EDUCATIONAL
SERVICES

VAX/VMS
Training

VAX/VMS
Device Driver
I/O Data Structures

digital

EY-2278E-MB-0001

I/O DATA STRUCTURES

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, October 1984

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

FIGURES

TABLES

INTRODUCTION

Because a driver and the operating system cooperate to process an I/O request, they must have a common I/O data base. The data structures (tables) utilized by the I/O system to support and control I/O operations include:

- **Driver tables** that allow VAX/VMS to load drivers, validate device functions, and call drivers at their entry points.
- **Control blocks** that describe every device unit, controller, bus adapter, and channel (logical path from a process to a device).
- **I/O request packets** that define individual requests for I/O activity.

This module examines the functions and format of the principal data structures, and presents an overview of the interrelationships among them. The names and contents of specific fields within a data structure may be found in Appendix A of the VAX/VMS Guide to Writing a Device Driver.

OBJECTIVES

Upon completion of this module you will be able to:

1. Identify the data structures used in the I/O subsystem of VMS.
2. Show the interrelationships among the principal I/O data structures.
3. Describe the function(s) of each of the principal I/O data structures.

RESOURCE

1. Guide to Writing a Device Driver for VAX/VMS

TOPICS

- I/O Data Base
  - Used as method of communication between:
    - User
    - Driver
    - VMS
  - Data Structures created and maintained by Driver, VMS and User (through $QIO System Service)

- List of Data Base Control Blocks
  - Channel Control Block (CCB)
  - I/O Request Packet (IRP)
  - Unit Control Block (UCB)
  - Function Decision Table (FDT)
  - Device Data Block (DDB)
  - Channel (controller) Request Block (CRB)
  - Interrupt Dispatch Block (IDB)
  - Adapter Control Block (APD)
  - Driver Dispatch Table (DDT)

CHANNEL CONTROL BLOCK (CCB)

- Provides the link between a process channel number and a device unit for a $QIO request (see Figure 2-1).
- Located in the channel control region of P1 space.
- Constructed by $ASSIGN system service.
- Provides a link for file-structured devices to a set of mapping pointers. The mapping pointers facilitate translating virtual block numbers in a file to logical block numbers on a volume. The mapping pointers are kept in a Window Control Block (WCB).

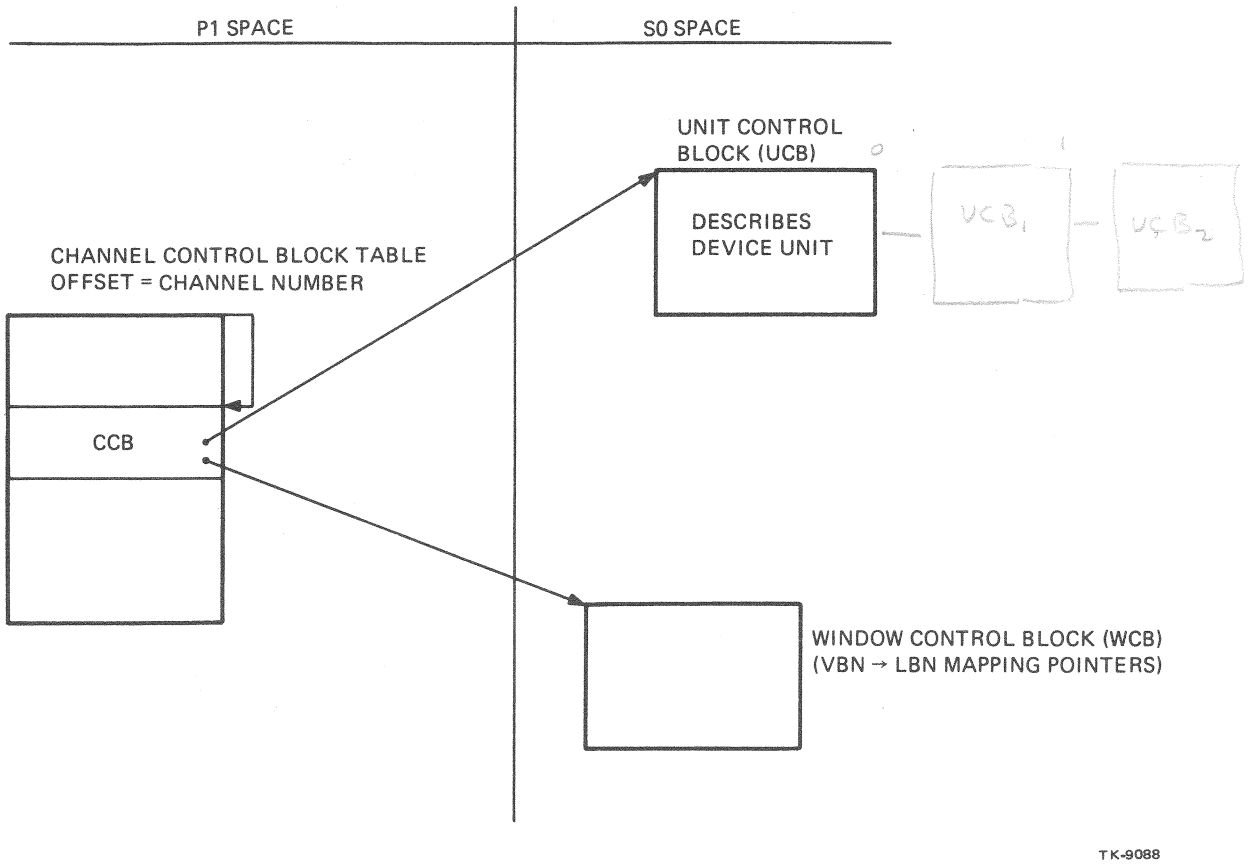


Figure 2-1 Channel Control Block

The channel number is used to locate the associated device unit (via the CCB). Mapping pointers (found in the WCB) are used to perform virtual-to-logical block translations for file-structured devices (disks).

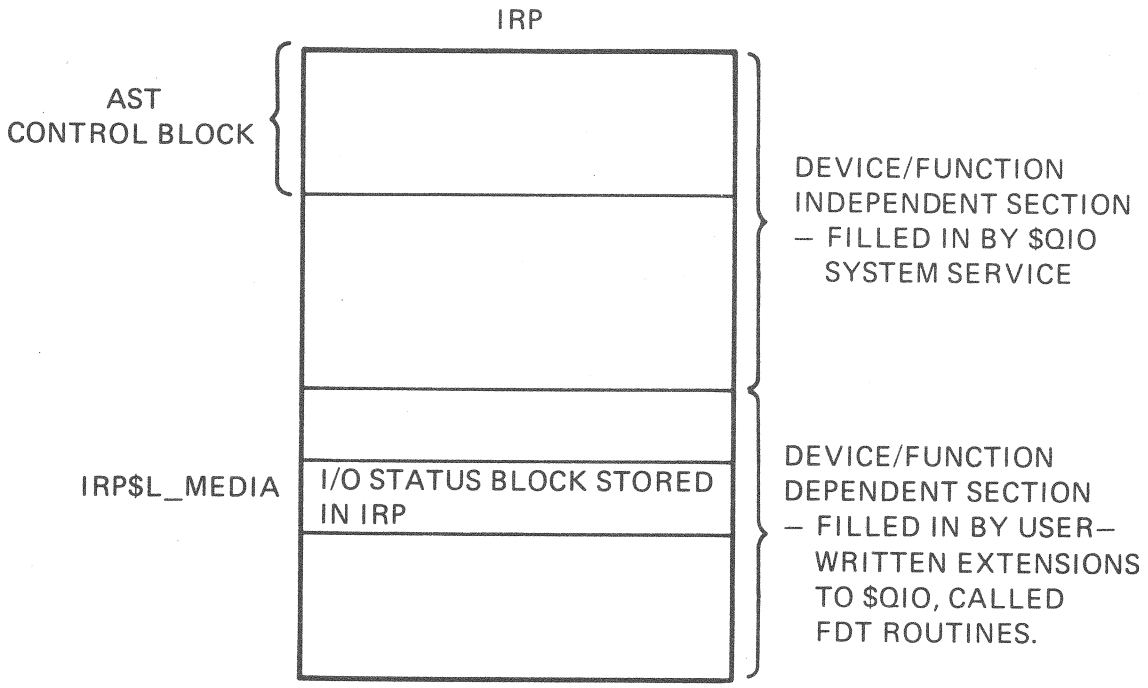
I/O REQUEST PACKET (IRP)

- Parameter block built to describe a particular I/O request from arguments supplied to the $QIO system service (see Figure 2-2).
- Allocated from nonpaged system dynamic memory by $QIO.
- Is comprised of two sections:

  1. function/device independent
  2. function/device dependent

- May be linked to a queue of outstanding I/O requests for the appropriate device unit.
- Queued to the appropriate device unit according to the base priority of the requesting process.
- On I/O completion, status information is stored in the IRP for return to the requesting process in the IOSB.
- Used as an AST Control Block to queue a special kernel AST for return to requesting process context by the I/O Post routine.

IRP Lookaside list
IRPCOUNT
IRPCOUNTV
IRPSIZE – 160



Note: IRP$_media = IRP$L_IOST1
IRP$_media+4=IRP$L_IOST2

Figure 2-2 I/O Request Packet

The IRP carries all of the information necessary to describe a particular I/O request.

UNIT CONTROL BLOCK (UCB)

I/O in VMS is unit based, that is, simultaneous operations are allowed at the unit (rather than the controller) level. There is one UCB per device unit (see Figure 2-3). The UCB is the largest driver data structure. It is also the focus of the driver operation. It serves the following functions:

- Contains device characteristics and unit status information, for example, whether the device is shareable and/or whether the unit is busy.
- When an IRP has been built, a check is made to see if the unit is busy. If so, the IRP is linked to a queue of outstanding requests for the unit. If not, the driver is activated with the IRP set as the current request for the unit.
- Used as a listhead for the queue of IRPs for the unit.
- Provides pointers to the Device Data Block (DDB), the Driver Dispatch Table (DDT) and the Channel Request Block (CRB) (all of which are described later).
- Provides storage for device dependent parameters. (A section at the bottom of the UCB is reserved for this purpose.)
- Contains details of the I/O operation in progress (byte count, buffer address, etc.).
- Contains the Fork Context Block, the area where R3, R4 and the PC are stored when the driver process is put into a wait state.
- UCBs for device units on the same controller are linked together.

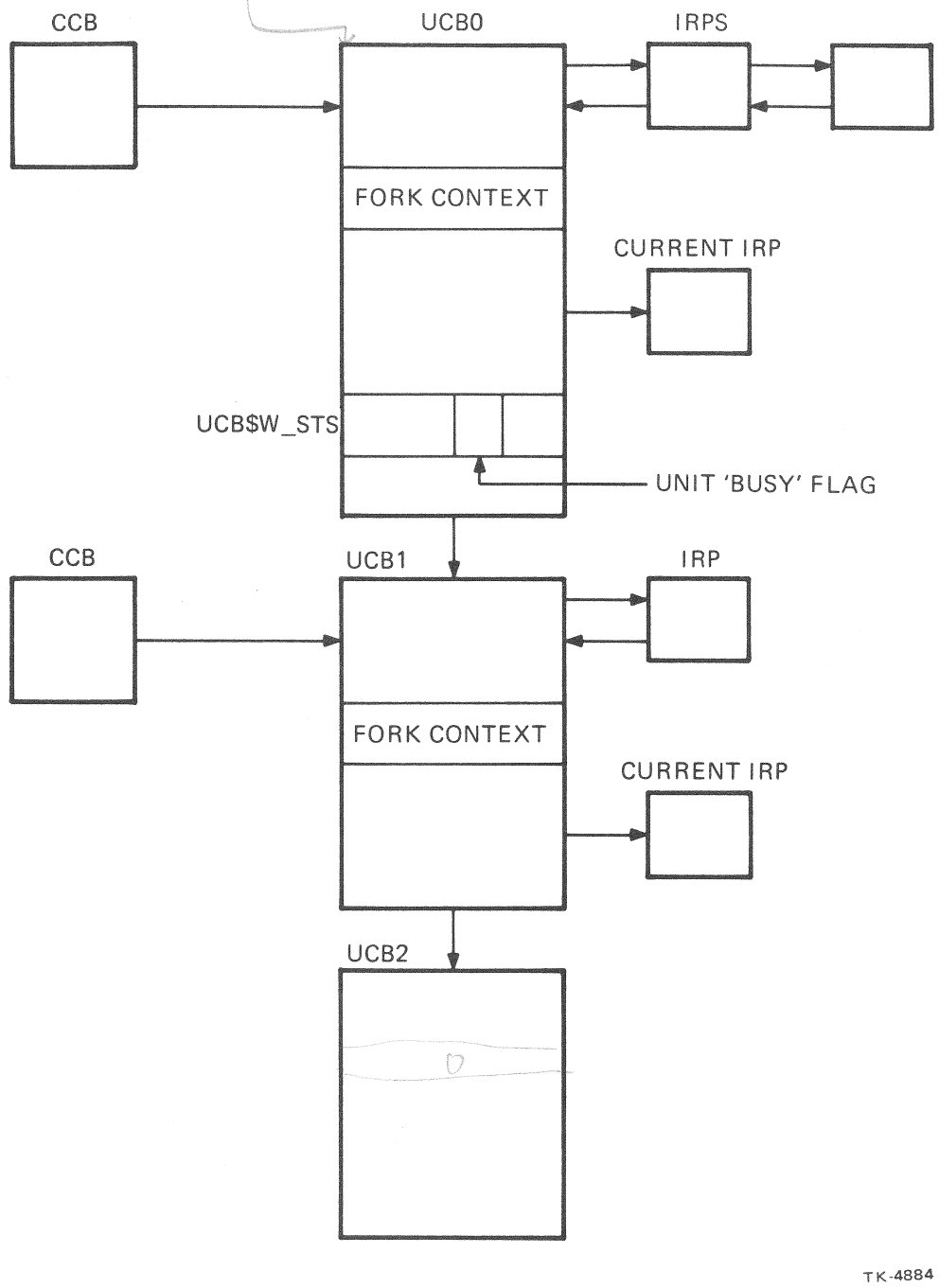


Figure 2-3 Unit Control Block

UCBs are the largest and most important data structures in processing I/O operations. UCBs describe every device unit on the system, and contain fork process context.

Fork Queue

The UCB holds the **fork context block.** The driver, executing as a fork process triggered by a device or fork interrupt, runs with limited context held in two registers, R3 and R4. Thus, when the driver wishes to wait for a device interrupt or to fork, it simply "folds" its context into the UCB by storing R3, R4 and the PC of the next instruction in the fork context block.

When creating a fork, the UCB is linked to a list of pending fork processes at the appropriate IPL, that is, the fork block is placed in the appropriate fork queue.

The Fork Dispatch Code is executed as the result of a software interrupt at a fork IPL. The next fork block is removed from the fork queue corresponding to that IPL, and the driver context is restored from the fork context block to continue processing (see Figure 2-4).

The fork process runs to completion, or until a resource becomes unavailable, at which point, it folds its context into the fork context block. Control is returned to the fork dispatcher. The next fork block is dequeued, and the process is repeated.

Finally, when the fork queue is empty, the software interrupt at fork IPL (which triggered the fork dispatcher) is dismissed with an REI instruction.

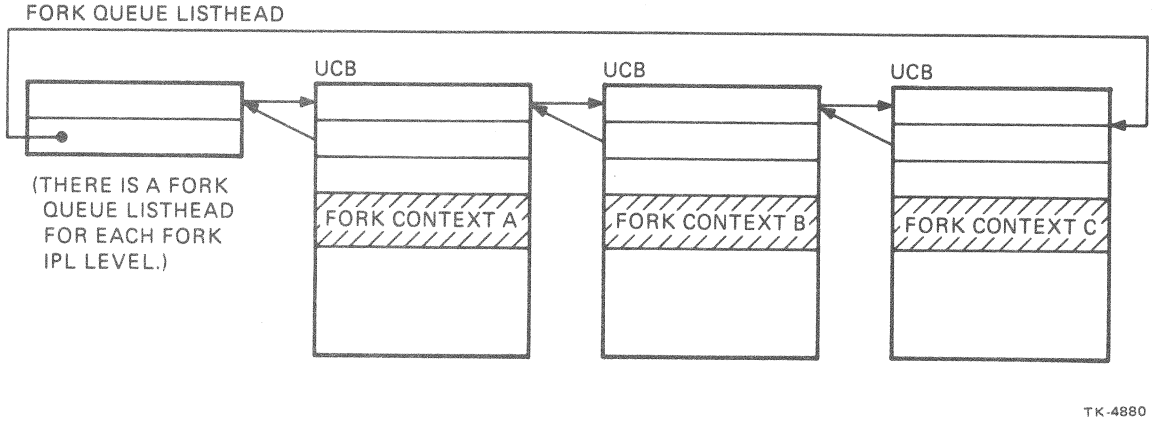


Figure 2-4 Fork Queues - Doubly Linked Lists of UCBs

FUNCTION DECISION TABLE (FDT)

- Is used to specify the legal $QIO function codes for the device driver.
- Is used to indicate which $QIO functions require an ancillary memory buffer (buffered I/O functions).
- Contains a table of dispatch pointers to FDT routines which are used to perform device/function dependent preprocessing (validating the $QIO request, and filling in fields in the IRP).

The function code specified in a $QIO call is a number in the range of Ø-63. There is a legal function mask at the start of the FDT which is 64 bits long (each bit corresponds to a specific function code).

The $QIO service checks the function code specified to see if it is a legal function by using the function code as a bit number index into the legal function mask. If the corresponding bit is set, the function is recognized by the driver.

Similarly, a decision is made to see if the function is buffered (for quota checking) by examining the appropriate bit in the buffered function mask. A clear bit indicates that the function is a direct I/O.

Dispatching to the appropriate FDT routine(s) is performed by scanning the function selection masks and checking to see if the bit corresponding to the function code is set. For each selection mask with the appropriate bit set, the associated FDT routine is called.

Notice that this is an extremely flexible dispatching mechanism. Many function codes may dispatch to a given routine, or every function code may dispatch to many routines easily. Note also that special system macros exist which allow you to construct the FDT with minimal effort.

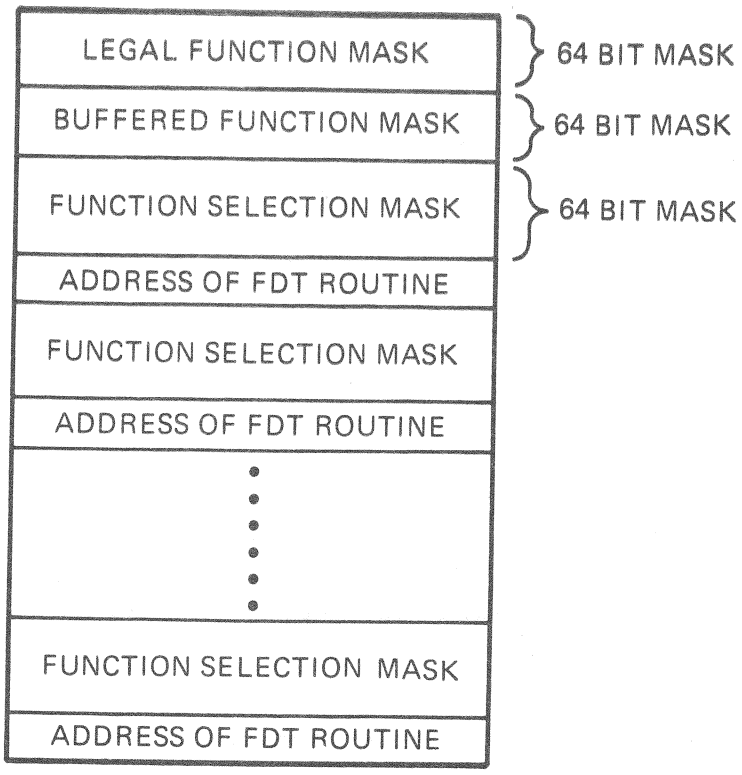


Figure 2-5 Function Decision Table

Every driver has an FDT located via a pointer stored in the appropriate DDT.

DEVICE DATA BLOCK (DDB)

For every device type in the system (for every controller), there is a DDB which contains:

- Information common to all devices of the same type connected to a particular controller (device type name).
- Pointers (UCB,DDT) to other data structures (see Figure 2-6).

The $ASSIGN system service follows the forward pointers from IOC$GL_DEVLIST until it finds the correct DDB, then it follows the forward pointers until it finds the appropriate UCB, which it stores in the CCB. The $QIO system service follows the back pointer from the UCB (which it finds through a pointer in the CCB), to the Driver Dispatch Table, where FDT processing begins.

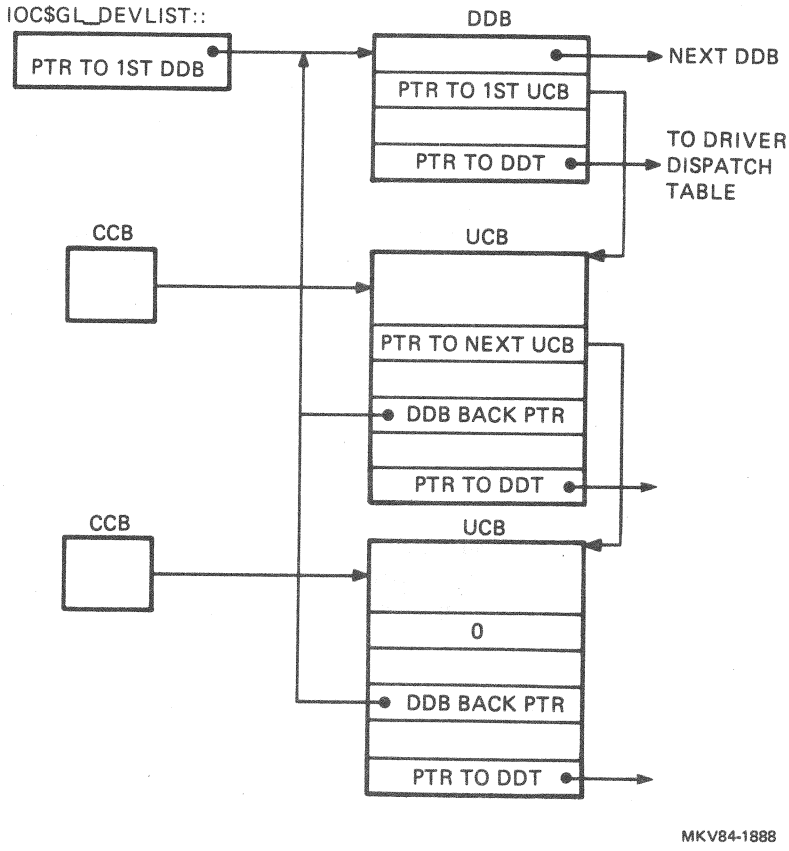


Figure 2-6 Device Data Block

The DDBs are singly linked. The symbol IOC$GL_DEVLIST is the address of a longword which contains the address of the first DDB.

CHANNEL REQUEST BLOCK (CRB)

Although simultaneous operations on units under the same controller are allowed (overlapping seeks on disk units under a given controller), certain operations require exclusive use of the controller (for example, a data transfer). Arbitration for the controller is performed using the CRB (better thought of as a Controller Request Block, since it has nothing to do with the channel number used by the process). The CRB:

- Contains code to dispatch interrupts from the controller, and contains a pointer to an associated Interrupt Data Block (see Figure 2-7).
- Contains a 'busy' flag for the controller, used to arbitrate requests for simultaneous actions when multiple units are connected to a single controller.
- Contains a listhead for the queue of units (UCBs) awaiting access to the controller.

For UNIBUS devices, there is one CRB per device controller. For MASSBUS devices, there is a primary CRB for the MASSBUS adapter, and a secondary CRB for each subcontroller (TMØ3 magnetic tape controller) under the MASSBUS adapter. No CRB is created for single-device controllers (disk controllers) for MASSBUS devices. The I/O data base differences for MASSBUS devices are discussed in detail later.

Interrupt Dispatch Block (IDB)

For device interrupts, the CRB transfers control to the appropriate driver interrupt service routine, which is used to locate the associated IDB. For UNIBUS devices, the IDB has a pointer to the appropriate UNIBUS Adapter Block (ADP), which is used to control operations through the UNIBUS Adapter, and in initial dispatching of a UBA interrupt (discussed in more detail later). The IDB:

- Contains a table of UCB addresses for units under this controller.
- Contains a pointer to the appropriate device hardware control and status registers.
- Contains a pointer to the UCB that currently "owns" the controller.

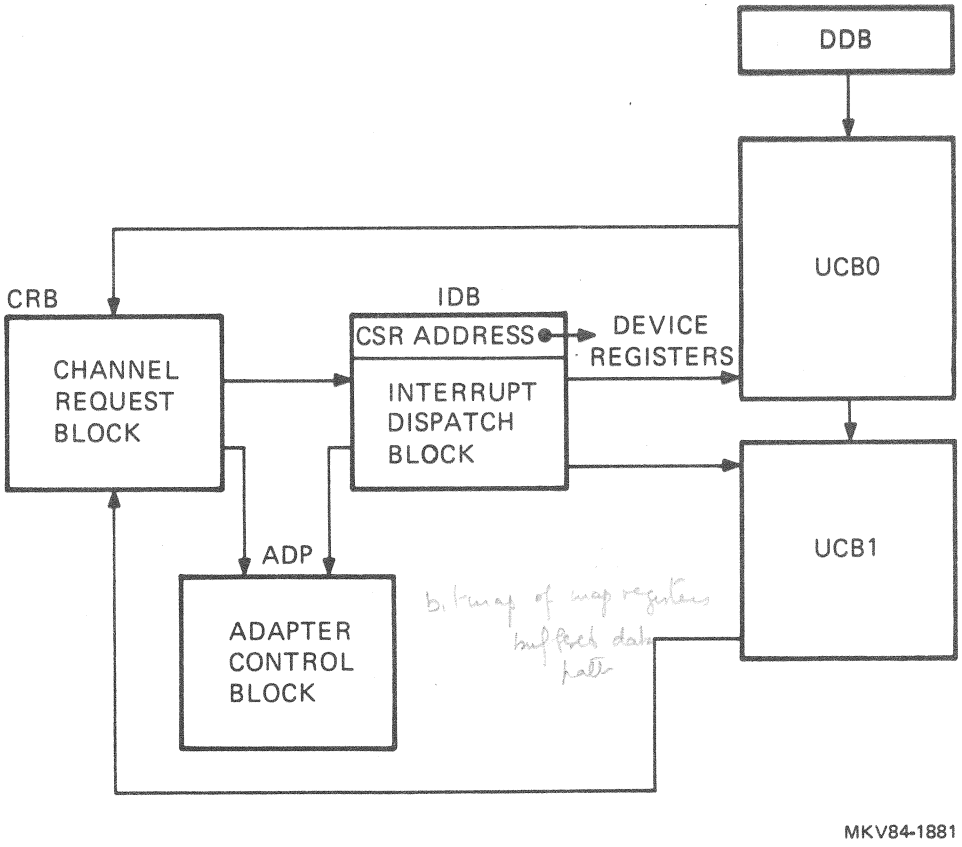


Figure 2-7 Channel Request Block, Interrupt Dispatch Block, and Adapter Control Block

The CRB, IDB and ADP are closely related data structures. They are used for controller arbitration and interrupt dispatching.

NOTE

The CSR address is the virtual address (SØ) in I/O space that translates to the device's CSR physical address.

Channel (Controller) Wait Queue

When the driver wishes to perform a function that requires exclusive ownership of the controller, a routine is called to request the 'channel'. This routine checks to see if the controller is busy, and if so, saves the driver context in the UCB and links the UCB in the controller wait queue (see Figure 2-8).

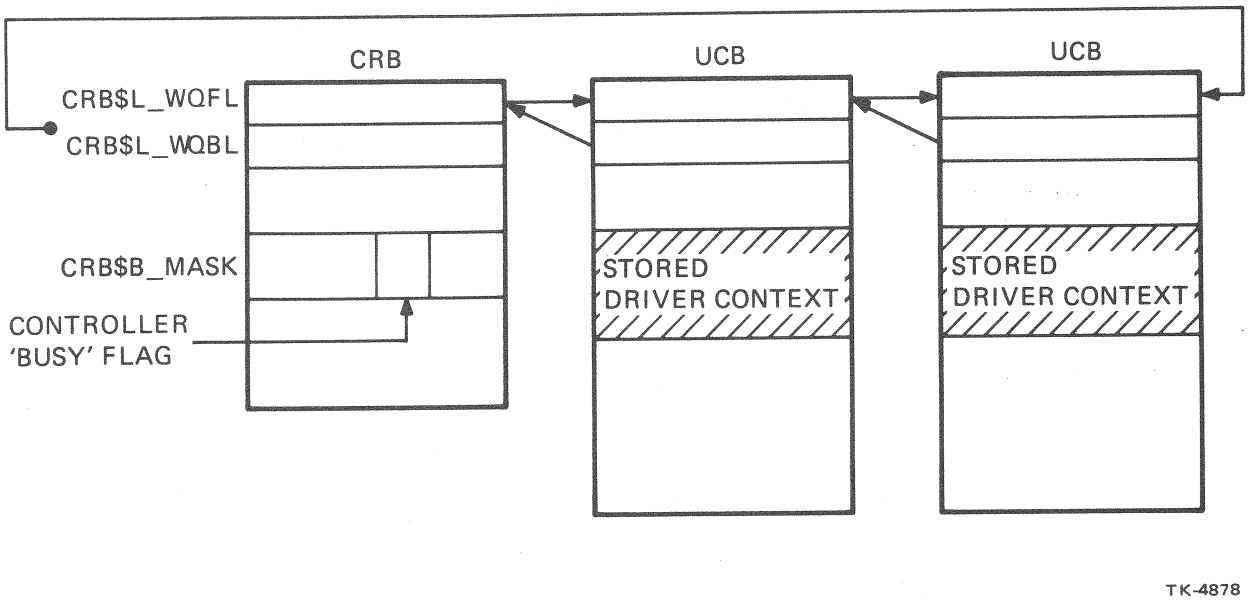


Figure 2-8 Channel (Controller) Wait Queue, with Two UCBs Waiting for the Channel

ADAPTER CONTROL BLOCK (ADP)

- Is used to locate UBA Control and Status Registers (see Figure 2-9).
- Contains information used to allocate UBA mapping registers and UBA data paths.
- Contains data path wait queue listhead and mapping register wait queue listhead (see Figure 2-10).
- Contains code used in servicing interrupts.

The ADP can be located by pointers in the CRB and IDB for UNIBUS devices.

The UNIBUS ADP contains a bit map used to allocate data paths. A one-word map contains 16 bits corresponding to the 16 data paths.

When the driver requires mapping registers to perform a UBA transfer through a buffered data path, routines are called to locate a free data path (by scanning the datapath bit map). If a data path is obtained, the map register section is searched for a set of contiguous map registers sufficient to map the DMA transfer.

Note that a bit set in the data path bit map indicates that the data path is free, while a clear bit indicates that it is allocated.

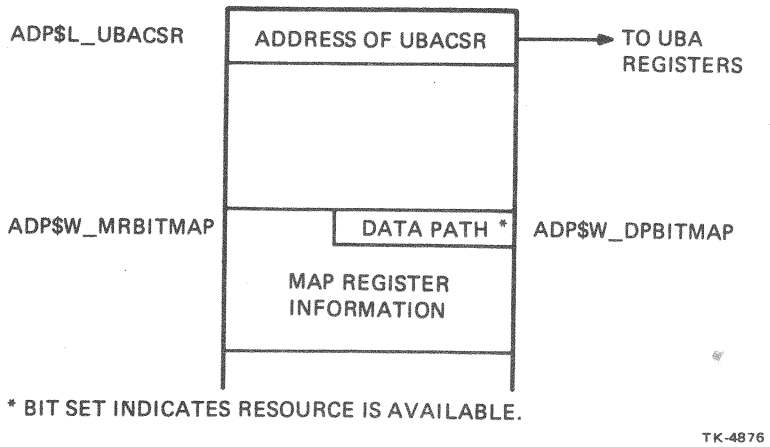


Figure 2-9 The UNIBUS Adapter Control Block

Data Path and Mapping Register Wait Queues

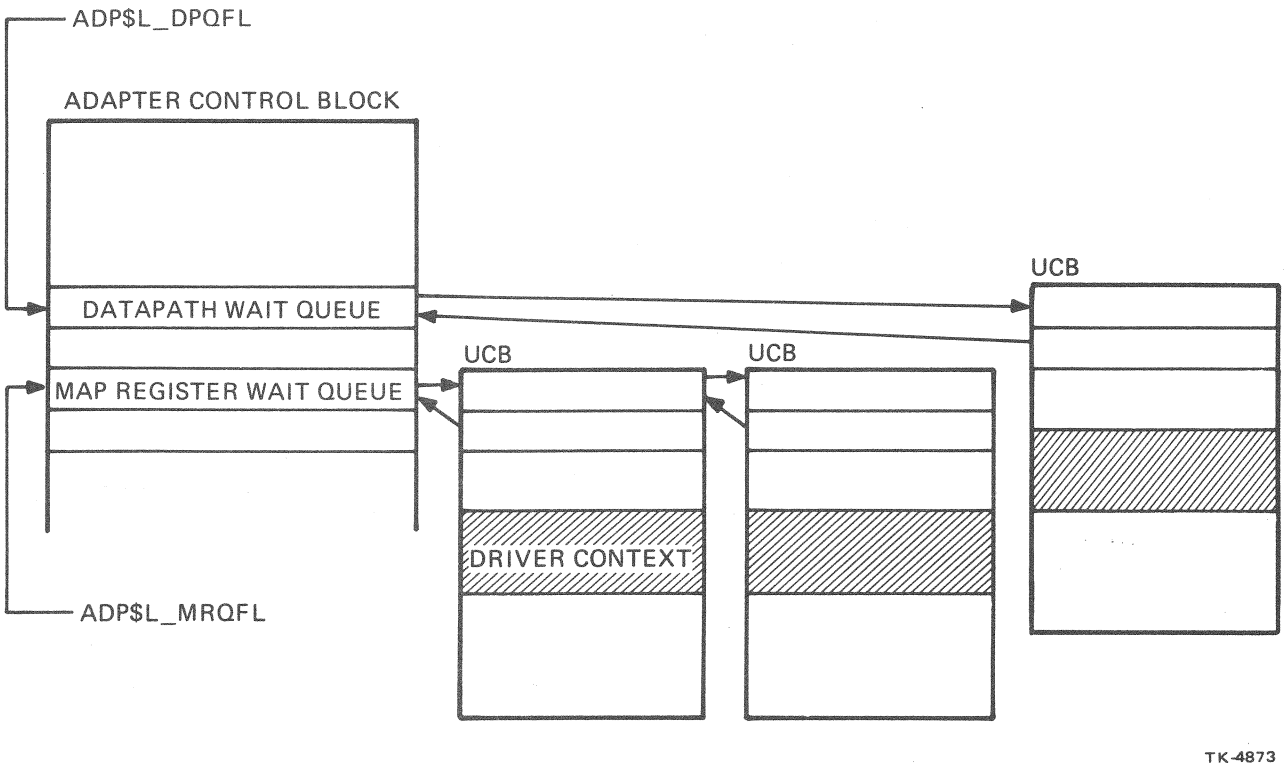


Figure 2-10 Data Paths and Mapping Register Wait Queues
(The ADP contains listheads for two resource wait queues: data paths and mapping registers.)

If there is no data path available, or there is an insufficient number of contiguous mapping registers, the allocation routines store the driver fork process context in the fork block, and place the UCB in the appropriate wait queue.

When an I/O operation completes, the data path and map registers are released by I/O completion routines. At this time, the associated wait queues are examined, and, if not empty, the next UCB is dequeued. Its driver context is restored, and the fork process is resumed.

The ADP is also used in dispatching interrupts for UNIBUS devices to the appropriate CRB for subsequent dispatching to the driver.

DRIVER DISPATCH TABLE (DDT)

The DDT is located via a pointer stored in the appropriate DDB. There is one DDT per device driver. The DDT:

- Contains addresses of device driver entry points.
- Contains the error log/diagnostic register dump routine address and buffer sizes (discussed later).
- Contains a pointer to the Function Decision Table (FDT).

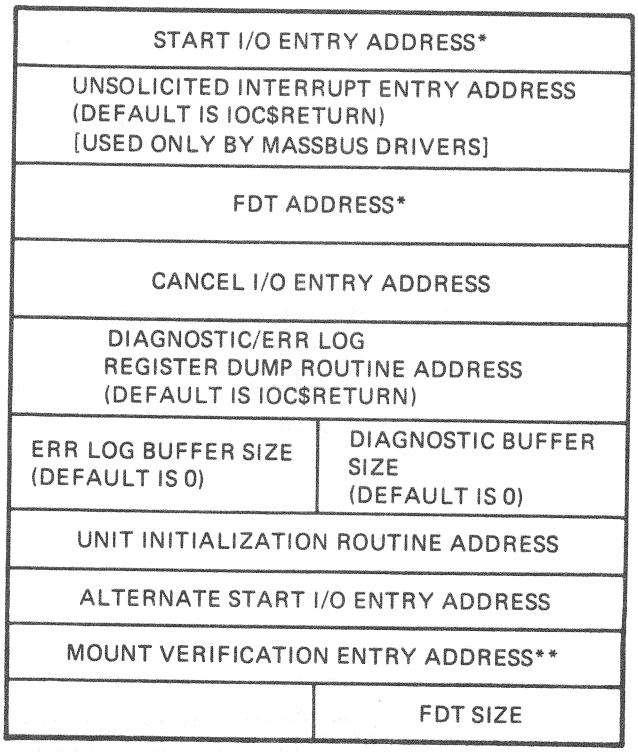


Figure 2-11 Driver Dispatch Table

UNIBUS Device Data Structures

Figure 2-12 shows an example of the DDB, UCB, IDB, and ADP data base structure formed for a UNIBUS configuration consisting of three disks (all RKØ6s on one controller), a line printer, and a card reader. Note that only one CRB and one DDB are needed to describe the three disk units.

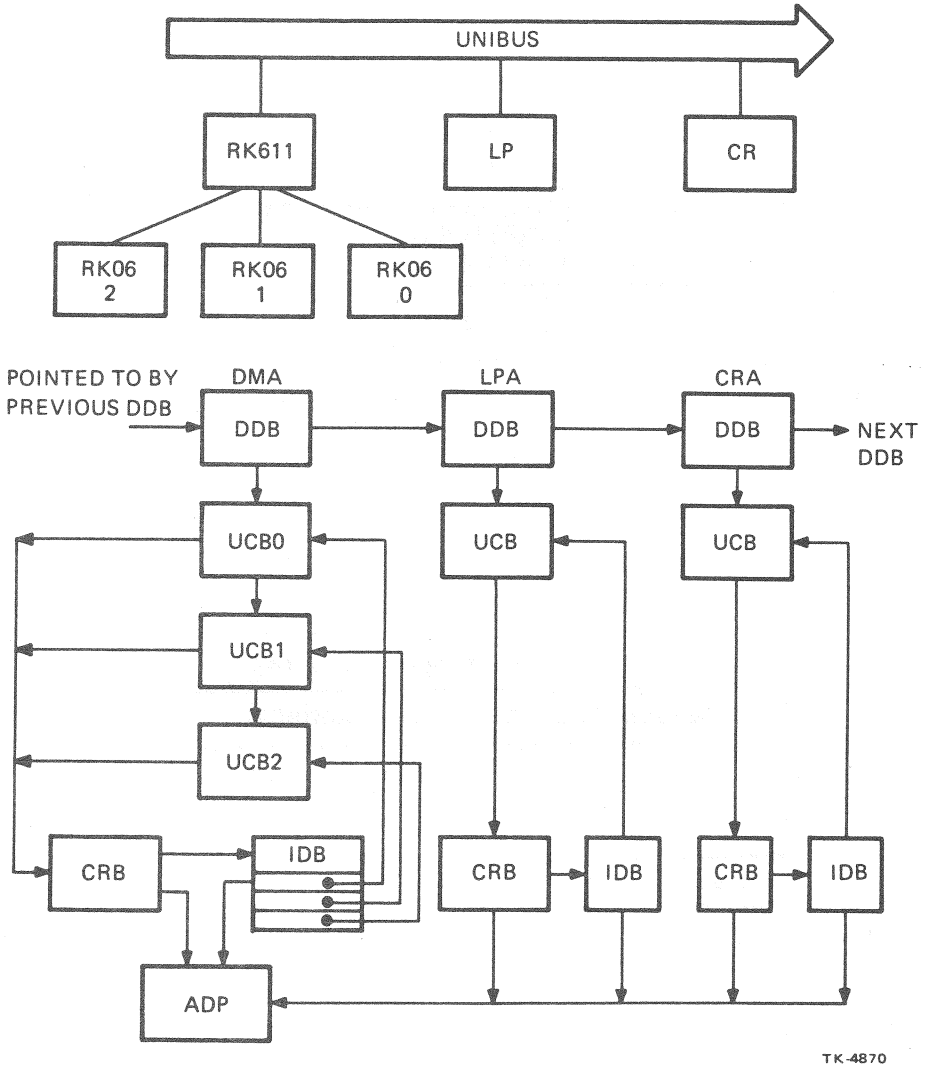


Figure 2-12 UNIBUS Device Data Structures

I/O Data Base for MASSBUS Devices

The I/O data base created for MASSBUS devices differs slightly from the data base created for UNIBUS devices. The data structures that differ are the CRB and IDB. For a single-device controller on the MASSBUS, a CRB and IDB are created for the MASSBUS Adapter (MBA), but not for the single-device controller. The IDB for the MBA contains a pointer to the UCB of each device on a single-device controller, so that the appropriate device can be located when servicing interrupts.

For multidevice controllers (magnetic tape drives on the TMØ3 formatter), a second CRB and IDB are created to describe each multidevice controller. In this case, the MBA IDB points to the multidevice controller's CRB (from which each unit can be located, since the CRB points to the IDB, which points to each unit's UCB). Refer to Figure 2-13 for a sample data base configuration.

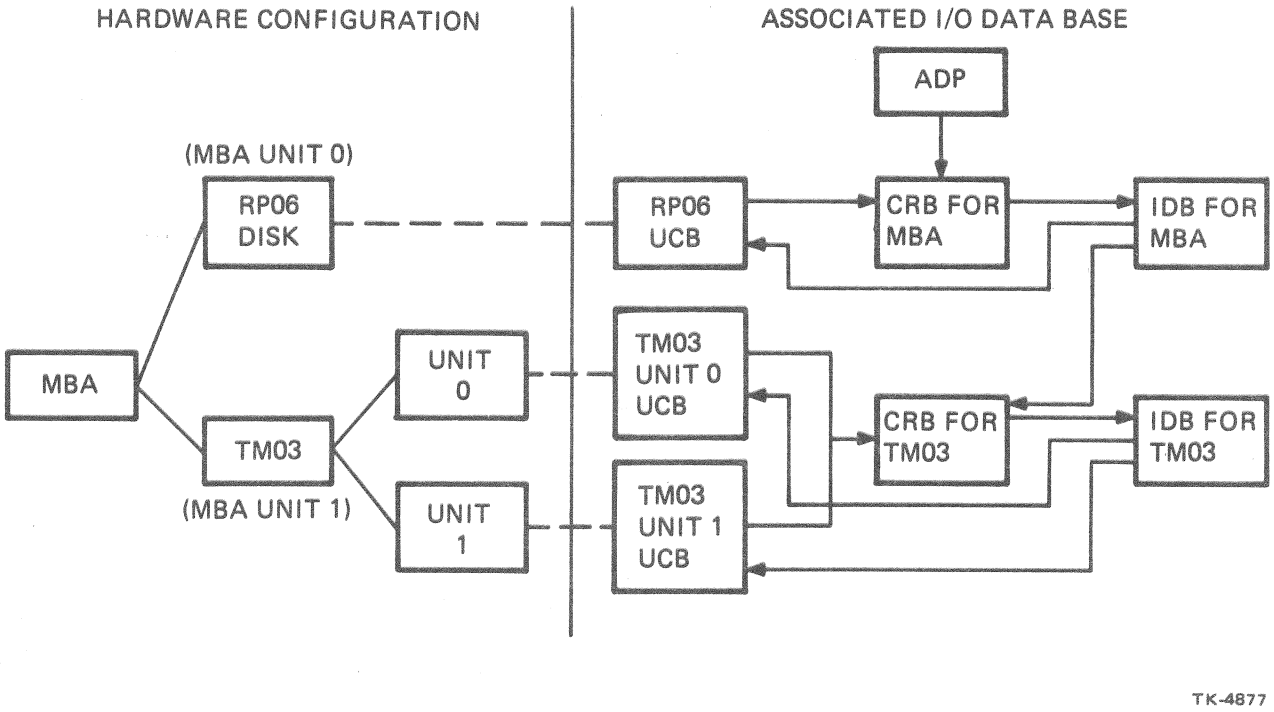


Figure 2-13 Sample MASSBUS Configuration

Illustrates the I/O data base formed for a MASSBUS configuration having one RPØ6 and two magnetic tape drives.

I/O DATA STRUCTURE OVERVIEW

Figure 2-14 illustrates the layout of I/O data base. Different "paths" are taken through the data base for different operations:

1 The $ASSIGN system service follows:

IOC$GL_DEVLIST -> DDBs -> UCBs to initialize CCB

2 $QIO dispatching uses:

chan.# in $QIO -> CCB -> UCB -> DDT -> FDT ->

where the FDT routines preprocess the I/O request.

3 Dispatching interrupts:

device interrupt -> ADP -> CRB -> IDB -> UCB

where the fork context block is restored by the driver's interrupt service routine, and driver execution is resumed.

4 Servicing unsolicited interrupts on MASSBUS:

unsolicited int. -> ADP -> CRB -> IDB -> UCB

where it is determined that the interrupt is unexpected, at which time these pointers are followed:

UCB -> DDB -> DDT -> unsolicited interrupt entry point in driver

I/O DATA STRUCTURE SUMMARY

Table 2-1 describes the various data structures in the I/O data base, including the major functions of the structures, where they are located, and whether or not a driver should reference a particular data structure.

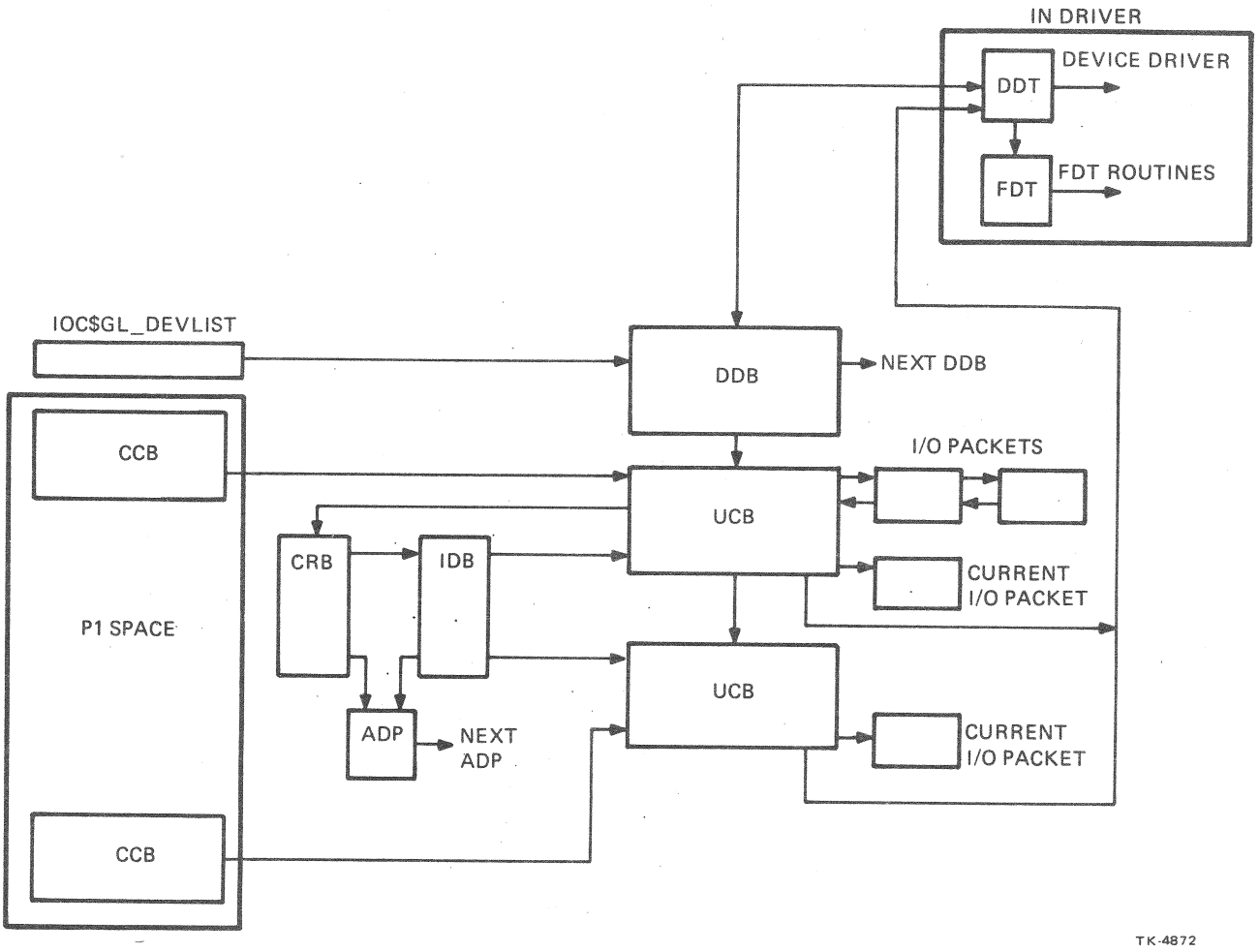


Figure 2-14 Layout of the I/O Data Base

Table 2-1 I/O Data Structure Summary

Name	Function	Creator	Number	Location	Driver Reference
CCB	• Links a process channel number with a device unit.	$ASSIGN Channel Service	1 per assigned channel	Process Control Region	No
IRP	• Carries information that describes a particular I/O request. • Used as an AST control block after I/O is complete.	$QIO System Service	1 per I/O Request	Nonpaged System Memory	Yes
DDB	• Contains information common to all devices of same type. • Points to other data structures.	Driver Writer and Loader	1 per device type (controller)	Nonpaged System Memory	No
UCB	• Contains device characteristics and unit status information. • Used as listhead for IRP queue for the unit. • Provides pointers to DDB, CRB. • Provides storage for device dependent information. • Contains details of current I/O operation in progress. • Contains the Fork context block.	Driver Writer and Loader	1 per device unit	Nonpaged System Memory	Yes
CRB	• Contains busy flag for controller arbitration. • Contains listhead for the queue of UCBs awaiting access to the controller. • Contains code to dispatch interrupts from the controller. • Contains a pointer to IDB.	Driver Writer and Loader	1 per controller or subcontroller	Nonpaged System Memory	Yes

Table 2-1 I/O Data Structure Summary (Cont)

Name	Function	Creator	Number	Location	Driver Reference
WCB	• Contains mapping pointers for virtual -> logical translations.	System	1 per open file	Nonpaged System Memory	No
IDB	• Contains a table of UCB addresses for units under this controller. • Contains a pointer to the current owning unit's UCB. • Contains a pointer to the appropriate device controller CSR.	Driver Loader	1 per controller or subcontroller	Nonpaged System Memory	Yes
ADP	• Contains address of UBA CSR. • Contains a bit map of UBA mapping registers and of UBA data paths.	System	1 per Adapter	Nonpaged System Memory	No
DDT	• Contains addresses of driver entry points. • Contains error log/ diagnostic routine address and buffer size. • Contains pointer to FDT.	Driver Writer and Loader	1 per Driver	Nonpaged System Memory	No
FDT	• Contains legal function mask. • Contains buffered I/O function mask. • Contains function selection mask(s) and address(es) of FDT routines(s).	Driver Writer	1 per Driver	Nonpaged System Memory	No

digital

EY-2278E-MB-0001
Printed in U.S.A.