected to system data bus.
CLK 0	9	10	I	Clock 0: Clock input of Counter 0.
OUT 0	10	12	O	Output 0: Output of Counter 0.
GATE 0	11	13	I	Gate 0: Gate input of Counter 0.
GND	12	14		Ground: Power supply connection.
OUT 1	13	16	O	Out 1: Output of Counter 1.
GATE 1	14	17	I	Gate 1: Gate input of Counter 1.
CLK 1	15	18	I	Clock 1: Clock input of Counter 1.
GATE 2	16	19	I	Gate 2: Gate input of Counter 2.
OUT 2	17	20	O	Out 2: Output of Counter 2.
CLK 2	18	21	I	Clock 2: Clock input of Counter 2.
A_1, A_0	20-19	23-22	I	Address: Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.

A_1	A_0	Selects
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register

$\overline{CS}$	21	24	I	Chip Select: A low on this input enables the 82C54 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
$\overline{RD}$	22	26	I	Read Control: This input is low during CPU read operations.
$\overline{WR}$	23	27	I	Write Control: This input is low during CPU write operations.
V_{CC}	24	28		Power: +5V power supply connection.
NC		1, 11, 15, 25		No Connect

FUNCTIONAL DESCRIPTION

General

The 82C54 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 82C54 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 82C54 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 82C54 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 82C54 are:

- Real time clock
- Even counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 82C54 to the system bus (see Figure 3).

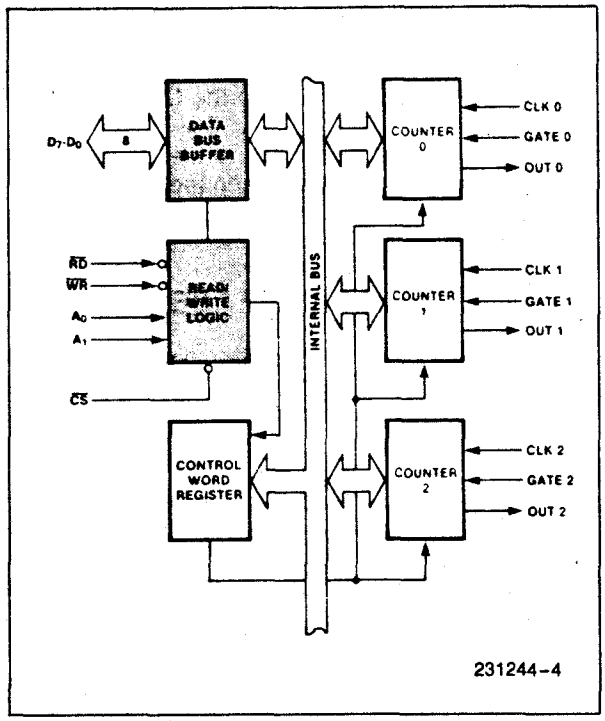


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 82C54. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 82C54 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 82C54 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 82C54 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A_1, A_0 = 11. If the CPU then does a write operation to the 82C54, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

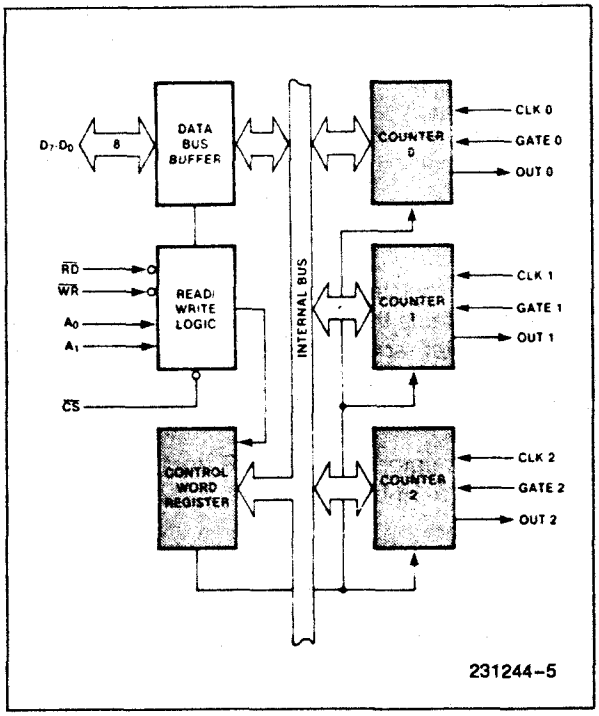


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

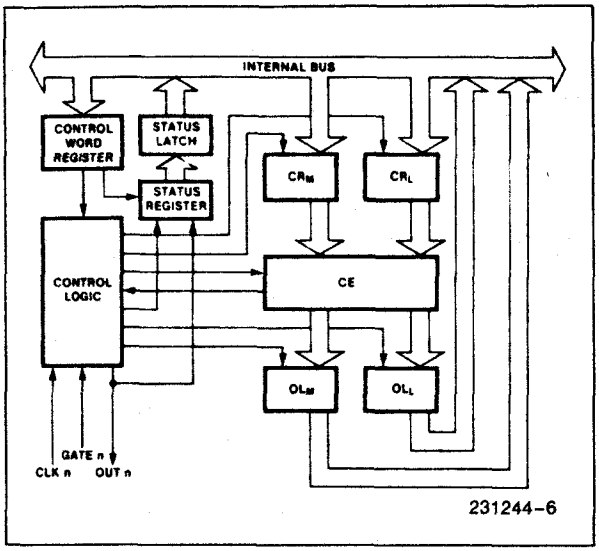


Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 82C54, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

82C54 SYSTEM INTERFACE

The 82C54 is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

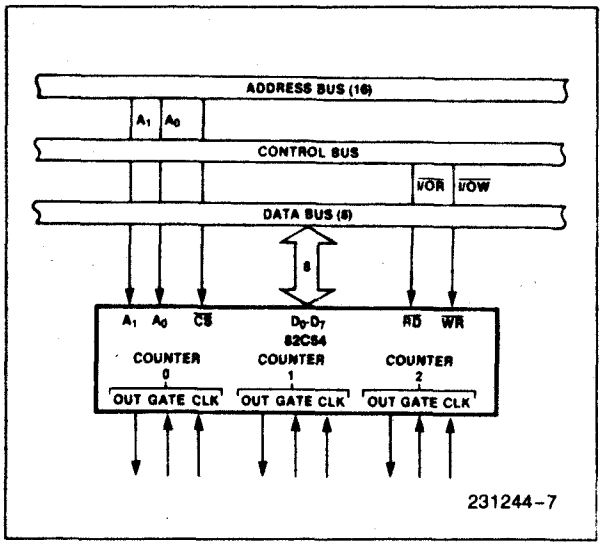


Figure 6. 82C54 System Interface

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 82C54 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 82C54

Counters are programmed by writing a Control Word and then an initial count. The control word format is shown in Figure 7.

All Control Words are written into the Control Word Register, which is selected when A_1, A_0 = 11. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1, A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

A_1, A_0 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

Write Operations

The programming procedure for the 82C54 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1, A_0 inputs), and each Control Word specifies the Counter it applies to (SC0, SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

		A_1	A_0
Control Word —	Counter 0	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
Control Word —	Counter 1	1	1
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 2	1	1
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0

		A_1	A_0
Control Word —	Counter 0	1	1
Counter Word —	Counter 1	1	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 2	1	0

NOTE:
In all four examples, all counters are programmed to read/write two-byte counts.
These are only four of many possible programming sequences.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 82C54.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when A_1, A_0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

A_1, A_0 = 11; $\overline{CS}$ = 0; $\overline{RD}$ = 1; $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1, SC0 - specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4 - 00 designates Counter Latch Command

X - don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 82C54 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies; A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back command. This command allows the user to check the count value, programmed Mode, and current state of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1 = 1.

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counter(s)
D_3: 1 = Select counter 2
D_2: 1 = Select counter 1
D_1: 1 = Select counter 0
D_0: Reserved for future expansion; must be 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the

count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting $\overline{\text{STATUS}}$ bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

D_7 1 = Out Pin is 1
0 = Out Pin is 0
D_6 1 = Null count
0 = Count available for reading
D_5-D_0 Counter Programmed Mode (See Figure 7)

Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

THIS ACTION:	CAUSES:
A. Write to the control word register:[1]	Null count = 1
B. Write to the count register (CR);[2]	Null count = 1
C. New count is loaded into CE (CR → CE);	Null count = 0

[1] Only the counter specified by the control word will have its null count set to 1. Null count bits of other counters are unaffected.

[2] If the counter is programmed for two-byte counts (least significant byte then most significant byte) null count goes to 1 when the second byte is written.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both $\overline{\text{COUNT}}$ and $\overline{\text{STATUS}}$ bits D5,D4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

Command								Description	Results
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Mode Definitions

The following are defined for use in describing the operation of the 82C54.

CLK PULSE: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

TRIGGER: a rising edge of a Counter's GATE input.

COUNTER LOADING: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

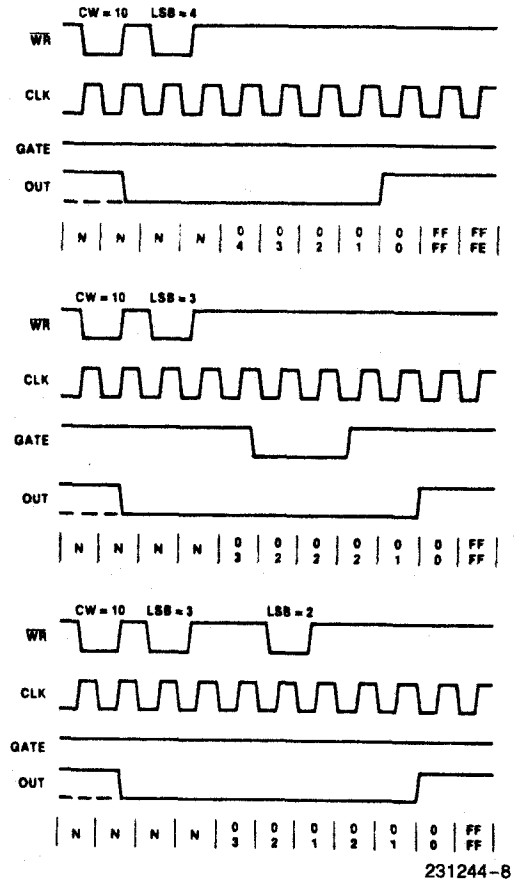
After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.



231244-8

NOTE:
The Following Conventions Apply To All Mode Timing Diagrams:
1. Counters are programmed for binary (not BCD) counting and for Reading/Writing least significant byte (LSB) only.
2. The counter is always selected ($\overline{CS}$ always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, hex is written to the counter.
4. LSB stands for "Least Significant Byte" of count.
5. Numbers below diagrams are count values.
The lower number is the least significant byte.
The upper number is the most significant byte. Since the counter is programmed to Read/Write LSB only, the most significant byte cannot be read.
N stands for an undefined count.
Vertical lines show transitions between count values.

Figure 15. Mode 0

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

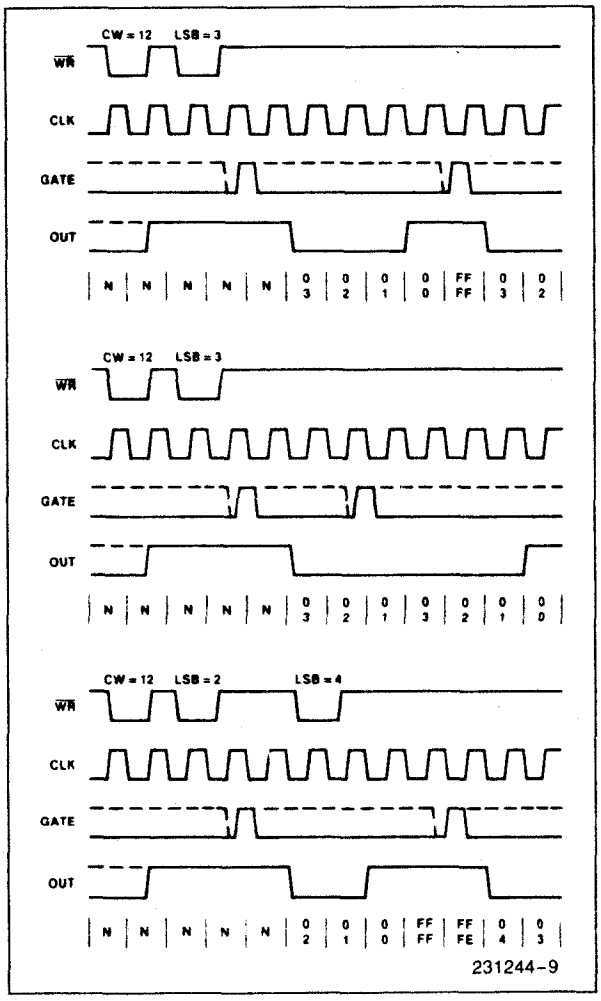


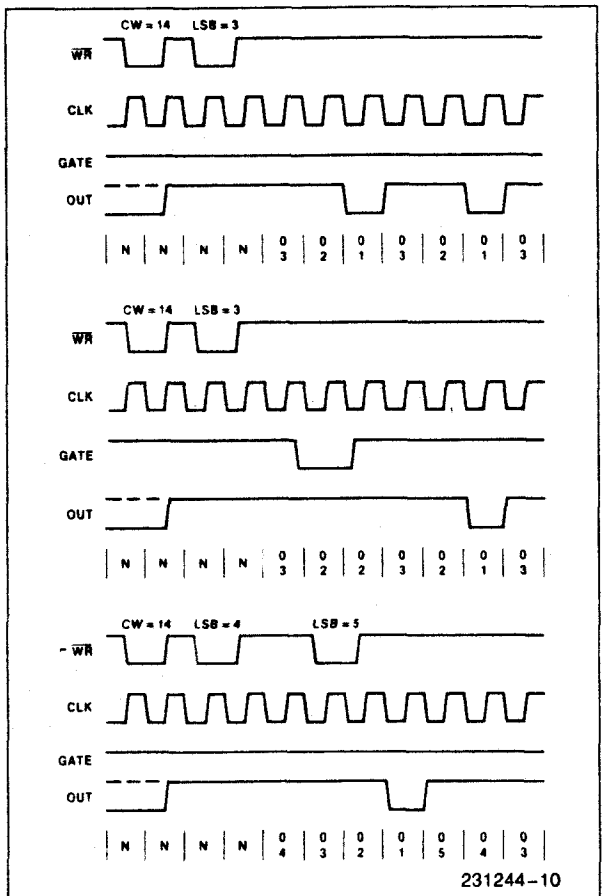
Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typicially used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 17. Mode 2

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

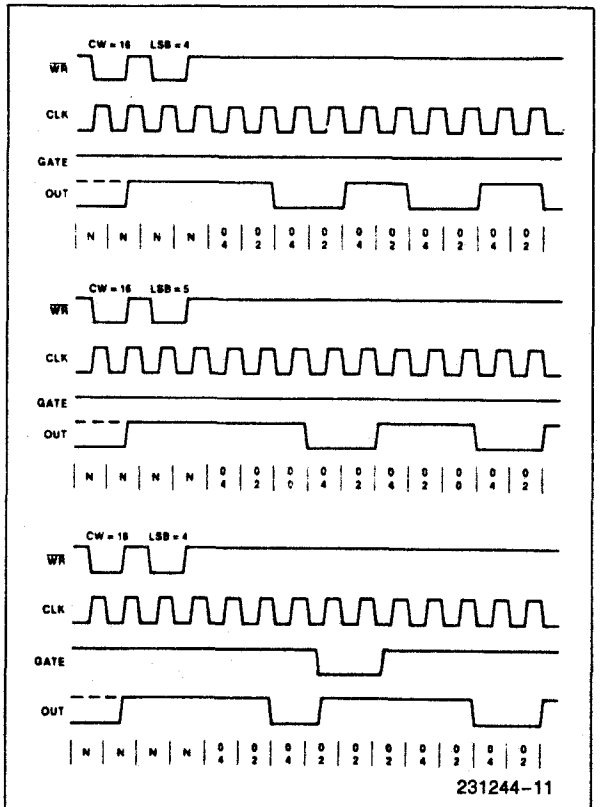
After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N −1)/2 counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N+1 CLK pulses after the new count of N is written.

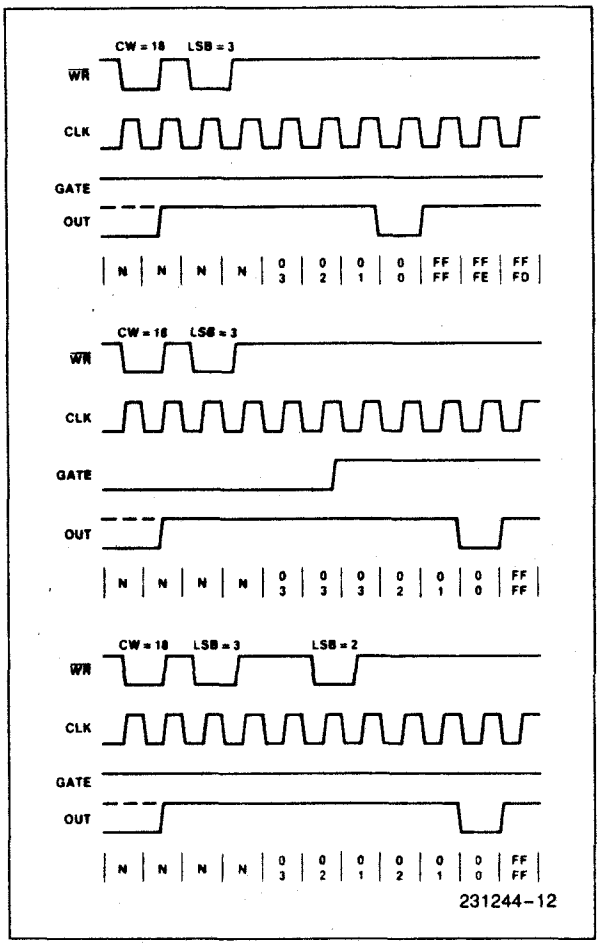


Figure 19. Mode 4

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N+1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

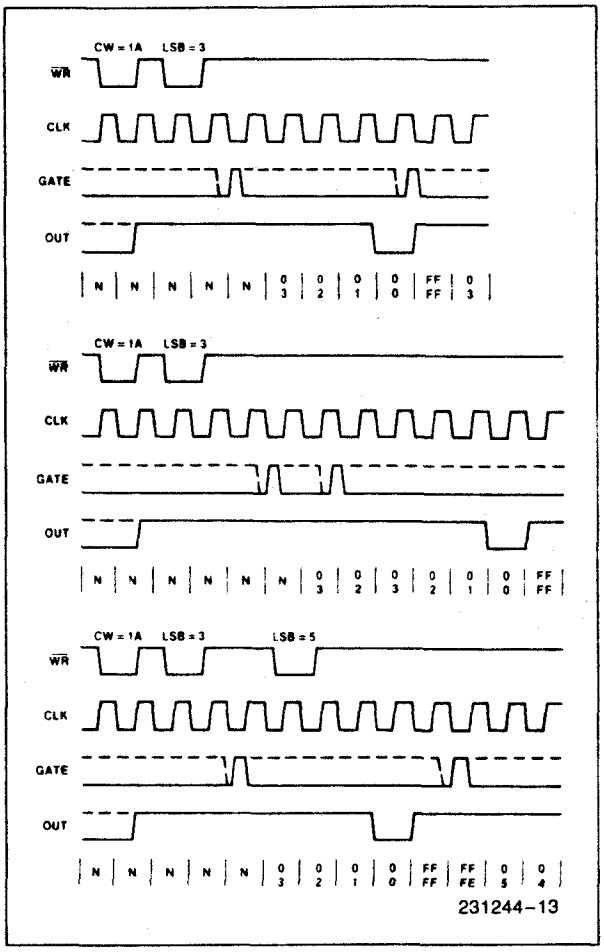


Figure 20. Mode 5

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 21. Gate Pin Operations Summary

MODE	MIN COUNT	MAX COUNT
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0

NOTE:
0 is equivalent to 2^{16} for binary counting and 10^4 for BCD counting

Figure 22. Minimum and Maximum Initial Counts

Operation Common to All Modes

Programming

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to 70°C
Storage Temperature .....−65° to +150°C
Supply Voltage .....−0.5 to +8.0V
Operating Voltage .....+4V to +7V
Voltage on any Input.....GND −2V to +6.5V
Voltage on any Output ..GND−0.5V to V_{CC} + 0.5V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(T_A=0°C to 70°C, V_{CC}=5V± 10%, GND=0V) (T_A = −40°C to +85°C for Extended Temperature)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} − 0.4		V V	I_{OH} = −2.5 mA I_{OH} = −100 µA
I_{IL}	Input Load Current		±2.0	µA	V_{IN}=V_{CC} to 0V
I_{OFL}	Output Float Leakage Current		±10	µA	V_{OUT}=V_{CC} to 0.0V
I_{CC}	V_{CC} Supply Current		20	mA	Clk Freq= 8MHz 82C54 10MHz 82C54-2
I_{CCSB}	V_{CC} Supply Current-Standby		10	µA	CLK Freq = DC $\overline{CS}$ = V_{CC}. All Inputs/Data Bus V_{CC} All Outputs Floating
I_{CCSB1}	V_{CC} Supply Current-Standby		150	µA	CLK Freq = DC $\overline{CS}$ = V_{CC}. All Other Inputs, I/O Pins = V_{GND}, Outputs Open
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND(5)
C_{OUT}	Output Capacitance		20	pF	

A.C. CHARACTERISTICS

(T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND =0V) (T_A = −40°C to +85°C for Extended Temperature)

BUS PARAMETERS (Note 1)

READ CYCLE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{AR}	Address Stable Before $\overline{RD}$ ↓	45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}$ ↓	0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}$ ↑	0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		95		ns
t_{RD}	Data Delay from $\overline{RD}$ ↓		120		85	ns
t_{AD}	Data Delay from Address		220		185	ns
t_{DF}	$\overline{RD}$ ↑ to Data Floating	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		165		ns

NOTE:
1. AC timings measured at V_{OH} = 2.0V, V_{OL} = 0.8V.

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{AW}	Address Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{SW}	$\overline{CS}$ Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{WA}	Address Hold Time After $\overline{WR}\uparrow$	0		0		ns
t_{WW}	$\overline{WR}$ Pulse Width	150		95		ns
t_{DW}	Data Setup Time Before $\overline{WR}\uparrow$	120		95		ns
t_{WD}	Data Hold Time After $\overline{WR}\uparrow$	0		0		ns
t_{RV}	Command Recovery Time	200		165		ns

CLOCK AND GATE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{CLK}	Clock Period	125	DC	100	DC	ns
t_{PWH}	High Pulse Width	60(3)		30(3)		ns
t_{PWL}	Low Pulse Width	60(3)		50(3)		ns
T_R	Clock Rise Time		25		25	ns
t_F	Clock Fall Time		25		25	ns
t_{GW}	Gate Width High	50		50		ns
t_{GL}	Gate Width Low	50		50		ns
t_{GS}	Gate Setup Time to CLK $\uparrow$	50		40		ns
t_{GH}	Gate Hold Time After CLK $\uparrow$	50(2)		50(2)		ns
T_{OD}	Output Delay from CLK $\downarrow$		150		100	ns
t_{ODG}	Output Delay from Gate $\downarrow$		120		100	ns
t_{WC}	CLK Delay for Loading(4)	0	55	0	55	ns
t_{WG}	Gate Delay for Sampling(4)	−5	50	−5	40	ns
t_{WO}	OUT Delay from Mode Write		260		240	ns
t_{CL}	CLK Set Up for Count Latch	−40	45	−40	40	ns

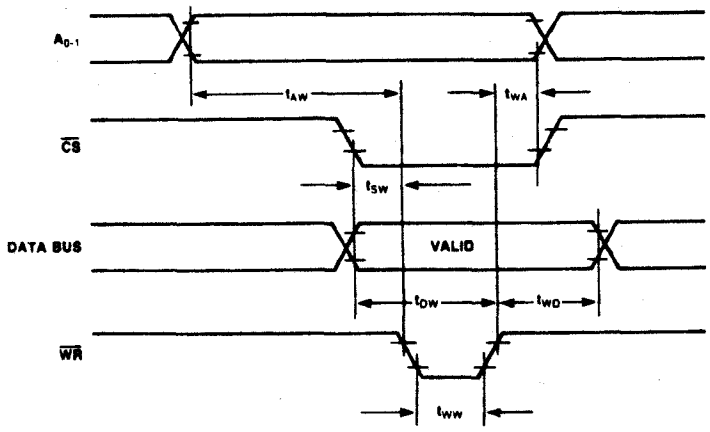
NOTES:
2. In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 82C54-2) of the rising clock edge may not be detected.
3. Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.
4. Except for Extended Temp., See Extended Temp. A.C. Characteristics below.
5. Sampled not 100% tested. T_A = 25°C.
6. If CLK present at T_{WC} min then Count equals N+2 CLK pulses, T_{WC} max equals Count N+1 CLK pulse. T_{WC} min to T_{WC} max, count will be either N+1 or N+2 CLK pulses.
7. In Modes 1 and 5, if GATE is present when writing a new Count value, at T_{WG} min Counter will not be triggered, at T_{WG} max Counter will be triggered.
8. If CLK present when writing a Counter Latch or ReadBack Command, at T_{CL} min CLK will be reflected in count value latched, at T_{CL} max CLK will not be reflected in the count value latched. Writing a Counter Latch or ReadBack Command between T_{CL} min and T_{WL} max will result in a latched count vallue which is ± one least significant bit.

EXTENDED TEMPERATURE (T_A = −40°C to +85°C for Extended Temperature)

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{WC}	CLK Delay for Loading	−25	25	−25	25	ns
t_{WG}	Gate Delay for Sampling	−25	25	−25	25	ns

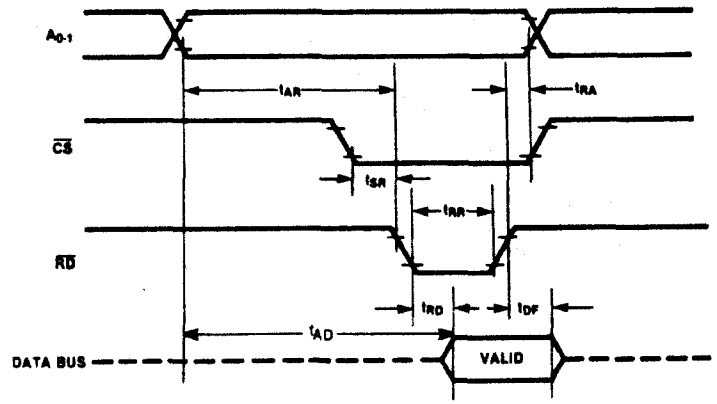
WAVEFORMS

WRITE



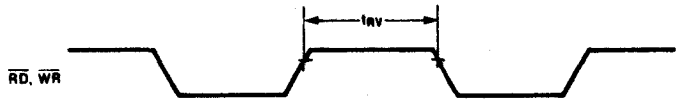
231244-14

READ



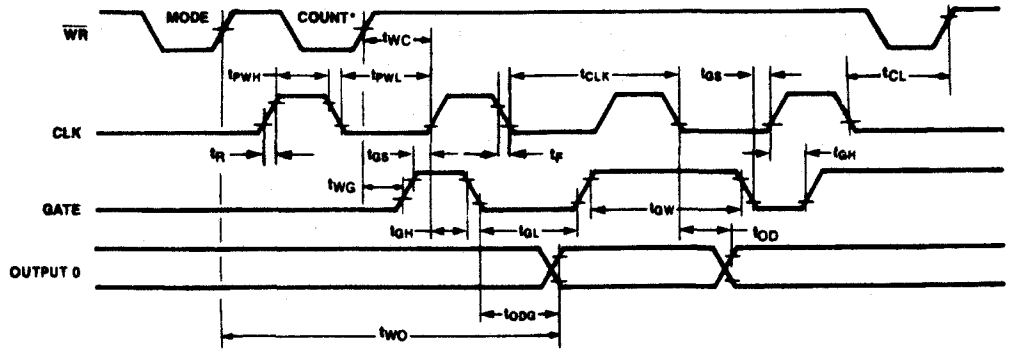
231244-15

RECOVERY



231244-16

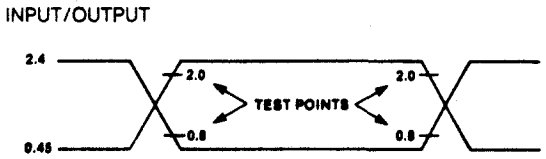
CLOCK AND GATE



231244-17

* Last byte of count being written

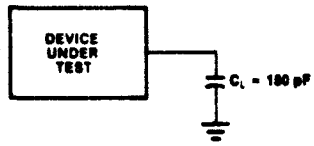
A.C. TESTING INPUT, OUTPUT WAVEFORM



231244-18

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT



231244-19

C_L = 150 pF

C_L includes jig capacitance

APPENDIX D

CONFIGURING THE AD3700 FOR SIGNAL*MATH

Jumper and Switch Settings

When running SIGNAL*MATH, you may have to change some of the AD3700's on-board jumpers from their current positions. All jumpers must be set to the factory positions as described in Chapter 1, except for P3 and P5, which can be configured for any of the three possible input ranges.

S1 — Base Address

SIGNAL*MATH assumes that the base address of your AD3700 is the factory setting of 300 hex (768 decimal). If you change this setting, you **must** run the ADAINST program and reset the base address.

NOTE: When using the ADAINST program, you can enter the base address in decimal or hexadecimal notation. When entering a hex value, you must precede the number by a dollar sign (for example, $300).

Running ADAINST

After the jumpers and switch are set and the AD3700 board is installed in the computer, you are ready to configure SIGNAL*MATH so that it is compatible with your board's settings. This is done by running the ADAINST driver installation program. After running the program, open AD3700.EXE from the *Open a File* menu. You will see a screen similar to the screen shown in Figure D-1 below. The factory default settings are shown in the illustration. Your settings may or may not match the default settings, depending on whether you have made changes to these decimal or hexadecimal value (hex values must be preceded by a dollar sign (for example, $300)). Refer to your board's manual if you need help in determining the correct value to enter.

EOC IT (End-of-Convert Interrupt). In this block, enter the IRQ channel number to be used by the end-of-convert interrupt (see BA + 6 description in Chapter 4).

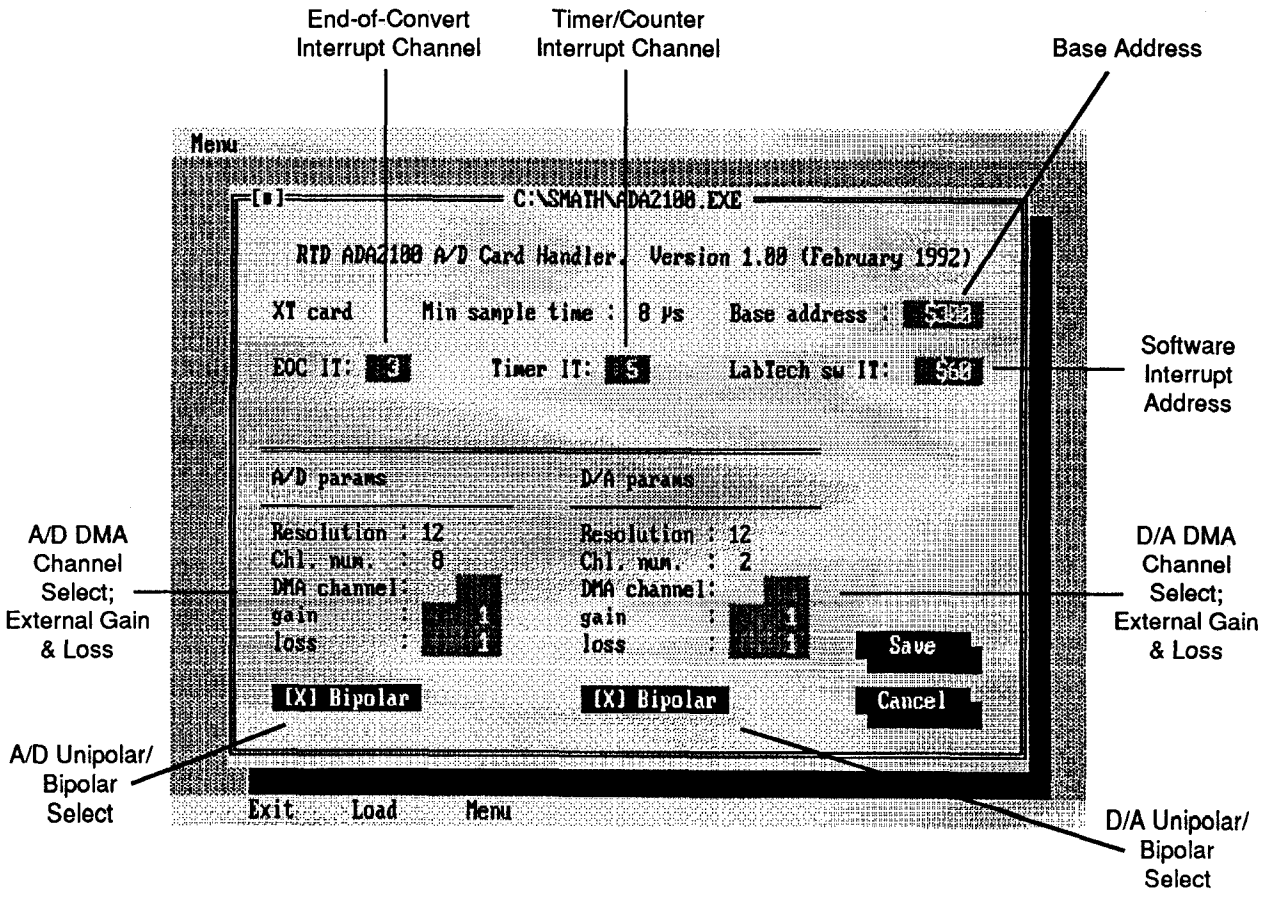


Fig. D-1 — ADAINST.EXE Screen

Timer IT (Timer/Counter Interrupt). This block is not used on the AD3700, and should be left blank.

LabTech SW IT (LABTECH NOTEBOOK Software Interrupt). This sets the software interrupt address where LABTECH NOTEBOOK's labLINX driver is installed. The factory setting is $60. This setting can be ignored when running SIGNAL*MATH.

A/D Parameters. Six A/D board parameters are listed: resolution, number of channels, active DMA channel, gain, loss, and input voltage polarity.

Resolution and number of channels are fixed by the program for your board.

If you are using DMA transfer, you must enter the channel number you are using in the DMA channel block. Valid channels numbers are 1 and 3.

The next two blocks, gain and loss, are provided so that you can make adjustments for **external** gain or loss, including resistor configurable gain circuitry you added to the board. For example, if you have a gain circuit installed, then you must tell SIGNAL*MATH this gain. If your input signal is externally attenuated, then you can adjust for this by setting a value other than 1 for loss. Numbers must be entered as whole decimal values. The factory default setting for gain and loss is 1.

For a bipolar input range, an X should be placed before Bipolar on the screen (default setting). For unipolar operation, remove the X.

D/A Parameters. These settings are not applicable to the AD3700.

APPENDIX E

CONFIGURING THE AD3700 FOR ATLANTIS

If you have purchased ATLANTIS data acquisition and real time monitoring application software for your AD3700, please note that the ATLANTIS drivers for your board must be loaded from your driver disk into the same directory as the ATLANTIS.EXE program. All jumpers must be set to the factory positions as described in Chapter 1, except for P3 and P5, which can be configured for any of the three possible input ranges. When running ATLANTIS, make sure the base address setting in the program and on the board match.

S1 — Base Address

ATLANTIS assumes that the base address of your AD3700 is the factory setting of 300 hex (see Chapter 1). If you changed this setting, you **must** run the ATINST program and reset the base address.

NOTE: The ATINST program requires the base address to be entered in **decimal** notation.

APPENDIX F

WARRANTY

LIMITED WARRANTY

Real Time Devices, Inc. warrants the hardware and software products it manufactures and produces to be free from defects in materials and workmanship for one year following the date of shipment from REAL TIME DEVICES. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, REAL TIME DEVICES will repair or replace, at its option, any defective products or parts at no additional charge, provided that the product is returned, shipping prepaid, to REAL TIME DEVICES. All replaced parts and products become the property of REAL TIME DEVICES. **Before returning any product for repair, customers are required to contact the factory for an RMA number.**

THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY PRODUCTS WHICH HAVE BEEN DAMAGED AS A RESULT OF ACCIDENT, MISUSE, ABUSE (such as: use of incorrect input voltages, improper or insufficient ventilation, failure to follow the operating instructions that are provided by REAL TIME DEVICES, "acts of God" or other contingencies beyond the control of REAL TIME DEVICES), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN REAL TIME DEVICES. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES ARE EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND REAL TIME DEVICES EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES FOR MECHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED TO THE DURATION OF THIS WARRANTY. IN THE EVENT THE PRODUCT IS NOT FREE FROM DEFECTS AS WARRANTED ABOVE, THE PURCHASER'S SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. UNDER NO CIRCUMSTANCES WILL REAL TIME DEVICES BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, AND SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

3700 Board User-Selected Settings	
Base I/O Address:	
(hex)	(decimal)