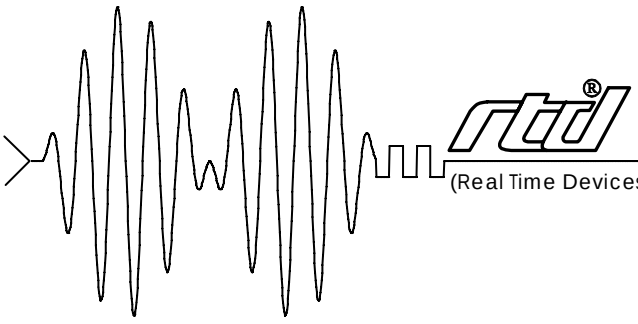
DM6916
User's Manual



RTD Embedded Technologies Inc.
"Accessing the Analog World"®

BDM-610010018
Rev. A

DM6916
User's Manual

RTD Embedded Technologies, INC.
103 Innovation Blvd.
State College, PA 16803-0906

Phone: +1-814-234-8087
FAX: +1-814-234-5218

E-mail
sales@rtd.com
techsupport@rtd.com

web site
http://www.rtd.com

Revision History

Rev. A New manual naming method

Published by:

RTD Embedded Technologies, Inc.
103 Innovation Blvd.
State College, PA 16803-0906

Copyright 1999, 2002, 2003 by RTD Embedded Technologies, Inc.
All rights reserved
Printed in U.S.A.

The RTD Logo is a registered trademark of RTD Embedded Technologies. cpuModule and utilityModule are trademarks of RTD Embedded Technologies. PhoenixPICO and PheonixPICO BIOS are trademarks of Phoenix Technologies Ltd. PS/2, PC/XT, PC/AT and IBM are trademarks of International Business Machines Inc. MS-DOS, Windows, Windows 95, Windows 98 and Windows NT are trademarks of Microsoft Corp. PC/104 is a registered trademark of PC/104 Consortium. All other trademarks appearing in this document are the property of their respective owners.

Table of Contents

INTRODUCTION ***i*-1**

Pulse Width Modulators (PWM) *i*-3
8254 Timer/Counters *i*-3
What Comes With Your Module *i*-3
Module Accessories *i*-3
Using This Manual *i*-3
When You Need Help *i*-3

CHAPTER 1 — MODULE SETTINGS **1-1**

Factory-Configured Switch and Jumper Settings 1-3
JP1, JP2 — Interrupt Channel Select (Factory Setting: Jumper installed on G; IRQ Disabled) 1-4
JP3 — 8254 Clock and Gate Source Select (Factory Settings: See Figure 1-4) 1-6
JP4 — Interrupt Source Select (Factory Setting: OT2) 1-7
SW1 — Base Address (Factory Setting: 300 hex (768 decimal)) 1-8

CHAPTER 2 — MODULE INSTALLATION **2-1**

Module Installation 2-3
External I/O Connections 2-3
Connecting the PWM and Digital Outputs 2-4
Connecting the Timer/Counter I/O 2-4
Connecting the External Interrupt 2-4
Running the 6916DIAG Diagnostics Program 2-5

CHAPTER 3 — HARDWARE DESCRIPTION **3-1**

Pulse Width Modulators 3-3
Timer/Counters 3-3

CHAPTER 4 — I/O MAPPING **4-1**

Defining the I/O Map 4-3
BA + 0: Pulse Width Modulator 0.0 Duty Cycle (Write only) 4-4
BA + 1: Pulse Width Modulator 0.1 Duty Cycle (Write only) 4-4
BA + 2: Pulse Width Modulator 0.2 Duty Cycle (Write only) 4-4
BA + 3: PWM Control and Digital Output Register (Write only) 4-4
BA + 4: Pulse Width Modulator 1.0 Duty Cycle (Write only) 4-5
BA + 5: Pulse Width Modulator 1.1 Duty Cycle (Write only) 4-5
BA + 6: Pulse Width Modulator 1.2 Duty Cycle (Write only) 4-5
BA + 7: PWM Control and Digital Output Register (Write only) 4-5
BA + 8: Pulse Width Modulator 2.0 Duty Cycle (Write only) 4-6
BA + 9: Pulse Width Modulator 2.1 Duty Cycle (Write only) 4-6
BA + 10: Pulse Width Modulator 2.2 Duty Cycle (Write only) 4-6
BA + 11: PWM Control and Digital Output Register (Write only) 4-6
BA + 12: 8254 Timer/Counter 0 (Read/Write) 4-7
BA + 13: 8254 Timer/Counter 1 (Read/Write) 4-7
BA + 14: 8254 Timer/Counter 2 (Read/Write) 4-7
BA + 15: 8254 Timer/Counter Control Word (Write Only) 4-7
BA + 16: Clear IRQ/IRQ Enable (Read/Write) 4-7
BA + 17: IRQ Status (Read Only) 4-8

BA + 18: Reserved 4-8
BA + 19: Reserved 4-8
Programming the DM6916 4-9
Clearing and Setting Bits in a Port 4-9

CHAPTER 5 — DIGITAL OUTPUTS 5-1

CHAPTER 6 — TIMER/COUNTERS 6-1

CHAPTER 7 — INTERRUPTS 7-1

JP4: Jumper Selectable Interrupts 7-3
Selecting the Interrupt Channel 7-3
Interrupt Sharing 7-3
Basic Programming For Interrupt Handling 7-4
What Is an Interrupt? 7-4
Interrupt Request Lines 7-4
8259 Programmable Interrupt Controller 7-4
Interrupt Mask Register (IMR) 7-4
End-of-Interrupt (EOI) Command 7-4
What Exactly Happens When an Interrupt Occurs? 7-5
Using Interrupts in Your Programs 7-5
Writing an Interrupt Service Routine (ISR) 7-5
Saving the Startup Interrupt Mask Register (IMR) and Interrupt Vector 7-7
Restoring the Startup IMR and Interrupt Vector 7-7
Common Interrupt Mistakes 7-7

APPENDIX A — DM6916 SPECIFICATIONS A-1

APPENDIX B — CONNECTOR PIN ASSIGNMENTS B-1

APPENDIX C — COMPONENT DATA SHEETS C-1

APPENDIX D — WARRANTY D-1

List of Illustrations

1-1 Module Layout Showing Factory-Configured Settings 1-4
1-2 Interrupt Channel Select Jumper, JP1 and JP2 1-5
1-3 Pulling Down the Interrupt Request Lines 1-5
1-4 8254 Clock and Gate Sources Jumpers, JP3 1-6
1-5 8254 Circuit Diagram 1-6
1-6 Interrupt Source Select Jumper, JP4 1-7
1-7 Base Address Switch, SW1 1-8
2-1 CN3 I/O Connector Pin Assignments 2-4
3-1 DM6916 Block Diagram 3-3
3-2 Timer/Counter Circuit Block Diagram 3-4
6-1 8254 Timer/Counter Circuit Block Diagram 6-3

INTRODUCTION

The DM6916 Pulse Width Modulator (PWM) dataModule® turns your IBM PC-compatible cpuModule™ or other PC/104 computer into a high-performance control system. Ultra-compact for embedded and portable applications, the module features:

- Nine 8-bit Pulse Width Modulated Outputs
- 9 buffered digital outputs
- Three 16-bit timer/counters and on-board 8 MHz clock,
- Operation from single +5V supply,
- DOS example programs with source code in BASIC and C,
- Diagnostics software.

The following paragraphs briefly describe the major functions of the module. A detailed discussion of module functions is included in subsequent chapters.

Pulse Width Modulators (PWM)

The DM6916 has nine 8-bit pulse width modulator circuits. The PWM function is divided into three blocks with each having three PWMs and four digital outputs.

Each block consists of an 8-bit counter that is driven by either an on-board 8 MHz clock, the 8 MHz clock divided by a 16-bit counter/timer or an external clock. There are three 8-bit PWM registers. The counter output is compared to each register and the PWM outputs are high if the count is less than the register and low if the count is greater than the register. Three buffered digital output lines available for control functions such as direction.

8254 Timer/Counters

An 8254 programmable interval timer provides three 16-bit, 8 MHz timer/counters to support a wide range of user timing and counting functions.

What Comes With Your Module

You receive the following items in your module package:

- DM6916 module with stackthrough bus header
- Mounting hardware
- Example programs in BASIC and C with source code & diagnostics software
- User's manual

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

Module Accessories

Hardware accessories for the DM6916 include the XD68 cable and TB68 terminal block board.

Using This Manual

This manual is intended to help you install your new module and get it running quickly, while also providing enough detail about the module 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 application programs.

When You Need Help

This manual and the example programs in the software package included with your module provide enough information to properly use all of the module's features. If you have any problems installing or using this dataModule, 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. You can also contact us through our E-mail address **techsupport@rtd.com**.

CHAPTER 1

MODULE SETTINGS

The DM6916 has jumper and switch settings you can change if necessary for your application. The module is factory-configured as listed in the table and shown on the layout diagram in the beginning of this chapter. Should you need to change these settings, use these easy-to-follow instructions before you stack the module with your computer system.

Factory-Configured Switch and Jumper Settings

Table 1-1 lists the factory settings of the user-configurable jumpers and switch on the DM6916 module. Figure 1-1 shows the module 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 SW1, the base address switch, to avoid address contention when you first use your module in your system.

Table 1-1 Factory Settings

Switch/ Jumper	Function Controlled	Factory Settings (Jumpers Installed)
JP1	Connects a JP4 jumper selectable or digital interrupt source to an interrupt channel; pulls tri-state buffers to ground (G) for multiple interrupt applications	Jumper installed on G (ground for buffer); interrupt channels disabled
JP2	Connects interrupt source jumpered on JP4 to an AT interrupt channel	no jumper
JP3	Sets the clock and gate sources for the 8254 timer/counter	CLK0: OSC; CLK1: OT0 CLK2:OT1; GT2:EG2 (timer/counters cascaded)
JP4	Selects one of four interrupt sources for interrupt generation	OT2
JP5	Not Used	Not Installed
JP6	Not Used	Not Installed
JP7	Not Used	Not Installed
JP8	Not Used	Not Installed
JP9	Not Used	Not Installed
JP10	Not Used	Not Installed
JP11	Not Used	Not Installed
SW1	Sets the base address	300 hex (768 decimal)

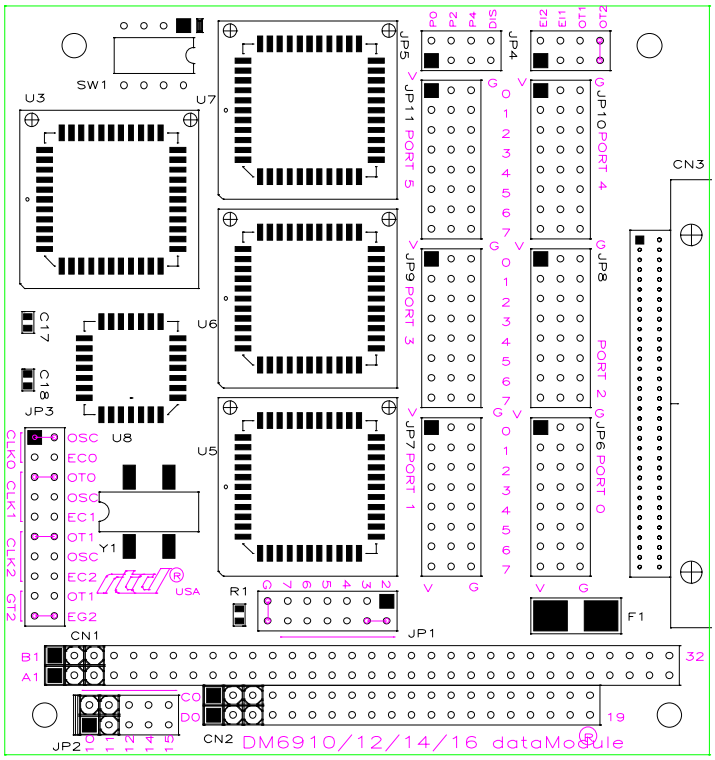


Fig. 1-1 — Module Layout Showing Factory-Configured Settings

JP1, JP2 — Interrupt Channel Select (Factory Setting: Jumper installed on G; IRQ Disabled)

These header connectors, shown in Figure 1-2, lets you connect any one of four jumper selectable (JP4) interrupt sources and the Advanced Digital Interrupt sources to an interrupt channel, IRQ2 through IRQ15. XT channels 2 through 7 are jumpered on JP1 and AT channels 10 through 15 are jumpered on JP2. In AT computers channels 2 and 9 are the same channel. To activate a channel, you must install a jumper vertically across the desired IRQ channel's pins. Only one channel on either JP1 or JP2 should be jumpered at any time. Figure 1-2a shows the factory settings.

This module supports an interrupt sharing mode where the pins labeled G connect a 1 kilohm pull-down resistor to the output of a high-impedance tri-state driver which carries the interrupt request signal. This pull-down resistor drives the interrupt request line low whenever interrupts are not active. Whenever an interrupt request is made, the tri-state buffer is enabled, forcing the output high and generating an interrupt. After the interrupt has been serviced, you must return the IRQ line low, disabling the tri-state buffer and pulling the output low again. This is done by clearing the IRQ for the source which generated the interrupt. You also can have two or more modules that share the same IRQ channel. You can tell which module issued the interrupt request by monitoring each module's IRQ status bit(s). If you are not planning on sharing interrupts or if you are not sure that your CPU supports interrupt sharing, it is best to disable this feature and use the interrupts in the normal mode. This will insure compatibility with all CPUs. See chapter 4 for details on disabling the interrupt sharing circuit.

NOTE: When using multiple modules sharing the same interrupt, only one module should have the G jumper installed. The rest should be disconnected. Whenever you operate a single module, the G jumper should be installed. Whenever you operate the module with interrupt sharing disabled, the G jumper should be removed.

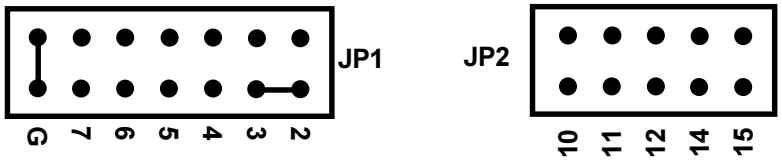


Fig. 1-2 — Interrupt Channel Select Jumpers, JP1 and JP2

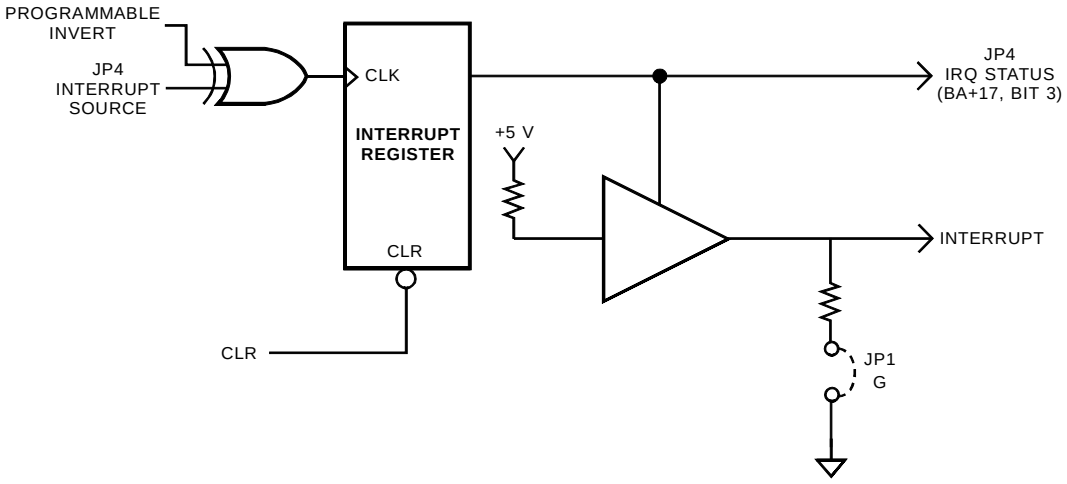


Fig. 1-3 — Pulling Down the Interrupt Request Lines

JP3 — 8254 Clock and Gate Source Select (Factory Settings: See Figure 1-4)

This header connector, shown in Figure 1-4, lets you select the clock sources for the three 8254 16-bit timer/counters. Figure 1-5 shows a block diagram of the timer/counter circuitry to help you in making these connections.

The clock source for Counter 0 is selected by placing a jumper on one of the two leftmost pairs of pins on the header, OSC or EC0. OSC is the on-board 8 MHz clock, and EC0 is an external clock source which can be connected through I/O connector CN3, pin 55. Counter 1 has three clock sources: OT0, which cascades it to Counter 0; OSC, which is the on-board 8 MHz clock; and EC1, which is an external clock source connected through I/O connector CN3, pin 59. Counter 2 has three clock sources: OT1, which cascades it to Counter 1; OSC, which is the on-board 8 MHz clock; and EC2, which is an external clock source connected through I/O connector CN3, pin 61.

The gate of Counter 2 can be connected to the output of Counter 1 (OT1) or to an external gate source (EG2) connected through I/O connector CN3, pin 62. When no external gate source is connected, this line is tied high.

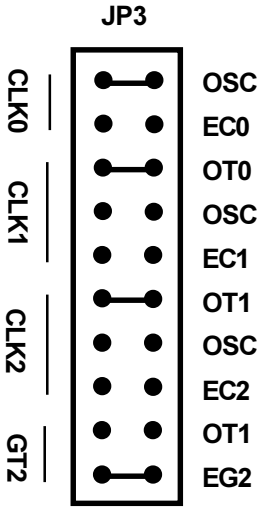


Fig. 1-4 — 8254 Clock and Gate Sources Jumpers, JP3

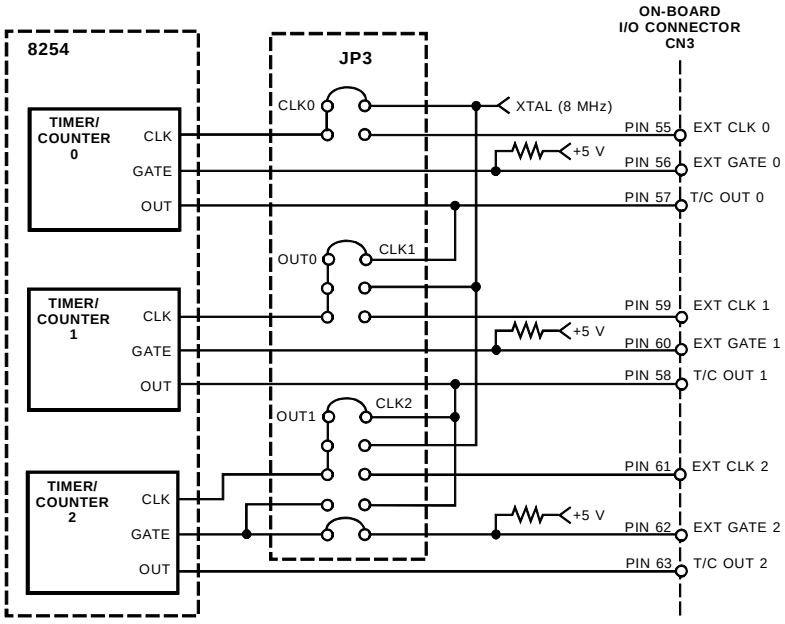


Fig. 1-5 — 8254 Circuit Diagram

JP5 — Not Used

This header connector is not used on the DM5816 and should have no jumpers installed.

JP4 — Interrupt Source Select (Factory Setting: OT2)

This header connector, shown in Figure 1-6, lets you select one of four interrupt sources for interrupt generation. These sources are not part of the Advanced Digital Interrupt circuitry. The four sources are: EI1, external interrupt 1, CN3-64; EI2, external interrupt 2, CN3-54; OT1, the output of timer/counter 1; and OT2, the output of timer/counter 2. To connect an interrupt source, place the jumper across the desired set of pins.

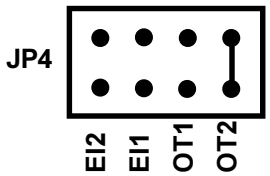


Fig. 1-6 — Interrupt Source Select Jumper, JP4

SW1 — Base Address (Factory Setting: 300 hex (768 decimal))

One of the most common causes of failure when you are first trying your module is address contention. Some of your computer's I/O space is already occupied by internal I/O and other peripherals. When the module 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 DM6916 has an easily accessible DIP switch, SW1, which lets you select any one of 16 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 one of the values listed in Table 1-2. The table shows the switch settings and their corresponding decimal and hexadecimal (in parentheses) values. Make sure that you verify the order of the switch numbers on the switch (1 through 4) before setting them. When the switches are pulled forward, they are OPEN, or set to logic 1, as labeled on the DIP switch package. When you set the base address for your module, record the value in the table inside the back cover. Figure 1-7 shows the DIP switch set for a base address of 300 hex (768 decimal).

Table 1-2 Base Address Switch Settings, SW1

Base Address Decimal / (Hex)	Switch Setting 4 3 2 1	Base Address Decimal / (Hex)	Switch Setting 4 3 2 1
512 / (200)	0 0 0 0	768 / (300)	1 0 0 0
544 / (220)	0 0 0 1	800 / (320)	1 0 0 1
576 / (240)	0 0 1 0	832 / (340)	1 0 1 0
608 / (260)	0 0 1 1	864 / (360)	1 0 1 1
640 / (280)	0 1 0 0	896 / (380)	1 1 0 0
672 / (2A0)	0 1 0 1	928 / (3A0)	1 1 0 1
704 / (2C0)	0 1 1 0	960 / (3C0)	1 1 1 0
736 / (2E0)	0 1 1 1	992 / (3E0)	1 1 1 1
0 = closed, 1 = open			

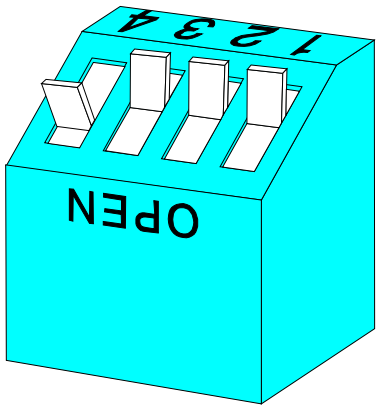


Fig. 1-7 — Base Address Switch, SW1

CHAPTER 2

MODULE INSTALLATION

The DM6916 is easy to install in your cpuModule™ or other PC/104 based system. This chapter tells you step-by-step how to install and connect the module.

After you have installed the module and made all of your connections, you can turn your system on and run the 6916DIAG board diagnostics program included on your example software disk to verify that your module is working.

Module Installation

Keep the module in its antistatic bag until you are ready to install it in your cpuModule™ or other PC/104 based system. When removing it from the bag, hold the module at the edges and do not touch the components or connectors.

Before installing the module in your system, 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 module operation and erratic response.

The DM6916 comes with a stackthrough bus connector. The stackthrough connector lets you stack another module on top of your DM6916.

To install the module, follow the procedures described in the computer manual and the steps below:

1. Turn OFF the power to your system.
2. Touch a metal rack to discharge any static buildup and then remove the module from its antistatic bag.
3. Select the appropriate standoffs for your application to secure the module when you install it in your system (two sizes are included with the module).
4. Holding the module by its edges, orient it so that the CN1 bus connector's pin 1 lines up with pin 1 of the expansion connector onto which you are installing the module.
5. After carefully positioning the module so that the pins are lined up and resting on the expansion connector, gently and evenly press down on the module until it is secured on the connector.

   NOTE: Do not force the module onto the connector. If the module does not readily press into place, remove it and try again. Wiggling the module or exerting too much pressure can result in damage to the DM6916 or to the mating module.
6. After the module is installed, connect the cable to I/O connector CN3 on the module.
7. Make sure all connections are secure.

External I/O Connections

Figure 2-1 shows I/O connector pinout for CN3. Refer to this diagram as you make your I/O connections.

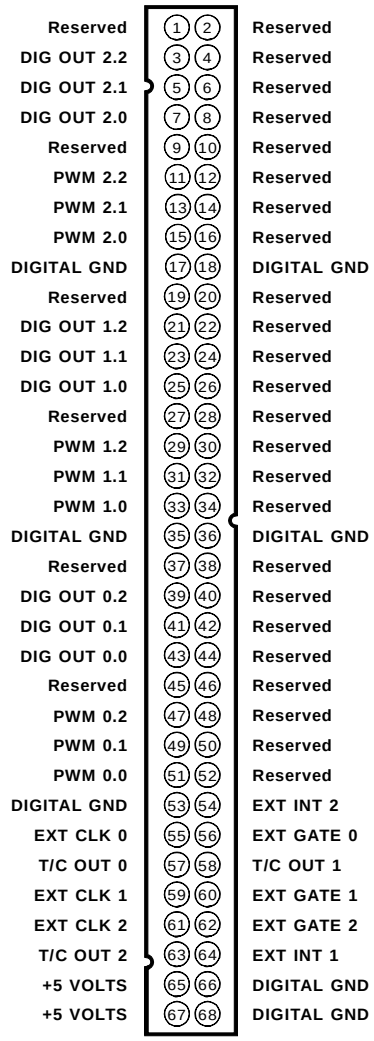


Fig. 2-1 — CN3 I/O Connector Pin Assignment

Connecting the PWM and Digital Outputs

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

Connecting the Timer/Counter I/O

External connections to the timer/counters on the DM6916 can be made by connecting the high side of the external device to the appropriate signal pin on I/O connector CN3 and the low side to a DIGITAL GND.

Connecting the External Interrupt

The DM6916 can receive externally generated interrupt signals – EXT INT1, through I/O connector CN3, pin 64, and EXT INT2, through I/O connector CN3, pin 54 – and route them to an IRQ channel through on-board header connectors JP4, JP1 and JP2. Interrupt generation is enabled through software. When interrupts are enabled, a rising or falling edge on the EXT INT line will cause the selected IRQ line to go high, depending on the setting of BA + 16, bit 1, and the IRQ status bit will change from 0 to 1. The pulse applied to the EXT INT pin should have a duration of at least 100 nanoseconds.

Running the 6916DIAG Diagnostics Program

Now that your module is ready to use, you will want to try it out. An easy-to-use, menu-driven diagnostics program, 6916DIAG, is included with your example software to help you verify your module'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 DM6916 hardware. The major circuits are the Pulse Width Modulators and the timer/ counters.

The DM6916 has two major circuits, the pulse width modulators and the timer/counters. Figure 3-1 shows the block diagram of the module. This chapter describes the hardware which makes up the major circuits.

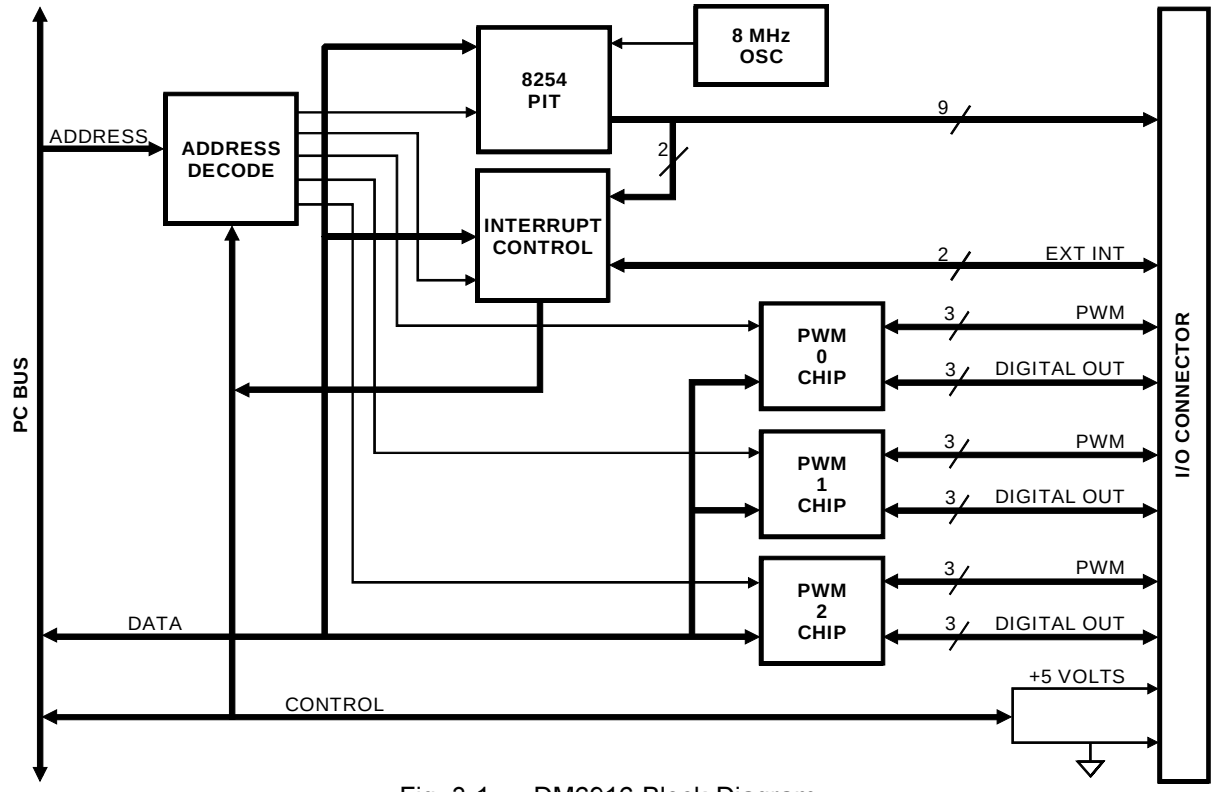


Fig. 3-1 — DM6916 Block Diagram

Pulse Width Modulators

The DM6916 has three pulse width modulator circuits. Each circuit has one 8-bit up/down counter and three 8-bit compare registers to generate the PWM output. Nine digital output lines are available for control functions (see I/O connector pinout diagrams in Appendix B).

Timer/Counters

An 8254 programmable interval timer provides three 16-bit, 8-MHz timer/counters to support a wide range of timing and counting functions. Figure 3-2 shows the timer/counter circuitry.

Each 16-bit timer/counter has two inputs, CLK in and GATE in, and one output, timer/counter OUT. Each 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.

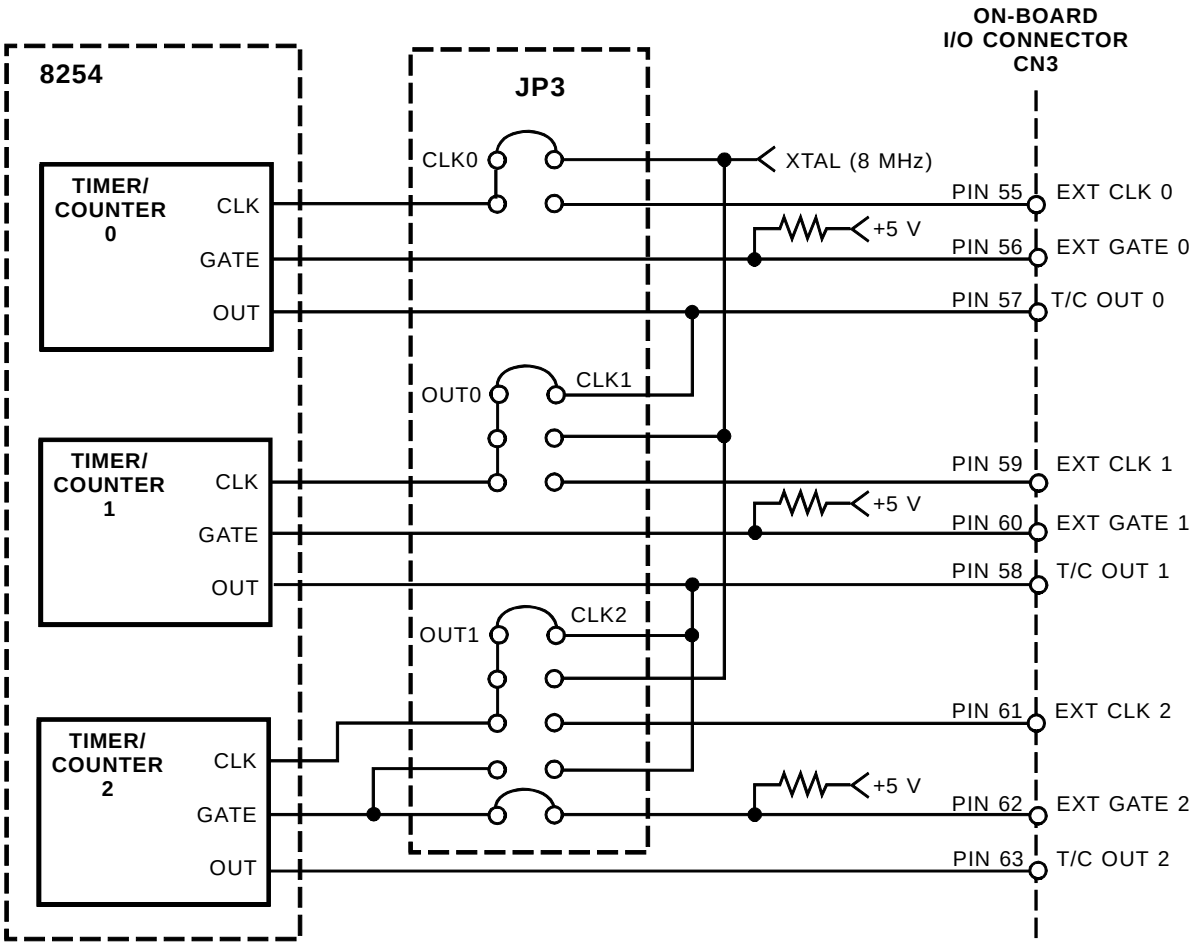


Fig. 3-2 — Timer/Counter Circuit Block Diagram

CHAPTER 4

I/O MAPPING

This chapter provides a complete description of the I/O map for the DM6916, general programming information, and how to set and clear bits in a port.

Defining the I/O Map

The I/O map for the DM6916 is shown in Table 4-1 below. As shown, the board occupies 20 consecutive I/O port locations.

The base address (designated as BA) can be selected using DIP switch SW1, located on the edge of the module, as described in Chapter 1, *Module Settings*. This switch can be accessed without removing the module from the stack. The following sections describe the register contents of each address used in the I/O map.

Table 4-1 DM6916 I/O Map

Register Description	Read Function	Write Function	Address * (Decimal)
PWM 0.0 Data	Reserved	8-bit PWM 0.0 duty cycle value	BA + 0
PWM 0.1 Data	Reserved	8-bit PWM 0.1 duty cycle value	BA + 1
PWM 0.2 Data	Reserved	8-bit PWM 0.2 duty cycle value	BA + 2
PWM 0 Control Word	Reserved	Program PWM 0 control register	BA + 3
PWM 1.0 Data	Reserved	8-bit PWM 1.0 duty cycle value	BA + 4
PWM 1.1 Data	Reserved	8-bit PWM 1.1 duty cycle value	BA + 5
PWM 1.2 Data	Reserved	8-bit PWM 1.2 duty cycle value	BA + 6
PWM 1 Control Word	Reserved	Program PWM 1control register	BA + 7
PWM 2.0 Data	Reserved	8-bit PWM 2.0 duty cycle value	BA + 8
PWM 2.1 Data	Reserved	8-bit PWM 2.1 duty cycle value	BA + 9
PWM 2.2 Data	Reserved	8-bit PWM 2.2 duty cycle value	BA + 10
PWM 2 Control Word	Reserved	Program PWM 2 control register	BA + 11
8254 TC Counter 0	Read value in Counter 0	Load count in Counter 0	BA + 12
8254 TC Counter 1	Read value in Counter 1	Load count in Counter 1	BA + 13
8254 TC Counter 2	Read value in Counter 2	Load count in Counter 2	BA + 14
8254 Control Word	Reserved	Program counter mode	BA + 15
Clear IRQ/IRQ Enable	Clear interrupt line (JP4)	Enable interrupt line (JP4), Disable interrupt sharing	BA + 16
IRQ Status	Read interrupt status	Reserved	BA + 17
Reserved	Reserved	Reserved	BA + 18
Reserved	Reserved	Reserved	BA + 19
* BA = Base Address			

BA + 0: Pulse Width Modulator 0.0 Duty Cycle (Write only)

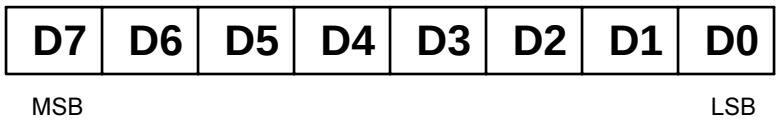
This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 1: Pulse Width Modulator 0.1 Duty Cycle (Write only)

This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 2: Pulse Width Modulator 0.2 Duty Cycle (Write only)

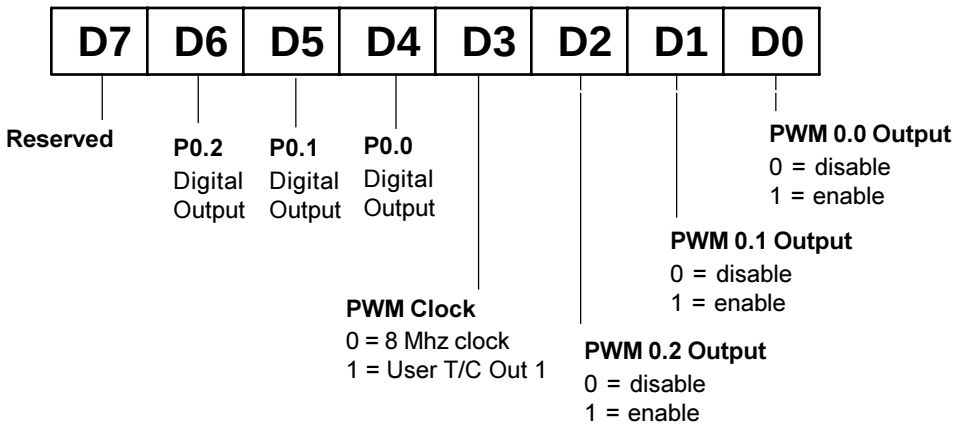
This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.



Typical of all PWM registers.

BA + 3: PWM Control and Digital Output Register (Write Only):

Bit 0 — 0 = disables PWM 0.0, 1 = enables PWM 0.0
Bit 1 — 0 = disables PWM 0.1, 1 = enables PWM 0.1
Bit 2 — 0 = disables PWM 0.2, 1 = enables PWM 0.2
Bit 3 — 0 = 8 MHz clock, 1 = output of timer 1 for clock
Bit 4 — Digital Output 0.0
Bit 5 — Digital Output 0.1
Bit 6 — Digital Output 0.2
Bit 7 — Reserved (Write with a 0)



BA + 4: Pulse Width Modulator 1.0 Duty Cycle (Write only)

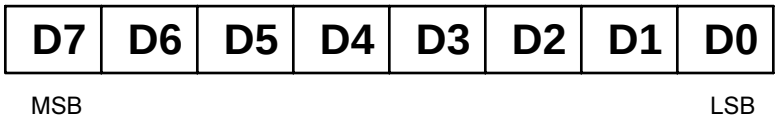
This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 5: Pulse Width Modulator 1.1 Duty Cycle (Write only)

This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 6: Pulse Width Modulator 1.2 Duty Cycle (Write only)

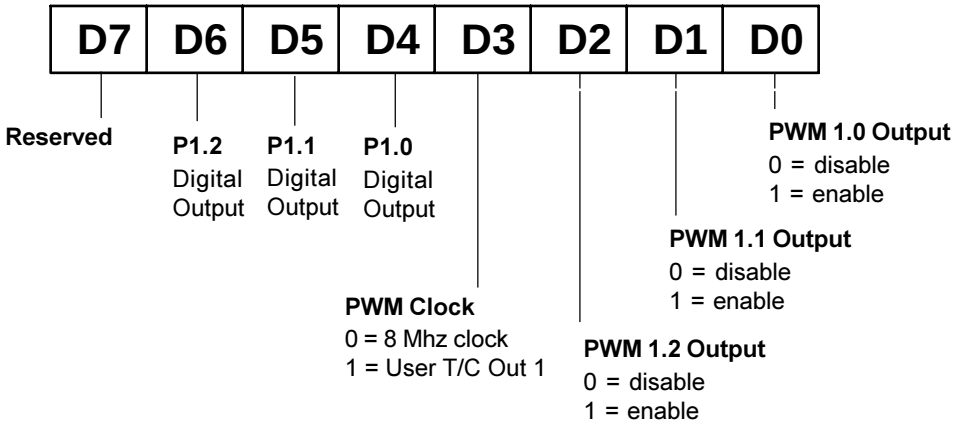
This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.



Typical of all PWM registers.

BA + 7: PWM Control and Digital Output Register (Write Only):

Bit 0 — 0 = disables PWM 1.0, 1 = enables PWM 1.0
Bit 1 — 0 = disables PWM 1.1, 1 = enables PWM 1.1
Bit 2 — 0 = disables PWM 1.2, 1 = enables PWM 1.2
Bit 3 — 0 = 8 MHz clock, 1 = output of timer 1 for clock
Bit 4 — Digital Output 1.0
Bit 5 — Digital Output 1.1
Bit 6 — Digital Output 1.2
Bit 7 — Reserved (Write with a 0)



BA + 8: Pulse Width Modulator 2.0 Duty Cycle (Write only)

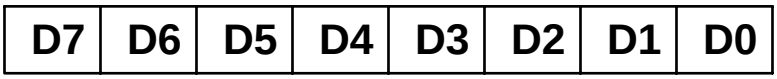
This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 9: Pulse Width Modulator 2.1 Duty Cycle (Write only)

This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

BA + 10: Pulse Width Modulator 2.2 Duty Cycle (Write only)

This 8-bit port is the duty cycle register of the pulse width modulator. A write sets the duty cycle to DATA/256. The data will be 0 to 255.

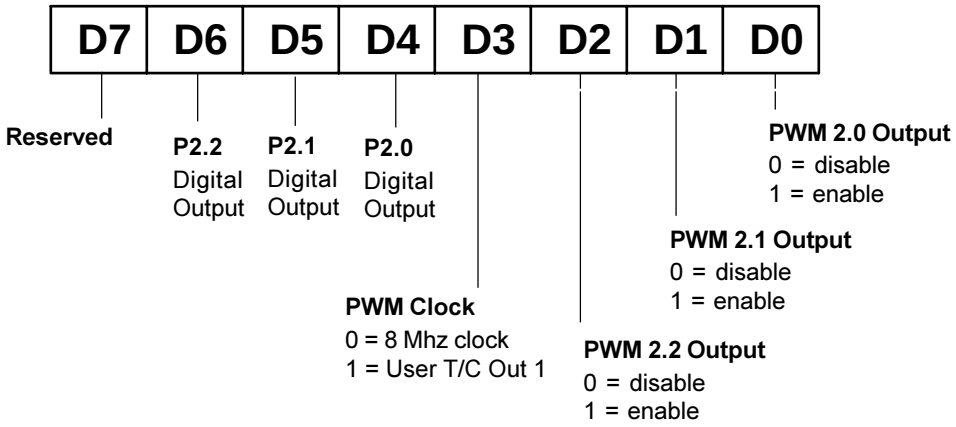


MSB LSB

Typical of all PWM registers.

BA + 11: PWM Control and Digital Output Register (Write Only):

Bit 0 — 0 = disables PWM 2.0, 1 = enables PWM 2.0
Bit 1 — 0 = disables PWM 2.1, 1 = enables PWM 2.1
Bit 2 — 0 = disables PWM 2.2, 1 = enables PWM 2.2
Bit 3 — 0 = 8 MHz clock, 1 = output of timer 1 for clock
Bit 4 — Digital Output 2.0
Bit 5 — Digital Output 2.1
Bit 6 — Digital Output 2.2
Bit 7 — Reserved (Write with a 0)



BA + 12: 8254 Timer/Counter 0 (Read/Write)

This address is used to read/write timer/counter 0. A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 13: 8254 Timer/Counter 1 (Read/Write)

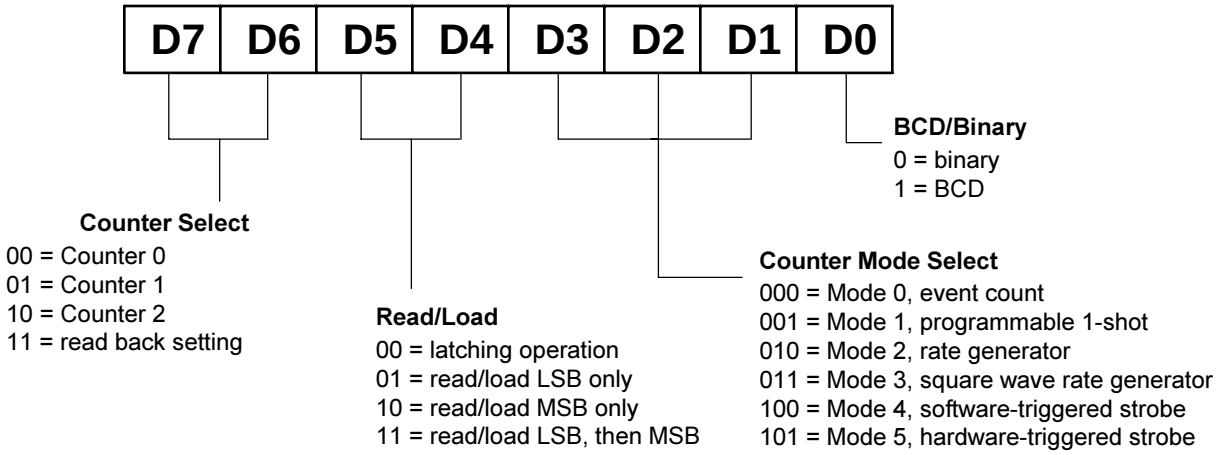
This address is used to read/write timer/counter 1. A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 14: 8254 Timer/Counter 2 (Read/Write)

This address is used to read/write timer/counter 2. A read shows the count in the counter, and a write loads the counter with a new value. Counting begins as soon as the count is loaded.

BA + 15: 8254 Timer/Counter Control Word (Write Only)

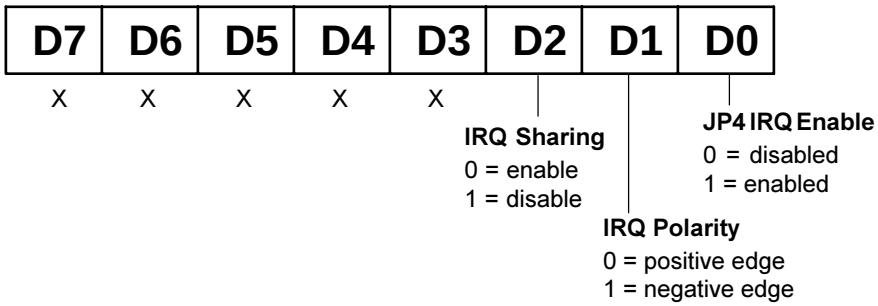
This address is used to write to the control register for the 8254. The control word is defined below.



BA + 16: Clear IRQ/IRQ Enable (Read/Write)

A read clears the JP4 jumper-selectable IRQ status flag at BA + 17, bit 3.

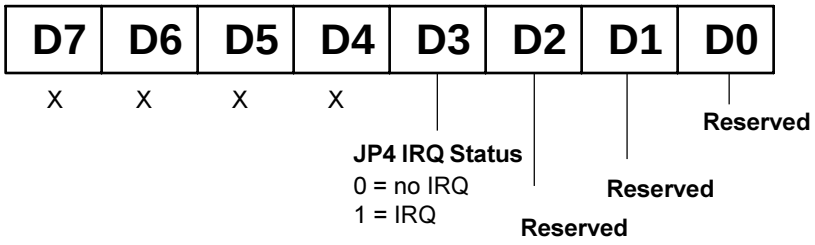
IRQ Enable Register:



A write enables JP4 interrupts and selects whether the interrupt will occur on the positive (rising) edge or negative (falling) edge of the pulse. Bit 2 is used to enable and disable the interrupt sharing circuit. If you are not using shared interrupts, it is best to disable this feature to ensure compatibility with all CPUs.

BA + 17: IRQ Status (Read Only)

A read shows the status of the jumper-selectable interrupt circuit at JP4.



BA + 18: Reserved

BA + 19: Reserved

Programming the DM6916

This section gives you some general information about programming and the DM6916.

The module is programmed by reading from and writing to the correct I/O port locations. 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 DM6916, 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 C, Pascal, 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	\ 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 DM6916!**

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. Note that most registers in the DM6916 cannot be read back; therefore, you must save the value in your program.

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_SAVE = V_SAVE 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_Save = V_Save OR 8;
Port[PortAddress] := V_Save;
```

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_save = v_save & 171;
outportb(port_address, v_save);
```

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 al, v_save
or al, 168
mov dx, PortAddress
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_save = v_save & 199;
v_save = v_save | 40;
outportb(port_address, v_save);
```

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.

CHAPTER 5

DIGITAL OUTPUTS

This chapter explains the digital output circuitry on the DM6916.

The DM6916 has three TTL/CMOS digital output lines available for digital control applications on each of the three pulse width modulator chips. Chapter 4 shows how to program these lines.

CHAPTER 6

TIMER/COUNTERS

This chapter explains the 8254 timer/counter circuit on the DM6916.

An 8254 programmable interval timer provides three 16-bit, 8-MHz timers for timing and counting functions such as frequency measurement, event counting, and interrupts. These timer/counters can be configured in a number of ways to support your application. Figure 6-1 shows a block diagram of the timer/counter circuitry.

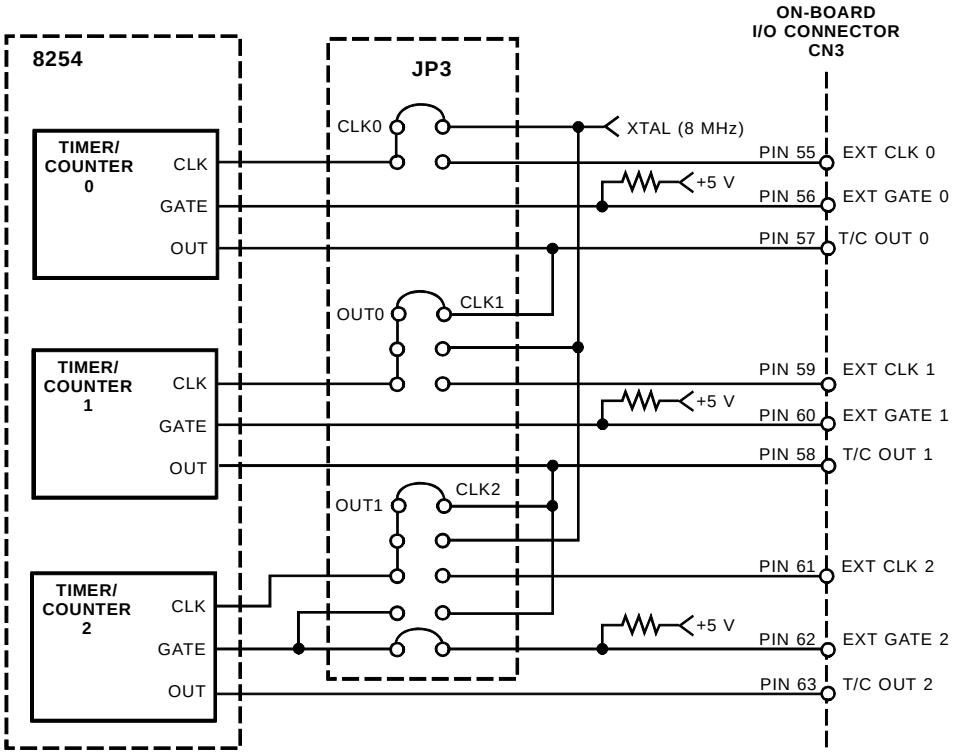


Fig. 6-1 — 8254 Timer/Counter Circuit Block Diagram

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 discussion in Chapter 4.

The timer/counter outputs are available at P6 where they can be used for interrupt generation, as an A/D trigger, or for timing and counting functions.

The timers can be programmed to operate in one of six modes, depending on your application. 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.

Appendix C provides the 8254 data sheet.

CHAPTER 7

INTERRUPTS

This chapter explains JP4 jumper selectable interrupts and basic interrupt programming techniques.

The DM6916 has one interrupt circuit which can generate interrupts on any IRQ channel 2 through 15, depending on the setting of the jumper on JP1 or JP2.

JP4: Jumper Selectable Interrupts

The DM6916 circuitry has four jumper selectable interrupt sources which can be set by installing a jumper across the desired pair of pins at JP4.

To use these interrupts, an interrupt source must be jumpered on JP4, an interrupt channel must be jumpered on JP1 or JP2, and the IRQ enable must be set high (BA + 16, bit 0). BA + 16, bit 1 sets the polarity of the interrupt.

Selecting the Interrupt Channel

The IRQ channel is selected by installing a jumper on header connector JP1 or JP2 across the desired pair of pins, as described in Chapter 1. A jumper is also installed across the G pins if you are using the interrupt sharing feature.

Interrupt Sharing

This module is capable of sharing interrupts with multiple modules. This circuit is described in chapter 1. If you are not planning on using shared interrupts or you are not sure that your CPU can support shared interrupts, you should disable this sharing circuit by setting bit 2 at BA + 16 to a "1". By doing this the board works in normal interrupt mode and is compatible with all CPUs.

Basic Programming For Interrupt Handling

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 DM6916 can interrupt the processor when a variety of conditions are met. By using these interrupts, you can write software that effectively deals with real world events.

Interrupt Request Lines

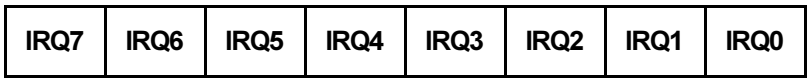
To allow different peripheral devices to generate interrupts on the same computer, the PC bus has eight different interrupt request (IRQ) lines. A transition from low to high on one of these lines generates an interrupt request which is handled by the PC's interrupt controller. The interrupt controller 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 module.

8259 Programmable Interrupt Controller

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

- Interrupt Mask Register (IMR)

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



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

- End-of-Interrupt (EOI) Command

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

What Exactly Happens When an Interrupt Occurs?

Understanding the sequence of events when an interrupt is triggered is necessary to properly write software interrupt handlers. When an interrupt request line is driven high by a peripheral device (such as the DM6916), 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 DM6916 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 clear the interrupt status flag of the DM6916 and write an end-of-interrupt command to the 8259 controller. Finally, when exiting the ISR, in addition to popping all the registers you pushed on entrance, you must use the IRET instruction and **not** a plain RET. The IRET automatically pops the flags, CS, and IP that were pushed when the interrupt was called.

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

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

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

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

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

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

Your ISR should have this structure:

- Push any processor registers used in your ISR. Most C and Pascal interrupt routines automatically do this for you.
- Put the body of your routine here.
- Clear the interrupt status flag for the source which caused the interrupt:
  - Clear jumper selectable interrupt status flag by reading BA + 16.
  - Clear the Port 0 digital interrupt flag by setting bits 1 and 0 at BA + 3 to 00 and reading BA + 2.
  - Clear the Port 2 digital interrupt flag by setting bits 1 and 0 at BA + 7 to 00 and reading BA + 6.
  - Clear the Port 4 digital interrupt flag by setting bits 1 and 0 at BA + 11 to 00 and reading BA + 10.
- 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. Only the clear interrupt command sequence for the source which caused the interrupt needs to be included:

In C:

```
void interrupt ISR(void)
{
    /* Your code goes here. Do not use any DOS functions! */
    inportb(BaseAddress + 16);              /* Clear jumper selectable interrupt */
    outportb(BaseAddress + 3, 0);           /* Set Port 0 digital I/O clear mode */
    inportb(BaseAddress + 2);               /* Clear Port 0 digital interrupt */
    outportb(BaseAddress + 7, 0);           /* Set Port 2 digital I/O clear mode */
    inportb(BaseAddress + 6);               /* Clear Port 2 digital interrupt */
    outportb(BaseAddress + 11, 0);          /* Set Port 4 digital I/O clear mode */
    inportb(BaseAddress + 10);              /* Clear Port 4 digital interrupt */
    outportb(0x20, 0x20);                   /* Send EOI command to 8259 */
}
```

In Pascal:

```
Procedure ISR; Interrupt;
begin
    { Your code goes here. Do not use any DOS functions! }
    c := Port[BaseAddress + 16];            { Clear jumper selectable interrupt }
    Port[BaseAddress + 3] := 0;             { Set Port 0 digital I/O clear mode }
    c := Port[BaseAddress + 2];             { Clear Port 0 digital interrupt }
    Port[BaseAddress + 7] := 0;             { Set Port 2 digital I/O clear mode }
    c := Port[BaseAddress + 6];             { Clear Port 2 digital interrupt }
    Port[BaseAddress + 11] := 0;            { Set Port 4 digital I/O clear mode }
    c := Port[BaseAddress + 10];            { Clear Port 4 digital interrupt }
    Port[$20] := $20;                       { Send EOI command to 8259 }
end;
```

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

The next step after writing the ISR is to save the startup state of the interrupt mask register and the interrupt vector that you will be using. The IMR is located at I/O port 21H. The interrupt vector you will be using is located in the interrupt vector table which is simply an array of 256-bit (4-byte) pointers and is located in the first 1024 bytes of memory (Segment = 0, Offset = 0). You can read this value directly, but it is a better practice to use DOS function 35H (get interrupt vector). Most C and Pascal compilers provide a library routine for reading the value of a vector. The vectors for the hardware interrupts are vectors 8 through 15, where IRQ0 uses vector 8, IRQ1 uses vector 9, and so on. Thus, if the DM6916 will be using IRQ3, you should save the value of interrupt vector 11.

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

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

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

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

Restoring the Startup IMR and Interrupt Vector

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

Common Interrupt Mistakes

- Remember that hardware interrupts are numbered 8 through 15, even though the corresponding IRQs are numbered 0 through 7.
- Two of the most common mistakes when writing an ISR are forgetting to clear the interrupt status of the DM6916 and forgetting to issue the EOI command to the 8259 interrupt controller before exiting the ISR.

APPENDIX A

DM6916 SPECIFICATIONS

DM6916 Characteristics Typical @ 25° C

Interface

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

Pulse Width Modulators

Number of channels 9
Resolution 8-bits
Max internal clock ..... 8 MHz
Max external clock ..... 10 MHz
Output type ..... TTL
Output level ..... 0 - +5 volts
Isource ..... -12 mA
Isink ..... 24 mA

Digital Outputs

Number of lines 9
I/O type ..... TTL
Output levels ..... 0 - +5 volts
Isource ..... -12 mA
Isink ..... 24 mA

Timer/Counters ..... CMOS 82C54

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

Miscellaneous Inputs/Outputs (PC bus-sourced)

±5 volts, ±12 volts, ground

Power Requirements

+5V @ 238 mA = 1.18W typical

Connectors

CN3: 68-pin right angle connector
Mating Connector Manufacturer and Part #: 3M 82068-6000

Environmental

Operating temperature ..... -40 to +85°C
Storage temperature ..... -40 to +85°C
Humidity ..... 0 to 90% non-condensing

Size

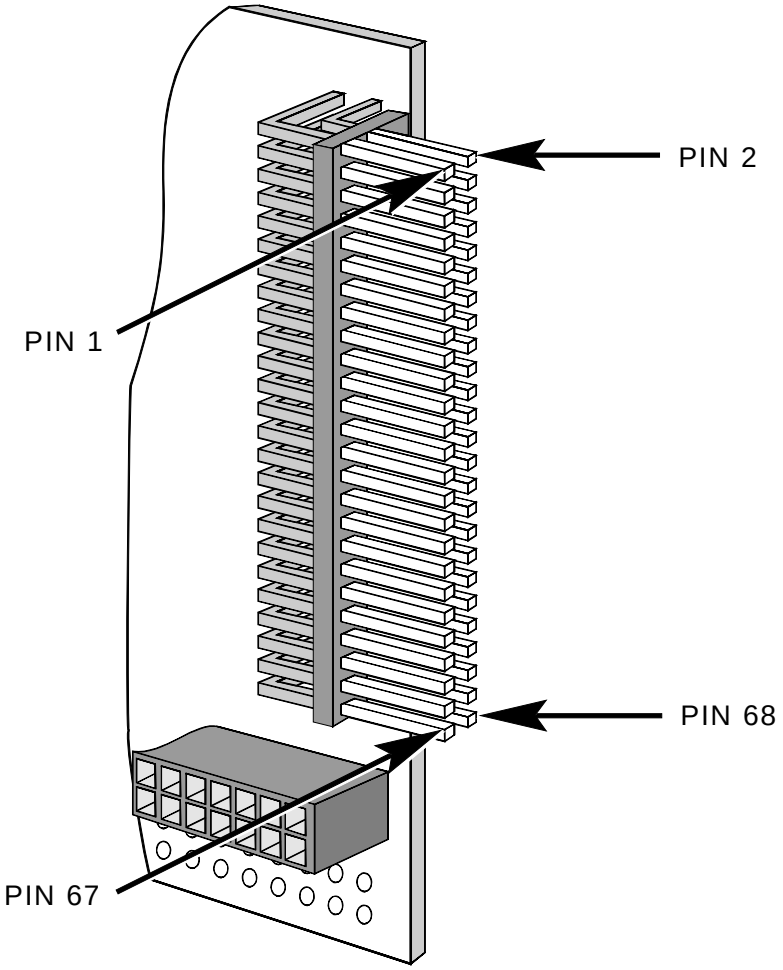
3.55"L x 3.775"W x 0.6"H (90mm x 96mm x 15mm)

APPENDIX B

CONNECTOR PIN ASSIGNMENTS

CN3 Connector:

Reserved	1	2	Reserved
DIG OUT 2.2	3	4	Reserved
DIG OUT 2.1	5	6	Reserved
DIG OUT 2.0	7	8	Reserved
Reserved	9	10	Reserved
PWM 2.2	11	12	Reserved
PWM 2.1	13	14	Reserved
PWM 2.0	15	16	Reserved
DIGITAL GND	17	18	DIGITAL GND
Reserved	19	20	Reserved
DIG OUT 1.2	21	22	Reserved
DIG OUT 1.1	23	24	Reserved
DIG OUT 1.0	25	26	Reserved
Reserved	27	28	Reserved
PWM 1.2	29	30	Reserved
PWM 1.1	31	32	Reserved
PWM 1.0	33	34	Reserved
DIGITAL GND	35	36	DIGITAL GND
Reserved	37	38	Reserved
DIG OUT 0.2	39	40	Reserved
DIG OUT 0.1	41	42	Reserved
DIG OUT 0.0	43	44	Reserved
Reserved	45	46	Reserved
PWM 0.2	47	48	Reserved
PWM 0.1	49	50	Reserved
PWM 0.0	51	52	Reserved
DIGITAL GND	53	54	EXT INT 2
EXT CLK 0	55	56	EXT GATE 0
T/C OUT 0	57	58	T/C OUT 1
EXT CLK 1	59	60	EXT GATE 1
EXT CLK 2	61	62	EXT GATE 2
T/C OUT 2	63	64	EXT INT 1
+5 VOLTS	65	66	DIGITAL GND
+5 VOLTS	67	68	DIGITAL GND



CN3 Mating Connector Part Number	
Manufacturer	**Part Number**
3M	82068-6000

APPENDIX C

COMPONENT DATA SHEETS

Intel 82C54 Programmable Interval Timer
Data Sheet Reprint

APPENDIX D

WARRANTY AND RETURN POLICY

Return Policy

If you wish to return a product to the factory for service, please follow this procedure:

Read the Limited Warranty to familiarize yourself with our warranty policy.

Contact the factory for a Return Merchandise Authorization (RMA) number.

Please have the following available:

- Complete board name
- Board serial number
- A detailed description of the board's behavior

List the name of a contact person, familiar with technical details of the problem or situation, **along with their phone and fax numbers, address, and e-mail address** (if available).

List your shipping address!!

Indicate the shipping method you would like used to return the product to you.
We will not ship by next-day service without your pre-approval.

Carefully package the product, using proper anti-static packaging.

Write the RMA number in large (1") letters on the outside of the package.

Return the package to:

RTD Embedded Technologies, Inc.

103 Innovation Blvd.

State College PA 16803-0906
USA

LIMITED WARRANTY

RTD Embedded Technologies, 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 RTD Embedded Technologies, INC. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, RTD Embedded Technologies 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 RTD Embedded Technologies. All replaced parts and products become the property of RTD Embedded Technologies. 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 RTD Embedded Technologies, "acts of God" or other contingencies beyond the control of RTD Embedded Technologies), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN RTD Embedded Technologies. 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 RTD Embedded Technologies 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 RTD Embedded Technologies 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.

RTD Embedded Technologies, Inc.
103 Innovation Blvd.
State College PA 16803-0906
USA
Our website: www.rtd.com

DM6916 User Settings	
Base I/O Address:	
(hex)	(decimal)
IRQ Channel:	