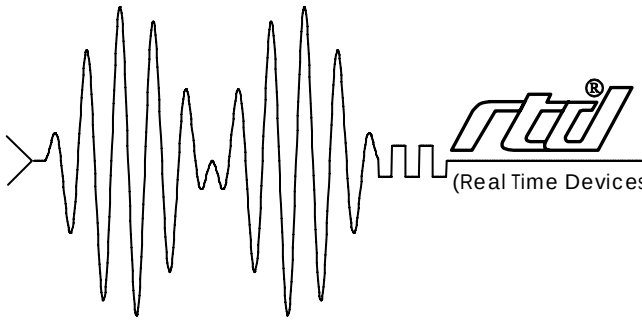
DM6620HR
User's Manual



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

BDM-610010008
Rev. A

DM6620HR
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

Table of Contents

List of Illustrations

INTRODUCTION

The DM6620 analog output dataModule® turns your IBM AT-compatible cpuModule™ or other PC/104 computer into a high-speed, high-performance waveform generator and control system. Ultra-compact for embedded and portable applications, the DM6620 module features:

- Four fast-settling 12-bit analog output channels,
- ±5, ±10, 0 to +5, or 0 to +10 volt software programmable analog output range,
- Four 1024 sample D/A buffers for gap-free high speed output under Windows™ and DOS
- Simultaneous updating of all output channels,
- Cycle mode for waveform generation,
- DMA transfer,
- 3 independent sample counters,
- 4-bit analog output data/trigger marker,
- 8 bit programmable digital I/O lines with Advanced Digital Interrupt modes,
- 8 port programmable digital I/O lines,
- Six 16-bit timer/counters (three available to user) and on-board 8 MHz clock,
- +5 volt only operation, 2.1W power consumption,
- Windows™ example programs in Visual Basic and C,
- 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.

Digital-to-Analog Conversion

The digital-to-analog (D/A) circuitry features four independent 12-bit analog output channels with individually programmable output ranges of -5 to +5 volts, 0 to +5 volts, -10 to +10 volts or 0 to +10 volts. Each channel has it's own 1024 sample buffer for data storage before being output. Data can be continuously written to the buffer producing a non-repetitive output waveform or a set of data can be written into the buffer and continuously cycled to produce a repeating waveform. Data can be written into the output buffers by I/O instruction or by DMA transfer. Updating of the analog outputs can be done through software or by several different clocks and triggers. The outputs can be updated simultaneously or independently.

8254 Timer/Counters

Two 8254 programmable interval timers provide six (three each) 16-bit, 8 MHz timer/counters to support a wide range of board operations and user timing and counting functions. The Sample Counter TC is used for board operations providing 3 independent 16-bit counters used to count output samples and generate interrupts. The User TC has three 16-bit timer/counters for user functions. The outputs from these timer/counters can be used to update the D/A outputs.

Digital I/O

The DM6620 has 16 buffered TTL/CMOS digital I/O lines which are grouped as eight independent, bit programmable lines at Port 0, and an 8-bit programmable port at Port 1. The bit programmable lines support RTD's two Advanced Digital Interrupt modes. An interrupt can be generated when any bit changes value (event interrupt), or when the lines match a programmed value (match interrupt). For either mode, masking can be used to monitor selected lines. Pull-up or pull-down resistors are provided for all 16 lines. Instructions for activating these pull-up/pull-down resistors are given at the end of Chapter 1, *Module Settings*.

What Comes With Your Module

You receive the following items in your module package:

- DM6620 interface module with stackthrough bus header
- Mounting hardware
- Windows™ example programs in Visual Basic and C
- Example programs in BASIC and C with source code & diagnostics software
- User's manual

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

Module Accessories

In addition to the items included in your module 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 module's application.

Hardware Accessories

Hardware accessories for the DM6620 include the OP series optoisolated digital input boards, the MR series mechanical relay output boards, the OR16 optoisolated digital input/mechanical relay output board, the TB50 terminal board and XB50 prototype/terminal board for easy signal access and prototype development, the DM16 extender board for testing your module in a conventional desktop computer, and XT50 twisted pair wire flat ribbon cable assembly for external interfacing.

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 board 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@rtdusa.com**.

CHAPTER 1

MODULE SETTINGS

The DM6620 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.

Also note that by placing solder connections on the bottom of the board at JS1 and JS2, you can configure each set of digital I/O lines to be pulled up or pulled down. This procedure is explained at the end of this chapter.

Factory-Configured Switch and Jumper Settings

Table 1-1 lists the factory settings of the user-configurable jumpers and switch on the DM6620 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 S1, the base address switch, to avoid address contention when you first use the module in your system.

Table 1-1 Factory Settings

Switch/ Jumper	Function Controlled	Factory Settings (Jumpers Installed)
JP1	Sets the clock source for User TC Counters 0, 1 & 2	Clk 0: XTAL; Clk 1: OT0; Clk 2: OT1 (timer/counters cascaded)
JS1	Activates pull-up/ pull-down resistors on Port 0 digital I/O lines	All bits pulled up (solder connections between COM & V)
JS2	Activates pull-up/ pull-down resistors on Port 1 digital I/O lines	All bits pulled up (solder connections between COM & V)
S1	Sets the base address	300 hex (768 decimal)

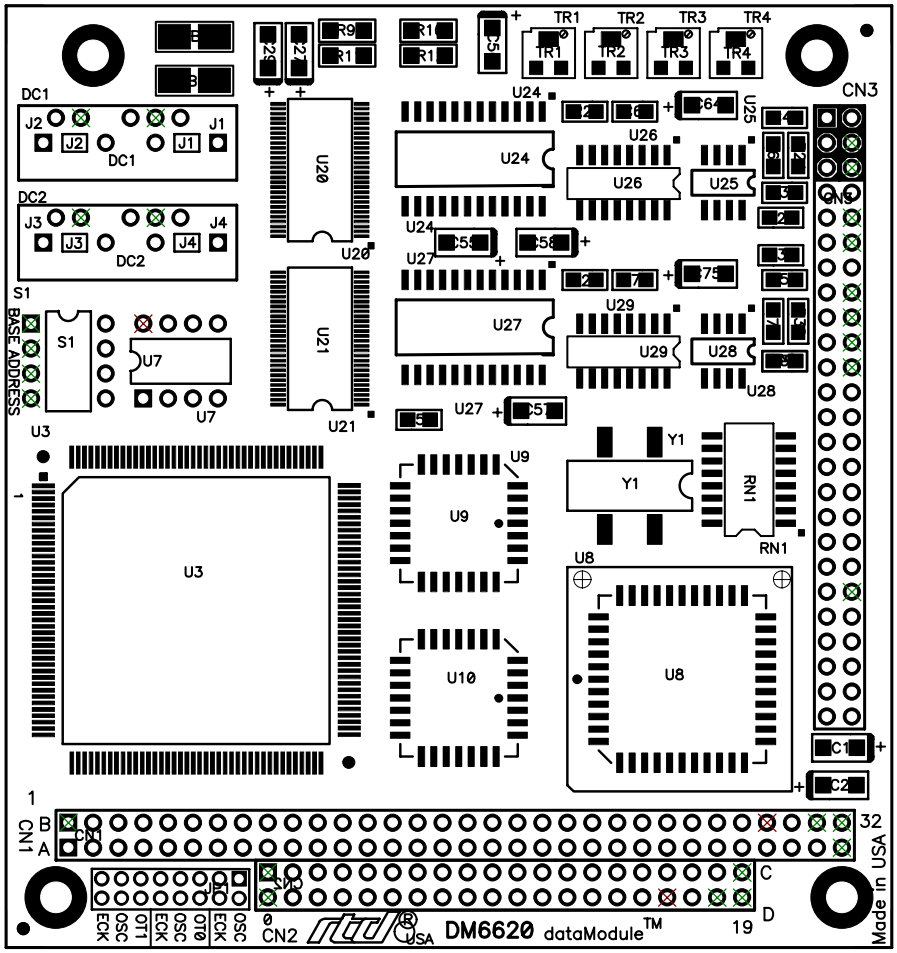


Fig. 1-1 — Module Layout Showing Factory-Configured Settings

JP1 — User TC Clock Source Select (Factory Settings: Clk 0: XTAL, Clk 1: OT0, Clk 2: OT1)

This header connector, shown in Figure 1-2, lets you select the clock sources for User TC Counters 0, 1 and 2, the 16-bit timer/counters available for user functions. Figure 1-3 shows a block diagram of the User TC 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 rightmost pairs of pins on the header, OSC or ECK. Osc is the on-board 8 MHz clock; and ECK is an external pacer clock which can be connected through I/O connector CN3, pin 39.

The next three pins, OT0, OSC and ECK, set the clock source for timer/counter Counter 1. OT0 is the output of Counter 0; OSC is the on-board 8 MHz clock; and ECK is an external pacer clock which can be connected through I/O connector CN3, pin 39.

The last three pins, OT1, OSC and ECK, set the clock source for timer/counter Counter 2. OT1 is the output of Counter 1; OSC is the on-board 8 MHz clock; and ECK is an external pacer clock which can be connected through I/O connector CN3, pin 39.

Counters 0, 1 and 2 are factory set as a 48-bit cascaded counter clocked by the 8 MHz system clock.

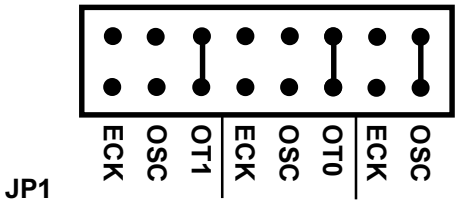


Fig. 1-2 — User TC Clock Sources Jumpers, JP1

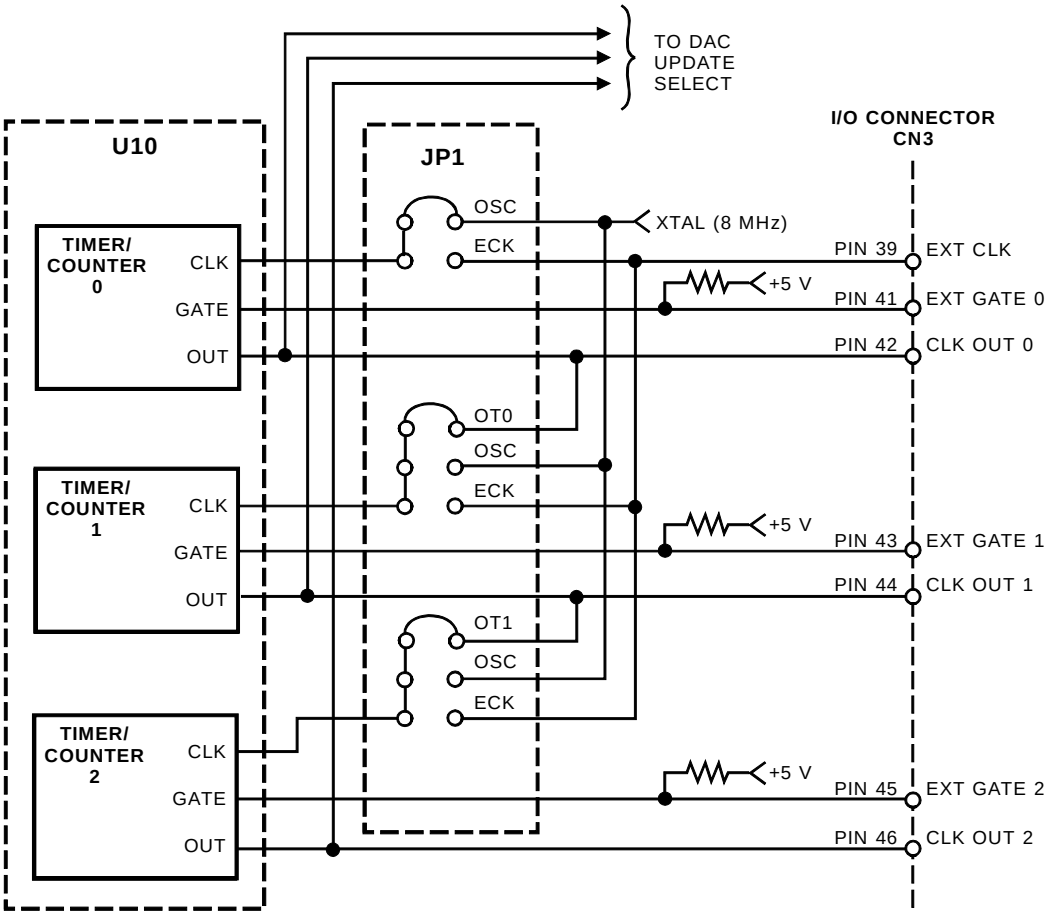


Fig. 1-3 — User TC Circuit Diagram

S1 — 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 DM6620 has an easily accessible DIP switch, S1, 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, S1

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			

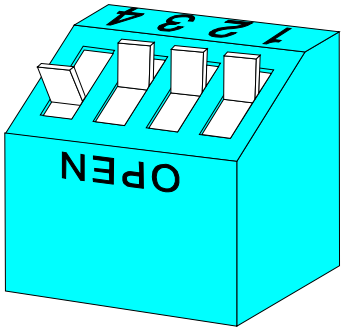


Fig. 1-4 — Base Address Switch, S1

JS1 and JS2, Pull-up/Pull-down Resistors on Digital I/O Lines

The DM6620 has 16 TTL/CMOS compatible digital I/O lines which can be interfaced with external devices. These lines are divided into two groups: Port 0 with eight individual bit programmable lines, and Port 1 with eight port programmable lines. Resistors are connected to these lines and can be configured as either pull-up or pull-down resistors.

10 k ohm pull-up/pull-down resistors are installed on the module, and a solder connection must be made on the bottom of the board to configure their operation. The solder connections are made at JS1 for Port 0 and JS2 for Port 1. The factory default is pull-up for both ports. This is done by placing a solder short between the middle (common) pad and V (+5 volts). To configure the resistors as pull-down resistors, remove the existing solder connection and make one between the middle (common) pad and G (ground). To disable the pull-up/pull-down resistor, remove the solder connection.

WARNING: Do not install a connection between all three pads as this will damage the board!!

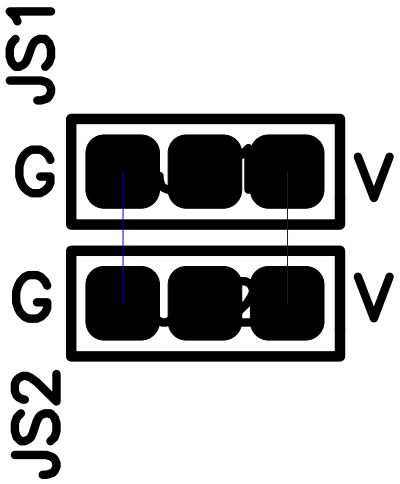


Fig. 1-5 — Ports 0 and 1 Pull-up/Pull-down Resistor Connections

CHAPTER 2

INSTALLATION

The DM6620 is easy to install in your cpuModule™ or other PC/104 based system. This chapter tells you step-by-step how to connect the module.

After you have made all of your connections, you can turn your system on and run the 6620DIAG board diagnostics program included on your example software disk to verify that the module is working.

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 DM6620 comes with stackthrough connectors for CN1 and CN2. These stackthrough connectors let you stack another module on top of your DM6620. Pins B10 and C19 are keying pins and will be plugged on the top of the board and removed from the bottom

NOTE: The DM6620 module will only work with an AT cpuModule. Do not try to use it with an XT cpuModule.

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.
4. Holding the module by its edges, orient it so that the 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 DM6420 or to the mating module.
6. After the module is installed, connect the cable to I/O connector CN3 on the module. When making this connection, note that there is no keying to guide you in orientation. You must make sure that pin 1 of the cable is connected to pin 1 of CN3 (pin 1 is marked on the module with a small square). For twisted pair cables, pin 1 is the dark brown wire; for standard single wire cables, pin 1 is the red wire.
7. Make sure all connections are secure.

External I/O Connections

Figure 2-1 shows the DM6620's CN3 I/O connector pinout. Refer to this diagram as you make your I/O connections. Note that +12 volts at pin 47 and -12 volts at pin 49 are available only if your computer bus supplies them (these voltages are not provided by the module).

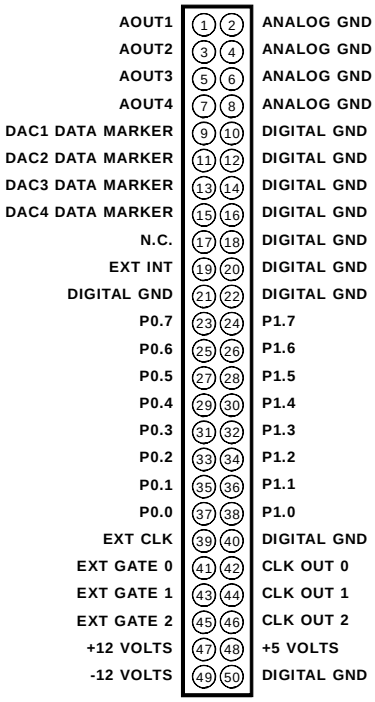


Fig. 2-1 — CN3 I/O Connector Pin Assignments

Connecting the Analog Outputs

For each of the four D/A outputs, connect the high side of the device receiving the output to the AOUT channel (CN3-1, CN3-3, CN3-5 or CN3-7) and connect the low side of the device to an ANALOG GND (CN3-2, 4, 6, 8).

Connecting the Timer/Counters, Digital I/O and DataMarkers

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 6620DIAG 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, 6620DIAG, 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 DM6620 hardware. The major circuits are the D/A, the timer/counters, and the digital I/O lines.

The DM6620 has three major circuits, the D/A, the timer/counters, and the digital I/O lines. Figure 3-1 shows the block diagram of the module. This chapter describes the hardware which makes up the major circuits.

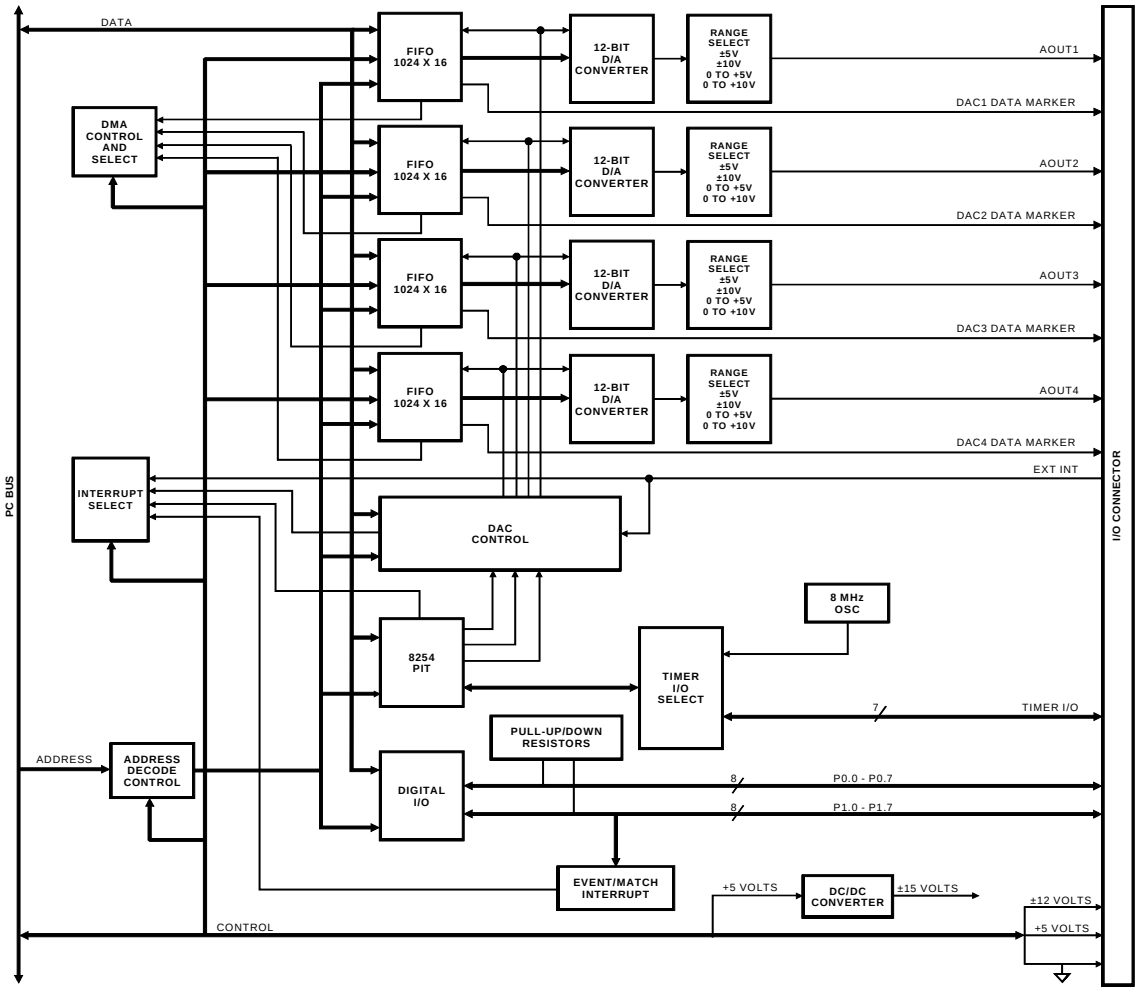


Fig. 3-1 — DM6620 Block Diagram

Digital-to-Analog Conversion

The digital-to-analog (D/A) circuitry features four independent 12-bit analog output channels with individually programmable output ranges of -5 to +5 volts, 0 to +5 volts, -10 to +10 volts or 0 to +10 volts. Each channel has it's own 1024 sample buffer for data storage before being output. Data can be continuously written to the buffer producing a non-repetitive output waveform or a set of data can be written into the buffer and continuously cycled to produce a repeating waveform. Data can be written into the output buffers by I/O instruction or by DMA transfer. Updating of the analog outputs can be done through software or by several different clocks and triggers. The outputs can be updated simultaneously or independently.

Timer/Counters

Two 8254 programmable interval timers provide six 16-bit, 8-MHz timer/counters to support a wide range of timing and counting functions. The 8254 at U9 is the Sample Counter TC. All three of these 16-bit timer/counters, Counter 0, Counter 1 and Counter 2, can be programmed as sample counters and used to generate interrupts. These are useful for signaling the CPU to load new data into the D/A buffer.

The 8254 at U10 is the User TC. On the User TC, Counters 0, Counter 1 and Counter 2 are available to the user. These timer/counters can also be used to generate clocks as update signals for the D/A converters.

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. The sample counters should be programmed for mode 2 operation.

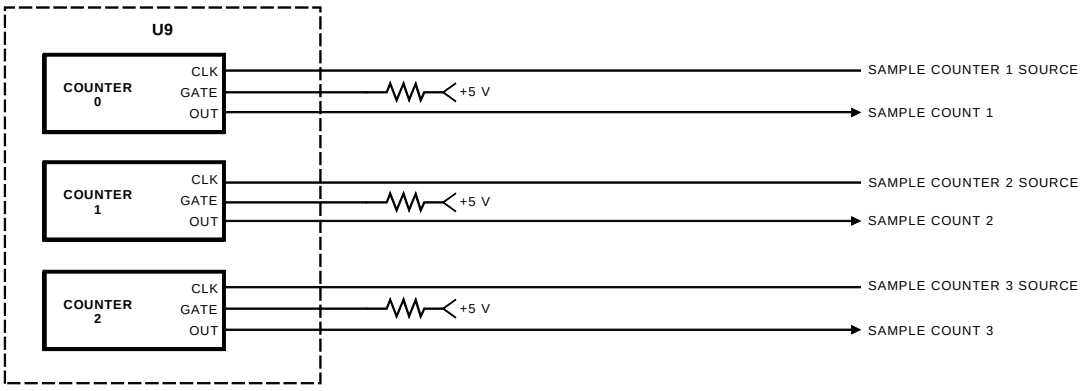


Fig. 3-2 — Sample Counter TC Circuit Block Diagram

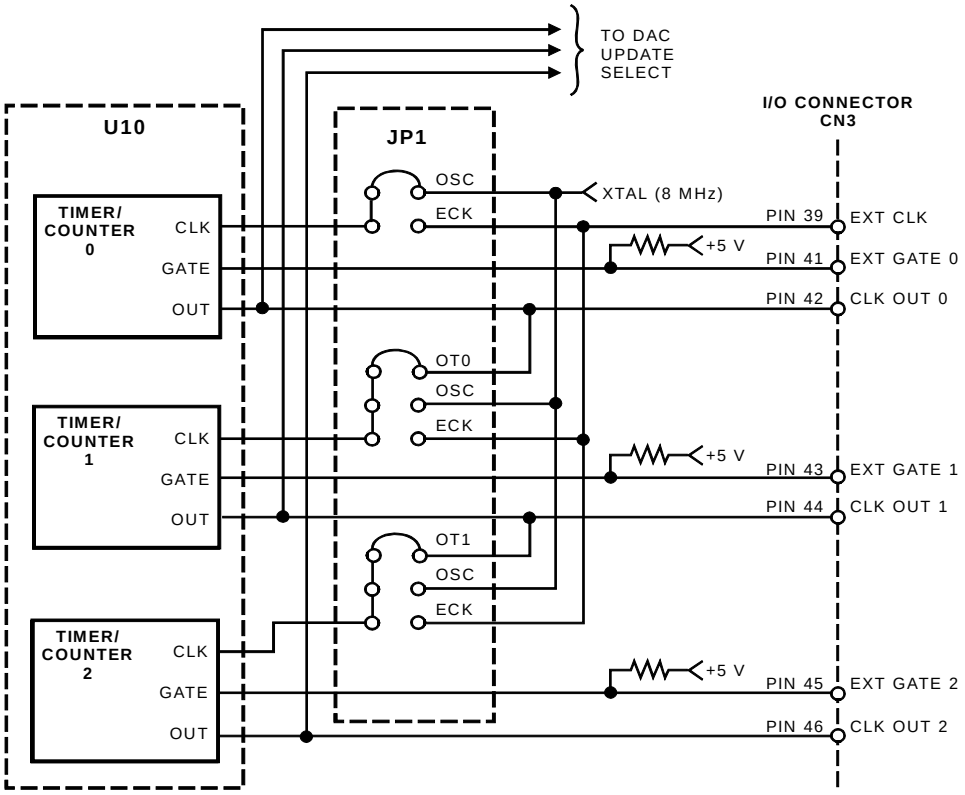


Fig. 3-3 — User TC Circuit Block Diagram

Digital I/O

The 16 digital I/O lines can be used to transfer data between the computer and external devices. Eight lines are bit programmable and eight lines are byte, or port, programmable.

Port 0 provides eight bit programmable lines which can be independently set for input or output. Port 0 supports RTD's two Advanced Digital Interrupt modes. An interrupt can be generated when the lines match a programmed value or when any bit changes its current state. A Mask Register lets you monitor selected lines for interrupt generation. A 1024 sample buffer is also connected to Port 0 to provide buffering for high speed digital inputs.

Port 1 can be programmed as an 8-bit input or output port.

Chapter 10 details digital I/O operations and Chapter 7 explains digital interrupts.

CHAPTER 4

I/O MAPPING

This chapter provides a complete description of the I/O map for the DM6620, general programming information, and how to set and clear bits in a port.

Defining the I/O Map

The I/O map for the DM6620 is shown in Table 4-1 below. As shown, the board occupies 32 consecutive I/O port locations.

The base address (designated as BA) can be selected using DIP switch S1, located on the edge of the board as described in Chapter 1, *Board 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 DM6620 I/O Map

Register Description	Read Function	Write Function	Address * (Decimal)
Board ID	Read Board ID	Reserved	BA + 0
Set IRQ Source & Channel	Reserved	Program IRQ register	BA + 2
Set DMA Source & Channel	Reserved	Program DMA register	BA + 4
Read Board Status/ Update DACs	Read Staus register	Update All DACs	BA + 6
Initialize Sample Counter/ Set Sample Counter Source	Provides software trigger to load sample counter 1	Program Sample Counter Source	BA + 8
Initialize Sample Counter/ Set DAC Output Range	Provides software trigger to load sample counter 2	Program DAC Range	BA + 10
Initialize Sample Counter/ Set DAC Control	Provides software trigger to load sample counter 3	Program DAC Control	BA + 12
Clear / Clear Mask Register	Clears board circuits programmed by a write to this address	Sets the board circuits to be cleared	BA + 14
Update DAC1/ DAC1 FIFO	Update DAC1	Load DAC1 FIFO	BA + 16
Update DAC2/ DAC2 FIFO	Update DAC2	Load DAC2 FIFO	BA + 18
Update DAC3/ DAC3 FIFO	Update DAC3	Load DAC3 FIFO	BA + 20
Update DAC4/ DAC4 FIFO	Update DAC4	Load DAC4 FIFO	BA + 22
8254 SC TC Counter 0 & User TC Counter 0	Read value in SC or User TC Counter 0 (dependent on BA + 4)	Load count in SC or User TC Counter 0 (dependent on BA + 4)	BA + 24
8254 SC TC Counter 1 & User TC Counter 1	Read value in SC or User TC Counter 1 (dependent on BA + 4)	Load count in SC or User TC Counter 1 (dependent on BA + 4)	BA + 25
8254 SC TC Counter 2 & User TC Counter 2	Read value in SC or User TC Counter 2 (dependent on BA + 4)	Load count in SC or User TC Counter 2 (dependent on BA + 4)	BA + 26
8254 Clock TC & User TC Control	Reserved	Program counter mode for SC or User TC (dependent on BA + 4)	BA + 27
Digital I/O Port 0 (Bit Programmable)	Read Port 0 digital input lines	Program Port 0 digital output lines	BA + 28
Digital I/O Port 1 (Port Programmable)	Read Port 1 digital input lines	Program Port 1 digital output lines	BA + 29
Port 0 Clear/ Direction/Mask/Compare	Clear digital IRQ status flag/read Port 0 direction, mask or compare register (dependent on BA + 31)	Clear digital chip/program Port 0 direction, mask or compare register (dependent on BA + 31)	BA + 30
Read Digital I/O Status/ Set Digital Control Register	Read digital status word	Program digital control register & digital interrupt enable	BA + 31

* BA = Base Address

BA + 0: Board ID Register (16-bit operation)

Read: A read returns the board identifier. Each time you read this address the value returned will be "6620".

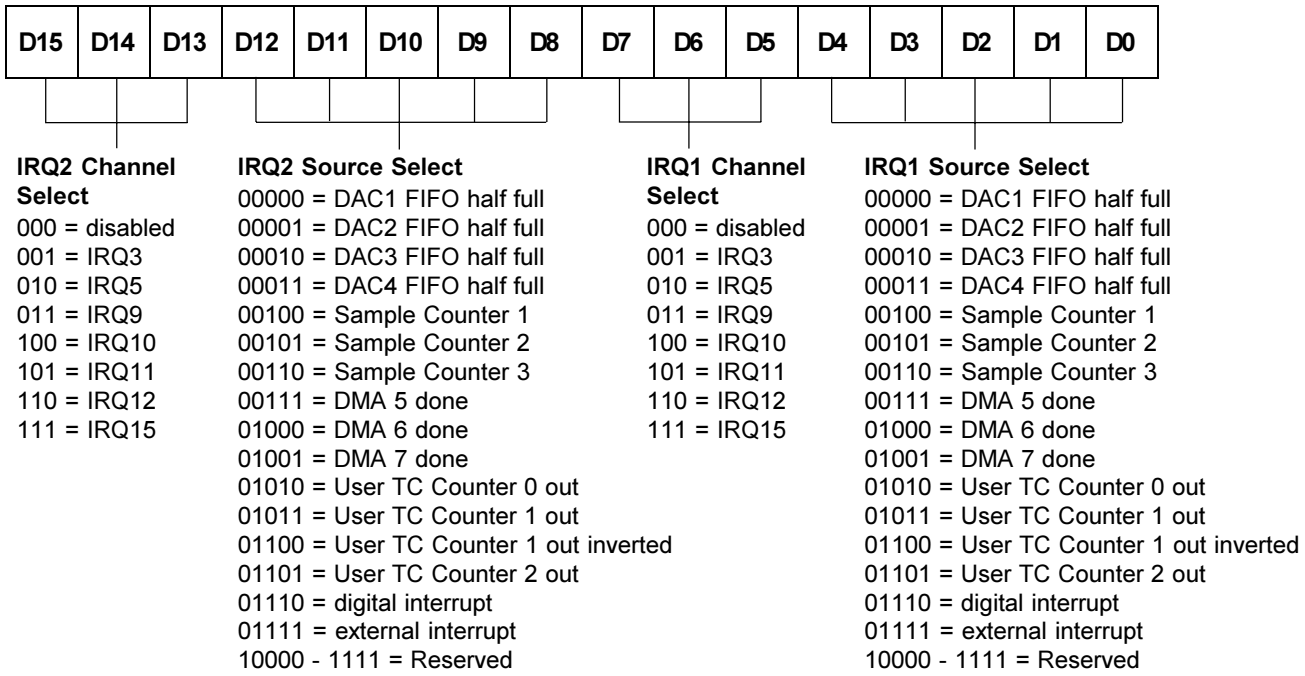
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	0	0	1	1	0	0	0	1	0	0	0	0	0

Write: Reserved.

BA + 2: Program IRQ Source and Channel (16-bit operation)

Read: Reserved.

Write: This register programs the software selectable interrupt source and channel. The IRQ circuitry is driven by an open collector device which is turned off when the IRQ channel is set to disable. The IRQ sources are described below:



IRQ Sources:

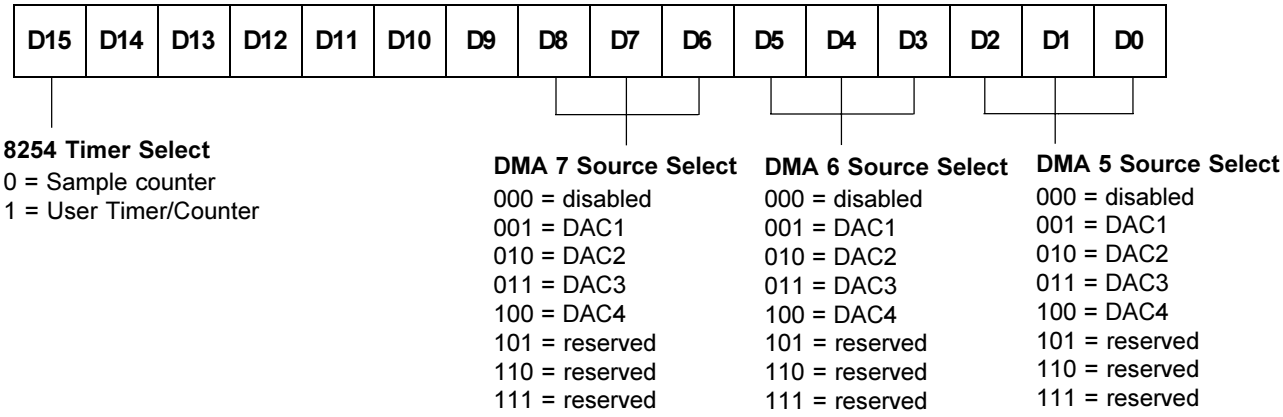
DAC1 FIFO half full - an interrupt is generated when the DAC1 FIFO goes below half full.
DAC2 FIFO half full - an interrupt is generated when the DAC2 FIFO goes below half full.
DAC3 FIFO half full - an interrupt is generated when the DAC3 FIFO goes below half full.
DAC4 FIFO half full - an interrupt is generated when the DAC4 FIFO goes below half full.
Sample Counter 1 - an interrupt is generated when sample counter 1 reaches "0".
Sample Counter 2 - an interrupt is generated when sample counter 2 reaches "0".
Sample Counter 3 - an interrupt is generated when sample counter 3 reaches "0".
DMA 5 done - an interrupt is generated when DMA channel 5 is done.
DMA 6 done - an interrupt is generated when DMA channel 6 is done.
DMA 7 done - an interrupt is generated when DMA channel 7 is done.
User TC Counter 0 out - an interrupt is generated when user TC Counter 0's count reaches 0.
User TC Counter 1 out - an interrupt is generated when user TC Counter 1's count reaches 0.
User TC Counter 1 out inverted - an interrupt is generated when user TC Counter 1's count reaches 0 (useful for frequency counting).

User TC Counter 2 out - an interrupt is generated when user TC Counter 2's count reaches 0.
Digital interrupt - an interrupt is generated when an advanced digital interrupt occurs.
External interrupt - an interrupt is generated when the external interrupt line is pulsed (CN3-19).

BA + 4: Program DMA Channel and Source (16-bit operation)

Read: Reserved.

Write: This register programs the source for each of the DMA channels. It is also used to select which 8254 Timer/Counter chip is being addressed at BA+24 to BA+27. The DMA sources and timer options are described below:



DMA Sources:

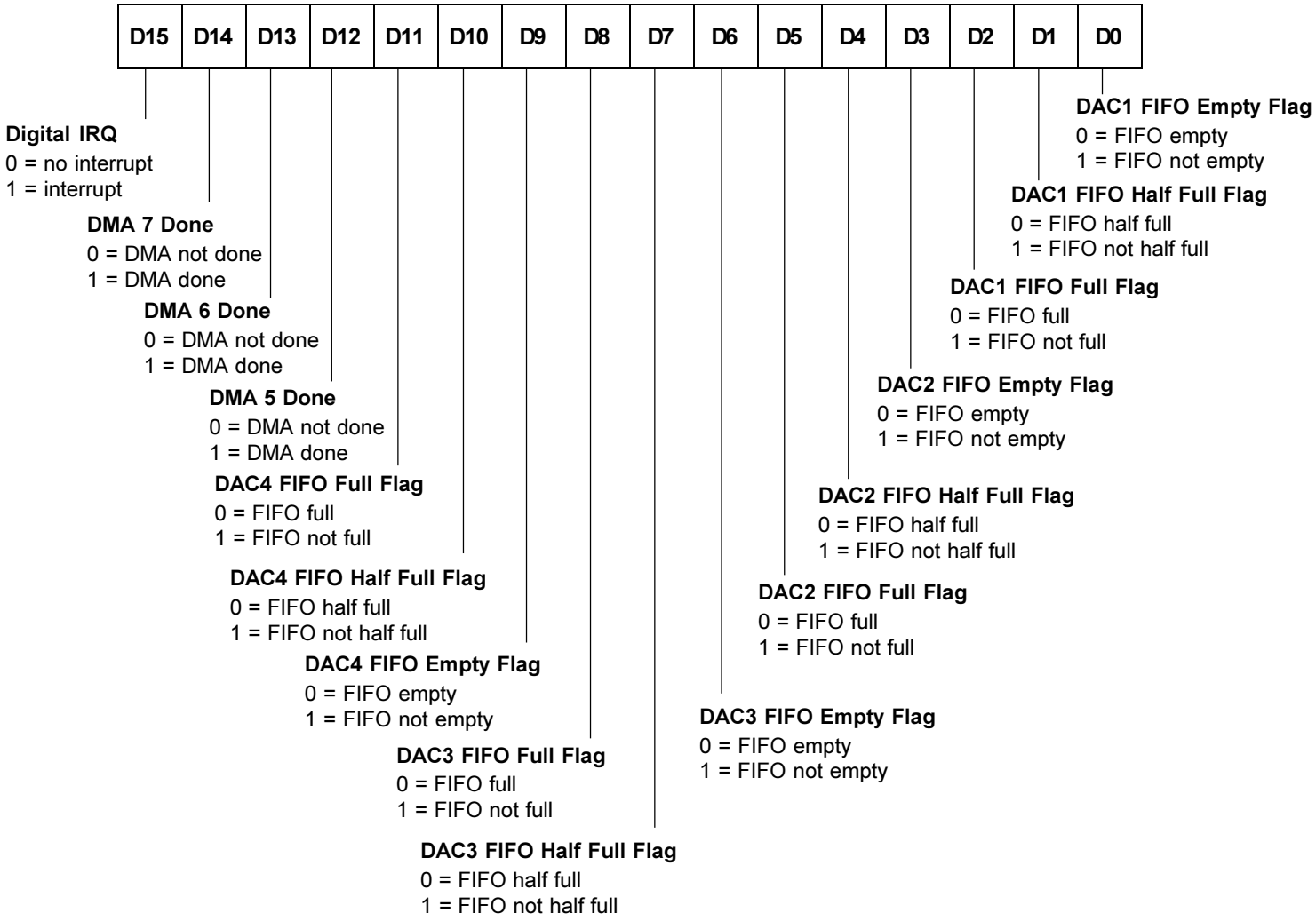
DAC1 - DMA transfers are requested when the DAC1 FIFO is not full.
DAC2 - DMA transfers are requested when the DAC2 FIFO is not full.
DAC3 - DMA transfers are requested when the DAC3 FIFO is not full.
DAC4 - DMA transfers are requested when the DAC4 FIFO is not full.

8254 Timer Select:

Sample counter - Selects the first 8254 chip which contains the 3 sample counters.
User Timer/Counter - Selects the second 8254 which contains the 3 User Timer/Counters.

BA + 6: Read Status / Update All DACs (16-bit operation)

Read: A read returns the status bits defined below.



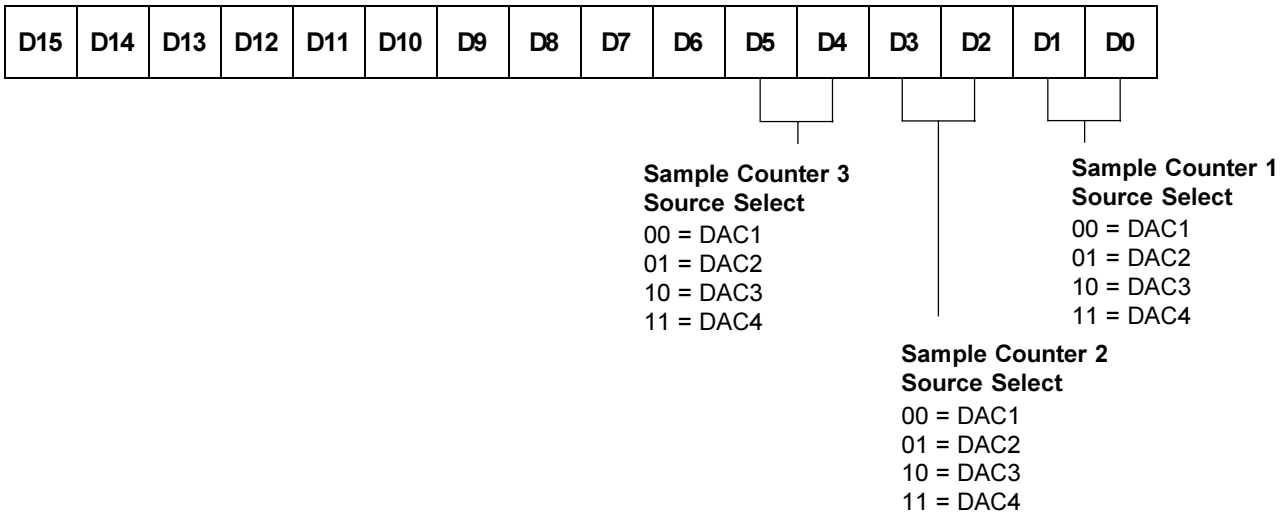
Bit 0 – Goes high when the DAC1 FIFO is not empty.
Bit 1 – Goes low when the DAC1 FIFO is more than half full.
Bit 2 – Goes low when the DAC1 FIFO is full.
Bit 3 – Goes high when the DAC2 FIFO is not empty.
Bit 4 – Goes low when the DAC2 FIFO is more than half full.
Bit 5 – Goes low when the DAC2 FIFO is full.
Bit 6 – Goes high when the DAC3 FIFO is not empty.
Bit 7 – Goes low when the DAC3 FIFO is more than half full.
Bit 8 – Goes low when the DAC3 FIFO is full.
Bit 9 – Goes high when the DAC4 FIFO is not empty.
Bit 10 – Goes low when the DAC4 FIFO is more than half full.
Bit 11 – Goes low when the DAC4 FIFO is full.
Bit 12 – Goes high when a DMA transfer is completed on channel 5.
Bit 13 – Goes high when a DMA transfer is completed on channel 6.
Bit 14 – Goes high when a DMA transfer is completed on channel 7.
Bit 15 – Goes high when an Advanced Digital Interrupt occurs from the digital I/O chip.

Write: A write to this address is used to update all of the D/A outputs simultaneously with a software command. The data written is irrelevant. To use this update method, it must be selected at BA+12.

BA + 8: Load Sample Counter 1 / Select Sample Counter Source (16-bit operation)

Read: A read provides a software trigger so that sample counter 1 can be loaded with the correct value. This software correction is used as an easy means to compensate for the operating structure of the 8254. Two pulses of the counter are required to actually load the desired count and prepare the counter to count down correctly (this can be looked at as the initialization procedure for the sample counter). A pulse is sent to sample counter 1 (Sample Counter, Counter 0) each time you read this address. Without this correction, the initial count sequence will be off by two pulses. Once the counter is properly loaded and starts, any subsequent countdowns of this count will be accurate. Note that sample counter 1 must be programmed for Mode 2 operation.

Write: A write to this address is used to select the input source for the different sample counters. The sample counter sources are described below.



Sample Counter Sources:

DAC1 - The sample counter counts down one count for each sample that is read from the DAC1 FIFO and sent out the D/A.

DAC2 - The sample counter counts down one count for each sample that is read from the DAC2 FIFO and sent out the D/A.

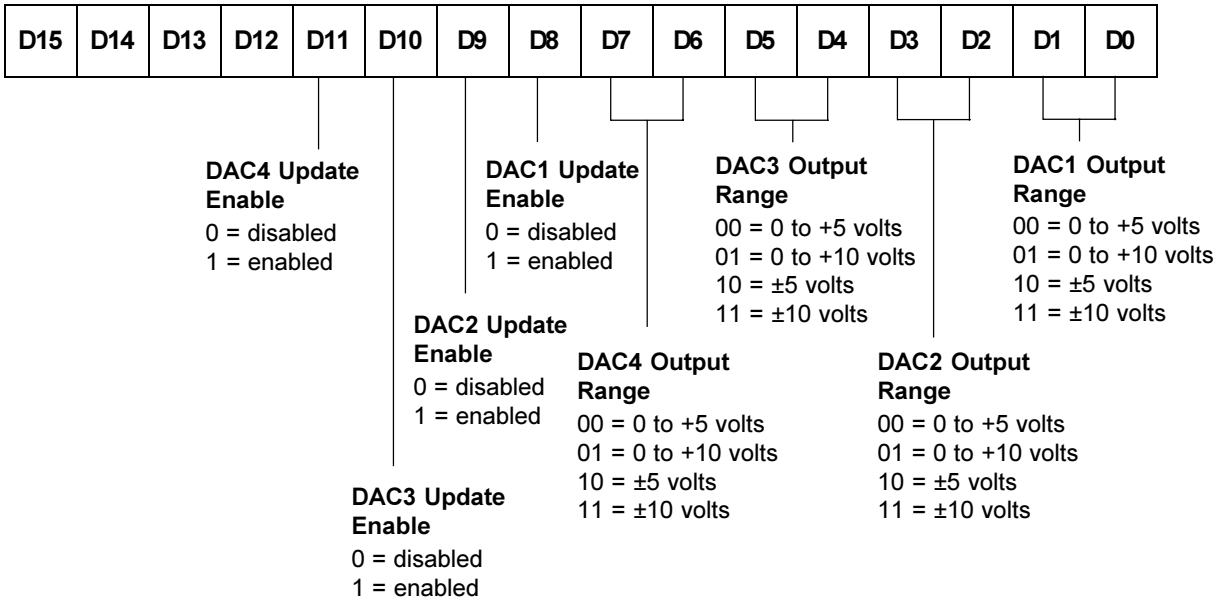
DAC3 - The sample counter counts down one count for each sample that is read from the DAC3 FIFO and sent out the D/A.

DAC4 - The sample counter counts down one count for each sample that is read from the DAC4 FIFO and sent out the D/A.

BA + 10: Load Sample Counter 2 / Select DAC Output Range (16-bit operation)

Read: A read provides a software trigger so that sample counter 2 can be loaded with the correct value. This software correction is used as an easy means to compensate for the operating structure of the 8254. Two pulses of the counter are required to actually load the desired count and prepare the counter to count down correctly (this can be looked at as the initialization procedure for the sample counter). A pulse is sent to sample counter 2 (Sample Counter, Counter 1) each time you read this address. Without this correction, the initial count sequence will be off by two pulses. Once the counter is properly loaded and starts, any subsequent countdowns of this count will be accurate. Note that sample counter 2 must be programmed for Mode 2 operation.

Write: A write to this address is used to select the output range for each of the D/A converters. The different output ranges are described below. At power-up or reset, this register is reset and all of the outputs will go to zero volts.



DAC Output Ranges:

0 to +5 volts - The coding for this output range is straight binary where a code of "0" outputs "0 volts" and a code of all "1's" (4095) outputs "+4.99878 volts".

0 to +10 volts - The coding for this output range is straight binary where a code of "0" outputs "0 volts" and a code of all "1's" (4095) outputs "+9.99756 volts".

±5 volts - The coding for this output range is 2's complement where a code of "0" outputs "0 volts", a code of "0111 1111 1111" (2047) outputs "+4.99756 volts", a code of "1000 0000 0000" (2048) outputs "-5.000000 volts" and a code of all "1's" (4095) outputs "-.00244 volts".

±10 volts - The coding for this output range is 2's complement where a code of "0" outputs "0 volts", a code of "0111 1111 1111" (2047) outputs "+9.99512 volts", a code of "1000 0000 0000" (2048) outputs "-10.000000 volts" and a code of all "1's" (4095) outputs "-.00488 volts".

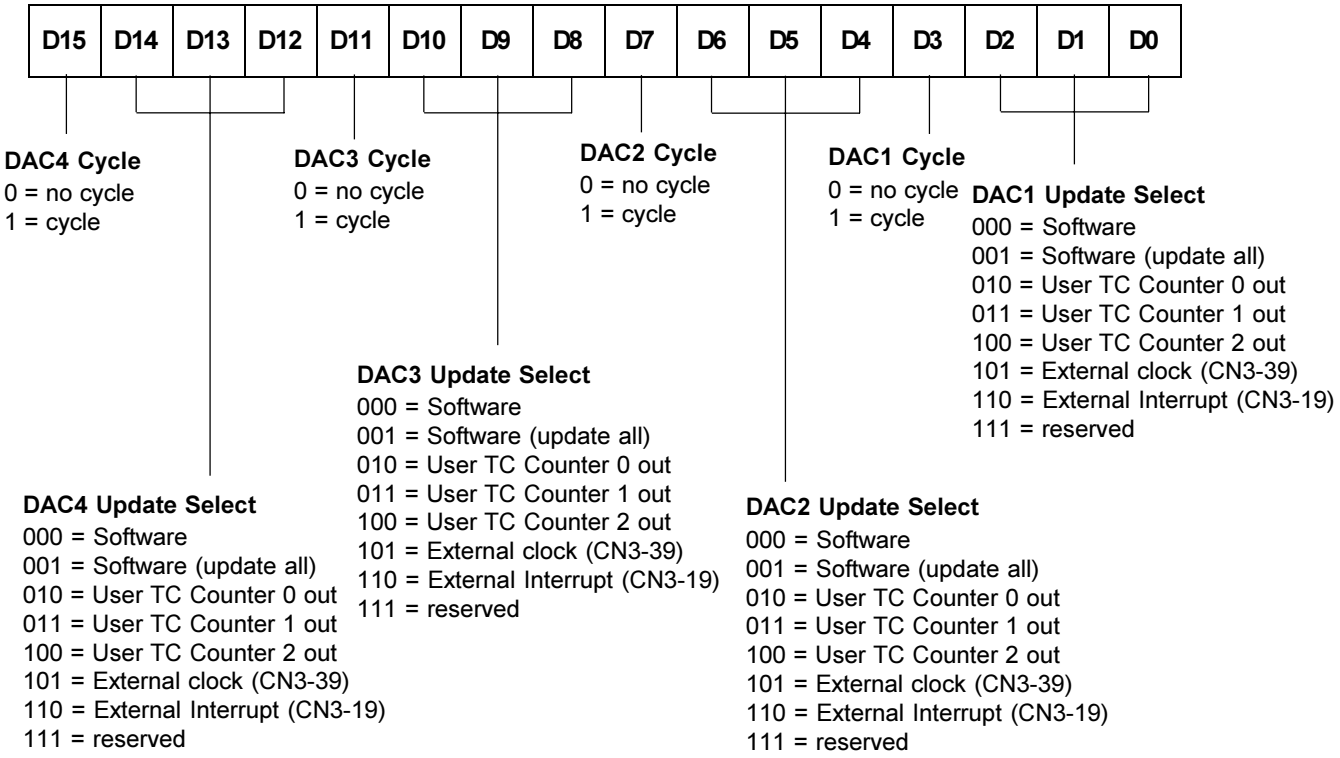
DAC Update:

enable/disable - This bit is used to enable and disable the DAC update signal. This allows loading of the FIFO's and simultaneous enabling to ensure synchronous operation.

BA + 12: Load Sample Counter 3 / Select DAC Update (16-bit operation)

Read: A read provides a software trigger so that sample counter 3 can be loaded with the correct value. This software correction is used as an easy means to compensate for the operating structure of the 8254. Two pulses of the counter are required to actually load the desired count and prepare the counter to count down correctly (this can be looked at as the initialization procedure for the sample counter). A pulse is sent to sample counter 3 (Sample Counter, Counter 2) each time you read this address. Without this correction, the initial count sequence will be off by two pulses. Once the counter is properly loaded and starts, any subsequent countdowns of this count will be accurate. Note that sample counter 3 must be programmed for Mode 2 operation.

Write: A write to this address is used to select the update signal and cycle mode for each D/A output. The different update signals and cycle modes are described below.



DAC Update Signals:

Software - This option uses the individual software update commands for each DAC . (read at BA+16 to BA+22)

Software (update all) - This option uses the software update command to update all the DAC outputs simultaneously. (write at BA+6)

User TC Counter 0 out - This option uses the OUT 0 signal from the User Timer/Counter.

User TC Counter 1 out - This option uses the OUT 1 signal from the User Timer/Counter.

User TC Counter 2 out - This option uses the OUT 2 signal from the User Timer/Counter.

External clock (CN3-39) - This option uses an external signal connected to the board at CN3-39.

External Interrupt (CN3-19) - This option uses an external signal connected to the board at CN3-19.

DAC Cycle:

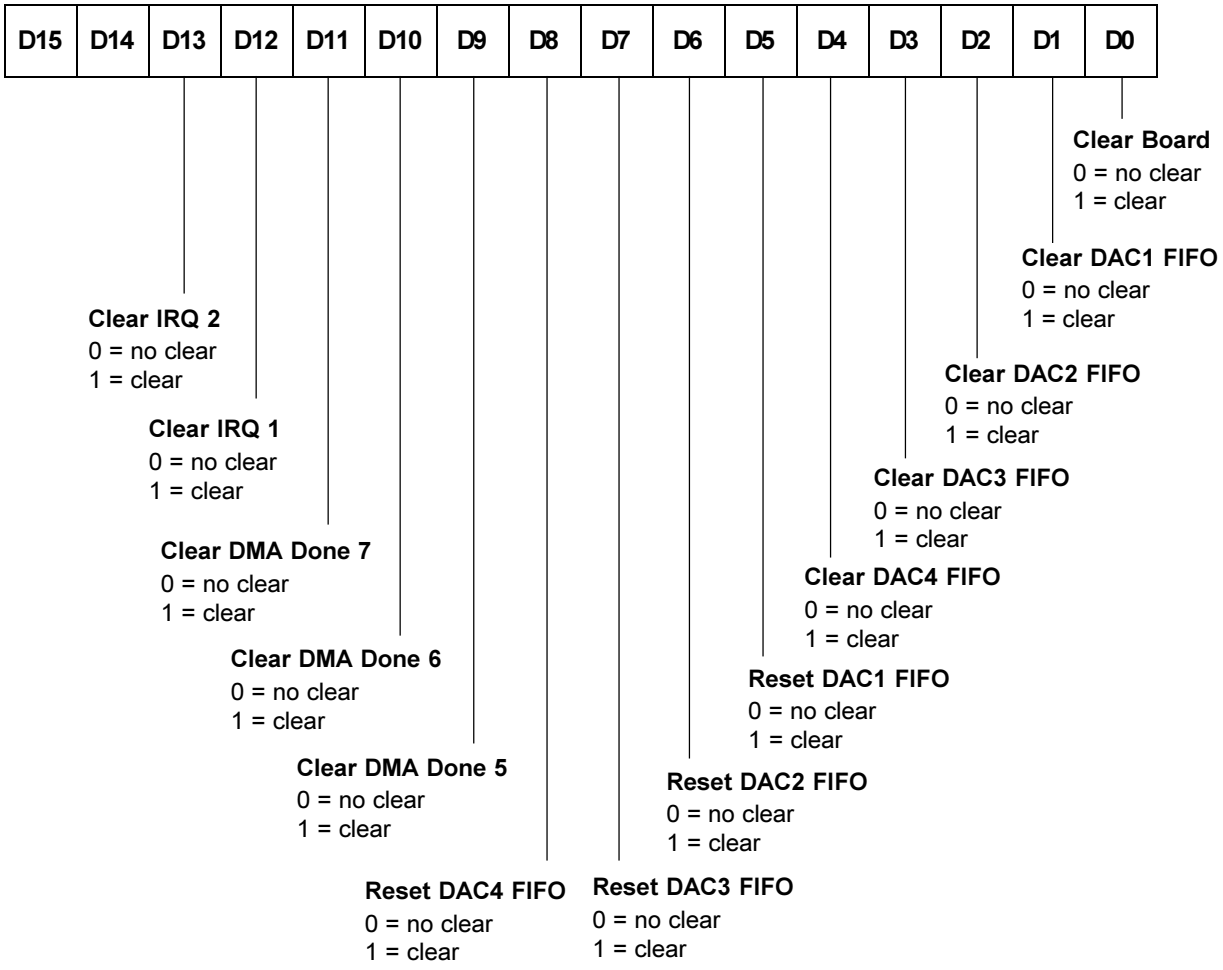
No Cycle - In this mode data is read from the DAC FIFO until it is empty. If the D/A continues to read from the empty FIFO, the data is undetermined.

Cycle - In this mode data is read from the DAC FIFO until it is empty. When the empty flag is encountered, the FIFO automatically resets the internal pointer to the beginning of the data and repeats. This is useful for loading the FIFO with a set of data and repeating it to generate a waveform.

BA + 14: Clear Register (16-bit operation)

Read: A read clears selected circuits on the board, depending on the value programmed at this same address, as described below.

Write: The value programmed in this register determines which clear, enable, and reset operations are carried out when a read at this address is executed. Setting a bit high clears or enables the defined operation. This register's bits are described below:



Clear Options:

Clear Board - Clears all of the board registers and initializes the board for operation.
Clear DAC1 FIFO - Clears all the data from the DAC1 FIFO, setting the empty flag low.
Clear DAC2 FIFO - Clears all the data from the DAC2 FIFO, setting the empty flag low.
Clear DAC3 FIFO - Clears all the data from the DAC3 FIFO, setting the empty flag low.
Clear DAC4 FIFO - Clears all the data from the DAC4 FIFO, setting the empty flag low.
Reset DAC1 FIFO - Resets the internal FIFO pointer to the first data location. (does not erase any data)
Reset DAC2 FIFO - Resets the internal FIFO pointer to the first data location. (does not erase any data)
Reset DAC3 FIFO - Resets the internal FIFO pointer to the first data location. (does not erase any data)
Reset DAC4 FIFO - Resets the internal FIFO pointer to the first data location. (does not erase any data)
Clear DMA Done 5 - Clears the DMA done flag for channel 5.
Clear DMA Done 6 - Clears the DMA done flag for channel 6.
Clear DMA Done 7 - Clears the DMA done flag for channel 7.
Clear IRQ 1 - Clears the interrupt 1 circuitry.
Clear IRQ 2 - Clears the interrupt 2 circuitry.

For example, if you want to clear the DAC1 FIFO and the DAC3 FIFO, you would write a 10 to this address to set bits 1 and 3 high, followed by a read to carry out the clear operation.

Clear FIFO and DMA Done Flag (value written = 10):

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

BA + 16: Update DAC1 Output / Load DAC1 FIFO (16-bit operation)

Read: A read provides a software trigger to update the DAC1 output. This command will update only the DAC1 output as opposed to the "update all" command at BA+6.

Write: A write to this address is used to load data into the DAC1 FIFO. The word sent to this address contains the DAC1 data in the upper 12 bits and the Data Marker in the lowest bit. In unipolar modes, the D/A data is straight binary. In bipolar modes, the D/A data is 2's complement. The bit pattern is described below.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	X	X	X	Data Marker
(MSB)											**(LSB)**				

BA + 18: Update DAC2 Output / Load DAC2 FIFO (16-bit operation)

Read: A read provides a software trigger to update the DAC2 output. This command will update only the DAC2 output as opposed to the "update all" command at BA+6.

Write: A write to this address is used to load data into the DAC2 FIFO. The word sent to this address contains the DAC2 data in the upper 12 bits and the Data Marker in the lowest bit. In uni-polar modes, the D/A data is straight binary. In bi-polar modes, the D/A data is 2's complement. The bit pattern is described below.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	X	X	X	Data Marker
(MSB)											**(LSB)**				

BA + 20: Update DAC3 Output / Load DAC3 FIFO (16-bit operation)

Read: A read provides a software trigger to update the DAC3 output. This command will update only the DAC3 output as opposed to the "update all" command at BA+6.

Write: A write to this address is used to load data into the DAC3 FIFO. The word sent to this address contains the DAC3 data in the upper 12 bits and the Data Marker in the lowest bit. In unipolar modes, the D/A data is straight binary. In bipolar modes, the D/A data is 2's complement. The bit pattern is described below.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	X	X	X	Data Marker
(MSB)											(LSB)				

BA + 22: Update DAC4 Output / Load DAC4 FIFO (16-bit operation)

Read: A read provides a software trigger to update the DAC4 output. This command will update only the DAC4 output as opposed to the "update all" command at BA+6.

Write: A write to this address is used to load data into the DAC4 FIFO. The word sent to this address contains the DAC4 data in the upper 12 bits and the Data Marker in the lowest bit. In unipolar modes, the D/A data is straight binary. In bipolar modes, the D/A data is 2's complement. The bit pattern is described below.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	X	X	X	Data Marker
(MSB)											(LSB)				

BA + 24: TC Counter 0 (8-bit Operation)

Read/Write. A write loads the first counter in one of the two timer/counters on the board with a new 16-bit value in two 8-bit steps, LSB followed by MSB. The counter must be loaded in two 8-bit steps! Counting begins as soon as the count is loaded. The timer/counter being loaded is selected by writing to BA + 4, bit 15. A read shows the count in the counter.

BA + 25: TC Counter 1 (8-bit Operation)

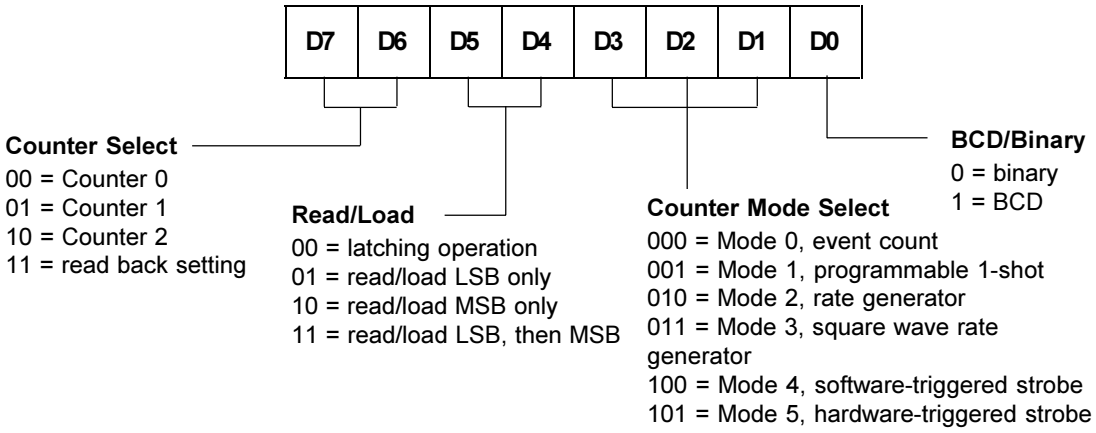
Read/Write. A write loads the second counter in one of the two timer/counters on the board with a new 16-bit value in two 8-bit steps, LSB followed by MSB. The counter must be loaded in two 8-bit steps! Counting begins as soon as the count is loaded. The timer/counter being loaded is selected by writing to BA + 4, bit 15. A read shows the count in the counter.

BA + 26: TC Counter 2 (8-bit Operation)

Read/Write. A write loads the third counter in one of the two timer/counters on the board with a new 16-bit value in two 8-bit steps, LSB followed by MSB. The counter must be loaded in two 8-bit steps! Counting begins as soon as the count is loaded. The timer/counter being loaded is selected by writing to BA + 4, bit 15. A read shows the count in the counter.

BA + 27: Timer/Counter Control Word (8-bit Operation)

Write only. Accesses the selected timer/counter's control register to directly control the three 16-bit counters, 0, 1, and 2.



BA + 28: Digital I/O Port 0, Bit Programmable Port (8-bit Operation)

Read/Write.

Port 0:

D7	D6	D5	D4	D3	D2	D1	D0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

This port transfers the 8-bit Port 0 bit programmable digital input/output data between the module and external devices. The bits are individually programmed as input or output by writing to the Direction Register at BA + 28. For all bits set as inputs, a read reads the input values and a write is ignored. For all bits set as outputs, a read reads the last value sent out on the line and a write writes the current loaded value out to the line.

Note that when any reset of the digital circuitry is performed (clear chip or computer reset), all digital lines are reset to inputs and their corresponding output registers are cleared.

BA + 29: Digital I/O Port 1, Byte Programmable Port (8-bit Operation)

Read/Write.

Port 1:

D7	D6	D5	D4	D3	D2	D1	D0
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.1

This port transfers the 8-bit Port 1 digital input or digital output byte between the module and an external device. When Port 1 is set as inputs, a read reads the input values and a write is ignored. When Port 1 is set as outputs, a read reads the last value sent out of the port and a write writes the current loaded value out of the port.

Note that when any reset of the digital circuitry is performed (clear chip or computer reset), all digital lines are reset to inputs and their corresponding output registers are cleared.

BA + 30: Read/Program Port 0 Direction/Mask/Compare Registers (8-bit Operation)

Read/Write. A read clears the IRQ status flag or provides the contents of one of digital I/O Port 0's three control registers; and a write clears the digital chip or programs one of the three control registers, depending on the setting of bits 0 and 1 at BA + 30. When bits 1 and 0 at BA + 30 are 00, the read/write operations clear the digital IRQ status flag (read) and the digital chip (write). When these bits are set to any other value, one of the three Port 0 registers is addressed.

Direction Register (BA + 31, bits 1 and 0 = 01):

For all bits:
0 = input
1 = output

D7	D6	D5	D4	D3	D2	D1	D0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

This register programs the direction, input or output, of each bit at Port 0.

Mask Register (BA + 31, bits 1 and 0 = 10):

For all bits:
0 = bit enabled
1 = bit masked

D7	D6	D5	D4	D3	D2	D1	D0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

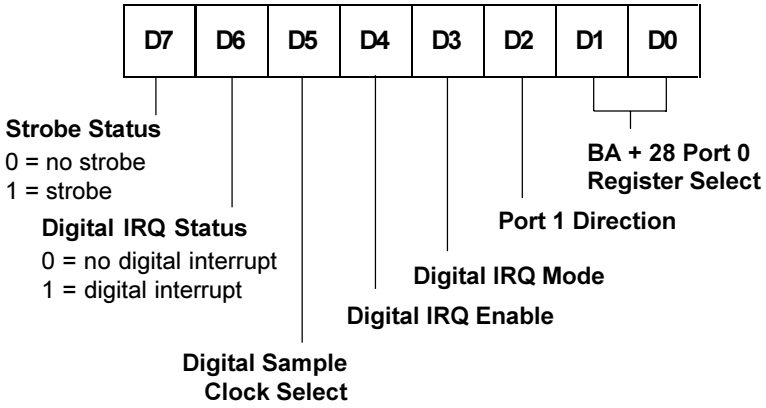
In the Advanced Digital Interrupt modes, this register is used to mask out specific bits when monitoring the bit pattern present at Port 0 for interrupt generation. In normal operation where the Advanced Digital Interrupt feature is not being used, any bit which is masked by writing a 1 to that bit will not change state, regardless of the digital data written to Port 0. For example, if you set the state of bit 0 low and then mask this bit, the state will remain low, regardless of what you output at Port 0 (an output of 1 will not change the bit's state until the bit is un-masked).

Compare Register (BA + 31, bits 1 and 0 = 11):
This register is used for the Advanced Digital Interrupt modes. In the match mode where an interrupt is generated when the Port 0 bits match a loaded value, this register is used to load the bit pattern to be matched at Port 0. Bits can be selectively masked so that they are ignored when making a match. NOTE: Make sure that bit 3 at BA + 30 is set to 1, selecting match mode, BEFORE writing the Compare Register value at this address. In the event mode where an interrupt is generated when any Port 0 bit changes its current state, the value which caused the interrupt is latched at this register and can be read from it. Bits can be selectively masked using the Mask Register so a change of state is ignored on these lines in the event mode.

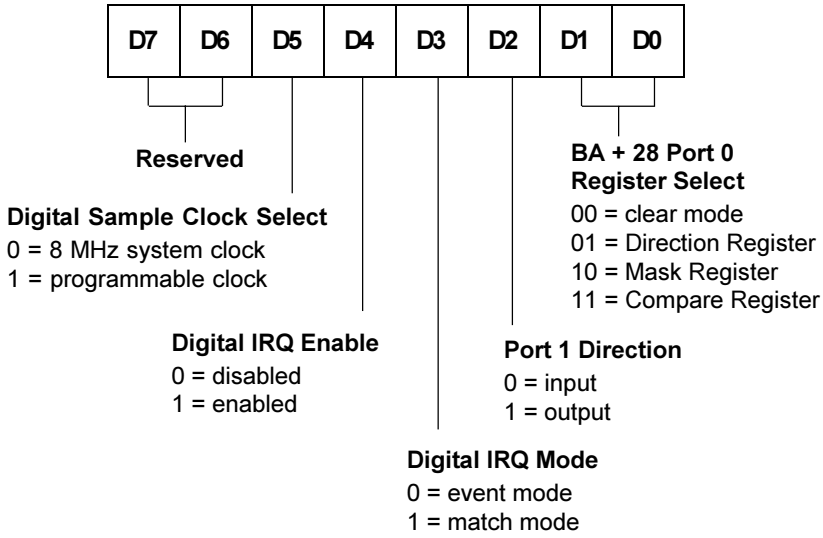
BA + 31: Read Digital IRQ Status/Program Digital Mode (8-bit Operation)

Read/Write.

Digital IRQ/Strobe Status:
A read shows you whether a digital interrupt has occurred (bit 6), whether a strobe has occurred (bit 7, when using the strobe input as described in Chapter 7), and lets you review the states of bits 0 through 5 in this register. If bit 6 is high, then a digital interrupt has taken place. If bit 7 is high, a strobe has been issued.



Digital Mode Register:



Bits 0 and 1 – Select the clear mode initiated by a read/write operation at BA + 28 or the Port 0 control register you talk to at BA + 28 (Direction, Mask, or Compare Register).

Bit 2 – Sets the direction of the Port 1 digital lines.

Bit 3 – Selects the digital interrupt mode: event (any Port 0 bit changes state) or match (Port 0 lines match the value programmed into the Compare Register at BA + 28).

Bit 4 – Disables/enables digital interrupts.

Bit 5 – Sets the clock rate at which the digital lines are sampled when in a digital interrupt mode. Available clock sources are the 8 MHz system clock and the output of User TC Counter 1 (16-bit programmable clock). When a digital input line changes state, it must stay at the new state for two edges of the clock pulse (62.5 nanoseconds when using the 8 MHz clock) before it is recognized and before an interrupt can be generated. This feature eliminates noise glitches that can cause a false state change on an input line and generate an unwanted interrupt. This feature is detailed in Chapter 7.

Bit 6 – Read only (digital IRQ status).

Bit 7 – Reserved.

Programming the DM6620

This section gives you some general information about programming and the DM6620 board.

The DM6620 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. Because the DM6620 is AT bus compatible, most operations are done in a 16-bit word format. The 8254 timer/counters must be programmed in 8-bit operations. High-level languages such as Pascal, C, and C++ make it very easy to read/write these ports. The table below shows you how to read from and write to I/O ports in Turbo C and Turbo Pascal.

Language	Read 8 Bits	Write 8 Bits	Read 16 Bits	Write16 Bits
Turbo C	Data=inportb(Address)	outportb(Address,Data)	Data=inport(Address)	outport(Address,Data)
Turbo Pascal	Data:=Port[Address]	Port[Address]:=Data	Data:=PortW[Address]	PortW[Address]:=Data

In addition to being able to read/write the I/O ports on the DM6620, 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 and C.

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

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 the correct function for 8- and 16-bit operations with the 6620!**

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.

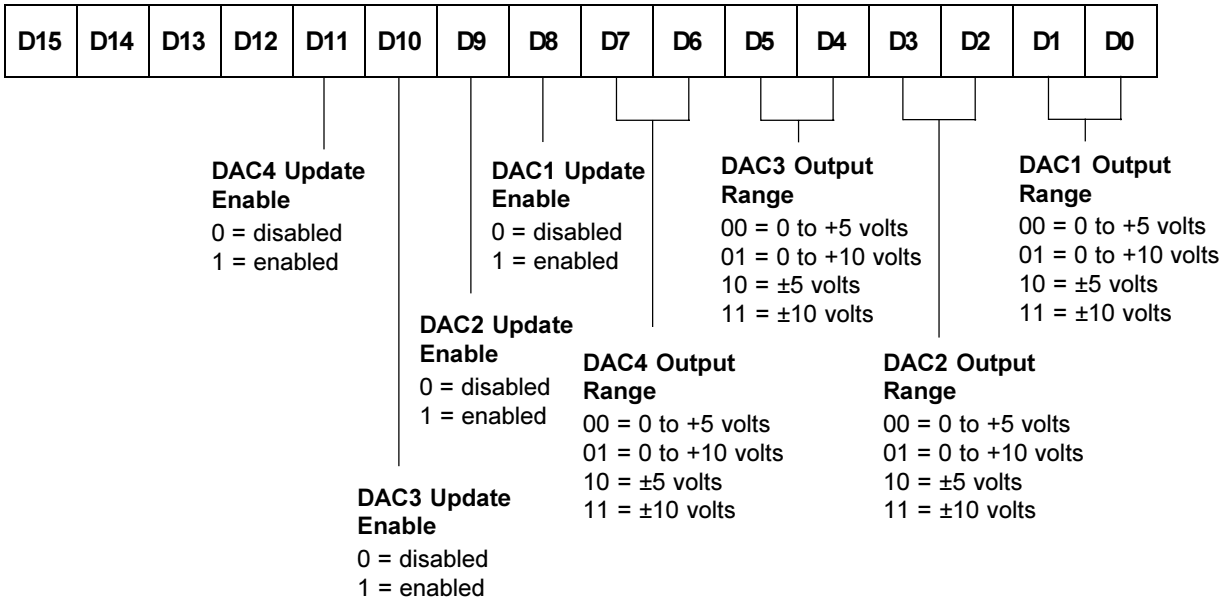
CHAPTER 5

D/A CONVERSIONS

This chapter explains how to perform D/A conversions on the DM6620.

The analog outputs are generated by four 12-bit D/A converters with independent software programmable output ranges of ±5, ±10, 0 to +5, or 0 to +10 volts. All channels support DMA transfer. DAC1 data is written to BA + 16, DAC2 data is written to BA + 18, DAC3 data is written to BA + 20 and DAC4 data is written to BA + 22. The DAC Output Range register at BA + 10 and the DAC Update register at BA + 12 are described below.

DAC Output Range Register (BA + 10):



DAC Output Ranges:

0 to +5 volts - The coding for this output range is straight binary where a code of "0" outputs "0 volts" and a code of all "1's" (4095) outputs "+4.99878 volts".

0 to +10 volts - The coding for this output range is straight binary where a code of "0" outputs "0 volts" and a code of all "1's" (4095) outputs "+9.99756 volts".

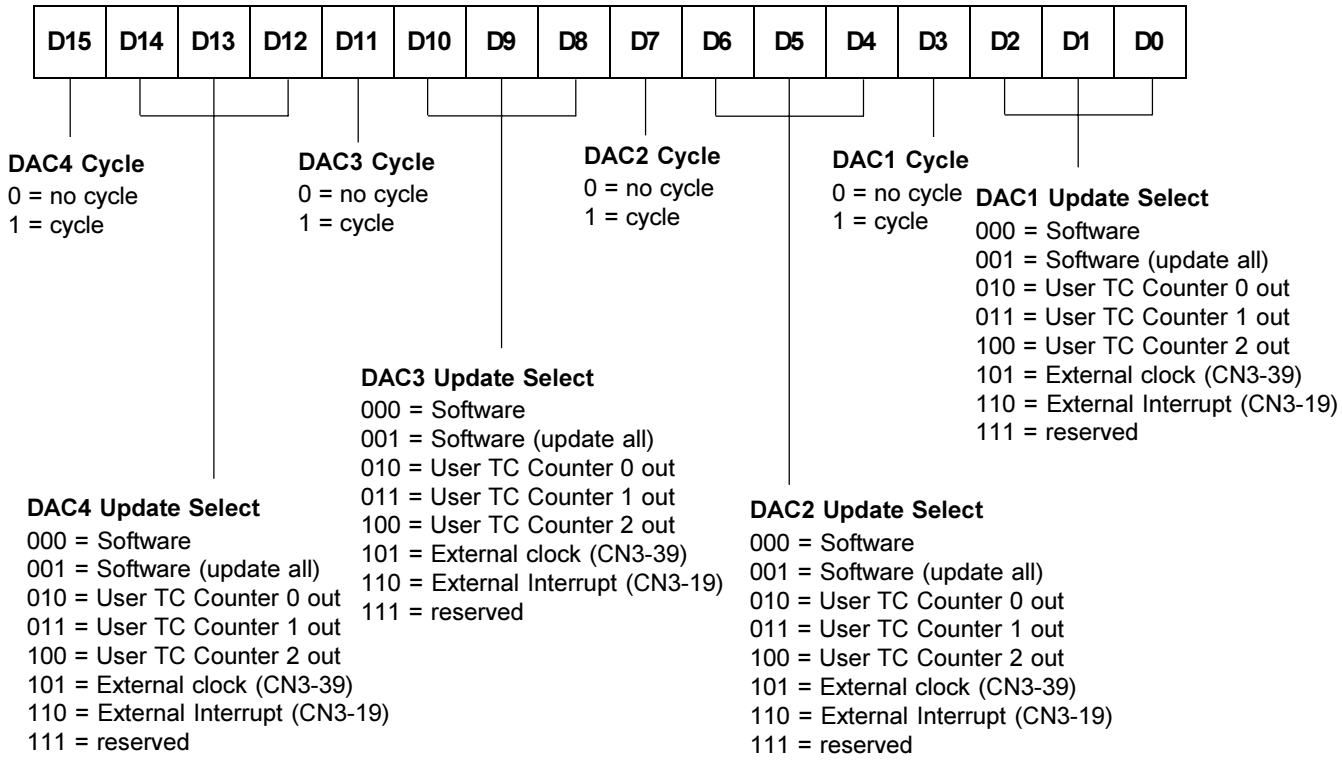
±5 volts - The coding for this output range is 2's complement where a code of "0" outputs "0 volts", a code of "0111 1111 1111" (2047) outputs "+4.99756 volts", a code of "1000 0000 0000" (2048) outputs "-5.000000 volts" and a code of all "1's" (4095) outputs "-.00244 volts".

±10 volts - The coding for this output range is 2's complement where a code of "0" outputs "0 volts", a code of "0111 1111 1111" (2047) outputs "+9.99512 volts", a code of "1000 0000 0000" (2048) outputs "-10.000000 volts" and a code of all "1's" (4095) outputs "-.00488 volts".

DAC Update:

enable/disable - This bit is used to enable and disable the DAC update signal. This allows loading of the FIFO's and simultaneous enabling to ensure synchronous operation.

DAC Update Register (BA + 12):



DAC Update Signals:

Software - This option uses the individual software update commands for each DAC . (read at BA+16 to BA+22)

Software (update all) - This option uses the software update command to update all the DAC outputs simultaneously. (write at BA+6)

User TC Counter 0 out - This option uses the OUT 0 signal from the User Timer/Counter.

User TC Counter 1 out - This option uses the OUT 1 signal from the User Timer/Counter.

User TC Counter 2 out - This option uses the OUT 2 signal from the User Timer/Counter.

External clock (CN3-39) - This option uses an external signal connected to the board at CN3-39.

External Interrupt (CN3-19) - This option uses an external signal connected to the board at CN3-19.

DAC Cycle:

No Cycle - In this mode data is read from the DAC FIFO until it is empty. If the D/A continues to read from the empty FIFO, the data is undetermined.

Cycle - In this mode data is read from the DAC FIFO until it is empty. When the empty flag is encountered, the FIFO automatically resets the internal pointer to the beginning of the data and repeats. This is useful for loading the FIFO with a set of data and repeating it to generate a waveform.

A write to the proper Base Address programs the DAC 12-bit output in the format shown below. Output coding is two's complement for bipolar and straight binary for unipolar ranges. After the data has been written to the DAC channel, an update signal must occur to output the data to the D/A. If the update selection is software, the program must now issue the update command. If one of the hardware clocks is being used, the program does not need to issue the update command.

A write also sets the DAC Data Marker outputs.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	X	X	X	Data Marker
(MSB)											**(LSB)**				

The following tables list the key digital codes and corresponding output voltages for the D/A converters.

Unipolar D/A Bit Weight	Ideal Output Voltage (millivolts)	
	0 to +5 Volts	0 to +10 Volts
4095	+4998.78	+9997.56
2048	+2500.00	+5000.00
1024	+1250.00	+2500.00
512	+625.00	+1250.00
256	+312.50	+625.00
128	+156.25	+312.50
64	+78.13	+156.25
32	+39.06	+78.13
16	+19.53	+39.06
8	+9.77	+19.53
4	+4.88	+9.77
2	+2.44	+4.88
1	+1.22	+2.44
0	0.00	0.00

Bipolar D/A Bit Weight	Ideal Output Voltage (millivolts)	
	-5 to +5 Volts	-10 to +10 Volts
2047	+4997.56	+9995.12
1024	+2500.00	+5000.00
512	+1250.00	+2500.00
256	+625.00	+1250.00
128	+312.50	+625.00
64	+156.25	+312.50
32	+78.13	+156.25
16	+39.06	+78.13
8	+19.53	+39.06
4	+9.77	+19.53
2	+4.88	+9.77
1	+2.44	+4.88
0	0.00	0.00
-1	-2.44	-4.88
-2	-4.88	-9.77
-4	-9.77	-19.53
-8	-19.53	-39.06
-16	-39.06	-78.13
-32	-78.13	-156.25
-64	-156.25	-312.50
-128	-312.50	-625.00
-256	-625.00	-1250.00
-512	-1250.00	-2500.00
-1024	-2500.00	-5000.00
-2048	-5000.00	-10000.00

1024 Sample Buffer

Each DAC channel has a 1024 sample buffer for storing data to be sent to the D/A converter. This means that you can fill the buffer with data and set up the D/A to output this data automatically. This is very useful for outputting high speed data or generating waveforms with precise timing requirements. By setting the cycle bit at BA + 12 , you can fill the buffer with one cycle of a wave, start the D/A update clock and the buffer will continue to repeat until the clock is stopped. Combining this feature with the variety of update sources, you can build a flexible waveform generator.

If you are trying to generate a non-repetitive waveform, you can combine the sample buffer capability with the DAC sample counter. To utilize this feature of the DM6620 properly, you should load the buffer with data, program the DAC sample counter for half the buffer size (512 samples) and use the sample counter to generate an interrupt. When an interrupt is received, you should reload the buffer with 512 new samples. By continuing this cycle, you can generate a non-repetitive waveform at high speeds.

Status of the FIFO buffers can be monitored at BA + 6. Any samples that are written to the FIFO after it is full will be ignored. You can write up to 1024 samples to the buffer before it is full. Each update pulse (either software or from one of the clocks) will remove a sample from the buffer and send it out the D/A. Each read after the FIFO buffer is empty will send "undetermined" data out the D/A.

At power-up or reset, the D/A outputs are set to 0 volts. Before loading data into the sample buffer it is best to clear the buffer by writing and reading the proper bits at BA + 14. When you issue the "Clear DAC FIFO" command, all data in the buffer is erased. If you issue the "Reset DAC FIFO" command, the data in the buffer is not erased, however the address pointer is set back to the beginning of the buffer: This is useful when you are generating waveforms and stop the updating in the middle of a cycle.

DAC Update Enables

These enable bits, located in the register at BA + 10, are used to enable and disable updating of the D/A outputs. When these bits are low (disabled) the update signal will have no effect on the output. This is useful when connecting external signals or using clocks to update the D/A outputs. Any clock pulses received by the board while the update is disabled will be ignored. Updating will occur on the first clock pulse after the enable bit is set high. **NOTE: You must enable the update bit even when the update signal is software.**

DAC Cycle Bit

The cycle bit is used to make the buffer data repeat. Under normal operation, without the cycle bit set, data is written into the buffer and the update clock reads data out of the buffer. When the buffer is empty, the data will be undefined as explained above. If you set the cycle bit high, the data in the buffer will repeat. If you load a data set into the buffer, when the update clock reaches the end of the data it will automatically wrap around to the beginning and start over. This is useful for generating waveforms. It is important to note that when you are operating in this mode, you do not have to fill the complete FIFO buffer before using it. You can load the FIFO with any number of samples between 1 and 1024. The board will automatically repeat only the number of samples that are stored in the buffer.

DMA Transfer

Each DAC buffer can be accessed through DMA. This allows you to load D/A data into a memory buffer and have the DMA controller in the CPU fill the FIFO buffer automatically. This feature is useful if the amount of D/A data is greater than the 1024 samples that the FIFO buffer will hold.

A detailed discussion about DMA transfers can be found in chapter 6. Two important things to remember when using DMA with the DAC are to program the DMA controller in the CPU for read operations, since you are reading data from memory and sending it out the D/A, and be sure to issue a "Clear DAC FIFO" command at BA + 14 right before you unmask the DMA channel bit. This will ensure proper DMA operation.

DAC Sample Counter

The DAC sample counters are useful when using clocks to output data to the D/A. These 16-bit counters will

count update pulses sent to the D/A's and can be polled to read the current count or can be used to generate interrupts when the count reaches 0. These counters can be loaded to any starting value up to 65535 and count down. When the count reaches 0 it will automatically be reloaded with the original starting value. Using the sample counters to cause an interrupt is useful when generating non-repeating waveforms. The sample counter interrupt can be used to signal the CPU when to load new data into the FIFO buffer.

A read at BA + 8, 10 or 12 provides a software trigger so that the appropriate DAC sample counter can be loaded with the correct value. This software correction is used as an easy means to compensate for the operating structure of the 8254. Two pulses of the counter are required to actually load the desired count and prepare the counter to count down correctly (this can be looked at as the initialization procedure for the DAC sample counter). A pulse is sent to the DAC sample counter each time you read this address. Without this correction, the initial count sequence will be off by two pulses. Once the counter is properly loaded and starts, any subsequent countdowns of this count will be accurate. You must program the register at BA + 8 to select the input source for each of the 3 sample counters. Note that the DAC sample counter 8254 must be programmed for Mode 2 operation.

DAC Data Markers

The DAC Data Markers are used to send out digital pulses synchronized to the D/A output. Since each DAC FIFO buffer is 16 bits wide and the D/A only uses 12 bits, there are 4 bits left over. One of these bits is used for the Data Marker. This bit location can be filled with data and this data is sent out on the appropriate pins synchronized to the D/A output. This is useful for sending out a trigger pulse each time a waveform crosses zero or to send out pulses to trigger A/D conversions at the proper time in the D/A waveform. Each DAC channel has 1 Data Marker bit. These outputs can be accessed at the pins on connector CN3.

CHAPTER 6

DATA TRANSFERS USING DMA

This chapter explains how data transfers are accomplished 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 memory and write it to a peripheral device.

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 8237 DMA controller.
6. Fill DMA buffer with desired data.
7. Program device accepting data (DM6620).
8. Enable DMA channel.
9. Wait until DMA is complete.
10. Disable DMA channel.

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 DM6620 can use DMA channel 5, 6, or 7, selected through software. You can arbitrarily choose any of these; in most cases your choice will be 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 system 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 8237 DMA controller will read the data from to send out to the D/A board. This buffer can be either static or dynamically allocated. The buffer must start on a word boundary (i.e., even numbered address). You should force your compiler to use word alignment for data. 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);
```

• Calculating the Page and Offset of a Buffer

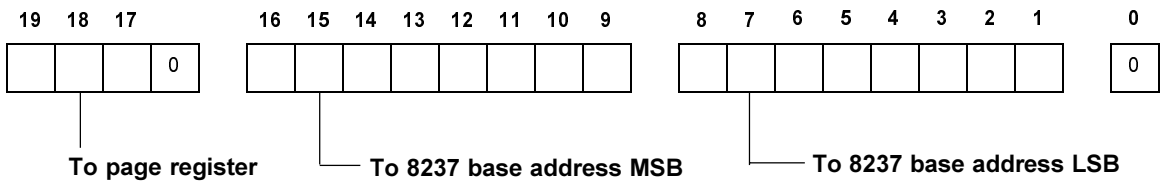
Once you have a buffer into which to place your data, you must inform the 8237 DMA controller of the location of this buffer. This is a little more complex than it sounds because the DMA controller uses a **page**:offset memory scheme, while you are probably used to thinking about your computer's memory 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 read data at a specified offset into a specified page (for example, start reading at word 512 of page 3). Each time a word of data is read by the controller, the offset is automatically incremented so the next word will be read from 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)

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 as a long integer, or DWORD, or the results will be meaningless.) The linear address is a 20-bit value, with the upper 4 bits representing the page and the lower 16 bits representing the offset into the page. Even though the upper 4 bits are the page, only the upper 3 bits, D17, D18, and D19, are sent to what is called the page register. The remaining bit for the page, D16, is sent to the base address register of the DMA controller along with bits D1 through D15. Since the buffer sits on a word boundary, bit D0 must be zero, and is ignored. The following diagram shows you to which registers the components of the 20-bit linear address are sent.



The following examples show you how to calculate the linear address and break it into components to be sent to the various registers.

In Pascal:

```
Segment := SEG(Buffer);                          { get segment of buffer }
Offset := OFS(Buffer);                           { get offset of buffer }
LinearAddress := Segment * 16 + Offset;          { calculate linear address }
PageBits := (LinearAddress DIV 65536) AND $0E;   { determine page corresponding
                                                   to this,linear address and
                                                   clear least significant bit }
OffsetBits := (LinearAddress SHR 2) MOD 65536;   { shift linear address to ignore
                                                   D0 then extract bits D1-D16 }
```

In C:

```
segment = FP_SEG(&Buffer);                       /* get segment of buffer */
offset = FP_OFS(&Buffer);                        /* get offset of buffer */
linear_address = segment * 16 + offset;          /* calculate linear address */
pagebits = (linear_address / 65536) & 0x0E;      /* determine page corresponding
                                                    to this linear address and
                                                     clear least significant
                                                    bit */
offset_bits = (linear_address >> 2) % 65536;     /* shift linear address to
                                                     ignore D0 then extract bits
                                                     D1-D16 */
```

Beware! There is one big catch when using page-based addresses. The 8237 DMA controller cannot read properly from 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 read from such a buffer because the DMA controller can only read from one page without reprogramming. When it reaches the end of the current page, it does not start reading from the next page. Instead, it starts reading 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 reading data from it will almost always causes erratic behavior.

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 read from 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, with the least significant bit set to zero. The DMA page register is separate from the DMA controller, as shown in the table below.

DMA Channel	Location of Page Register
5	8B/(139)
6	89/(137)
7	8A/(138)

• The DMA Controller

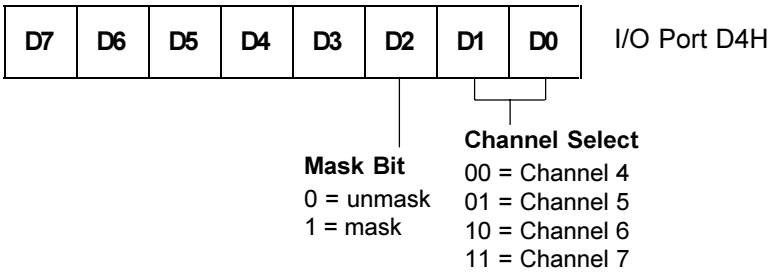
The DMA controller is made up of two complex 8237 chips, one for DMA channels 0-3, and one for channels 4-7, that occupy 32 contiguous bytes of the AT I/O port space starting with port C0H. A complete discussion of how it operates is beyond the scope of this manual; only relevant information is included here. The DMA controller is programmed by writing to the DMA registers in your AT. The table below lists these registers.

DMA Registers	
Address hex/(decimal)	**Location of Page Register**
8B/(139)	Channel 5 DMA Page Select
C4/(196)	Channel 5 DMA Base Address
C6/(198)	Channel 5 DMA Count
89/(137)	Channel 6 DMA Page Select
C8/(200)	Channel 6 DMA Base Address
CA/(202)	Channel 6 DMA Count
8A/(138)	Channel 7 DMA Page Select
CC/(204)	Channel 7 DMA Base Address
CE/(206)	Channel 7 DMA Count
D4/(212)	Mask Register
D6/(214)	Mode Register
D8/(216)	Byte Pointer Flip-Flop

If you are using DMA channel 5, write your page offset bits to port C4H and the count to C6H; for channel 6, write the offset to C8H and the count to CAH; for channel 7, write the offset to CCH and the count to CEH. The page offset bits are the bits you calculated as shown above. Count indicates the number of samples that you want the DMA controller to transfer. The value that you write to the DMA controller is (number of samples - 1). The mask register and mode register are described below.

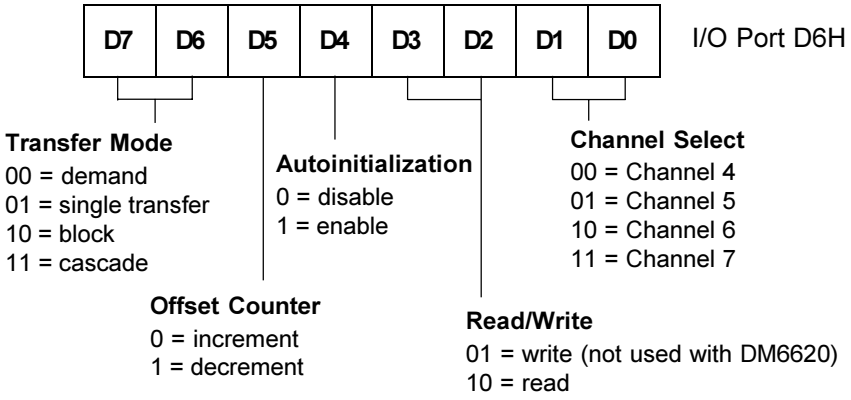
• DMA Mask Register

The DMA 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 DM6620 has been programmed to accept 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.



• 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 write mode cannot be used with the DM6620. Autoinitialization allows the DMA controller to automatically start over once it has transferred the requested number of words. 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. Block mode transfer is not supported by this board.



• Programming the DMA Controller

To program the DMA controller, follow these steps:

1. Disable DMA on the channel you are using.
2. Write the DMA mode register to choose the DMA parameters.
3. Write the page offset bits (D1-D16) of your buffer.
4. Write the number of samples to transfer.
5. Write the page register.
6. Enable DMA on the channel you are using.

• Programming the DM6620 for DMA

Once you have set up the DMA controller, you must program the DM6620 for DMA. The following steps list this procedure:

1. Program Conversion and Trigger mode.
2. Program the DMA channel at BA + 2.
3. Issue the start trigger.

• Monitoring for DMA Done

There are two ways to monitor for DMA done. The easiest is to poll the DMA done bits in the DM6620 status register (BA +6). 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.

• Common DMA Problems

- Check to be sure that your buffer does not straddle a page boundary.
- Remember that the value for the number of samples for the DMA controller to transfer is equal to (the number of samples - 1).
- 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 mask register.

CHAPTER 7

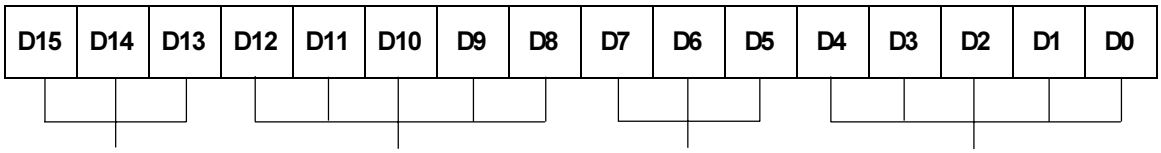
INTERRUPTS

This chapter explains software selectable interrupts, digital interrupts, and basic interrupt programming techniques.

The DM6620 has two completely independent interrupt circuits which can generate interrupts on IRQ channels 3, 5, 9, 10, 11, 12 or 15. By using these two circuits, complex data acquisition systems can be configured.

Software Selectable Interrupt Sources

Each interrupt circuit on the DM6620 has 16 software selectable interrupt sources which can be programmed in bits 0 through 4 and bits 8 through 12 of the Interrupt Register at BA + 2, as described and shown below.



IRQ2 Channel Select
000 = disabled
001 = IRQ3
010 = IRQ5
011 = IRQ9
100 = IRQ10
101 = IRQ11
110 = IRQ12
111 = IRQ15

IRQ2 Source Select
00000 = DAC1 FIFO half full
00001 = DAC2 FIFO half full
00010 = DAC3 FIFO half full
00011 = DAC4 FIFO half full
00100 = Sample Counter 1
00101 = Sample Counter 2
00110 = Sample Counter 3
00111 = DMA 5 done
01000 = DMA 6 done
01001 = DMA 7 done
01010 = User TC Counter 0 out
01011 = User TC Counter 1 out
01100 = User TC Counter 1 out inverted
01101 = User TC Counter 2 out
01110 = digital interrupt
01111 = external interrupt
10000 - 1111 = Reserved

IRQ1 Channel Select
000 = disabled
001 = IRQ3
010 = IRQ5
011 = IRQ9
100 = IRQ10
101 = IRQ11
110 = IRQ12
111 = IRQ15

IRQ1 Source Select
00000 = DAC1 FIFO half full
00001 = DAC2 FIFO half full
00010 = DAC3 FIFO half full
00011 = DAC4 FIFO half full
00100 = Sample Counter 1
00101 = Sample Counter 2
00110 = Sample Counter 3
00111 = DMA 5 done
01000 = DMA 6 done
01001 = DMA 7 done
01010 = User TC Counter 0 out
01011 = User TC Counter 1 out
01100 = User TC Counter 1 out inverted
01101 = User TC Counter 2 out
01110 = digital interrupt
01111 = external interrupt
10000 - 1111 = Reserved

IRQ Sources:

DAC1 FIFO half full - an interrupt is generated when the DAC1 FIFO goes below half full.
DAC2 FIFO half full - an interrupt is generated when the DAC2 FIFO goes below half full.
DAC3 FIFO half full - an interrupt is generated when the DAC3 FIFO goes below half full.
DAC4 FIFO half full - an interrupt is generated when the DAC4 FIFO goes below half full.
Sample Counter 1 - an interrupt is generated when sample counter 1 reaches "0".
Sample Counter 2 - an interrupt is generated when sample counter 2 reaches "0".
Sample Counter 3 - an interrupt is generated when sample counter 3 reaches "0".
DMA 5 done - an interrupt is generated when DMA channel 5 is done.
DMA 6 done - an interrupt is generated when DMA channel 6 is done.
DMA 7 done - an interrupt is generated when DMA channel 7 is done.
User TC Counter 0 out - an interrupt is generated when user TC Counter 0's count reaches 0.
User TC Counter 1 out - an interrupt is generated when user TC Counter 1's count reaches 0.
User TC Counter 1 out inverted - an interrupt is generated when user TC Counter 1's count reaches 0 (useful for frequency counting).
User TC Counter 2 out - an interrupt is generated when user TC Counter 2's count reaches 0.
Digital interrupt - an interrupt is generated when an advanced digital interrupt occurs.
External interrupt - an interrupt is generated when the external interrupt line is pulsed (CN3-19).

Software Selectable Interrupt Channel

Each interrupt circuit on the DM6620 has 7 software selectable interrupt channels which can be programmed in bits 5 through 7 and bits 13 through 15 of the Interrupt Register at BA + 2. The interrupt output is driven by an open collector device which is turned off when the IRQ channel is set to disable. At power up or reset, this register is set to all zero's.

Advanced Digital Interrupts

The bit programmable digital I/O circuitry supports two Advanced Digital Interrupt modes, event mode or match mode. These modes are used to monitor input lines for state changes. The mode is selected at BA + 31, bit 3 and enabled at BA + 31, bit 4.

Event Mode

When enabled, this mode samples the Port 0 input lines at a specified clock rate (using the 8 MHz system clock or a programmable clock in User TC Counter 1), looking for a change in state in any one of the eight bits. When a change of state occurs, an interrupt is generated and the input pattern is latched into the Compare Register. You can read the contents of this register at BA + 30 to see which bit caused the interrupt to occur. Bits can be masked and their state changes ignored by programming the Mask Register with the mask at BA + 30.

Match Mode

When enabled, this mode samples the Port 0 input lines at a specified clock rate (using the 8 MHz system clock or a programmable clock in User TC Counter 1) and compares all input states to the value programmed in the Compare Register at BA + 30. When the states of all of the lines match the value in the Compare Register, an interrupt is generated. Bits can be masked and their states ignored by programming the Mask Register with the mask at BA + 30.

Sampling Digital Lines for Change of State

In the Advanced Digital Interrupt modes, the digital lines are sampled at a rate set by the 8 MHz system clock or the clock programmed in User TC Counter 1. With each clock pulse, the digital circuitry looks at the state of the next Port 0 bits. To provide noise rejection and prevent erroneous interrupt generation because of noise spikes on the digital lines, a change in the state of any bit must be seen for two edges of a clock pulse to be recognized by the circuit. Figure 7-1 shows a diagram of this circuit.

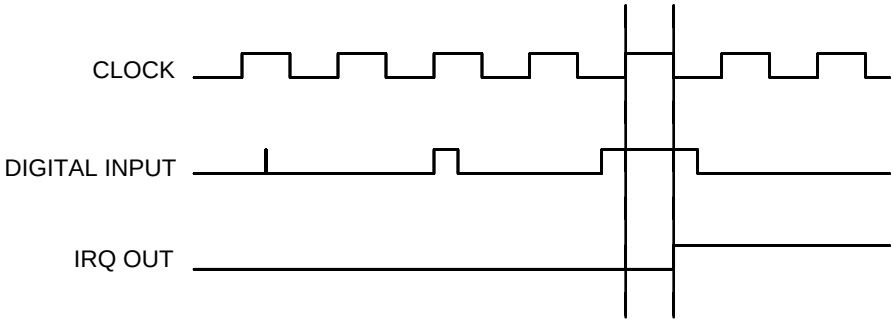


Fig. 7-1 — Digital Interrupt Timing Diagram

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 DM6620 board can interrupt the processor when a variety of conditions are met, such as DMA done, timer countdown finished, sample counter, and external trigger. By using these interrupts, you can write software that effectively deals with real world events.

• Interrupt Request Lines

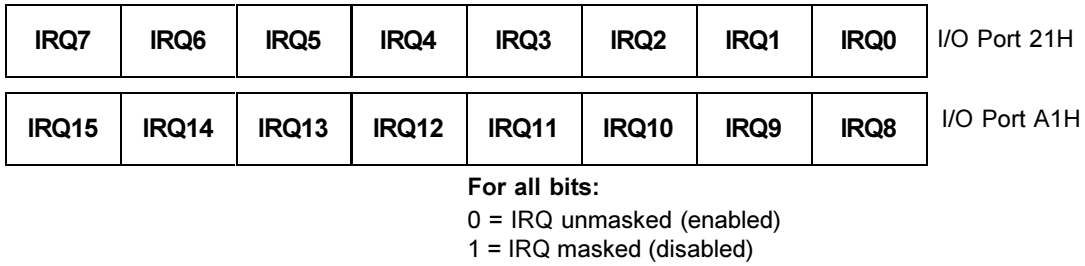
To allow different peripheral devices to generate interrupts on the same computer, the AT bus has 16 different interrupt request (IRQ) lines. A transition from low to high on one of these lines generates an interrupt request which is handled by one of the AT's two interrupt control chips. One chip handles IRQ0 through IRQ7 and the other chip handles IRQ8 through IRQ15. The controller which handles IRQ8-IRQ15 is chained to the first controller through the IRQ2 line. When an IRQ line is brought high, the interrupt controllers check to see if interrupts are to be acknowledged from that IRQ and, if another interrupt is already in progress, they decide 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 determined by the number of the IRQ. Because of the configuration of the two controllers, with one chained to the other through IRQ2, the priority scheme is a little unusual. IRQ0 has the highest priority, IRQ1 is second-highest, then priority jumps to IRQ8, IRQ9, IRQ10, IRQ11, IRQ12, IRQ13, IRQ14, and IRQ15, and then following IRQ15, it jumps back to IRQ3, IRQ4, IRQ5, IRQ6, and finally, the lowest priority, IRQ7. This sequence makes sense if you consider that the controller that handles IRQ8-IRQ15 is routed through IRQ2.

• 8259 Programmable Interrupt Controllers

The chips responsible for handling interrupt requests in the PC are the 8259 Programmable Interrupt Controllers. The 8259 that handles IRQ0-IRQ7 is referred to as 8259A, and the 8259 that handles IRQ8-IRQ15 is referred to as 8259B. To use interrupts, you need to know how to read and set the 8259 interrupt mask registers (IMR) and how to send the end-of-interrupt (EOI) command to the 8259s.

• Interrupt Mask Registers (IMR)

Each bit in the interrupt mask register (IMR) contains the mask status of an IRQ line; in 8259A, bit 0 is for IRQ0, bit 1 is for IRQ1, and so on, while in 8259B, bit 0 is for IRQ8, bit 1 is for IRQ9, 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 for IRQ0-IRQ7 is programmed through port 21H, and the IMR for IRQ8-IRQ15 is programmed through port A1H.



• End-of-Interrupt (EOI) Command

After an interrupt service routine is complete, the appropriate 8259 interrupt controller must be notified. When using IRQ0-IRQ7, this is done by writing the value 20H to I/O port 20H only; when using IRQ8-IRQ15, you must write the value 20H to I/O ports 20H and A0H.

• 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 DM6620), the interrupt controllers check to see if interrupts are enabled for that IRQ, and then checks to see if other interrupts are active or requested and determine which interrupt has priority. The interrupt controllers then interrupt 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 example programs included on your DM6620 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 command to the 8259 controller(s). Since 8259B generates a request on IRQ2 which is handled by 8259A, an EOI must be sent to both 8259A and 8259B for IRQ8-IRQ15. 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 these requirements, 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 and port A0H (if you are using IRQ8-IRQ15).
- 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 8259A (for all IRQs)*/
   outportb(0x20, 0xA0);           /* Send EOI command to 8259B (if using IRQ8-
                                      15) */
}
```

In Pascal:

```
Procedure ISR; Interrupt;
begin
   { Your code goes here. Do not use any DOS functions! }
   Port[$20] := $20;              { Send EOI command to 8259A (for all IRQs) }
   Port[$A0] := $20;              { Send EOI command to 8259B (if using IRQ8-
                                    15) }
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 for IRQ0-IRQ7 is located at I/O port 21H; the IMR for IRQ8-IRQ15 is located at I/O port A1H. The interrupt vector you will be using is located in the interrupt vector table which is simply an array of 256 four-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 IRQ0-IRQ7 are vectors 8 through 15, where IRQ0 uses vector 8, IRQ1 uses vector 9, and so on. The vectors for IRQ8-IRQ15 are vectors 70H through 77H, where IRQ8 uses vector 70H, IRQ9 uses vector 71H, and so on. Thus, if the DM6420 will be using IRQ15, you should save the value of interrupt vector 77H.

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 for IRQ0-IRQ7, or at I/O port A1H for IRQ8-IRQ15 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 on 8259A is arranged so that bit 0 is for IRQ0, bit 1 is for IRQ1, and so on. The IMR on 8259B is arranged so that bit 0 is for IRQ8, bit 1 is for IRQ9, 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 (IRQ0-IRQ7) or I/O port A1H (IRQ8-IRQ15).

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 vectors 8-15 are for IRQ0-IRQ7 and vectors 70H-77H are for IRQ8-IRQ15.

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 before your program started. To restore the IMR, write the value that was saved when your program started to I/O port 21H for IRQ0-IRQ7 or I/O port A1H for IRQ8-IRQ15. Restore the interrupt vector that was saved at startup with either DOS function 25H (set 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 for IRQ0-IRQ7 and 70H through 77H for IRQ8-IRQ15.
- The most common mistake when writing an ISR is forgetting to issue the EOI command to the appropriate 8259 interrupt controller before exiting the ISR.
- Remember to clear the appropriate IRQ circuit on the DM6620 at BA + 14.

CHAPTER 8

TIMER/COUNTERS

This chapter explains the two 8254 timer/counter circuits on the DM6620.

Two 8254 programmable interval timers provide six 16-bit, 8-MHz timer/counters to support a wide range of timing and counting functions. The 8254 at U9 is the Sample Counter TC. All three of these 16-bit timer/counters, Counter 0, Counter 1 and Counter 2, can be programmed as sample counters and used to generate interrupts. These are useful for signaling the CPU to load new data into the D/A buffer.

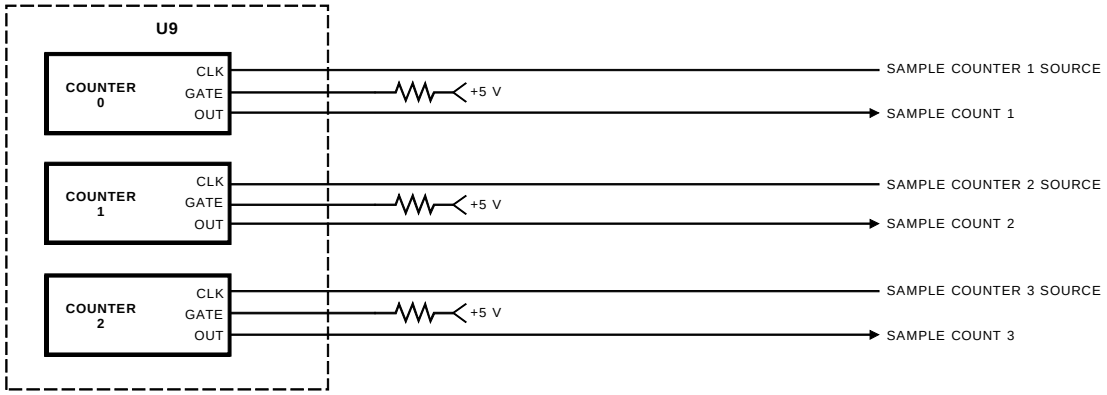


Fig. 8-1 — Sample Counter TC Circuitry

The 8254 at U10 is the User TC. On the User TC, Counters 0, Counter 1 and Counter 2 are available to the user. These timer/counters can also be used to generate clocks as update signals for the D/A converters.

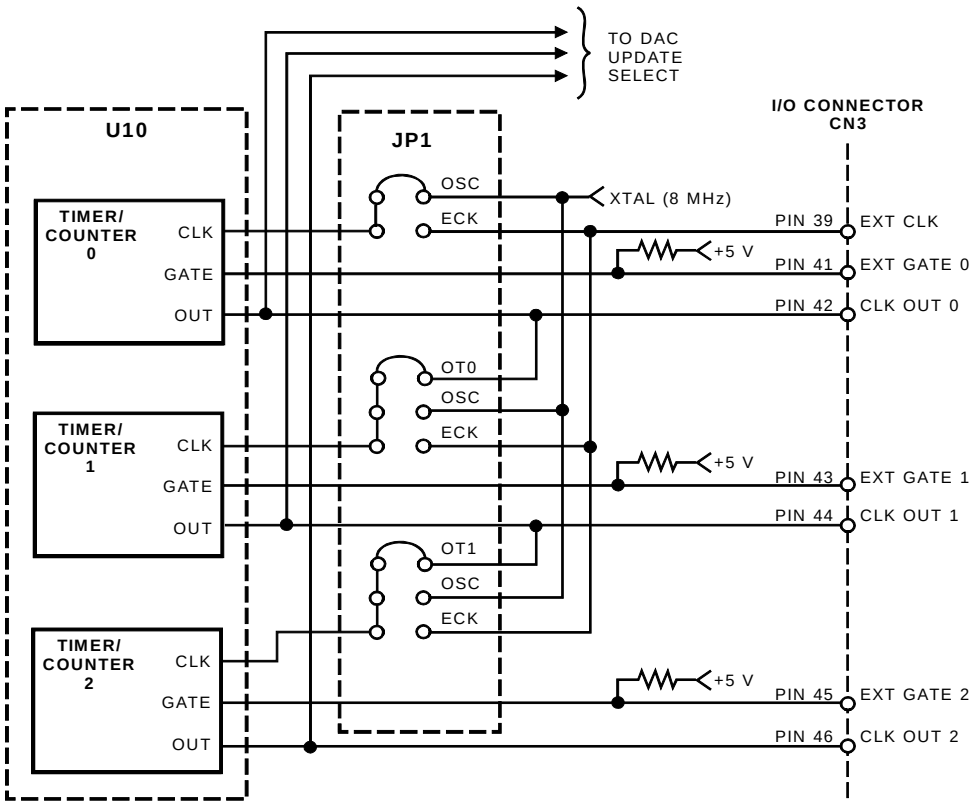


Fig. 8-2 — User TC Circuitry

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 discussion in Chapter 4.

The outputs from the User TC Counters are available at the CN3 I/O connector. These outputs can also be selected through software to be used as the update clock for the D/A converters.

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.

CHAPTER 9

DIGITAL I/O

This chapter explains the bit programmable and port programmable digital I/O circuitry on the DM6620.

The DM6620 has 16 buffered TTL/CMOS digital I/O lines available for digital control applications. These lines are grouped in two 8-bit ports. The eight bits in Port 0 can be independently programmed as input or output. Port 1 can be programmed as an 8-bit input or output port.

Port 0, Bit Programmable Digital I/O

The eight Port 0 digital lines are individually set for input or output by writing to the Port 0 Direction Register at BA + 30. The input lines are read and the output lines are written at BA + 28.

Direction Register:

For all bits:
0 = input
1 = output

D7	D6	D5	D4	D3	D2	D1	D0
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

Advanced Digital Interrupts: Mask and Compare Registers

The Port 0 bits support two Advanced Digital Interrupt modes. An interrupt can be generated when the data read at the port matches the value loaded into the Compare Register. This is called a match interrupt. Or, an interrupt can be generated whenever any bit changes state. This is an event interrupt. For either interrupt, bits can be masked by setting the corresponding bit in the Mask Register high. In a digital interrupt mode, this masks out selected bits when monitoring the bit pattern for a match or event. In normal operation where the Advanced Digital Interrupt mode is not activated, the Mask Register can be used to preserve a bit's state, regardless of the digital data written to Port 0.

When using event interrupts, you can determine which bit caused an event interrupt to occur by reading the contents latched into the Compare Register.

Port 1, Port Programmable Digital I/O

The direction of the eight Port 1 digital lines is programmed at BA + 31, bit 2. These lines are configured as all inputs or all outputs, with their states read and written at BA + 29.

Resetting the Digital Circuitry

When a digital chip clear (BA + 31, bits 1 and 0 = 00 followed by a write to BA + 30), clear board (BA + 14), or reset command is issued, all of the digital I/O lines are set up as inputs.

Strobing Data into Port 0

When not in an Advanced Digital Interrupt mode, external data can be strobed into Port 0 by connecting a trigger pulse through the STRB IN pin at CN3-39. This data can be read from the Compare Register at BA + 30.

CHAPTER 10

EXAMPLE PROGRAMS

This chapter discusses the example programs included with the DM6620.

Included with the DM6620 is a set of example programs that demonstrate the use of many of the module's features. These examples are in written in C and BASIC. Also included is an easy-to-use menu-driven diagnostics program, 6620DIAG, which is especially helpful when you are first checking out your module after installation and when calibrating the module (Chapter 12).

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

C Programs

These programs are source code files so that you can easily develop your own custom software for your DM6620. All of the programs use the files, DRVR6620.C, DIO5812.C and PCUTILS.C. These files contain all of the routines for setting up the board and acquiring data.

DRVR6620.C contains all the functions needed to control the D/A converter and the Timer/Counters. These functions are used to set up the D/A outputs and the timer/counter chips.

DIO5812.C contains all the functions needed to control the digital I/O chip. This chip is the same one used on the Real Time Devices' DM5812 module providing two 8-bit ports. Port 0 can have its lines set as input or output on a bit by bit basis. This allows maximum flexibility when connecting your signals. Port 1 is set to be input or output as a group. In addition, Port 0 supports RTD's two Advanced Digital Interrupt modes. An interrupt can be generated when the lines match a programmed value or when any bit changes its current state. A Mask Register lets you monitor selected lines for interrupt generation.

PCUTILS.C and DMAUTIL.C contain functions to help program the CPU for interrupts and DMA.

Quick Basic Programs

These programs are source code files so that you can easily develop your own custom software for your DM6620. All of the programs rely on the DRVR6620.LIB and the DRVR6620.QLB library files. These library files contain all of functions needed to interface to the DM6620. Make sure the proper library is loaded when starting Quick Basic by typing QB/L DRVR6620. These libraries were created using Borland C 3.1 and were generated from the files DRVR6620.C and DIO5812.C. Should you need to recompile the libraries, contact the factory for details on this procedure.

CHAPTER 11

CALIBRATION

This chapter tells you how to calibrate the DM6620 using the 6620DIAG diagnostic program included in the example software package and the trimpots on the module.

This chapter tells you how to calibrate the D/A converter. The D/A converter is factory-calibrated and 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 6620DIAG diagnostics program is a convenient way to monitor conversions while you calibrate the module.

Calibration is done with the module installed in your system. You can access the trimpots at the edge of the module. Power up the system and let the board circuitry stabilize for 15 minutes before you start calibrating.

Required Equipment

The following equipment is required for calibration:

- Digital Voltmeter: 5-1/2 digits
- Small Screwdriver (for trimpot adjustment)

While not required, the 6620DIAG diagnostics program (included with example software) is helpful when performing calibrations. Figure 12-1 shows the module layout with the trimpots located along the top edge.

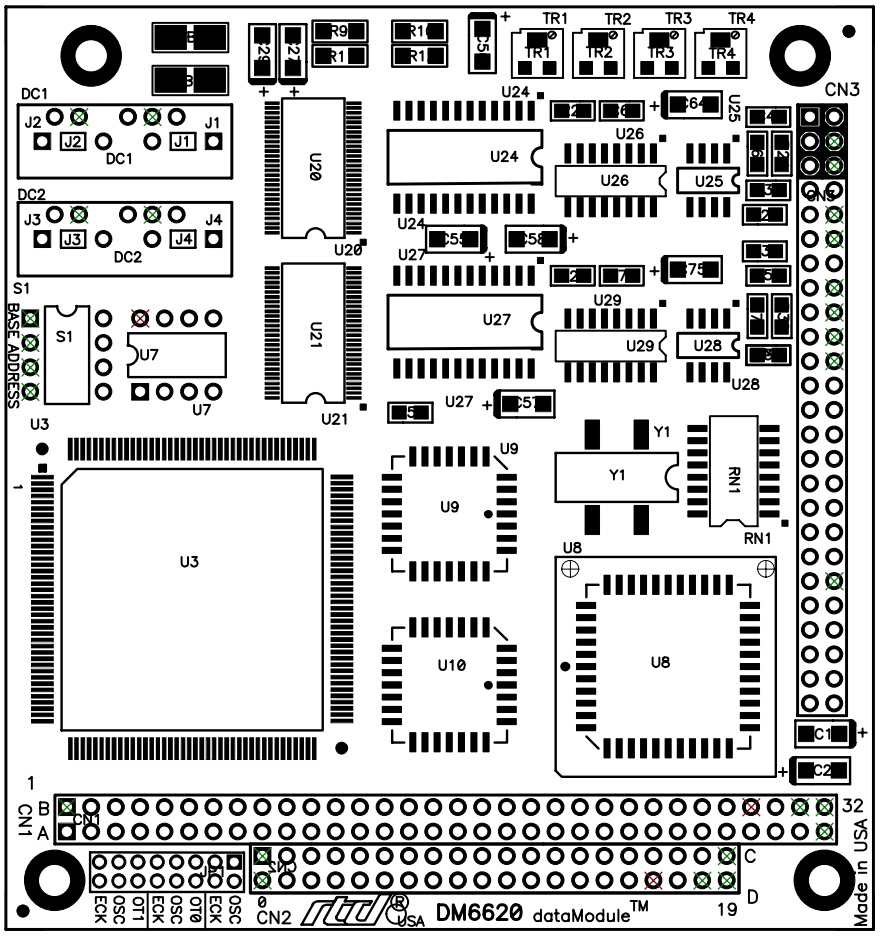


Fig. 11-1 — Module Layout

D/A Calibration

The D/A circuit requires no calibration for the 0 to +5 and ±5 volts ranges. The following paragraph describes the calibration procedure for the 0 to +10 and ±10 volt ranges.

To calibrate for the 0 to +10 and ±10 volt ranges, program the DAC outputs for a 0 to +10 volt range at the DAC Range register, BA + 10. Now program the D/A outputs with a digital value of 2048. The ideal DAC output for a code of 2048 is +5.000 volts. Connect a voltmeter to the DAC outputs and adjust TR1 for DAC1, TR2 for DAC2, TR3 for DAC3 and TR4 for DAC4 until 5.000 volts is read on the meter.

The following tables show the ideal output voltage per bit weight for unipolar and bipolar ranges.

Unipolar D/A Bit Weight	Ideal Output Voltage (millivolts)	
	0 to +5 Volts	0 to +10 Volts
4095	+4998.78	+9997.56
2048	+2500.00	+5000.00
1024	+1250.00	+2500.00
512	+625.00	+1250.00
256	+312.50	+625.00
128	+156.25	+312.50
64	+78.13	+156.25
32	+39.06	+78.13
16	+19.53	+39.06
8	+9.77	+19.53
4	+4.88	+9.77
2	+2.44	+4.88
1	+1.22	+2.44
0	0.00	0.00

Bipolar D/A Bit Weight	Ideal Output Voltage (millivolts)	
	-5 to +5 Volts	-10 to +10 Volts
2047	+4997.56	+9995.12
1024	+2500.00	+5000.00
512	+1250.00	+2500.00
256	+625.00	+1250.00
128	+312.50	+625.00
64	+156.25	+312.50
32	+78.13	+156.25
16	+39.06	+78.13
8	+19.53	+39.06
4	+9.77	+19.53
2	+4.88	+9.77
1	+2.44	+4.88
0	0.00	0.00
-1	-2.44	-4.88
-2	-4.88	-9.77
-4	-9.77	-19.53
-8	-19.53	-39.06
-16	-39.06	-78.13
-32	-78.13	-156.25
-64	-156.25	-312.50
-128	-312.50	-625.00
-256	-625.00	-1250.00
-512	-1250.00	-2500.00
-1024	-2500.00	-5000.00
-2048	-5000.00	-10000.00

APPENDIX A

DM6620HR SPECIFICATIONS

DM6620HR Characteristics Typical @ 25° C

Interface

Switch-selectable base address, I/O mapped
Software programmable interrupts & DMA channel

D/A Converter

Analog outputs ..... 4 channels
Resolution ..... 12 bits
Output ranges ..... 0 to +5, ±5, or 0 to +10 volts
Relative accuracy ..... ±1 bit, max
Full-scale accuracy ..... ±5 bits, max
Non-linearity ..... ±1 bit, max
Full-scale settling time ..... 5 µsec, typ
Output current ..... 5 ma, typ

D/A Sample Buffer

FIFO Size (each channel) ..... 1024 samples

Digital I/O

Number of lines ..... 8 bit programmable & 8 port programmable
I/O type ..... TTL
Input/Output levels ..... 0 to +5 volts
Isource ..... -12 MA
Isink ..... 24 mA

User 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

DM6620: +5 volts @ 420 ma, 2.1 W typ.

CN3 Connector

50-pin right angle header

Environmental

Operating temperature ..... -40 to +85°C
Storage temperature ..... -55 to +125°C
Humidity ..... 0 to 90% non-condensing

Size

3.55"L x 3.775"W x 0.6"H (90mm x 96mm x 16mm)

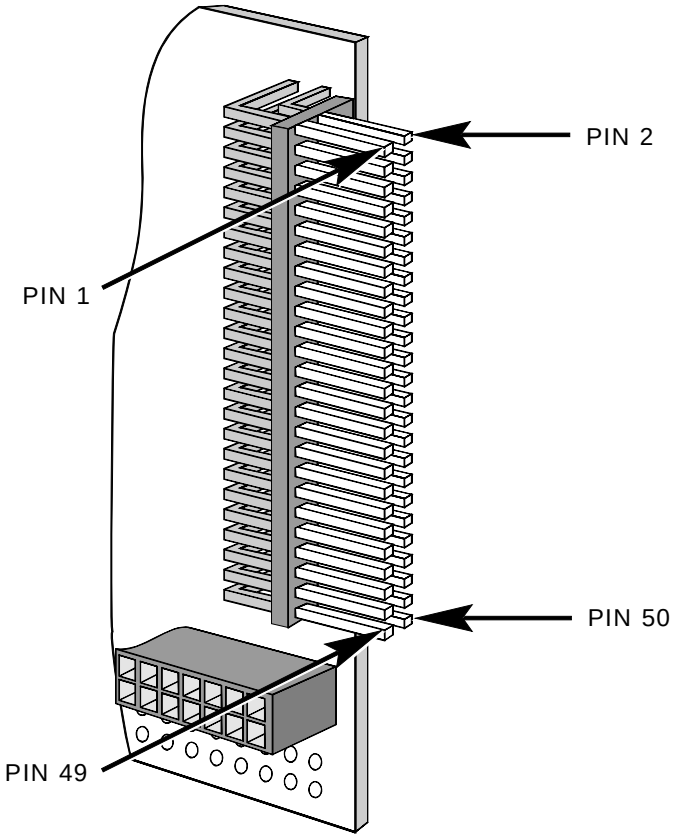
Weight

3 oz. (86 grams)

APPENDIX B

CN3 CONNECTOR PIN ASSIGNMENTS

AOUT1	1	2	ANALOG GND
AOUT2	3	4	ANALOG GND
AOUT3	5	6	ANALOG GND
AOUT4	7	8	ANALOG GND
DAC1 DATA MARKER	9	10	DIGITAL GND
DAC2 DATA MARKER	11	12	DIGITAL GND
DAC3 DATA MARKER	13	14	DIGITAL GND
DAC4 DATA MARKER	15	16	DIGITAL GND
N.C.	17	18	DIGITAL GND
EXT INT	19	20	DIGITAL GND
DIGITAL GND	21	22	DIGITAL GND
P0.7	23	24	P1.7
P0.6	25	26	P1.6
P0.5	27	28	P1.5
P0.4	29	30	P1.4
P0.3	31	32	P1.3
P0.2	33	34	P1.2
P0.1	35	36	P1.1
P0.0	37	38	P1.0
EXT CLK	39	40	DIGITAL GND
EXT GATE 0	41	42	CLK OUT 0
EXT GATE 1	43	44	CLK OUT 1
EXT GATE 2	45	46	CLK OUT 2
+12 VOLTS	47	48	+5 VOLTS
-12 VOLTS	49	50	DIGITAL GND



NOTE:
On the DM6620, +12 volts at pin 47 and -12 volts at pin 49 are available only if supplied by the computer bus.

CN3 Mating Connector Part Numbers	
Manufacturer	**Part Number**
AMP	1-746094-0
3M	3425-7650

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

DM6620 User Settings	
Base I/O Address:	
(hex)	(decimal)
IRQ Channel:	
DMA Channel:	