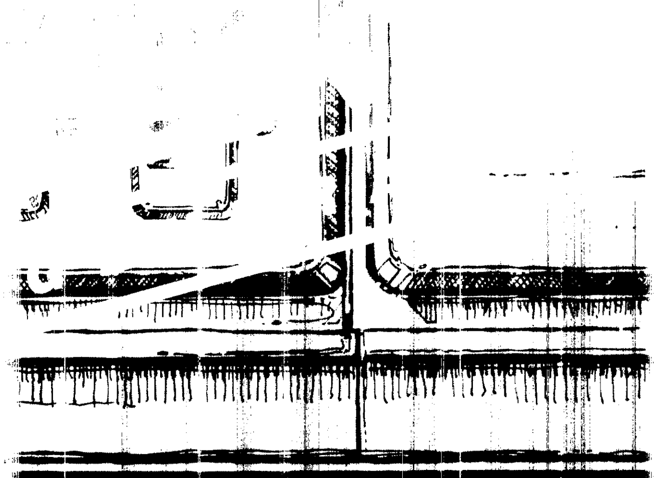
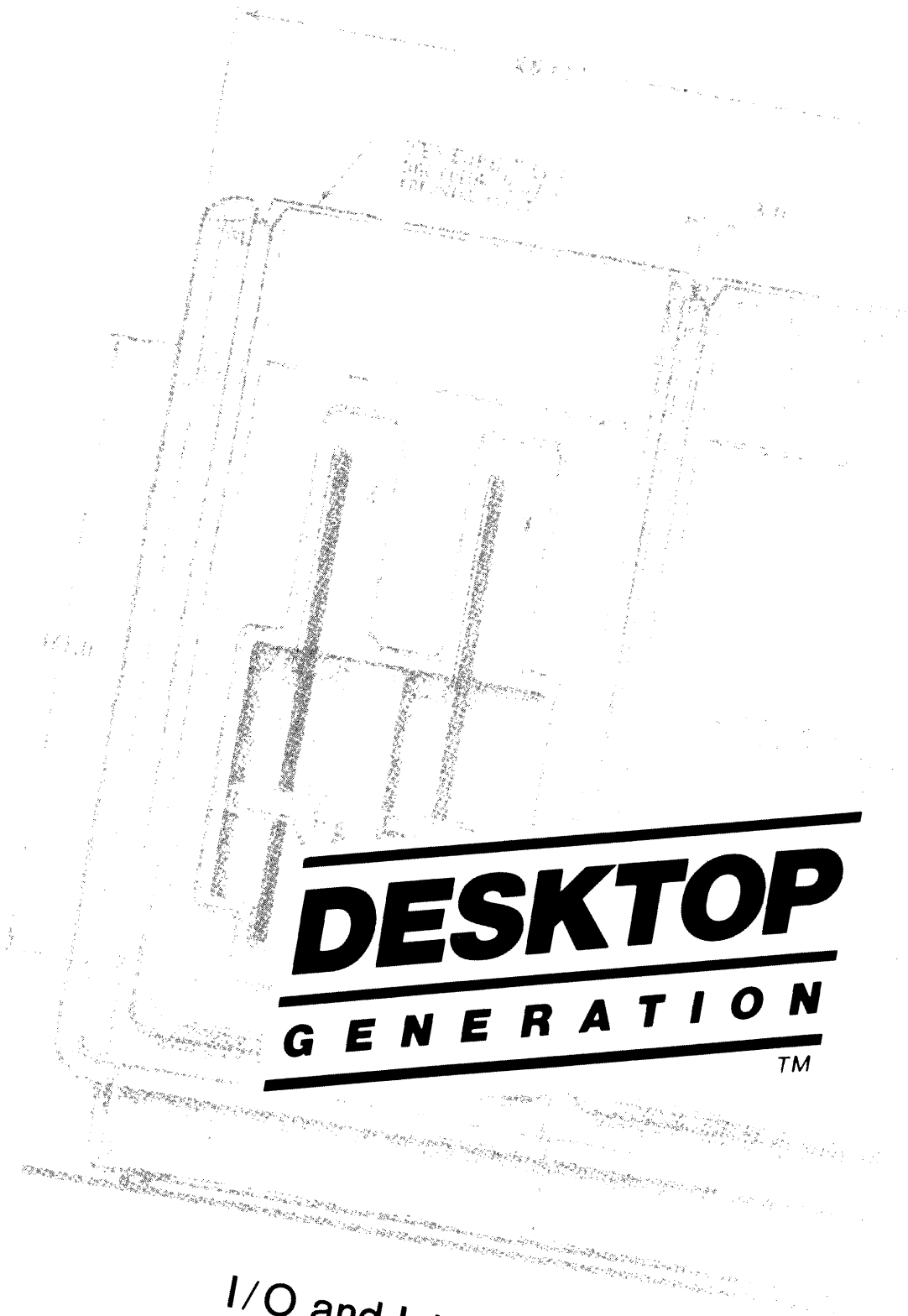


Data General

Technical Referen



DESKTOP GENERATION™

I/O and Interfacing

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

The terms and conditions governing the sale of DGC hardware products and the licensing of DGC software consist solely of those set forth in the written contracts between DGC and its customers. No representation or other affirmation of fact contained in this document including but not limited to statements regarding capacity, response-time performance, suitability for use or performance of products described herein shall be deemed to be a warranty by DGC for any purpose, or give rise to any liability of DGC whatsoever.

In no event shall DGC be liable for any incidental, indirect, special or consequential damages whatsoever (including but not limited to lost profits) arising out of or related to this document or the information contained in it, even if DGC has been advised, knew or should have known of the possibility of such damages.

Ordering No. 014-000774

Printed in the United States of America
Rev. 00, October 1983

Preface

The technical reference manuals for Desktop Generation ™ computers and their peripherals are written for assembly language programmers, systems analysts, and engineers. This set of manuals, together with two companion programmer's references, contains the information you need to: 1) write assembly language software, including I/O subroutines; 2) knowledgeably expand your system; 3) learn how your system operates at the card level; and 4) design custom interfaces.

This manual introduces the microI/O bus and describes the I/O interface required to communicate with this bus and its host Desktop Generation computer. Other technical and programmer's references for Desktop Generation computers are listed and summarily described under "Related Manuals" in this preface.

Organization

This book includes four chapters, three appendixes, and an index. It is organized so that portions of it can be read selectively.

- Chapter 1 defines important concepts and terms.
- Chapter 2 introduces the I/O instructions that allow the program to communicate with peripheral controllers and discusses the program interrupt and data channel facilities.
- Chapter 3 describes the I/O bus to I/O controller interface that is standard to all peripherals connected to the microI/O bus of Desktop Generation computers.
- Chapter 4 describes a general-purpose interface card suitable for building a custom interface to a Desktop Generation computer.
- Appendix A describes a general-purpose wiring card included in Data General's assemblies.
- Appendix B presents the 7-bit International Standards Organization (ISO) character code and its ASCII equivalent.
- Appendix C lists the I/O device codes for Desktop Generation computers.

A documentation comment form follows the index. It invites you to help Data General improve its publications by commenting on this book.

Related Manuals

A comprehensive documentation set supports all the hardware and software products available for Desktop Generation computers. The hardware-related books listed below fall into three categories: the technical reference series; the user guides for operating, installing, and testing; and the introductory guide for Desktop Generation computers.

The following technical and programmer's references address the needs of assembly language programmers and engineers.

16-bit Real Time ECLIPSE Assembly Language Programming

Global in nature, this book explains the processor-independent concepts, functions, and instruction sets of 16-bit ECLIPSE computers. DGC ordering no. 014-000688.

Model 10 and 10/SP Computer Systems
Technical Reference

In addition to the functional and physical organization of Model 10 and 10/SP computers and their technical specifications, this book explains their processor-unique concepts, functions, and instruction set features. The theory of operation for the basic components of Models 10 and 10/SP. DGC ordering no. 014-000766.

Model 10 and 10/SP System Console
Programmer's Reference

Describes the organization and alphanumeric and graphic features of the system console. Defines the command sets and includes guidelines for programming the monochrome and optional color monitors at assembly and high-level language levels. DGC ordering no. 014-000770.

Model 20 and 30 Computer Systems
Technical Reference

In addition to the functional and physical organization of Model 20 and 30 computers and their technical specifications, this manual explains their processor-unique concepts, functions, and instruction set features. Provides detailed information for programming the systems' I/O devices, including the diskette subsystem, and explains the theory of operation for the basic components of Models 20 and 30. DGC odering no. 014-000767.

Communications Interfaces
Technical Reference

Discusses the functional and physical organization of the asynchronous/ synchronous communications interfaces available for Desktop Generation computers. Defines their I/O instruction sets, offers guidelines for writing assembly language I/O subroutines, and contains theory of operation for each communications card. DGC ordering no. 014-000769.

Sensor I/O
Technical Reference

Defines instruction sets, offers guidelines for writing assembly language I/O subroutines, describes theory of operation at an overview level, and explains how to connect field wiring for the 4222 digital I/O interface, 4223 analog-to-digital interface, 4224 digital-to-analog interface, and 4335 analog subsystem. DGC ordering no. 014-000775.

Disk Subsystem Technical Reference

Describes the functional and physical organization of the Model 6271 disk subsystem. Defines the I/O instruction set and provides guidelines for programming the subsystem. DGC ordering no. 014-000768.

IEEE-488 Bus Interface
Technical Reference

Provides the information needed to interface, program in assembly language, and troubleshoot this card in a Desktop Generation system. Reviews the contents of the IEEE-488 bus standard, summarizing its commands, messages, and states, and includes a theory of operation. DGC ordering no. 014-000773.

The following books are how-to manuals written for anyone who needs to know how to install, operate, and test a Desktop Generation system.

Installing Model 10 and 10/SP Systems

The first book that a Model 10 or 10/SP owner should read, explains how to unpack and install either system and its optional peripherals. Simple instructions and ample illustrations make the book accessible to any reader. DGC ordering no. 014-000901.

Operating Model 10 and 10/SP Systems

A logical follow-on to Model 10 and 10/SP installation, this guide takes you from powering up the system and its optional peripherals through performing such routine operations as loading paper in a printer and inserting or removing diskettes. Brings you to the point of loading the system software. Amply illustrated and written for users at any level of experience. DGC ordering no. 014-000900.

Testing Model 10 and 10/SP Systems

Follows the installation and operating manuals with instructions for verifying the operation of Model 10 or 10/SP systems and their optional peripherals. Steps you through the power-up test and Customer Diagnostics and explains how to troubleshoot customer-replaceable components. Simple instructions and diagrams make the book accessible to any user. Includes phone numbers for Data General assistance. DGC ordering no. 014-000902.

Installing Model 20 and 30 Systems

The first book a Model 20 or 30 owner should read, explains how to unpack and install either system and its optional peripherals. Accessibly written and illustrated, for users at any level of experience. DGC ordering no. 014-000904.

Operating Model 20 and 30 Systems

Follows Model 20 and 30 installation, leading you from powering up the system and its optional peripherals through performing such routine operations as loading paper in a printer and inserting or removing diskettes. Brings you to the point of loading the system software. The simple instructions and generous illustrations are suitable for any reader. DGC ordering no. 014-000903.

Testing Model 20 and 30 Systems

A follow-on to the installation and operating mauals, explains how to verify the operation of Model 20 or 30 systems and their optional peripherals. Simple instructions and diagrams lead you through the power-up test, Customer Diagnostics, and trouble-shooting of customer-replaceable components. Includes phone numbers for Data General assistance. DGC ordering no. 014-000905.

This last book is a product overview, addressed to all Desktop Generation users.

The Desktop Generation

Introduces the Desktop Generation, summarizing each model of the family, and describes its many hardware and software products, features, and capabilities. Includes a brief history of Data General, a sampling of applications, and an overview of the customer service and support programs available to you as a Desktop Generation user. DGC ordering no. 014-000751.

Conventions

The following conventions are used throughout this manual.

MNEMONIC	Uppercase sans serif letters indicate a signal name or instruction mnemonic. When a signal is active low, it is barred—for example, $\overline{\text{FDCHE}}$.
argument	Italicized lowercase letters mean that a particular instruction takes an argument. In your program, you must replace this symbol with the exact code for the argument you need.
[optional]	Brackets signify an optional argument. If you decide to use this argument, do not include the brackets in your code; they only set off the choice.

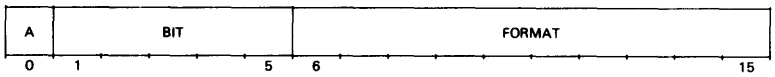
In dialogs between system and user, we use this typeface to show your input:

USER INPUT

and this typeface to show the system's response:

SYSTEM RESPONSE.

In addition, we use the following diagram to show the arrangement of the 16 bits in an instruction. The diagram is always divided into 16 boxes, numbered 0 through 15.



Contents

Overview 1

While you do not need to know much about the input/output architecture of a Desktop Generation computer to use a peripheral sold by Data General Corporation, you do need to understand that architecture in order to design and use custom-built I/O equipment.

This book is intended to supply the background you need to build interfaces. This chapter defines important concepts and terms. Our presentation assumes that you have a working knowledge of digital circuits and some experience with digital computers.

Interfacing

A great advantage of the present generation of microcomputers is that they can be connected, or interfaced, to a variety of custom peripherals. In Data General's microcomputers, interfaces stand between the devices they control and the CPU, communicating with the peripherals individually and with the processor over an *I/O bus*, a set of wires carrying signals to all interfaces.

It is easy to interface to Data General's microcomputers for a number of reasons:

- You can build a reliable interface — even a large one — on a single printed circuit card measuring 7.5 x 10.4 inches, using a minimum of off-card connections.
- Data General provides an LSI circuit, called an I/O controller (microNOVA IOC), which encapsulates the protocol logic required on each peripheral device interface.
- Data General also makes available general-purpose interface cards that simplify the job of designing and building an interface.

Definition of Terms

A peripheral generally consists of two units, a device and a controller. The device (typically called a drive, a transport, or a terminal) reads, writes, stores, or processes information. For example, a terminal's keyboard reads information; a plotter writes information; a disc drive stores information; and an A/D converter processes information.

The controller portion of the peripheral is the interface between the CPU and the device. It interprets commands from the computer to the device and passes information between them.

As shown in Figure 1-1, the communications channel through which all CPU/controller information passes is called the input/output (I/O) bus. Since this bus is shared by all of the controllers as well as by the CPU, it is, by necessity, a half-duplex bus; i.e., only one operation can occur at any time. The direction of all information transfers on the I/O bus is defined relative to the computer.

Output always refers to moving information from the processor to a controller; input always refers to moving information from a controller to the processor.

Types of Information

The information transferred between the CPU and a controller can be classified into three types: status, control, and data. Status information tells the CPU about the state of the peripheral. Is it busy? Is it ready? Is it operating properly? Control information is transferred by the processor to a controller to tell the peripheral what to do. Data are the information that come from or go to the device during reading, writing, storing, or processing.

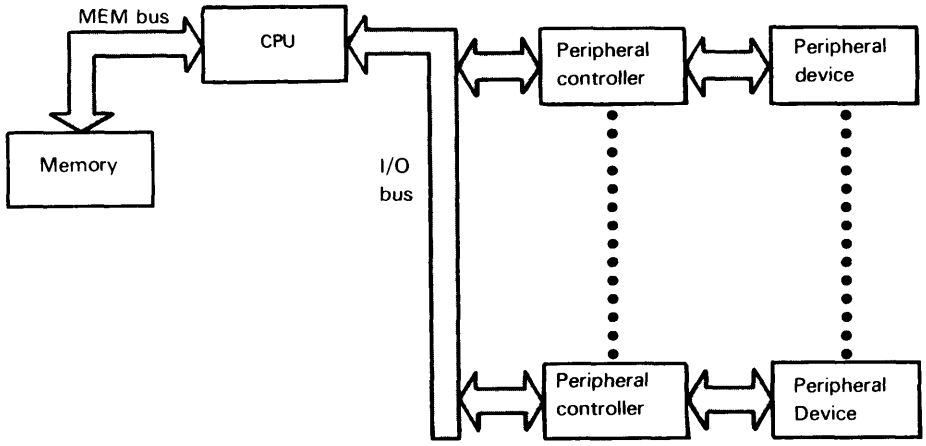


Figure 1-1 System diagram

Types of Information Transfer

Information can be transferred between the CPU and a peripheral in one of two ways: under direct program control or under data channel control. An information transfer occurring under direct program control moves a word or part of a word between an accumulator in the CPU and a register in the controller. This type of transfer occurs when an appropriate I/O instruction is executed in the program. An information transfer under data channel control generally moves a block of data, one word at a time, between memory and the device, through a register in the peripheral's controller.

The block of data is transferred automatically via the data channel once the transfer for a particular peripheral is set up via direct program control.

Direct Program Control

Direct program control of information transfers, also called programmed I/O, is a way of transferring single words or parts of words to or from peripherals. Among the peripherals which transfer data in this way are line printers, terminals, paper tape readers and punches. The data moving through an accumulator are readily available to the program for manipulation or decision making. In the case of input, for example, the program can decide whether to read another word or character based on the value of the word or character that was just read.

However, programmed I/O is slow, because at least one instruction, and most likely several, must be executed for each character or word transferred. It is generally used only for peripherals that do not have to transfer large quantities of information quickly.

Data Channel Control

Some peripherals, such as disk subsystems, are used to access large blocks of data. In order to reduce the amount of program overhead required, these blocks are transferred under data channel control. The commands used to set up the data channel transfer are transferred to the controller under direct program control. The block of data is then automatically transferred between memory and the controller via the data channel. Thus, the program defines the block of memory to be transferred, but it does not have to transfer each word of the block.

Once the data channel transfer for a block of data has been set up and initiated by the program, no further action by the program is required to complete the transfer. The program can proceed with other tasks while the block transfer is taking place. Each time the controller is ready to transfer a word from the block, it requests data channel service. When the CPU responds to the request, the word is transferred. Because several instructions do not have to be executed for each word transferred, block transfers can occur at a higher rate.

Since the actual transfer of a word via the data channel could conflict with the program instructions being executed, the program pauses during the transfer of each word. This pause is transparent to the programmer, except that it lengthens the time required for program execution.

Program Interrupt Facility

When transferring information under either direct program control or data channel control, the program must be able to determine when the transfer is complete, so that it can start a new transfer or proceed with a task that was dependent on the transfer just completed. Peripherals have status flags which can provide the program with this needed information. The I/O instruction set allows the program to check the status of these flags and make decisions based on the results of the checks. However, these status checks are time consuming. Thus, all Data General computers incorporate a program interrupt facility, which eliminates the need to continually perform status tests.

The program interrupt facility provides a peripheral with a convenient means of notifying the processor that it requires service by the program. This is accomplished by allowing the peripherals to interrupt normal program flow on a priority basis. When a peripheral completes an operation or encounters a situation requiring processor intervention, it can request a program interrupt of the processor. The processor honors such a request by interrupting the program in process, saving the address where the interruption occurred, and transferring control to the interrupt handling routine. This routine identifies the peripheral requiring service, transfers control to the service routine for that peripheral, and after the peripheral has been serviced, restores the system to the state it was in when the interrupt occurred.

For systems requiring large amounts of I/O to many devices, a multilevel priority structure up to 16 levels deep can be established by software. This structure can be set up to provide rapid service to those devices which are crucial to the efficient operation of the computer system; the less critical devices can be interrupted by the more critical devices.

Input/Output Programming

2

Information transfers between the processor and the various peripherals are governed by eight instructions that constitute the I/O instruction set. This chapter covers only those I/O instructions that allow the program to communicate with the peripheral's controllers and to control the program interrupt facility. The chapter introduces these instructions to those who have no experience with Data General's I/O system.

The Typical Controller

The specific effects of I/O instructions necessarily depend on the peripherals to which they are addressed. However, their general functions (loading and reading registers, issuing control signals, and testing flags) are the same for all peripherals; different peripherals merely use the available functions in different ways. In order to understand the general functions performed by the I/O instructions and how these functions are typically used by peripheral controllers, you should know something about the architecture of a peripheral controller.

From the point of view of the program, a peripheral controller operates as a collection of data registers, control registers, and status flags, with which communications are established (see Figure 2-1). With these registers and flags, the program can route data between the CPU and the device, as well as monitor the operation of the device.

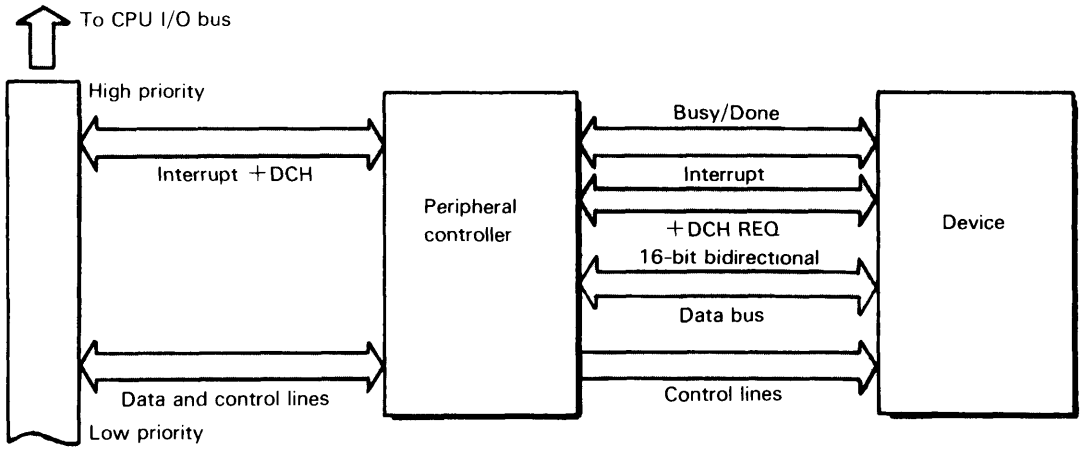


Figure 2-1 Peripheral controller

The distinction made here between registers and flags is generally one of information content. A flag contains a single bit of information, while a register is made up of a number of bits. Groups of contiguous bits in a register which convey a single piece of information are referred to as *fields.*

The paragraphs below describe only the basic components of a typical controller. The additional structure required for a peripheral using the program interrupt facility or the data channel is discussed in the chapters describing those facilities. What follows merely typifies the workings of a controller; in any concrete situation, the controller is tailored to the specific devices it controls. Specifications of the I/O interface for Desktop Generation computers are given in Chapter 3.

The registers in a controller are of three types, depending on the kind of information that is stored in them: data registers, control registers, and status registers.

Data Registers

Data registers (or data buffers) store data in the controller as they pass between the device and the computer. These buffers are needed because the computer and the device usually operate at different speeds. Since the operation of nearly all peripherals involves the transfer of a word or part of a word of data between the computer and the device, nearly all peripheral controllers contain a data buffer.

In the case of a peripheral that transfers data under direct program control, the data buffer is directly accessible to the program. An I/O instruction transfers data between the register in the controller and an accumulator in the central processor.

In the case of a peripheral that operates under data channel control, the data are transferred between the register in the controller and memory. Data buffers in the controllers which use the data channel need not be, and usually are not, accessible to the CPU through programmed I/O.

Control Registers

Control registers allow the program to supply the controller with information necessary for the operation of the device, such as drive or communication line numbers, data block sizes, and command specification. A unit of control information is called a *control parameter*.

Control parameters typically allow the program to select one of a number of peripheral units in a subsystem, the operation to be performed, and the initial values for flags and counters in the controller. The program specifies control parameters to the controller with an I/O instruction, wherein the desired parameters are coded into the appropriate fields of the accumulator used in the transfer.

Status Registers

Status registers are used to indicate to the program the state of the peripheral. They consist primarily of status flags, but can also contain control parameters. The control parameters contained in status registers are commonly those that change during the operation of the peripheral; these are therefore important to the program, which must check on the progress of the peripheral's operation. For example, a program transferring consecutive sectors of information on a disk in a single operation can read the current sector address and sector count during the operation in order to determine how far the operation is from completion. Status flags are set by the controller to indicate error conditions or to notify the computer of the basic state of the peripheral.

The classification of controller registers into the three types described above is only a general one. A register may contain more than one type of information. Often a register serves as a control register when loaded by the program and as a status register when read by the program. The disk address/sector counter register mentioned in the preceding paragraph is an example of a combined control and status register.

Busy and Done Flags

Busy and Done flags—the two fundamental flags in a controller—serve a dual purpose. Together they denote the basic state of the peripheral and can be tested by the program to determine that state. In addition, the program can manipulate these flags in order to control the operation of the peripheral.

The function of the Busy and Done flags is device dependent. However, most devices use the flags in the following manner. To place the peripheral in operation, the program sets the Busy flag to 1. The Busy flag remains in this state for the duration of the operation, indicating that the peripheral is in use and should not be disturbed by the program. When the peripheral completes its operation, the controller sets the Busy flag to 0 and the Done flag to 1.

The setting of the Done flag to 1 can be used to trigger a program interrupt. Whether or not a program interrupt occurs depends on the state of the interrupt facility.

For a relatively simple peripheral, the Busy and Done flags alone may furnish enough status information to allow the program to service the peripheral adequately. However, a more complex peripheral will generally require additional status flags to specify its internal operating conditions more completely to the program.

The difference between these additional status flags and the Busy and Done flags is that the Busy and Done flags may be tested directly with a single I/O instruction while any other status flag requires that its value first be read into an accumulator from the status register. The program may then perform any test it requires on the status word after it is read.

Error Flags

Status flags that indicate errors or malfunctions in the operation of a peripheral are termed *error flags*. Two types of error flags can be characterized according to their effect on the operation of the peripheral when they are set.

The first, or passive, type is merely set by the controller in the course of the operation when the associated error occurs. No immediate indication of this type of error is given to the program, and the operation is allowed to continue to completion.

The second or active type of error flag is set by the controller when the program attempts to start an operation which is not allowed. In this case, the operation never begins and the Done flag is immedately set to 1 to notify the program. This type of error flag is used to prevent a severe and probably irrecoverable error from occurring.

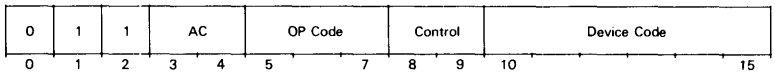
In either case, the program should respond, error or not, when it notices that a peripheral is "done." It need only check the appropriate error flag or flags before assuming that the operation it initiated was satisfactorily completed.

For example, the controller of a diskette subsystem contains error flags to indicate address and data errors. During a read operation, when a checkword error occurs, the Checkword Error flag is set to 1. No immediate notification of the data error is given to the program and the read operation is allowed to finish. The error can be detected at the completion of the operation, when the program should check for errors. At this time, appropriate action can be taken, such as trying to read the misread sector of the diskette again or printing an error message on the console terminal.

The Address Error flag, on the other hand, immediately sets the Done flag to 1 to notify the program.

Instruction Format

The general format of the I/O instructions is shown below.



Bits 0 to 2 are 011, identifying this as an I/O instruction; bits 3 and 4 specify an accumulator; bits 5 to 7 contain the operation code; bits 8 and 9 specify a flag control function or test condition; and bits 10 to 15 specify the code of the device.

Device Code Field

Bits 10 to 15 in an I/O instruction select the peripheral that is to respond to the instruction. The instruction format thus allows 64 device codes, numbered 0 to 77_8. In all Desktop Generation computers, device codes 0, 1, 2, and 3 are reserved, and device code 77_8 is used to implement a number of specific processor functions, such as controlling the program interrupt facility. The remaining device codes are available for referencing peripherals. Many of these codes have been assigned by Data General Corporation to standard peripherals and the assembler recognizes convenient mnemonics for these codes. The list of the standard device code assignments and their associated mnemonics is given in Appendix C.

Flag Control Field

Device flag commands are issued and Busy and Done flags are either manipulated or tested according to the setting of bits 8 and 9 of the I/O instructions. In those instructions that allow flag manipulation, bits 8 and 9 are referred to as the *f* field. Table 2-1 shows the available flag control commands. The control function of each command is device dependent.

Table 2-1 Flag control commands

F Field	Command	Mnemonic
00	(none)	(omitted)
01	Start	S
10	Clear	C
11	Pulse	P

The *I/O Skip* instruction allows testing of the Busy and Done flags; in this case, bits 8 and 9 are referred to as the *t* field. Table 2-2 shows the test conditions that these bits can select.

Table 2-2 Test conditions selected by I/O Skip instruction

T Field	Mnemonic	Next Instruction Is Skipped If:
00	BN	Busy flag is 1 (nonzero).
01	BZ	Busy flag is 0 (zero).
10	DN	Done flag is 1 (nonzero).
11	DZ	Done flag is 0 (zero).

Two important features of the I/O instruction set result from the nature of the flag control field. First, because the flag control field is separate from the operation code field, a single I/O instruction can both transfer information between the controller and the computer and simultaneously control the operation of the peripheral. Secondly, the use of the flag control field as a *t* field allows the direct testing of a controller's Busy or Done flag in a single instruction. Thus, quick decisions based on the basic state of the peripheral can be made by the program.

Operation Code Field

The 3-bit operation code field selects one of the eight I/O instructions. In two of these instructions, no information transfer is specified; instead, bits 8 and 9 may specify either a control function or a flag test condition as described above. The remaining six instructions involve an information transfer between the computer and the designated peripheral controller; they may also specify a control function to be performed after the information transfer has been completed.

The program can, therefore, access up to six registers in any one controller. Up to three of these six registers are output registers which can be loaded by the program with either data or control information. The other three are input registers from which the program can read either data or status information. Frequently, two different I/O instructions, one input and one output, reference the same register in a controller. However, this is not in any way required by the nature of the I/O instruction set.

In order to give names and mnemonics to the I/O instructions in their general form, the registers in a peripheral controller that are accessible to the program are referred to with letter designations. The three input registers are called the "A input buffer," the "B input buffer," and the "C input buffer." Similarly, the three output registers are called the "A output buffer," the "B output buffer," and the "C output buffer". Thus, for example, to read data from a peripheral controller's A input buffer, A *Data In A* instruction (DIA) is issued to that peripheral.

Table 2-3 lists the eight operation codes, their associated mnemonics, and the instructions specified.

Table 2-3 Operation codes

Operation Code Field	Mnemonic	Instruction
000	NIO	No input or output, but performs the flag control function specified.
001	DIA	Reads data into the specified accumulator from the A input buffer.
010	DOA	Writes data out from the specified accumulator to the A output buffer.
011	DIB	Reads data into the specified accumulator from the B input buffer.
100	DOB	Writes data out from the specified accumulator to the B output buffer.
101	DIC	Reads data into the specified accumulator from the C input buffer.
110	DOC	Writes data out from the specified accumulator to the C output buffer.
111	SKP	Skips the next instruction if the test selected for the Busy or Done flag is true.

Accumulator Field

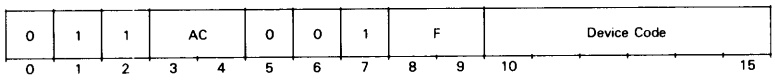
Bits 3 and 4 in an I/O instruction select one of the central processor's four accumulators: AC0, AC1, AC2, AC3. In those instructions which involve an information transfer between the processor and a peripheral controller, the specified accumulator either furnishes the information for an output transfer or receives the information in an output transfer. In the two I/O instructions which do not involve an information transfer, the accumulator field is ignored. The assembler sets bits 3 and 4 in these instructions to 0.

Instructions

The eight I/O instructions are listed and described below. For an explanation of the coding conventions used, see the preface to this manual.

Data in A

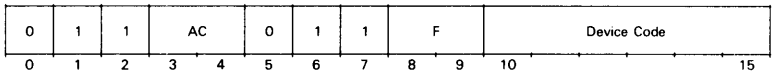
DIA*[f]* *ac,device*



Places the contents of the A input buffer of the specified controller in the specified accumulator. The previous contents of the accumulator are lost. After the data transfer, performs the function specified by *f*.

Data In B

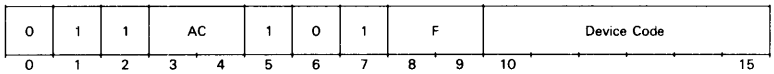
DIB*[f]* *ac,device*



Places the contents of the B input buffer of the specified controller in the specified accumulator. The previous contents of the accumulator are lost. After the data transfer, performs the function specified by *f*.

Data In C

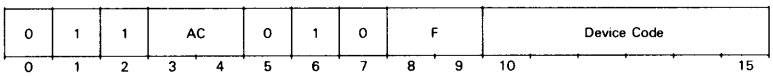
DIC*[f]* *ac,device*



Places the contents of the C input buffer of the specified controller in the specified accumulator. The previous contents of the accumulator are lost. After the data transfer, performs the function specified by *f*.

Data Out A

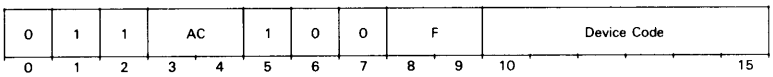
DOA*[f]* *ac,device*



Places the contents of the specified accumulator into the A output buffer of the specified controller. After the data transfer, performs the function specified by *f*. The number of bits loaded into the buffer depends on the controller. The contents of the specified accumulator remain unchanged.

Data Out B

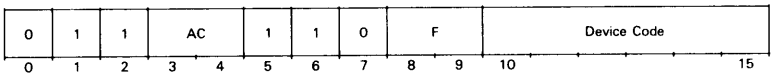
DOB*[f]* *ac,device*



Places the contents of the specified accumulator into the B output buffer of the specified controller. After the data transfer, performs the function specified by *f*. The number of bits loaded into the buffer depends on the controller. The contents of the specified accumulator remain unchanged.

Data Out C

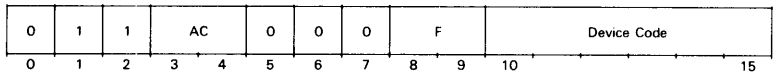
DOC*[f]* *ac,device*



Places the contents of the specified accumulator into the C output buffer of the specified controller. After the data transfer, performs the function specified by *f*. The number of bits loaded into the buffer depends on the controller. The contents of the specified accumulator remain unchanged.

No I/O Transfer

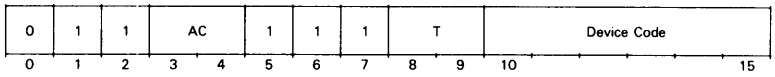
NIO*[f]* *ac,device*



Performs the function specified by *f*. When the assembler encounters the mnemonic NIO, it sets the AC field bits to 00. However, these bits are ignored and may have any value. The contents of all the accumulators are unchanged.

I/O Skip

SKP*[f]* *ac,device*



Skips the next sequential instruction if the test condition specified by *t* is true for the specified controller. When the assembler encounters the mnemonic SKP*[t]*, it sets the AC field bits to 00. However, these bits are ignored and may have any value. The contents of all of the accumulators and the Busy and Done flags for the specified device remain unchanged.

Program Interrupt Facility

When a peripheral completes an operation, the controller sets its Done flag to 1 to indicate that program service is required. The program can test the state of the Done flag repeatedly with *I/O Skip* instructions to determine when this occurs. However, continual interrogation of the Done flag by the program is generally wasteful of computing time; this is especially important when flag checks have to be done frequently in order to ensure that service is not delayed for so long that the peripheral loses data. A peripheral uses the program interrupt facility to conveniently notify the processor that service is required.

All peripherals that use the program interrupt facility have access to a single direct line to the processor, the *interrupt request line*, along which their requests for service are communicated. An interrupt request can be generated by a peripheral when the peripheral's Done flag is set to 1. The processor can respond to, or *honor*, an interrupt request by interrupting the normal flow of program execution and transferring control to an interrupt handling routine. By manipulating a number of flags distributed among the processor and the peripherals, the programmer can control which peripherals may request interrupts and when the processor may start an interrupt.

Program Interrupt Operations

This subsection details the operation of the program interrupt facility instructions that control the program interrupt scheme.

Control Flags

The operation of the program interrupt facility is governed by the Interrupt On flag (ION) in the central processor and by the Done and Interrupt Disable flags in each peripheral using the facility. By manipulating these flags, the program can choose to disregard interrupt requests altogether, or it can selectively ignore certain peripherals.

The major control flag for the program interrupt facility is the Interrupt On flag. To enable the interrupt facility, the program sets ION to 1, allowing the processor to respond to interrupt requests transmitted to it along the interrupt request line. Setting ION to 0 disables the entire interrupt facility, causing the processor to ignore all interrupt requests.

ION is manipulated by the program exactly like a Busy flag for the central processor. A Start command in any I/O instruction directed to the CPU (device code 77_8) sets ION to 1 following the next instruction. A Clear command in such an instruction sets ION to 0. ION is also set to 0 when the CPU honors an interrupt request.

The controller for each peripheral using the program interrupt facility contains an Interrupt Disable flag, which allows the program to disable interrupts from that peripheral. When a peripheral's Interrupt Disable flag is set to 1, the peripheral is prevented from making an interrupt request.

The Interrupt Disable flags of all peripherals are manipulated at once with a single CPU I/O instruction, called *Mask Out* (MSKO). This instruction sets up the Interrupt Disable flags of all peripherals connected to the program interrupt facility according to a mask contained in the accumulator specified by the instruction. Each peripheral is assigned by its hardware to a bit position in the mask; since there are 16 bit positions, there are also 16 interrupt levels. (See Appendix C for mask bit assignments for standard peripherals.) When a *Mask Out* instruction is issued, each peripheral's Interrupt Disable flag is set to the value of the assigned bit of the mask. When an *I/O Reset* instruction (IORST) is issued, all Interrupt Disable flags are set to 0.

Interrupt Requests

A peripheral's interrupt requests are governed by its Done and Interrupt Disable flags. When a peripheral completes an operation, it sets its Done flag to 1, which initiates a program interrupt request. If its Interrupt Disable flag is 0, the request is communicated to the CPU. If the ION flag is 1, the processor will honor the interrupt request as soon as it can. If the Interrupt Disable flag is 1, the request is not communicated to the CPU; it is blocked until the Interrupt Disable flag is set back to 0.

The processor interrupts the sequential flow of program instructions if all of the following conditions hold.

1. The processor has just completed an instruction or a data channel transfer occurring between two instructions.
2. At least one peripheral is requesting an interrupt.
3. Interrupts are enabled; i.e., ION is 1.
4. No peripheral is waiting for a data channel transfer; i.e., there are no outstanding data channel requests. (The data channel has priority over program interrupts.)

When the processor finishes an instruction, it takes care of all data channel requests (including those initiated during data channel transfers) before it services any pending interrupt. When no more peripherals are waiting for data channel transfers, the processor begins an interrupt if ION is 1 and at least one peripheral is requesting an interrupt.

The processor starts an interrupt by automatically executing the following sequence:

1. It sets ION to 0 so that no further interrupts may be started.
2. It stores the contents of the program counter (which point to the next instruction in the interrupted program) in location 0, so that a return to the interrupted program can be made after the interrupt service routine has finished.
3. It simulates a JMP@1 instruction to transfer control to the interrupt service routine. Location 1 should contain the address of the routine or the first part of an indirect address chain that points to the routine.

Servicing An Interrupt

The interrupt service routine (or handler) should save the state of the processor, identify which peripheral requires service, and service the peripheral.

Saving the state of the processor involves saving both the contents of any accumulators that will be used in the interrupt service routine and the carry bit if it will be used. The state of the processor must be saved so that it may be restored before the interrupted program is allowed to resume.

There are two ways in which the interrupt handler can identify which peripheral requires service.

1. The interrupt handler can execute a polling routine. This routine is merely a sequence of I/O SKIP instructions, which test the states of the Done flags of all peripherals in use. With this method, peripheral priorities are determined by the order in which the tests are performed. Note that the polling technique disregards the state of the Interrupt Disable flags. Masked out peripherals will be recognized if their Done flags are 1, even though these devices could not have caused the interrupt.
2. The interrupt handler can issue an *Interrupt Acknowledge* instruction (INTA). This instruction reads into a specified accumulator the device code of the first peripheral on the I/O bus that is requesting an interrupt. Note that with this method the Interrupt Disable flags are significant. Masked out peripherals cannot request an interrupt and, therefore, cannot respond to the *Interrupt Acknowledge* instruction.

After determining which peripheral requires service, the interrupt handler generally transfers control to a peripheral service routine. This routine performs the information transfer to or from that peripheral (if required) and either starts the peripheral on a new operation or idles the peripheral if no more operations are to be performed at this time.

When all service for the peripheral has been completed, either the peripheral service routine or the main interrupt handler should perform the following sequence to dismiss the interrupt.

1. Set the peripheral's Done flag to 0 (by starting a new operation or clearing the device) to dismiss the interrupt request which was just honored. If this is not done, the undismissed interrupt request will cause another interrupt — this time incorrectly — as soon as the interrupt handler finishes and attempts to return control to the interrupted program.
2. Restore the pre-interrupt states of the accumulators and the carry bit.

3. Set ION to 1 to enable interrupts again.
4. Jump back to the interrupted program (usually in a JMP@0 instruction).

The instruction that enables interrupts (usually INTEN) sets the Interrupt On flag to 1, but the processor does not allow the state of the ION flag to change to 1 until the next instruction begins. Thus, after the instruction that turns interrupts back on, the processor always executes one more instruction (assumed to be the return to the interrupted program) before another interrupt can start. The program must give this final return instruction immediately after enabling interrupts; this ensures that no waiting interrupt can overwrite the contents of location 0 before they are used to return control to the interrupted program.

Figure 2-2 shows how normal program flow is altered during a program interrupt. The interrupt handler is shaded to indicate that this block of instructions is not interruptible since the processor sets the ION flag to 0 to disable further interrupts when the interrupt occurs. Interrupts are not enabled again until the interrupt handler executes its *Interrupt Enable* instruction just prior to returning control to the interrupted program.

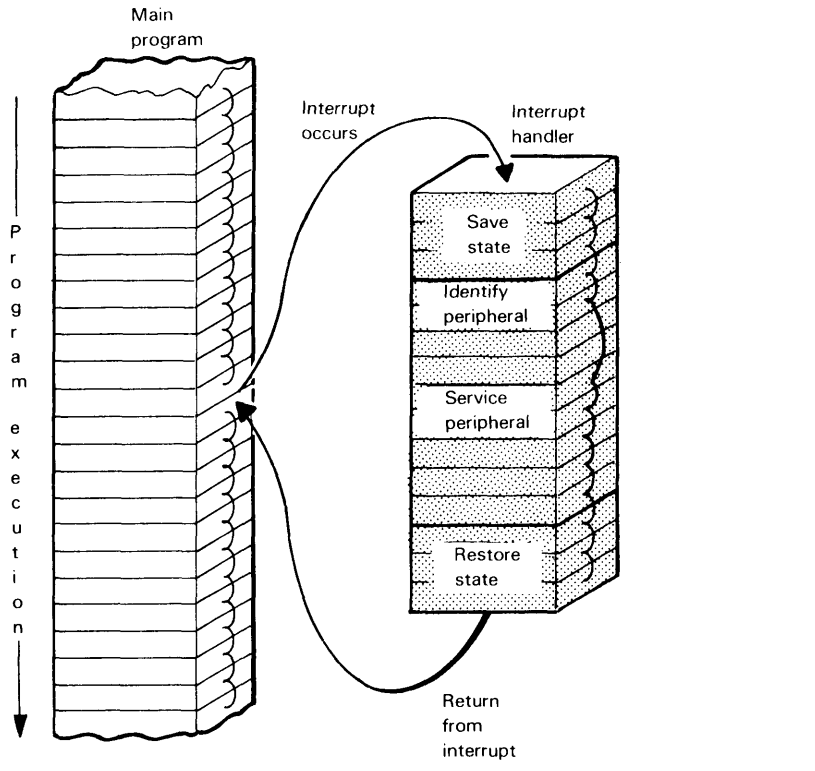


Figure 2-2 Program flow with interrupt

Program Interrupt Instructions

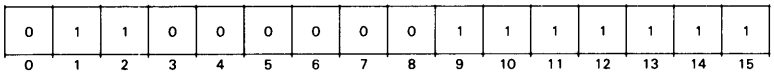
The instructions controlling the program interrupt facility use special device code 77_8 (mnemonic CPU). When this device code is used on a Desktop Generation computer, bits 8 and 9 of the skip instruction test the state of ION and PWR FF (Power fail); in the other instructions these bits turn interrupts on or off by setting ION to 1 (Start command) or 0 (Clear command).

The assembler recognizes a number of "shorthand" mnemonics for instructions that control program interrupts. In the instructions listed below, these mnemonics follow the instruction name. A standard mnemonic, recognized by the assembler as equivalent to the shorthand version, is listed below that.

There is one important difference between the standard mnemonic and the shorthand version. In the latter case, mnemonics for enabling and disabling interrupts (setting the Interrupt On flag) cannot be appended. For example, either MSKO 2 or DOB 2,CPU will set the Interrupt Disable flags according to the mask contained in AC2. To set the Interrupt On flag to 0 at the same time requires using the mnemonic DOBC 2,CPU.

Interrupt Enable (INTEN)

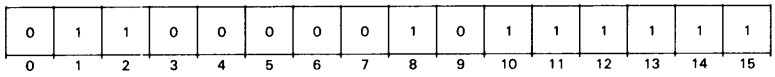
NIOS 0,*CPU*



Sets the Interrupt On flag to 1 to allow the processor to respond to interrupt requests. When the Interrupt On flag changes state (from 0 to 1), the processor executes one more instruction before it can start an interrupt. The assembler recognizes the mnemonic INTEN as equivalent to NIOS CPU.

Interrupt Disable (INTDS)

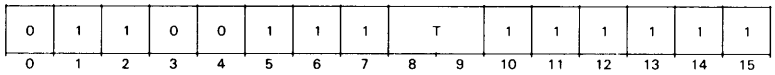
NIOC 0,*CPU*



Sets the Interrupt On flag to 0 to prevent the processor from responding to interrupt requests. The assembler recognizes the mnemonic INTDS as equivalent to NIOC CPU.

CPU Skip

SKP*[t]* *CPU*



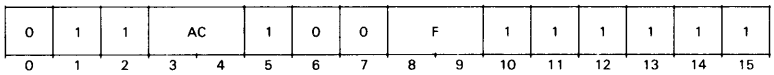
Skips the next sequential word if the test condition specified by *t* is true. The test performed is based upon the value of bits 8 and 9, which can be set by appending an optional mnemonic to the CPU Skip mnemonic. Table 2-4 shows the mnemonics and the tests.

Table 2-4 Test results

Mnemonic	T Field	Operation
BN	00	Tests for Interrupt On = 1.
BZ	01	Tests for Interrupt On = 0.
DN	10	Tests for Power Fail = 1.
DZ	11	Tests for Power Fail = 0.

Mask Out (MSKO)

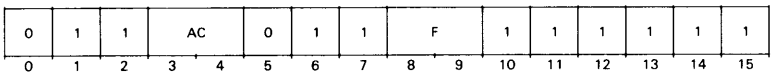
DOB*[f]* *ac*,CPU



Sets the Interrupt Disable flags in all the peripherals according to the mask contained in the specified accumulator. (A 1 in the mask bit sets the flags in all the peripherals assigned to that bit to 1; a 0 sets them to 0.) Then sets the Interrupt On flag according to the function specified by *f*. The contents of the specified accumulator remain unchanged. (Appendix C contains the mask bit assignments for standard peripherals.) The assembler recognizes the instruction MSKO *ac* as equivalent to DOB *ac*,CPU.

Interrupt Acknowledge (INTA)

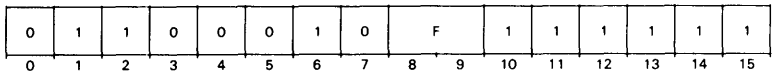
DIB*[f]* *ac*,CPU



Places in bits 10 to 15 of the specified accumulator the device code of that peripheral requesting an interrupt which is closest to the processor along the I/O bus. Sets bits 0-9 to 0. After the data transfer, sets the Interrupt On flag according to the function specified by *f*. If no peripheral is requesting an interrupt, sets the specified accumulator to 0. The assembler recognizes the instruction INTA *ac* as equivalent to DIB *ac*,CPU.

I/O Reset (IORST)

DOA*[f]* *0*,CPU



Resets all the peripherals connected to the I/O bus; sets their Busy, Done, and Interrupt Disable flags to 0; and, depending on the peripheral, performs any other required initialization. Then sets the Interrupt On flag according to the function specified by *f*.

The assembler recognizes the mnemonic IORST as equivalent to DOAC 0,CPU — that is, as the instruction defined here with *f* set to 10.

At power up the processor performs the equivalent of an IORST instruction.

Priority Interrupts

If the Interrupt On flag remains 0 throughout the interrupt service routine, the routine cannot be interrupted and there is only one level of peripheral priority. All peripherals that have not been disabled by the program are, for the most part, equally able to request interrupts and receive interrupt service.

Only when two or more peripherals are requesting an interrupt at exactly the same time is a priority distinction made. When this happens, priority is determined either by the order in which *I/O Skip* instructions are given or, if the *Interrupt Acknowledge* instruction is used, by the order of peripherals along the I/O bus. In a system with peripherals of widely differing speeds and/or service requirements, a more extensive priority structure may be required. The program interrupt facility hardware and instructions allow the program to implement up to 16 interrupt priority levels.

For example, suppose that a paper tape reader and a data terminal are being operated at the same time. While the tape is being read, an interrupt is requested as each new frame of data is available; the program must read this data within 430 microseconds, typically, before it is overwritten in the data buffer by the data from the next frames.

If the terminal service routine takes 300 microseconds, tape reader service will never be delayed longer than this and a single-level program interrupt scheme will suffice. However, this interrupt scheme will not work if the terminal service routine takes 600 microseconds, since a paper tape reader interrupt request that is initiated soon after terminal service begins will not be honored in time and a frame of data will be lost. In order to avoid losing data, the program interrupt scheme used must allow the tape reader to interrupt the lengthy terminal service routine. This involves creating a 2-level priority structure and assigning the paper tape reader to the higher priority level.

In general, a multiple-level priority interrupt scheme is used to allow higher-priority peripherals to interrupt the service routines of lower-priority peripherals. A hierarchy of priority levels can be established through program manipulation of the Interrupt Disable flags of all peripherals in the system.

When the interrupt request from a peripheral of a certain priority is honored, the interrupt handler sets up the new priority level. The interrupt handler establishes new Interrupt Disable flag values for all peripherals according to an appropriate *interrupt priority mask* used with the *Mask Out* instruction. Peripherals whose Interrupt Disable flags are set to 1 by the corresponding bit of this priority mask are *masked out*, or disabled. They are thereby regarded as being serviced. Before proceeding with the peripheral service routine, the Interrupt On flag is set to 1 so that the higher-priority peripherals may interrupt the current service routine.

Interrupt Priority Mask

The bit of the priority mask that governs the Interrupt Disable flag for a given peripheral is assigned to that peripheral by the hardware. The program cannot change that bit. Although lower-speed devices are generally assigned to higher-numbered mask bits, no implicit priority ordering is intended. The manner in which these priority levels are ordered is completely up to the programmer.

By means of the priority mask, the program can establish any desired priority structure, with one limitation: in the cases in which two or more peripherals are assigned to the same bit of the priority mask, these peripherals are constrained to be at the same priority level. When a peripheral causes an interrupt, a decision must be made whether to place at a higher or lower priority level all other peripherals that share the same mask bit with the interrupting peripheral. If a decision is made to mask out all peripherals which share that priority mask bit, the interrupting peripheral is also masked out.

Priority Interrupt Handler

A priority interrupt handler differs from a single-level interrupt handler in several ways. The handler must be *re-entrant*. This means that if a peripheral service routine is interrupted by another, higher-priority peripheral, no information required by the handler to restore the state of the machine is lost. The two items of information which should be saved, in addition to those saved by a single-level interrupt handler, are the return address (the contents of location 0) and the current priority mask. This information must be stored in different locations each time the interrupt handler is entered at a higher level. Doing this ensures that the necessary return information belonging to an earlier interrupt is not overwritten by a higher-level interrupt. A common method of storing return information for a re-entrant interrupt handler is through the use of a push-down stack.

The interrupt handler (including the peripheral service routines) for a multilevel priority scheme should perform the following tasks:

1. Save the state of the processor, i.e., the contents of the accumulators, the carry bit, location 0, and the current priority mask on the stack.
2. Identify the peripheral that requested the interrupt.
3. Transfer control to the service routine for that peripheral.
4. Establish the new priority mask with a *Mask Out* instruction for that peripheral's service routine and store it in memory at the location reserved for the current priority mask for that level of interrupt.
5. Enable interrupts. Now, any peripheral not masked out can interrupt this service routine.
6. Service the peripheral that requested the interrupt.

7. Disable interrupts in preparation for dismissal of this interrupt level, so that no interrupts will occur during the transition to the next lower level.

8. Restore the state of the processor, including the former contents of the accumulators and the carry bit, and reinstitute the pre-interrupt priority mask with a *Mask Out* instruction.

9. Enable interrupts.

10. Transfer control to the return address which was saved from location 0.

Figure 2-3 shows a simplified representation of program flow in a priority interrupt environment. Shaded areas indicate noninterruptible sections of instructions. Additional higher-priority interrupts could increase the depth of interrupts still further.

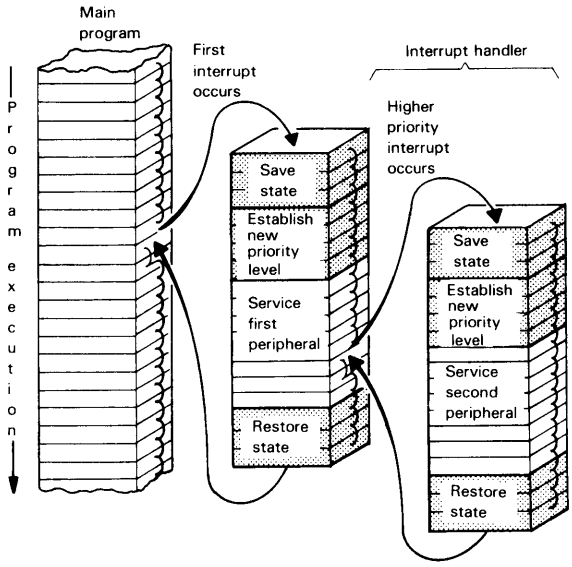


Figure 2-3 Program flow with priority interrupts

Data Channel Facility

Peripherals that need to transfer large blocks of data quickly generally accomplish their data transfers via the data channel facility. The actual data channel transfers do not disturb the state of the processor, since the data are transferred between registers in the controller and memory.

The term *direct memory access* (DMA) is often used to describe the way the data channel addresses memory. The amount of program overhead in the form of executing I/O instructions and loading or storing data is greatly reduced. The time required for program execution is lengthened, however, since the processor pauses, as soon as it is able, each time a word is to be transferred. The transfer then occurs and the processor continues. The program need only set up the peripheral for the transfer and can then perform other, unrelated tasks.

The data channel allows many peripherals to be active at the same time, providing individual controllers with access to memory on demand. Peripherals using the data channel operate under a priority structure imposed on them by the channel. In cases where more than one controller requests access to the data channel at the same time, priority is given to the controller that is closest to the processor on the I/O bus.

Controller Structure

Understanding the operation of the data channel requires a knowledge of the structure of the controllers that use it. The controllers usually contain the normal Busy and Done flags; status, control, and data registers; and the program interrupt components. Additional components are added to handle the functions necessary to operate the data channel. Some of these components, generally available to the program, are in the form of additional control and status registers.

Two registers usually added are a *word counter* and a *memory address counter.* The word counter is used by the program to specify the size of the data block to be transferred (number of words). The memory address counter is used to specify the address in memory which is used in the data transfer.

Word Counter

The word counter is loaded by the program with the two's complement of the number of words in the block. Each time a word is transferred, the controller automatically increments the counter by 1. When the counter overflows, the controller terminates data channel transfers.

The size of the word counter is usually 16 bits. The block size specified is both program and device dependent. Although the word counter specifies the negative of the desired block size, the most significant bit of the register need not be a 1; it is not a sign bit for the number. No sign bit is necessary, because the word count is treated as negative by the controller, by virtue of being incremented instead of decremented. Thus, a word count of 0 is valid; in fact, it specifies the largest possible block size. Table 2-5 further illustrates the correspondence between the desired word count and the value which must be loaded into a 16-bit or 12-bit word counter.

Table 2-5 Values for word counter

Negative Word Count (Decimal)	16-Bit Value (Octal)	12-Bit Value (Octal)
−1	177777	7777
−2	177776	7776
−8	177770	7770
−100	177634	7634
−2047	174001	4001
−2048	174000	4000
−2049	173777	3777
−4095	170001	0001
−4096	170000	0000
−4097	167777	
−8192	160000	
−32768	100000	
−65535	000001	
−65536	000000	

Address Register

The address register always contains the memory address to be used by the controller for the next data transfer. It is loaded by the program with the address of the first word in the block to be transferred. During each transfer, the controller increments the memory address counter by 1. Therefore, successive transfers will be from consecutive memory locations.

This represents the most common use of data channel transfers, but the addressing of memory does not have to be sequential. The address can be stepped in either direction by some fixed number. For example, one word of every page of memory could be addressed without touching any other part of that page.

Transfer Sequence

The actual data channel transfer sequence is a two-way communication between processor and controller. It proceeds as follows. When a peripheral has a word of data ready to be transferred to memory or wants to receive a word from memory, it issues a data channel request to the processor. The processor pauses as soon as it can and begins the data channel cycle by acknowledging the peripheral's data channel request. The acknowledgment signal dismisses the peripheral's data channel request. In addition, it causes the peripheral to send back to the processor the address of the memory location involved in the transfer, together with the direction of the transfer, i.e., to or from memory. Following the receipt of the address, the data are transferred in the appropriate direction.

Upon completion of each data transfer, the processor/controller interaction is over. The controller carries out any tasks necessary to complete the data transfer, such as transferring the data to the device itself for an output operation. The processor starts another data channel transfer if any data channel requests are pending, starts a program interrupt if one is being requested and there are no data channel requests, or resumes program execution.

The controller increments both the memory address counter and the word counter during the transfer. If the word count becomes 0, the controller terminates further transfers, sets the Busy flag to 0 and the Done flag to 1, and initiates a program interrupt request. If the word counter has not yet overflowed, the peripheral continues its operation, issuing another data channel request when it is ready for the next transfer.

Processor Pauses

The processor can pause for a data channel transfer only at certain, well-defined times. For Desktop Generation computers, data channel transfers can occur between noninterruptible instructions and during interruptible instructions.

Priorities

Program execution has priority over the data channel, except at certain points in the processor's operation when the data channel has priority (not only over normal program execution but also over any pending program interrupt requests). At these points, the processor handles all existing data channel requests (including those generated while data channel transfers are in progress), before starting a program interrupt or resuming normal instruction execution.

Thus, if data channel requests are being generated by a number of peripherals as fast as or faster than the processor can handle them, all processing time will be spent handling data channel transfers. However, when the data channel is being used at less than its maximum rate, processing time is shared between the data channel, which receives as much as it needs, and the program, which uses the rest.

Programming a Peripheral

Programming a peripheral for a data channel block transfer typically involves the following steps.

1. Check the peripheral's status, usually by testing the Busy flag and/or reading a status word and checking one or more error or ready bits. If an error has occurred, the program should take appropriate action. If no error has occurred but the peripheral is not yet ready, the program should wait for the peripheral to complete its operation. When the peripheral is ready, the program may proceed.
2. Locate the data block in the device. This usually involves giving a peripheral "address" by specifying a unit number, channel number, sector number, or the like.
3. Locate the data block in memory by loading the memory address counter with the address of the first word of the block.
4. Specify the size of the data block by loading the proper value into the word counter.
5. Specify the type of transfer and initiate the operation. If the peripheral is capable of several different operations, specifying the type of transfer usually involves loading a control register in the controller. The operation itself is usually initiated by one of the I/O control commands (Start or Pulse).

Setting up and initiating the data channel operation is the major part of programming a data channel block transfer. However, the program should check for errors when the operation is complete and take appropriate action if any errors occurred.

Timing

On systems which depend heavily on input/output, both the direct program control and data channel facilities can be overloaded. This means that certain peripherals may lose data or perform poorly, since the system cannot respond to them in time.

This section explains how a system can be overloaded and what steps can be taken to minimize the detrimental effects.

Direct Program Control

Nearly all peripherals operating under direct program control request program service by setting their Done flags to 1. Whether the CPU determines that the Done flag is set to 1 by repeatedly checking it or by responding to interrupt requests, there may be a significant delay between the time when the peripheral requests program service and the time when the CPU carries out that service. This delay is called programmed I/O latency.

When the program interrupt facility is not used, programmed I/O latency consists of two intervals:

1. The time interval between the peripheral's setting the Done flag to 1 and the CPU's checking the flag.
2. The time required by the peripheral service routine to transfer data to/from the peripheral and set the Done flag to 0 (by idling the peripheral or instructing it to begin a new operation).

The first interval can be diminished by performing frequent checks on the Done flag; the second can be diminished by writing an efficient peripheral service routine.

When the program interrupt facility is used, the programmed I/O latency consists of at least four intervals and possibly as many as seven:

1. The time between the setting of the Done flag to 1 and the end of the instruction being executed by the CPU.
2. The time the interrupt facility needs to store the program counter in location 0 and simulate a JMP @1 instruction.
3. The time required by the interrupt handler to identify the peripheral and transfer control to the service routine.
4. The time required by the service routine to transfer data to/from the peripheral and set the Done flag to 0.
5. The time during which CPU operation is suspended because data channel transfers are in progress (see the following section).
6. The time during which the CPU does not respond to the peripheral's interrupt request because the interrupt system is disabled (for example, during the servicing of an interrupt from another peripheral).
7. The time during which the peripheral's Interrupt Disable flag is set to 1 during the servicing of an interrupt of a higher priority peripheral.

The first interval is determined by the longest noninterruptible instruction that the CPU can execute.

The second interval also depends on the machine; in general, it is approximately two to three times as long as a memory reference. The third, fourth, sixth, and seventh intervals are determined by software and account for the bulk of the programmed I/O latency. The fifth interval is determined by the nature and number of the data channel devices operating in the system.

Maximum programmed I/O latency of the peripheral is the longest allowable delay between the time when a peripheral sets its Done flag to 1 and the time when the CPU transfers data to/from that peripheral and sets the Done flag to 0. When the actual programmed I/O latency for a peripheral exceeds the maximum programmed I/O latency, the specific effects depend on the device in question. In the worst case, data may be incorrectly read or written.

A peripheral service routine must usually perform certain computations (updating pointers to buffers, byte counters, etc.), but rarely are these computations so complex that they cannot be accomplished within the constraints of the maximum allowable programmed I/O latency. However, if several peripherals are compet-

ing for service at the same time, it may be necessary to jeopardize the performance of some of them by deferring their requests for program service until the CPU has serviced the higher-priority requests. For this reason, all Data General computers incorporate the priority interrupt facility described earlier.

The object of the priority interrupt facility is to minimize the loss of important data. In order for the programmer to achieve this end, the assignment of the software priority levels should be made in view of the following considerations:

1. The maximum allowable programmed I/O latency for each peripheral,
2. The result of exceeding the maximum allowable programmed I/O latency for each peripheral (slow down or data loss),
3. The cost of losing data.

Data Channel Control

Problems with time constraints may also be encountered when transferring data via the data channel. When a peripheral needs data channel service, it makes a data channel request. However, the CPU can allow data channel peripherals to access memory only at certain times. (At such times, it is said that data channel breaks are enabled.) In addition, there may be more than one peripheral waiting to access memory at any one time. Consequently, there may be a significant delay between the time when a peripheral requests access to memory and the time when the transfer actually occurs. This delay is called data channel latency and consists of the following intervals:

1. The time between the peripheral's request for memory access and the next data channel break,
2. The time required to complete data channel transfers to/from higher priority peripherals also requesting memory access.

The length of the first interval depends on the design of the CPU. The length of the second interval depends on the number of data channel peripherals operating in the system at a higher priority and the frequency of their use.

Most peripherals using the data channel control operate under fixed time constraints. Disk drives and magnetic tape transports are typical data channel peripherals. In each of these devices, a magnetic medium moves past a read or write head at constant velocity. If data are not read or written at the correct instant, the data will be transferred to or from the wrong place on the magnetic media. Consequently on input, such devices must be allowed to write a word into memory before the next word is assembled by the controller, and on output, the controller must be able to read a word from memory before the surface is positioned under the write head. In either case, if the data channel latency is too long, data cannot be properly transferred. Most peripherals operating under data channel control set an error flag when this happens, so the service routine can take appropriate action to recover from the error, if possible.

The maximum allowable data channel latency of a peripheral is the maximum time the peripheral can wait for a data channel transfer. The range of times is from tens of microseconds to several hundred microseconds. At the time the system is configured, data channel priorities should be assigned to peripherals on the basis of the following considerations:

1. The maximum allowable data channel latency of the peripheral. A peripheral with a short allowable latency usually should receive a higher priority than one with a long allowable latency.
2. The recovery time of a peripehral (i.e., how long before it can repeat a transfer that failed because of excessive data channel latency), if the peripheral can recover.
3. The cost of losing data from the peripheral if the peripheral cannot recover.

If data channel latency seems to be a problem in a system, latency might be improved by changing programs to make less frequent use of long instructions and lengthy indirect chains. In addition, there is an upper limit on the number of data channel transfers/second that a computer can support. In cases where this limit is exceeded, the only solution is to reduce the number of peripherals using the data channel at the same time.

A final consideration is that high data channel use reduces the speed of program execution since the processor pauses for each transfer. This may adversely affect the CPU's capacity to respond to interrupts and service those peripherals operating under direct program control.

Bus-to-Controller Interface 3

I/O controllers require unique interfaces to communicate with the device, or devices, under their control. However, they require a standard interface to communicate with their host CPU and its I/O bus.

This chapter describes the I/O bus to I/O controller interface that is standard to all peripherals connected to the microI/O bus of Desktop Generation computers.

The I/O Bus

The I/O Bus of Desktop Generation computers transfers information and commands between the CPU and the individual devices. As shown in Figure 3-1, the I/O bus is basically bidirectional with a few unidirectional control lines. The data path on the bus consists of two bidirectional lines. This high-speed, serial data path carries all data and commands for the I/O controllers.

The I/O bus contains 13 lines for data transfers, control, timing, and priority enforcement. These lines are:

- Two differential pairs of lines for data,
- Two differential pairs of lines for transfer synchronization.
- One general control line.
- Four protocol control lines.

The two differential data line pairs, BI/O DATA1 and BI/O DATA2, carry commands, addresses, and data for all I/O controllers. The BI/O DATA1 line moves the most significant byte (8 bits) of the word, plus a control bit; the BI/O DATA2 line moves the least significant byte and another control bit. The control bits define the type of information transferred.

The two differential pairs for transfer synchronization are BMCLOCK and BI/O CLOCK. The undirectional BMCLOCK line is the master system clock for all peripheral interfaces. The bidirectional BI/O CLOCK line is generated by the source of the transfer on the I/O bus, either the active IOC or the CPU. The BI/O CLOCK operates at the same frequency as the two data lines (one half the frequency of the BMCLOCK).

The control line CLEAR is used to reset all peripheral controllers on the I/O bus.

The four protocol control lines are INTR and INTP (for program interrupts), DCHR and DCHP (for data channel breaks). INTR and DCHR are the request lines; INTP and DCHP are the priority lines for the two types of service requests the controller can make to the CPU.

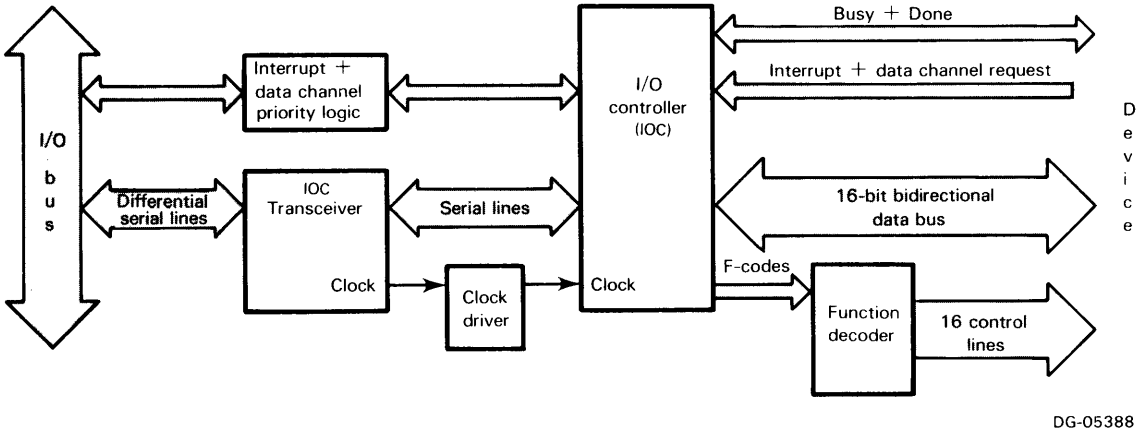


Figure 3-1 Input/output controller

The Transceiver

The I/O controller (IOC) transceiver is the first element of the peripheral controller as seen from the I/O bus. This device takes the data and timing lines of the I/O bus, sends to the controller the bidirectional I/O CLOCK and the data lines, and receives from the controller the I/O INPUT line.

The transceiver illustrates a major feature of peripheral interfacing for Desktop Generation computers; that is, the main conceptual blocks are well-defined large-scale integration (LSI) or medium-scale integration (MSI) chips readily available from Data General Corporation. The expertise required to build a peripheral interface does not have to go beyond understanding the interactions of the modules.

The data lines I/O DATA1 and I/O DATA2 are the principal lines between the transceiver and the I/O controller. These lines are latched versions of the BI/O data lines. Each data line carries one control bit plus one byte of information.

On transmission from the CPU, the IOC is first told which of four types of information it is going to receive: request enable, data channel address request, data, or I/O instruction. If the first bit of I/O DATA1 is 1, then the transfer has the short format and the value of the first bit of I/O DATA2 determines which of the first two types of transfers are being made. If the first bit of I/O DATA1 equals 0, then a full word is being transferred and the first bit of I/O DATA2 tells the IOC that either data or an instruction is being passed (see Table 3-1).

Table 3-1 Instruction/data codes

First Bit I/O DATA1	First Bit I/O DATA2	Information Type
1	1	Request enable
1	0	Data channel address request
0	1	Data
0	0	I/O instruction

Under programmed I/O, the first bit of the nine bits transferred on each data line is the I/O instruction code for the instruction being passed. The 2-bit code generated tells the controller the type of transfer being made, as well as the length of the word transferred. The remaining byte on each line makes up the instruction to be carried out by the IOC. The second word passed is the data; the first bit of each line gives the data code, followed by the actual byte of data for that line.

For data channel transfers, the data word remains the same, but the first word is the memory address. In the address transfer, the first bit of each line has the data code. The second bit of I/O DATA1 gives the direction of the transfer. Since only 15 bits are used for an address, this is still an 18 bit transfer.

I/O Controller and Decoder

The I/O controller (IOC), an LSI component, is a finite state machine, which handles the requirements of peripheral devices connected to the I/O bus of Desktop Generation computers. The IOC shown in Figure 3-2 represents the heart of any device controller for a peripheral in a Desktop Generation system.

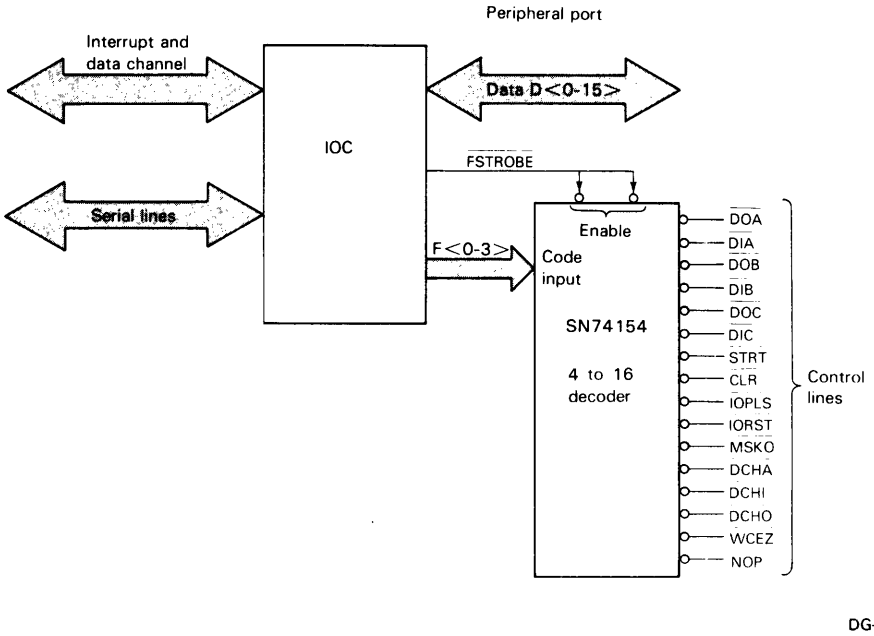


Figure 3-2 I/O controller and decoder

I/O Controller

The controller has three basic parts: a register file and bus structure, state control logic, and Busy/Done and request logic. The prinicpal register is the I/O shift register (IOSR). This register, connected to the two serial data lines and the I/O CLOCK line, provides the conversion for transfers via the I/O bus. The IOSR loads the instruction register (IR) for the state change logic.

Four of the other registers have specific functions during data channel transfers (see Table 3-2). The inverter/drivers buffer transfers between the IOC internal bus and the device registers. A clear distinction should be made here between IOC registers and device registers; the former are not accessible to the programmer. The data in or out commands refer only to device registers, if any, and not to IOC registers.

Table 3-2 IOC registers

Registers	Function
IOSR (I/O shift register)	Performs the serial/parallel conversion of information transferred on I/O DATA<1,2>.
Instruction register	Holds all I/O instructions received by the IOC.
Initialization register	Holds the device code, external register enable bit, external Busy/Done enable bit, and the polarity bit.
Address register	Holds the 15-bit memory address used during data channel transactions when internal registers are used.
Word count register	Holds the two's complement of the number of words remaining in a data channel block transfer when internal registers are used.
T register	Buffers the 15-bit memory address and the direction bit used during data channel transfers.

Two internal registers used during data channel transfers could be made external to the chip as two device registers. The address register and the word count register can be device registers B and C, respectively. To convert the internal registers into external registers, data lines D7 to D15 must be loaded into the initialization register of the IOC during power up. For details, see Table 3-3 and Figure 3-3; also see *microNOVA Integrated Circuits Data Manual.*

Table 3-3 Settings for internal/external registers

Data Pin(s)	Loads	Signal Level	Effect
D7	External Busy/Done enable bit	Low	External Busy/Done flags are enabled.
		High	Internal Busy/Done flags are enabled.
D8	External register enable bit	Low	External address and word count registers are enabled.
		High	Internal address and word count registers are enabled.
D9	Polarity bit	Low	Data on D<0-15> interpreted using positive logic (high = 1).
		High	Data on D<0-15> interpreted using negative logic (high = 0).
D<10-15>	Device code	Low and high	When loaded, the 6-bit device code on these lines is interpreted using negative logic (high = 0).

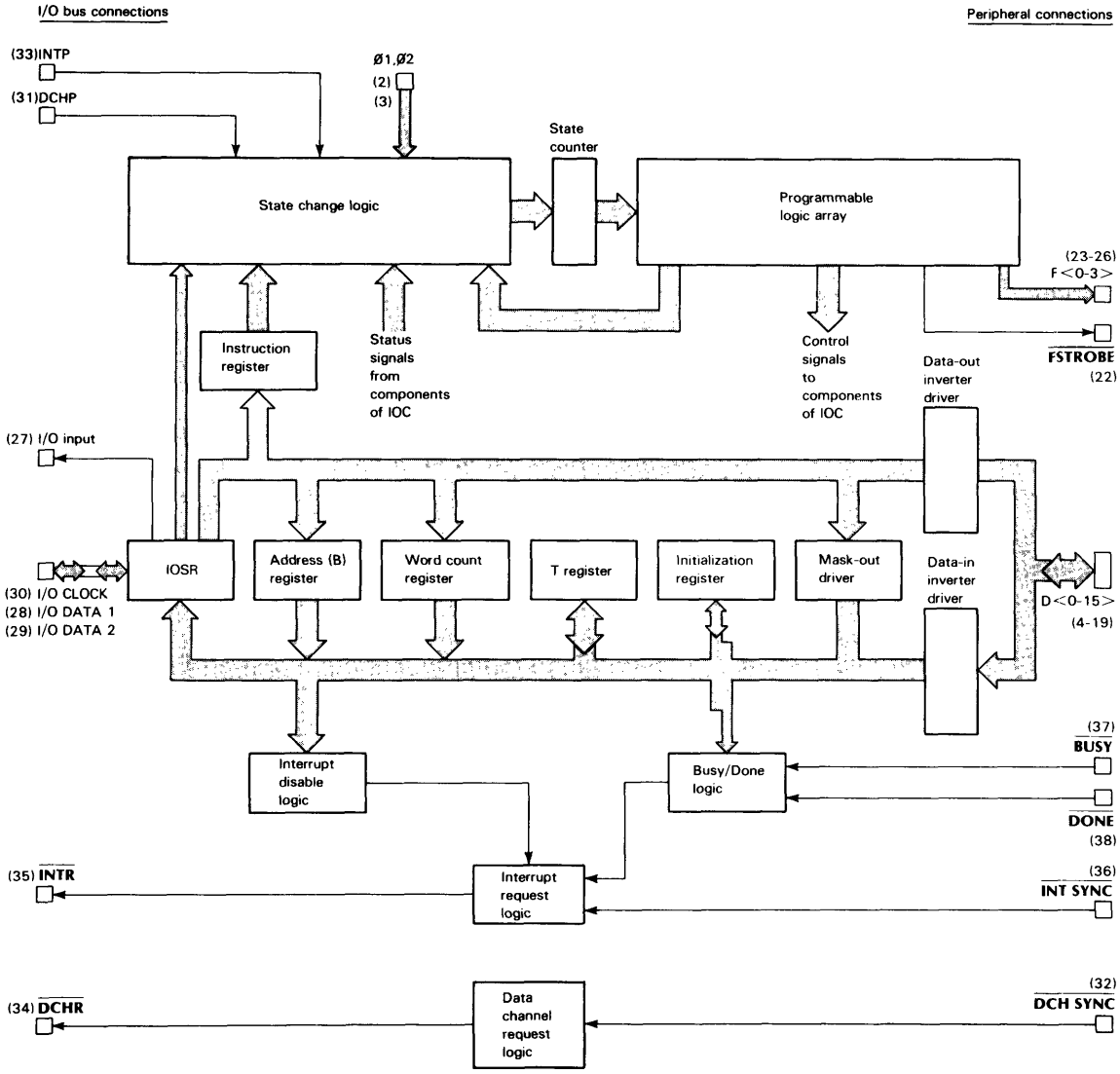


Figure 3-3 IOC internal structure

The IOC control logic contains three subunits: the state change logic, the state counter, and the programmable logic array (PLA). The state change logic takes input from the IOSR, the IR, the programmable logic array (PLA), and the interrupt enable lines; it takes status information from all parts of the IOC. The output of the state change logic is an address placed in the state counter. The counter is the address buffer for the PLA. The PLA feeds back to the state change logic, sends control information to the rest of the IOC (using part of the internal bus), and strobes the four control lines to the decoder for the device.

The last major piece of the IOC is the busy/done and request logic. The busy/done logic (Table 3-4) is connected to the device by two bidirectional lines that provide the active low forms of the signals. These signals are controlled by lines from the internal bus. The interrupt disable logic is also directly controllable from the bus. These two sets of logic combine with the interrupt synchronization line from the device to form the interrupt request logic for signal INTR(0). The data channel request logic does not use the bus and thus is not under direct user control.

Table 3-4 Busy/Done logic

Bits 8	Bits 9	Program Mnemonic	Effect
0	0	—	No effect.
0	1	S	Sets Busy to 1 and Done to 0.
1	0	C	Sets both Busy and Done to 0.
1	1	P	Sets Done to 0; has no effect on Busy.

The decoder module refers to any standard 4 to 16 decoder for the F(0-3) control lines. These lines are generated by the PLA along with a strobe signal to synchronize the decoder. With these lines, the peripheral device now sees the I/O bus as the standard parallel bus.

IOC Decoder to Device

The main advantage of the IOC becomes clear when you understand how to use it to reduce design time. The device has available 16-bit parallel bidirectional data lines and 16 control lines from the decoder. With these lines, the Busy/Done flags, and the interrupt lines, both types of data transfers can be made. The peripheral designer does not have to be concerned with any further details of the controller, it is used like any other standard module.

The advanced peripheral designer may work with devices requiring more than the normal three registers. At this point the decode logic would be part of the device rather than the interface.

The control lines to the device (see Table 3-5) are all active low. They represent the complete command set; what is available to the programmer is a subset.

Table 3-5 IOC to device commands

Function Code F<0-3>	Label	Function
0000	DOA	Loads the A register with data from IOC data lines D<0-15>.
0001	DIA	Gates data into the IOC from the A register via data lines D<0-15>.
0010	DOB	Loads the B register with data from IOC data lines D<0-15>.
0011	DIB	Gates data into the IOC from the B register via data lines D<0-15>. If internal data channel registers are being used, D<0-15> are ignored.
0100	DOC	Loads the C register with data from IOC data lines D<0-15>.
0101	DIC	Gates data into the IOC from the C register via data lines D<0-15>.
0110	STRT	Start I/O device.
0111	CLR	Clear I/O device.
1000	IOPLS	I/O pulse.
1001	IORST	Gates the device code, polarity, external register enable, and external Busy/Done enable bits into the IOC via the following data lines: D<0-15> = Device code D9 = Polarity D8 = External register enable D7 = External Busy/Done enable These data are always interpreted using negative logic (high = 0).
1010	MSKO	Gates the device maskout priority bit into the IOC via data lines D<0-15>.
1011	DCHA	Gates the data channel direction bit into the IOC via data line DO. If external registers are being used, gates the external address register into the IOC via data lines D<0-15>.
1100	DCHI	Gates the data from the peripheral data register onto the IOC's data lines (D<0-15>) during a Data Channel in transaction.
1101	DCHO	Loads the peripheral's data register with data from the IOC's data lines (D<0-15>) during a Data Channel Out transaction.
1110	WCEZ	Indicates that the internal data channel Word Count register has overflowed (equals zero). Denotes the end of a block data channel transfer.
1111	NOP	No function.

Timing

There are two types of programmed I/O transactions, depending on the level of I/O INPUT during the data transfer portion of the transaction. For data-out the I/O INPUT line remains high, while for data-in this line goes low before the data are transferred (see Figure 3-4).

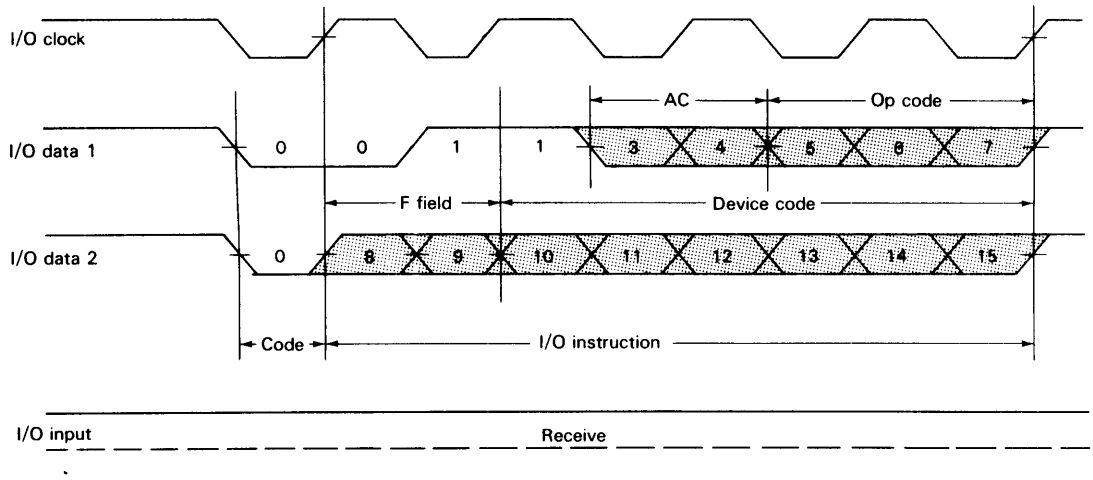


Figure 3-4 I/O instruction format

The format of the instruction word (Figure 3-5) transferred on each of the data lines simplifies decoding for the IOC. Device codes of 00_8 to 03_8 and 77_8 are *illegal device codes* for any Desktop Generation computer.

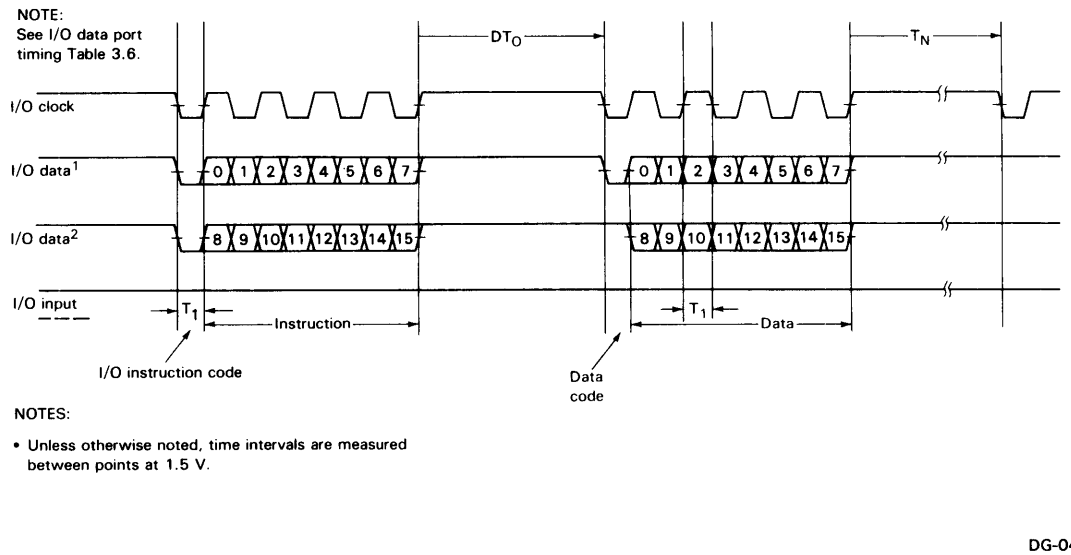


Figure 3-5 I/O instruction format (CPU to IOC)

Data-Out Transfers (CPU to IOC)

Data-out transfers move a 16-bit word of data from the CPU to one or all IOCs. This is a full-word transfer of 18 bits, with the first bit on each data line setting the code needed by the IOC. The transaction begins with an I/O instruction transfer, followed by a data transfer, both of which start from the CPU. The instruction transmitted is the I/O instruction executed by the CPU.

An IOC executing a data-out instruction must start receiving the data in the window defined by DT_0. T_n defines the minimum setting time between commands. The data and instruction codes are defined above (see Figure 3-5).

Data-In Transfers (IOC to CPU)

Data-in transfers move a 16-bit word from an IOC to the CPU. The interval between instruction transfers and the format of the instruction transferred are the same for the data-in transfers, but the data transfer interval is different.

The window that the CPU has to start receiving data, DT_1, is a longer interval. The I/O INPUT line must go low one-half clock cycle before the I/O CLOCK does in transferring the data. Additionally, it must stay low the same half cycle after the data are transferred (see Figure 3-6).

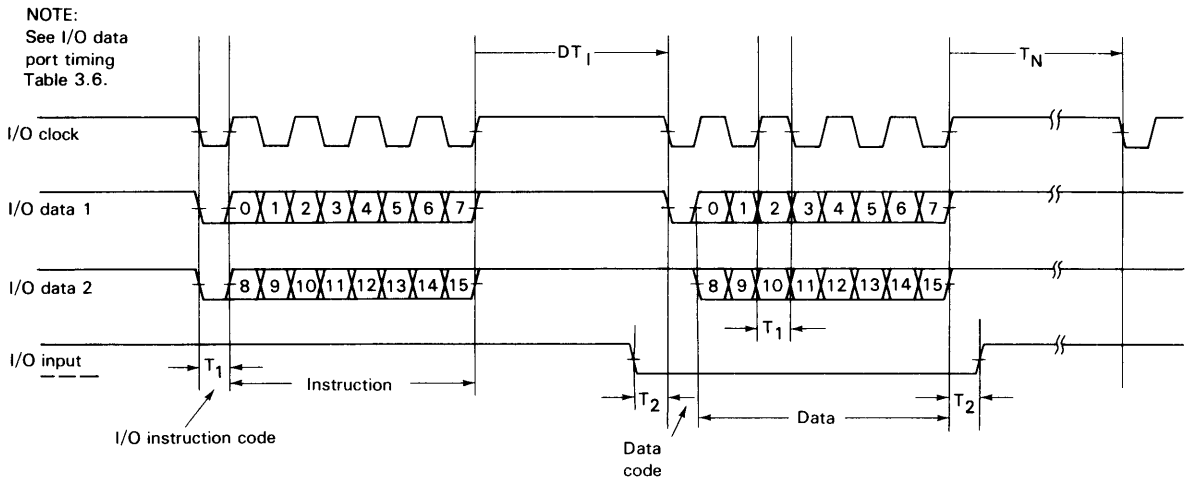


Figure 3-6 Programmed I/O transfers (IOC to CPU)

Data Channel Transfers

Normal data channel transfers first send the address of the memory location. Bit 0 of this address determines if the transfer is a memory read or write; 0 means memory read, 1 means memory write. In either case, the IOC generates the address to be accessed and the direction of transfer.

Data Channel Out Transfers (CPU to IOC)

A data channel out transfer moves a word of data from memory out to an IOC. The IOC must wait at least until the C interval has passed after the request has been made before the address and direction bit can be transferred. DC_0 defines the minimum time between the address and data transfers, as shown in Figure 3-7.

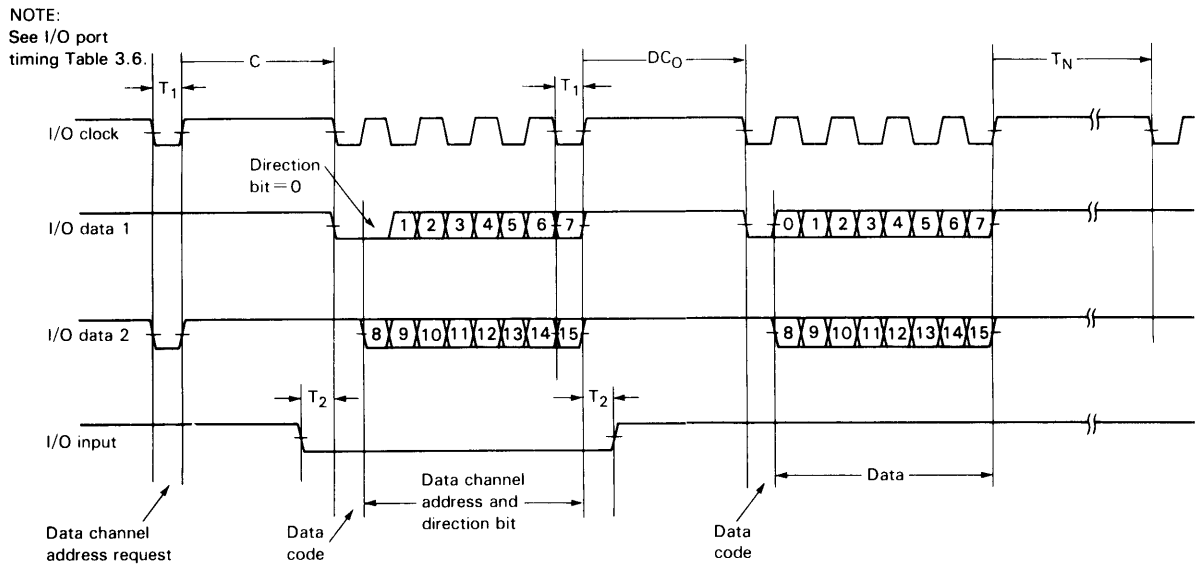
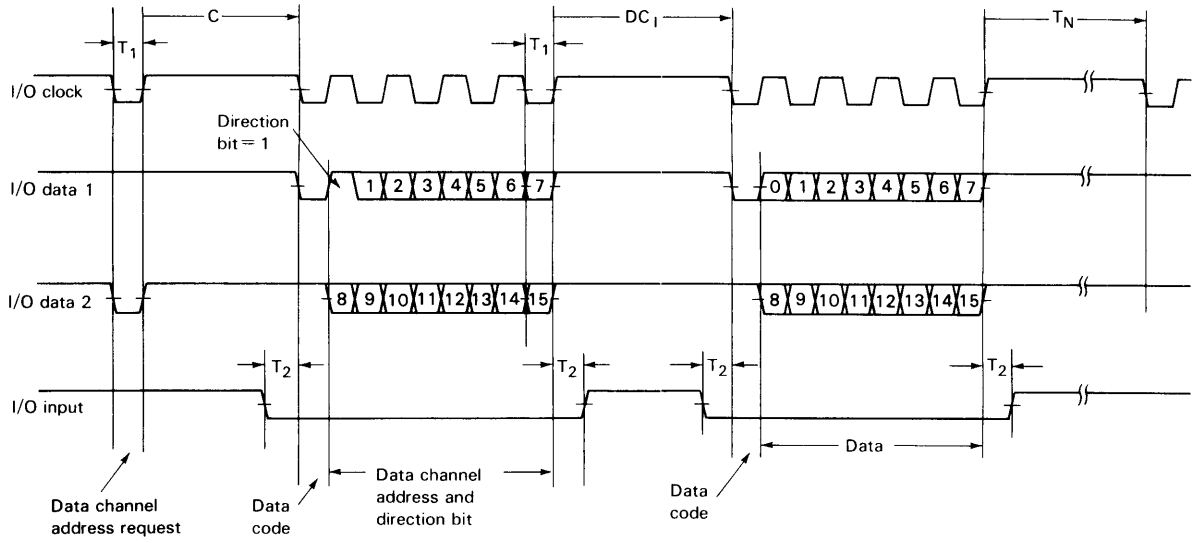


Figure 3-7 Data channel out

Data Channel In Transfers (IOC to CPU)

A data channel in transfer writes a word of data into memory from an IOC (see Figure 3-8). The DC_1 interval needed to wait before data are transmitted to memory is longer than the DC_0 time. Also the I/O INPUT line must go low one-half clock pulse before data are transferred.

NOTE:
See I/O data port timing Table 3.6.



NOTES:
- Unless otherwise noted, time intervals are measured between points at 1.5 V.

DG-04159

Figure 3-8 Data channel in

For specific information, see *microNOVA Integrated Circuits Data Manual.*

Peripheral Control Lines

Three pulses from the FSTROBE(0) line of the IOC set three critical time frames for the passing of control and data signals to the device. At time A, the four control lines to the decoder are active along with the data lines, if data are going to the device. At time B, the appropriate control line goes low and the device is active until time D. If data are going from the device into the CPU, the data lines become active at time C when the appropriate control line has become a stable low signal. The data in window closes, at time E, before the function codes become inactive at point F (see Figure 3-9).

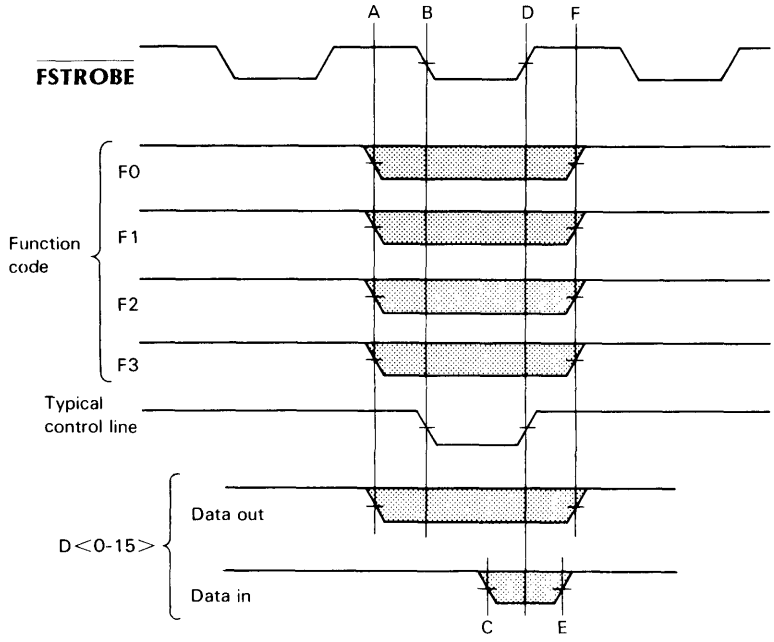


Figure 3-9 Device control signal timing

Table 3-6 I/O data port timing table

Class	Mnemonic	Min.*	Max.*	Units
All	T_1	110	130	ns
	T_2	110	130	ns
	T_{SKEW}	−5	+15	ns
	T_N(next command)	840	—	ns
I/O Instructions	DT_O (Data Out Transfer)	470	850	ns
	DT_I (Data In Transfer)	1190	1330	ns
Request Enable and Data Channel Address Request	A	360	540	ns
Data Channel Transactions	C (Data Channel Address Request to first transfer)	710	850	ns
	DC_O (Data Channel Out)	590	1930	ns
	DC_1 (Data Channel In)	1190	1210	ns

**All of the above times assume a MASTER CLOCK frequency of 8.333 MHz.*

General-Purpose Interface Card

4

In many applications, it is necessary to interface nonstandard equipment to the computer. If no I/O device controller is available for a particular piece of equipment, the customer must design his own. Much of the logic contained in an I/O device controller is the same for every I/O device. This common logic can be designed once, laid out on a printed circuit card, and then used by customers in the construction of their own I/O device controllers.

This chapter describes a general-purpose interface card included in Data General's assemblies. The card contains those logic elements common to most I/O device controllers, including an I/O controller (IOC) and I/O bus transceiver. Drilled holes or sockets and wire-wrap pins are provided for you to add those elements and interconnections needed to control particular equipment.

Principal Components

Data General's model 4210 general-purpose interface card is an IOC-based interfacing aid. The card contains an IOC module capable of providing a generalized front end to the Desktop Generation computer's I/O bus. The IOC module handles all Busy/Done/interrupt processing and data channel interfacing. Model 4210 has predrilled holes and etched conductors to assist in the building of an interface. Model 4210 with the model 4211 option has DIP sockets and wire-wrap pins installed in the predrilled holes.

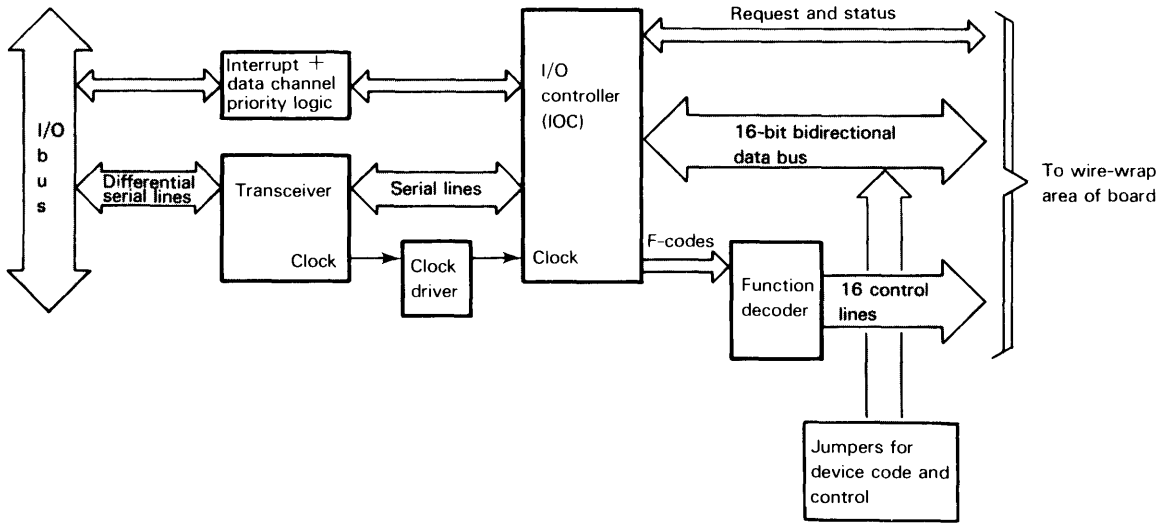
Table 4-1 summarizes the characteristics of a general-purpose interface card.

Table 4-1 Summary of characteristics

Operations	I/O command decoding, interrupt and data channel requests, and data channel transfers
Registers	15-bit data channel address 16-bit data channel word count Data channel request flag Interrupt request flag External register enable flag Polarity flag Device code register Busy flag Done flag
Busses	I/O bus same as CPU
Board dimensions	7.5 x 10.4 in (19 x 26 cm)
Maximum operating temperature	131°F (55°C)

Figure 4-1 shows the principal components plus data and control paths for the general-purpose interface (GPI). The GPI contains an IOC and its support circuits, a function decoder, and data buffer, sockets for device select, and IOC initialization jumpers. All data lines and control signals available for device use are brought to wire-wrap pins. The I/O transceiver, IOC, clock driver, and I/O control buffer connect the GPI to the I/O bus.

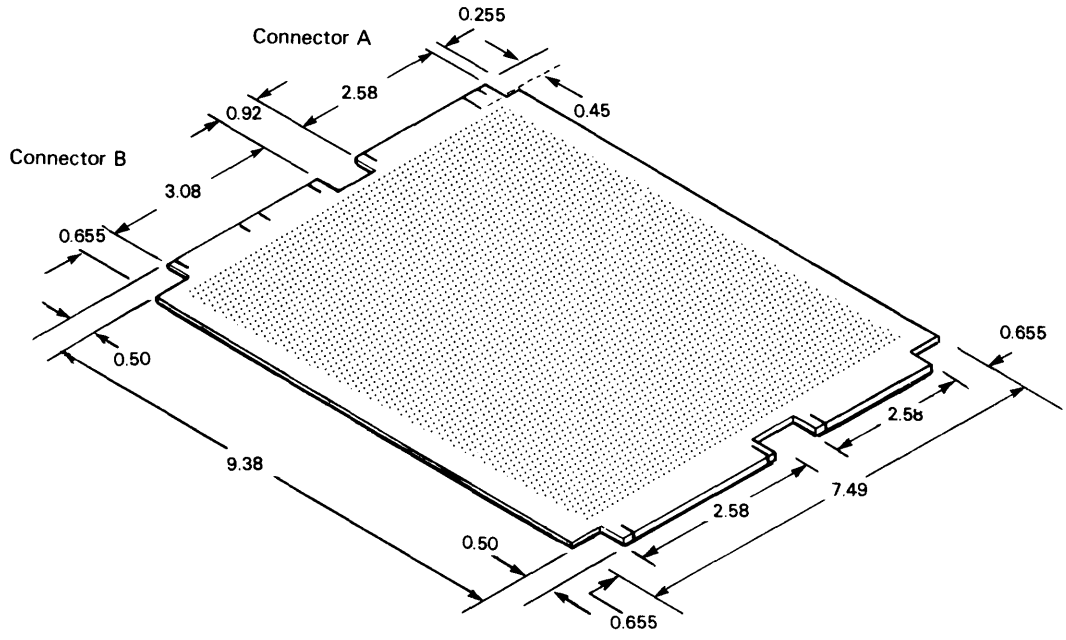
Sixteen bidirectional data lines (bit parallel) extend from the IOC in two groups. One group passes via a 16-bit buffer to wire-wrap pins and normally is used for output signals from the CPU to customer-designed circuits. The buffer provides TTL fanout capacity without overloading the IOC. The other group of data lines passes directly to wire-wrap pins and normally is used for input signals from the customer circuit to the CPU. The input lines are designed to be driven by low-leakage, open-collector, TTL drivers.



DG-05866

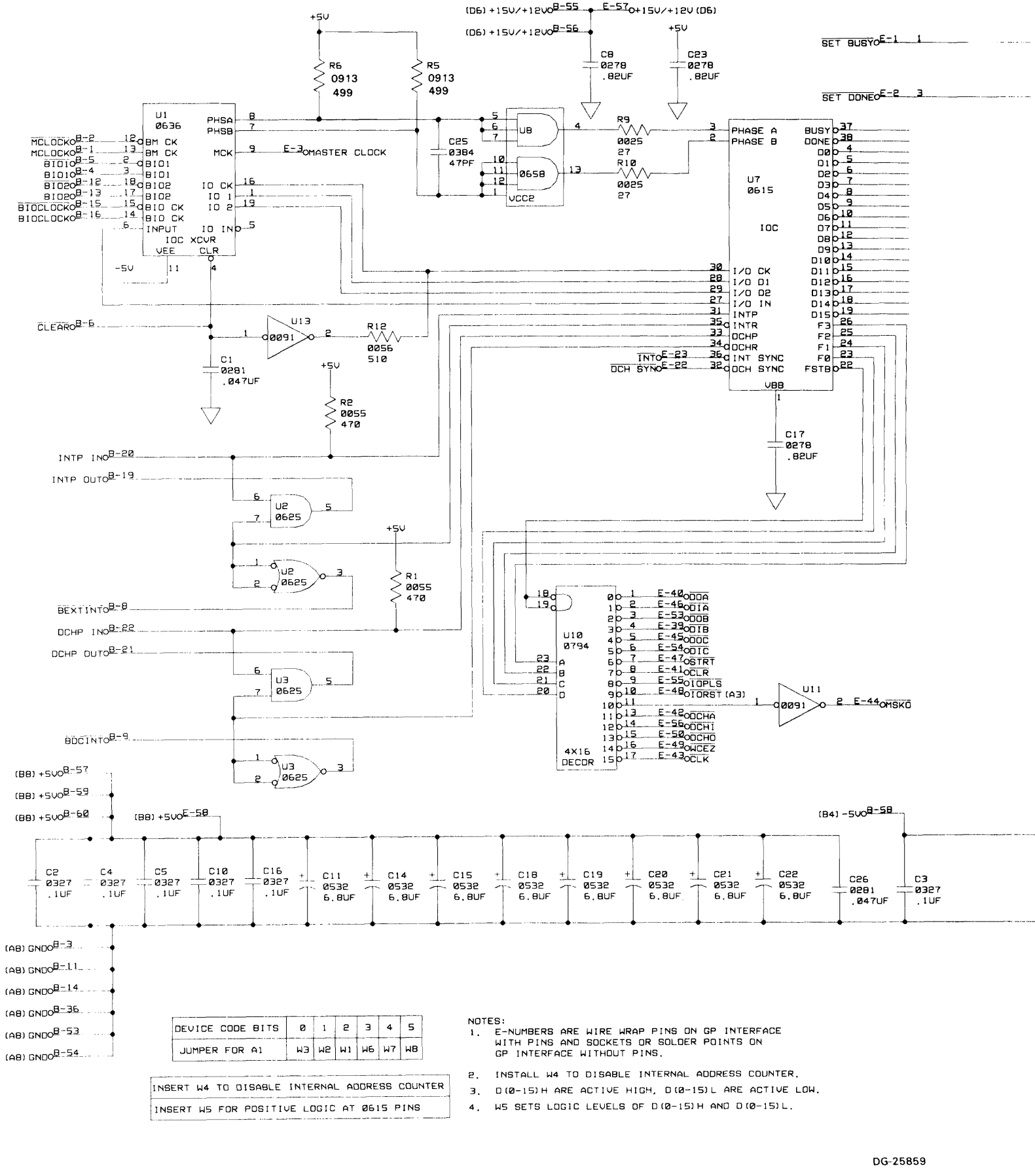
Figure 4-1 General-purpose interface card

Figure 4-2 illustrates dimensions of the GPI card. A logic diagram for the GPI appears in Figure 4-3.



DG-02420

Figure 4-2 General-purpose interface card dimensions



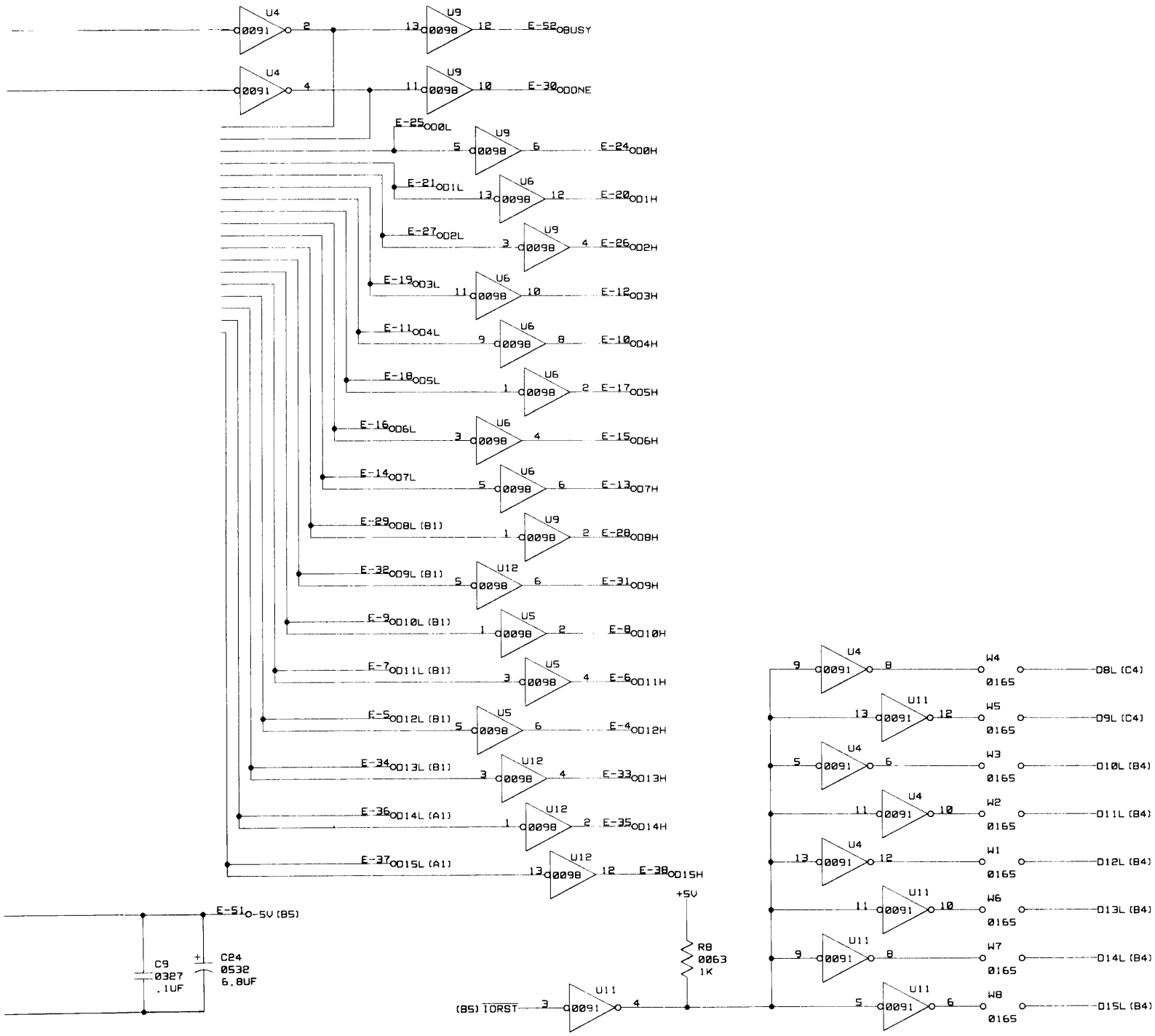
DEVICE CODE BITS	0	1	2	3	4	5
JUMPER FOR A1	W3	W2	W1	W6	W7	W8

INSERT W4 TO DISABLE INTERNAL ADDRESS COUNTER
INSERT W5 FOR POSITIVE LOGIC AT 0615 PINS

NOTES:
1. E-NUMBERS ARE WIRE WRAP PINS ON GP INTERFACE WITH PINS AND SOCKETS OR SOLDER POINTS ON GP INTERFACE WITHOUT PINS.
2. INSTALL W4 TO DISABLE INTERNAL ADDRESS COUNTER.
3. D(0-15)H ARE ACTIVE HIGH, D(0-15)L ARE ACTIVE LOW.
4. W5 SETS LOGIC LEVELS OF D(0-15)H AND D(0-15)L.

DG-25859

Figure 4-3 General-purpose interface card logic diagram



W5	OUT	IN
D(0-15)H	H=1 L=0	H=0 L=1
D(0-15)L	H=0 L=1	H=1 L=0

Programming

The GPI card consists of an IOC module and space in which the rest of an I/O interface can be implemented. The IOC module contains a B register which can be both loaded and retrieved and a C register which can be loaded. When the card is used to implement a data channel controller, the B register can be used as the memory address register and the C register can be used as the word count register. Alternatively, the IOC module can directly control the loading and retrieving of up to three input and three output registers.

The IOC's registers respond to standard I/O instructions with the user-defined device code.

Summary of GPIO Bus Signals

The 55 signals which comprise the GPIO bus can be divided into five groups, each summarized here in table form.

Data Signals

D(0-15)H Data Out. All data for both data channel and programmed I/O are transferred from lines. Each line is buffered to drive 10 standard TTL loads. The contents of the polarity bit (controlled by jumper W5) determines whether a low level should be interpreted as a 0 or a 1.

D(0-15)L Data Input. All data and addresses for both data channel and programmed I/O are transferred from the device interface to the IOC via these 16 input lines. The interrupt disable mask bit is determined by one of these lines when the MSKO signal is asserted. The device code, external register select bit, and the polarity bit are carried on these lines when the signal IORST is asserted (see IORST, MSKO, and Jumpers). The device interface should drive these lines with open collector gates. The contents of the polarity bit determines whether a low level should be interpreted as a 0 or a 1.

The following control signals are asserted low (1 = 0 V). They can drive up to 10 TTL loads.

Programmed I/O Signals

DIA Data In A. Asserted by the IOC upon receipt of its DIA instruction. To be used by the device interface to place the contents of its A input buffer on D(0-15)L.

DIB Data In B. Asserted by the IOC upon receipt of its DIB instruction. To be used by the device interface to place the contents of its B input buffer on D(0-15)L if external registers are enabled (see Jumpers).

DIC Data In C. Equivalent to DIA, except that it applies to the C input buffer.

DOA Data Out A. Asserted by the IOC upon receipt of its DOA instruction. To be used by the device interface to load the contents of D(0-15)H into its A output buffer.

DOB Data Out B. Asserted by the IOC upon receipt of its DOB instruction. To be used by the device interface to load the contents of D(0-15)H into its B output buffer.

DOC Data Out C. Equivalent to DOB, except that it applies to the C output buffer.

STRT Start. Asserted by the IOC upon the receipt of any of its nonskip I/O instructions in which bits 8 and 9 = 01 (i.e., instructions in which the S control function is specified). Asserted during DIA, DIB, DIC, DOA, DOB, and DOC instructions after the data transfer has occurred. Usually used to initiate the device interface by setting the Busy flag to 1 and the Done flag to 0.

CLR Clear. Asserted by the IOC upon the receipt of any of its nonskip I/O instructions in which bits 8 and 9 = 10 (i.e., instructions in which the C control function is specified). Asserted during DIA, DIB, DIC, DOA, DOB, and DOC instructions after the data transfer has occurred. Usually used to terminate device operation by setting the Busy and Done flags to 0.

IOPLS I/O Pulse. Asserted by the IOC upon receipt of any of its nonskip I/O instructions in which bits 8 and 9 = 11 (i.e., instructions in which the P control function is specified). Asserted during DIA, DIB, DIC, DOA, DOB, and DOC instructions after the data transfer has occurred. Usually used to initiate special device operations.

SET BUSY Asserted by the interface when it is busy and should not be disturbed by the IOC. In the IOC, sets the Busy flag to 1 and the Done flag to 0.

SET DONE Asserted by the interface to notify the IOC that it has completed its operation. In the IOC it sets the Done flag to 1 and the Busy flag to 0.

Program Interrupt Signals

INT SYNC Interrupt Synchronize. Asserted by the interface to notify the program that it has completed its operation. In the IOC it directly initiates a program interrupt request without disturbing either the Busy or Done flags.

MSKO Mask Out. Asserted by the IOC during the execution of a MSKO instruction. Loads the selected data line into the priority mask bit register.

Data Channel Signals

DCH SYNC Data Channel Synchronize. Asserted by the device interface to request data channel service.

DCHA Data Channel Acknowledge. Asserted by the IOC at the beginning of each data channel cycle in response to a data channel request from the device interface. The interface should respond by placing the memory address on D(1-15)L and the direction of transfer on D(0)L. A logical 1 on D(0)L indicates an input transfer and a logical 0 indicates an output transfer.

DCHI Data Channel Input. Asserted by the IOC during a data channel input cycle. To be used by the interface to place the contents of its data register onto the D(0-15)L lines.

DCHO Data Channel Output. Asserted by the IOC during a data channel output cycle after the IOC has placed the contents of its data register onto the D(0-15)H lines. To be used by the interface to load the contents of this bus into its data register.

WCEZ Word Count Equals Zero. Asserted by the IOC during the data channel cycle that overflows the word count register. Can be used to clock the Done flip-flop in the interface.

CLK Clock. Follows FSTROBE when the IOC is not performing an operation.

System Control Signals

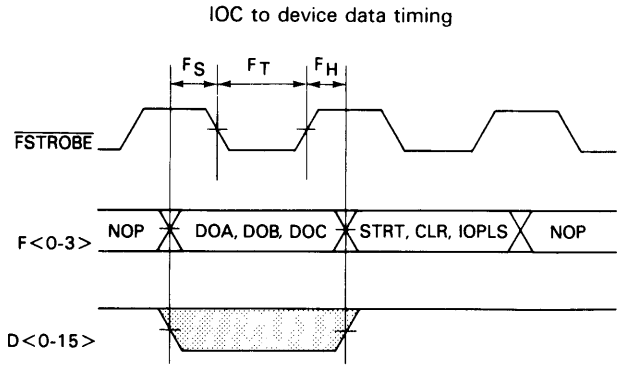
IORST I/O Reset. Asserted by the IOC during the IORST instruction. This signal initializes the IOC by loading the device code, the polarity bit, and the external register bit. It should also initialize the interface.

Interface Timing

This section shows timing diagrams and tables for programmed I/O and data channel operations.

Programmed Transfers

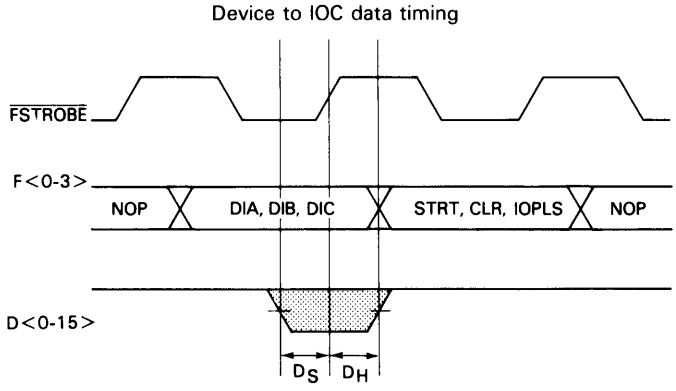
A transfer occurring under direct program control moves a word or part of a word between the IOC and a register in the device interface. Refer to Figure 4-4.



NOTES

Unless otherwise noted, time intervals are measured between points at 1.5V.

See timing Tables 4-2 and 4-3.



DG-04142,DG-04143

Figure 4-4 Timing, programmed transfers

Data In

The I/O controller asserts DIA, DIB, or DIC. It also asserts STRT, CLR, or IOPLS if they are specified by the I/O instruction. When a DIA, DIB, or DIC signal asserts, any data on the GPIO data bus (D0-D15)L lines will be gated from the interface, through the IOC, and onto the I/O data bus.

Data Out

The I/O controller places the data received from the CPU onto the GPIO data bus D(0-15)H lines and asserts DOA, DOB, or DOC. It also asserts STRT, CLR, or IOPLS if they are specified by the I/O instruction.

Data Channel Transfers

An information transfer occurring under data channel control moves a block of data, one word at a time, between the IOC and the device. Refer to Figure 4-5.

Data In

The IOC generates DCHA and the interface responds by placing the memory address on D(1-15)L if the external register is enabled and by placing the direction of transfer on D(0)L. If the word count register in the IOC overflows, WCEZ is generated to indicate that this is the last word in the data block. The IOC then generates DCHI and the interface responds by placing its data on D(0-15)L.

Data Out

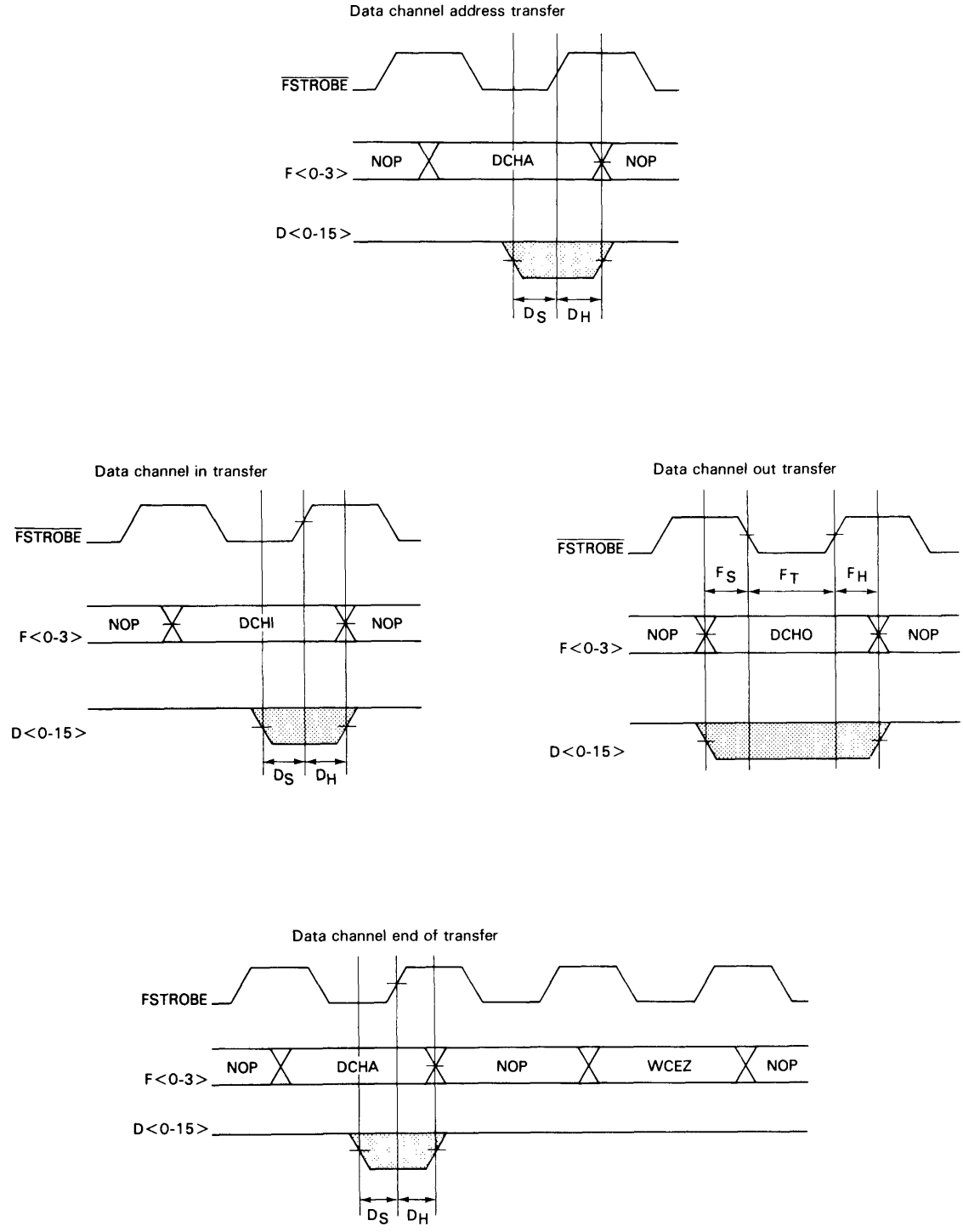
The IOC generates DCHA and the interface responds by placing the memory address on D(1-15)L if the external register is enabled and by placing the direction of transfer on D(0)L. If the word count register in the IOC overflows, WCEZ is generated to indicate that this is the last word in the data block. The IOC then generates DCHO; the interface responds by loading the contents of D(0-15)H into its data register.

For more detailed information refer to the *microNOVA Integrated Circuits Data Manual.*

Table 4-2 I/O data port timing table

Class	Mnemonic	Min.*	Max.*	Units
All	T_1	110	130	ns
	T_2	110	130	ns
	T_{SKEW}	−5	+15	ns
	T_N(next command)	840	—	ns
I/O Instructions	DT_O (Data Out Transfer)	470	850	ns
	DT_I (Data In Transfer)	1190	1330	ns
Request Enable and Data Channel Address Request	A	360	540	ns
Data Channel Transactions	C (Data Channel Address Request to first transfer)	710	850	ns
	DC_O (Data Channel Out)	590	1930	ns
	DC_I (Data Channel In)	1190	1210	ns

**All of the above times assume a MASTER CLOCK frequency of 8.333 MHz.*



NOTES

Unless otherwise noted, time intervals are measured between points at 1.5V.

See timing Tables 4-2 and 4-3.

DG-04145-48

Figure 4-5 Timing, data channel transfers

Table 4-3 Peripheral port timing table

Operation	Mnemonic	Description	Min.	Max.	Unit
Common to all operations	F_S	Function code setup time prior to $\overline{\text{FSTROBE}}$	60		ns
	F_T	$\overline{\text{FSTROBE}}$ duration	180	300	ns
	F_H	Function pin hold time	60		ns
I/O data out instruction and Data Channel Out Transfer		Data output timing same as function code timing above			
I/O data in instruction and Data Channel in transfer	D_S to	Data setup time prior to $\overline{\text{FSTROBE}}$	120	240	ns
	D_H	Data hold time after $\overline{\text{FSTROBE}}$	0	120	ns
	T_{AI}	DCHA to DCHI	336		μs
	T_{AO}	DCHA to DCHO	656		μs

Jumpers

The device code, polarity bit, and external register select bit are selected by jumpers in the IOC section of the card (see Figure 4-6).

Device Select

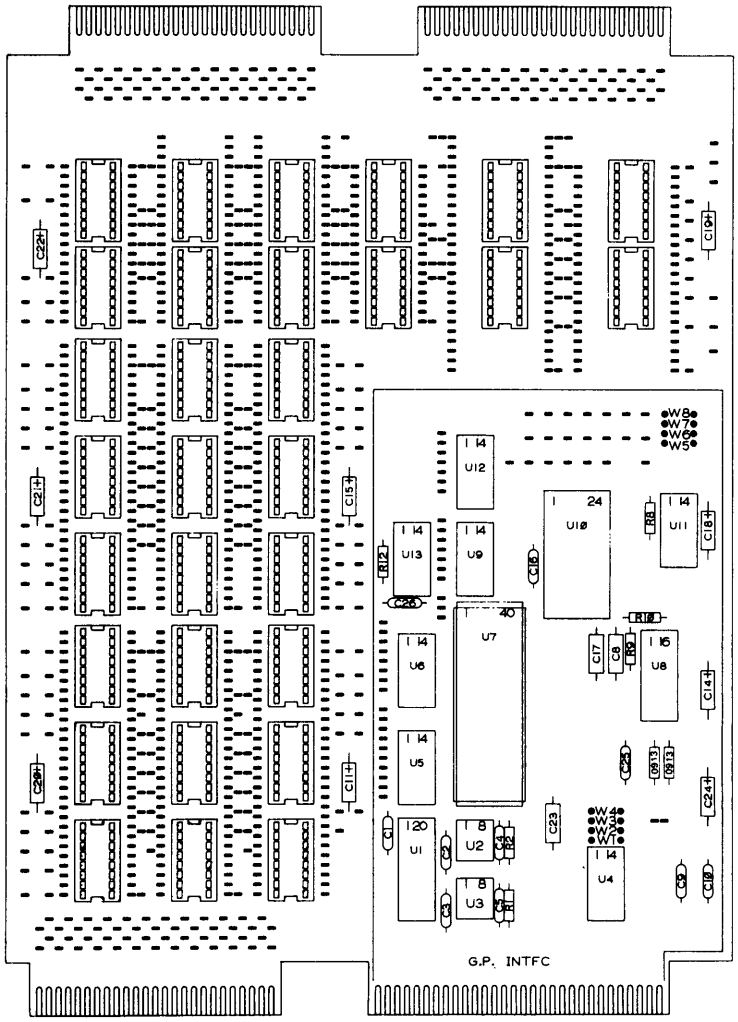
The device code is selected with jumper connections W1-W3 and W6-W8. The device codes for specific devices are given in Appendix C.

Table 4-4 Device select jumpers

Bits of device code field	10	11	12	13	14	15
Jumper insertion to specify 1	W3	W2	W1	W6	W7	W8

Polarity Select

Jumper W5 selects the polarity bit. The polarity bit is a 1-bit register that determines the sense of the data bits transmitted and received via the IOC. If W5 is in, the polarity bit is set to a 1 and a low level (0 V) on the data pins of the IOC is interpreted as a 0. A high level (5 V) is interpreted as a 1. If W5 is out, the polarity bit is set to a 0 and a low level on the data pins of the IOC is interpreted as a 1. The high level is interpreted as a 0. Note that the buffered outputs, D(0-15)H, are inverted.



DG-25860

Figure 4-6 General-purpose interface physical layout

Table 4-5 Polarity select jumper

W5	Data	Polarity (GPIO Bus)
In	D(0-15)H	Zero = +5 V; One = 0 V
Out		Zero = 0 V; One = +5 V
In	D(0-15)L	Zero = 0 V; One = +5 V
Out		Zero = +5 V; One = 0 V

External Register Select

Jumper W4 controls the selection of the external (device interface) or internal (IOC) memory address and word count registers.

Table 4-6 External register select jumper

W4	Location of Registers
In	External to IOC
Out	Internal to IOC

Priority Mask Bit Select

The interrupt priority mask bit is selected by jumpering the mask signal (MSKO, pin 44) to one of the D(0-15)L lines.

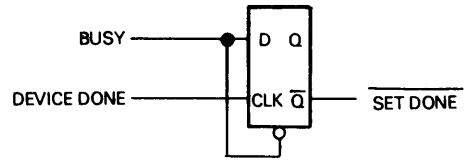
Data Lines and Drive Capability

The outputs of the I/O controller (IOC) chip are capable of driving only one TTL load. Therefore, all the data out lines, D(0-15)H, have been TTL buffered and are capable of sinking 16 mA. The outputs of the 4 to 16 decoder are also capable of sinking 16 mA. The data input lines, D(0-15)L, should be driven with open collector drivers. Each control signal to the IOC (INTSYNC), pin 23; DCHSYNC, pin 22; SET BUSY, pin 1; and SET DONE, pin 2, constitute one TTL input load.

The supply voltages required (+5 Vdc, pin 58; +12 Vdc, pin 57; and −5 Vdc, pin 51) must be supplied to the board by the system into which it is installed. The maximum current drain on the +5 Vdc should be one ampere.

Busy/Done

A suggested circuit for generating the SET DONE signal is given in Figure 4-7. SET BUSY can be generated in the same way.



DG-05872

Figure 4-7 Set signal done

Interface Wire-wrap Pins

Wire-wrap pins are provided in the IOC section of the model 4211 card to facilitate the connection of the GPIO bus to the custom device controller. Table 4-7 lists the wire-wrap pins associated with each bus signal. The location of the pins may be found by referring to the physical layout of the card. The model 4210 GPIO card does not include wire-wrap pins, but features etched circuit holes in the same locations.

Table 4-7 Wire-wrap pins (GPIO card)

Pin	Signal	Pin	Signal
1	SET BUSY	30	DONE
2	SET DONE	31	D9H
3	MASTER CLOCK	32	D9L
4	D12H	33	D13H
5	D12L	34	D13L
6	D11H	35	D14H
7	D11L	36	D14L
8	D10H	37	D15L
9	D10L	38	D15H
10	D4H	39	DIB
11	D4L	40	DOA
12	D3H	41	CLR
13	D7H	42	DCHA
14	D7L	43	CLK
15	D6H	44	MSKO
16	D6L	45	DOC
17	D5H	46	DIA
18	D5L	47	STRT
19	D3L	48	IORST
20	D1H	49	WCEZ
21	D1L	50	DCHO
22	DCH SYN	51	-5V
23	INT SYN	52	BUSY
24	D0H	53	DOB
25	D0L	54	DIC
26	D2H	55	OPLS
27	D2L	56	DCHI
28	D8H	57	+15V
29	D8L	58	+5V

Additional Printed Circuit Card

A

The model 1114 predrilled circuit card is a general-purpose wiring card suitable for building a special-purpose, custom interface to a Desktop Generation computer. The card has two male edge connectors on the backpanel edge, one of which plugs into the printed circuit backpanel socket; the other plugs into a device cable connector, in the event the card is used as an I/O device controller. Figure A-1 presents the dimensions of the card.

The backpanel connector has 60 contacts, while the device connector has 50 contacts. Two additional 50-contact edge connectors are included on the forward edge of the card to provide flexibility to the interface designer. A series of holes in the card is designed to accept standard integrated circuit packages. Up to fifty 14-pin packages may be installed on one card, and an even larger number of smaller packages may be used. Each hole and connector finger is connected by etch to an adjacent wire-wrap pin.

ASCII Character Set

B

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT. TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ^
95	137	5F	← OR _
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

DG-05495

Figure B-1 ASCII Character Codes

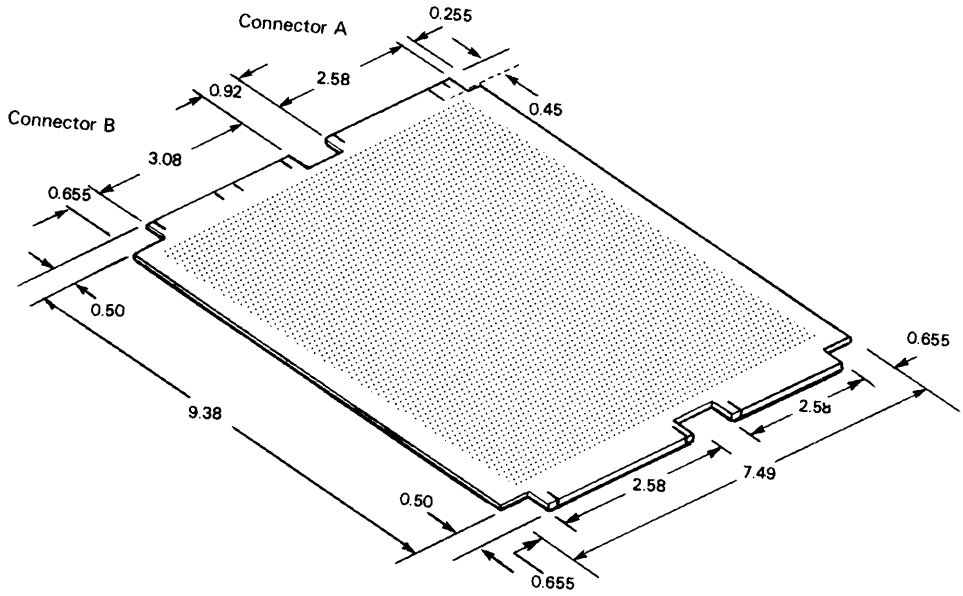


Figure A-1 Model 1114 card dimensions

DG-02420

Peripheral Device Codes

C

Table C-1 I/O device codes for Desktop Generation Computers

Octal Device Code	Mnemonic	Priority Mask Bit	Device Name
00	—	—	Returned by power monitor in response to INTA
01	APL	—	Auto program load register
02	PAR	—	Parity checking
03	MAP	—	Memory allocation and protection
04			
05			
06	ATP	—	Attached processor
07			
10	TTI	14	TTY input
11	TTO	15	TTY output
12			
13			
14	RTC	13	Real-time clock
15			
16			
17	LPT	12	Line printer
20	DEO	7	368.64-Kbyte diskette; up to 2 diskettes in subsystem
21	ADCV	8	A-D converter
22	MTA	10	Magnetic tape
23	DACV	8	D-A converter

Table C-1 Table C-1 I/O device codes for Desktop Generation Computers

Octal Device Code	Mnemonic	Priority Mask Bit	Device Name
24			
25			
26	DEP	7	15.98-Mbyte disk; up to 2 disk drives in subsystem
27			
30			
31			
32			
33			
34	ASLM	11	Sync/async controller
35			
36			
37			
40			
41	TLC	4 or 11	IEEE-488 Bus interface
42	DIO	5	Digital I/O interface
43	PIT	11	Programmable interval timer
44			
45			
46			
47			
50			
51			
52			
53			
54			
55			
56			
57	LPT1	12	Second line printer
60			
61	ADCV1	8	Second A-D interface
62			
63	DACV1	8	Second D-A interface
64			
65			
66			
67			
70			
71			
72			
73			
74	ASLM1	11	Second sync/async controller
75			
76			
77	CPU	—	Central processor and console functions

Index

Within the index, the letter "f" following a page entry indicates "and the following page"; the letters "ff" following a page entry indicate "and the following pages".

D

E

F

G

H

I

W

moisten & seal

Documentation Comment Form

Manual Title _____

Manual No. _____

Your Name _____

Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

Please help us improve our future publications by answering the questions below. Use the space provided for your comments. Thank you.

	Yes	No
Is this manual easy to read?	☐	☐
Is it easy to understand?	☐	☐
Are the topics logically organized?	☐	☐
Is the technical information accurate?	☐	☐
Can you easily find what you want?	☐	☐
Does it tell you everything you need to know?	☐	☐
Do the illustrations help you?	☐	☐

If you wish to order manuals, contact your sales representative or dealer.

Comments:

Date

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Design Services (E219)
4400 Computer Drive
Westboro, MA 01581