VAX 9000 Family
XJA Technical Description

Order Number EK-KA90A-TD-001

digital equipment corporation
maynard, massachusetts

DIGITAL INTERNAL USE ONLY

First Edition, May 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Restricted Rights: Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

Copyright © Digital Equipment Corporation 1990

All Rights Reserved.
Printed in U.S.A.

The postpaid Reader's Comment Card included in this document requests the user's critical evaluation to assist in preparing future documentation.

FCC NOTICE: The equipment described in this manual generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference, in which case the user at his own expense may be required to take measures to correct the interference.

The following are trademarks of Digital Equipment Corporation:

BI	KDM	RSTS	VAX FORTRAN
CI	KLESI	RSX	VAX MACRO
DEC	MASSBUS	RT	VAXBI
DECmate	MicroVAX	RV20	VAXcluster
DECUS	NI	RV64	VAXELN
DECwriter	PDP	TA	VMS
DHB32	P/OS	TK	VT
DIBOL	Professional	ULTRIX	Work Processor
DRB32	RA	UNIBUS	XMI
EDT	Rainbow	VAX	
KDB50	RD	VAX C	digital™

® IBM is a registered trademark of International Business Machines Corporation.

® Intel is a registered trademark of Intel Corporation.

™ Hubbell is a trademark of Harvey Hubbel, Inc.

® Motorola is a registered trademark of Motorola, Inc.

This document was prepared and published by Educational Services Development and Publishing, Digital Equipment Corporation.

DIGITAL INTERNAL USE ONLY

Contents

2 JXDI Bus

3 XMI Bus

4 XJA

5 Register Descriptions

Index

Figures

Tables

About This Manual

This manual describes the operation of the VAX 9000 system XJA adapter, the JXDI bus that connects the XJA with the system control unit (SCU), and the XMI bus that connects the XJA to the various I/O adapters.

This manual is a reference manual for Customer Services personnel as well as a training resource for Educational Services.

Intended Audience

The content, scope, and level of detail in this manual assumes that the reader:

- Is familiar with the VAX architecture and VMS operating system at the user level
- Has experience maintaining midrange and large VAX systems

Manual Structure

The manual has five chapters and an index.

- Chapter 1, General Description, provides an overview of the I/O channel. It briefly describes the functions of the JXDI bus, the XJA, and the XMI bus; and the physical makeup of the I/O channel. Also described are interrupts and the various types of I/O transactions. The chapter closes with a discussion of VAX 9000 system I/O space.
- Chapter 2, JXDI Bus, provides a detailed description of the JXDI bus. DMA, CPU, and interrupt transactions are described along with the data formats used for each type of transaction. The chapter includes a detailed description of all the bus signals.
- Chapter 3, XMI Bus, provides a detailed description of the XMI bus, including a description of all the signals transferred on the bus. Read, write, and interrupt transactions are described along with the data formats used for each type of transaction. XMI address mapping is also discussed.
- Chapter 4, XJA, provides a detailed functional description of the XJA adapter. The functional areas of the XJA are identified, followed by a description of DMA, CPU, and interrupt transactions, and how the functional areas operate during the various types of transactions. Each functional area is then described in detail to explain how the area accomplishes its roll in the various transactions. Other areas described are the XJA registers, the XJA clock system, and the add-on self-test (AOST) logic.
- Chapter 5, Register Descriptions, provides a detailed description of the XJA registers. It includes a description of the bits in each register and the function each bit serves.

- The index provides an alphabetical list of topics and subjects described in this manual. An entry with an *f* appended to the page number (for example: XMI bus, card cage slot assignments, 1–8f) indicates a figure reference. An entry with a *t* appended to the page number (for example: XRC, definition of, 1–11t) indicates a table reference.

DIGITAL INTERNAL USE ONLY

1
General Description

This chapter provides an overview of the XJA I/O channel and the types of I/O and interrupt transactions that execute through the channel. The parts of the channel (JXDI bus, XJA, XMI bus) are described along with a brief explanation of how the parts function during the various transactions. A brief discussion of the XJA registers is also included. The latter part of the chapter discusses system address space.

1.1 I/O Configuration

Figure 1–1 is an overview of the VAX 9000 system I/O section. The system can have up to four I/O channels that interface with the I/O control unit (ICUs) in the system control unit (SCU). The SCU can have up to two ICUs, each capable of interfacing with up to two I/O channels.

An I/O channel consists of the following components:

- JXDI bus
- XJA
- XMI bus
- XMI node adapters

1.1.1 JXDI Bus

The JBox-to-XJA data interconnect (JXDI) is a bus that interfaces the ICU with the XMI-to-JBox adapter (XJA). The bus is composed of four cables each with 30 lines. Of the 120 total lines, 107 are signal and 13 are ground. Most of the signal lines are in differential pairs. The JXDI cables are about 4 meters (12 feet) long.

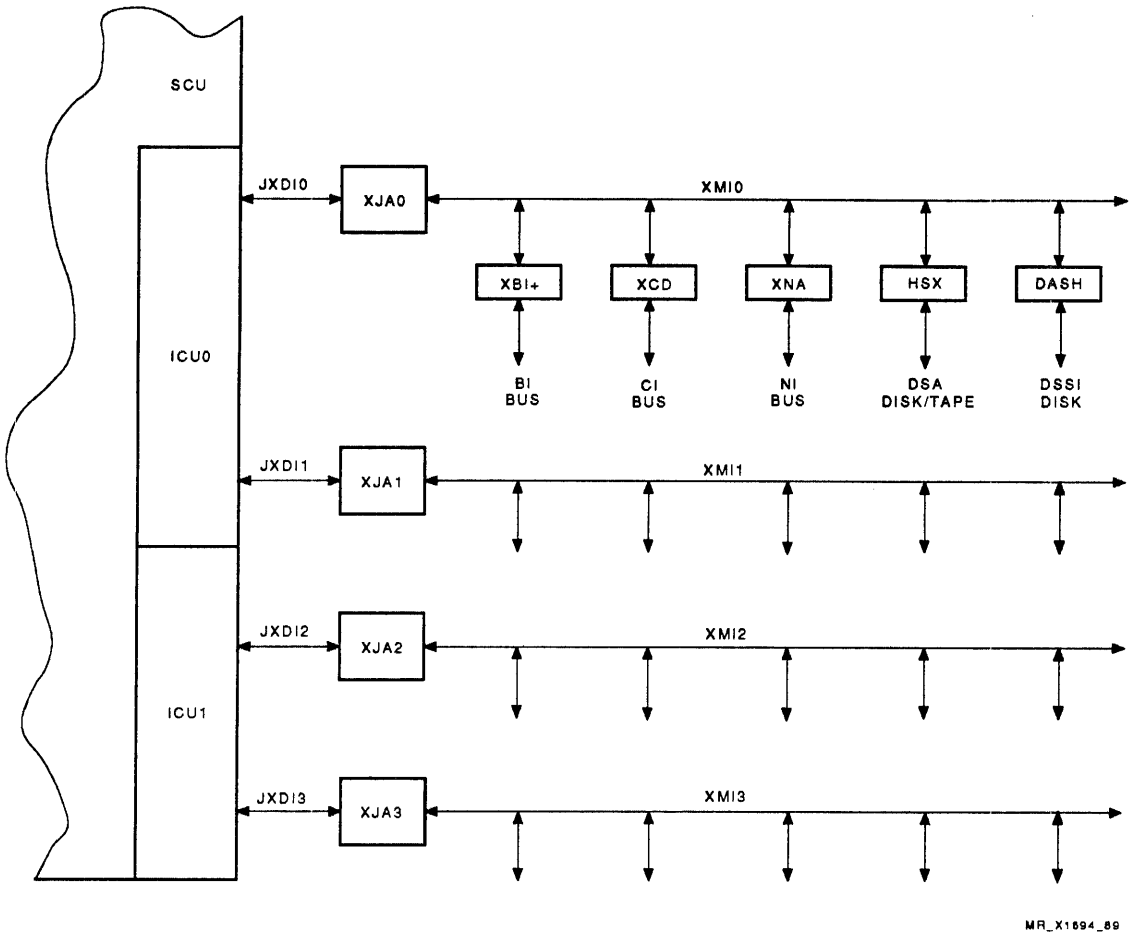


Figure 1–1 System I/O Block Diagram

1.1.2 XJA

The XJA is the interfacing adapter between the SCU and the XMI bus. The XJA:

- Transfers data between the JXDI and the XMI bus
- Formats the data as required for the JXDI and the XMI bus
- Checks data and reports any errors
- Generates interrupts resulting from I/O errors
- Contains the registers required to monitor and control the I/O system

Figure 1–2 is a simplified block diagram of the XJA. The XJA has 12 chips plus some board mounted discrete components. The 12 chips are:

- One XDC chip (XJA data path CMOS gate array)
- Two XDE chips (XJA data path ECL gate array)
- One XCE chip (XJA JXDI control ECL gate array)
- Seven XLATCH chips
- One XCLOCK chip

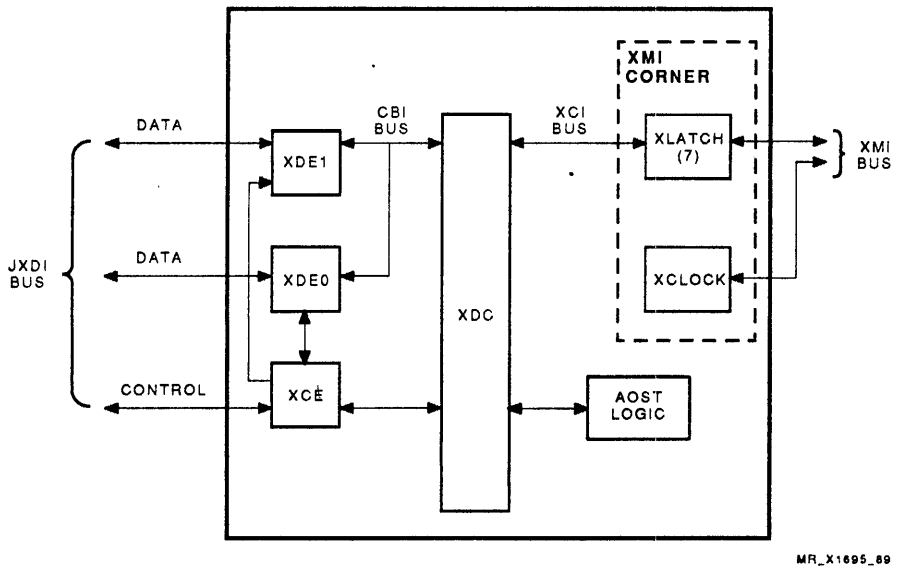


Figure 1–2 XJA Simplified Block Diagram

The XDC chip is an LCA10129 series CMOS gate array. The chip implements LSI logic and contains most of the XJA logic. The XDC chip controls and formats the data flowing between the XMI corner and XDE chips, generates interrupts, and contains all the XJA registers.

The XDE and XCE chips are AMMC Q3500 series bipolar gate arrays. The XDE chips provide bidirectional data paths between the JXDI and the XDC chip, and they perform parity checking of the data received from the JXDI. The XCE chip provides the control interface with the JXDI and controls the data flow through the XDE chips.

The XLATCH chips (DC530) and the XCLOCK chip (DC531) comprise the XMI corner. The XMI corner is a standard XMI interface required of all XMI nodes. The seven XLATCH chips provide a bidirectional data path between the XDC and the XMI bus. The XCLOCK chip generates clocks that are supplied to the XLATCH chips and the XDC chip.

The XJA contains add-on self-test (AOST) logic that interfaces with the XDC chip. The logic provides test inputs that check XJA operation and reports any test failures.

The XJA has two internal buses, the CBI and XCI. The CBI bus (CMOS to bipolar interconnect) is a bidirectional data bus connecting the XDE chips to the XDC chip. The XCI bus (XMI to CMOS interconnect) is a bidirectional data bus connecting the XMI corner to the XDC chip.

1.1.3 XMI Bus

The XMI bus is a standard bus used by the VAX 9000 system as an I/O bus. The XMI bus is a pended, synchronous bus with centralized arbitration. Bus node adapters are housed in an XMI card cage (Figure 1–3) with the higher numbered slots having a higher arbitration priority than the lower numbered slots.

The bus has a clock card (CCARD) module containing the arbitration logic and the clock generation logic for the bus clocks. The XMI bus has a 64-ns cycle.

1.1.4 XMI Node Adapters

The XMI node adapters are all passive adapters. That is, no CPUs interface to the XMI bus through any of these adapters. The VAX 9000 system CPUs are the only CPUs that exist in the entire system configuration.

Five types of adapters are supported on the XMI bus. These are identified in Table 1–1 along with the device to which they interface.

There are some physical and address space limitations on the number and types of adapters that can be used. A summary of these limitations is given in Table 1–2.

Table 1–1 Types of XMI Devices

Device	Description	Adapter Mnemonic	Adapter Module(s)
KFMSA	DSSI[1] disk interface	DASH	T2036
DEMNA	XMI-to-NI adapter	XNA	T2020
CIXCD	XMI-to-CI adapter	XCD	T2080
KDM70	Local DSA disk/tape interface	HSX	T2022 and T2023[2]
DWMJA	XMI-to-JBox adapter	XJA	T1061
DWMBB	XMI-to-BI adapter	XBI+	T2018 (XBIA) and T1043[3](XBIB)

[1]Digital storage system interconnect.

[2]The HSX adapter requires two slots in the XMI card cage.

[3]The T1043 module is in the VAXBI expander cabinet.

Table 1–2 VAX 9000 system I/O Adapter Limitations

Limitation	Reason
Maximum of 12 adapters per card cage	Only 12 physical slots available (Section 1.2).
Maximum of 8 XBI+ adapters per XMI card cage	Maximum of 8 BI units accessible per I/O cabinet (Section 1.2).
Maximum of 14 XBI+ adapters per system	System I/O space has room for only 14 BI windows (Section 1.5.1).

1.2 Physical Configuration

The XJA is an extended T-series module (T1061) that is housed in the XMI card cage located in the I/O cabinet. In the VAX 9000 model 210 system, there is one XMI card cage per I/O cabinet, resulting in a limitation of one XJA I/O channel per VAX 9000 model 210 system. In the VAX 9000 model 400 systems, there are two XMI card cages per I/O cabinet, allowing up to the maximum of four XJA I/O channels per system. The JXDI cables are located in the rear of the cabinets and connect the ICU in the SCU cabinet to the XJA module in the I/O cabinet.

Another physical limitation exists with regard to the XBI+ (XMI-to-BI adapter). Up to eight BI units may be located in two VAXBI expander cabinets (four BI units per cabinet) with one XBI+ adapter required per BI unit. Consequently, no more than eight XBI+ adapters per I/O cabinet can be used. This limits models 410 and 420 to eight XBI+ adapters. In the model 430 and 440, a third and a fourth VAXBI expander cabinet house additional BI units that connect to the XBI+ adapters in the XMI card cages in the second I/O cabinet (Table 1–2).

NOTE
A third BI expander cabinet cannot be used with models 410 and 420 due to cabling limitations. Model 210 has only one VAXBI expander cabinet.

Figure 1–3 shows the XMI card cage. The cage has 14 slots, 2 of which are used by the XJA and the CCARD modules, and the remaining 12 are for other XMI adapter modules. The CCARD module contains the XMI arbitration and clock generation logic and is an integral part of the XMI bus. The CCARD is centrally located in the card cage (slot 7) to distribute the XMI clocks radially to the XMI adapters. This radial distribution minimizes clock skew between the adapters and improves signal integrity. The XJA can fit on the left side of the CCARD but not on the left side of any of the other XMI adapters. Therefore, the XJA is always located in slot 8 beside the CCARD. The only other restriction on placement of the XMI card cage adapter modules is that slot 1 or 14 must be used to implement the XMI_DEFAULT function on the XMI bus (Section 3.2.4). After the first XMI adapter is placed in slot 1 or 14, other adapters may be located in any slot. Adapters in the higher numbered slots have a higher arbitration priority than those in the lower numbered slots.

Each slot in an XMI card cage has a 4-bit, hardwired, node ID number that identifies the slot and the adapter in the slot. XMI adapters match the node ID against selected address bits to determine if an XMI transaction is directed to their node (Sections 1.5.5.1 and 1.5.5.2).

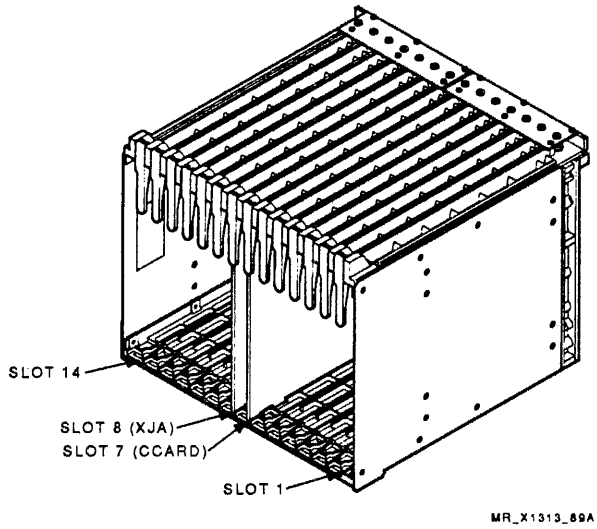


Figure 1–3 XMI Card Cage

1.3 XJA Registers

The 23 registers in the XJA are used for monitoring and controlling XJA operations. The registers and their functions are listed in Table 1–3. The XJA registers fall into two groups: XMI space registers and XJA private registers. The XMI space registers are accessed from the XMI bus. If a system CPU wishes to access an XMI space register, it places its command/address on the XMI bus with the XJA as the addressed XMI adapter. The XJA private registers cannot be accessed from the XMI bus. Only the system CPU can access the XJA private registers and this is done completely within the XJA.

Table 1–3 XJA Registers

Register	Mnemonic	Description
XMI Space Registers		
Device	XDEV	Describes the node device.
Bus error	XBER	Contains a summary of the XMI status and errors.
Failing address	XFADR	Saves the low-order four bytes of a failing XMI command/address.
Failing address extension	XFAER	Saves the high-order four bytes of a failing XMI command/address.
XJA general purpose	XJAGPR	Used for diagnostic testing.
Full system emulation mode control	FAEMC	Controls XJA operation in full system emulation mode.
Add-on self-test status	AOSTS	Contains the results of various tests run by AOST logic.
XJA serial number	SERNUM	Contains the year and week of manufacture, manufacturing plant, and serial number of the XJA.
XJA Private Registers		
Error summary	ERRS	Contains summary of errors detected by the XJA.
Force command	FCMD	Forces XJA transactions for testing purposes.
Interprocessor interrupt source	IPINTRSRC	Identifies the source of interprocessor interrupts.
XJA diagnostic control	DIAG	Controls diagnostic testing of the XJA.
DMA failing address	DMAFADDR	Saves address and length information of a failing DMA or interrupt transaction.
DMA failing command	DMAFCMD	Saves command, mask, and address information of a failing DMA or interrupt transaction.
Error interrupt control	ERRINTR	Disables various error interrupts during diagnostic testing.

Table 1–3 (Cont.) XJA Registers

Register	Mnemonic	Description
Configuration	CNF	Contains the XJA number and node ID, and main memory size and starting address.
XBI ID A	XBIIDA	Contains 4-bit node ID of BI adapters 0 through 7.
XBI ID B	XBIIDB	Contains 4-bit node ID of BI adapters 8 through D(hex).
Error SCB offset	ERRSCB	Contains SCB offset for XJA detected errors at IPL 17(hex).
SCB offset IPL 14(hex)	IDENT4	Contains SCB offset for XMI generated interrupts at IPL 14(hex).
SCB offset IPL 15(hex)	IDENT5	Contains SCB offset for XMI generated interrupts at IPL 15(hex).
SCB offset IPL 16(hex)	IDENT6	Contains SCB offset for XMI generated interrupts at IPL 16(hex).
SCB offset IPL 17(hex)	IDENT7	Contains SCB offset for XMI generated interrupts at IPL 17(hex).

1.4 XJA Transactions

Figure 1–4 is a block diagram of the XJA. Table 1–4 defines the mnemonics of XJA chips, buses, and logic areas used in Figure 1–4 and in the following discussion.

Table 1–4 XJA Mnemonics

Mnemonic	Definition
AOST	Add-on self-test
CBI	CMOS to bipolar interconnect
JXDI	JBox to XJA data interconnect
RCM	Receive control machine
REG	XJA registers
RRF	Receive register file
TCM	Transmit control machine
TRF	Transmit register file
XCE	XJA JXDI control ECL gate array
XCI	XMI bus to CMOS interconnect
XDC	XJA data path CMOS gate array
XDE	XJA data path ECL gate array
XJA	XMI to JBox adapter
XMI	General-purpose memory interconnect
XRC	XMI receive logic

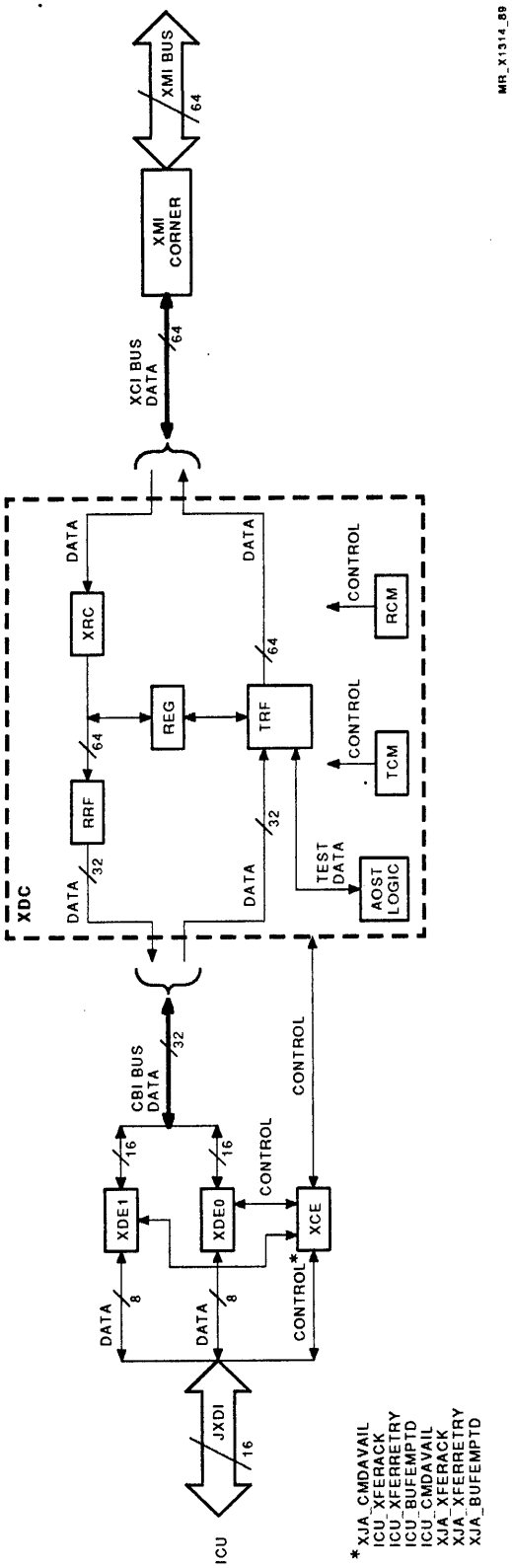


Figure 1–4 XJA Block Diagram

Data packets that flow between the JXDI bus and the XMI bus undergo format, timing, and data length changes as they pass through the XJA. Packets from the XMI bus have data lengths of 64 bits per cycle and are clocked at a 64-ns rate. From the XMI bus, the 64-bit data packets pass through the XMI corner and the XRC, and then into the RRF where the data length is changed to 32 bits per cycle. The 32-bit data cycles are clocked at a 32-ns rate. The 32-bit data packets cross the CBI bus and then split into two 16-bit sections that are applied to the XDE chips. The XDE chips change the data lengths from 16 to 8 bits per cycle clocked at a 16-ns rate. The 8-bit outputs of the XDE chips are combined to form a 16-bit data length per cycle for the JXDI data packets. The 16-bit JXDI data cycles are clocked at the 16-ns rate.

Data packets traversing the XJA in the other direction undergo a similar change in timing and data length. JXDI packets from the ICU have a 16-bit data length and are clocked at a 16-ns rate. The data is split into two 8-bit sections and applied to the XDE chips. In the XDE, the data is assembled into 16-bit data lengths clocked at a 32-ns rate. The 16-bit data outputs from the XDE chips are combined into 32-bit longwords and clocked across the CBI bus at a 32-ns rate. The 32-bit data packet is sent to the TRF where it is changed into a 64-bit data length clocked at a 64-ns rate. The 64-bit data packet is transmitted through the XMI corner to the XMI bus.

Three separate clock systems are used in the XJA. One is a 64-ns clock system to control the 64-bit data interfacing with the XMI bus. A second controls the movement of data from the XJA to the ICU. This clock system includes a 32-ns clock to transfer data from the RRF, across the CBI bus, and into the XDE chips; and a 16-ns clock to transfer data from the XDE to the JXDI. A third clock system controls the movement of data from the ICU to the XJA. This clock system includes a 16-ns clock to transfer data from the JXDI into the XDE, and a 32-ns clock to transfer data from the XDE across the CBI and into the TRF.

Four types of transactions occur within the XJA:

- **DMA (direct memory access)** — This transaction is a read or write of the VAX 9000 system main memory by an XMI device.
- **CPU** — This transaction is a read or write of an I/O register by the VAX 9000 system CPU. The I/O register may be in an XMI device or in the XJA.
- **Interrupts** — This transaction may be initiated by the XJA or by an XMI device. An interrupt initiated by the XJA is transferred over the JXDI to the system CPU for processing. An interrupt initiated by an XMI device is received by the XJA and passed on to the system CPU over the JXDI.
- **AOST (add-on self-test)** — This transaction consists of a test packet injected into the XJA by the AOST logic. The test packet is processed as a CPU transaction and the results are checked in the AOST logic.

1.4.1 DMA Transactions

Figure 1–5 is a flow diagram of a DMA transaction. Refer to the flow diagram and to the XJA block diagram (Figure 1–4) during the following discussion.

DMA transactions are initiated by XMI devices to the VAX 9000 system main memory. The DMA command/address is contained in one 64-bit XMI cycle. If the transaction is a write, the write data follows the command/address cycle. Write transactions can be quadword writes (one XMI cycle of write data) or octaword writes (two XMI cycles of write data).[1] Read transactions can be quadword, octaword, or hexword. A hexword transaction is four XMI cycles of read data.

A DMA request packet on the XMI passes through the XMI corner and is received by the XRC logic. The XRC checks parity and checks that the address is valid. If either of these checks fails, the packet is not acknowledged to the XMI commander (the XMI node that transmitted the command/address packet). If the XRC accepts the packet, it returns an acknowledgment to the XMI commander and notifies the RCM, which then controls the processing of the packet.

The RCM places the packet in a receive buffer in the RRF and notifies the XCE that a packet is ready to be sent to the ICU. The ICU has two receive buffers that receive data from the XJA. The XCE contains an ICU buffer available counter with which it monitors the status of the two ICU receive buffers. If an ICU receive buffer is available, the XCE commands the RRF to unload the packet from the RRF receive buffer. The RRF unloads the packet, assembles it into a 32-bit format, and sends it across the CBI to the XDEs.

The 32-bit data from the RRF is split into two halves with one half going to each XDE. The XCE controls the transfer of data through the XDEs and the disassembly of the data from 16-bit words into 8-bit bytes. At the same time, the XCE asserts XJA_CMDAVAIL to the ICU, informing it that a packet is to follow starting with the next cycle. The 8-bit outputs of the XDEs are combined to form 16-bit words for the JXDI. The packet is sent to the ICU in the 16-bit word format.

The ICU checks parity on the packet data and if an error is detected, it asserts ICU_XFERRETRY to the XCE, commanding the XCE to retry the transfer. The XCE once again unloads the packet from the RRF receive buffer and repeats the transmission process to the ICU. The ICU also asserts ICU_BUFEMPTD to the ICU buffer available counter in the XCE, informing the XCE that the ICU buffer reserved for the packet is still available.

[1] The XJA also supports hexword writes, but the SCU does not.

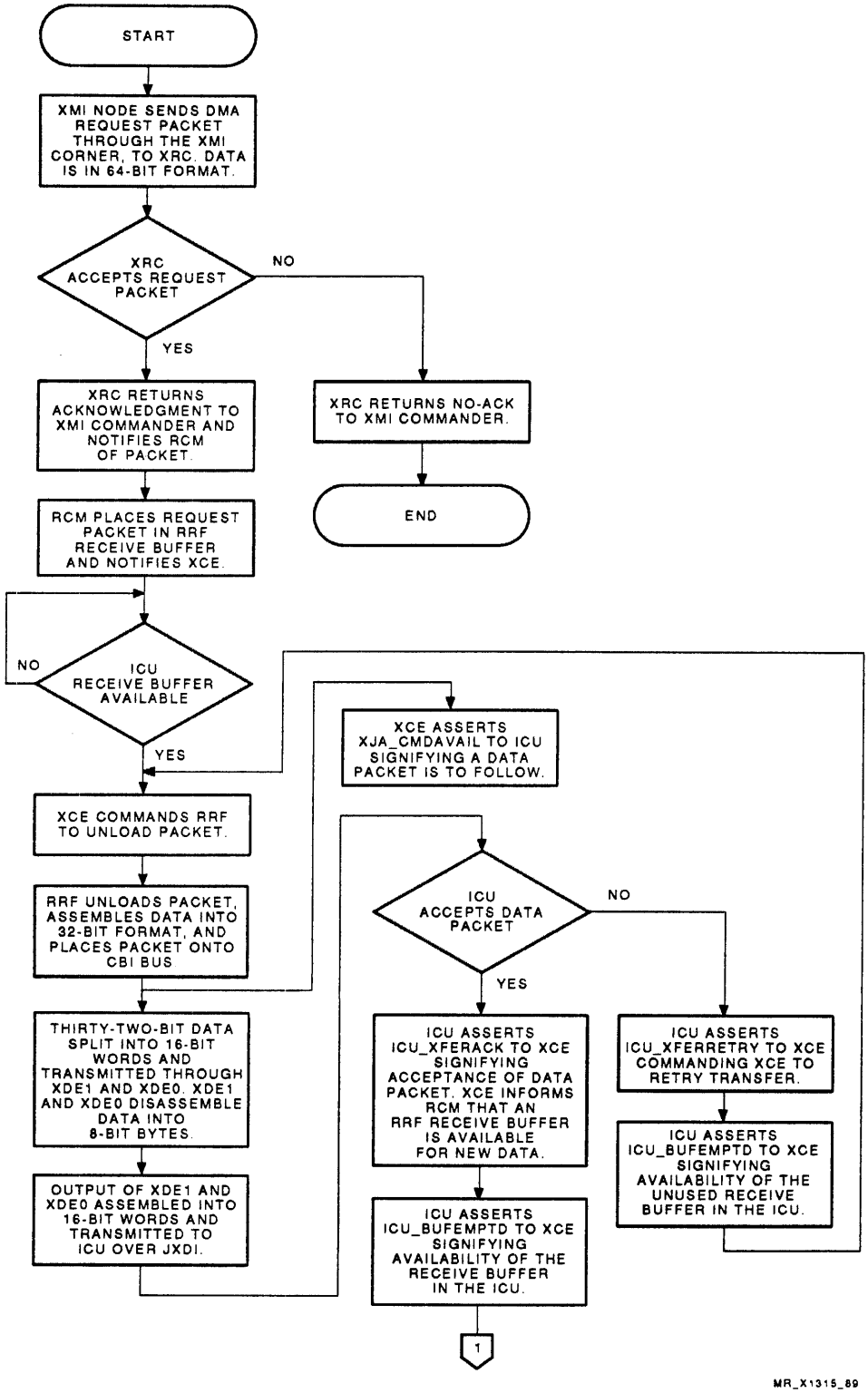


Figure 1–5 (Cont.) Flow Diagram of DMA Transactions

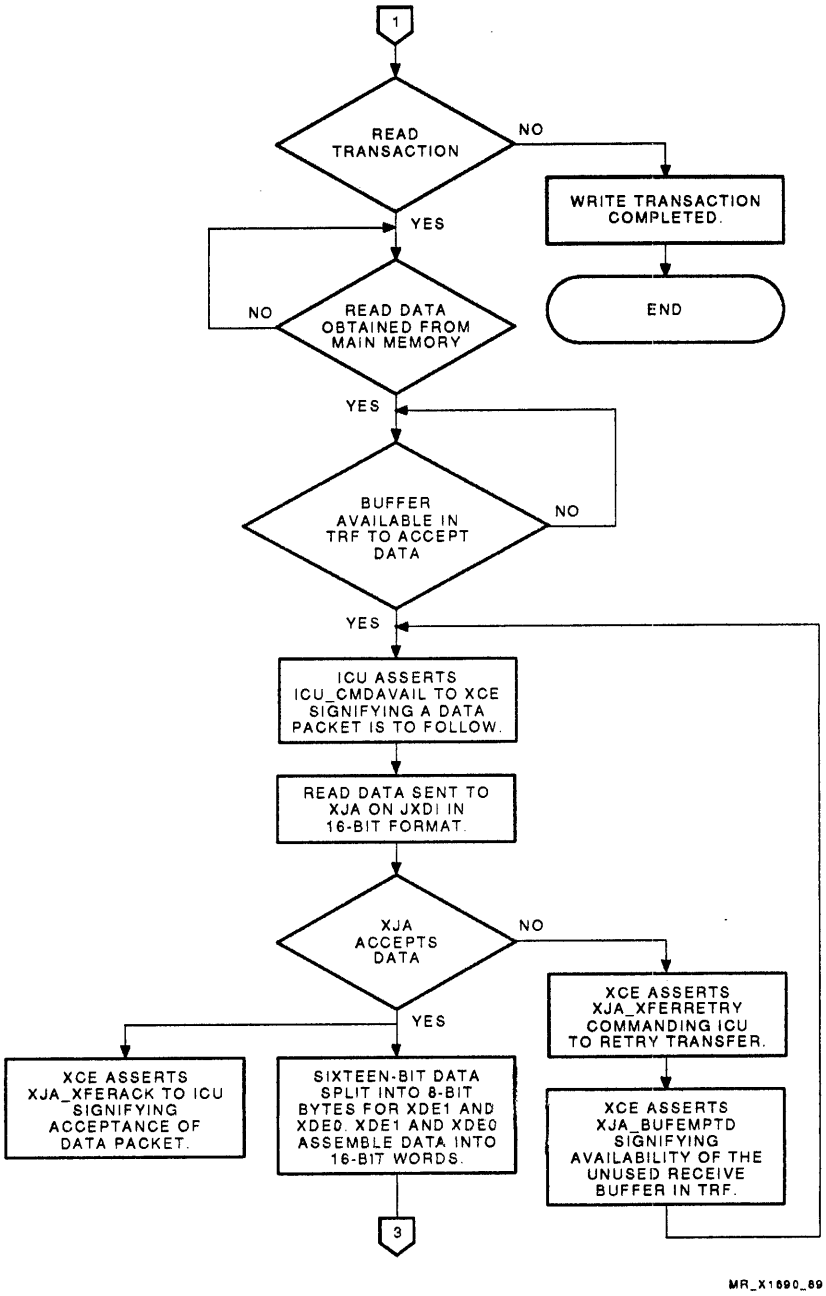


Figure 1–5 (Cont.) Flow Diagram of DMA Transactions

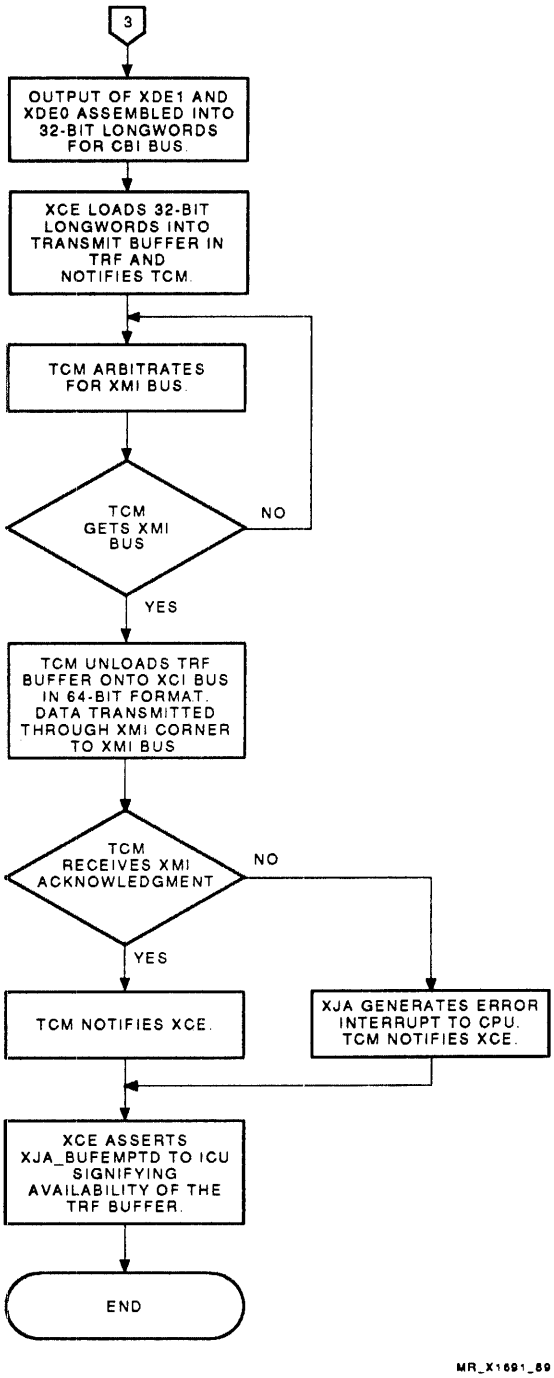


Figure 1–5 Flow Diagram of DMA Transactions

If the ICU finds no parity error in the packet, it asserts ICU_XFERACK to the XCE, indicating that the packet is accepted. The XCE then informs the RCM that the receive buffer in the RRF is available to receive another packet from the XRC.

The RRF has five receive buffers allowing packets to queue up while waiting to be sent to the ICU. The RCM monitors the status of these buffers and suppresses XMI traffic if these buffers become full. Once buffers become available, the RCM removes the XMI suppress command and allows XMI traffic to resume.

When the ICU transfers the packet from its receive buffer to the SCU, it asserts ICU_BUFEMPTD to the ICU buffer available counter in the XCE, informing the XCE that the receive buffer in the ICU is available to receive another packet.

If the DMA transaction was a write, the transaction is completed. If the transaction was a read, the ICU obtains the read data from main memory and then checks if a TRF buffer is available in the XJA to receive the data. The ICU contains an XJA buffer available counter similar to the ICU buffer available counter in the XCE. The counter monitors the status of the XJA buffers that receive the data packets from the ICU. There are three such XJA buffers in the TRF. If a TRF buffer is available, the ICU asserts ICU_CMDAVAIL to the XCE, informing the XCE that a packet is to follow starting with the next cycle. The read data packet is then sent to the XJA in 16-bit word format over the JXDI bus.

The XDE checks parity on the packet data and informs the XCE if an error is detected. If an error is detected, the XCE asserts XJA_XFERRETRY to the ICU commanding it to retry the transfer. The XCE also asserts ICU_BUFEMPTD to the XJA buffer available counter in the ICU, informing the ICU that the TRF buffer reserved for the packet is still available.

If the XDE finds no parity error in the packet, it notifies the XCE, which then asserts XJA_XFERACK to the ICU, indicating that the packet is accepted.

The 16-bit data cycles from the ICU are split into two 8-bit halves, with one half going to each XDE. Under control of the XCE, the XDEs assemble the 8-bit bytes into 16-bit words. The 16-bit outputs of the XDEs are combined into 32-bit longwords, transferred over the CBI, and loaded into one of the three TRF buffers by the XCE. The XCE then notifies the TCM that a packet is in the TRF buffer ready to be transmitted to the XMI bus.

The TCM arbitrates for the XMI bus and, when it wins the bus, unloads the read data packet from the TRF buffer, assembles the data into a 64-bit format, and transmits the data through the XMI corner to the XMI bus where it is returned to the XMI commander that initiated the transaction. When the commander successfully receives the packet, it returns an acknowledgment to the TCM, which informs the XCE of the successful transmission of the packet. The XCE then asserts XJA_BUFEMPTD to the XJA buffer available counter in the ICU, informing it that the TRF buffer is available to receive another packet from the ICU.

If no acknowledgment is received from the XMI commander, the XJA generates an error interrupt to the VAX 9000 CPU. The TCM notifies the XCE that the transaction has ended so that the XCE can assert XJA_BUFEMPTD, signifying the availability of the TRF buffer.

1.4.2 CPU Transactions

Figure 1–6 is a flow diagram of CPU transactions. CPU transactions are initiated by the system CPU or service processor unit (SPU) to read or write a register in an XMI device or in the XJA. As such, the read or write data is longword only.

Basically, the RCM controls the reception of packets from the XMI bus and the TCM controls the transmission of packets to the XMI bus. In executing CPU transactions, control is frequently transferred back and forth between the RCM and the TCM.

CPU transactions (both reads and writes) can be divided into three types. Transactions to:

- XMI nodes (other than the XJA)
- XJA private registers
- XMI space registers (XMI space registers are in the XJA.)

Refer to the flow diagram Figure 1–6 and to the XJA block diagram (Figure 1–4) during the following discussion of CPU transactions.

1.4.2.1 CPU Transactions to XMI Nodes (Excluding XJA)

By means of the XJA buffer available counter described in Section 1.4.1, the ICU determines if a buffer is available in the TRF to accept the CPU command/address packet. If a buffer is available, the ICU asserts ICU_CMDAVAIL to the XCE, informing it that a packet follows, starting with the next cycle. The CPU packet is placed on the JXDI in 16-bit format.

The XDE checks parity on the packet data and informs the XCE if an error is detected. If an error is detected, the XCE asserts XJA_XFERRETRY to the ICU, commanding it to retry the transfer. The XCE also asserts ICU_BUFEMPTD to the XJA buffer available counter in the ICU, informing the ICU that the TRF buffer reserved for the packet is still available.

If the XDE finds no parity error in the packet, it reports this information to the XCE, which then asserts XJA_XFERACK to the ICU, indicating that the packet is accepted.

The 16-bit data cycles from the ICU are split into two 8-bit halves, with one half going to each XDE. Under control of the XCE, the XDEs assemble the 8-bit bytes into 16-bit words. The 16-bit outputs of the XDEs are combined into 32-bit longwords, transferred over the CBI, and loaded into one of the three TRF buffers by the XCE. The XCE then notifies the TCM that a packet is in the TRF buffer and is ready to be processed.

The TCM arbitrates for the XMI bus and, when it wins the bus, unloads the TRF buffer, assembles it into 64-bit format, and transmits it through the XMI corner to the XMI bus. When the target node successfully receives the packet, it returns an acknowledgment to the TCM, which informs the XCE of the successful packet transmission. The XCE then asserts XJA_BUFEMPTD to the XJA buffer available counter in the ICU, informing it that the TRF buffer is now available to receive another packet from the ICU.

If no acknowledgment is received from the target node, the XJA generates an error interrupt to the VAX 9000 CPU. The TCM notifies the XCE that the transaction has ended so that the XCE can assert XJA_BUFEMPTD, signifying the availability of the TRF buffer. The transaction is then aborted.

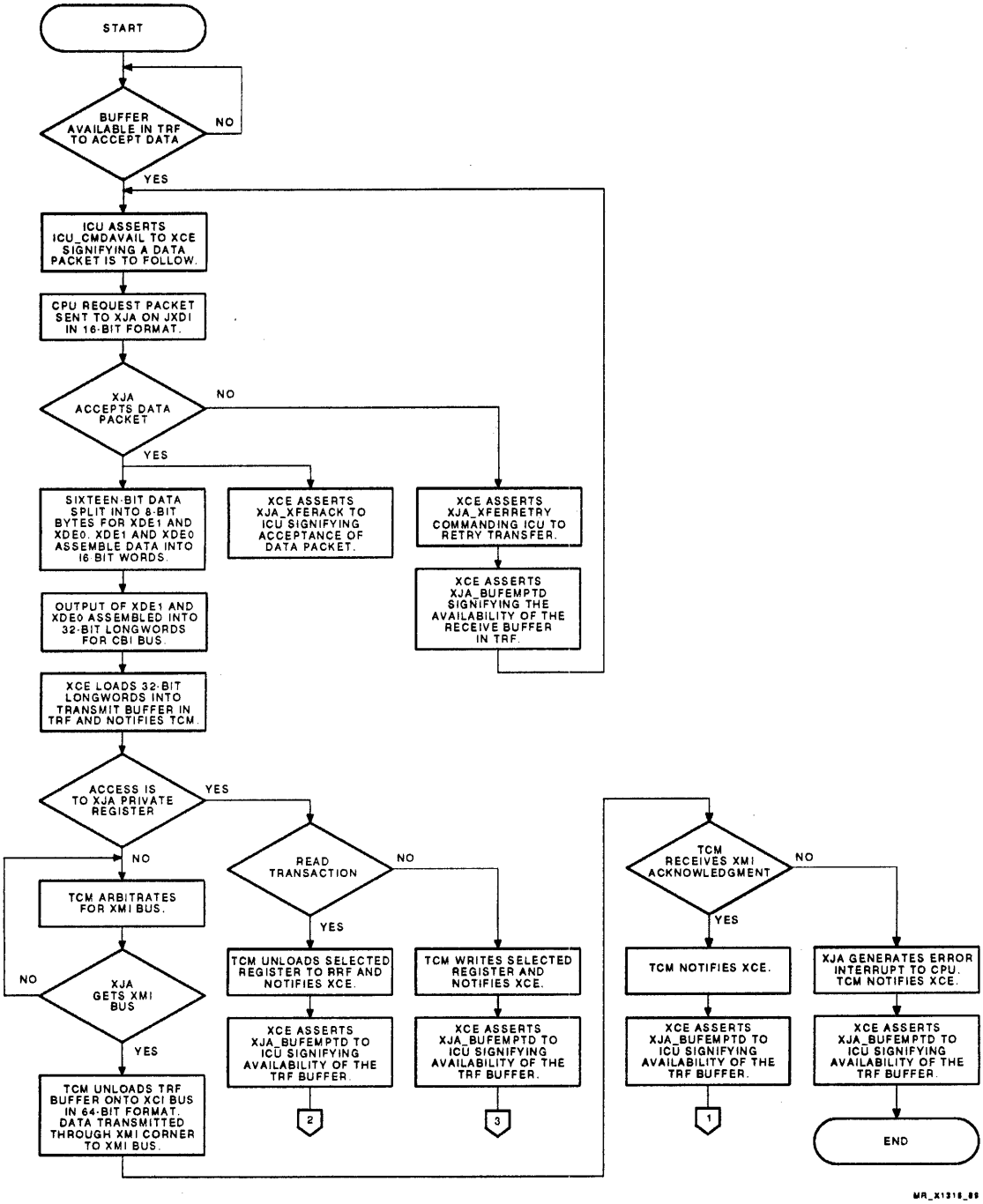


Figure 1–6 (Cont.) Flow Diagram of CPU Transactions

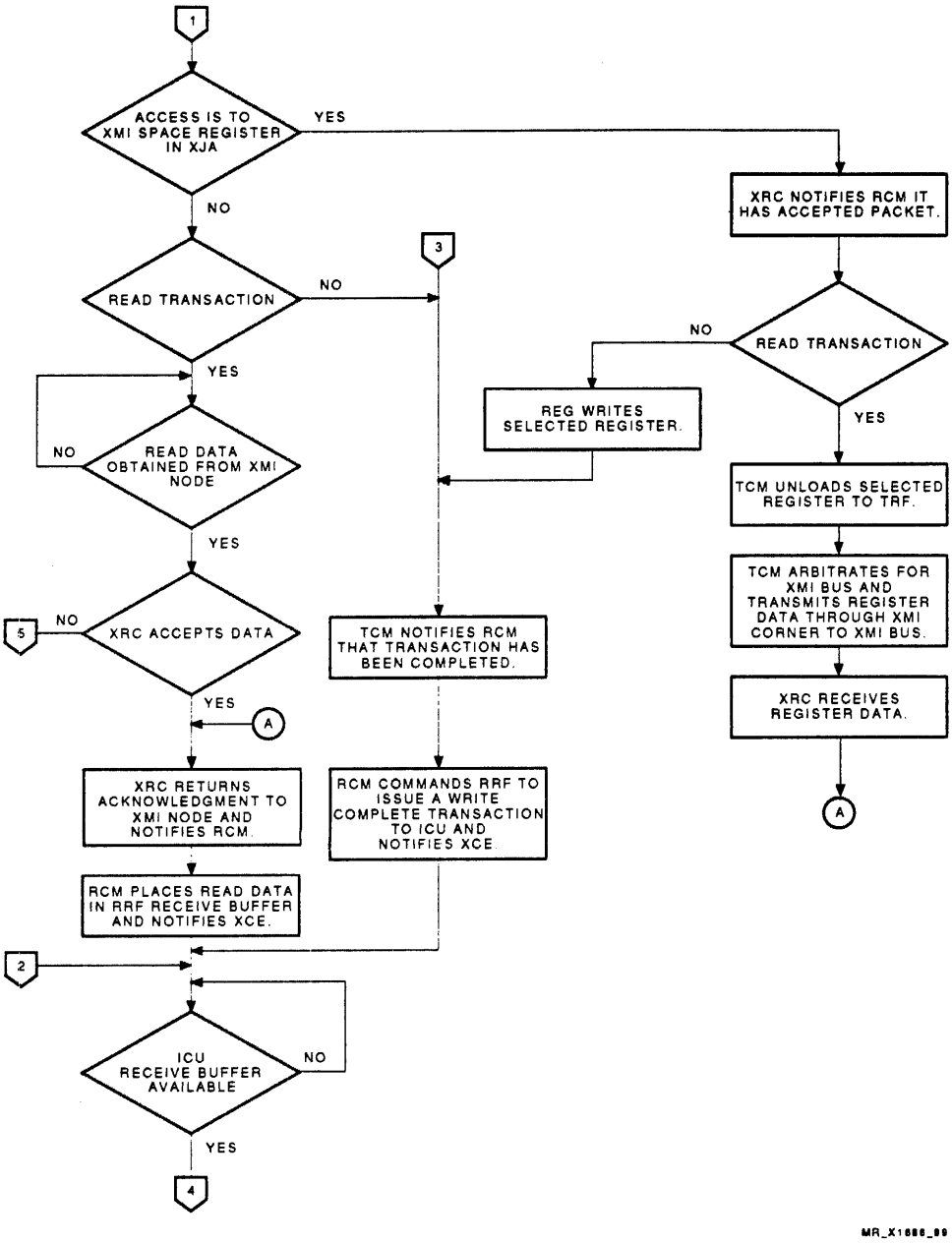


Figure 1–6 (Cont.) Flow Diagram of CPU Transactions

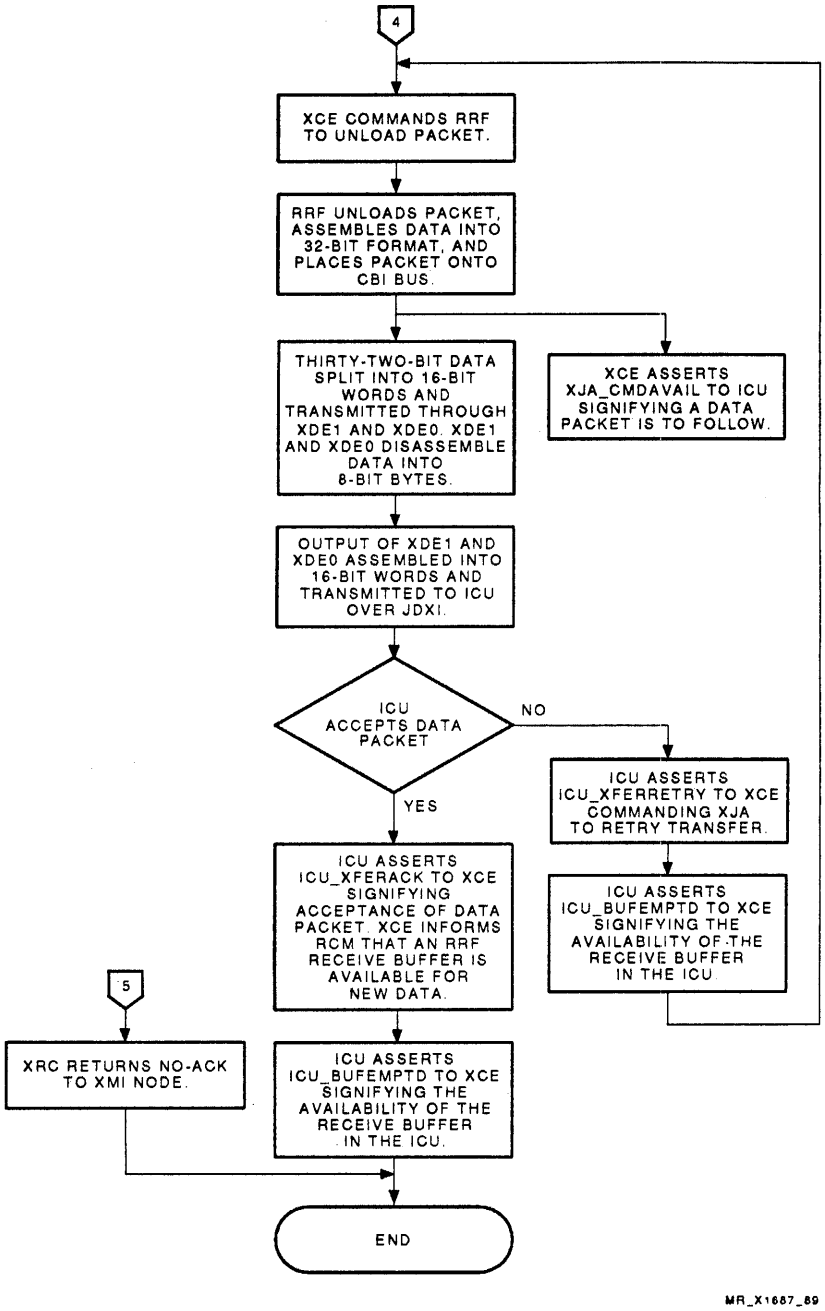


Figure 1–6 Flow Diagram of CPU Transactions

For a write transaction, a write complete packet must now be sent to the ICU. Unlike a DMA write transaction, a CPU write transaction requires that a write complete packet be returned to the transaction commander (the CPU), informing the commander that the transaction is complete. This packet is necessary because only one CPU transaction can be outstanding at a time. When the CPU initiates a transaction, it must know that the transaction is complete before it can initiate another transaction. The CPU knows a transaction is complete when it receives the read data for a read transaction or a write complete packet for a write transaction.

To initiate the write complete transaction, the TCM notifies the RCM that the write data is successfully transmitted to the XMI target node. The RCM commands the RRF to assemble the write complete packet and notifies the XCE that the packet is to be sent to the ICU.

For a read transaction, the read data packet from the XMI target node is received through the XMI corner and checked by the XRC. If the packet is not accepted by the XRC, a no-ack is returned to the target node and the transaction is aborted. If the packet is accepted, the XRC returns an acknowledgment to the XMI node and notifies the RCM. The RCM loads the packet into a receive buffer in the RRF and notifies the XCE.

The XCE, being notified of either a read data packet or a write complete packet in the RRF, checks for the availability of a receive buffer in the ICU by means of its ICU buffer available counter. If the XCE finds that a receive buffer is available, it commands the RRF to transfer the packet to the ICU. The RRF unloads the packet, assembles it into 32-bit format, and sends the packet across the CBI to the XDEs.

The 32-bit data from the RRF is split into two halves, with one half going to each XDE. The XCE controls the transfer of data through the XDEs and the disassembly of the data from 16-bit words into 8-bit bytes. At the same time, the XCE asserts XJA_CMDAVAIL to the ICU, informing the ICU that a packet is to follow, starting with the next cycle. The 8-bit outputs of the XDEs are combined to form 16-bit words for the JXDI. The packet is sent to the ICU in the 16-bit word format.

The ICU checks parity on the packet data and if an error is detected, it asserts ICU_XFERRETRY to the XCE, commanding the XCE to retry the transfer. The XCE once again unloads the packet from the RRF receive buffer and repeats the transfer process to the ICU. The ICU also asserts ICU_BUFEMPTD to the ICU buffer available counter in the XCE, informing the XCE that the ICU buffer reserved for the packet is still available.

If the ICU finds no parity error in the packet, it asserts ICU_XFERACK to the XCE, indicating that the packet is accepted. The XCE then informs the RCM that the receive buffer in the RRF is available to receive another packet from the XRC.

When the ICU transfers the packet from its receive buffer to the SCU, the ICU asserts ICU_BUFEMPTD to the ICU buffer available counter in the XCE, informing the XCE that the receive buffer in the ICU is available to receive another packet.

1.4.2.2 CPU Transactions to XJA Private Registers

Figure 1–6 shows a CPU transaction to an XJA private register. The transaction is identical to a CPU transaction to an XMI node up to where the CPU request packet is loaded into the TRF buffer and it is determined that the access is to an XJA private register (third diamond in the flow diagram).

If the CPU request is to write the private register, the TCM selects the register, loads it with the write data, then notifies the XCE, which asserts XJA_BUFEMPTD back to the ICU. XJA_BUFEMPTD informs the XJA buffer available counter in the ICU that the TRF buffer is available for another packet.

The TCM also notifies the RCM that the write transaction is complete. The RCM then commands the RRF to assemble a write complete packet for transfer to the ICU, and notifies the XCE that a packet is to be transferred to the ICU. The packet is transferred to the ICU in the same manner as a CPU read of an XMI node.

If the CPU request is to read the private register, the TCM selects the register, unloads the register data to the RRF, then notifies the XCE, which asserts XJA_BUFEMPTD back to the ICU. XJA_BUFEMPTD informs the XJA buffer available counter in the ICU that the TRF buffer is available for another packet. The read data return packet is transferred to the ICU in the same manner as for a CPU read of an XMI node.

1.4.2.3 CPU Transactions to XMI Space Registers

Figure 1–6 shows a CPU transaction to an XMI space register in the XJA, which is identical to a CPU transaction to an XMI node up to where the CPU request packet is transmitted onto the XMI bus (sixth diamond in the flow diagram).

In this case, the XJA is the target node. When the XRC receives the request packet, it notifies the RCM, which determines if the request is to read or write the XMI space register. If the request is to write an XMI space register, the XRC notifies REG, which selects the register and loads it with the write data. The RCM then commands the RRF to assemble a write complete packet for transfer to the ICU, and notifies the XCE that a packet is to be transferred to the ICU. The packet is transferred to the ICU in the same way as for a CPU read of an XMI node.

To request a read of an XMI space register, the RCM notifies the TCM, which selects the register, unloads the register data into the TRF, and then arbitrates for the XMI bus. When the TCM wins the XMI bus, it transmits the register data packet through the XMI corner to the XMI bus. The register data packet is then received by the XRC just as any read data packet is received from any XMI node. The packet is processed and sent to the ICU in the same way as any other XMI read return packet.

1.4.3 Interrupt Transactions

Figure 1–7 is a flow diagram of interrupt transactions within the VAX 9000 I/O subsystem. Interrupts within the VAX 9000 I/O subsystem are generated by the XJA or by a device on the XMI bus.

1.4.3.1 XJA Interrupts

XJA generated interrupts result from a data or protocol error on the JXDI, on the XMI bus, or within the XJA. Some XJA errors are classified as fatal, that is, operation of the XJA is unpredictable and the XJA may not be able to respond to CPU commands. If the error is nonfatal, operation of the XJA is predictable and the XJA can respond to CPU commands; however, some CPU transactions may fail.

If the error detected by the XJA is fatal, the XJA asserts an XJA_FATALERR line on the JXDI. The SCU responds by interrupting the system CPU at an IPL of 1D(hex). The CPU then proceeds to service the interrupt by executing the appropriate service routine.

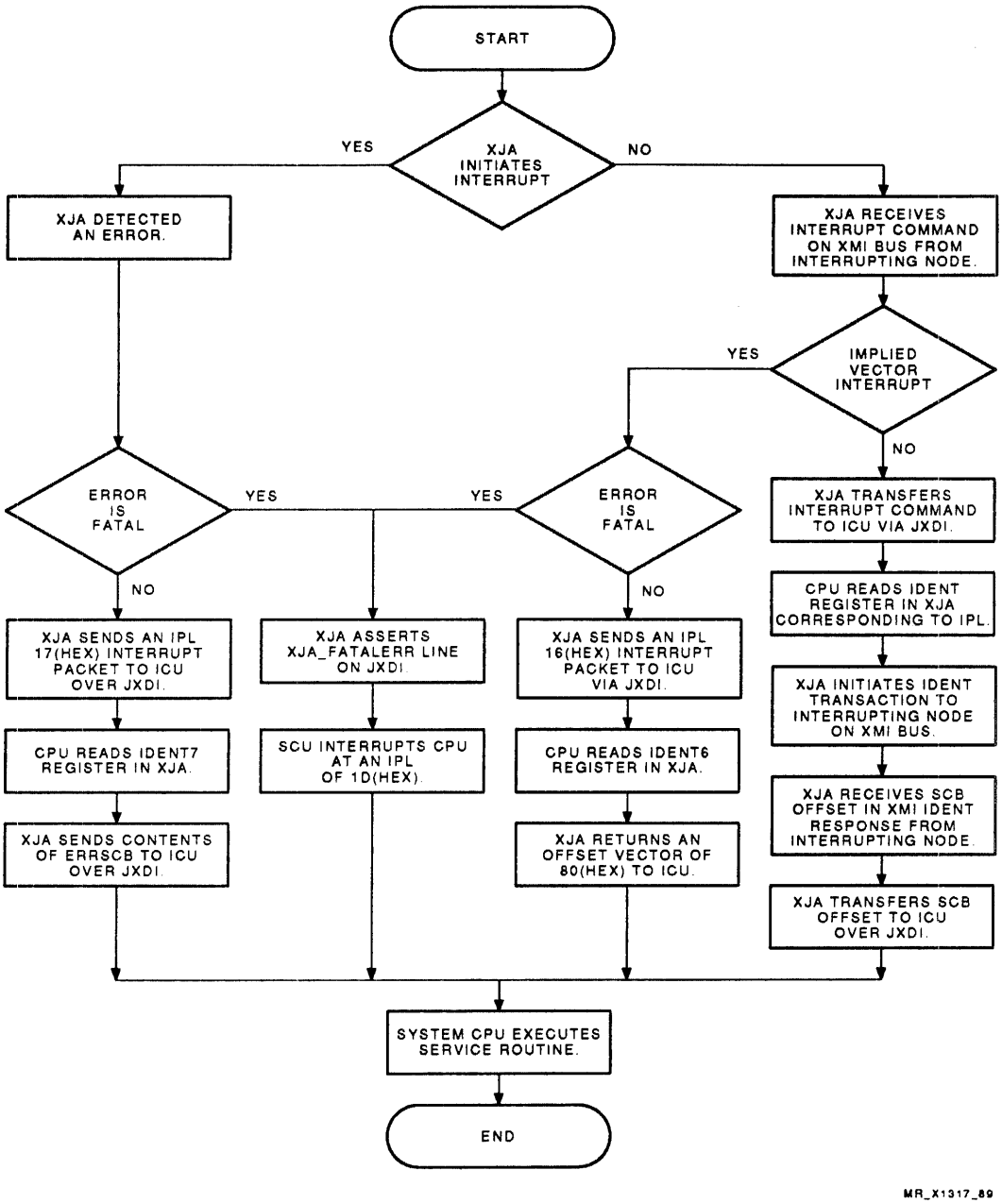


Figure 1–7 Flow Diagram of Interrupt Transactions

If the error detected is nonfatal, the XJA sends an interrupt packet to the ICU. The interrupt packet is a one-cycle packet specifying an IPL of 17(hex)[1]. The CPU responds by reading the system control block (SCB) offset IPL register in the XJA, corresponding to the IPL of the interrupt (IDENT7 register). The IDENT7 register is read using a CPU read transaction. Reading the IDENT7 register while an XJA-detected error is pending causes the XJA to return the contents of the error SCB offset register (ERRSCB) to the ICU. The ERRSCB register contains the vector required by the CPU to locate the appropriate service routine. When the system CPU receives the SCB offset, it proceeds to service the interrupt.

1.4.3.2 XMI Interrupts

XMI interrupts are generated by an XMI device requiring service by the system CPU. XMI interrupts fall into two classes: normal interrupts and implied vector interrupts. Normal XMI interrupts require communication between the XJA and the interrupting device to obtain the SCB offset vector required to service the interrupt. In an implied vector interrupt, the SCB offset (vector) is implied by the type of interrupt, therefore, no communication is required with the interrupting device.

If the XMI interrupt is a normal interrupt, the XJA sends an interrupt packet to the ICU, specifying the IPL of the interrupt. The CPU responds by reading the contents of the SCB offset IPL register (IDENT register) in the XJA, corresponding to the IPL of the interrupt. The XJA responds to the CPU read request by initiating an IDENT (identify) transaction to the interrupting node on the XMI bus. The interrupting node returns an IDENT response containing the SCB offset vector, to the XJA. The XJA passes the offset vector to the CPU as read data return for the CPU read transaction. The CPU then uses the vector to locate the required service routine.

If the XMI interrupt is an implied vector interrupt, the XJA determines whether the interrupt is fatal or nonfatal. If the interrupt is fatal, the XJA_FATALERR line on the JXDI is asserted, causing the SCU to interrupt the CPU at an IPL of 1D(hex). The CPU then proceeds to service the interrupt.

If the interrupt is nonfatal, the XJA transfers the interrupt to the CPU by sending an interrupt packet at an IPL of 16(hex) to the ICU. The CPU responds by reading the IDENT register corresponding to the interrupting IPL (IDENT6 register). When the IDENT6 register is read while an implied vector interrupt is pending, the XJA returns an offset vector of 80(hex), which the CPU uses to locate the appropriate service routing. Note that an XMI IDENT transaction was not required for the implied vector interrupt.

[1] XJA-detected nonfatal interrupts are always at an interrupt priority level (IPL) of 17(hex). See Section 4.9 on I/O interrupts.

1.4.4 Add-On Self-Test

The add-on self-test (AOST) logic (Figure 1–4) is a built-in self-test feature that aids in checking the XJA logic and buses. It is not a 100% check of XJA functionality. It is used in association with XJA diagnostics and power-up tests to ensure reliable operation of the XJA.

In AOST mode, the AOST logic inserts a test packet into the TRF. The test packet is a CPU read or write of an XJA register. The packet is processed by the XJA and the response data is sent through the XDE as it would be for a normal read data return transfer. However, in AOST mode, the output of the XDE is not transferred to the JXDI. The response data is looped back into the XDE and returned to the TRF, where it is transferred to the AOST logic. The logic checks the response data and reports any errors.

1.5 System Address Space

Figure 1–8 shows the VAX 9000 address space. Thirty-four address bits are used to specify 15.5 Gbytes of main memory (0 0000 0000 to 3 DFFF FFFF) and 512 mbytes of I/O space (3 E000 0000 to 3 FFFF FFFF). DMA transactions to main memory are checked in the XRC for a valid memory space address. DMA addresses outside the correct range are not acknowledged. CPU transactions to I/O space are addressed to the 512 Mbyte I/O space region.

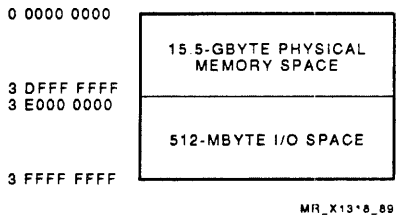


Figure 1–8 System Address Space

1.5.1 System I/O Space Allocation

System I/O space divides into three regions:

XMI node space
BI window space
XJA private register space

Figure 1–9 is a breakdown of the VAX 9000 I/O space. The first four 8-Mbyte segments are designated as XMI node space for the four XMI I/O channels. Each segment has sixteen 512-Kbyte address slots for the XMI adapters (including XBI+ adapters). Address slots 0 and 15 are not used. The remaining 14 address slots correspond to the 14 physical slots in the XMI card cage (Figure 1–3). Address slot 7 is not used as physical slot 7 is taken by the XMI CCARD. (The CCARD is not a node on the XMI bus.) Address slot 8 (node 8) is used for the XJA. This leaves 12 address slots (1 through 6 and 9 through 14) available for XMI adapters.

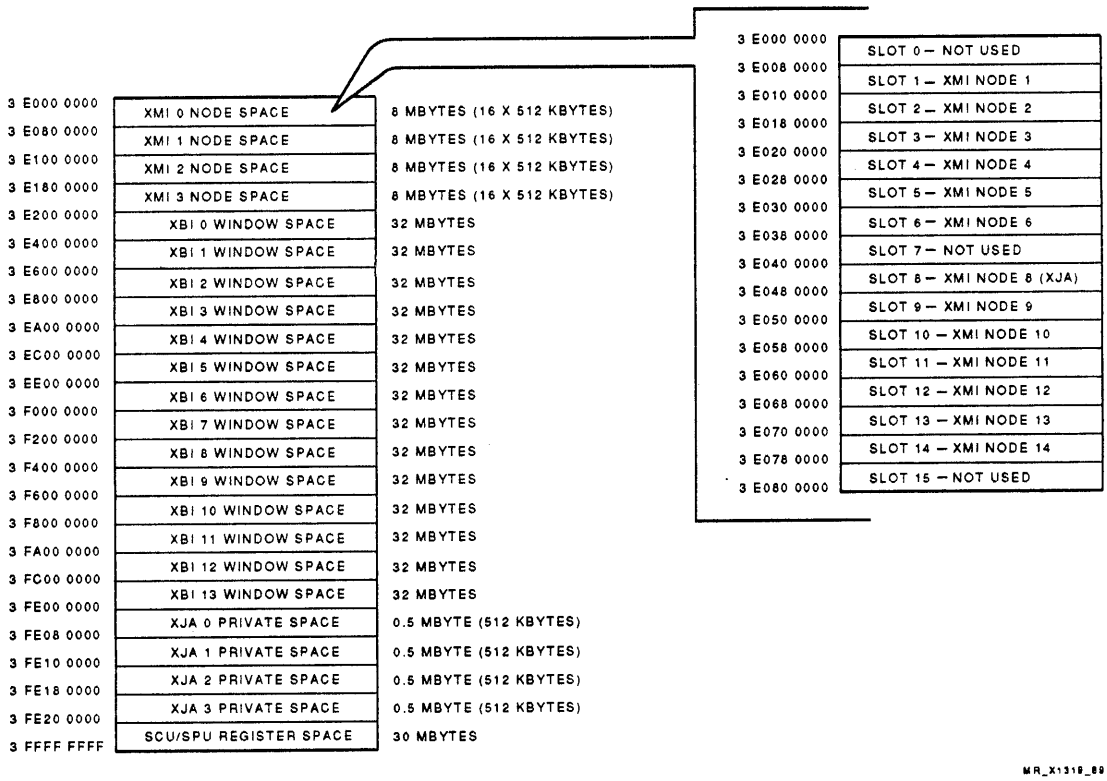


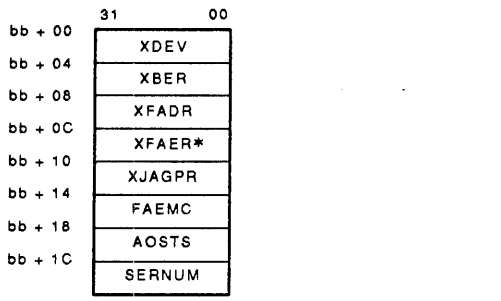
Figure 1–9 System I/O Space Allocation

Located within each of the adapter address slots are the adapters XMI space registers. Figure 1–10 shows the address of the eight XMI space registers. The base block (bb) address is the address of the first XMI node space segment (3 E000 0000) plus 80 0000 for each XMI node segment, plus 8 0000 for each XMI node adapter. The register addresses are the bb address plus the address specified by bits [04:02].

Following the XMI node space is the XBI+ adapter space (windows). Space is allocated for 14 XBI+ adapters. The BI window segments are 32 Mbytes.

Following the BI window space, are four 512-Kbyte address segments for the XJA private registers.

The final 30 Mbytes of I/O space is allocated to SCU and SPU registers. These registers are located in the SCU and SPU, respectively, therefore, access to these registers is not through the VAX 9000 I/O channels.



bb = 3 E000 0000 + (XJA NUMBER X 80 0000) + (XMI NODE NUMBER X 8 0000)
*XFAER IS ALSO ACCESSED AT ADDRESS bb + 2C.

MR_X1320_89

Figure 1–10 Address of XMI Space Registers

1.5.2 XJA Private Register Space

CPU access to the XJA private registers does not involve an XMI transaction. The XJA recognizes the address as being XJA private space and proceeds to access the specified register directly within the XJA. The address specifies the XJA channel as well as the register itself. Figure 1–11 shows the address of the 15 XJA private registers. The bb address is the address of the first XJA private register space segment (3 FE00 0000) plus 8 0000 for each XJA private register segment. The register addresses are the bb address plus the address specified by bits [06:02].

The SCU performs the function of determining which XJA is being referenced by the CPU request. Each XJA channel receives only the CPU requests for its own private registers. This function is described in more detail in Section 1.5.5.

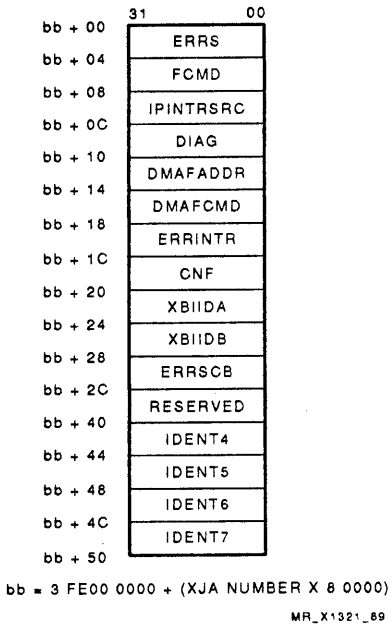


Figure 1–11 Address of XJA Private Registers

1.5.3 XMI I/O Space Allocation

The XMI bus is a standard bus used in system configurations other than the VAX 9000 system. As such, the XMI bus must conform to XMI standard protocol. One of the XMI standards is a 40-bit addressing scheme with the MSB (bit [39]) dividing the XMI space into memory space (bit [39] = 0) and I/O space (bit [39] = 1). Figure 1–12 is a breakdown of XMI I/O space for XMI channel 0, starting at address 80 0000 0000 (bit [39] = 1).

XMI protocol designates the first 24 Mbytes of XMI I/O space as XMI private space. This space is not used by the VAX 9000 system.

The next 8 Mbytes is the node space for the XMI adapters. The node space has 16 512-Kbyte address slots comparable to the 16 address slots of Figure 1–9.

The node space is followed by fifteen 32-Mbyte segments of XBI+ window space. The VAX 9000 system has only 14 BI windows in its I/O space, therefore the last BI window in XMI I/O space is not used.

1.5.4 XMI Address to VAX 9000 System Address

In executing a DMA transaction, a data packet goes from an XMI address to a VAX 9000 system address. As previously mentioned, the XRC functions to check the XMI address for a valid reference to the VAX 9000 main memory. To do this, the XRC checks that the:

- XMI address bit [39] (the MSB) is 0, signifying that the XMI reference is to memory space.
- XMI address bits [38:34] are all 0s. Otherwise, the reference is above VAX 9000 system space.
- XMI address bits [33:29] are not all 1s. If address bits [33:29] are all 1s, the reference is to VAX 9000 I/O space (Figure 1–8).

If the preceding is true, the DMA reference is to VAX 9000 memory space and XMI address bits [33:02] specify the target location in main memory.

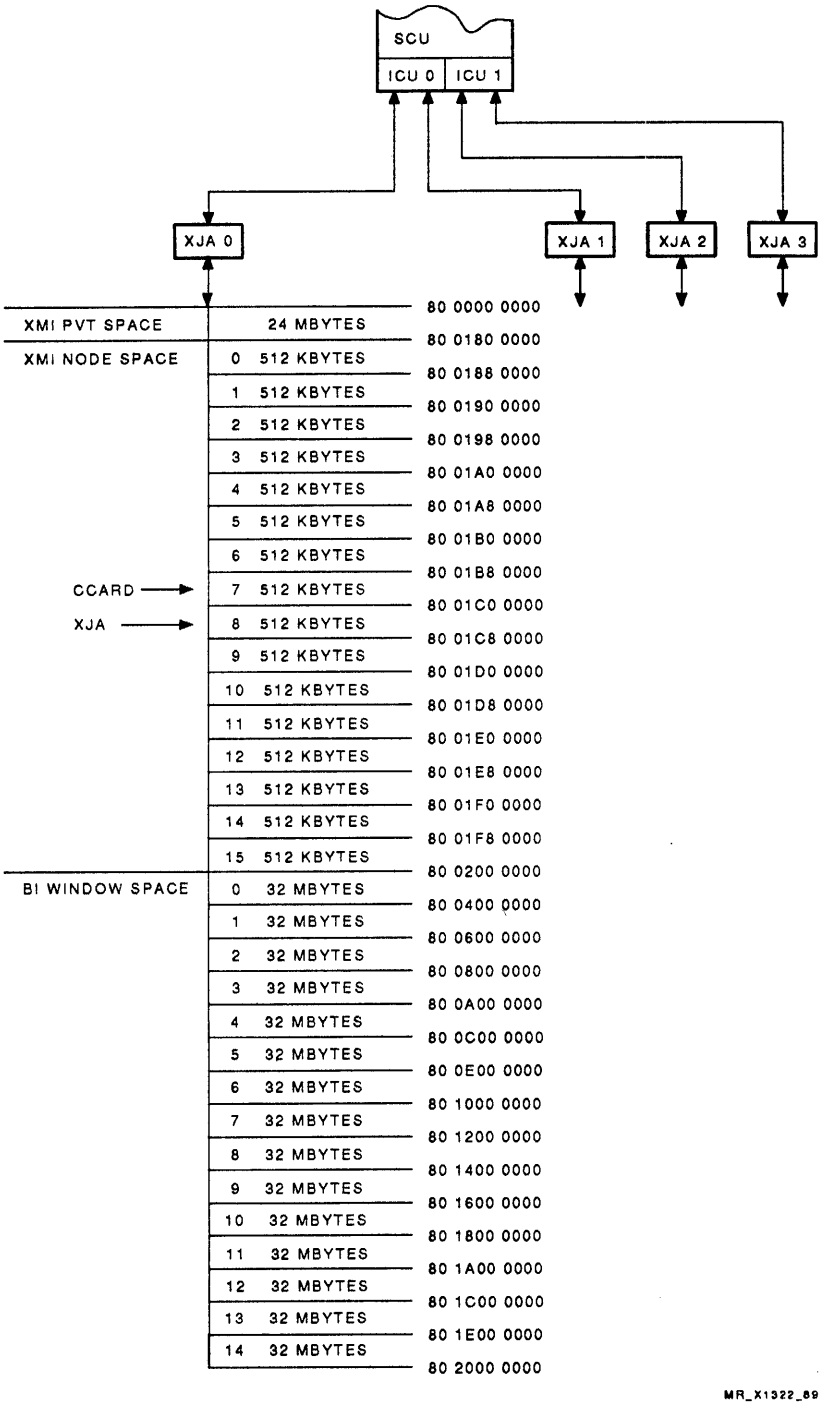


Figure 1–12 XMI I/O Space Allocation

1.5.5 VAX 9000 System Address to XMI Address

Figure 1–13 shows the VAX 9000 I/O space allocation given in Figure 1–9 but with the addition of binary address bits [33:16]. Refer to Figure 1–13 during the following discussion of VAX 9000 system to XMI addressing.

All CPU transactions are to one of the three VAX 9000 I/O space regions previously mentioned:

Node space containing all I/O adapters (including the BI adapter's XMI space registers)
BI window space containing all the BI adapters
XJA private register space

MBYTES			BINARY 33 16	HEX
0			11 1110 0000 0000 0000	3 E000 0000
8	XMI 0 NODE SPACE	8 MBYTES	11 1110 0000 1000 0000	3 E080 0000
16	XMI 1 NODE SPACE	8 MBYTES	11 1110 0001 0000 0000	3 E100 0000
24	XMI 2 NODE SPACE	8 MBYTES	11 1110 0001 1000 0000	3 E180 0000
32	XMI 3 NODE SPACE	8 MBYTES	11 1110 0010 0000 0000	3 E200 0000
64	XBI 0 WINDOW SPACE	32 MBYTES	11 1110 0100 0000 0000	3 E400 0000
96	XBI 1 WINDOW SPACE	32 MBYTES	11 1110 0110 0000 0000	3 E600 0000
128	XBI 2 WINDOW SPACE	32 MBYTES	11 1110 1000 0000 0000	3 E800 0000
160	XBI 3 WINDOW SPACE	32 MBYTES	11 1110 1010 0000 0000	3 EA00 0000
192	XBI 4 WINDOW SPACE	32 MBYTES	11 1110 1100 0000 0000	3 EC00 0000
224	XBI 5 WINDOW SPACE	32 MBYTES	11 1110 1110 0000 0000	3 EE00 0000
256	XBI 6 WINDOW SPACE	32 MBYTES	11 1111 0000 0000 0000	3 F000 0000
288	XBI 7 WINDOW SPACE	32 MBYTES	11 1111 0010 0000 0000	3 F200 0000
320	XBI 8 WINDOW SPACE	32 MBYTES	11 1111 0100 0000 0000	3 F400 0000
352	XBI 9 WINDOW SPACE	32 MBYTES	11 1111 0110 0000 0000	3 F600 0000
384	XBI 10 WINDOW SPACE	32 MBYTES	11 1111 1000 0000 0000	3 F800 0000
416	XBI 11 WINDOW SPACE	32 MBYTES	11 1111 1010 0000 0000	3 FA00 0000
448	XBI 12 WINDOW SPACE	32 MBYTES	11 1111 1100 0000 0000	3 FC00 0000
480	XBI 13 WINDOW SPACE	32 MBYTES	11 1111 1110 0000 0000	3 FE00 0000
480.5	XJA 0 PRIVATE SPACE	512 KBYTES	11 1111 1110 0000 1000	3 FE08 0000
481.0	XJA 1 PRIVATE SPACE	512 KBYTES	11 1111 1110 0001 0000	3 FE10 0000
481.5	XJA 2 PRIVATE SPACE	512 KBYTES	11 1111 1110 0001 1000	3 FE18 0000
482.0	XJA 3 PRIVATE SPACE	512 KBYTES	11 1111 1110 0010 0000	3 FE20 0000
512.0	SCU/SPU REGISTER SPACE	30 MBYTES	11 1111 1111 1111 1111	3 FFFF FFFF

MR_X1323_89

Figure 1–13 System I/O Address Bits

When a CPU transaction is initiated, the SCU determines which XJA channel is being addressed and transmits the CPU request packet to the correct channel. When an XJA receives a CPU request, the XJA knows that the SCU makes this determination and the request packet is for that XJA channel.

The SCU first checks that address bits [33:29] are all 1s to verify that the CPU reference is to I/O space. The SCU then strips off address bits [33:30] as not required and transfers an address field of [29:02] to the XJA.[1]

The SCU then checks address bits [28:25] to determine to what I/O area the CPU is directing its request. If address bits [28:25] are all 0s, the CPU request is for XMI node space. In this case, address bits [24:23] specify the XJA channel being referenced and the SCU directs the request to that channel.

If address bits [28:25] are all 1s, the CPU request is for XJA private register space. In this case, address bits [20:19] specify the XJA channel being referenced and the SCU directs the CPU request to that channel.

When address bits [28:25] are neither all 1s nor all 0s, the CPU request is to BI window space. In this case, address bits [28:25] specify which BI window is being referenced. The VAX 9000 operating system (and the SCU) knows the physical location of all the BI adapters and how they map into VAX 9000 I/O space. Therefore, the SCU knows to which XJA channel to direct the CPU request for a specific BI window.

Figure 1–14 is a flow diagram of address processing of CPU requests to the three regions of VAX 9000 I/O space (XMI node space, XMI BI window space, and XJA private register space). Address processing of CPU requests to XJA private register space was already described in Section 1.5.2.

Address processing of CPU requests to XMI node space or XMI BI window space involve going from a VAX 9000 system address to an XMI address. Because the XMI I/O address space shown in Figure 1–12 applies to all four XMIs, address translation is required to convert the VAX 9000 system address from that shown in Figure 1–9 to the XMI I/O space address of Figure 1–12.

1.5.5.1 VAX 9000 System Address to XMI Node Space Address

When the CPU reference is to XMI node space (CPU address bits [28:25] = 0s), the XJA must translate the CPU address to the base block address for XMI node space. It can be seen in Figure 1–12 that this address is 80 0180 000. The XJA does this by:

- Setting XMI address bit [39] to 1. This is done by placing CPU address bit [29] (which is always a 1) onto XMI address bit [39].
- Setting XMI address bits [38:25] to 0s.
- Setting XMI address bits [24:23] to 1s.

The preceding establishes an XMI address of 80 0180 0000. CPU address bits [22:02] are used directly for XMI address bits [22:02] to reference the specific XMI adapter and the XMI space register within the adapter. The adapter is located in the XMI card cage by matching address bits [22:19] to the 4-bit node ID number. Address bits [18:02] select the register location within the adapter.

[1] Although bit [29] is always a 1, it is transferred to the XJA to be used as XMI address bit [39] to address the XMI's I/O space (Section 1.5.5.1).

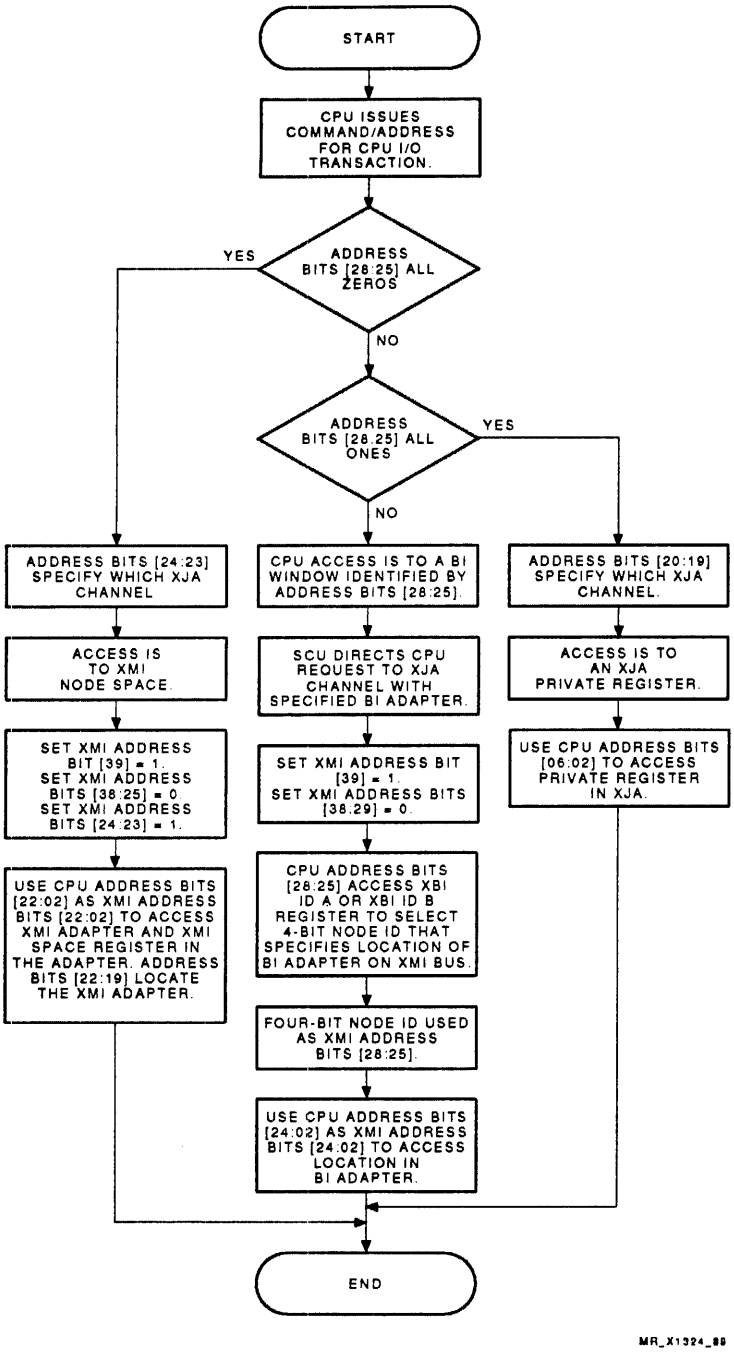


Figure 1–14 Flow Diagram of CPU Request Address Processing

1.5.5.2 VAX 9000 System Address to BI Window Space Address

When the CPU reference is to BI window space (CPU address bits [28:25] are both 1s and 0s), the XJA must translate the CPU address into the XMI address for the referenced BI adapter. The SCU knows which of the 14 window segments of Figure 1–9 is being referenced by means of address bits [28:25]. The VAX 9000 operating system has mapped the location of all the BI adapters and assigned each of them a window in the system I/O space. Therefore, the SCU knows the XMI channel that contains the referenced window and directs the CPU request to that XJA. In addition, the operating system has already loaded the XBI ID A and XBI ID B registers in each XJA with the 4-bit node ID for all the BI adapters in the system.

Referring to the XMI I/O space shown in Figure 1–12, notice that BI window space starts at XMI address 80 0200 0000 and increases by 200 0000 for each BI window. The XJA assembles this address on the XMI by:

- Setting XMI address bit [39] to 1.
- Setting XMI address bits [38:29] to 0s.
- Using address bits [28:25] (which identify the number of the BI adapter within the system) to reference the XBI ID A or XBI ID B register and unload the 4-bit node ID specifying the location of the referenced adapter within the XMI card cage. The 4-bit node ID is placed on the XMI bus as address bits [28:25], which are matched to the 4-bit node ID number of the referenced BI adapter.

CPU address bits [24:02] are then used to reference the desired location within the BI window.

2
JXDI Bus

This chapter provides a detailed description of the JXDI bus. It provides a description of all the JXDI signals and all the transactions that execute over the JXDI. This includes DMA, CPU, and interrupt transactions. Data packet formats for the various types of transactions are also described.

2.1 JXDI Description

The JXDI interfaces 19 signals between the XJA adapter and the ICU portion of the SCU (Figure 2–1).

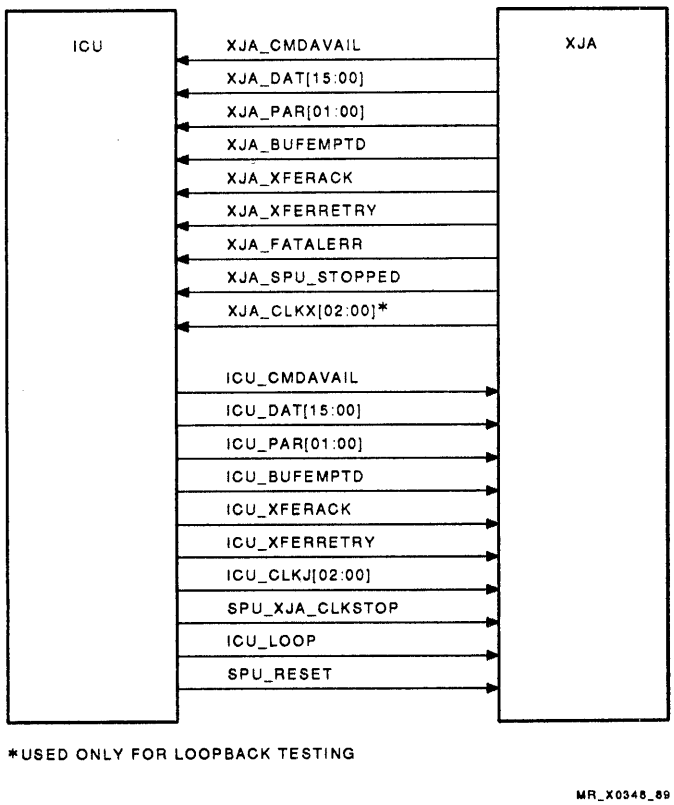


Figure 2–1 JXDI Signals

All JXDI signals are unidirectional. Signals going from the XJA to the ICU are prefixed with XJA. Those signals going from the ICU to the XJA are prefixed with ICU. Two signals not prefixed (SPU_RESET and SPU_XJA_CLKSTOP) are generated by the SPU and transmitted to the XJA through the ICU. Table 2–1 describes the functions of the JXDI signals.

Six of the JXDI signals are symmetrical. That is, six signals going from the XJA to the ICU have identical signals going from the ICU to the XJA. The XJA or ICU prefix identifies the source of the symmetrical signals.

Table 2–1 JXDI Signal Functions

Signal	Function
XJA to ICU	
XJA_CMDAVAIL[1]	The XJA command available signal informs the ICU that a data packet is to follow, starting with the next cycle. The XJA monitors the state of the ICU receive buffers (there are two). XJA_CMDAVAIL cannot assert (and a data packet cannot be sent) unless the ICU has a buffer available to receive the packet.
XJA_DAT[15:00][1]	The XJA data signal transfers command, address, mask, ID, length, interrupt priority level (IPL), and read/write data over the 16 data lines.
XJA_PAR[01:00][1]	The XJA parity signal transfers odd parity over XJA_DAT[15:00]. XJA_PAR[01] is parity for XJA_DAT[15:08] while XJA PAR[00] is parity for XJA_DAT[07:00].
XJA_XFERACK[1]	The XJA transfer acknowledge signal is asserted to the ICU after a data packet is accepted by the XJA. The signal informs the ICU that the associated transmit buffer can now be cleared.
XJA_XFERRETRY[1]	The XJA transfer retry signal is asserted to the ICU after a data packet is rejected by the XJA due to a parity error or the XJA being busy. The signal requests the ICU to retransmit the packet that is still in the ICU transmit buffer.
XJA_BUFEMPTD[1]	The XJA buffer emptied signal informs the ICU that an XJA receive buffer (there are three) has just been emptied, or a data packet sent by the ICU was not accepted due to a parity error or the XJA being busy. If the data packet was not accepted, the buffer emptied signal means that the receive buffer reserved for the unaccepted data packet is still available.
XJA_SPU_STOPPED	This signal is in response to SPU_XJA_CLKSTOP from the SPU (through the ICU). XJA_SPU_STOPPED informs the SPU (through the ICU) that no XJA transmissions are in progress and the XJA will not transmit any data packets onto the JXDI until SPU_XJA_CLKSTOP negates.
XJA_FATALERR	The XJA fatal error signal is asserted when the XJA detects a fatal error. During a fatal error, the XJA cannot be relied upon to act in a predictable manner. It may not be able to respond to CPU requests. XJA_FATALERR results in the CPU initiating a fatal-error service routine.
XJA_CLKX[02:00]	This signal line is used only for loopback testing. (See ICU_LOOP description in this table.)

[1]Symmetrical

Table 2–1 (Cont.) JXDI Signal Functions

Signal	Function
ICU to XJA	
ICU_CMDAVAIL[1]	The ICU command available signal informs the XJA that a data packet is to follow, starting with the next cycle. The ICU monitors the state of the XJA receive buffers (there are three). ICU_CMDAVAIL cannot assert (and a data packet cannot be sent) unless the XJA has a buffer available to receive the packet.
ICU_DAT[15:00][1]	The ICU data signal transfers command, address, mask, ID, length, and read/write data over the 16 data lines.
ICU_PAR[01:00][1]	The ICU parity signal transfers odd parity over ICU_DAT[15:00]. ICU_PAR[01] is parity for ICU_DAT[15:08] while ICU_PAR[00] is parity for ICU_DAT[07:00].
ICU_XFERACK[1]	The ICU transfer acknowledge signal is asserted to the XJA after a data packet is accepted by the ICU. The signal informs the XJA that the associated transmit buffer can now be cleared.
ICU_XFERRETRY[1]	The ICU transfer retry signal is asserted to the XJA after a data packet is rejected by the ICU due to a parity error. The signal requests the XJA to retransmit the packet that is still in the XJA transmit buffer.
ICU_BUFEMPTD[1]	The ICU buffer emptied signal informs the XJA that an ICU receive buffer (there are two) has just been emptied, or a data packet sent by the XJA was not accepted due to a parity error. In the latter case, the buffer emptied signal means that the receive buffer reserved for the unaccepted data packet is still available.
ICU_CLKJ[02:00]	This signal is a 3-line fanout of a 16-ns clock that the ICU sends to the XJA for clocking data into the XJA receive buffers and signals into the XJA receive logic (Figure 2–2). ICU_CLKJ[02:00] is the same clock used for the ICU transmit buffers and transmit logic, therefore obtaining signal synchronization on both ends of the JXDI.
SPU_RESET	This signal is generated by the SPU and sent to the XJA through the ICU. It is used during system initialization.
SPU_XJA_CLKSTOP	This signal is generated by the SPU and sent to the XJA through the ICU. It informs the XJA of impending clock stoppage. Upon receiving this signal, the XJA completes the current JXDI transmission (if any) and does not initiate any new transmissions until the signal is negated. XJA_SPU_STOPPED informs the SPU (through the ICU) that the XJA is quiet and clocks can be stopped.
ICU_LOOP	This is a test command that loops back all symmetrical, ICU-sourced JXDI signals onto the corresponding XJA lines. For example, ICU_CMDAVAIL loops back to the ICU on the XJA_CMDAVAIL line. In addition, SPU_RESET loops back onto XJA_FATALERR, and ICU_CLKJ[02:00] loops back onto a signal line no longer used, called XJA_CLKX[02:00]. SPU_XJA_CLKSTOP does not loop back.

[1]Symmetrical

The XJA and the ICU have transmit buffers from which to transmit data to the JXDI, and receive buffers that receive data from the JXDI (Figure 2–2). The master clock module (MCM) supplies an ICU clock that controls the transmit buffers and transmit logic in the ICU. This clock is transferred to the XJA as ICU_CLKJ[02:00], where it controls the receive buffers and receive logic in the XJA. The ICU data, signals, and clock (ICU_CLK[02:00]) experience the same delay over the JXDI. Therefore, ICU_CLKJ[02:00] arrives at the XJA at the correct time to clock in the ICU data and signals. However, due to the JXDI delay, the data clocked into the XJA is asynchronous with the system clocks and must be synchronized later in the XJA.

Another clock from the MCM (XJA_CLKB) is supplied to the XJA where it controls the transmit buffers and transmit logic. XJA_CLKB is delayed with respect to the ICU clock that clocks the ICU receive buffers and receive logic. The delay compensates for the XJA to ICU skew and time delay resulting in the XJA data and signals being clocked into the ICU synchronously with respect to the system clocks.

The JXDI bus cycle time is equal to the nominal CPU cycle time of 16 ns.

The transfers executing over the JXDI are part of DMA, CPU, or interrupt transactions.

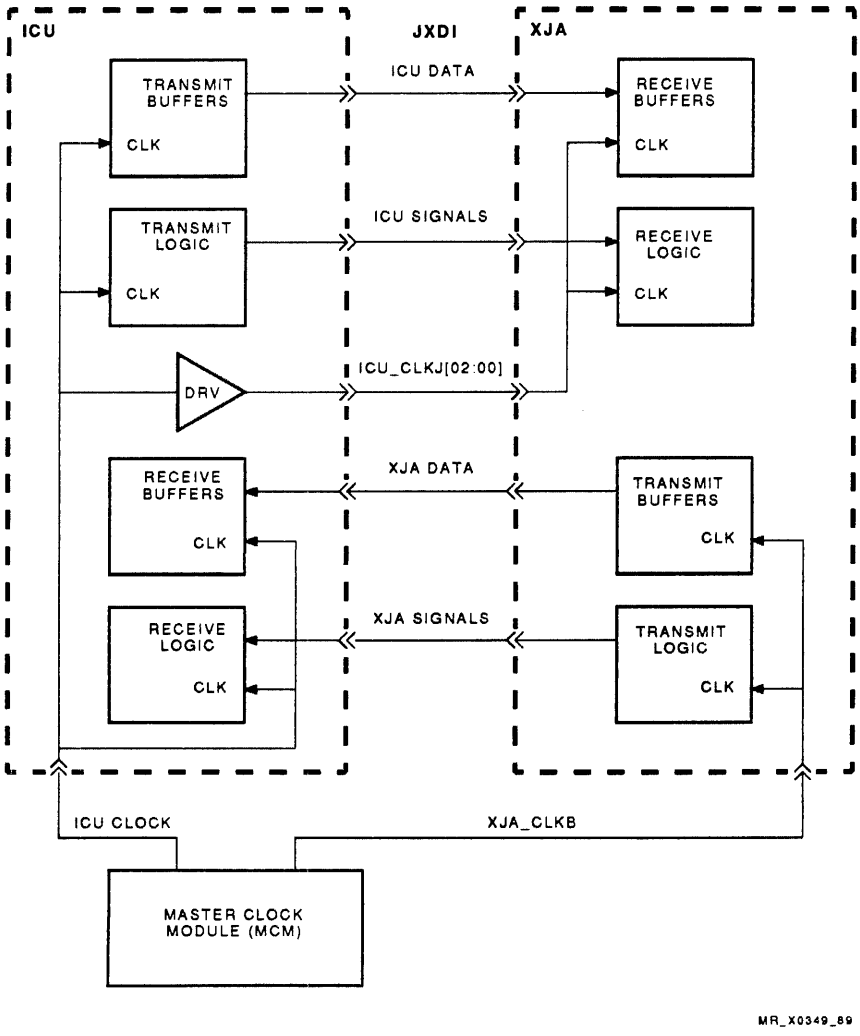


Figure 2–2 ICU and XJA Transmit and Receive Buffers

2.1.1 DMA Transactions

DMA transactions are reads and writes of VAX 9000 main memory from the XMI bus. A DMA transaction may be quadword, octaword, or hexword in length. Up to four DMA transactions may be outstanding at a time.

2.1.2 CPU Transactions

CPU transactions are system CPU reads and writes of registers in the XJA or out on the XMI bus. CPU transactions are longwords only. Only one CPU transaction can be outstanding at a time.

2.1.3 Interrupt Transactions

Interrupt transactions notify the system CPU of errors detected by the XJA, faults on the XMI bus, or interrupts generated by the XMI nodes.

2.1.4 JXDI Transfer Functions

Ten transfer functions (Table 2–2) execute on the JXDI. The table shows the 4-bit command code for each function. The table also indicates if the function executes in both directions or only in one direction, and whether the function is executing in a DMA or a CPU transaction. Note that an interrupt request from the XJA to the ICU can occur during a DMA transfer or a CPU transfer. The functions are described in the following sections where the application of the JXDI signals listed in Table 2–1 are illustrated.

Table 2–2 JXDI Transfer Functions

		Transfer Type	
XJA_DAT[03:00]	**Command**	**XJA to ICU**	**ICU to XJA**
0 0 0 0	READ_REQUEST	DMA	CPU
0 0 0 1	READ_LOCK_REQUEST	DMA	–
0 0 1 0	READ_DATA_RETURN	CPU	DMA
0 0 1 1	READ_LOCK_DATA_RETURN	—	DMA
0 1 0 0	WRITE_REQUEST	DMA	CPU
0 1 0 1	WRITE_UNLOCK_REQUEST	DMA	–
0 1 1 0	Reserved	—	–
0 1 1 1	Reserved	—	–
1 0 0 0	INTERRUPT_REQUEST[1]	DMA, CPU	–
1 0 0 1	READ_LOCKED_STATUS	—	DMA
1 0 1 0	READ_ERROR_STATUS	CPU	DMA
1 0 1 1	WRITE_COMPLETE	CPU	–
1 1 0 0	Reserved	—	–
1 1 0 1	Reserved	—	–
1 1 1 0	Reserved	—	–
1 1 1 1	Reserved	—	–

[1]An interrupt can occur during a DMA or a CPU transaction.

2.2 DMA Read

Figure 2–3 is a flow diagram of the transfer functions that execute on the JXDI during a DMA read or read lock transaction. Refer to the flow diagram throughout the following discussion.

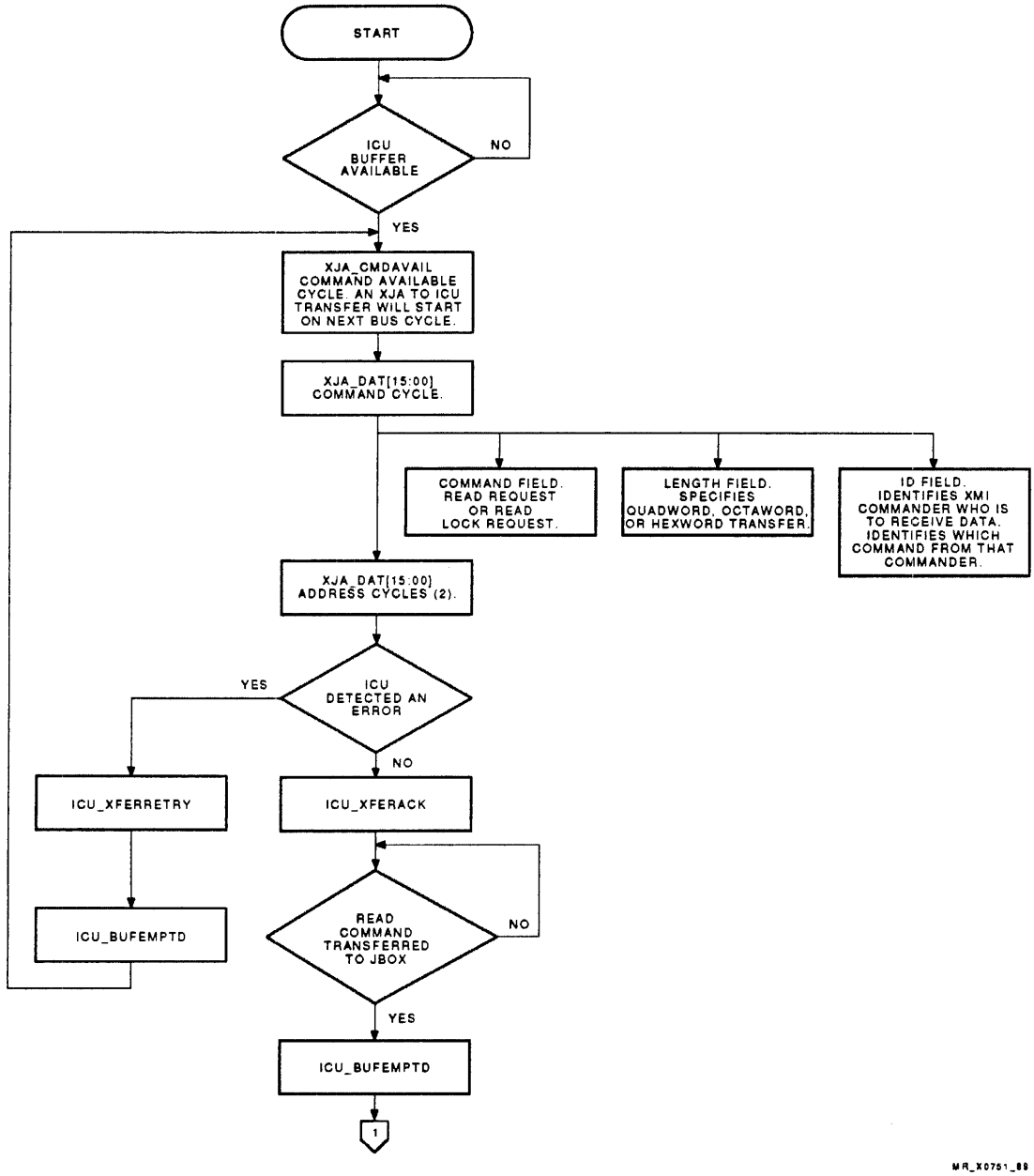


Figure 2–3 (Cont.) Flow Diagram of JXDI DMA Read or Read Lock

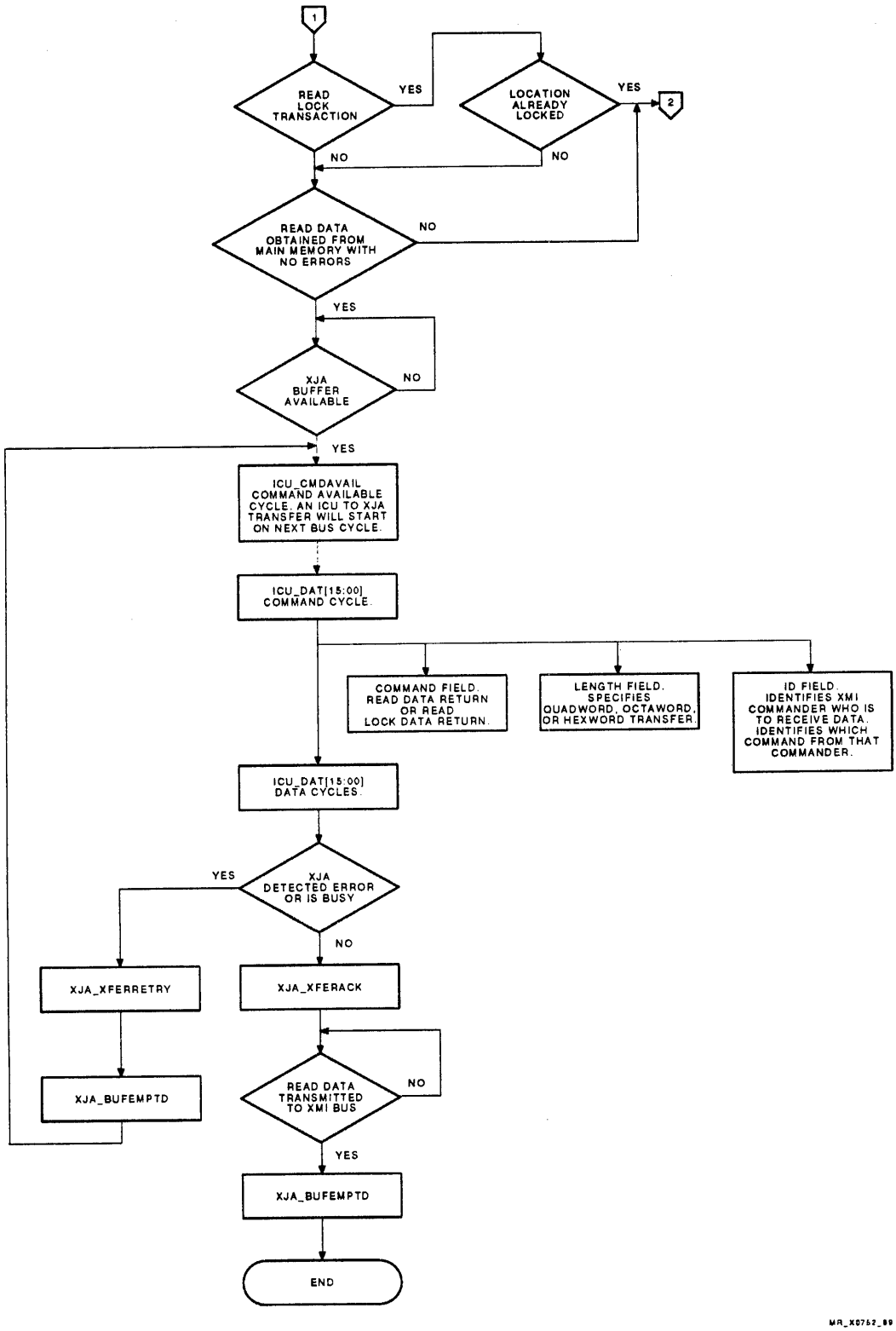


Figure 2–3 (Cont.) Flow Diagram of JXDI DMA Read or Read Lock

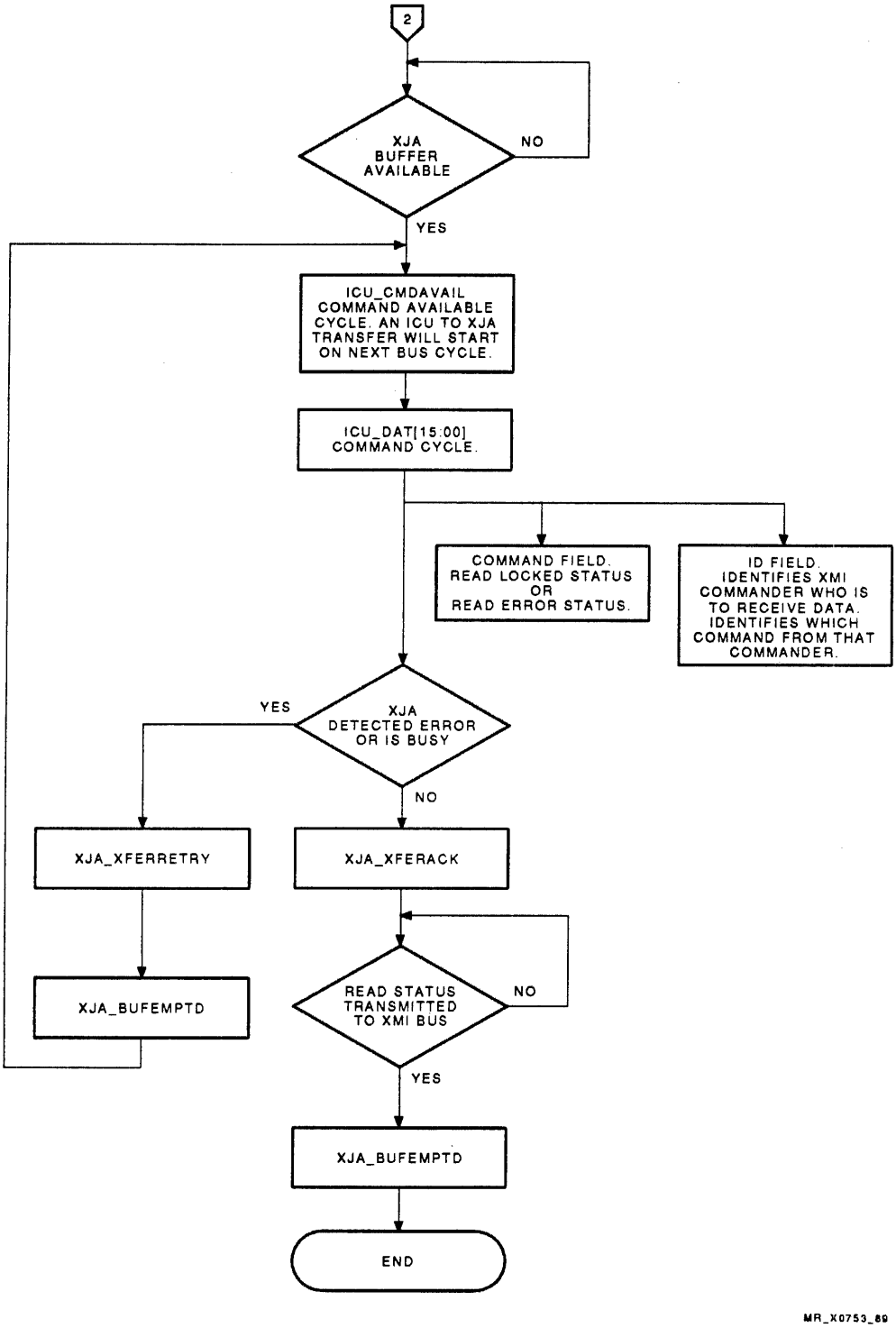


Figure 2–3 Flow Diagram of JXDI DMA Read or Read Lock

2.2.1 Command Available Cycle

The XJA asserts XJA_CMDAVAIL (command available) to inform the ICU that a transfer follows starting with the next cycle. The XJA is allowed to assert XJA_CMDAVAIL only if an ICU buffer is available to receive the transmitted information. The availability of an ICU buffer is indicated by a buffer empty counter in the XJA that monitors the state of the ICU receive buffers (Section 2.2.7).

2.2.2 Read Request Command Cycle

The XJA data lines (XJA_DAT[15:00]) carry information to the ICU in three bus cycles: a command cycle followed by two address cycles (Figure 2–4). The command cycle contains:

A command field
A length field
An ID field

CYCLE	TYPE	15 14 13 12 11 10 09 08								07 06 05 04 03 02 01 00							
0	COMMAND	R	R	ID						M	R	LEN				CMD	
1	ADDRESS 0	A [29:26, 05:02]								A [13:06]							
2	ADDRESS 1	A [33:30, 25:22]								A [21:14]							

R = RESERVED

NOTE: BIT 7 IS THE M (MORE) BIT IN A READ COMMAND CYCLE.
IT MUST BE ZERO IN A READ LOCK COMMAND CYCLE.
THE M BIT IS CURRENTLY NOT USED BY THIS SYSTEM.

MR_X0350_89

Figure 2–4 DMA Read Request or Read Lock Request Bus Cycles

2.2.2.1 Command Field

The command field occupies data lines XJA_DAT[03:00]. Table 2–2 defines the command field code. The command code for this transfer is 0000 (read request) or 0001 (read lock request).

2.2.2.2 Length Field

The length field occupies data lines XJA_DAT[05:04]. Table 2–3 specifies the length codes with ICU_DAT[05:04] replacing XJA_DAT[05:04].

Table 2–3 Length Field Codes

XJA_DAT[05:04]	Length
0 0	Hexword (32 bytes)
0 1	Reserved
1 0	Quadword (8 bytes)
1 1	Octaword (16 bytes)

2.2.2.3 ID Field

The ID field occupies data lines XJA_DAT[13:08]. The field contains a unique code used to identify the node ID of the XMI commander, and with which of the commander's transactions the command cycle is associated. (An XMI commander can have four transactions outstanding at one time.) The node ID of the XMI commander is the destination for the returned read data.

The ID field is encoded as shown in Table 2–4.

Table 2–4 ID Code for DMA Transaction Commanders

XJA_DAT[13:08]	Function
[13:10]	Identifies XMI node of transaction commander
[09:08]	Identifies transaction

2.2.3 Address Cycles

The second and third bus cycles are address cycles. Each address cycle carries 16 address bits as shown in Figure 2–4. Address bits [01:00] are not used as bytes and words are not supported as valid DMA transaction lengths. Therefore, the 32 address bits transferred are [33:02].

2.2.4 Memory Address Wraps

All reads to main memory are aligned on quadword boundaries. Octaword and hexword reads are wrapped, if not addressed to an octaword or hexword boundary, respectively. For example, in part A of Figure 2–5, an octaword read to address 18(hex) reads quadword 3 at address 18(hex) and then wraps back to read quadword 2 at address 10(hex). Therefore, the octaword read is entirely in an octaword boundary. The read does not cross over a boundary into the next octaword.

Part B of Figure 2–5 illustrates a hexword read to address 18(hex). With regard to wrapping, a hexword read is treated as two octaword reads with the second octaword read in the same sequence as the first. The first quadword read is quadword 3 at address 18(hex). Considering this as the first octaword read, the wrap goes back to address 10(hex) to read quadword 2 (the second half of the first octaword read). The second octaword, in the hexword boundary, consists of quadwords 0 and 1. This octaword is read in the same sequence as the first octaword resulting in quadword 1 being the third quadword read and quadword 0 being the last.

Part C of Figure 2–5 illustrates a hexword read to address 14(hex). Note that the read is made to a nonquadword boundary. In this case, the reference backs up to the next quadword boundary, which is 10(hex). The first octaword is read in order with quadword 2 read first followed by quadword 3 at address 18(hex). The read wraps back to the second octaword within the hexword boundary, which is read in order as was the first. Therefore, quadword 0 becomes the third quadword read and quadword 1 becomes the fourth.

Figure 2–5 Octaword and Hexword Wraparound Reads

DIGITAL INTERNAL USE ONLY

2.2.9.3 ID Field

The ID field occupies data lines ICU_DAT[13:08]. The field contains the code used to identify the node of the XMI commander and the specific command being executed. (An XMI commander can have four transactions outstanding at a time.) The node ID of the XMI commander is the destination for the returned read data.

The ID field code is shown in Table 2–4 with ICU_DAT[13:08] replacing XJA_DAT[13:08].

2.2.10 Data Cycles

Data bus cycles follow the command cycle with each cycle carrying one word of data. There will be 4, 8, or 16 data cycles, depending on the length of the read transaction.

2.2.11 Data Parity

ICU_PAR[01:00] provides odd parity for each bus cycle (command and data). Parity bit [01] is parity over data bits [15:08]. Parity bit [00] is parity over data bits [07:00]. The parity bit is asserted when the number of bits asserted in the data field is even.

2.2.12 Acknowledge Cycle and Retry Cycle

The XJA checks parity on each cycle of received data. If no parity error is found in the data packet, and the XJA is not busy, the XJA asserts XJA_XFERACK (transfer acknowledge) to the ICU indicating that the entire data packet was received by the XJA without parity errors. XJA_XFERACK allows the ICU to clear the data from its transmit buffer.

If a parity error is found or the XJA is busy, the XJA asserts XJA_XFERRETRY (transfer retry) to the ICU, and the ICU must retry the failed transfer. XJA_XFERRETRY asserts as soon as the parity error is detected. It does not wait for the end of the data packet. If the retry also experienced a parity error, more retries are executed. If subsequent retries also fail, the SCU experiences a timeout for failing to move the data packet across the JXDI. The timeout results in a system interrupt and execution of the appropriate service routine.

2.2.13 XJA Buffer Emptied Cycle

When the XJA empties its receive buffer, it returns XJA_BUFEMPTD to the ICU. XJA_BUFEMPTD indicates that the XJA has emptied the receive buffer and can accept new data from the ICU. The XJA has three receive buffers. The ICU monitors the status of these buffers by means of a buffer empty counter. The counter is preset to a count of three and is decremented by one for each ICU transmission. If the counter reaches zero, all the XJA receive buffers are full, the ICU is inhibited from asserting ICU_CMDAVAIL, and no ICU transmission can occur (Section 2.2.8).

If the XJA did not accept the data packet, it asserts XJA_BUFEMPTD to inform the ICU that the receive buffer reserved for the unaccepted data packet is still available.

2.2.14 Read Locked Status or Read Error Status

If this is a read lock transaction and the read location in main memory was already locked, or the read data obtained from main memory had an uncorrectable error, an error status transfer is initiated.

If an XJA receive buffer is available, the ICU asserts ICU_CMDAVAIL informing the XJA that a transfer follows. The ICU then transmits a command cycle (Figure 2–7) with the command field specifying a read locked status (1001) or a read error status (1010) as the case may be (Table 2–2).

The ID field identifies the XMI commander and the command number (Table 2–4).

As is the case for all JXDI data transfers, parity bits (ICU PAR[01:00]) are generated for the data bits and parity is checked by the XJA. If no parity error is found, and the XJA is not busy, the XJA returns XJA_XFERACK to the ICU to clear its transmit buffers.

If the XJA encountered a parity error or is busy, it returns XJA_XFERRETRY to the ICU, which retries the transfer. If more retries are required and also fail, the SCU experiences a timeout for failing to move the status word across the JXDI. The timeout results in a system interrupt and execution of the appropriate service routine.

When the XJA empties its receive buffer (or asserts XJA_XFERRETRY), it asserts XJA_BUFEMPTD to the ICU to inform the ICU of the availability of an XJA receive buffer (Section 2.2.13).

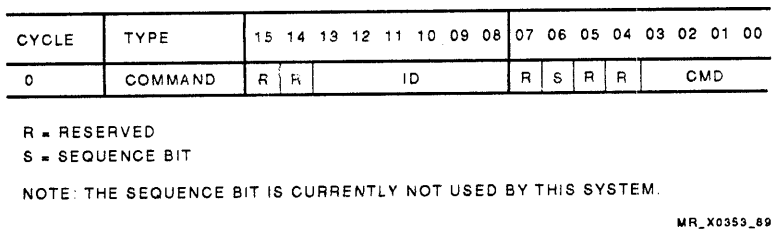


Figure 2–7 DMA Read Locked Status or Read Error Status Command Cycle

2.3 DMA Write

NOTE
DMA hexword writes are not currently supported by the SCU. The XJA does support hexword writes, so they are included in the following description.

Figure 2–8 is a flow diagram of the transfer functions that execute on the JXDI during a DMA write or write unlock transaction. Refer to the flow diagram throughout the following discussion.

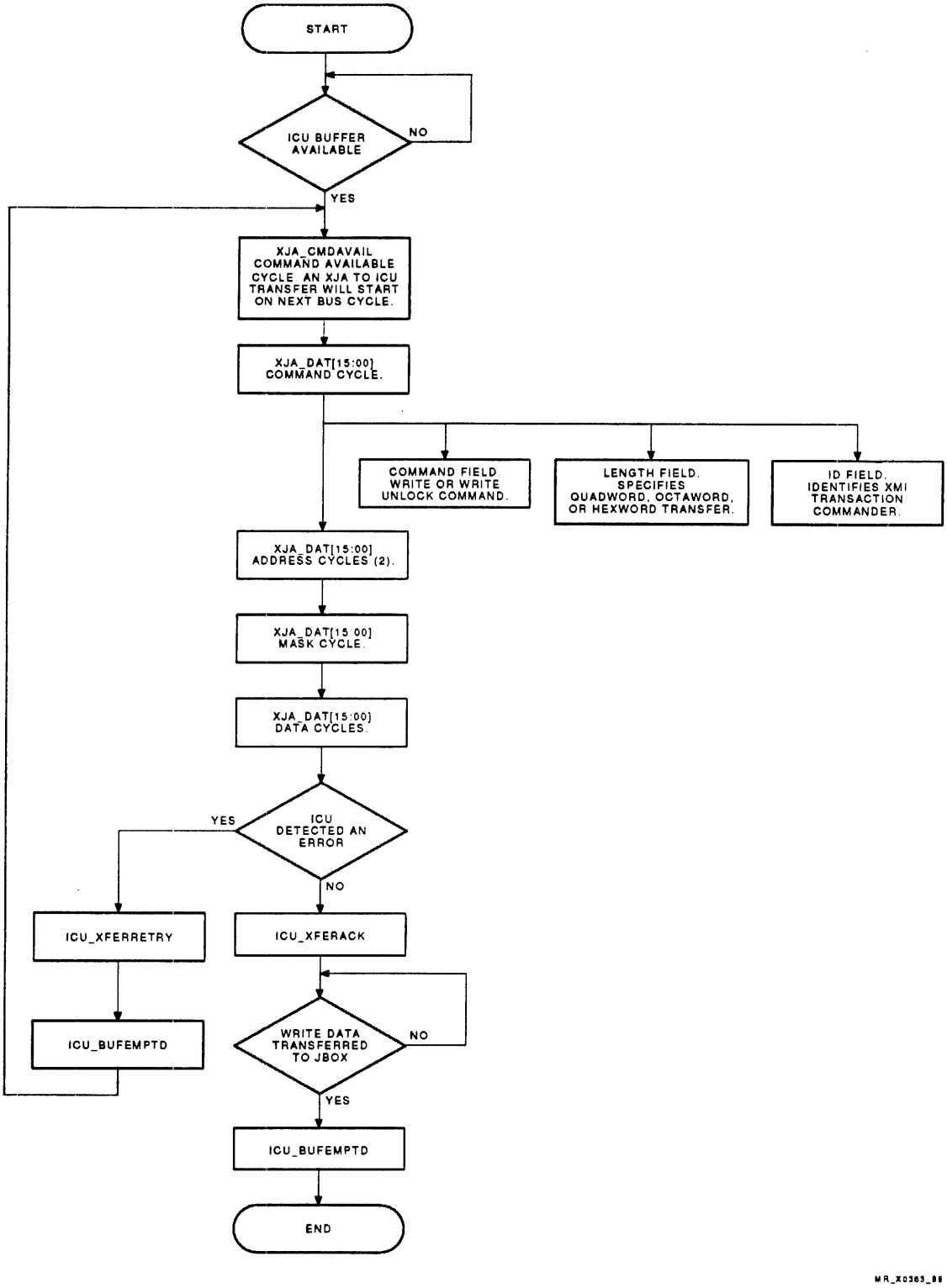


Figure 2–8 Flow Diagram of JXDI DMA Write or Write Unlock

2.3.1 Command Available Cycle

The XJA asserts XJA_CMDAVAIL (command available) to inform the ICU that a transfer follows starting with the next cycle. The XJA is only allowed to assert XJA_CMDAVAIL if an ICU buffer is available to receive the transmitted information. The availability of an ICU buffer is indicated by the XJA buffer empty counter (Section 2.3.8).

2.3.2 Write Command Cycle

The XJA data lines (XJA_DAT[15:00]) carry command, address, mask, and data information to the ICU. The format of the information is illustrated in Figure 2–9. The command cycle contains:

A command field
A length field
An ID field

CYCLE	TYPE	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
0	COMMAND	R R ID	M S LEN CMD
1	ADDRESS 0	A [29:26, 05:02]	A [13:06]
2	ADDRESS 1	A [33:30, 25:22]	A [21:14]
3	MASK BITS	F E D C 7 6 5 4	B A 9 8 3 2 1 0
4	DATA	BYTE 4	BYTE 0
5	DATA	BYTE 5	BYTE 1
6	DATA	BYTE 6	BYTE 2
7	DATA	BYTE 7	BYTE 3
8	DATA	BYTE C	BYTE 8
9	DATA	BYTE D	BYTE 9
10	DATA	BYTE E	BYTE A
11	DATA	BYTE F	BYTE B
12	DATA	BYTE 14	BYTE 10
13	DATA	BYTE 15	BYTE 11
14	DATA	BYTE 16	BYTE 12
15	DATA	BYTE 17	BYTE 13
16	DATA	BYTE 1C	BYTE 18
17	DATA	BYTE 1D	BYTE 19
18	DATA	BYTE 1E	BYTE 1A
19	DATA	BYTE 1F	BYTE 1B

R = RESERVED
S = SEQUENCE BIT

NOTE: THE SEQUENCE BIT IS CURRENTLY NOT USED BY THIS SYSTEM.

BIT 7 IS THE M (MORE) BIT IN A WRITE COMMAND CYCLE.
IT MUST BE ZERO IN A WRITE UNLOCK COMMAND CYCLE.
THE M BIT IS CURRENTLY NOT USED BY THIS SYSTEM.

MR_X0354_89

Figure 2–9 DMA Write or Write Unlock Data Cycles

2.3.2.1 Command Field

The command field occupies data lines XJA_DAT[03:00]. Table 2–2 defines the command field code. The command code for this transaction is 0100 (write request) or 0101 (write unlock request).

2.3.2.2 Length Field

The length field occupies data lines XJA_DAT[05:04]. Table 2–3 specifies the length codes.

2.3.2.3 ID Field

The ID field occupies data lines XJA_DAT[13:08]. The field contains a unique code used to identify the XMI commander as the source of the request. The ID field code is shown in Table 2–4.

2.3.3 Address Cycles

The second and third bus cycles are address cycles. Each address cycle carries 16 address bits as shown in Figure 2–9. Bits [01:00] are not used as bytes and words are not supported as valid DMA transaction lengths. Therefore, the 32 address bits transferred are [33:02].

All writes to main memory are aligned on quadword boundaries. Quadword, octaword, and hexword writes are done on quadword, octaword, and hexword boundaries, respectively (Figure 2–5). Therefore, address bits [02:00] can be ignored on quadword writes, bits [03:00] can be ignored on octaword writes, and bits [04:00] can be ignored on hexword writes.

2.3.4 Mask Cycle

The fourth bus cycle contains the byte mask field for the write data that is to follow. Bit 0 corresponds to byte 0, bit 1 corresponds to byte 1, and so on. When the write transaction is a quadword, the unused mask bits (bits [F:C] and [B:08]) are deasserted. When the write transaction is an octaword, all 16 mask bits are used to specify the mask status of their associated bytes. When the write transaction is a hexword, the mask bits are all 1s since in a hexword transaction there is no masking; all 32 bytes are valid.

2.3.5 Data Cycles

Data cycles are next, with each cycle carrying one word of data. There will be 4, 8, or 16 data cycles, depending on the length of the write transaction.

2.3.6 Data Parity

XJA_PAR[01:00] provides odd parity for each bus cycle (command, address, mask, data). Parity bit [01] is parity over data bits [15:08]. Parity bit [00] is parity over data bits [07:00]. The parity bit is asserted when the number of bits asserted in the data field is even.

2.3.7 Acknowledge Cycle and Retry Cycle

The ICU checks parity on each cycle of the received data. If the data packet was received without any parity errors, the ICU asserts ICU_XFERACK (transfer acknowledge) to the XJA indicating that all the write data was received by the ICU without parity errors. ICU_XFERACK allows the XJA to clear the write data from its transmit buffer.

If a parity error was found, the ICU asserts ICU_XFERRETRY (transfer retry), indicating that a parity error was detected by the ICU and the XJA must retry the failed transfer. If more retries are required and also fail, the SCU notes the continuous parity errors, interrupts the CPU, and executes a service routine.

2.3.8 ICU Buffer Emptied Cycle

When the ICU transfers the write data from its receive buffer to the JBox, it returns ICU_BUFEMPTD to the XJA. ICU_BUFEMPTD indicates that the ICU has emptied the receive buffer and can accept new data from the XJA. The ICU has two receive buffers. The XJA monitors the status of these buffers by means of a buffer empty counter. The counter is preset to a count of two and is decremented by one for each XJA transmission. If the counter reaches zero, all the ICU receive buffers are full, the XJA is inhibited from asserting XJA_CMDAVAIL, and no XJA transmission can occur (Section 2.3.1).

If the ICU commanded a retry, it also asserts ICU_BUFEMPTD to increment the buffer empty counter in the XJA.

2.4 CPU Read

Only one CPU transaction can be outstanding at a time. All CPU transactions are longwords only. Figure 2–10 is a flow diagram of the transfer functions that execute on the JXDI during a CPU read transaction. Refer to the flow diagram throughout the following discussion.

2.4.1 Command Available Cycle

The ICU asserts ICU_CMDAVAIL (command available) to inform the XJA that a transfer follows starting with the next cycle. The ICU is only allowed to assert ICU_CMDAVAIL if an XJA buffer is available to receive the transmitted information (Section 2.2.8).

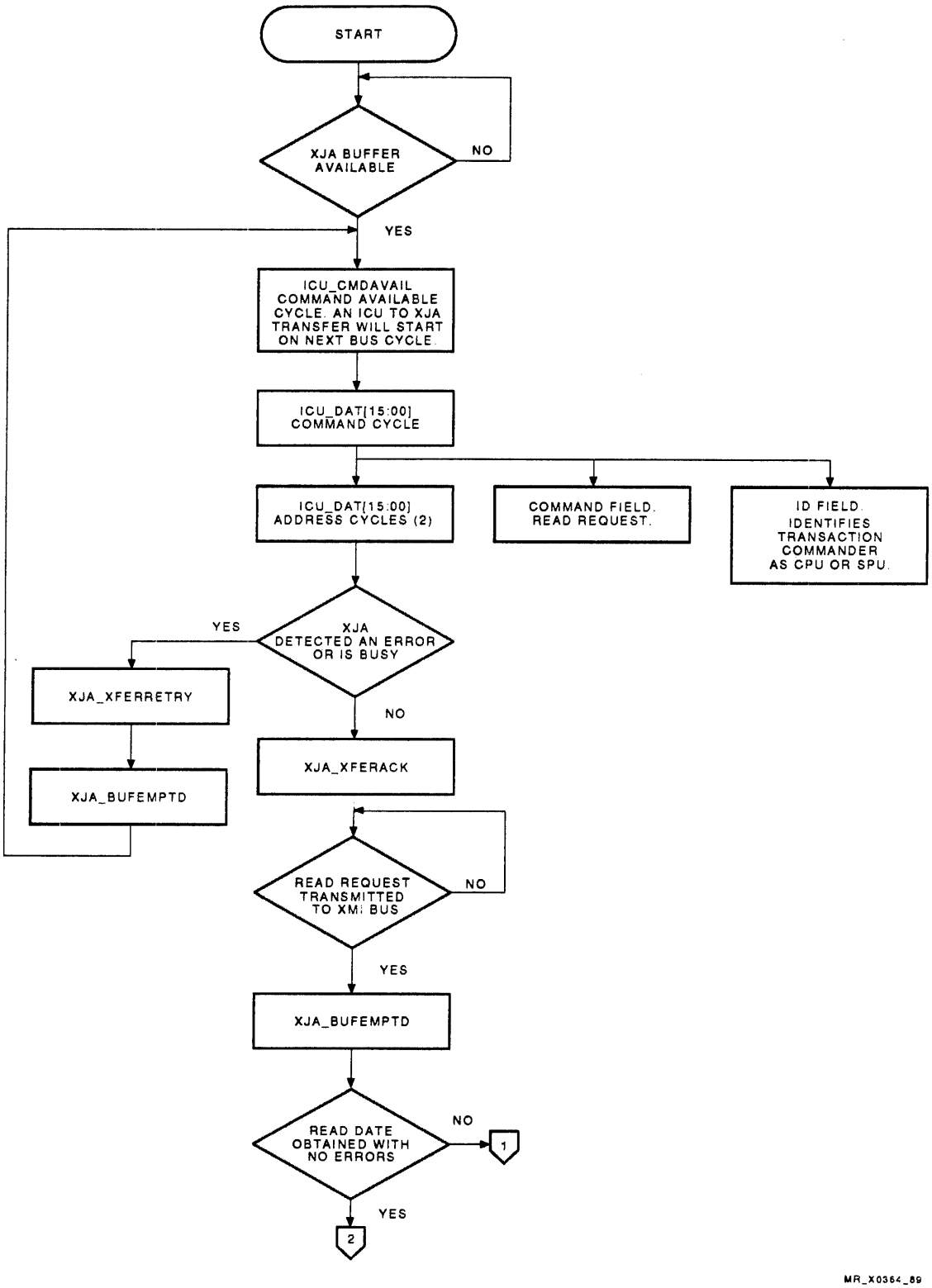


Figure 2–10 (Cont.) Flow Diagram of JXDI CPU Read

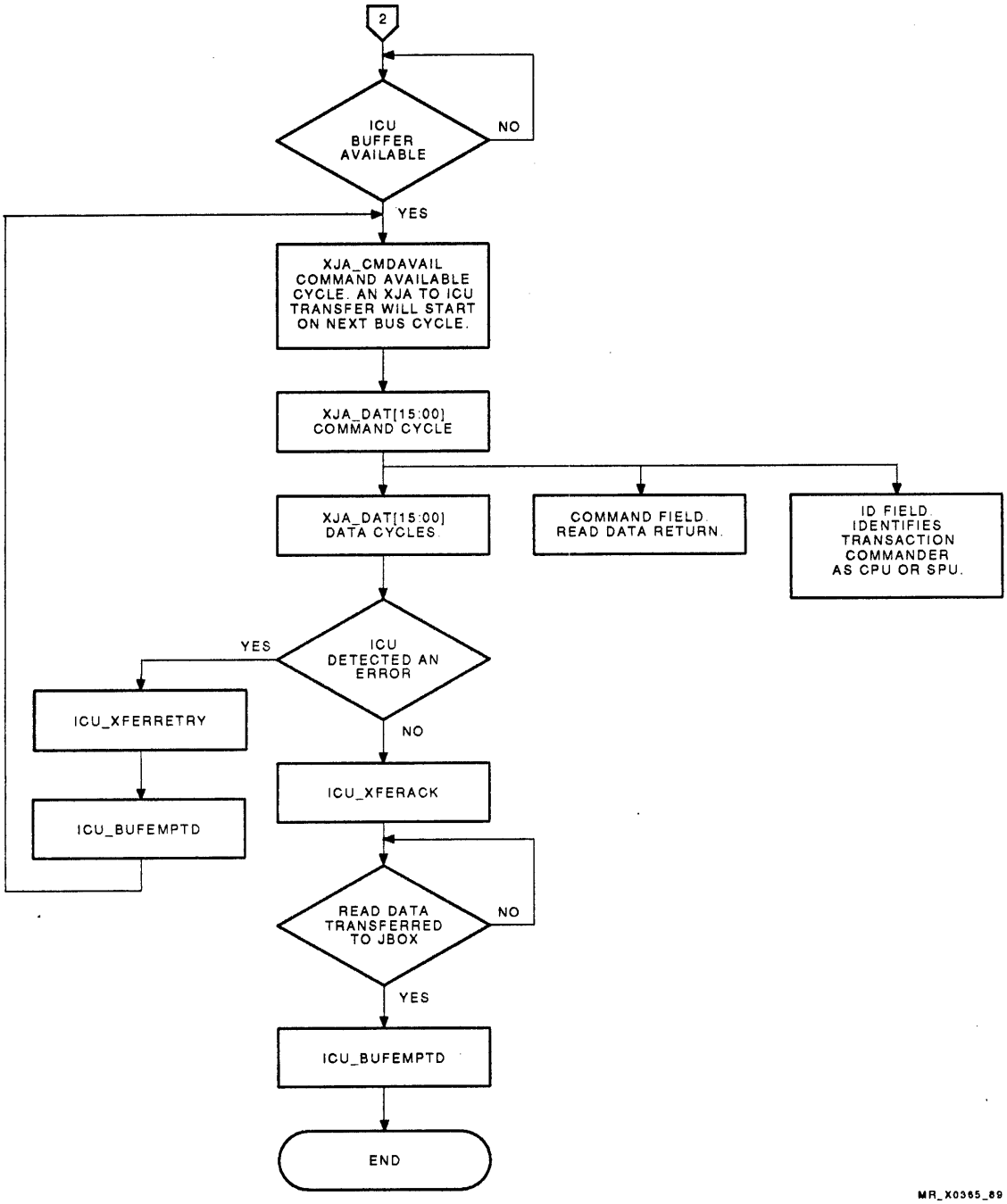


Figure 2–10 (Cont.) Flow Diagram of JXDI CPU Read

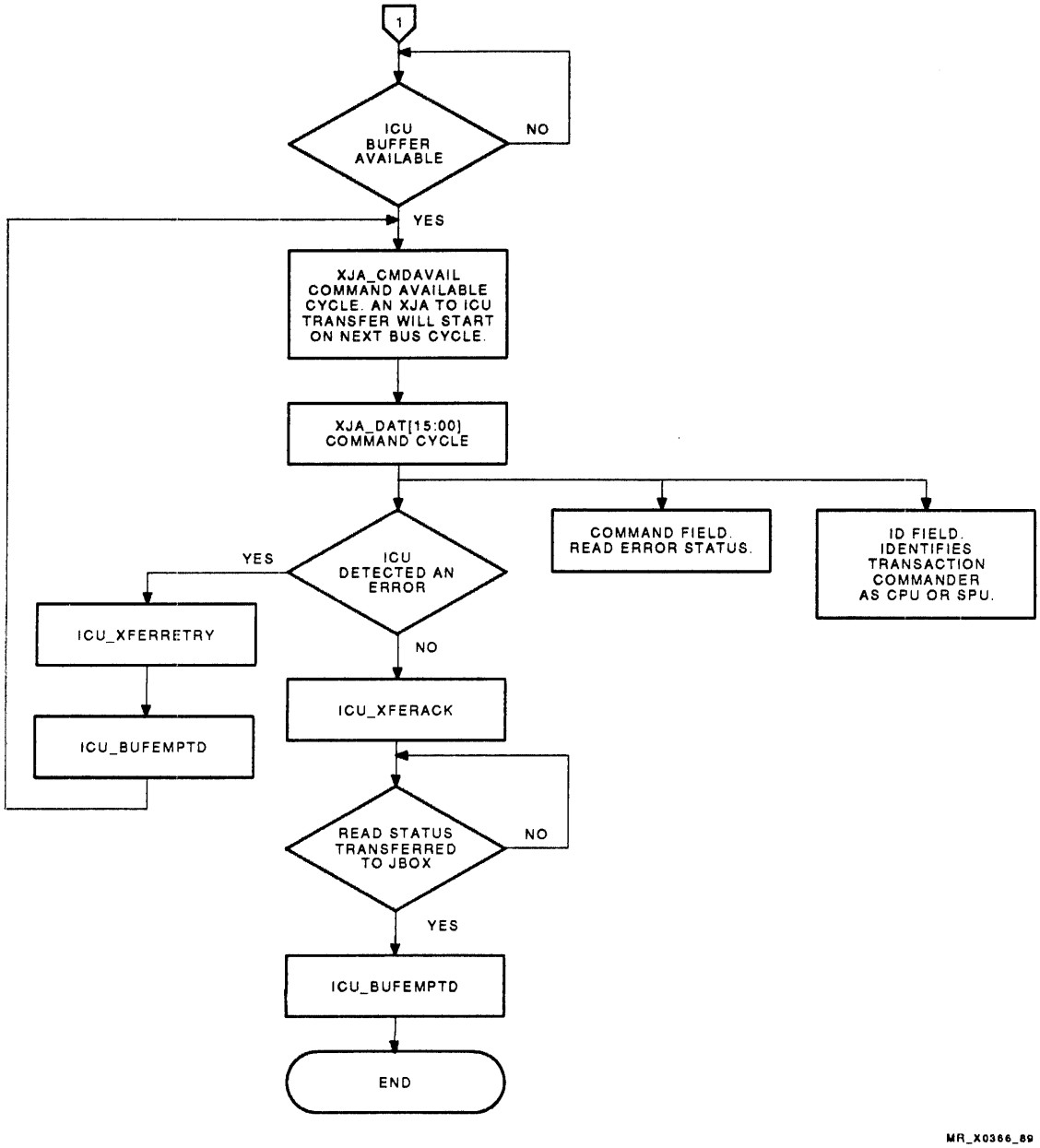


Figure 2–10 Flow Diagram of JXDI CPU Read

2.4.2 Read Request Command Cycle

The ICU data lines (ICU_DAT[15:00]) carry information to the XJA in three bus cycles: a command cycle followed by two address cycles (Figure 2–11). The command cycle contains:

A command field
An ID field

CYCLE	TYPE	15 14 13 12 11 10 09 08								07 06 05 04 03 02 01 00							
0	COMMAND	R	R	ID						R	R	R	R	CMD			
1	ADDRESS 0	A [29:26, 05:02]								A [13:06]							
2	ADDRESS 1	MASK*				A [25:22]				A [21:14]							

R = RESERVED
*SPECIFIES ADDRESS BITS [01:00]

MR_X0355_89

Figure 2–11 CPU Read Request Bus Cycles

2.4.2.1 Command Field

The command field occupies data lines ICU_DAT[03:00]. Table 2–2 defines the command field code. The command code for this transfer is 0000 (read request).

2.4.2.2 ID Field

The ID field occupies data lines ICU_DAT[13:08]. The ID code is used to identify the source of the request as either the CPU or the service processor unit (SPU) (Table 2–5).

Table 2–5 ID Code for CPU Transaction Commanders

ICU_DAT[13:8]	Transaction Commander
0 0 0 0 0 0	CPU
0 0 0 0 0 1	SPU

NOTE
The ID field is currently not used as the system control unit (SCU) knows the commander for all CPU type transactions. The six ID bits are always 0.

2.4.3 Address Cycles

The second and third data cycles are address cycles. The first address cycle carries 16 address bits as shown in Figure 2–11. The second address cycle carries 12 address bits in locations ICU_DAT[11:00]. Bits ICU_DAT[15:12] of the second address cycle comprise the mask field that specifies which bytes are to be read as shown in Table 2–6. The XJA uses the mask field to determine address bits [01:00] as shown in the table. These address bits are required when the CPU read is dealing with a VAXBI word-mode or byte-mode transaction.

Table 2–6 Address Bits [01:00] Versus Mask Field

ICU_DAT[15:12][1]	Read Byte/Word	Address Bits [01:00]
0 0 0 0	None	0 0
0 0 0 1	Byte 1	0 0
0 0 1 0	Byte 2	0 1
0 0 1 1	Bytes 1 and 2 (word 1)	0 0
0 1 0 0	Byte 3	1 0
1 0 0 0	Byte 4	1 1
1 1 0 0	Bytes 3 and 4 (word 2)	1 0
1 1 1 1	All bytes (longword)	0 0

[1]An asserted bit reads the associated byte.

2.4.4 Data Parity

ICU_PAR[01:00] provides odd parity for each bus cycle (command and address). Parity bit [01] is parity over data bits [15:08]. Parity bit [00] is parity over data bits [07:00]. The parity bit is asserted when the number of bits asserted in the data field is even.

2.4.5 Acknowledge Cycle and Retry Cycle

The XJA checks parity on each cycle of the received data. If the data packet was received without any parity errors, and the XJA was not busy, the XJA asserts XJA_XFERACK (transfer acknowledge) to the ICU indicating that the read request was received by the XJA without parity errors.

If a parity error was found or the XJA was busy, the XJA asserts XJA_XFERRETRY to the ICU. In either case, the ICU must retry the failed transfer. If more retries are required and also fail, the SCU experiences a timeout for failing to move the data packet across the JXDI. The timeout results in a system interrupt and execution of the appropriate service routine.

2.4.6 XJA Buffer Emptied Cycle

When the XJA transfers the read request data from its receive buffer, it returns XJA_BUFEMPTD to the ICU. XJA_BUFEMPTD indicates that the XJA has emptied the receive buffer and can accept new data from the ICU (Section 2.2.13). If the XJA commanded a retry, it also asserts XJA_BUFEMPTD.

2.4.7 Command Available Cycle

If the read data was obtained by the XJA with no uncorrectable errors and an ICU receive buffer is available, the transaction continues with the XJA asserting XJA_CMDAVAIL. XJA_CMDAVAIL informs the ICU that the requested read data transfers to the ICU starting with the next cycle. The XJA is only allowed to assert XJA_CMDAVAIL if an ICU buffer is available to receive the transmitted information.

2.4.8 Return Data Command Cycle

The XJA data lines (XJA_DAT[15:00]) carry information to the ICU in one command cycle, followed by three reserved cycles, then four data cycles with each data cycle carrying one byte of data (Figure 2–12). The use of three reserve cycles and four data cycles to transmit the read longword is dictated by the ICU receive logic, which accepts only one data format. Note in Figure 2–9 that a DMA write data packet starts sending data to the ICU in the fourth bus cycle with the first longword (bytes 0 through 3) spread over the next four cycles.

The command cycle contains a command field and an ID field. The command field occupies data lines XJA_DAT[03:00]. Table 2–2 defines the command field code, which for this transfer is 0010 (read data return).

The ID field occupies data lines XJA_DAT[13:08]. The field is used to identify the transaction commander as the CPU or the SPU. The ID field is currently all 0s (Table 2–5).

CYCLE	TYPE	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
0	COMMAND	R R ID	R R R R CMD
1	DATA	RESERVED	RESERVED
2	DATA	RESERVED	RESERVED
3	DATA	RESERVED	RESERVED
4	DATA	RESERVED	BYTE 0
5	DATA	RESERVED	BYTE 1
6	DATA	RESERVED	BYTE 2
7	DATA	RESERVED	BYTE 3

R = RESERVED

MR_X0356_89

Figure 2–12 CPU Read Data Return Bus Cycles

2.4.9 Data Cycles

Following the command cycle are three reserved cycles, followed by four data cycles each carrying one byte of data. The data byte is in the lower order bits (bits [07:00]) with the higher order bits (bits [15:08]) being reserved.

2.4.10 Data Parity

XJA_PAR[01:00] provides odd parity for each bus cycle (command and data). Parity bit [01] is parity over data bits [15:08]. Parity bit [00] is parity over data bits [07:00]. The parity bit is asserted when the number of bits asserted in the data field is even.

2.4.11 Acknowledge Cycle and Retry Cycle

The ICU checks parity on each cycle of received data. If no parity error is found in the data packet, the ICU asserts ICU_XFERACK (transfer acknowledge) indicating that the entire data packet was received by the ICU without parity errors. ICU_XFERACK allows the XJA to clear the data from its transmit buffer.

If a parity error is found, the ICU asserts ICU_XFERRETRY indicating that a parity error was detected by the ICU. In this case, the XJA must retry the failed transfer. If more retries are required and also fail, the SCU notes the continuous parity errors, interrupts the CPU, and executes a service routine.

2.4.12 ICU Buffer Emptied Cycle

When the ICU transfers the read data from its receive buffer to the JBox, it returns ICU_BUFEMPTD to the XJA. ICU_BUFEMPTD indicates that the ICU has emptied the receive buffer and can accept new data from the XJA (Section 2.2.7). If the ICU commanded a retry, it also asserts ICU_BUFEMPTD.

2.4.13 Read Error Status

If the read data obtained by the XJA had an uncorrectable error, an error status transfer is initiated.

The XJA asserts XJA_CMDAVAIL informing the ICU that a transfer follows. The XJA then transmits a CPU status command cycle (Figure 2–13) with the command field specifying a read error status (1010) (Table 2–2). The ID field identifies the transaction commander as the CPU or the SPU. The ID field is currently all 0s (Table 2–5).

Parity bits (XJA PAR[01:00]) are generated for the data bits and parity is checked by the ICU. If no parity error is found, the ICU returns ICU_XFERACK to the XJA to clear its transmit buffers.

If the ICU encountered a parity error, it returns ICU_XFERRETRY to the XJA, which retries the transfer. If more retries are required and also fail, the SCU notes the continuous parity errors, interrupts the CPU, and executes a service routine.

When the ICU transfers the status word from its receive buffer to the JBox, it asserts ICU_BUFEMPTD to the XJA to inform the XJA of the availability of an ICU receive buffer. ICU_BUFEMPTD is also asserted if the ICU commands a retry.

CYCLE	TYPE	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	COMMAND	R	R	ID						R	R	R	R	CMD			

R = RESERVED

MR_X0357_89

Figure 2–13 CPU Status Command Cycle

2.5 CPU Write

Only one CPU transaction can be outstanding at a time. All CPU transactions are longwords only. Figure 2–14 is a flow diagram of the transfer functions that execute on the JXDI during a CPU write transaction. Refer to the flow diagram throughout the following discussion.

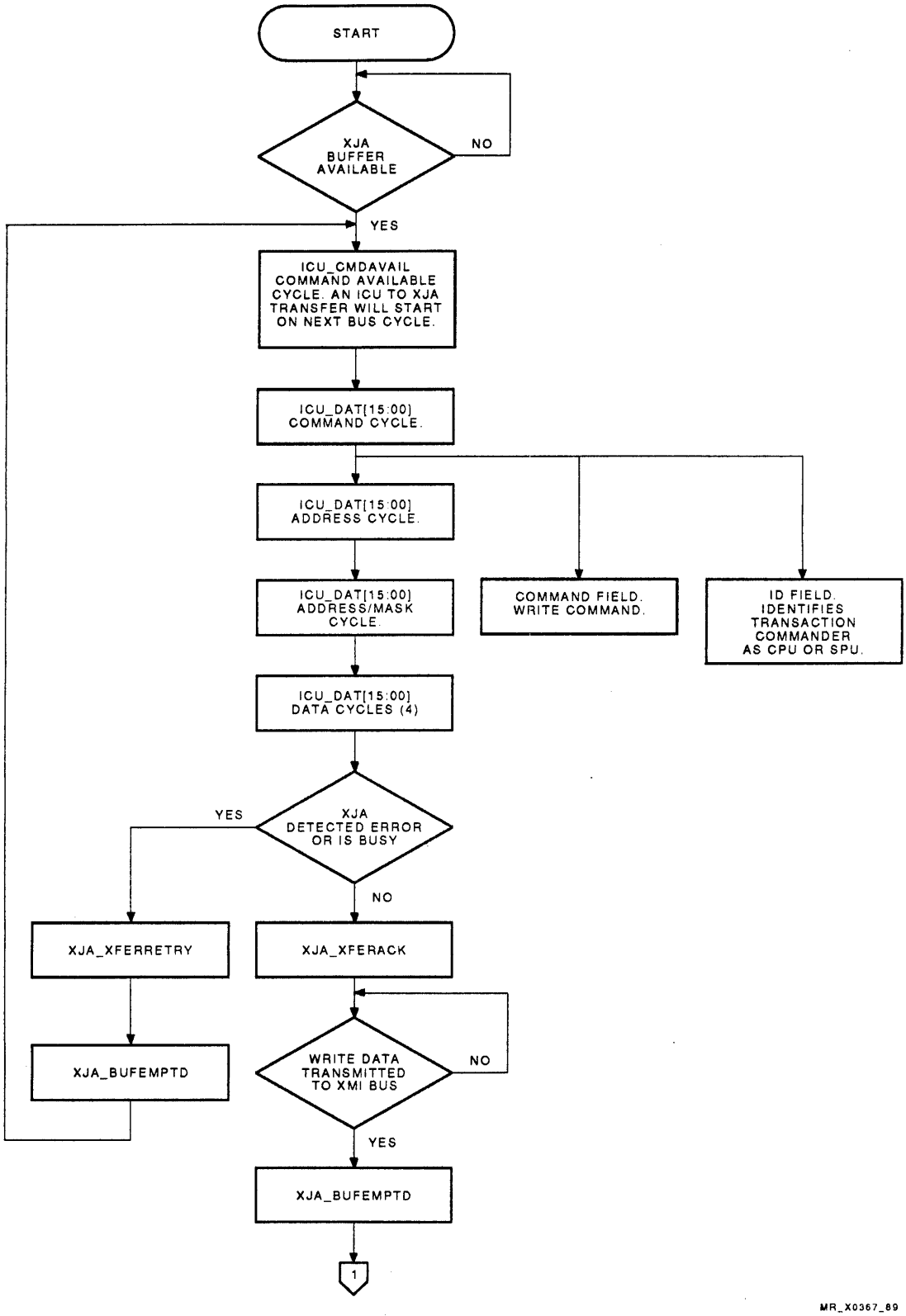


Figure 2–14 (Cont.) Flow Diagram of JXDI CPU Write

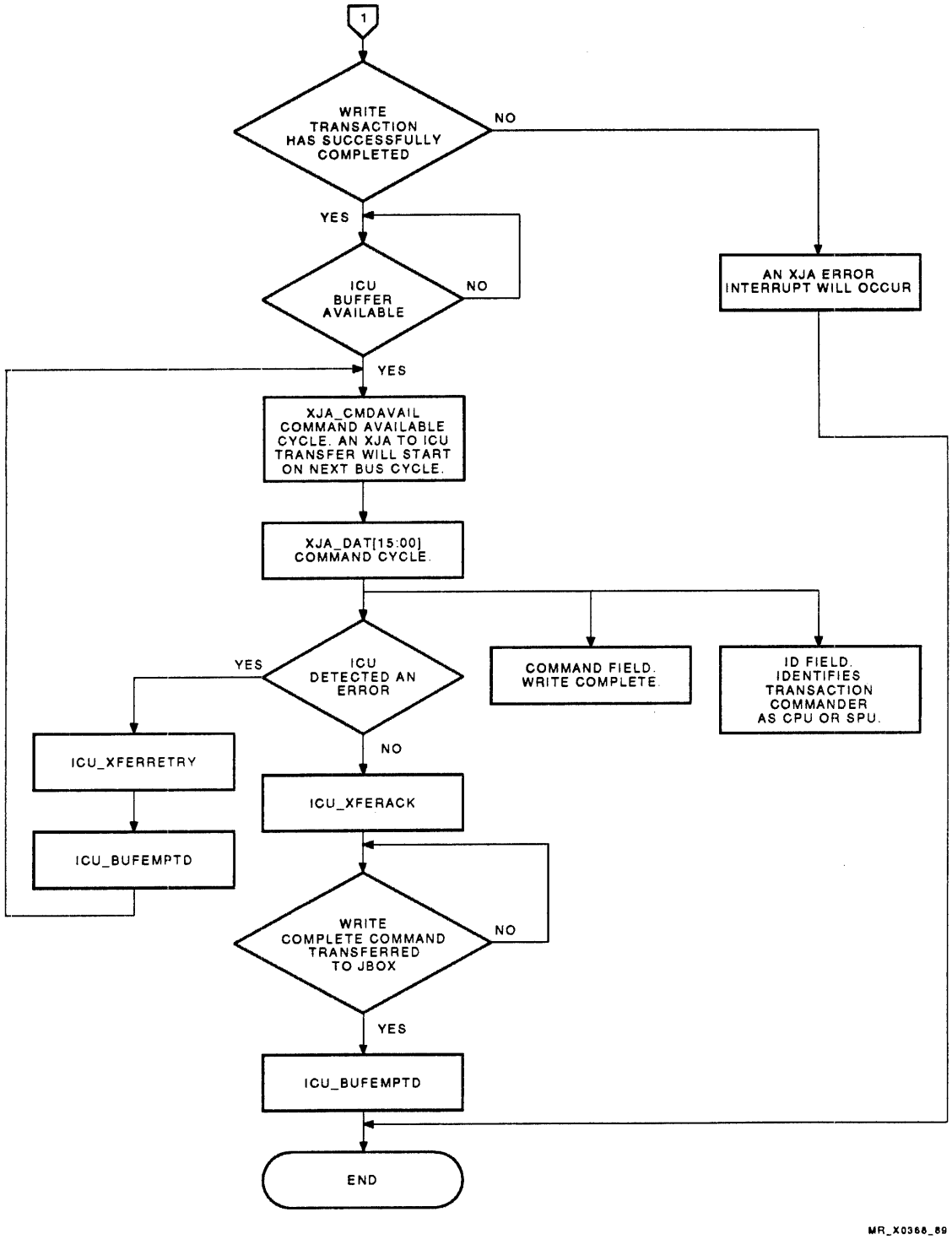


Figure 2–14 Flow Diagram of JXDI CPU Write

2.5.1 Command Available Cycle

The ICU asserts ICU_CMDAVAIL (command available) to inform the XJA that a transfer follows starting with the next cycle. The ICU is only allowed to assert ICU_CMDAVAIL if an XJA buffer is available to receive the transmitted information (Section 2.2.8).

2.5.2 Write Command Cycle

The ICU data lines (ICU_DAT[15:00]) carry command, address, mask, and data information to the XJA. The format of the information is illustrated in Figure 2–15.

The command cycle contains a command field and an ID field.

The command field occupies data lines ICU_DAT[03:00]. Table 2–2 defines the command field code which for this transaction is 0100 (write request).

The ID field occupies data lines ICU_DAT[13:08]. The field is used to identify the transaction commander as the CPU or the SPU. The ID field is currently all 0s (Table 2–5).

CYCLE	TYPE	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	COMMAND	R	R	ID						R	R	R	R	CMD			
1	ADDRESS 0	A [29:26, 05:02]								A [13:06]							
2	ADDRESS 1	MASK				A [25:22]				A [21:14]							
3	DATA	0	0	0	0	0	0	0	0	BYTE 0							
4	DATA	0	0	0	0	0	0	0	0	BYTE 1							
5	DATA	0	0	0	0	0	0	0	0	BYTE 2							
6	DATA	0	0	0	0	0	0	0	0	BYTE 3							

R = RESERVED

MR_X0358_89

Figure 2–15 CPU Write Data Cycles

2.5.3 Address/Mask Cycles

The second data cycle is an address cycle carrying 16 address bits as shown in Figure 2–15. The third data cycle has address bits in the lower 12 bit locations (ICU_DAT[11:00]) and mask bits in locations ICU_DAT[15:12]. Mask bits 15, 14, 13, and 12 are associated with bytes 3, 2, 1, and 0, respectively, of the write data longword and are used by the XJA to determine address bits [01:00] (Section 2.4.3 and Table 2–6).

2.5.4 Data Cycles

The address cycles are followed by four data cycles with each cycle carrying one byte of data. The use of four bus cycles to transfer a data longword is dictated by the XJA receive logic that accepts only one data format. Note in Figure 2–6 that the first longword of data received by the XJA (bytes 0 through 3) is spread over four bus cycles.

The data byte is in the lower order bits (bits [07:00]). The higher order bits ([15:00]) are reserved.

2.5.5 Data Parity

ICU_PAR[01:00] provides odd parity for each bus cycle (command, address, mask, data). Parity bit [01] is parity over data bits [15:08]. Parity bit [00] is parity over data bits [07:00]. The parity bit is asserted when the number of bits asserted in the data field is even.

2.5.6 Acknowledge Cycle and Retry Cycle

The XJA checks parity on each cycle of the received data. If no parity error is found in the data packet, and the XJA is not busy, the XJA asserts XJA_XFERACK (transfer acknowledge) indicating that all the write data was received by the XJA without parity errors. XJA_XFERACK allows the ICU to clear the write data from its transmit buffer.

If a parity error is found or the XJA is busy, the XJA asserts XJA_XFERRETRY (transfer retry). In either case, the ICU must retry the failed transfer. If more retries are required and also fail, the SCU experiences a timeout for failing to move the data packet across the JXDI. The timeout results in a system interrupt and execution of the appropriate service routine.

2.5.7 XJA Buffer Emptied Cycle

When the XJA has transfers the write data from its receive buffer, it returns XJA_ BUFEMPTD to the ICU. XJA_BUFEMPTD indicates that the XJA has emptied the receive buffer and can accept new data from the ICU (Section 2.2.13). If the XJA commanded a retry, it also asserts XJA_BUFEMPTD.

2.5.8 Write Complete Transaction

If the CPU write transaction completed successfully, the XJA notifies the ICU with a write complete transaction. After the ICU receives the write complete status word, it is free to send another CPU request to the XJA. (Only one CPU transaction may be outstanding at a time.)

The write complete transaction starts with the XJA asserting XJA_CMDAVAIL to inform the ICU that a transfer starts on the next cycle.

On the next bus cycle, the XJA transmits the CPU status command (Figure 2–13) on the XJA_DAT[15:00] data lines. The command field contains a write complete command code (1011) as listed in Table 2–2. The ID field identifies the transaction commander as the CPU or the SPU. The ID field is currently all 0s (Table 2–5).

XJA_PAR[01:00] provides odd parity for the write complete command cycle.

The ICU checks parity on the received data and asserts ICU_XFERACK (transfer acknowledge) if the data was received by the ICU with no parity errors. ICU_XFERACK allows the XJA to clear the data from its transmit buffer.

If the ICU detected a parity error, it asserts ICU_XFERRETRY (transfer retry) to the XJA which must retry the failed transfer. If more retries are required and also fail, the SCU notes the continuous parity errors, interrupts the CPU, and executes a service routine.

When the ICU transfers the write complete data from its receive buffer to the JBox, it returns ICU_BUFEMPTD to the XJA. ICU_BUFEMPTD indicates that the ICU has emptied the receive buffer and can accept new data from the XJA. If the ICU commanded a retry, it also asserts ICU_BUFEMPTD.

If the CPU write transaction did not complete successfully, the XJA notifies the SCU by initiating an error interrupt transaction (Section 2.6).

2.6 Interrupt Transactions

When a condition occurs in the XJA or on the XMI bus, requiring the generation of a CPU error interrupt, the XJA transmits an interrupt command to the ICU. If the interrupt is fatal, the XJA asserts the XJA_FATALERR line on the JXDI. If the interrupt is nonfatal, the interrupt takes the form of an interrupt transfer. See Chapter 4 for a detailed description of system interrupts.

The interrupt transfer executes in the same manner as other JXDI transfers (Figure 2–16).

The XJA asserts XJA_CMDAVAIL to the ICU.

In the next bus cycle, the XJA transmits the interrupt command word over the XJA_DAT[15:00] data lines. The command cycle is shown in Figure 2–17. The command field (XJA_DAT[03:00]) contains an interrupt request code (1000) as shown in Table 2–2. An interrupt priority level (IPL) field (XJA_DAT[05:04]) contains the priority level code for the specific type of interrupt. Table 2–7 gives the IPL codes.

The ICU checks parity on the received data and asserts ICU_XFERACK (transfer acknowledge) if the interrupt cycle was received by the ICU with no parity errors. ICU_XFERACK allows the XJA to clear the data from its transmit buffer.

If the ICU detected a parity error, it asserts ICU_XFERRETRY (transfer retry) to the XJA which must retry the failed transfer. If more retries are required and also fail, the SCU notes the continuous parity errors, interrupts the CPU, and executes a service routine.

When the ICU transfers the interrupt request data from its receive buffer, it returns ICU_BUFEMPTD to the XJA. ICU_BUFEMPTD indicates that the ICU has emptied the receive buffer and can accept new data from the XJA (Section 2.2.7). ICU_BUFEMPTD also asserts if a retry is commanded.

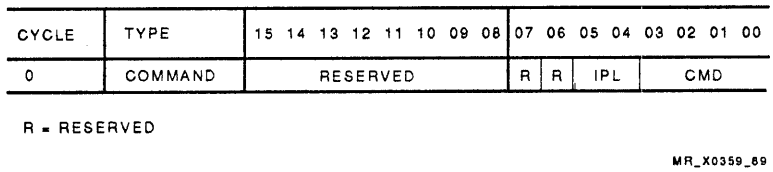


Figure 2–17 Interrupt Request Command Cycle

Table 2–7 IPL Codes

XJA_DAT[05:04]	IPL
0 0	14
0 1	15
1 0	16
1 1	17

2.7 Data Envelopes

Information is transferred over the JXDI in data packets ranging from one bus cycle (DMA and CPU status packets and interrupt packets) to 20 bus cycles (DMA write request/data packet). To optimize performance for the various packet sizes, packet envelopes are used to enclose the data packets. The envelopes need not be filled with data; however, good parity must be provided over the entire envelope. The ICU and the XJA check parity on all envelope cycles, whether or not the cycle contains data.

The envelope should enclose the data packet as tightly as possible to minimize traffic on the JXDI. Sufficient gaps between packets must be provided to allow the XJA and the ICU to respond when one transfer follows immediately after another.

Table 2–8 lists the five transactions discussed in this chapter, the transfer(s) that execute over the JXDI for each transaction, and the direction of each transfer. Also listed is the packet size and envelope size for each transfer. The figures referenced for each transfer illustrate the particular data packet being transferred.

Note that for XJA-to-ICU transfers, 3 envelope sizes are used (8, 12, and 20 cycles). For ICU-to-XJA transfers, 2 envelope sizes are used (12 and 20 cycles).

Table 2–8 JXDI Data Envelopes

Transaction	Transfer	Direction	Figure Number	Packet Size	Envelope Size
DMA read	Read request	ICU ← XJA	2–4	3	8
	Read data return, QW	ICU → XJA	2–6	5	12
	Read data return, OW	ICU → XJA	2–6	9	12
	Read data return, HW	ICU → XJA	2–6	17	20
	Status	ICU → XJA	2–7	1	12
DMA write	Write request/data, QW	ICU ← XJA	2–9	8	8
	Write request/data, OW	ICU ← XJA	2–9	12	12
	Write request/data, HW	ICU ← XJA	2–9	20	20
CPU read	Read request	ICU → XJA	2–11	3	12
	Read data return	ICU ← XJA	2–12	8	8
	Status	ICU ← XJA	2–13	1	8
CPU write	Write request/data	ICU → XJA	2–15	7	12
	Status	ICU ← XJA	2–13	1	8
Interrupt	Interrupt request	ICU ← XJA	2–17	1	8

3
XMI Bus

This chapter provides a detailed description of the XMI bus. It describes the XMI signals and the transactions that execute over the XMI bus, including read, write, and interrupt transactions. Data packet formats for the various types of transactions are also described. XMI address mapping to the VAX 9000 system is also discussed.

NOTE
Chapter 3 is not a complete description of all aspects of the XMI bus. It describes the bus only as it is used within the VAX 9000 system — as the system I/O bus. It does not cover all features and characteristics of the XMI bus nor does it describe XMI signals that are not used by the VAX 9000 system. For example, the XMI bus used as the VAX 9000 I/O bus can have up to 14 nodes, whereas the XMI bus can have up to 16 nodes.

Hexword writes to main memory are not currently supported by the SCU. The XJA does support hexword writes, so they are included in the description of the XMI bus.

3.1 XMI Description

The XMI bus is a limited-length, pended,[1] synchronous bus with centralized arbitration and a 64-ns bus cycle. Several transactions can be in progress at a time, allowing highly efficient use of bus bandwidth. Arbitration and data transfers occur simultaneously. In addition, the bus data lines are multiplexed to carry both data and command/address information. The bus supports quadword, octaword, and hexword reads and writes to memory. In addition, the bus supports longword length read and write operations to I/O space. These longword operations implement byte and word modes required by certain I/O devices.

[1] Nodes do not hold the bus while waiting for a response.

3.1.1 XMI Clock/Arbiter Module (CCARD)

Slot 7 of the XMI card cage contains a clock/arbiter module (CCARD) that:

- Arbitrates among the nodes requesting use of the bus
- Supplies a standard XMI clock to all the nodes

When a node wants to use the XMI bus, it issues a request that the arbiter honors with a grant signal (XMI_GRANT). There are two types of XMI requests, a command request and a response request. A command request asks for the bus to initiate a new transaction. A response request asks for the bus to respond to a previous request; for example, a response request to place read data on the XMI that was requested by a previous read request. The arbiter has two queues: one for command requests and one for response requests. The queues use an algorithm that approximates round-robin. The arbiter gives the response queue priority over the command queue. Therefore, an XMI response request has priority over an XMI command request. Priority is also affected by the position of the XMI adapter in the XMI card cage. The higher numbered slots have a higher priority.

The arbiter also responds to a hold signal (XMI_HOLD) and a suppress signal (XMI_SUP). XMI_GRANT gives the XMI bus to the requesting node for only one cycle. If the node needs the bus for another cycle, it asserts the hold signal and the arbiter allows the node to use the XMI for another cycle. The node can continue holding the bus by keeping XMI_HOLD asserted up to a maximum of four consecutive cycles.

If a node is momentarily unable to keep up with bus traffic, it asserts the suppress signal (XMI_SUP). When XMI_SUP is asserted, the arbiter does not respond to any new command requests until the node has caught up and removes the suppress signal.

The CCARD generates time and phase clock signals that are distributed radially to all the nodes (that is, individual clock signals go to each node). By centrally locating the CCARD on the XMI bus, and by distributing the clocks radially, skew is reduced between nodes and good signal integrity is assured.

3.1.2 XMI Corner

Each module on the XMI bus has a standard interface between the module and the XMI bus called the XMI corner. The corner uses a predefined etch and consists of seven latches and a clock chip. The corner interfaces all the control and data signals going to and coming from the XMI bus.

The clock chip receives two clock signals (XMI_TIME and XMI_PHASE) from the XMI CCARD. The clock chip generates a set of subclocks that are used to control the corner latches. The subclocks are also made available to the node-specific logic on the module. Refer to Chapter 4 for a more detailed description of the XMI corner.

3.2 XMI Signal Descriptions

XMI signals used by the VAX 9000 system are listed in Table 3–1.

Table 3–1 XMI Signals

Class	Signal	Description	XJA Input/Output
Arbitration	XMI_CMD_REQ	Command request	Output
	XMI_RES_REQ	Response request	Output
	XMI_GRANT	Request granted	Input
	XMI_HOLD	Hold the bus	Output
	XMI_SUP	Suppress new requests	Input/output
Information	XMI_FUNCTION[03:00]	Command function	Input/output
	XMI_DATA[63:00]	Data (address, mask, command)	Input/output
	XMI_ID[05:00]	Commander ID	Input/output
	XMI_PARITY[02:00]	Parity over information lines	Input/output
Response	XMI_CNF[02:00]	Confirmation	Input/output
Control	XMI_BAD	Node failure	Input/output
	XMI_FAULT	Uncorrectable error	Input
	XMI_DEFAULT	Idle XMI cycles	Input
	XMI_RESET	Initialize system	Input/output
	XMI_TIME	Clock reference	Input
	XMI_PHASE	Phase reference	Input
	XMI_AC_LO	Low ac line voltage	Input/output
	XMI_DC_LO	Impending loss of dc	Input
Miscellaneous	XMI_NODE_ID[03:00]	Node ID	Input

3.2.1 Arbitration Signals

This section describes the signals that control arbitration for the XMI bus.

3.2.1.1 XMI_CMD_REQ

XMI_CMD_REQ is asserted by an XMI node (including the XJA) when it wants access to the XMI bus to initiate a transaction. The asserting node is the XMI commander. XMI_CMD_REQ is received by the XMI central arbiter.

3.2.1.2 XMI_RES_REQ

XMI_RES_REQ is asserted by an XMI node when it wants access to the XMI bus to respond to a previous command. The asserting node is the XMI responder. The node could have read data requested by an XMI commander, or a vector requested in an interrupt transaction. The XMI central arbiter receives XMI_RES_REQ. The arbiter gives a higher priority to an XMI_RES_REQ response request than to an XMI_CMD_REQ command request.

3.2.1.3 XMI_GRANT

XMI_GRANT is asserted by the XMI arbiter to an XMI node that is granted the bus. The node starts its transfer on the next bus cycle.

3.2.1.4 XMI_HOLD

XMI_HOLD is a wired-OR signal used to implement multicycle transfers. When an XMI node is granted the bus, it controls the bus for the next cycle to transfer its data. The node can assert XMI_HOLD to hold the bus for another cycle regardless of command requests or response requests asserted by other nodes. XMI_HOLD can be asserted for a maximum of four consecutive cycles. It cannot be used to hold the bus for a new transaction.

3.2.1.5 XMI_SUP

XMI_SUP is a wired-OR signal used to suppress the initiation of new transactions on the XMI. XMI_SUP blocks all commander requests. It is used by a node when it is momentarily unable to keep up with bus traffic.

3.2.2 Information Signals

This section describes the signals that carry information over the XMI bus.

3.2.2.1 XMI_FUNCTION[03:00]

The XMI_FUNCTION[03:00] lines specify the function being performed on the XMI bus for the current cycle. Table 3–2 lists all the possible functions.

Table 3–2 Function Codes

XMI_FUNCTION[03:00]	Function
0 0 0 0	Null
0 0 0 1	Command cycle
0 0 1 0	Write data cycle
0 0 1 1	Reserved[1]
0 1 0 0	Locked response
0 1 0 1	Read error response
0 1 1 0	Reserved[1]
0 1 1 1	Reserved[1]
1 0 n n[2]	Good read data
1 1 n n[2]	Corrected read data

[1]Interpreted as a null function

[2]n n = Data cycle number
0 0 = Data cycle 0
0 1 = Data cycle 1
1 0 = Data cycle 2
1 1 = Data cycle 3

3.2.2.2 XMI_DATA[63:00]

The XMI_DATA[63:00] data lines are multiplexed between command and data information. On data cycles the lines carry 64 bits of read or write data; on command cycles the lines carry command, address, mask, and length information.

3.2.2.3 XMI_ID[05:00]

The XMI_ID[05:00] lines identify the node commander on the XMI bus and the transaction being referenced. The node ID is the upper four bits of the ID field (XMI_ID[05:02]) as shown in Table 3–3. The lower two bits of the ID field (XMI_ID[01:00]) identify the transaction number. An XMI node can have up to four transactions outstanding at one time. The transaction number is used to relate a response to the command associated with the response. For example, the transaction number associates a read-data-return transfer with the command that requested the data.

Table 3–3 XMI Node ID Codes

XMI_ID[05:00]	XMI Node Number
0 0 0 1 n n	1
0 0 1 0 n n	2
0 0 1 1 n n	3
0 1 0 0 n n	4
0 1 0 1 n n	5
0 1 1 0 n n	6
0 1 1 1 n n	7
1 0 0 0 n n	8
1 0 0 1 n n	9
1 0 1 0 n n	10
1 0 1 1 n n	11
1 1 0 0 n n	12
1 1 0 1 n n	13
1 1 1 0 n n	14

nn = Transaction number

3.2.2.4 XMI_PARITY[02:00]

XMI_PARITY[02:00] are three parity bits providing even parity over the other three information signals. Table 3–4 shows the information signals covered by each parity bit.

Table 3–4 Parity Coverage

Parity Bit	Signal
XMI_PARITY[02]	XMI_FUNCTION[03:00] and XMI_ID[05:00]
XMI_PARITY[01]	XMI_DATA[63:32]
XMI_PARITY[00]	XMI_DATA[31:00]

3.2.3 Response Signal: XMI_CNF[02:00]

XMI_CNF[02:00] are three lines used by a receiving node to notify the transmitting node of the status of the information transfer. Every transfer bus cycle must have a confirmation response. There are two confirmation responses as shown in Table 3–5.

Acknowledge means that a receiving node has accepted the transmitted information. No response means that no node has accepted the information. No response may be due to the target node detecting a parity error or being too busy to accept the information at this time.

Table 3–5 Confirmation Codes

XMI_CNF[02:00]	Response
0 0 0	No response
1 1 1	Acknowledge (accepted data)

3.2.4 Control Signals

This section describes signals that control operations on the XMI bus.

3.2.4.1 XMI_BAD

XMI_BAD reports the failure of a node to pass its self-test. XMI_BAD remains asserted by a node until the node passes its self-test. On power-up, all nodes must pass their self-test before this line negates.

3.2.4.2 XMI_FAULT

XMI_FAULT is a wired-OR signal line that is asserted by a node (except the XJA) when it detects an uncorrectable error of a catastrophic system-wide nature. The XJA monitors but does not drive this line. XMI_FAULT is asserted for one cycle during which all other nodes latch the signal.

3.2.4.3 XMI_DEFAULT

XMI_DEFAULT is generated by the XMI arbiter during idle XMI cycles, to default the bus. It is routed only to the adapters in slot 1 (first slot) and slot 14 (last slot).

3.2.4.4 XMI_RESET

XMI_RESET is asserted by any node that needs to initialize the system to the power-up state. The initialization is done by the CCARD and involves the sequencing of XMI_AC_LO and XMI_DC_LO.

3.2.4.5 XMI_TIME

XMI_TIME is a 21.3-ns square wave clock (46.9 MHz) generated by the CCARD. XMI_TIME provides a time reference for the XMI nodes.

3.2.4.6 XMI_PHASE

XMI_PHASE is generated by the CCARD and is used to reference the XMI_TIME clock to the beginning of an XMI cycle.

3.2.4.7 XMI_AC_LO

XMI_AC_LO indicates that the ac line voltage is below specifications.

3.2.4.8 XMI_DC_LO

XMI_DC_LO warns of the impending loss of dc power. It is also used for initialization on power-up.

3.2.5 Miscellaneous Signal: XMI_NODE_ID[03:00]

Each slot on the XMI backplane is wired with a unique 4-bit ID code. This code (XMI_NODE_ID[03:00]) is used by the node as XMI_ID[05:02] to identify the node during XMI transactions (Section 3.2.2.3).

3.3 Read and Read Lock Transactions

This section shows — cycle by cycle — the various types of read transactions. Figure 3–1 is a bus cycle diagram for longword, quadword, octaword, and hexword read (or read lock) transactions.

CYCLE	0	1	2	3		4	5	6	7	8	9	10
XMI_CMD_REQ	REQ											
XMI_RES_REQ						RES						
XMI_GRANT	GRNT					GRNT						
XMI_HOLD					...							
XMI_FUNCTION[03:00]		CMD					GRD0					
XMI_DATA[63:00]		READ					DAT0					
XMI_ID[05:00]		CMDR					CMDR					
XMI_PARITY[02:00]		PAR					PAR					
XMI_CNF[02:00]				ACK					ACK			

A. LONGWORD OR QUADWORD READ

CYCLE	0	1	2	3		4	5	6	7	8	9	10
XMI_CMD_REQ	REQ											
XMI_RES_REQ						RES						
XMI_GRANT	GRNT					GRNT						
XMI_HOLD					...		HOLD					
XMI_FUNCTION[03:00]		CMD					GRD0	GRD1				
XMI_DATA[63:00]		READ					DAT0	DAT1				
XMI_ID[05:00]		CMDR					CMDR	CMDR				
XMI_PARITY[02:00]		PAR					PAR	PAR				
XMI_CNF[02:00]				ACK					ACK	ACK		

B. OCTAWORD READ

CYCLE	0	1	2	3		4	5	6	7	8	9	10
XMI_CMD_REQ	REQ											
XMI_RES_REQ						RES						
XMI_GRANT	GRNT					GRNT						
XMI_HOLD					...		HOLD	HOLD	HOLD			
XMI_FUNCTION[03:00]		CMD					GRD0	GRD1	GRD2	GRD3		
XMI_DATA[63:00]		READ					DAT0	DAT1	DAT2	DAT3		
XMI_ID[05:00]		CMDR					CMDR	CMDR	CMDR	CMDR		
XMI_PARITY[02:00]		PAR					PAR	PAR	PAR	PAR		
XMI_CNF[02:00]				ACK					ACK	ACK	ACK	ACK

C. HEXWORD READ

MR_X0259_89

Figure 3–1 Read or Read Lock Transactions

3.3.1 Longword and Quadword Read

Part A of Figure 3–1 shows the bus cycles involved in a longword or quadword read (or read lock) transaction. Longword transactions are to I/O space only. Quadword transactions are to memory space only.

3.3.1.1 Arbitration

The commander node asserts XMI_CMD_REQ to request the XMI bus. When the arbiter returns XMI_GRANT (cycle 0), the commander begins its transfer in the next cycle (cycle 1). XMI_GRANT gives the commander control of the bus only for the next cycle.

3.3.1.2 Command Cycle

The function lines (XMI_FUNCTION[03:00]) specify cycle 1 as a command cycle (0001) (Table 3–2).

The data lines (XMI_DATA[63:00]) contain the command, length, and address fields as shown in part A of Figure 3–2.

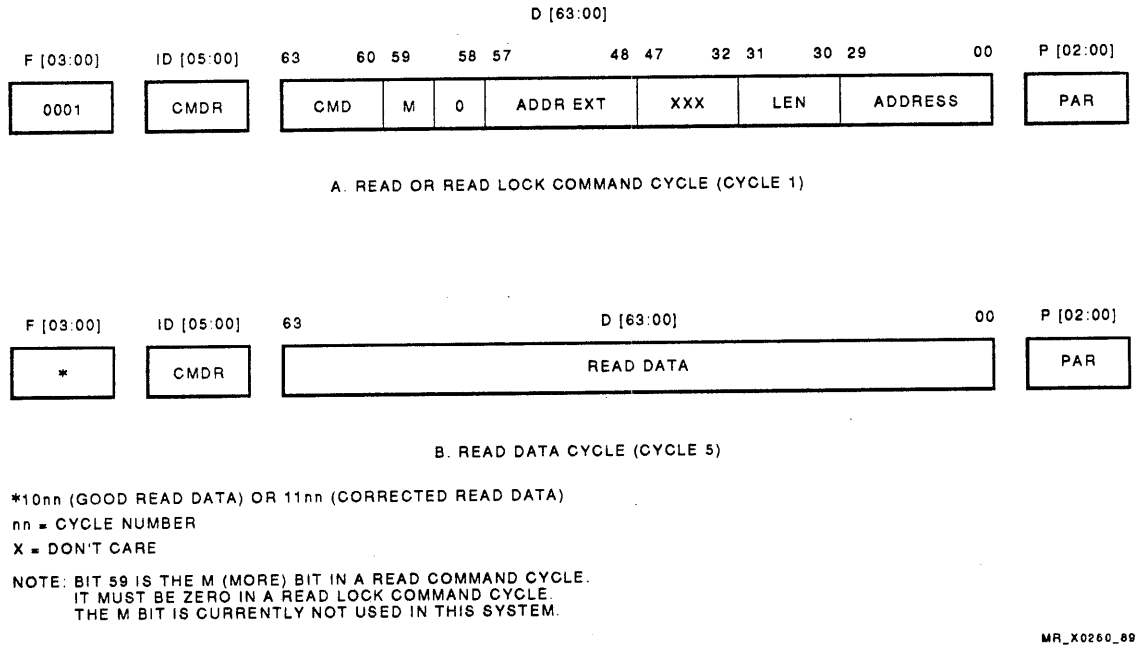


Figure 3–2 Command and Data Cycles for a Read or Read Lock Transaction

Data bits [63:60] comprise the command field specifying this transaction to be a read or a read lock operation. Table 3–6 lists the command codes. Bits [63:60] are 0001 (read) or 0010 (read lock).

Table 3–6 Command Codes

XMI DATA[63:60]	Command
0 0 0 1	Read
0 0 1 0	Read lock
0 1 1 0	Write unlock
0 1 1 1	Write
1 0 0 0	Interrupt
1 0 0 1	Identify
1 1 1 1	Vector interrupt

Data bits [31:30] comprise the length field specifying the size of the read transaction. Table 3–7 lists the length codes. For this transfer, bits [31:30] are 01 (longword) or 10 (quadword).

Table 3–7 Length Codes

XMI_DATA[31:30]	Length
0 0	Hexword
0 1	Longword
1 0	Quadword
1 1	Octaword

The address of the read data is specified by data bits [29:00] with bits [57:48] as the address extension (Section 3.5).

The ID lines (XMI_ID[05:00]) identify the transmitting node as the transaction commander. Table 3–3 lists the node ID codes.

The commander generates parity over the function, data, and ID lines as shown in Table 3–4. The parity bits are placed on the XMI bus in cycle 1 as XMI_PARITY[02:00].

3.3.1.3 Acknowledge Cycle

All the XMI nodes monitor the address placed on the XMI bus by the commander. When a node recognizes its address, it latches the bus data and executes the read function.

The receiving node, in addition to receiving the information off the function, data, and ID lines, also receives the three XMI parity bits (XMI_PARITY[02:00]). Parity is checked; if no error is found and the receiving node is able to accept the data at this time, the node returns an acknowledge code (Table 3–5) to the transmitter. Otherwise, a no response code is returned. The acknowledge cycle (cycle 3) is the second cycle after the command cycle. An acknowledge cycle is always the second cycle after the information cycle it is acknowledging.

3.3.1.4 Arbitration

When the responder is ready to send the requested read data to the commander, it arbitrates for the XMI bus by asserting XMI_RES_REQ. The arbiter gives a higher priority to a response request than to a new transaction request. When the arbiter grants the bus to the responder, it returns XMI_GRANT (cycle 4[1]) to the responder, which begins its transfer in the next cycle (cycle 5). XMI_GRANT gives the responder control of the bus for only one cycle.

3.3.1.5 Read Data Cycle

In cycle 5 (part B of Figure 3–2), the responder asserts a 1000 code on the XMI function lines, specifying cycle 5 as a good read-data-return cycle (Table 3–2). If the data is corrected read data, the function lines carry a 1100 code (Section 3.3.4.1).

The data lines (XMI_DATA[63:00]) contain the requested read data. If the transaction length is a longword, the longword is contained in bits [31:00]. Bits [63:32] are unspecified but must give good parity. If the transaction length is a quadword, all 64 data bits are used.

The responder places the commander ID on the ID lines (XMI_ID[05:00]). The commander monitors the bus traffic waiting for its return data. When the commander detects its ID, it latches the data lines that contain the read data requested.

XMI_PARITY[02:00] provides parity for the function, data, and ID lines.

3.3.1.6 Acknowledge Cycle

The transaction commander, in addition to receiving the information from the function, data, and ID lines, also receives the three XMI parity bits (XMI_PARITY[02:00]). Parity is checked; if no error is found and the commander is able to accept the data at this time, the commander returns an acknowledge code (Table 3–5) to the responder. Otherwise, a no response code is returned. The acknowledge cycle (cycle 7) is the second cycle after the good read-data-return cycle.

3.3.2 Octaword Read

Part B of Figure 3–1 shows the bus cycles involved in an octaword read transaction. Octaword transactions are to memory space only.

The first half of an octaword read transaction (arbitration, command cycle, and acknowledge cycle) is identical to the first half of a longword or quadword read transaction except that the length field of the command data word specifies an octaword length instead of a longword or quadword length.

3.3.2.1 Arbitration

The commander node asserts XMI_CMD_REQ to request the XMI bus. When the arbiter returns XMI_GRANT (cycle 0), the commander begins its transfer in the next cycle (cycle 1). XMI_GRANT gives the commander control of the bus only for the next cycle.

[1] Plus the cycles that passed while the responder was obtaining the read data.

3.3.2.2 Command Cycle

The function lines (XMI_FUNCTION[03:00]) specify cycle 1 as a command cycle (0001) (Table 3–2).

The data lines (XMI_DATA[63:00]) contain the command, length, and address fields as shown in Figure 3–2. Bits [63:60] comprise the command field specifying this transaction to be a read or a read lock operation. Table 3–6 lists the command codes. Bits [63:60] are 0001 (read) or 0010 (read lock).

Data bits [31:30] comprise the length field specifying the size of the read transaction. Table 3–7 lists the length codes. For this transfer, bits [31:30] are 11 (octaword).

The address of the read data is specified by data bits [29:00] with bits [57:48] as the address extension (Section 3.5).

The ID lines (XMI_ID[05:00]) identify the transmitting node as the transaction commander. Table 3–3 lists the node ID codes.

The commander generates parity over the function, data, and ID lines as shown in Table 3–4. The parity bits are placed on the XMI bus as XMI_PARITY[02:00].

3.3.2.3 Acknowledge Cycle

The receiving node, in addition to receiving the information from the function, data, and ID lines, also receives the three XMI parity bits (XMI_PARITY[02:00]). Parity is checked; if no error is found and the receiving node is able to accept the data at this time, the node returns an acknowledge code (Table 3–5) to the transmitter. Otherwise, a no response code is returned.

3.3.2.4 Arbitration

When the responder is ready to send the requested read data to the commander, it arbitrates for the XMI bus by asserting XMI_RES_REQ. When the arbiter grants the bus to the responder, it returns XMI_GRANT (cycle 4) to the responder, which begins its transfer in the next cycle (cycle 5). XMI_GRANT gives the responder control of the bus for only one cycle.

3.3.2.5 Read Data Cycles

In cycle 5, the responder asserts XMI_HOLD to keep the bus for the second read data return cycle. XMI_HOLD holds the bus only for the cycle after which it is asserted.

In cycle 5, the responder asserts a 1000 code on the XMI function lines, specifying cycle 5 as data cycle 0 of good return-read data (or a 1100 code specifying the data as corrected read data). In cycle 6 the responder asserts a 1001 code on the function lines, specifying cycle 6 as data cycle 1 of good return-read data (or a 1101 code specifying the data as corrected read data) (Table 3–2).

The data lines (XMI_DATA[63:00]) contain the requested read data. Each data cycle carries a quadword of data.

The responder places the commander ID on the ID lines for each of the data cycles. This is necessary as the two quadwords may be returned to the transaction commander in separate transfers (Section 3.3.5). When the commander detects its ID, it latches the data lines that contain the read data requested.

XMI_PARITY[02:00] provide parity for the function, data, and ID lines in each of the data cycles.

3.3.2.6 Acknowledge Cycles

The transaction commander checks parity for each data cycle; if no error is found and the commander is able to accept the data at this time, the commander returns an acknowledge code (Table 3–5) to the responder for each data cycle. The acknowledge code occurs in the second cycle after its associated data cycle. If a parity error is detected by the commander, a no response code is returned for that data cycle.

3.3.3 Hexword Read

The bus cycles involved in a hexword read are shown in part C of Figure 3–1. Hexword transactions are to memory space only.

The first half of a hexword read is identical to the first half of a longword, quadword, or octaword read, except that the length field (XMI_DATA[31:30]) of the command data longword in cycle 1 specifies a hexword code of 00 (Table 3–7).

The bus cycles in the second (response) half of a hexword read have the following differences from those in the response half of an octaword read:

1. Four read-data-return cycles, instead of two, transfer the four quadwords of the hexword.
2. The function code on the XMI_FUNCTION[03:00] lines for the four data cycles are as follows (Table 3–2):

   Cycle 0 = 1000 (good read data) or 1100 (corrected read data)
   Cycle 1 = 1001 (good read data) or 1101 (corrected read data)
   Cycle 2 = 1010 (good read data) or 1110 (corrected read data)
   Cycle 3 = 1011 (good read data) or 1111 (corrected read data)

3. XMI_HOLD is asserted for three cycles to hold the XMI bus long enough to transfer all the data quadwords.
4. An acknowledge confirmation code is placed on the XMI bus for four cycles, one for each of the data quadwords transmitted.

3.3.4 Read Error Responses

This section describes the three types of errors that can occur during a read transaction:

Correctable read error
Uncorrectable read error
Read-locked error

3.3.4.1 Correctable Read Error

When a responder corrects an error in the read data, it places a corrected read data response code (11nn) on the function lines (Table 3–2). As shown in the table, *nn* indicates the data cycle that has the corrected data. Otherwise, the transaction continues as a normal read.

3.3.4.2 Uncorrectable Read Error

Figure 3–3 shows the bus cycles involved in a read transaction where an uncorrectable error is detected. When the responder detects an uncorrectable error in the read data, it places a read error response (0101) on the function lines (Table 3–2), the commander ID on the ID lines, and parity bits on the parity lines. The data lines are unspecified but must provide good parity. The transfer is terminated at this point and no further read data is transmitted. Figure 3–3 is a hexword read transaction that terminates on the third data cycle (cycle 7) when an uncorrectable error is detected. The third and fourth data quadwords are not transmitted.

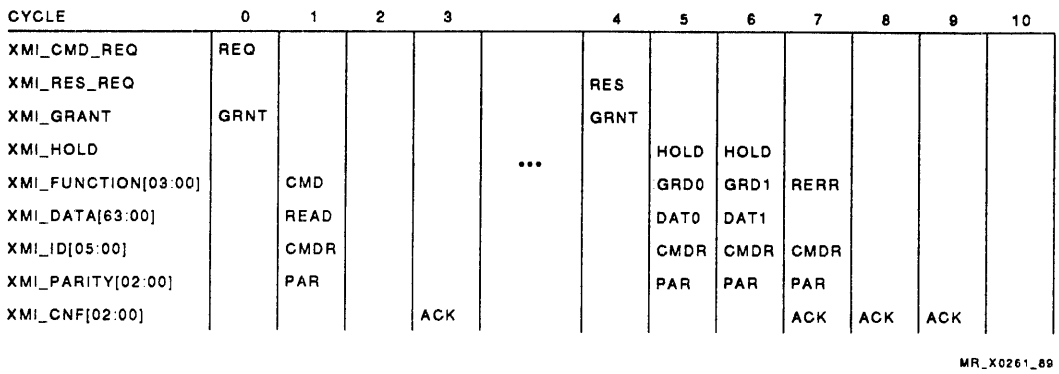


Figure 3–3 Uncorrectable Read Error in a Hexword Read Transaction

3.3.4.3 Read-Locked Error

Figure 3–4 shows the bus cycles involved in a read-lock transaction made to a location that is already read-locked. In the command cycle (cycle 1), data bits XMI_DATA[63:60] specify a read lock transaction (0010) (Table 3–6). When the responder determines that the addressed location is already read-locked, it places a locked response (0100) on the function lines (Table 3–2), the commander ID on the ID lines, and parity bits on the parity lines. The data lines are unspecified but must provide good parity.

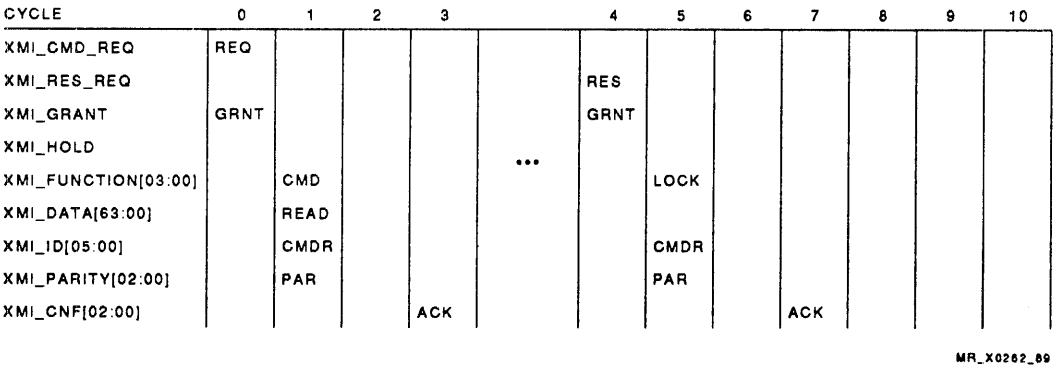


Figure 3–4 Read Lock Transaction to a Locked Location

3.3.5 Multiple Read-Data-Return Transfers

When a responder is not able to return all the requested read data in consecutive data cycles, it returns the data in separate transfers. The responder rearbitrates for the bus for each transfer. Each data cycle contains the commander ID because the commander must be able to recognize each transfer as part of the data it requested.

Figure 3–5 shows an octaword read transaction with the data quadwords returned in two separate transfers. Note that the second data cycle (cycle 9) has a function code specifying the data as the second quadword of good read data (1001) (Table 3–2). Each data quadword is thereby identified in the data transaction.

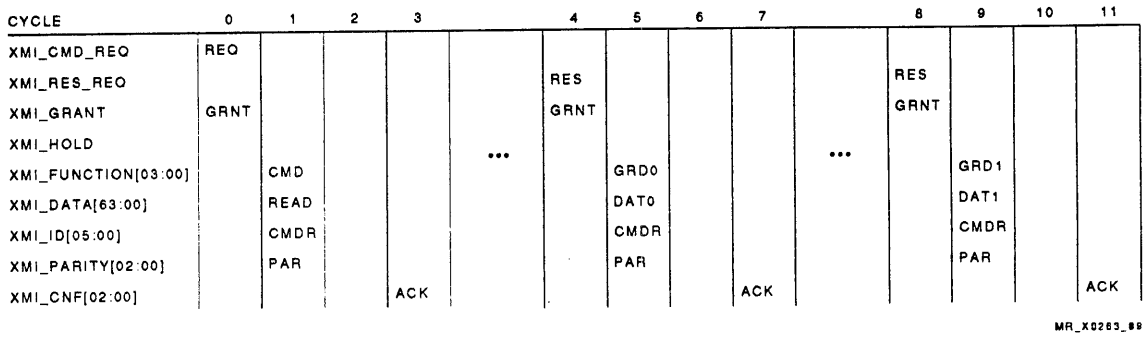


Figure 3–5 Octaword Read Transaction with Multiple Read-Data-Return Transfers

3.4 Masked Write and Write Unlock Transactions

This section shows — cycle by cycle — the various types of masked write transactions. All write transactions on the XMI bus are masked writes. Those transactions where all the data is written, have all their mask bits asserted (no bytes are masked). In the following discussion, masked writes are referred to as writes.

Write bus cycles are similar to read bus cycles. The commander gains the XMI bus through arbitration and then sends a command cycle, specifying a command, mask field, length, and address. Following the command cycle, the commander transmits one, two, or four cycles of write data. Unlike the read return data, the write data must be transmitted in consecutive cycles with no null cycles between.

Figure 3–6 is a bus cycle diagram for longword, quadword, octaword, and hexword write (or write unlock) transactions.

3.4.1 Longword and Quadword Write

Part A of Figure 3–6 shows the bus cycles involved in a longword or quadword write (or write unlock) transaction. Longword transactions are to I/O space only. Quadword transactions are to memory space only.

CYCLE	0	1	2	3	4	5	6	7	8	9	10
XMI_CMD_REQ	REQ										
XMI_RES_REQ											
XMI_GRANT	GRNT										
XMI_HOLD		HOLD									
XMI_FUNCTION[03:00]		CMD	WDAT								
XMI_DATA[63:00]		WRT	DAT0								
XMI_ID[05:00]		CMDR									
XMI_PARITY[02:00]		PAR	PAR								
XMI_CNF[02:00]				ACK	ACK						

A. LONGWORD OR QUADWORD WRITE

CYCLE	0	1	2	3	4	5	6	7	8	9	10
XMI_CMD_REQ	REQ										
XMI_RES_REQ											
XMI_GRANT	GRNT										
XMI_HOLD		HOLD	HOLD								
XMI_FUNCTION[03:00]		CMD	WDAT	WDAT							
XMI_DATA[63:00]		WRT	DAT0	DAT1							
XMI_ID[05:00]		CMDR									
XMI_PARITY[02:00]		PAR	PAR	PAR							
XMI_CNF[02:00]				ACK	ACK	ACK					

B. OCTAWORD WRITE

CYCLE	0	1	2	3	4	5	6	7	8	9	10
XMI_CMD_REQ	REQ										
XMI_RES_REQ											
XMI_GRANT	GRNT										
XMI_HOLD		HOLD	HOLD	HOLD	HOLD						
XMI_FUNCTION[03:00]		CMD	WDAT	WDAT	WDAT	WDAT					
XMI_DATA[63:00]		WRT	DAT0	DAT1	DAT2	DAT3					
XMI_ID[05:00]		CMDR									
XMI_PARITY[02:00]		PAR	PAR	PAR	PAR	PAR					
XMI_CNF[02:00]				ACK	ACK	ACK	ACK	ACK			

C. HEXWORD WRITE

MR_X0264_89

Figure 3–6 Write or Write Unlock Transactions

3.4.1.1 Arbitration

The commander node asserts XMI_CMD_REQ to request the XMI bus. When the arbiter returns XMI_GRANT (cycle 0), the commander begins its transfer in the next cycle (cycle 1). XMI_GRANT gives the commander control of the bus only for the next cycle.

3.4.1.2 Command Cycle

During the command cycle (cycle 1), the commander asserts XMI_HOLD to hold the bus for the data cycle that is to follow.

The function lines (XMI_FUNCTION[03:00]) specify the cycle as the command cycle (0001) (Table 3–2).

The data lines (XMI_DATA[63:00]) contain the command, length, mask, and address fields as shown in part A of Figure 3–7.

Data bits [63:60] comprise the command field specifying this transaction to be a write or a write unlock operation. Table 3–6 lists the command codes. Bits [63:60] are 0111 (write) or 0110 (write unlock).

Data bits [47:32] comprise the 16-bit mask field. Bits 32 through 47 are respectively associated with bytes 0 through 16 for longword, quadword, and octaword length transactions. An asserted mask bit (1) writes the corresponding byte. A negated mask bit (0) masks the corresponding byte. Unused mask bits in a longword or quadword transaction are 0. All the mask bits are used in an octaword transaction. In a hexword transaction, the mask bits are ignored and all 32 bytes are written.

Data bits [31:30] comprise the length field specifying the size of the write transaction. Table 3–7 lists the length codes. For this transfer, bits [31:30] are 01 (longword) or 10 (quadword).

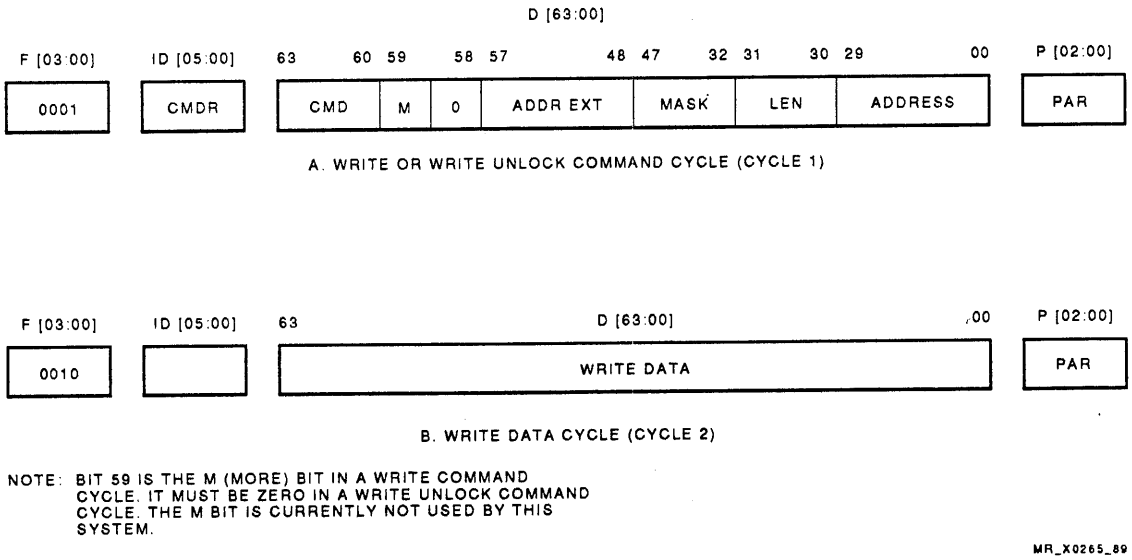


Figure 3–7 Command and Data Cycles for a Write or Write Unlock Transaction

The address of the write data is specified by data bits [29:00] with bits [57:48] as the address extension (Section 3.5).

The ID lines (XMI_ID[05:00]) identify the transmitting node as the transaction commander. Table 3–3 lists the node ID codes.

The commander generates parity over the function, data, and ID lines as shown in Table 3–4. The parity bits are placed on the XMI bus in the command cycle as XMI_PARITY[02:00].

3.4.1.3 Write Data Cycle

In cycle 2 (part B of Figure 3–7), the commander asserts a write-data cycle code (0010) on the XMI function lines (Table 3–2). The data lines (XMI_DATA[63:00]) contain the write data. If the transaction length is a longword, the longword is contained in bits [31:00]. Bits [63:32] are unspecified but must give good parity.

The commander does not place its ID on the bus during the write data cycle. This was already done in the command cycle.

XMI_PARITY[02:00] provides parity for the function and data lines.

3.4.1.4 Acknowledge Cycle

All the XMI nodes monitor the address placed on the XMI bus by the commander. When a node recognizes its address, it latches the bus data and executes the write function.

The receiving node, in addition to receiving the information from the function, data, and ID lines, also receives the three XMI parity bits (XMI_PARITY[02:00]) for the command cycle and the write data cycle. Parity is checked; if no error is found and the receiving node is able to accept the data at this time, the node returns an acknowledge code (Table 3–5) to the transmitter. Otherwise, a no response code is returned. Cycles 3 and 4 are the respective acknowledge cycles for the command and data cycles. An acknowledge cycle is always the second cycle after the information cycle it is acknowledging.

3.4.2 Octaword Write

Part B of Figure 3–6 shows the bus cycles involved in an octaword write transaction. Octaword transactions are to memory space only.

An octaword write is identical to a quadword write except for the following:

- XMI_HOLD is asserted for two cycles, instead of one, to hold the XMI bus for another data cycle to transfer the second quadword.
- The length field (XMI_DATA[31:30]) in cycle 1 specifies an octaword code (11) instead of a quadword code (Table 3–7).
- Two write data cycles, instead of one, transfer the two quadwords of the octaword.
- An acknowledge confirmation code is placed on the XMI bus for three cycles, instead of two: one for the command cycle, and one each for the two data cycles.

Note that the function code on the XMI_FUNCTION[03:00] lines for the two data cycles is the same (write data = 0010). The data cycles are not numbered (as they are for read return data) because all the write data must follow the command cycle with no null cycles between.

3.4.3 Hexword Write

Part C of Figure 3–6 shows the bus cycles involved in a hexword write transaction. Hexword transactions are to memory space only.

A hexword write is identical to an octaword write except for the following:

- XMI_HOLD is asserted for four cycles, instead of two to hold the XMI bus for two more data cycles to transfer the third and fourth quadwords.
- The length field (XMI_DATA[31:30]) in cycle 1 specifies a hexword code (00) instead of an octaword code (Table 3–7).
- Four write data cycles, instead of two, transfer the four quadwords of the hexword.
- An acknowledge confirmation code is placed on the XMI bus for five cycles, instead of three: one for the command cycle, and one each for the four data cycles.

The function code on the XMI_FUNCTION[03:00] lines for the four data cycles is the same (write data = 0010). The data cycles are not numbered.

3.5 XMI Address Mapping

The XMI bus has 40 bits of physical address space. When the MSB of an XMI address (bit 39) is 0, the XMI address is to memory. When bit 39 is 1, the XMI address is to I/O space. I/O space includes the XJA along with other nodes on the XMI bus.

The XMI data lines are multiplexed to carry both data and the XMI address as shown in Figures 3–2 and 3–7. XMI addresses are carried on data lines XMI_DATA[29:00] and XMI_DATA[57:48]. The XMI address bits are mapped to the XMI data lines as follows:

XMI_ADDRESS[28:00] → XMI_DATA[28:00]
XMI_ADDRESS[38:29] → XMI_DATA[57:48]
XMI_ADDRESS[39] → XMI_DATA[29]

XMI address bit 39 is mapped to data line 29 for system compatibility. An earlier version of the XMI bus had 30 address bits with bit 29 being the MSB. Systems using the 30-bit XMI bus used bit 29 (the MSB) for separation of memory space and I/O space. To make the 40-address-bit XMI bus compatible with these systems, data line 29 still carries the most significant address bit — now address bit 39.

The XMI address bits are directly mapped to the VAX 9000 address bits as follows:

XMI_ADDRESS[33:00] → VAX 9000 ADDRESS[33:00]

Table 3–8 illustrates the mapping between the JXDI address bits, the XMI address bits, and the XMI data bits. See Chapter 1 for a detailed description of I/O space addressing.

Table 3–8 XMI Address Mapping

JXDI Address Bits	XMI Address Bits	XMI Data Bits
28:00	28:00	28:00
33:29	33:29	52:48
–	38:34	57:53
–	39	29

3.6 Interrupt Transactions

Any device on the XMI bus can send an interrupt command to the system CPU by directing the interrupt to the XJA node. Two types of interrupts execute on the XMI bus: basic interrupts and implied vector interrupts (IVINTR).

A basic interrupt involves two XMI transactions:

- An interrupt (INTR) transaction where the interrupting node transmits an INTR cycle to the XJA requesting an interrupt of the CPU
- An identify (IDENT) transaction where the XJA issues an IDENT command requesting a vector from the interrupting node, and the interrupting node responds with an IDENT response containing the requested vector

An IVINTR transaction consists of one XMI transfer where the interrupting node sends a specific type of interrupt to the XJA. The required starting vector is implied by the type of interrupt.

All of the transfers involved in interrupts are single-cycle transfers. For each transfer, the transmitting node must arbitrate for the bus and receive XMI_GRANT before transmitting its data. In the second cycle after transmitting its data, the node receives an acknowledgment on the XMI confirmation lines. Parity is generated and checked on all interrupt transfers.

3.6.1 Basic Interrupts

Figure 3–8 shows the bus cycles that execute for the basic interrupt transactions.

3.6.1.1 INTR Transaction

An XMI device wishing to interrupt the CPU arbitrates for the bus with a command request (XMI_CMD_REQ). Upon receiving XMI_GRANT from the arbiter, it places function, ID, data, and parity bits on the XMI bus (part A of Figure 3–8).

The function bits specify a command cycle (Table 3–2). The ID bits identify the transmitting node (node 14 in this example, Table 3–3). The command field of the data longword specifies an interrupt command (Table 3–6). Bits [19:16] of the data longword are the interrupt priority level (IPL) field specifying the IPL of the interrupt. Table 3–9 gives the IPL codes for bits [19:16]. The example shows an interrupt priority level of 14.

Table 3–9 IPL Codes

XMI_DATA[19:16]	IPL
0 0 0 1	14
0 0 1 0	15
0 1 0 0	16
1 0 0 0	17

Bits [15:00] of the data word are the interrupt node specifier field (Figure 3–9), which selects the XMI node that is to receive the interrupt. For an interrupt command, the node specifier field is an interrupt destination mask. The field selects the destination of the interrupt command by masking out all nodes except the interrupt command destination.

In the example of part A of Figure 3–8, the mask is selecting node 8 (the XJA) to receive the interrupt.[1]

The interrupting node receives an acknowledge cycle from the XJA two cycles after the command cycle.

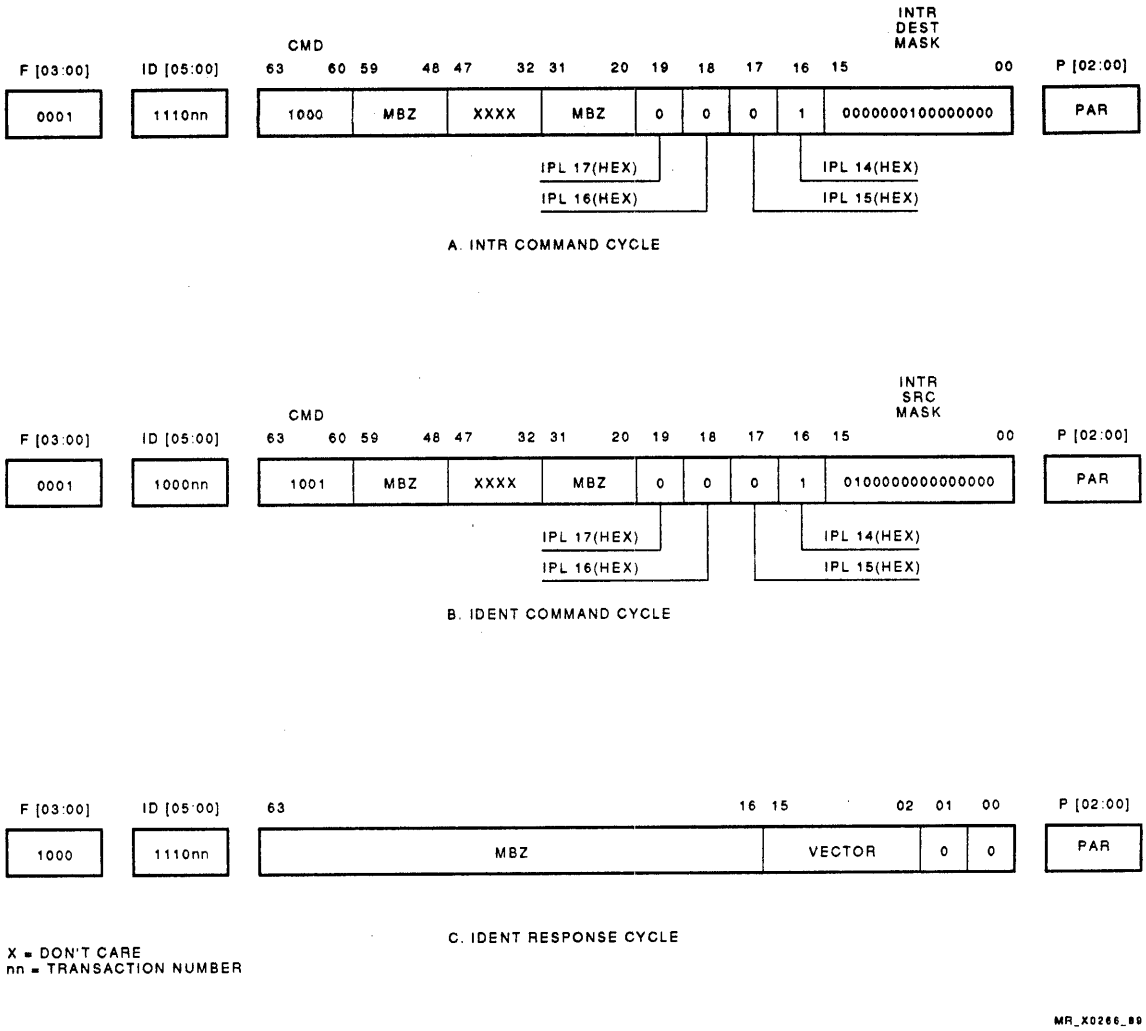


Figure 3–8 Basic Interrupt Transactions

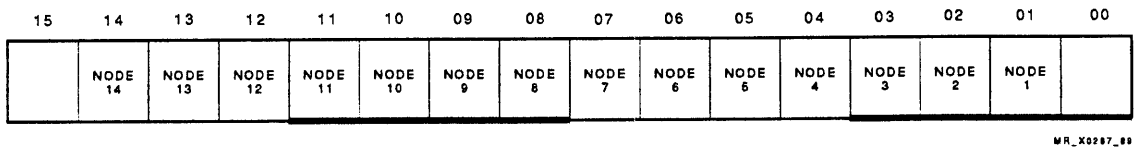


Figure 3–9 Interrupt Node Specifier Field

[1] All XMI interrupts are transmitted to the XJA.

3.6.1.2 IDENT Transaction

The XJA relays the interrupt to the CPU, which responds by requesting the XJA for an interrupt vector. This causes the XJA to initiate an IDENT transaction to the interrupting node, requesting the interrupt vector.

The XJA arbitrates for the bus with a command request (XMI_CMD_REQ). Upon receiving XMI_GRANT, it places function, ID, data, and parity bits on the XMI bus (part B of Figure 3–8).

The function bits specify a command cycle. The ID bits identify the transmitting node, which is the XJA (node 8). The command field of the data longword specifies an IDENT command (Table 3–6). Bits [19:16] of the data longword are the IPL field that specifies the interrupt priority level of the interrupt request (level 14).

Bits [15:00] of the data word (the interrupt node specifier field) are the interrupt source mask that identifies the node that issued the interrupt (node 14).

The interrupting node, upon receiving the IDENT command, returns the requested vector to the XJA in an IDENT response transfer. The IDENT response transfer is an XMI read-data-return function specifying good read data.

The interrupting device at node 14, arbitrates for the bus with a response request (XMI_RES_REQ). Upon receiving XMI_GRANT, it places function, ID, data, and parity bits on the XMI bus (part C of Figure 3–8).

The function bits specify a good read-data-return function (Table 3–2). The ID bits identify the transmitting node (node 14). The requested data (the vector) is contained in bits [15:02] of the data word. Bits [01:00] are 0 as the vectors are longword aligned.

3.6.2 IVINTR Interrupts

The implied vector interrupt (IVINTR) transaction is a single-cycle interrupt transaction. The interrupt priority and the value of the interrupt vector are implied by bits in the interrupt type field of the command data longword. Figure 3–10 shows the command cycle for the IVINTR transaction.

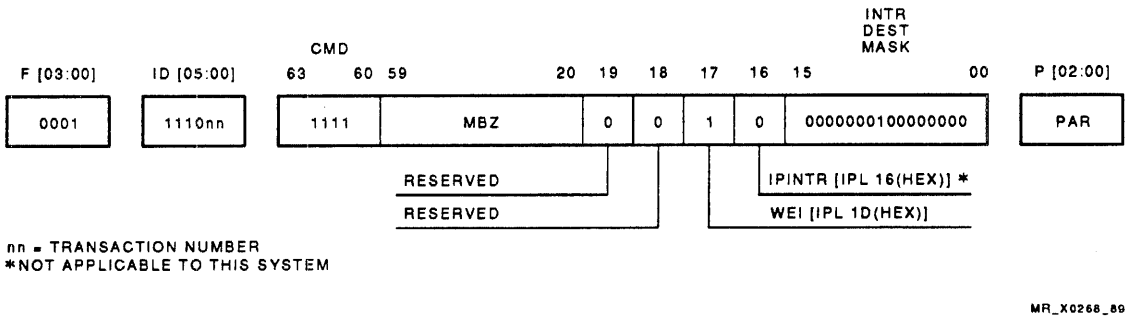


Figure 3–10 Implied Vector Interrupt Transaction (IVINTR)

The function bits specify a command function. The ID bits identify the transmitting (interrupting) node as node 14. The command field of the data longword specifies a vector interrupt command (Table 3–6). Bits [19:16] of the data longword specify the type of interrupt.

Only two types of interrupts are specified in the type field: an interprocessor interrupt (IPINTR) or a write error interrupt (WEI). An IPINTR interrupts the system CPU at an IPL of 16(hex). The XJA supplies the required vector [80(hex)] to the CPU.

NOTE
The VAX 9000 system does not have other processors on the XMI bus. Therefore, interprocessor interrupts are not applicable, even though the XJA supports this type of interrupt.

A WEI is a fatal error and the XJA notifies the system of the fatal error by asserting XJA_FATALERR on the JXDI. The system interrupts the CPU at an IPL of 1D(hex) and then supplies it a vector of 60(hex). See Chapter 4 for a more detailed description of system interrupts.

Bits [15:00] (the interrupt node specifier field) are the interrupt destination mask that selects the XMI node that is to process the interrupt. The destination mask in Figure 3–10 is selecting node 8 (the XJA).

Since the value of the interrupt vector is indicated by the type of interrupt, an IVINTR transaction does not require an associated IDENT transaction.

4
XJA

This chapter provides a detailed description of the XJA adapter. It describes what the functional areas of the XJA do during various transactions. Included are discussions of data path, data formats, and data timing throughout the XJA. Also described is the XJA self-test logic and the three clock systems used by the XJA.

4.1 Overview

Figure 4–1 is a block diagram of the XJA. The mnemonics used for the chips and functional logic areas are defined in Table 1–4.

The XDE chips function to transfer data packets between the JXDI and the XDC under control of the XCE chip. The XMI corner functions to transfer data packets between the XMI bus and the XDC through a standard XMI interface.

The XDC chip is divided into functional areas similar to those shown in the simplified block diagram of Figure 1–4. The XRC, RRF, and RCM work together to receive data packets from the XMI corner, process them, and when applicable, supply the packets to the CBI. These areas are described in Section 4.6, XDC Receive Logic. The TRF and TCM work together to accept data packets from the XDEs, process them, and when applicable, transmit the packets to the XMI corner. These areas are described in Section 4.7, XDC Transmit Logic. The REG interfaces with both the transmit and receive logic.

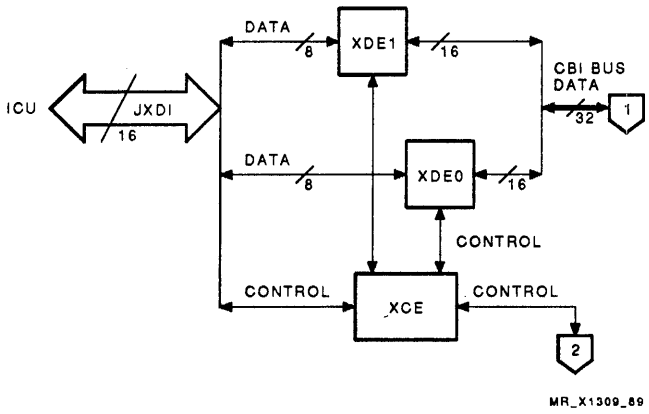


Figure 4–1 (Cont.) XJA Block Diagram

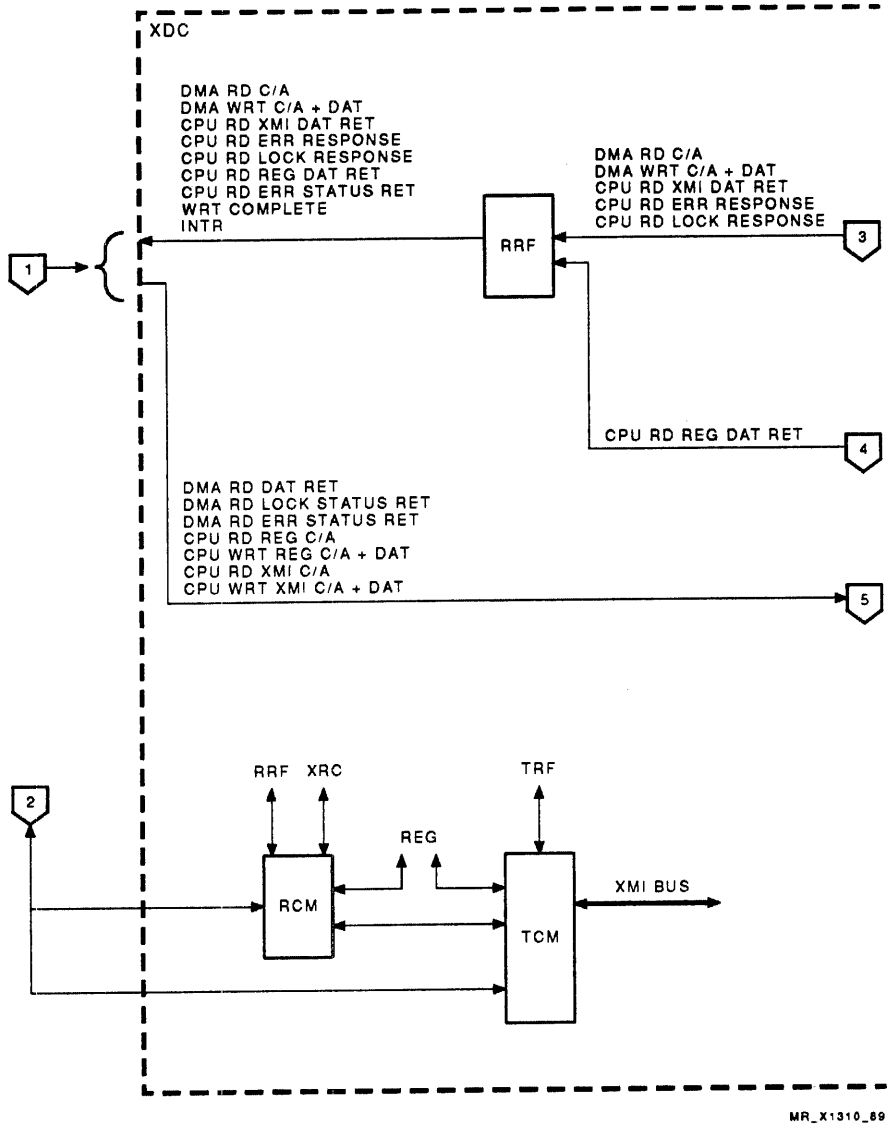


Figure 4–1 (Cont.) XJA Block Diagram

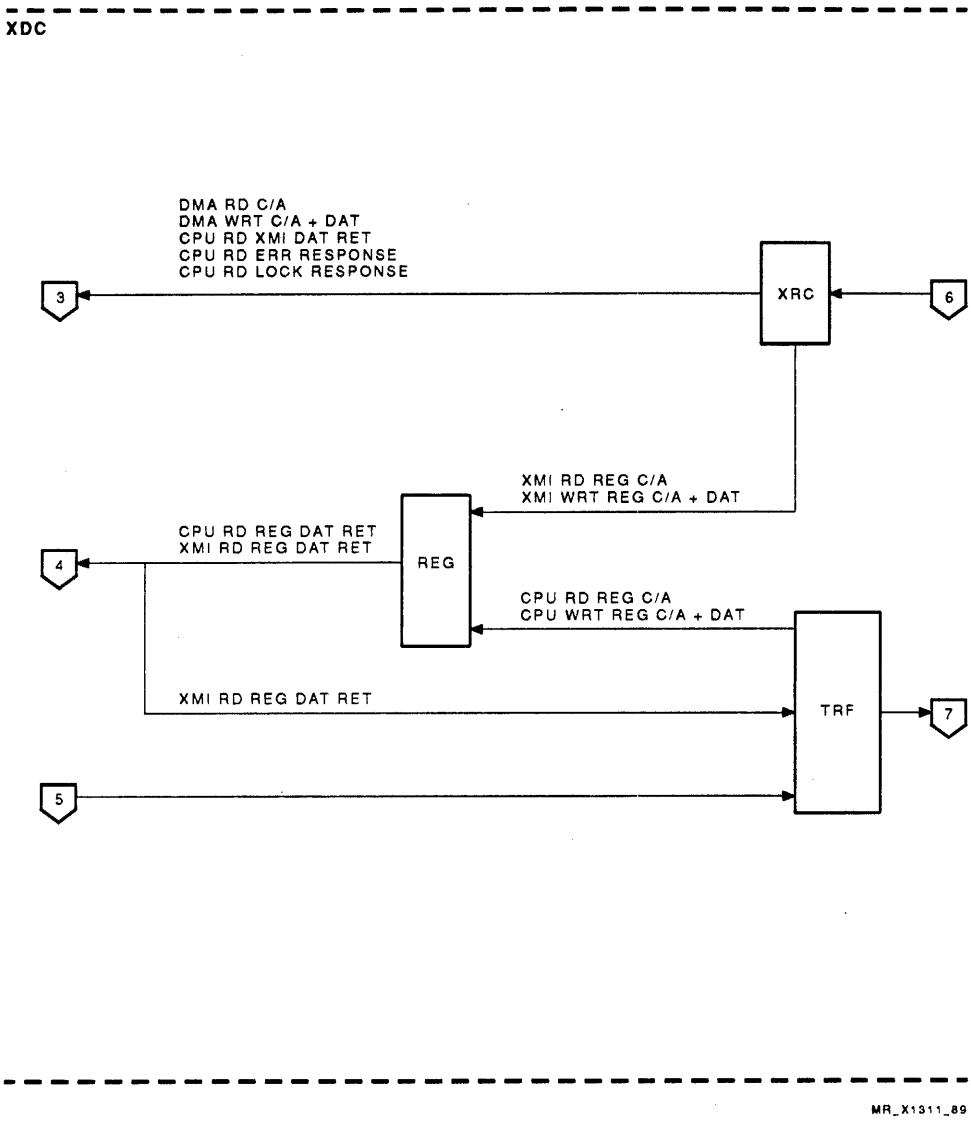


Figure 4–1 (Cont.) XJA Block Diagram

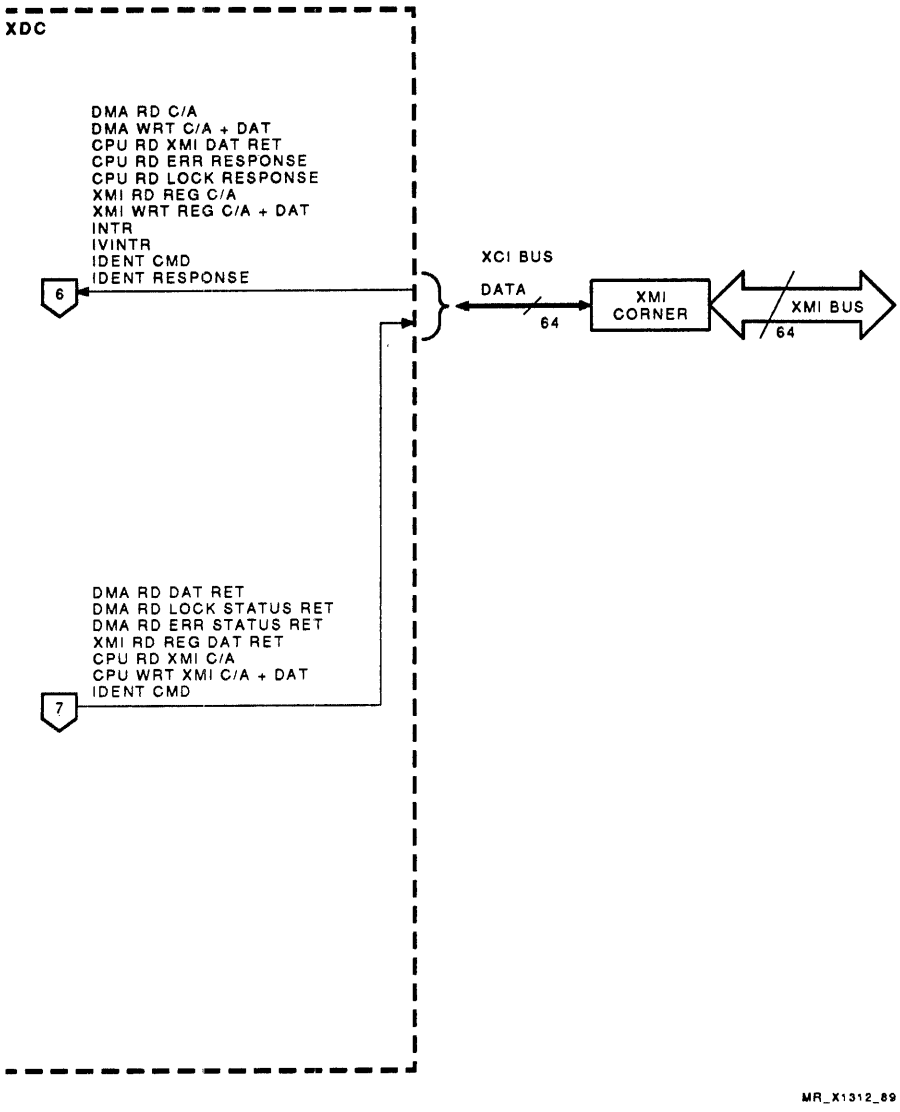


Figure 4–1 XJA Block Diagram

Thirteen specific types of transactions execute through the XJA:

- DMA read
- DMA write
- CPU read of XMI registers excluding XMI space registers in XJA
- CPU write of XMI registers excluding XMI space registers in XJA
- CPU read of XMI space registers in XJA
- CPU write of XMI space registers in XJA
- CPU read of XJA private registers
- CPU write of XJA private registers
- XMI normal interrupts
- XMI implied vector interrupts — interprocessor interrupts
- XMI implied vector interrupts — write error interrupts
- XJA generated nonfatal interrupts
- XJA generated fatal interrupts

Each of these transactions is summarized in a brief step-by-step description in the following paragraphs. The XDC transmit and receive logic process the various packet types differently. These differences are described in the XDC receive and transmit sections of this chapter. These descriptions provide the details of packet processing but lack the overall continuity provided by the following transaction summaries. The summary can be used to follow a transaction from start to finish while the XDC receive and transmit sections can provide the details of the transaction steps when desired.

Figure 4–1 lists the various types of transaction packets that are input to and output from the XDC functional areas.

4.1.1 DMA Read

This transaction occurs as follows:

1. XRC receives the DMA read C/A from the XMI bus. XRC notifies RCM.
2. RCM loads the DMA read C/A into the RRF buffer. RCM notifies XCE.
3. XCE unloads the DMA C/A from the RRF buffer and transfers it through the XDEs to the JXDI.
4. XCE transfers the DMA read data return from the JXDI, through the XDEs, and loads the data into the TRF buffer. XCE notifies TCM.
5. TCM unloads the DMA read data return from the TRF buffer and transmits it to the XMI bus.

4.1.2 DMA Write

This transaction occurs as follows:

1. XRC receives the DMA write C/A and data from the XMI bus. XRC notifies RCM.
2. RCM loads the DMA write C/A and data into the RRF buffer. RCM notifies XCE.
3. XCE unloads the DMA C/A and data from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.3 CPU Read of XMI Registers Excluding XMI Space Registers in XJA

This transaction occurs as follows:

1. XCE transfers the CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads the CPU read C/A from the TRF buffer and transmits it to the XMI bus.
3. XRC receives the CPU read data return from the XMI bus. XRC notifies RCM.
4. RCM loads the CPU read data return into the RRF buffer. RCM notifies XCE.
5. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.4 CPU Write of XMI Registers Excluding XMI Space Registers in XJA

This transaction occurs as follows:

1. XCE transfers the CPU write C/A and data from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads the CPU write C/A and data from the TRF buffer and transmits it to the XMI bus. TCM notifies RCM.
3. RCM forces the generation of a write complete packet in the RRF. RCM notifies XCE.
4. XCE unloads the write complete packet from the RRF and transfers it through the the XDEs to the JXDI.

4.1.5 CPU Read of XMI Space Registers in XJA

This transaction occurs as follows:

1. XCE transfers CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads CPU read C/A from the TRF buffer and transmits it to the XMI bus.
3. XRC receives the CPU read C/A from the XMI bus. XRC notifies RCM.

4. RCM notifies TCM.
5. TCM unloads the XMI space register to TRF.
6. TCM transmits the XMI space register data from the TRF to the XMI bus.
7. XRC receives the CPU read data return from the XMI bus. XRC notifies RCM.
8. RCM loads the CPU read data return into the RRF buffer.
9. RCM forces the generation of a read data return packet in the RRF. RCM notifies XCE.
10. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.6 CPU Write of XMI Space Registers in XJA

This transaction occurs as follows:

1. XCE transfers the CPU write C/A and data from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads the CPU write C/A and data from the TRF buffer and transmits it to the XMI bus.
3. XRC receives the CPU write C/A and data from the XMI bus.
4. XRC notifies REG and RCM.
5. REG loads the XMI space register with write data.
6. RCM forces the generation of a write complete packet in the RRF. RCM notifies XCE.
7. XCE unloads the write complete packet from the RRF and transfers it through the XDEs to the JXDI.

4.1.7 CPU Read of XJA Private Registers

This transaction occurs as follows:

1. XCE transfers the CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads the XJA private register to the RRF. TCM notifies RCM.
3. RCM forces the generation of a read data return packet in the RRF. RCM notifies XCE.
4. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.8 CPU Write of XJA Private Registers

This transaction occurs as follows:

1. XCE transfers the CPU write C/A and data from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
2. TCM unloads the TRF buffer, transfers the write data from the TRF to REG, and loads the data into the XJA private register. TCM notifies RCM.
3. RCM forces the generation of a write complete packet in the RRF. RCM notifies XCE.
4. XCE unloads the write complete packet from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.9 XMI Bus Initiated Normal Interrupts

This transaction occurs as follows:

1. XRC receives normal interrupt from the XMI bus at an IPL of 14, 15, 16, or 17. XRC notifies RCM of the interrupt.
2. RCM forces an interrupt packet in the RRF at the specified IPL. RCM notifies XCE.
3. RCM notifies TCM that an interrupt is pending and a CPU read XJA private register is coming. TCM posts a pending interrupt at the specified IPL.
4. XCE unloads the interrupt packet from the RRF and transfers it through the XDEs to the JXDI.
5. CPU initiates a read of the specified IDENT register.
6. XCE transfers the CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
7. TCM assembles an IDENT command packet in the TRF and transmits it to the XMI bus.
8. XRC receives the IDENT response data return from the XMI bus. XRC notifies RCM.
9. RCM loads the IDENT response into the RRF buffer.
10. RCM forces the generation of a read data return packet in the RRF. RCM notifies TCM and XCE.
11. TCM clears the posted interrupt at the specified IPL.
12. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.10 XMI Implied Vector Interrupts — Interprocessor Interrupts

NOTE
This type of interrupt does not currently occur in the VAX 9000 system as there are no CPUs on the XMI bus.

This transaction occurs as follows:

1. XRC receives IPINTR from the XMI bus. XRC notifies RCM.
2. RCM notifies TCM. TCM posts a pending IPL 16 interrupt.

3. RCM forces the generation of an IPL 16 interrupt packet in the RRF. RCM notifies XCE.
4. XCE unloads the interrupt packet from the RRF and transfers it through the XDEs to the JXDI.
5. CPU initiates a read of the IPL 16 SCB offset register (IDENT6).
6. XCE transfers the CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM. TCM notifies RCM.
7. RCM forces the generation of a read data return packet in the RRF with an SCB offset vector of 80(hex). RCM notifies TCM and XCE.
8. TCM clears the posted IPL 16 interrupt.
9. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.11 XMI Implied Vector Interrupts — Write Error Interrupts

This transaction occurs as follows:

1. XRC receives WEI from the XMI bus. XRC notifies REG.
2. REG asserts FATALERR to XCE.
3. XCE asserts XJA_FATALERR to the JXDI.

4.1.12 XJA Generated Nonfatal Interrupts

This transaction occurs as follows:

1. REG detects a nonfatal interrupt error bit set.
2. REG notifies TCM and RCM.
3. RCM posts IPL 17 interrupt pending. TCM looks for the next IDENT7 transaction.
4. RCM forces the generation of an IPL 17 interrupt packet. RCM notifies XCE.
5. XCE unloads the interrupt packet from the RRF and transfers it through the XDEs to the JXDI.
6. CPU initiates a read of the IPL 17 SCB offset register (IDENT7).
7. XCE transfers the CPU read C/A from the JXDI, through the XDEs, and loads it into the TRF buffer. XCE notifies TCM.
8. TCM unloads the error SCB offset register into the RRF buffer.
9. RCM clears the posted IPL 17 interrupt. TCM notifies RCM.
10. RCM forces the generation of a read data return packet in RRF. RCM notifies XCE.
11. XCE unloads the CPU read data return from the RRF buffer and transfers it through the XDEs to the JXDI.

4.1.13 XJA Generated Fatal Interrupts

This transaction occurs as follows:

1. REG detects a fatal interrupt error bit set.
2. REG asserts FATALERR to XCE.
3. XCE asserts XJA_FATALERR to the JXDI.

4.2 XJA Clock System

The XJA uses three separate, asynchronous, clock systems. The three systems are as follows:

- An XCI_C clock system to send and receive data from the XMI bus
- A CLKJ clock system to receive data from the ICU
- A CLKX clock system to send data to the ICU

Table 4–1 lists the signals, with their times, for each of the three clock systems.

Table 4–1 XJA Clocks

Clock System	Signal	Period (ns)
XCI_C	XCI_C12	64
	XCI_C23	64
	XCI_C34	64
	XCI_C45	64
	XCI_C56	64
	XCI_C61	64
	MCLK	32
CLKJ	ICU_CLKJ	16
	SCLKJ	32
CLKX	CLKX	16
	SCLKX	32
	UNLOAD_CLK	32

4.2.1 Crossing Asynchronous Boundaries Between Clock Systems

Each clock system drives a specific area of synchronous logic. When a given area of logic needs to communicate with another area of logic running on a different clock system, synchronizing logic must be used. The basic scheme that the XJA uses to communicate across asynchronous boundaries is shown in Figure 4–2 and described in the following three steps:

1. Logic A and logic B run on different clocks.
2. The data buffer is unidirectional and is loaded by CLKA (logic A's clock) and unloaded by CLKB (logic B's clock).
3. Control lines from logic A inform logic B that the data is completely loaded into the data buffer and can now be unloaded. Data is never loaded and unloaded at the same time.

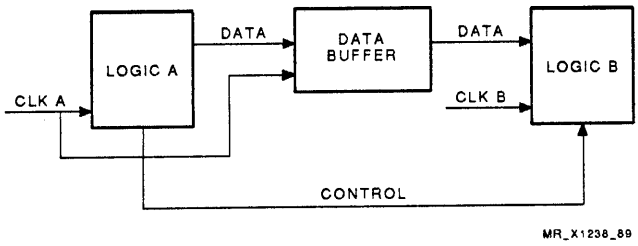


Figure 4–2 Data Communication Across Asynchronous Boundaries

4.2.2 XCI_C Clock System

Figure 4–3 is a block diagram of the XCI_C clock system.
The logic that interfaces to the XMI bus runs off the XCI_C clock system. There are six XCI_C clocks all with a time period of 64 ns. The XCI_C clocks are obtained from the XCLOCK chip in the XMI corner. The XCLOCK generates the XCI_C clocks from the XMI_TIME and XMI_PHASE clocks obtained from the XMI bus. Section 4.5.2 describes the function of XCLOCK and Figure 4–16 illustrates the six XCI clocks.

The XCI_C clock system:

- Clocks the receive data from the XMI bus, through the XRC, and into the RRF
- Supplies 64-ns clocks to the RCM to synchronize that portion of the RCM that controls the reception of the XMI data in the XRC and RRF
- Clocks the transmit data out of the TRF to the XMI bus
- Supplies 64-ns clocks to the TCM to synchronize that portion of the TCM that controls the transmission of data from the TRF to the XMI bus
- Clocks data into and out of the XJA registers

All the receive and transmit data synchronized by the 64-ns XCI_C clocks are in 64-bit format except register data, which is longword in length.

Two XCI_C clocks (XCI_C34 and XCI_C61) are ORed to generate an asymmetrical 32-ns clock called MCLK. MCLK is supplied to the XCE to clock in control signals from the RCM and the TCM that are synchronized to the XCI_C clock system.

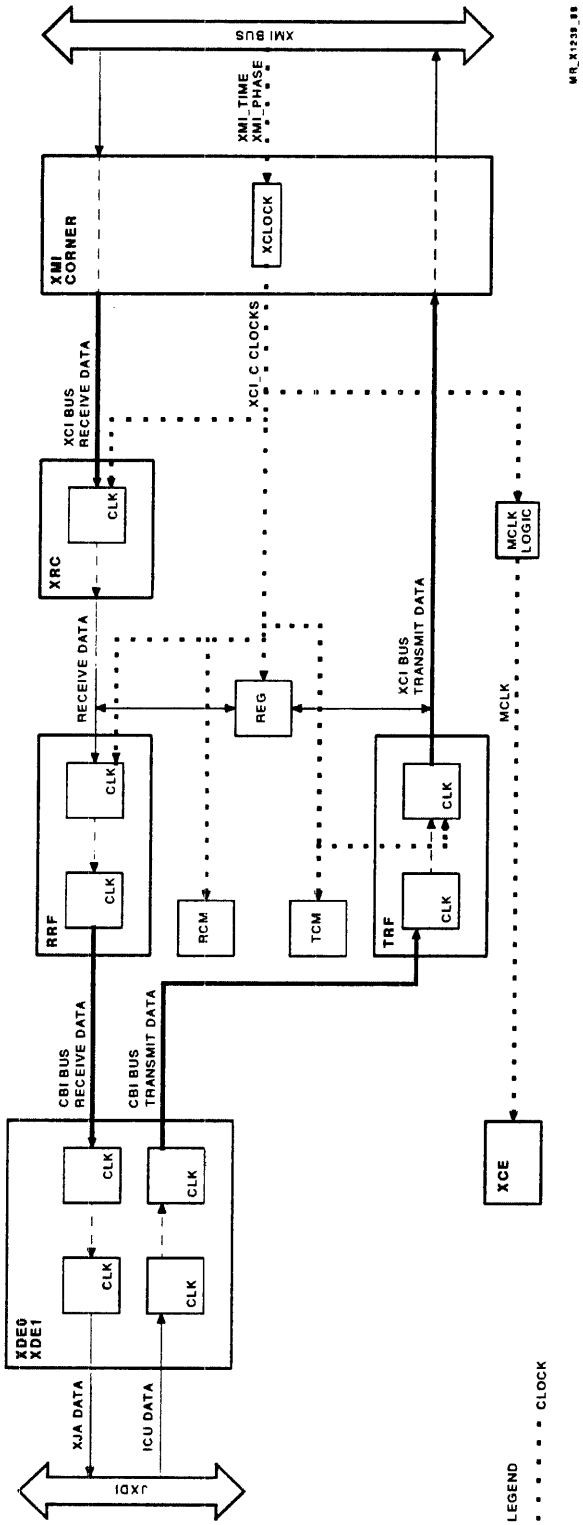


Figure 4–3 XCI_C Clock System

4.2.3 CLKJ Clock System

The logic that receives data from the ICU, by way of the JXDI, is clocked by the CLKJ clock system (Figure 4–4). ICU_CLKJ is received from the ICU as a 16-ns (nominal system cycle time) square wave. ICU_CLKJ is derived in the ICU from the system CLOCK_B. The XJA uses the 16-ns ICU_CLKJ to clock data from the JXDI into the XDE0 and XDE1 chips. ICU_CLKJ also synchronizes the portion of the XCE logic that controls the data flow into the XDE chips and the portion that processes the ICU control signals associated with the data flow from the ICU. The JXDI data that is transferred into the XDE chips is in 16-bit word format.

ICU_CLKJ is divided by two in the XCE to form a 32-ns clock called SCLKJ (slow CLKJ). SCLKJ is used to:

- Clock the transmit data out of the XDE0 and XDE1 chips onto the CBI bus
- Clock the transmit data off the CBI bus into the TRF
- Synchronize that portion of the TCM that controls the transfer of the transmit data from the CBI bus to the TRF
- Synchronize that portion of the XCE logic that controls the data flow from the XDE chips to the TRF.

4.2.4 CLKX Clock System

The logic that transmits data to the ICU, by way of the JXDI, is clocked by the CLKX clock system (Figure 4–5). CLKX is derived from a B phase clock (CLKB) obtained from the master clock module (MCM). It is used to clock data from the XDE0 and XDE1 chips to the ICU by way of the JXDI. CLKX synchronizes the portion of the XCE logic that controls the data flow out of the XDE chips and the portion that processes the ICU control signals associated with data flow to the ICU. The data that is transferred from the XDE chips to the ICU is in 16-bit word format.

The 16-ns CLKB is divided by two to produce two 32-ns waveforms (SCLKX and UNLOAD_CLK). UNLOAD_CLK is used to clock the 32-bit receive data from the RRF to the XDE chips over the CBI bus, and to synchronize the XCE logic that controls unloading of the receive data onto the CBI bus from the RRF. SCLKX is used to load the 32-bit data into the XDE chips, and to synchronize the XCE logic that controls the loading of the data into the XDE chips.

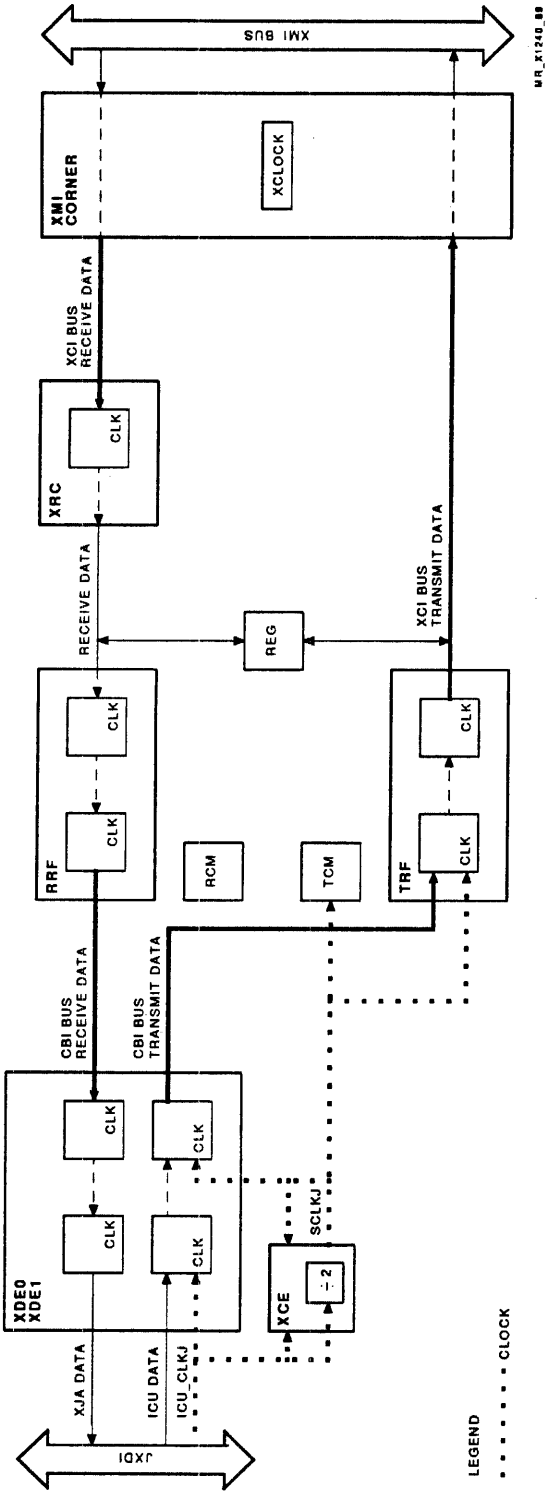


Figure 4–4 CLKJ Clock System

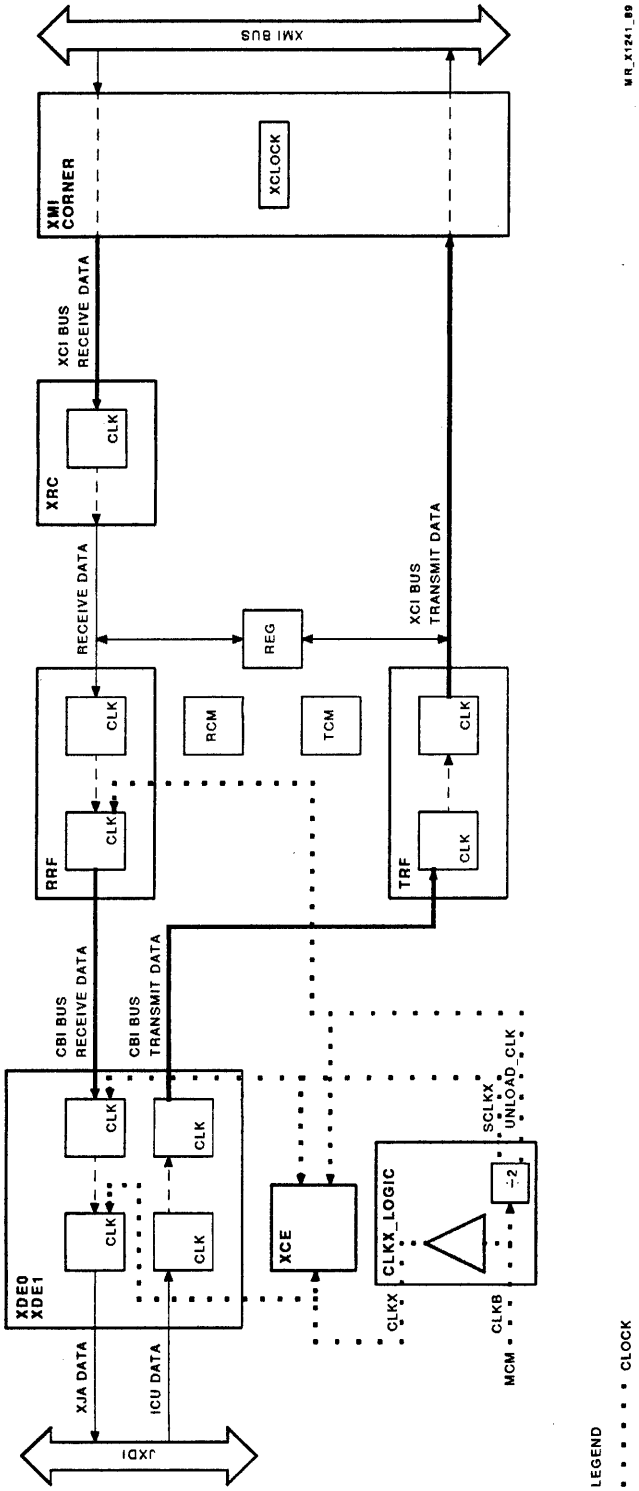


Figure 4–5 CLKX Clock System

4.3 XDE Data Flow

NOTE
Transmit and receive as described in the XDE data flow, is with respect to the JXDI as opposed to the rest of the XJA data flow that is with respect to the XMI bus.

4.3.1 Transmit Data Flow

Data to be transmitted to the JXDI, along with four bits of byte parity, is received from the CBI in longword format at a clock rate of 32 ns. The data is split into two channels (one channel per XDE chip) with a data word and two parity bits going to each channel. The two channels are identical, therefore only one channel is described. Figure 4–6 is a block diagram of one channel of the XDE transmit data path.

Data words are enabled from the CBI into the XDE channel by the asserted state of DIR from the XCE receive logic. The data word is split into two bytes with each byte and its associated parity bit clocked into a transmit buffer (TRANSMIT BUFFER 1) by the 32-ns SCLKX clock. Sixteen ns later, byte 1 is clocked into transmit buffer 2 by the opposite phase of SCLKX. A byte multiplexer is switched at a 16-ns rate to select alternately byte 0 and byte 1. The multiplexer output is clocked into a transmit register at a 16-ns rate by CLKX. The register output is routed to the JXDI. The data byte and parity bit from each channel are combined to form a 16-bit data word and two parity bits on the JXDI (Figure 4–1). The JXDI bus cycles are 16-ns.

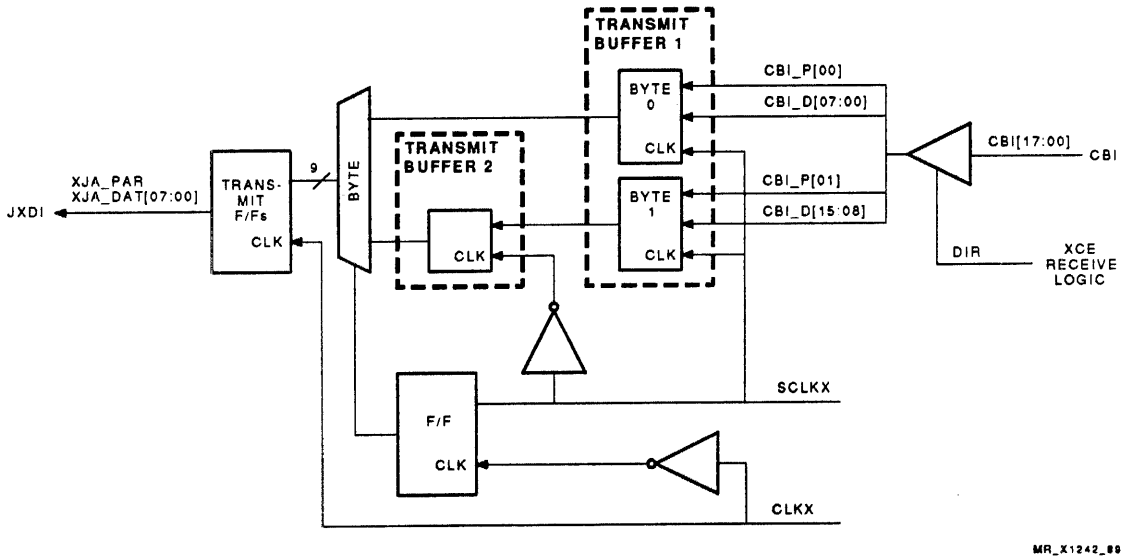


Figure 4–6 XDE Transmit Data Path

Figure 4–7 shows the timing of the transmit data. At T0, byte 0 and byte 1 are clocked into buffer 1 by SCLKX. Byte 1 is then clocked into transmit buffer 2 at time T2, by the opposite phase of SCLKX. The byte 0 output of buffer 1 and the byte 1 output of buffer 2 are applied to a byte multiplexer which functions to alternately select the two inputs at a 16-ns rate. The multiplexer switching is accomplished by a flip-flop clocked by the negative phase of the 16-ns CLKX with the 32-ns SCLKX being the D input. The multiplexer switching is shown in the timing diagram where it alternately selects between byte 0 and byte 1 at a 16-ns rate. The diagram shows data to the JXDI bus as it is clocked into the transmit register and output from the XDE transmit data path as XJA_PAR and XJA_DAT[07:00].

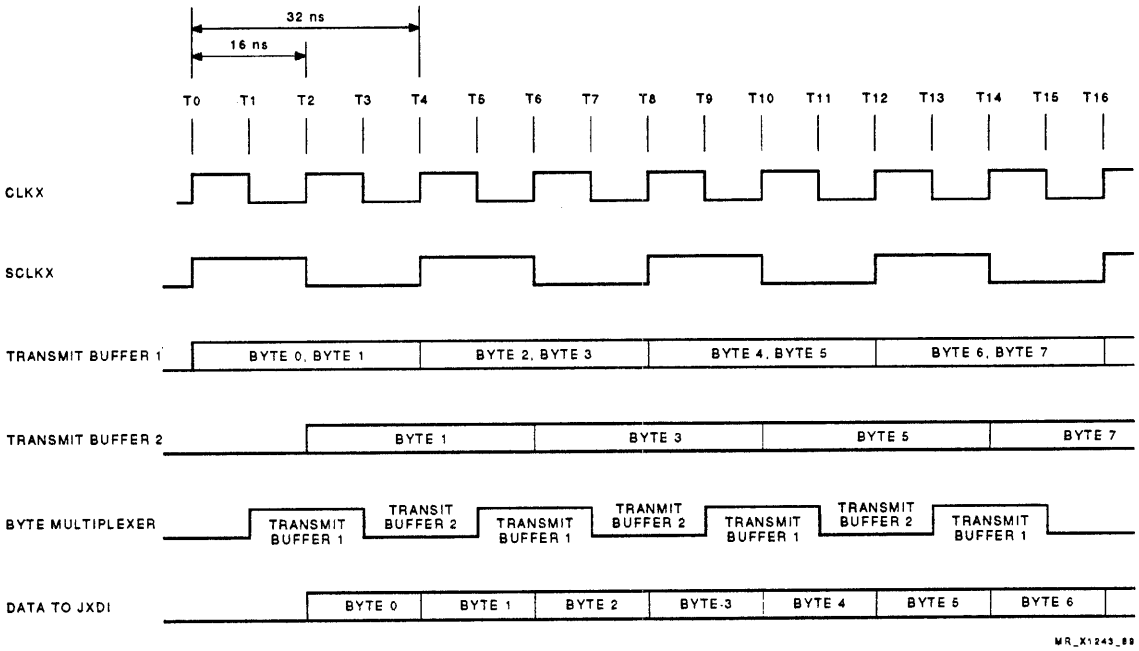


Figure 4–7 XDE Transmit Timing Diagram

4.3.2 Receive Data Flow

Data received from the JXDI is in word format and at a clock rate of 16-ns. The data is split into two channels (one channel per XDE chip) with a data byte and one parity bit going to each channel. The channels function to assemble the bytes into data words, which are then combined into longwords for the CBI bus. The two channels are identical (except for decoding the length bits), therefore only one channel is described. Figure 4–8 is a block diagram of one channel of the XDE receive data path.

The data byte and its associated parity bit is clocked into receive flip-flops by the 16-ns CLKJ clock from the JXDI. The output of the receive flip-flops is applied to a parity checker, which asserts PERR to the XCE receive logic if a parity error is detected on any of the data cycles.

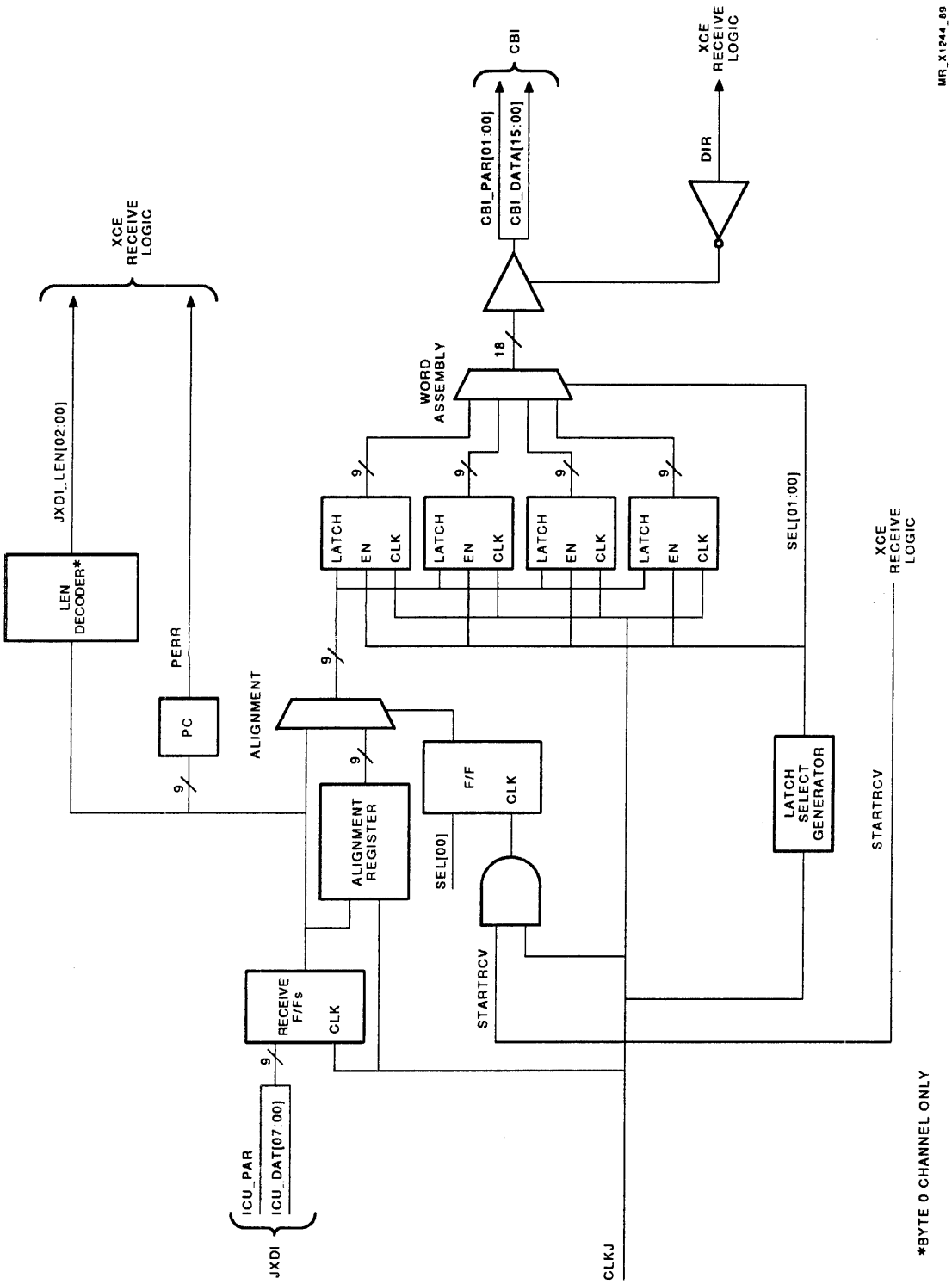


Figure 4–8 XDE Receive Data Path

The flip-flops in the byte 0 channel apply the length bits of the first data word (command/address word) to a length decoder. The decoder outputs JXDI_LEN[02:00] to the XCE receive logic, specifying the length of the data packet that is being received.

The output of the receive flip-flops is also applied to an alignment multiplexer and an alignment register. Sixteen ns later the data is clocked into the alignment register whose output is also applied to the alignment multiplexer. The multiplexer selects the direct data or the delayed data (using the alignment register) depending on the alignment of the input data.

If no alignment is required, the alignment multiplexer selects the data bytes from the receive flip-flops and applies them to four latches clocked by the CLKJ clock (Figure 4–9). The latches are enabled by SEL[01:00] from a latch select generator. The generator functions to produce two square waves (SEL[01:00]) as shown in Figure 4–9. The four bit states of SEL[01:00] select the four latches in order. The latches are loaded with bytes 0, 1, 2, and 3 and then cycled through again with the next four bytes. While latch 2 and 3 are being loaded, latch 0 and 1 are unloaded through a multiplexer and then onto the CBI. The multiplexer unloads latches 0 and 1 when SEL[01:00] is 11 (T4 through T6) and unloads latches 2 and 3 when SEL[01:00] is 01 (T8 through T10).

The data words are enabled onto the CBI by the negated state of DIR from the XCE receive logic. DIR is negated by the XCE when a data packet is being received from the ICU. The data words output from each channel are combined on the CBI to form data longwords (Figure 4–1).

4.3.3 Alignment of Received Data

Data alignment is shown in Figure 4–10. Word data from the JXDI is synced to the 16-ns CLKJ clock. The longword data on the CBI is synced to the 32-ns SCLKJ clock. Part A of Figure 4–10 shows the aligned data case where word 0 and word 1 are clocked in at T4 and T6, respectively. The next cycle of SCLKJ, clocks longword 0 (composed of word 0 and word 1) at time T8. Word 2 and word 3 make up longword 1 clocked at T12, and so on. In the aligned case, the first data word is clocked on an even cycle of CLKJ (with respect to SCLKJ).

The alignment multiplexer is switched by the ANDing of SEL0 and STARTRCV in a flip-flop. STARTRCV is received from the XCE receive logic where it is derived from ICU_CMDAVAIL. Therefore, STARTRCV precedes the received data by one cycle. SEL0 is a 32-ns clock used to assemble the data longwords for the CBI bus. In the aligned case (part B of Figure 4–10), STARTRCV asserts at T2 and data word 0 is clocked on the next data cycle (T4), resulting in an aligned state. SEL0 and STARTRCV do not AND, the multiplexer is not switched, and the direct data is used.

In the unaligned case, the first data word is clocked in on an odd cycle of CLKJ causing the situation shown in part C of Figure 4–10. Here longword 0 contains only the first word of received data (word 0) while longword 1 contains word 1 and word 2. The word data is unaligned with respect to the longword data. The alignment multiplexer corrects the misalignment by selecting the delayed data (delayed one 16-ns cycle), therefore placing the word data in alignment with the longword data. The alignment multiplexer is switched as seen in part D of Figure 4–10. STARTRCV asserts at T0 and data word 0 is clocked in at T2 resulting in an unaligned state. SEL0 and STARTRCV do AND, the multiplexer is switched, and the delayed data from the alignment register is used.

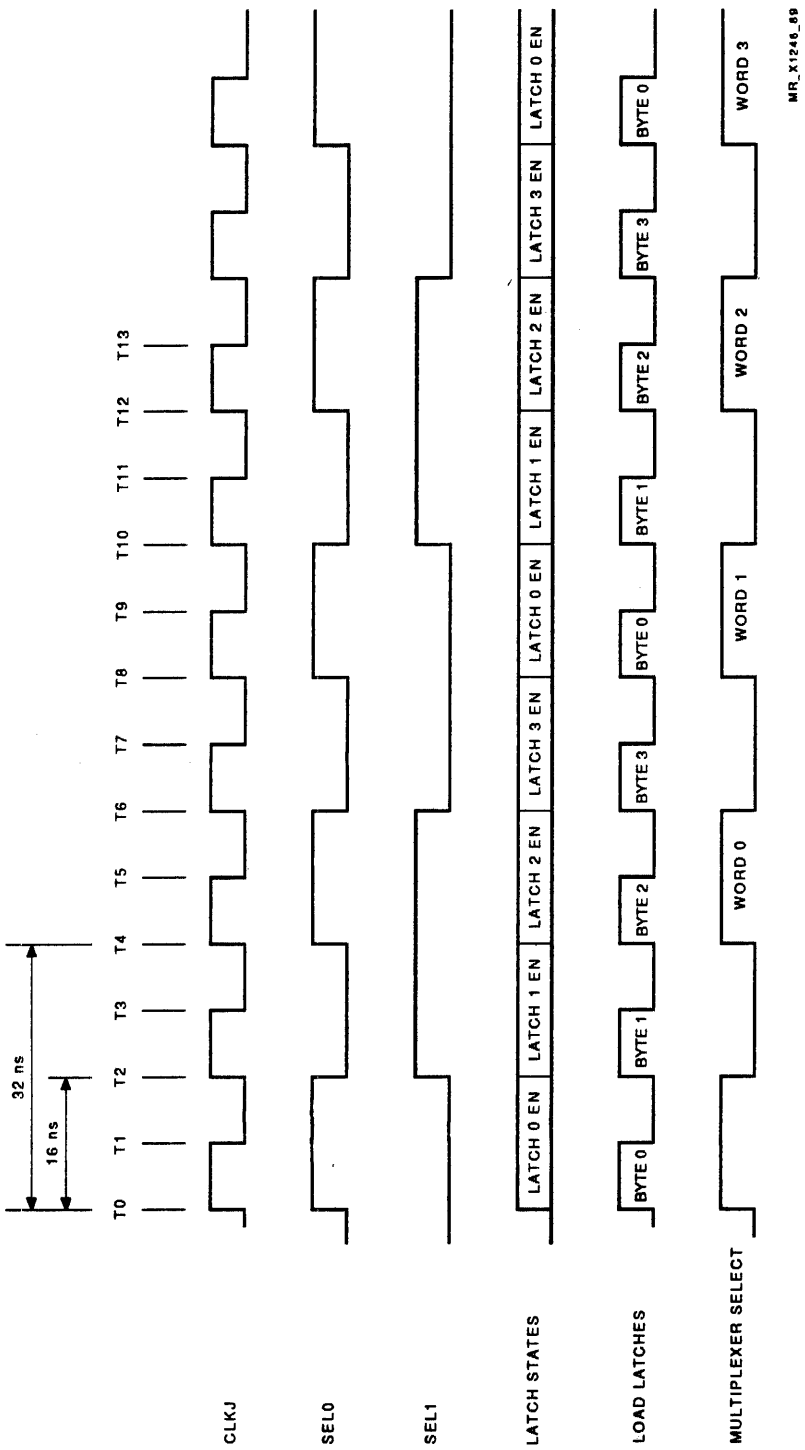


Figure 4–9 Assembly of Data Longwords

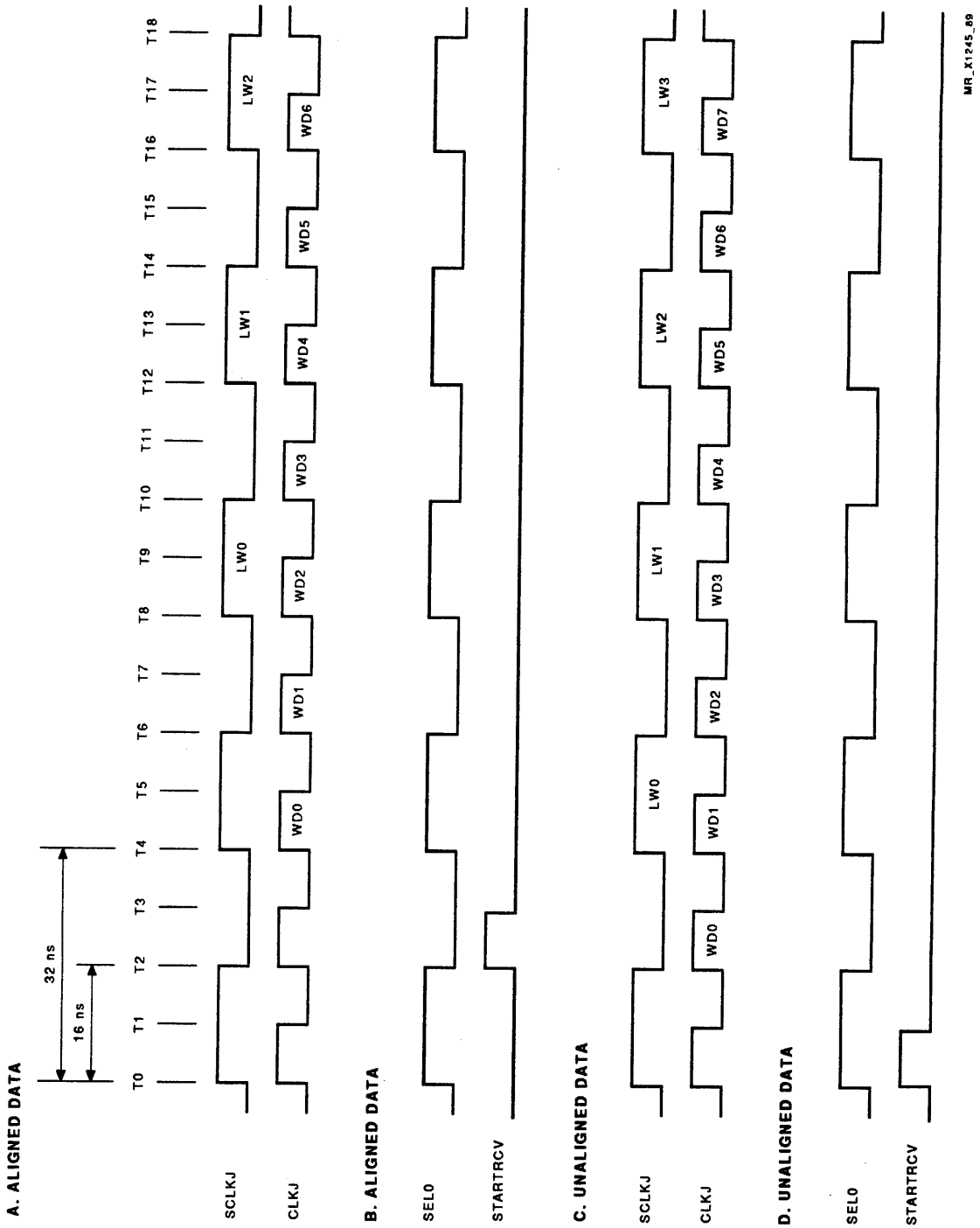


Figure 4–10 Alignment of Received Data

4.4 XCE Data Flow Control Logic

NOTE
Transmit and receive as described in the XDE data flow, is with respect to the JXDI as opposed to the rest of the XJA data flow that is with respect to the XMI bus.

4.4.1 Transmit Logic

The XCE transmit logic controls the transmission of data from the RRF, through the XDEs to the JXDI. Figure 4–11 is a block diagram of the XCE transmit logic. A transmit state machine (TSM) controls the transmission sequence and allows up to two transmissions to be outstanding at the same time. That is, a data packet can be transmitted before an ICU acknowledgment or retry command is received from the preceding transmission. Three identical transmit entry machines (TEMs) process transmit commands received from the RCM. A TEM is selected (in sequential order) to initiate a transmission. No TEM has priority over another. The block diagram shows only one TEM (TEM1).

Figure 4–12 is a flow diagram of an XCE transmit sequence. The sequence is the same regardless of which TEM initiates the transmission. TEM1 is used in the flow diagram to correspond with the block diagram.

A transmit sequence is initiated by the RCM when there is data in one of the RRF buffers. The RCM initiates the sequence by asserting STARTXMIT, SBUFNUM[02:00], and LENGTH[01:00] to the XCE.

SBUFNUM[02:00] (send buffer number) identifies which of the RRF buffers has the data packet that is to be transmitted. The SBUFNUM[02:00] code is given in Table 4–2.

LENGTH[01:00] specifies the length of the data packet as shown in Table 4–3.

Table 4–2 RRF Buffer Select Code

SBUFNUM[02:00]	Buffer Selected
0 0 0	DMA read/write buffer 0
0 0 1	DMA read/write buffer 1
0 1 0	DMA read/write buffer 2
0 1 1	DMA read/write buffer 3
1 0 0	CPU read and force command buffer
1 0 1	DMA read/write buffer 4

Table 4–3 Data Packet Length Code

LENGTH[01:00]	Length (CBI Cycles)	Function
0 0	3	CPU read return, status, or DMA read C/A
0 1	4	DMA write, quadword
1 0	6	DMA write, octaword
1 1	10	DMA write, hexword

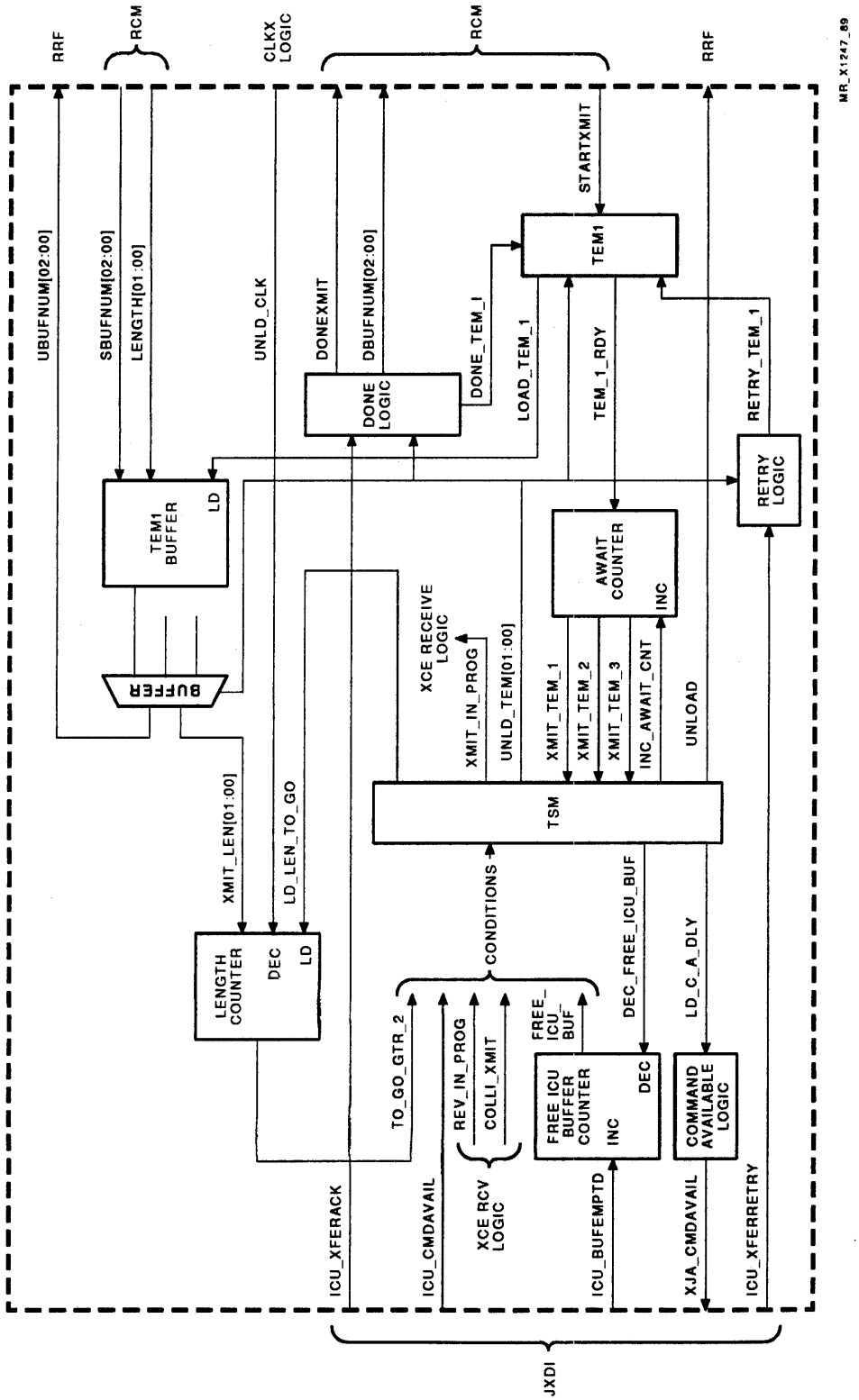


Figure 4–11 XCE Transmit Logic

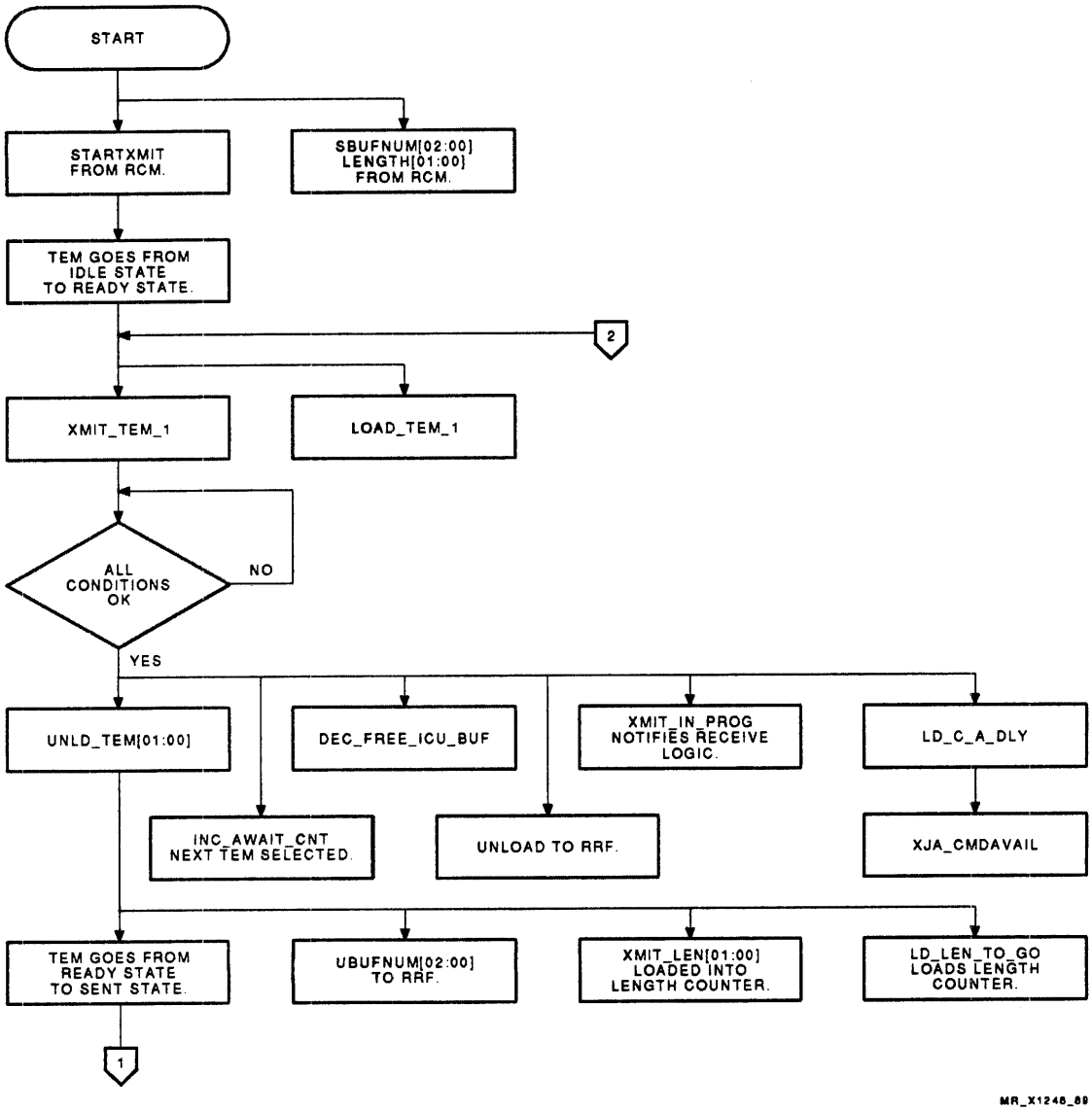


Figure 4–12 (Cont.) Flow Diagram of XCE Transmit Sequence

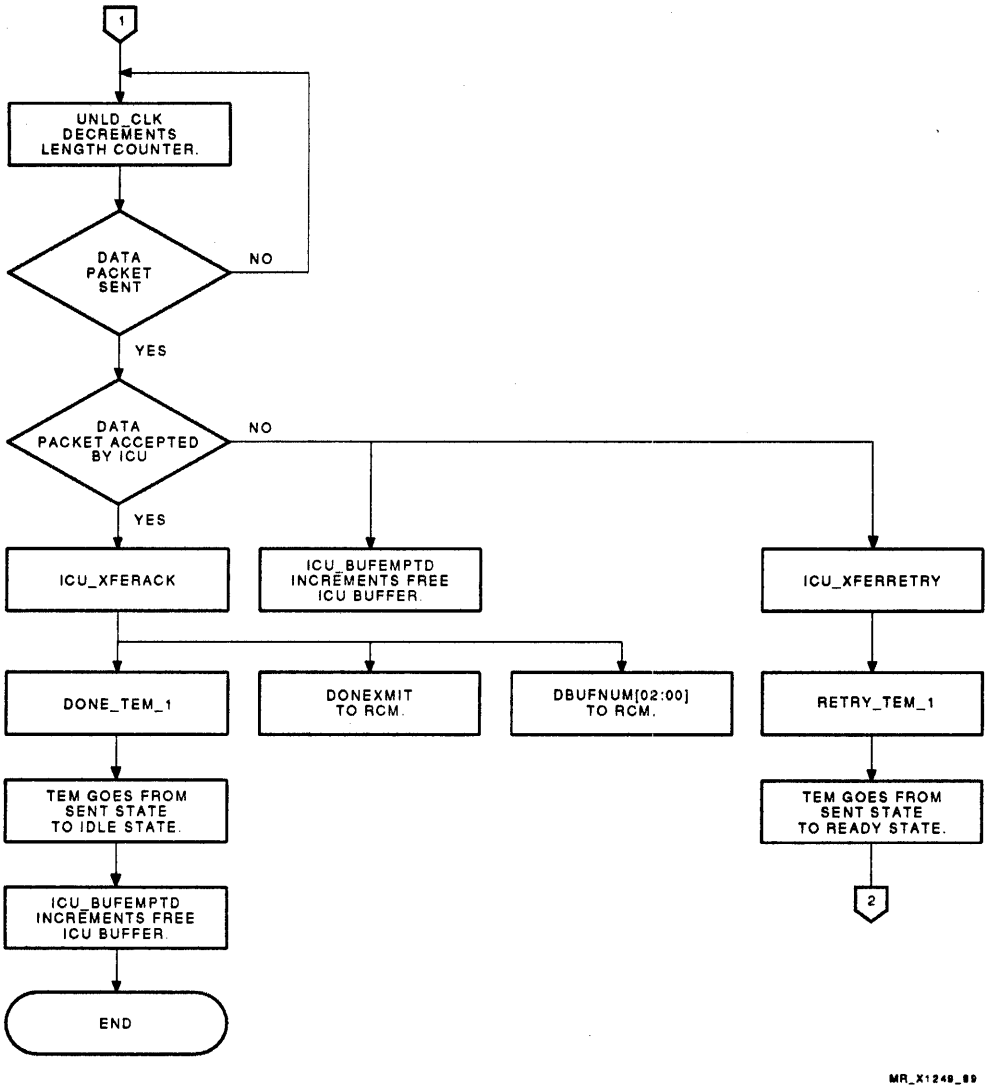


Figure 4–12 Flow Diagram of XCE Transmit Sequence

SBUFNUM[02:00] and LENGTH[01:00] are applied to three TEM buffers associated with the three TEMs.

STARTXMIT is applied to the three TEMs (TEM1 shown in the block diagram). The TEMs have three states: idle, ready, and sent. STARTXMIT moves TEM1 from the idle state to the ready state, causing the assertion of TEM_1_RDY to an await counter, and LOAD_TEM_1 to the TEM1 buffer. LOAD_TEM_1 loads the buffer number and length information into the TEM1 buffer.

The await counter sequences through the asserted ready inputs from the TEMs selecting the next one in order. When TEM_1_RDY asserts, the counter asserts a corresponding transmit command to the TSM (XMIT_TEM_1). XMIT_TEM_1 informs the TSM that a transmit request is pending and the buffer number and length information are loaded into the TEM1 buffer.

The TSM executes the transmission provided the following conditions are true:

- No data packet is on its way to the XJA from the ICU (ICU_CMAVAIL false).
- No data packet received from the ICU is still being transferred through the XDE to the TRF (RCV_IN_PROG false).
- No collision has occurred on the CBI for which the ICU has not initiated a retry (COLLI_XMIT false). A collision occurs when a data packet from the ICU is not accepted because the CBI is being used to transmit data.
- No transmit operation is already in progress (TO_GO_GTR_2 false).
- An ICU receive buffer is available to accept the data (FREE_ICU_BUF true).

If all the conditions are met, the TSM responds to XMIT_TEM_1 by unloading the RRF buffer and executing the transmission. To accomplish this, the TSM:

- Asserts XMIT_IN_PROG to the XCE receive logic to notify it that a transmit operation is executing.
- Asserts DEC_FREE_ICU_BUF, which decrements a free ICU buffer counter. The ICU now has one less free buffer as one of its buffers is now used to receive the data packet being transmitted.
- Asserts UNLOAD to the RRF to unload the selected buffer over the CBI and into the XDE.
- Asserts LD_C_A_DLY causing XJA_CMDAVAIL to output to the ICU, informing it that a data packet is to follow.
- Asserts INC_AWAIT_CNT to the await counter, which then selects the next TEM in the ready state. The TSM does not respond to the next XMIT signal from the await counter until it completes the current transmission as indicated by TO_GO_GTR_2 (one of the conditions before a transmission is initiated).
- Asserts UNLD_TEM[01:00], which identifies the TEM (TEM1) selected for this transmission. UNLD_TEM[01:00] is applied to a buffer multiplexer, which then selects the TEM1 buffer for output. The buffer number output (UBUFNUM[02:00]) is sent to the RRF to select the buffer that is to be unloaded. The length output (XMIT_LEN[01:00]) is loaded into the length counter to control the length of the transmission.
- Asserts LD_LEN_TO_GO to the length counter to load in the length value from the TEM1 buffer.

UNLD_TEM[01:00] is also applied to TEM1, causing it to move from the ready state to the sent state where it remains until the transmission is completed.

Done and retry logic also receive UNLD_TEM[01:00] to identify which transmission the next ICU acknowledge, or retry, is associated with.

UNLD_CLK, from the CLKX logic, decrements the length counter for each data cycle that executes. When the data packet is sent, the length counter informs the TSM of this by the negated state of the signal TO_GO_GTR_2. The TSM is now free to execute another transmission from one of the other TEMs provided all the correct conditions exist. This is true even though an acknowledge or retry signal is not yet received from the ICU for the data packet just sent.

If the data packet was accepted by the ICU, ICU_XFERACK is received by the XCE done logic. The done logic then:

- Asserts DONE_TEM_1 to TEM1, moving it from the sent state to the idle state.
- Asserts DBUFNUM[02:00] to the RCM to specify which RRF buffer was unloaded. The DBUFNUM[02:00] code is identical to the SBUFNUM[02:00] code shown in Table 4–2.
- Asserts DONEXMIT to the RCM, informing it that the data is accepted by the ICU and the RRF buffer specified by DBUFNUM[02:00] is available for new data.

When the ICU transfers the data from its receive buffer, it asserts ICU_BUFEMPTD to the XCE, which then increments the free ICU buffer counter, indicating that another ICU buffer is available to receive data.

If the data packet was not accepted by the ICU, ICU_XFERRETRY is received by the XCE retry logic. The retry logic then asserts RETRY_TEM_1 to TEM1, moving it from the sent state to the ready state for another try at transmitting the data. The ICU also sends ICU_BUFEMPTD to increment the free ICU buffer counter as the ICU buffer reserved for the transmit data was not used.

4.4.2 Receive Logic

The XCE receive logic controls the reception of data from the ICU, and the transfer of the data through the XDEs to the CBI. Figure 4–13 is a block diagram of the XCE receive logic. Figure 4–14 is a flow diagram of the XCE receive sequence.

A receive sequence is initiated when ICU_CMDAVAIL is received from the ICU, indicating that a data packet is to follow starting with the next cycle. ICU_CMDAVAIL is applied to start receive logic along with XMIT_IN_PROG from the XCE transmit logic.

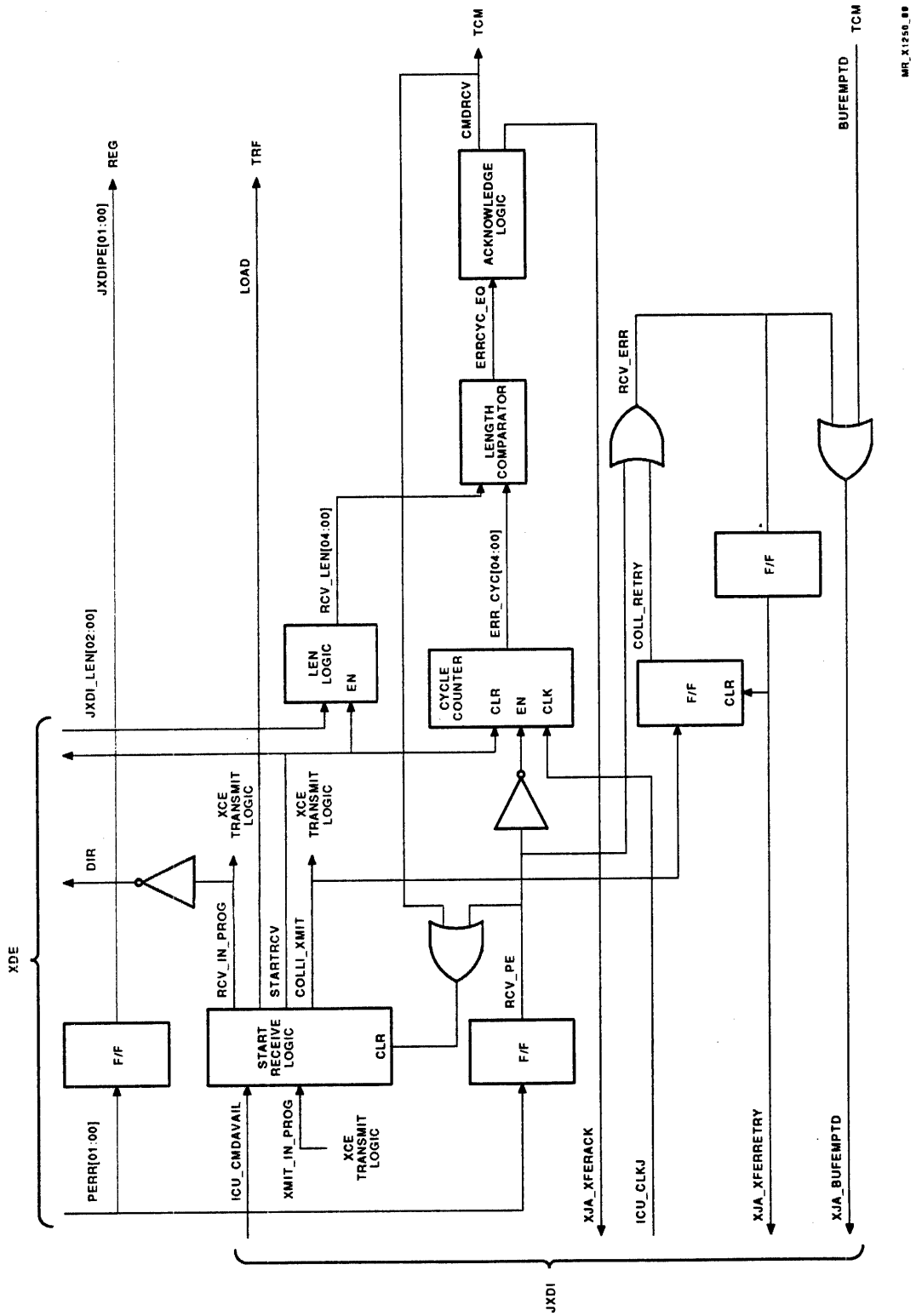


Figure 4–13 XCE Receive Logic

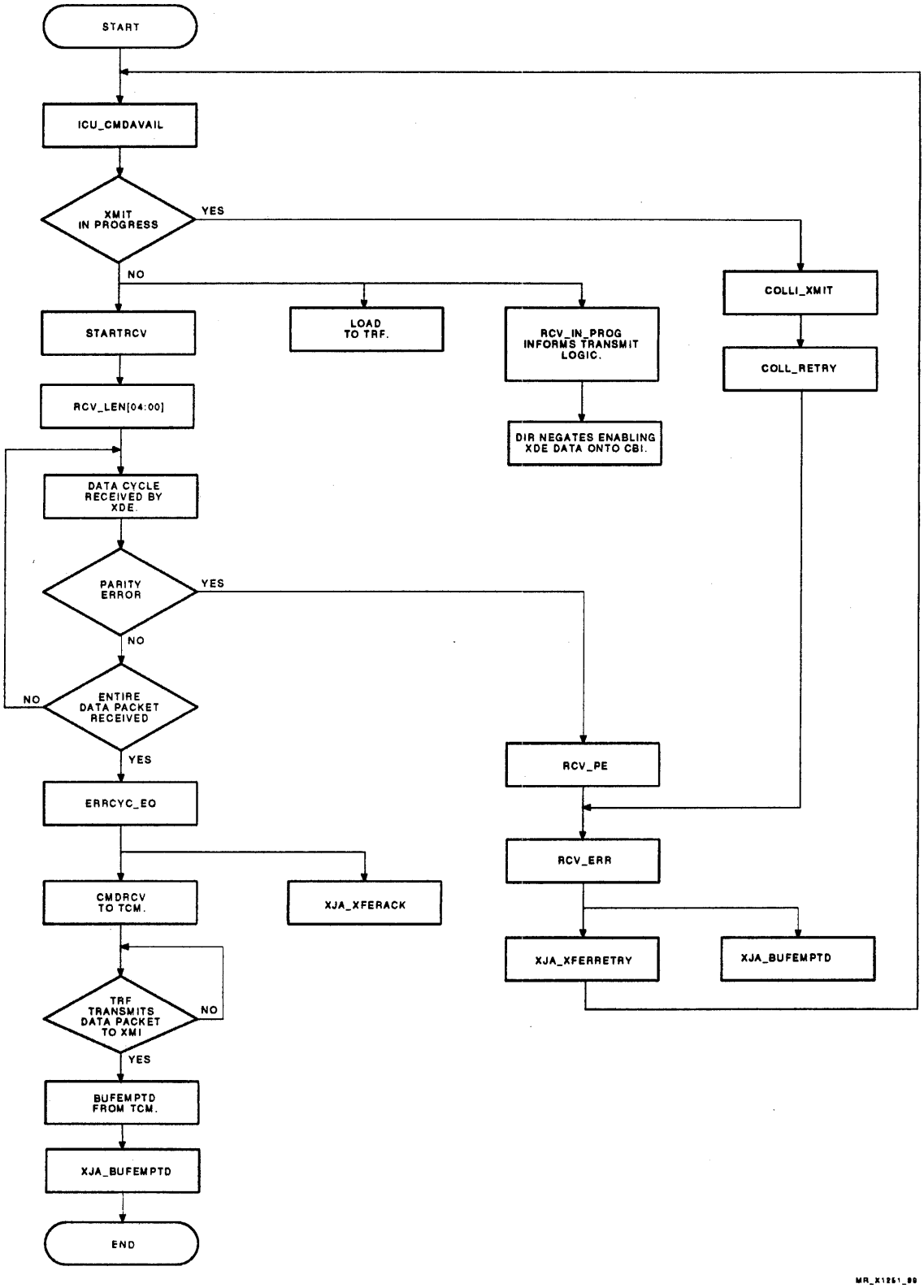


Figure 4–14 Flow Diagram of XCE Receive Sequence

If a transmit sequence is executing in the XCE transmit logic, XMIT_IN_PROG will be true. In this case the start receive logic asserts COLLI_XMIT indicating that a collision state exists. COLLI_XMIT is sent to the transmit logic where it inhibits any new transmissions from initiating until the ICU has retried the transmission. In addition, COLLI_XMIT asserts COLL_RETRY, which asserts RCV_ERR and then XJA_XFERRETRY. XJA_XFERRETRY commands the ICU to retry the transfer. RCV_ERR also asserts XJA_BUFEMPTD to inform the ICU that the TRF buffer reserved for the data packet is still available.

If a transmit sequence is not executing in the XCE transmit logic, XMIT_IN_PROG will be false. In this case, the start receive logic asserts:

- RCV_IN_PROG to the XCE transmit logic, informing it that a data packet is being received and inhibiting the transmit logic from initiating a new transmit sequence. When RCV_IN_PROG asserts, it causes DIR to negate. DIR is sent to the XDE where its negated state enables the XDE data path onto the CBI.
- LOAD to the TRF, informing it to prepare to receive a data packet over the CBI.
- STARTRCV to the XDE where it controls the alignment of the data being received from the JXDI (Section 4.3.3).

In addition, STARTRCV enables length logic, which receives JXDI_LEN[02:00] from the XDE. JXDI_LEN[02:00] is obtained from the command/address word of the data packet and indicates the size of the packet.

The length logic outputs the data packet length as RCV_LEN[04:00], which is loaded into a length comparator where the length is compared with the number of error-free data cycles as described in the following.

Each data cycle that passes through the XDE is checked for parity. The XCE is informed of any parity errors by means of PERR[01:00]. (Each byte is checked separately resulting in two parity error bits.) The parity error bits are sent to REG as JXDIPE[01:00], where they set parity error bits in the error summary register (ERRS).

In addition, PERR[01:00] are applied to a flip-flop, which then outputs RCV_PE. So long as there are no parity errors, RCV_PE remains false enabling a cycle counter. The counter is reset by STARTRCV at the beginning of the data packet and incremented by ICU_CLKJ from the JXDI. Therefore, with each error-free data cycle, the counter output (ERR_CYC[04:00]) is incremented. The error-free cycle count is compared to the length count in the length comparator. When the two counts are equal, the comparator asserts ERRCYC_EQ indicating that the entire data packet is received without any parity errors. ERRCYC_EQ is applied to acknowledge logic, which in turn asserts XJA_XFERACK to the ICU and CMDRCV to the TCM. CMDRCV informs the TCM that the data transfer from the JXDI to the TRF has completed successfully. CMDRCV also resets the start receive logic negating RCV_IN_PROG and asserting DIR.

When the TRF successfully transmits the data packet to the XMI bus, the TCM sends BUFEMPTD to the XCE, which then asserts XJA_BUFEMPTD to the ICU. XJA_BUFEMPTD informs the ICU that a TRF buffer is emptied and is available for new data.

If a parity error is detected during the reception of the data packet, the associated error bit from the XDE is asserted, causing RCV_PE to assert. RCV_PE disables the cycle counter, stopping the ERR_CYC[04:00] count. RCV_PE also asserts RCV_ERR, which then sends a retry command and a buffer empty signal to the ICU. It also resets the start receive logic.

4.5 XMI Corner

Figure 4–15 is a block diagram of the XMI corner.

Each node on the XMI bus has a standard interface called the XMI corner that connects the node-specific logic to the XMI bus. The XMI corner assures a standard electrical and logical interface to the XMI bus.

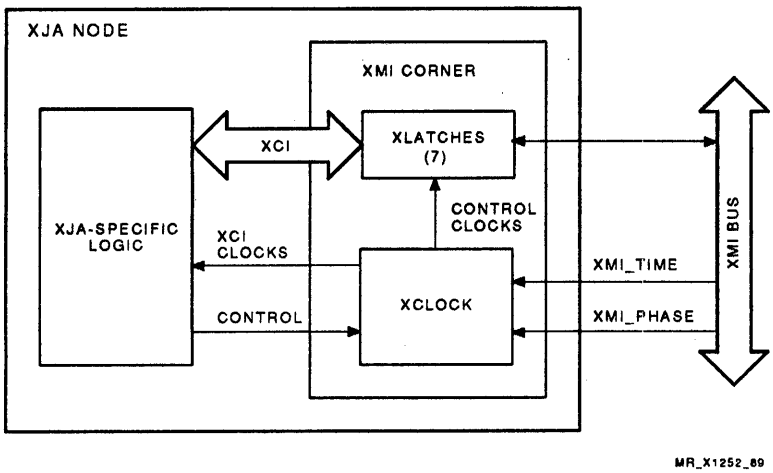


Figure 4–15 XMI Corner Block Diagram

4.5.1 Physical Description

The XMI corner consists of seven XLATCH chips and one XCLOCK chip. The XLATCHES interface the data paths and the XMI bus control signals to the XJA. The bus that interconnects the XJA specific logic to the XLATCHES is called the XCI (XMI bus to Corner Interface). The XCLOCK generates clocks to control the data flow through the XLATCHES and to provide the XJA logic with XCI clocks synchronized to the XMI bus.

The XMI corner has no control logic. All control signals are provided by the XJA logic.

4.5.2 XCI Clocks

The XMI bus is a synchronous bus with a cycle time of 64-ns. Timing of the XMI bus is controlled by clock signals XMI_TIME and XMI_PHASE. These two signals are distributed radially to the XMI corner of each node, thereby reducing skew and ensuring good signal integrity. XCLOCK receives XMI_TIME and XMI_PHASE and uses them to generate six subcycle clocks to clock data through the XLATCHES. The six subcycle clocks are also used in the XJA.

Timing of the XCI clocks is shown in Figure 4–16. XMI_TIME is a 21.3-ns clock. XMI_PHASE is a 64-ns asymmetrical clock. The rising edge of XMI_TIME that coincides with the negative phase of XMI_PHASE is selected to generate the first subcycle XCI_C12. The XCLOCK functions to generate another subcycle every 10.7 ns, resulting in six subcycles across a 64-ns cycle, with a phase relationship as shown in Figure 4–16. The subcycles generated in each XMI corner are synchronized to XMI_TIME and XMI_PHASE and are in sync with each other.

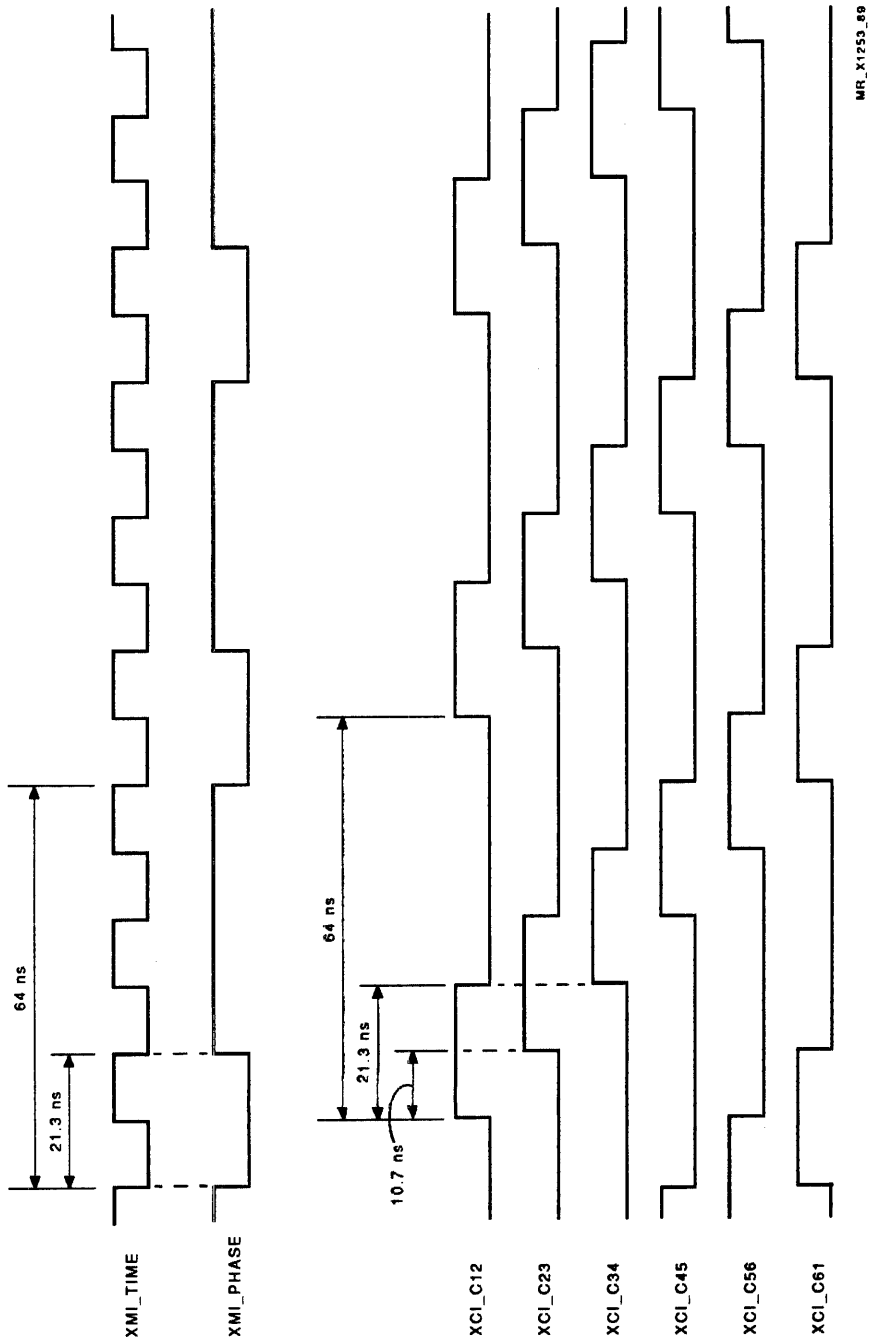


Figure 4–16 XCLOCK Timing

4.6 XDC Receive Logic

The XDC receive logic (Figure 4–17) consists of the XRC (XMI receive logic), the RRF (receive register file), and the RCM (receive control machine). The XRC interfaces with the XMI bus (through the XMI corner) and receives data packets from the XMI bus. It informs the RCM of the reception of the packets, which then control the packet processing.

DMA and CPU packets are transferred to the RRF where they are loaded into buffers. Upon command from the XCE, the buffers are unloaded and the data assembled into JXDI format.

XMI read/write register packets access the XJA registers under control of the RCM.

A general description of the XRC and RRF is given followed by a description of how the various types of data packets are processed through the XDC receive logic.

4.6.1 XRC

Data, ID, parity, and function information is received from the XMI corner latches as XCI_D[63:00], XCI_ID[05:00], XCI_P[02:00], and XCI_F[03:00], respectively. The information is latched and then output as D[63:00], ID[05:00], P[02:00], and F[03:00]. The signals are applied to a parity checker, which asserts XMIPE[02:00] and PE if an error is detected. XMIPE[02:00] identifies the location of the parity error as shown in Table 3–4. XMIPE[02:00] is sent to REG where it sets a bit in the XMI bus error register (XBER) and the error summary register (ERRS). PE is sent to the RCM where it terminates the receive process, and to confirmation logic that generates a no-ack response (XCI_TCNF) for the XMI bus. The preceding occurs for every XMI cycle.

An XRC decoder receives the function and data bits that identifies all the input cycles. The decoder transfers this information to RCM as CACYCTYP[03:00], DATCYCTYP[01:00], and WRLEN[01:00].

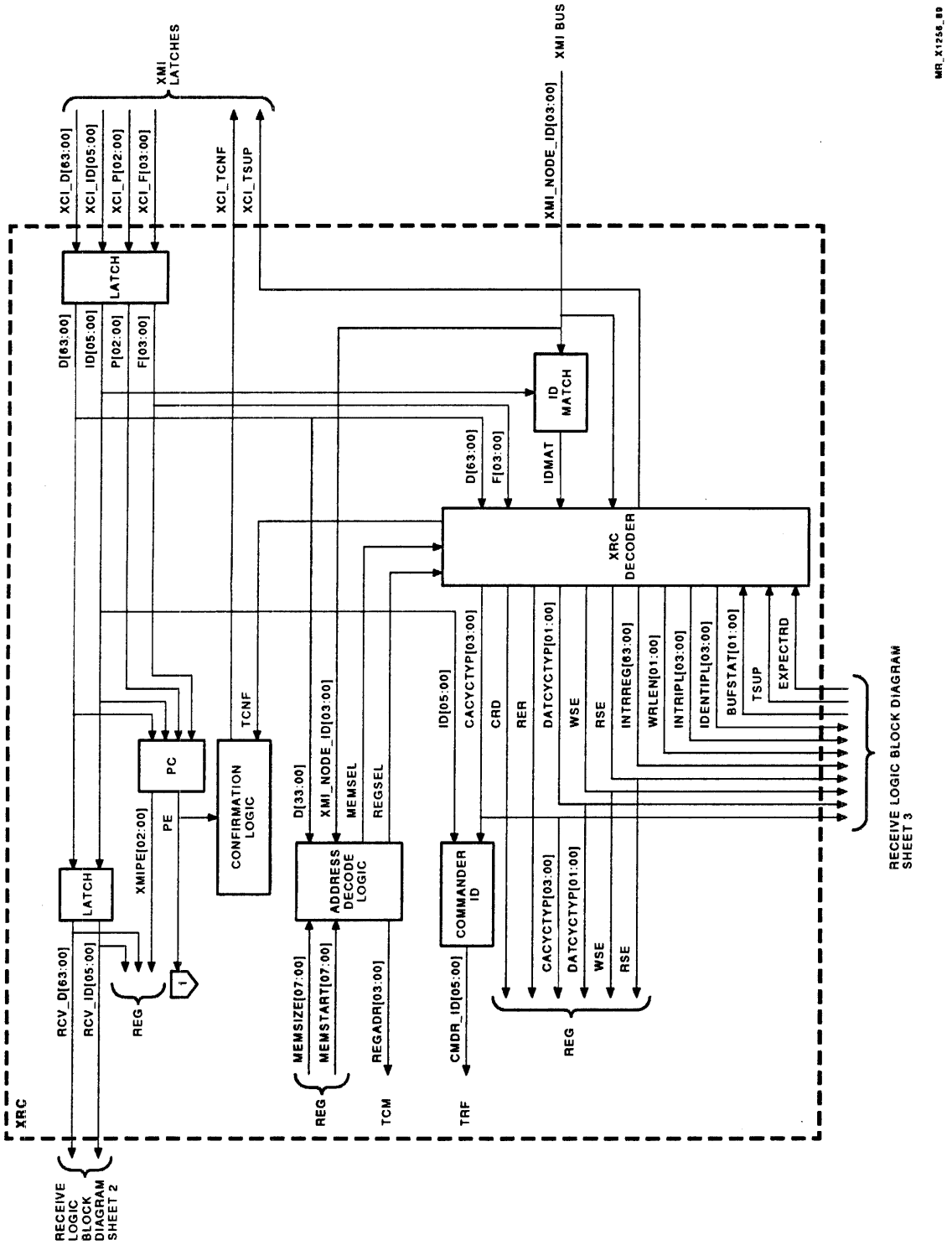


Figure 4–17 (Cont.) Receive Logic Block Diagram

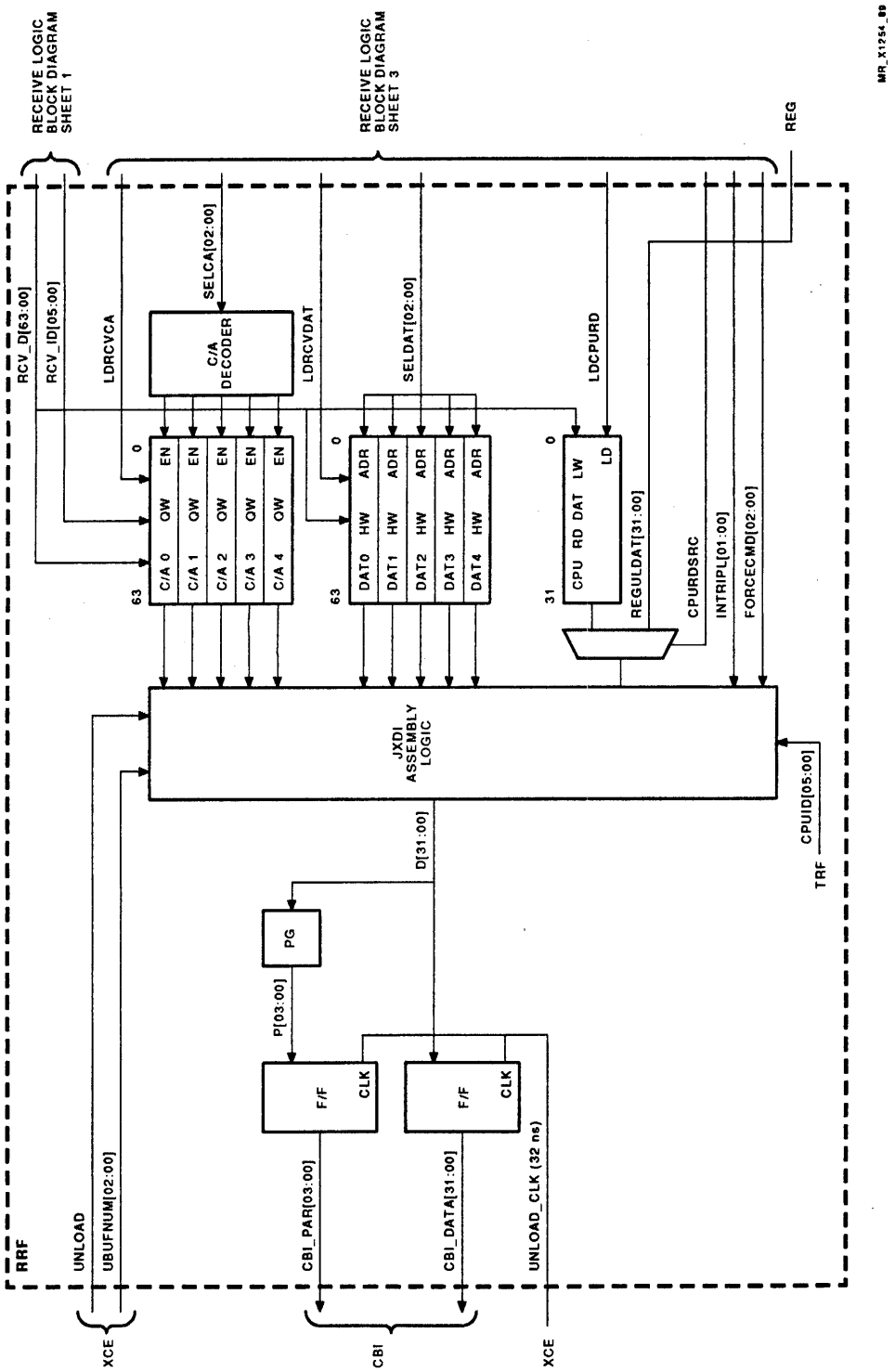


Figure 4–17 (Cont.) Receive Logic Block Diagram

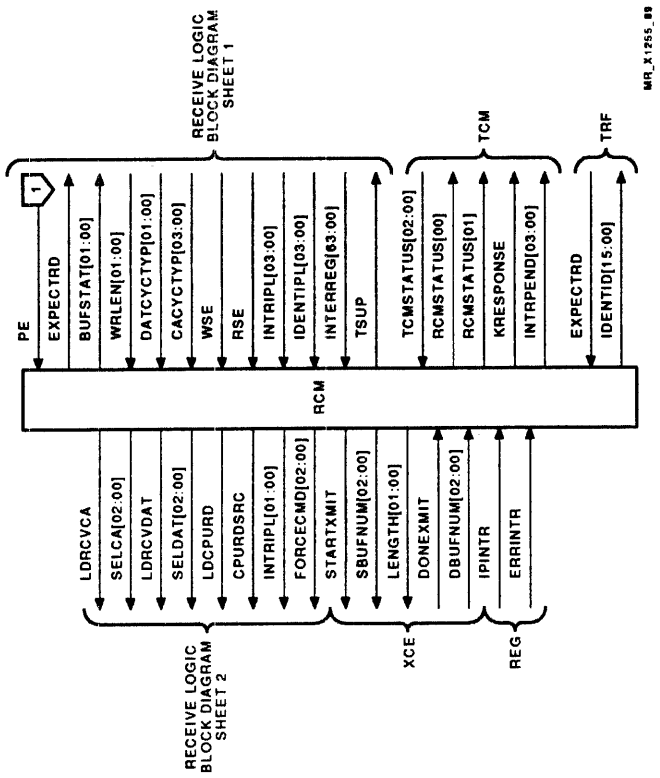


Figure 4–17 Receive Logic Block Diagram

CACYCTYP[03:00] identifies the command/address cycles as shown in Table 4–4.

Table 4–4 Command/Address Cycle Type

CACYCTYP[03:00]	Cycle Type
0 0 0 0	Not a command/address cycle
0 0 0 1	DMA write command/address
0 0 1 0	DMA read command/address
0 0 1 1	Interrupt command
0 1 0 0	Implied vector interrupt command
0 1 1 1	Identify command
1 0 0 1	XJA register write command/address
1 0 1 0	XJA register read command/address
1 1 0 0	Read locked response
1 1 0 1	Read error response

DACYCTYP[01:00] identifies the data cycles as shown in Table 4–5.

Table 4–5 Data Cycle Type

DATCYCTYP[01:00]	Cycle Type
0 0	Not a data cycle
0 1	Read data return (including IDENT responses)
1 0	Write data

WRLEN[01:00] specifies the length of write data as shown in Table 4–6.

Table 4–6 Write Length

WRLEN[01:00]	Length
0 0	1 quadword
0 1	2 quadwords (octaword)
1 0	4 quadwords (hexword)

Address decode logic examines the address bits from the XMI (D[33:00]) to determine if a data packet is accessing main memory or XJA registers. If data bit [29] is 0, access is to main memory. The decode logic receives MEMSIZE[07:00] and MEMSTART[07:00] from REG. MEMSIZE[07:00] specifies the number of 64-Mbyte sections of main memory available. MEMSTART[07:00] specifies the starting address of the available 64-Mbyte sections. If the access is to available main memory, the logic asserts MEMSEL to the XRC decoder.

If data bit [29] is 1, access is to I/O space. If data bits [22:19] equal XMI_NODE_ID[03:00], then the access is to an XMI space register in the XJA. In this case, the address decode logic asserts REGSEL to the decoder.

Confirmation logic provides ack or no-ack returns to the XMI bus for each cycle received. The XRC decoder checks conditions associated with the specific type of data packet and commands the confirmation logic (using TCNF) to ack the data cycle if the conditions are met. Otherwise, a no-ack response is returned to the XMI bus. As already mentioned, parity errors also result in XMI no-acks.

4.6.2 RRF

The RRF (receive register file) buffers data packets from the XRC and, under control of the RCM, restructures the data into the format required by the JXDI. The reformatted data is output to the CBI.

The RRF contains five DMA read/write C/A buffers, five hexword write data buffers, and one CPU read data buffer. The five write data buffers are associated with the five DMA read/write C/A buffers. If a write C/A is loaded into one of the DMA read/write C/A buffers, the write data is loaded into the associated write data buffer. Each write data buffer is entirely dedicated to its associated C/A buffer. If the write length is quadword, the quadword is loaded into the write buffer and the other three quadword locations are not used. A hexword write completely fills a write data buffer. If a read C/A is loaded into one of the DMA read/write C/A buffers, the associated write data buffer is not used.

The CPU read data buffer is 32 bits wide. It receives CPU read data return from the XRC.

RCM monitors the state of the RRF DMA buffers and informs the XRC decoder of buffer availability by use of BUFSTAT[01:00]. The buffer status code is shown in Table 4–7.

Table 4–7 Buffer Status Code

BUFSTAT[01:00]	Status
0 0	Less than two DMA buffers are full
1 0	Two DMA buffers are full
1 1	More than two DMA buffers are full

The XRC uses BUFSTAT[01:00] and the suppress line on the XMI bus to ensure that the XJA does not receive a DMA data packet for which there is no RRF buffer available. If the XRC decoder receives a DMA packet while BUFSTAT[01:00] is 10 (two RRF DMA buffers are full), it asserts XCI_TSUP to the XMI to suppress all XMI traffic. If BUFSTAT[01:00] indicates that more than two DMA buffers are full, the XRC keeps XCI_TSUP asserted until two or less buffers are full. The XRC asserts the suppress line when three buffers are full because it is possible to receive two more DMA C/As from the time that XCI_TSUP is asserted and the XMI bus is suppressed.[1]

The JXDI assembly logic functions to select the buffer to be unloaded and assembles the unloaded data into the format required by the JXDI. The data unloaded from the buffer is in 64-bit quadword format. The output from the assembly logic is in 32-bit longword format. The data longwords are clocked onto the CBI by the 32-ns UNLOAD_CLK from the XCE. The data appears on the CBI as CBI_DATA[31:00]. Byte parity is generated on the data, which then is clocked onto the CBI as CBI_PAR[03:00].

The assembly logic also functions to assemble data returns to the ICU other than DMA packets. Under control of the RCM, the assembly logic formats interrupt, error status, write complete, and other packets into JXDI format for the ICU. These cases are individually described in Section 4.6.3, Section 4.6.4, and Section 4.6.5.

4.6.3 Packet Processing

Data packets received from the XMI bus fall into eleven categories as seen in Figure 4–1.

- DMA read command/address cycle
- DMA write command/address cycle plus the associated data
- CPU read data return cycle received in response to a previous CPU read XMI command
- CPU read lock response cycle in response to a previous CPU read XMI command
- CPU read error response cycle in response to a previous CPU read XMI command
- XMI read of XMI space register command/address cycle
- XMI write of XMI space register command/address cycle plus the associated data
- Interrupt command cycle
- IDENT command cycle
- Implied vector interrupt command cycle
- IDENT response

The following describes the steps involved in processing the eleven types of data packets.

[1] There is not enough time to receive two DMA write data packets. At least one of the two would have to be a DMA read C/A packet.

4.6.3.1 DMA Read/Write Packet Processing

Figure 4–18 is a flow diagram of packet processing for DMA read and DMA write packets.

For a DMA cycle to be accepted (acknowledged) and processed, the following must be true:

- There must be no parity error.
- The XRC decoder must see a command function code for a C/A cycle or a write data function code for write data.
- The XRC decoder must see a write (or write unlock) or a read (or read lock) command in the command field of the command/address longword.
- For a write data cycle, the write data must follow a write command\address cycle and there must be the correct number of write data cycle(s) as specified in the WRLEN[01:00] code issued to the RCM. If this does not occur, a write sequence error (WSE) is issued to REG and RCM.
- The XRC decoder must know that the address field of the command/address longword specifies available main memory as indicated by MEMSEL from the address decode logic.

If the above conditions are met, the XRC decoder does the following:

- Asserts TCNF to the confirm logic, which then acknowledges receipt of the cycle to the XMI bus by asserting XCI_TCNF.
- Specifies to the RCM, a DMA read/write command/address cycle by using CACYCTYP[03:00] (Table 4–4) or a write data cycle by using DATCYCTYP[01:00] (Table 4–5).
- If a DMA write command/address cycle, specifies to the RCM the length of the write data by using WRLEN[01:00] (Table 4–6).
- Sends CACYCTYP[03:00] and DATCYCTYP[01:00] to REG to indicate the type of data currently on the RCV_D[63:00] and ID[05:00] lines. REG automatically saves C/A data for use in error analysis in the event of an error interrupt.

The data and ID are transferred to the RRF buffers as RCV_D[63:00] and RCV_ID[05:00], respectively. The RCM loads the C/A data and ID into a C/A buffer using SELCA[02:00] and LDRCVCA. SELCA[02:00] is decoded to enable one of the C/A buffer. LDRCVCA loads the command/address into the selected buffer.

If this is a write operation, the RCM loads the next data quadword (write data) into the write data buffer associated with the selected C/A buffer. The RCM selects the associated buffer using the SELDAT[02:00] select code. LDRCVDAT from the RCM is the loading command. SELDAT[02:00] enables the associated buffer while an internal counter selects the first quadword location in the buffer for the write data. If the write is octaword or hexword in length, SELDAT[02:00] holds the buffer enabled while the counter increments to the next quadword location(s) and LDRCVDAT loads in the write data quadword(s) that follow.

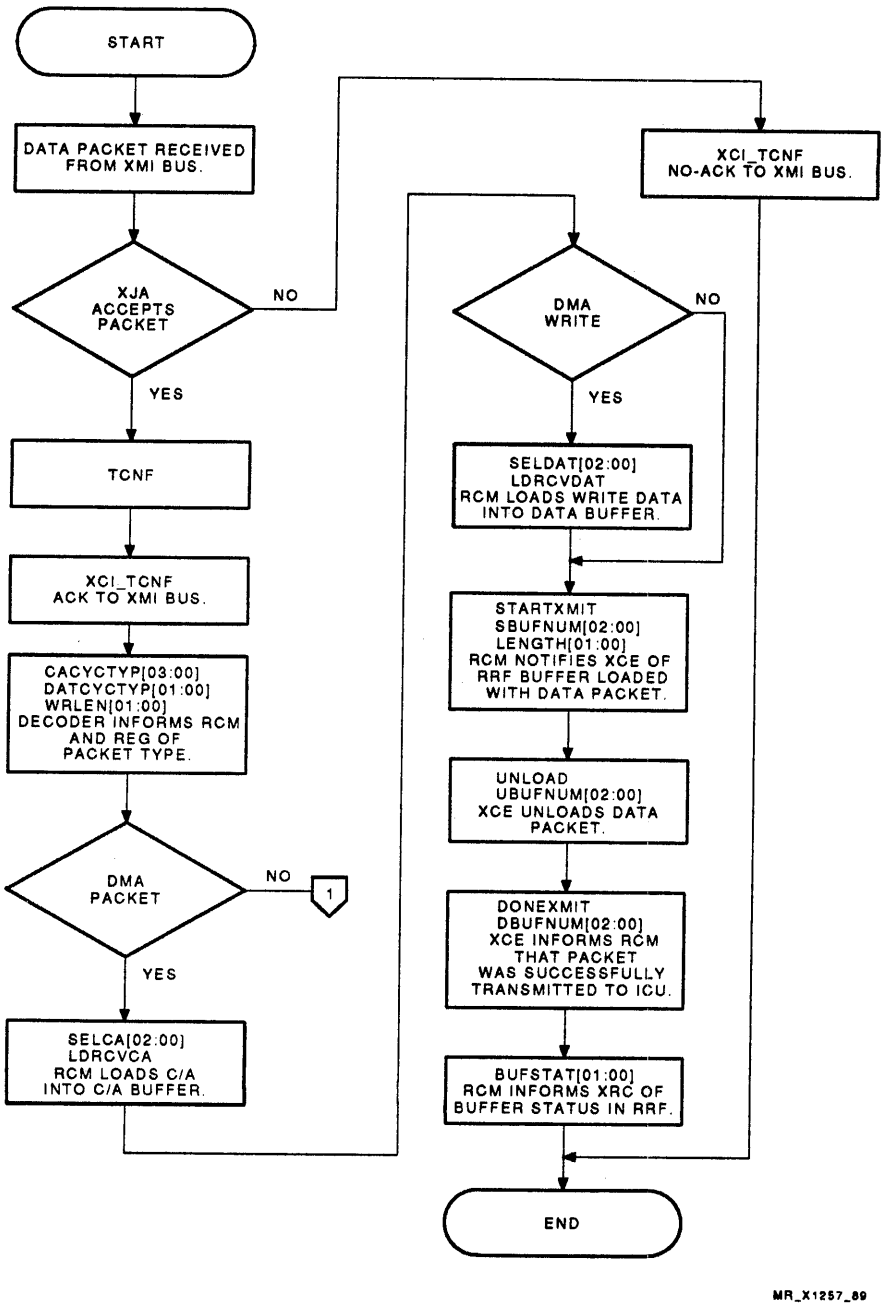


Figure 4–18 (Cont.) Reception of DMA Command/Address Packets and CPU Return Data/Status Packets from XMI Bus

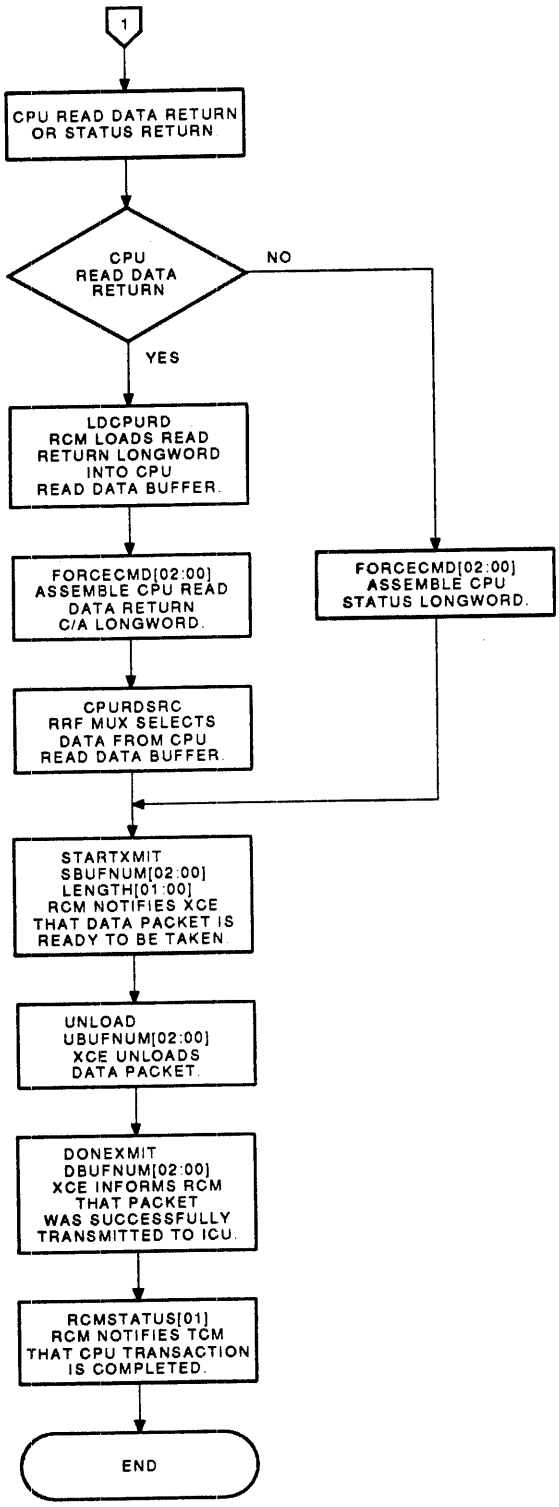


Figure 4–18 Reception of DMA Command/Address Packets and CPU Return Data/Status Packets from XMI Bus

When the data packet is loaded into the RRF, the RCM asserts STARTXMIT and SBUFNUM[02:00] to the XCE, informing it that a data packet is available in the buffer specified by SBUFNUM[02:00]. If the packet contains DMA write data, the RCM informs XCE of the size of the packet using LENGTH[01:00].

When the XCE is ready to take the data packet, it asserts UNLOAD and UBUFNUM[02:00]. UNLOAD transfers the data from the buffer selected by UBUFNUM[02:00], into the JXDI assembly logic which then assembles the data into the format required by the JXDI.

When the data packet is successfully transmitted to the ICU, the XCE notifies the RCM by asserting DONEXMIT and DBUFNUM[02:00]. DBUFNUM[02:00] identifies the buffer whose data was unloaded and sent to the ICU. This buffer is now available for another DMA data packet. This allows the RCM to monitor the status of the RRF C/A buffers and inform XRC of the status by means of BUFSTAT[01:00].

4.6.3.2 CPU Read Data Return and CPU Read Status Return

Figure 4–18 is also a flow diagram of the CPU read data return and CPU read status return packet processing.

The command/address or read data cycle is received by the XRC decoder, which then determines if the cycle is to be accepted. For a CPU read data return cycle or status return cycle to be received and processed, the following must be true:

- There must be no parity error.
- The XRC decoder must see a good read data return (or corrected read data return) or a read error response or read locked response function code for the C/A cycle.
- The XRC decoder must see an asserted IDMAT from ID match logic in the XRC. This occurs when the XJA node ID (XMI_NODE_ID[03:00]) matches the ID associated with the return read data (ID[05:02]), thereby identifying the XJA as the commander that requested the read data.
- The XRC decoder must see an asserted EXPECTRD from the TCM in the transmit logic. EXPECTRD was asserted to the RCM from the TCM, when a CPU read request was transmitted to the XMI bus. EXPECTRD indicates to the XRC decoder that read return data is expected. If EXPECTRD is not asserted, a read sequence error (RSE) is asserted to REG and RCM.

If the above conditions are met, the XRC decoder does the following:

- Asserts TCNF to the confirm logic which acknowledges receipt of the cycle to the XMI bus using XCI_TCNF.
- Specifies to the RCM, a read data return cycle using DATCYCTYP[01:00].
- If the function code specified corrected read return data, CRD is asserted to REG informing it of such.
- For a read lock response, specifies to the RCM a read locked response cycle using DATCYCTYP[01:00].
- For a read error response, specifies to the RCM a read error response cycle using DATCYCTYP[01:00]. In this case, the decoder also asserts RER to REG, indicating a read error response was received.

For a CPU read data return packet, the read data is loaded into the 32-bit CPU read data buffer in the RRF by LDCPURD from the RCM. The buffer output is applied to the JXDI assembly logic through a multiplexer.

The JXDI assembly logic forces the generation of some command/addresses as required for the JXDI. The command/address to be generated is specified by RCM using FORCECMD[02:00]. Table 4–8 gives the force command codes.

Table 4–8 Force Command Code

FORCECMD[02:00]	Command
0 0 0	INTR (from XMI bus)
0 0 1	CPU read return data
0 1 0	CPU read locked data return
0 1 1	Read error status
1 0 0	Read locked status
1 0 1	CPU write complete
1 1 0	IVINTR
1 1 1	ERRINTR (from XJA)

For a CPU read data return cycle, the RCM asserts a 001 force command code to the JXDI assembly logic, which generates a CPU read data return command/address as shown in Figure 2–12. The commander ID is obtained from TRF as CPUID[05:00]. TRF saved the commander ID when the CPU request was transmitted to the XMI bus. The RCM uses CPURDSRC to select the read data in the CPU read data buffer. The selected data is input to the assembly logic, which outputs the data after the command/address. The assembly logic outputs the data longword so it appears on the JXDI one byte per word as shown in Figure 2–12.

For a read error status return or a read lock status return, the RCM commands the assembly logic to format the correct command/address (Figure 2–13) again using the commander ID from the TRF.

The RCM asserts STARTXMIT and SBUFNUM[02:00] to the XCE, informing it that a data packet is available in the buffer specified by SBUFNUM[02:00]. In this case the command/address longword is in the JXDI assembly logic and, for a CPU read data return packet, the CPU read data buffer is specified as the source for the data longword.

The XCE responds by asserting UNLOAD and UBUFNUM[02:00]. UNLOAD transfers the data from the assembly logic and, in the case of a CPU read data return packet, from the CPU read data buffer as selected by UBUFNUM[02:00]. The assembled data is output from the assembly logic in the format required by the JXDI.

When the data packet is successfully transmitted to the ICU, the XCE notifies the RCM by asserting DONEXMIT and DBUFNUM[02:00]. DBUFNUM[02:00] identifies the buffer whose data was unloaded and sent to the ICU. The CPU read data buffer is now available for another CPU read data return longword. This allows the RCM to monitor the status of the forced commands and delete those commands that are successfully transmitted from its command queue, or resubmit a forced command to the TRF.

The RCM then asserts RCMSTATUS[01] to the TCM, informing it that the CPU transaction associated with the forced command, is completed. All the forced commands, except the three interrupts, result from transmissions from the transmit logic to the XMI bus. Therefore, when a noninterrupt force command is transferred to the ICU, the transmit logic is notified by the assertion of RCMSTATUS[01].

4.6.3.3 XMI Read/Write of XMI Space Register Packet Processing

Figure 4–19 is a flow diagram of packet processing for the read and write of XMI space registers.

For an XMI space register read/write cycle to be accepted and processed, the following must be true:

- There must be no parity error.
- The XRC decoder must see a command function code for a C/A cycle, or a write data function code for write data.
- The XRC decoder must see a write (or write unlock), or a read (or read lock) command in the command field of the command/address longword.
- For an XMI write register packet, the write data cycle must follow the write register command/cycle. If this does not occur, a write sequence error (WSE) is asserted to REG and RCM.
- The XRC decoder must know that the address field of the command/address longword specifies an XJA register location as indicated by REGSEL from the address decode logic.

If the above conditions are met, the XRC decoder does the following:

- Asserts TCNF to the confirm logic that acknowledges receipt of the cycle to the XMI bus.
- Specifies to the RCM, an XMI write register or an XMI read register command/address cycle using CACYCTYP[03:00], or a write data cycle using DATCYCTYP[01:00].
- Sends CACYCTYP[03:00] and DATCYCTYP[01:00] to REG to indicate the type of data currently on the RCV_D[63:00] and ID[05:00] lines. REG automatically saves C/A data for use in error analysis in the event of an error interrupt.
- For a read XMI space register C/A, specifies a read XJA register command to commander ID logic in the XRC. The logic passes the commander ID (ID[05:00]) to the TRF as CMDR_ID[05:00]. The TRF uses it for the ID (address) of the return read data to the XMI bus.

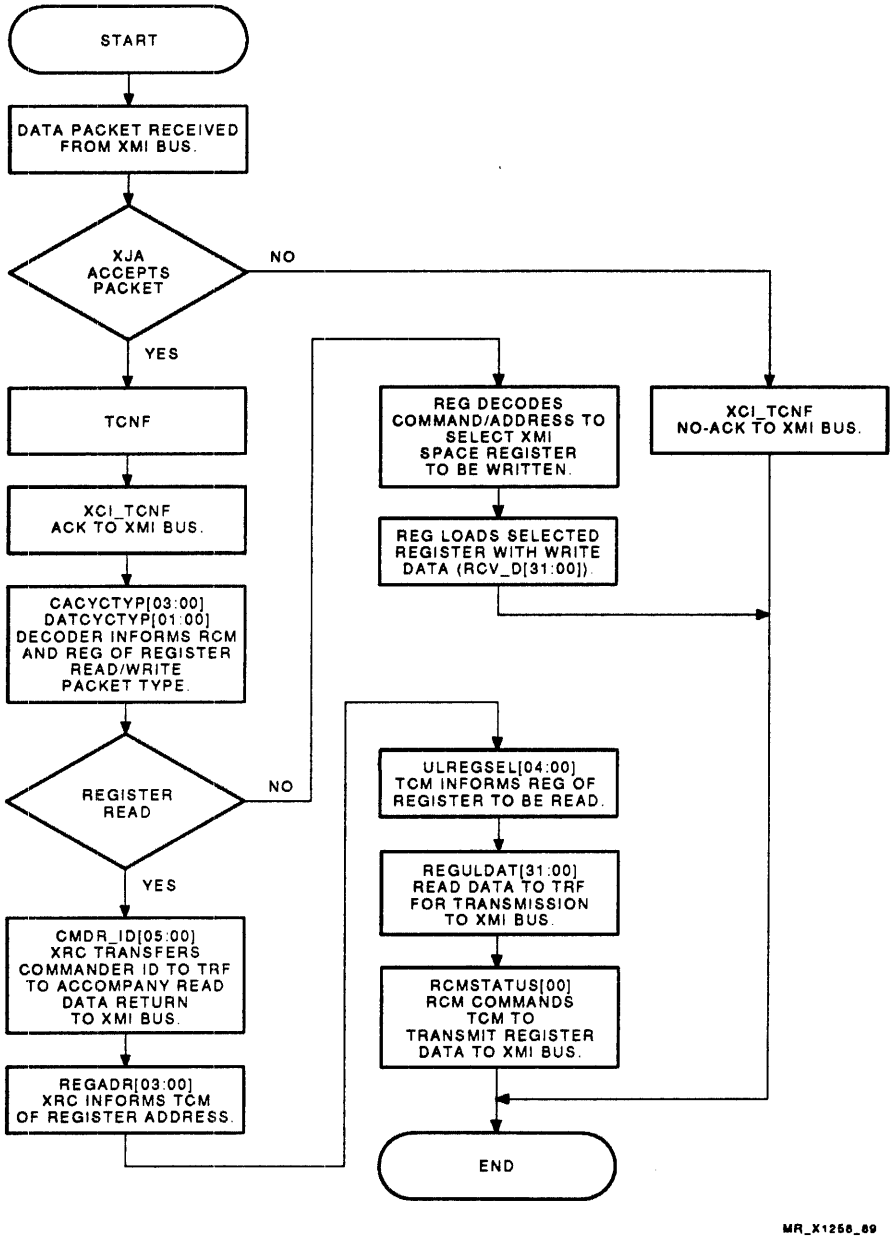


Figure 4–19 Reception of XMI Space Register Read/Write Command/Address Packets from XMI Bus

The address decode logic extracts the XJA target register from the command/address and outputs it as REGADR[03:00] to TCM. The read access to the XMI space registers is controlled by the TCM.

For a read of an XMI space register, the TCM uses REGADR[03:00] to generate ULREGSEL[04:00], which selects the register to be unloaded (read). The data from the selected register (REGULDAT[31:00]) is transmitted to the TRF where it is transmitted to the XMI bus by the TCM (Figure 4–23). The RCM commands the transmit logic to transmit the register data to the XMI bus by asserting RCMSTATUS[00] to the TCM.

For a write of an XMI space register, REG uses CACYCTYP[03:00] from XRC to identify the data cycle as a command/address cycle. REG uses the command/address data to select the XMI space register, which is loaded with the write data contained in the following data cycle.

4.6.3.4 Basic Interrupt and WEI Implied Vector Interrupt Packet Processing

For a basic interrupt or WEI implied vector interrupt command cycle to be accepted and processed, the following must be true:

- There must be no parity error.
- The XRC decoder must see a command function code.
- The XRC decoder must see an interrupt command or an implied vector interrupt command in the command field of the command/address longword.
- The XRC decoder must see that the interrupt destination field D[15:00] matches the node ID of the XJA (XMI_NODE_ID[03:00]).

If the above conditions are met, the XRC decoder does the following:

- Asserts TCNF to the confirm logic that acknowledges receipt of the cycle to the XMI bus.
- Specifies to the RCM, a basic interrupt command cycle or an implied vector interrupt command cycle through CACYCTYP[03:00].
- For a basic interrupt command, specifies to the RCM the IPL of the interrupt through INTRIPL[03:00].
- For a basic interrupt command, specifies to the RCM the status of pending interrupts using INTRREG[63:00]. The status of pending interrupts is shown in Table 4–9. If more than one IPL is set, RCM asserts XCT_TSUP to the XRC decoder and no more commands are acknowledged until the interrupts are serviced.

Table 4–9 Status of Pending Interrupts

INTRREG[63:00]	Pending Interrupt Levels
[15:00]	IPL 14, node 0 to 15
[31:16]	IPL 15, node 0 to 15
[47:32]	IPL 16, node 0 to 15
[63:48]	IPL 17, node 0 to 15

The RCM forces an interrupt or an implied vector interrupt to the JXDI assembly logic by asserting a FORCECMD[02:00] code of 000 or 110, respectively (Table 4–8). Both codes result in the same interrupt command format from the assembly logic as shown in Figure 2–17. The IPL used for the interrupt command is supplied by RCM as INTRIPL[01:00].

Sixty-four flip-flops in the XRC reflect the status of pending interrupts from the XMI bus (excluding implied vector interrupts). Sixteen flip-flops are assigned to each of the four IPLs. The sixteen flip-flops at each IPL correspond to the XMI nodes. This results in four flip-flops for each XMI node, one for each IPL. The VAX 9000 system has only 12 adapters on the XMI bus (excluding the XJA), so there would be only 48 flip-flops used to indicate the status of XMI pending interrupts.

When a normal interrupt is received by the XRC, it sets one of the flip-flops according to the interrupting node number and the IPL of the interrupt. When a pending interrupt is serviced (as indicated by an IDENT command on the XMI bus), the pending flip-flop associated with the interrupt is reset.

The XRC informs the RCM of the status of pending interrupts with the signal INTERREG[63:00]. When the RCM is not doing a DMA or CPU transfer and interrupt(s) are pending, it initiates an interrupt transfer to the XJA using the higher IPL interrupt first. When more than one interrupt exists at the same IPL, the higher numbered nodes have priority.

4.6.3.5 IDENT Command Cycle

IDENT commands are received by the XJA for the purpose of updating the status of pending interrupt levels. Nothing is transferred to the ICU and there is no acknowledgment returned to the XMI bus.

For a valid identify command cycle to be received and processed, the following must be true:

- There must be no parity error.
- The XRC decoder must see a command function code.
- The XRC decoder must see an identify command in the command field of the command/address longword.

If the above conditions are met, the XRC decoder does the following:

- Specifies to the RCM, an identify command cycle using CACYCTYP[03:00].
- Resets the associated pending interrupt level in the XRC decoder (Table 4–9). RCM is notified of the new interrupt status using INTERREG[63:00].

4.6.4 CPU Write Complete

When any CPU write transaction is completed, the TCM notifies the RCM of this using TCMSTATUS[02:00] (Table 4–12). The RCM then forces a CPU write complete command to the JXDI assembly logic in the RRF using FORCECMD[02:00] (see Table 4–8). The assembly logic assembles a CPU status return command cycle (Figure 2–13) using the CPUID[05:00] from the TRF and a CPU write complete code in the command field.

4.6.5 CPU Read of an XJA Private Register

Another function handled by the RRF is a CPU return of read data from an XJA private register. The processing of the CPU read request and the reading of the register is handled by the TCM in the transmit logic. The read data is made available to the multiplexer in the RRF and the RCM is notified of the available data by TCMSTATUS[02:00] from the TCM (Table 4–12). The RCM then forces a CPU read return data command (FORCECMD[02:00] = 001, Table 4–8) with the ID again being provided from TRF as CPUID[05:00]. For the data longword that follows, the RCM asserts CPURDSRC to select the register data (REGULDAT[31:00]) for the JXDI assembly logic, which assembles it into the JXDI required format.

4.7 XDC Transmit Logic

The XDC transmit logic (Figure 4–20) consists of a transmit control machine (TCM) and a transmit register file (TRF). The TRF receives data from the CBI, buffers it, and formats it for the XMI bus. The TCM controls the TRF, arbitrates for the XMI bus, and communicates with the XDC receive logic as required. CPU access to the XJA private registers is also handled by the transmit logic.

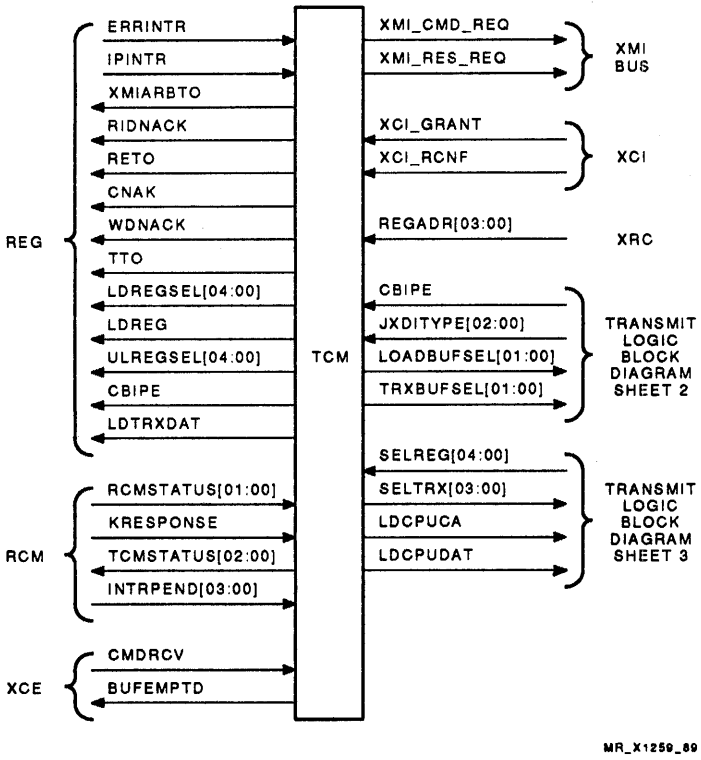


Figure 4–20 (Cont.) Transmit Logic Block Diagram

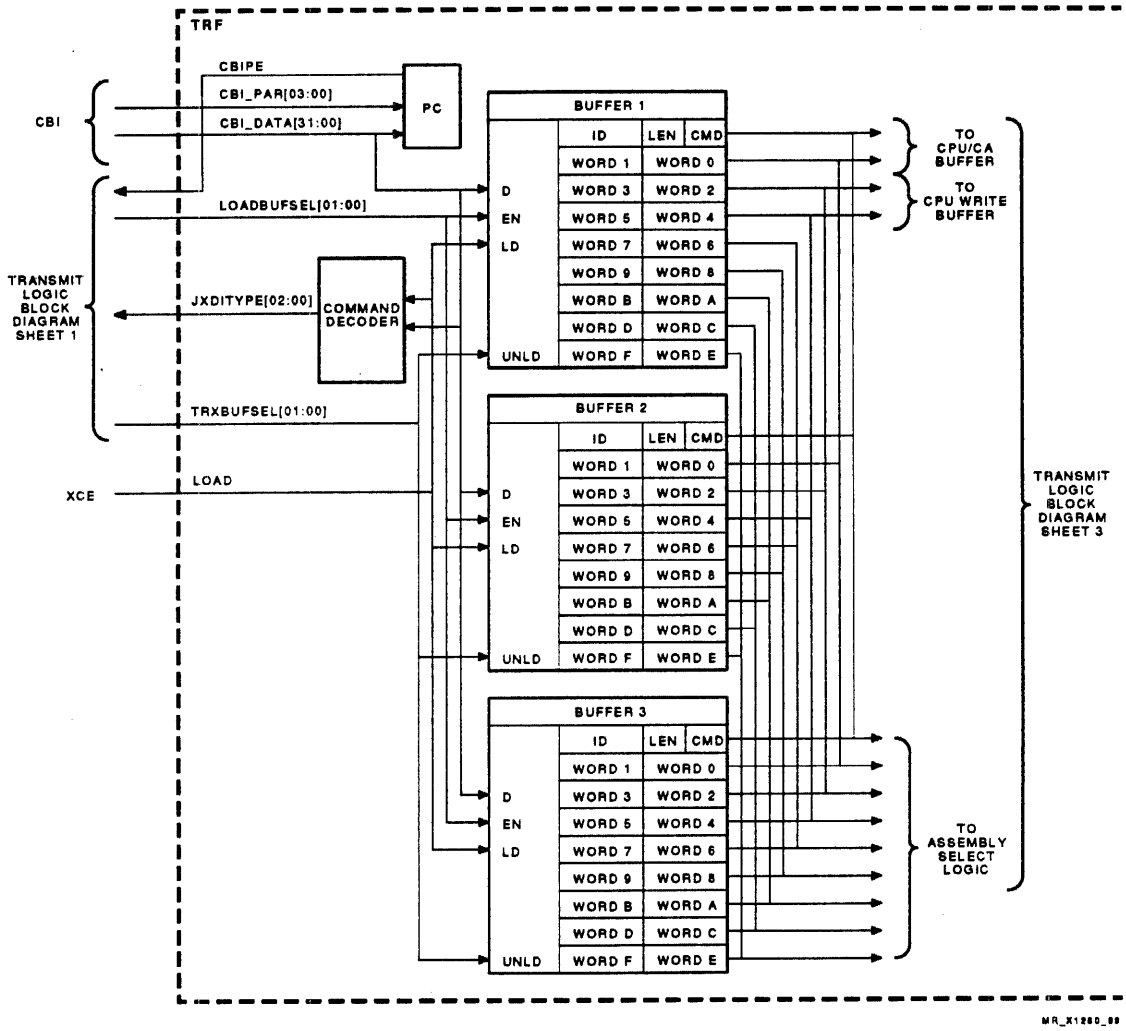


Figure 4–20 (Cont.) Transmit Logic Block Diagram

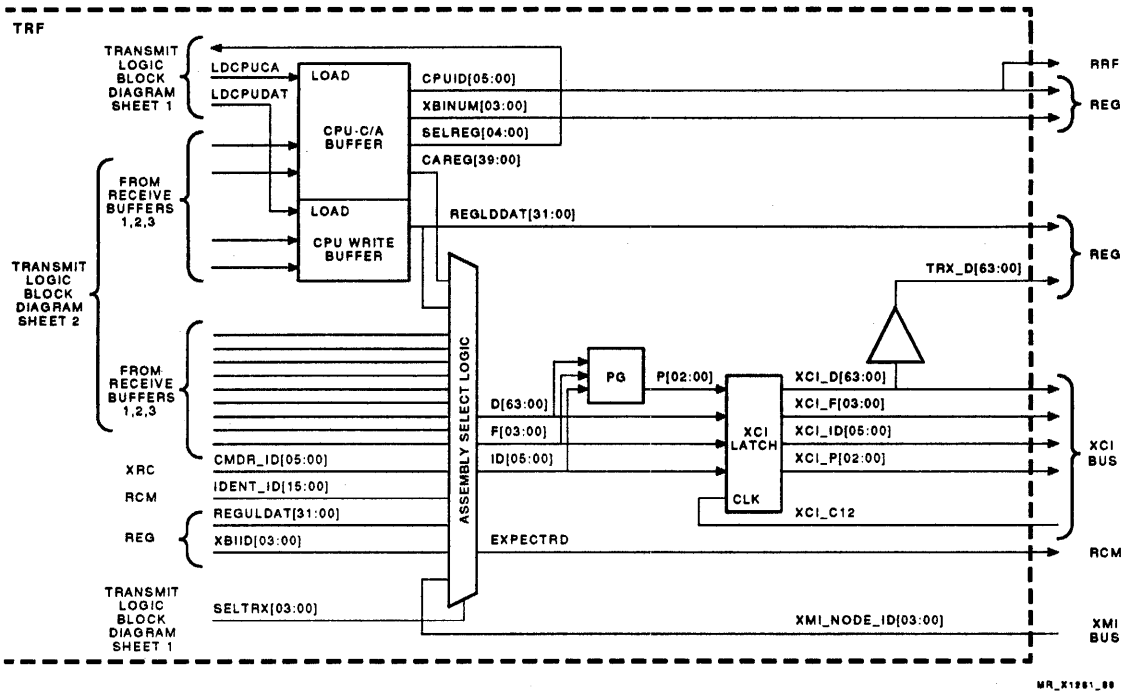


Figure 4–20 Transmit Logic Block Diagram

4.7.1 Transmit Register File

The TRF contains:

- Three buffers to store DMA read return data. Each of the buffers is capable of storing a read data return command/address and up to a hexword of read data.
- A CPU command/address buffer to store CPU command/addresses.
- A CPU write data buffer to store CPU write data.
- Assemble/select logic to assemble data into XMI format.

Data and parity are received from the CBI as CBI_DATA[31:00] and CBI_PAR[03:00], respectively. On each cycle, the data and parity bits are applied to a parity checker, which outputs CBIPE to the TCM if a parity error is found. The TCM terminates the data transfer and sends CBIPE to the error summary register (ERRS) in REG.

A command decoder decodes each command/address cycle from the CBI to determine the data type. The decoder outputs JXDITYPE[02:00] to the TCM, informing it of the type of data being loaded into the TRF buffer. The JXDITYPE[02:00] code is shown in Table 4–10.

Table 4–10 Data Type Code

JXDITYPE[02:00]	JXDI Transaction Type
0 0 0	No-op
0 0 1	Quadword read data return
0 1 0	Octaword read data return
0 1 1	Hexword read data return
1 0 0	CPU write
1 0 1	CPU read
1 1 0	Read locked status
1 1 1	Read error status

The assembly/select logic is controlled by SELTRX[03:00] from the TCM. Knowing the type of data in the TRF buffer, the TCM uses SELTRX[03:00] to command the assembly/select logic to generate the correct function code, obtain the commander ID, and format the buffer data as required for the XMI bus. The SELTRX[03:00] code is shown in Table 4–11.

Table 4–11 XMI Cycle Type Code

SELTRX[03:00]	XMI Function	Cycle Type
0 0 0 0	GRD0	Return good read data 0
0 0 0 1	GRD1	Return good read data 1
0 0 1 0	GRD2	Return good read data 2
0 0 1 1	GRD3	Return good read data 3
0 1 0 0	LOC	Return locked response
0 1 0 1	RER	Return error response
0 1 1 0	GRD0	Return XMI space register good read data to XMI bus
0 1 1 1	No-op	–
1 0 0 0	CMD	CPU C/A (read or write) to XMI bus
1 0 0 1	WDAT	CPU write data to XMI bus
1 0 1 0	No-op	–
1 0 1 1	No-op	–
1 1 0 0	CMD	IDENT command at IPL 14
1 1 0 1	CMD	IDENT command at IPL 15
1 1 1 0	CMD	IDENT command at IPL 16
1 1 1 1	CMD	IDENT command at IPL 17

All data packets transmitted from TRF to the XMI bus, are also sent to REG as TRX_D[63:00] and loaded into failing address register XFADR and failing address extension register XFAER. In the event of an error interrupt, data stored in XFADR and XFAER is used for error analysis. TRX_D[63:00] is loaded into XFADR and XFAER by LDTRXDAT from TCM.

Transactions processed by the transmit logic and discussed in the following are as follows:

DMA read data return
DMA read lock status return
DMA read error status return
CPU read XJA private register C/A
CPU write XJA private register C/A + write data
CPU read XMI C/A (including XMI space registers in XJA)
CPU write XMI C/A + write data (including XMI space registers in XJA)
XMI read XMI space register data return
IDENT command to the XMI bus

In controlling the transmission of data packets to the XMI bus, the TCM communicates with the RCM because many transmissions to the XMI bus have associated actions required in the receive logic. For example, when the TCM has transmitted a CPU write packet from the TRF to the XMI bus, it must notify the RCM to force a write complete packet back to the ICU. The TCM communicates with the RCM using TCMSTATUS[02:00]. Table 4–12 lists the types of communications the TCM sends to the RCM.

Table 4–12 TCM Status Code

TCMSTATUS[02:00]	Informs RCM of
0 0 0	No-op
0 0 1	XJA private register read completed
0 1 0	ERRINTR IDENT received
0 1 1	XMI space register read completed
1 0 0	Read error
1 0 1	Write transaction completed
1 1 0	No-op
1 1 1	IPINTR IDENT received

4.7.2 DMA Read Data Return

Figure 4–21 is a flow diagram of the DMA read data return, read lock status return, and read error status return to the XMI bus.

LOADBUFSEL[01:00] from the TCM selects the TRF buffer to be loaded with the data packet. Buffer 1 is selected first as seen in Table 4–13.

Table 4–13 TRF Buffer Select Code

LOADBUFSEL[01:00] TRXBUFSEL[01:00]	Buffer Selected
0 0	1
0 1	2
1 0	3
1 1	Reserved

The XCE loads a CBI data packet into buffer 1 by asserting LOAD for the duration of the data packet. The first data longword (command/address) is decoded by the command decoder, which then informs the TCM of the type of data packet using JXDITYPE[02:00] (Table 4–10). The command/address longword is loaded into the first location of buffer 1. As the next two cycles of read data are received from the CBI, the buffer increments its address to load the data longwords into the next two locations. If the length of the transfer is octaword, the XCE keeps LOAD asserted to hold the buffer in the load mode, enabling it to increment through the next two locations loading in two more longwords. If the transfer length is hexword, four more longwords are loaded into the buffer filling it to capacity.

When the packet is completely loaded into the buffer, the XCE notifies the TCM of this by asserting CMDRCV. The TCM then:

- Increments LOADBUFSEL[01:00] to enable buffer 2 for the next data packet.
- Asserts TRXBUFSEL[01:00] to select the buffer to be unloaded to the assembly/select logic. Buffer 1 is selected as seen in Table 4–13.

The TCM checks to see if a register read request was received from the XMI bus by checking RCMSTATUS[00] from the RCM (asserted means a register read request is pending). If an XMI register read is pending, it has priority and executes first (Figure 4–23).

If no XMI register read is pending, the TCM arbitrates for the XMI bus by asserting XMI_RES_REQ. If the TCM does not receive an XMI bus grant before 256 XMI cycles pass (approximately 16.4 µs), it asserts XMIARBTO to REG, indicating that an arbitration timeout error has occurred. XMIARBTO sets a bit in the error summary register (ERRS).

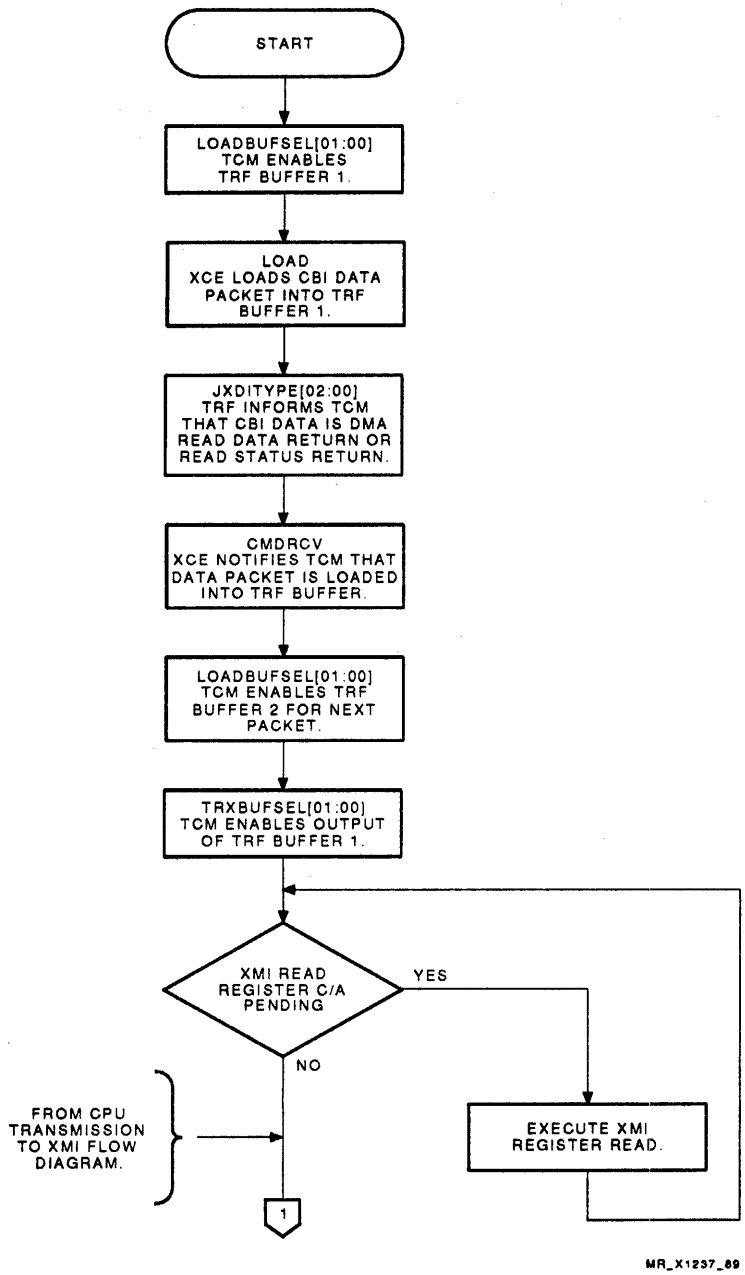


Figure 4–21 (Cont.) Transmission of DMA Read Data/Status Return to XMI Bus

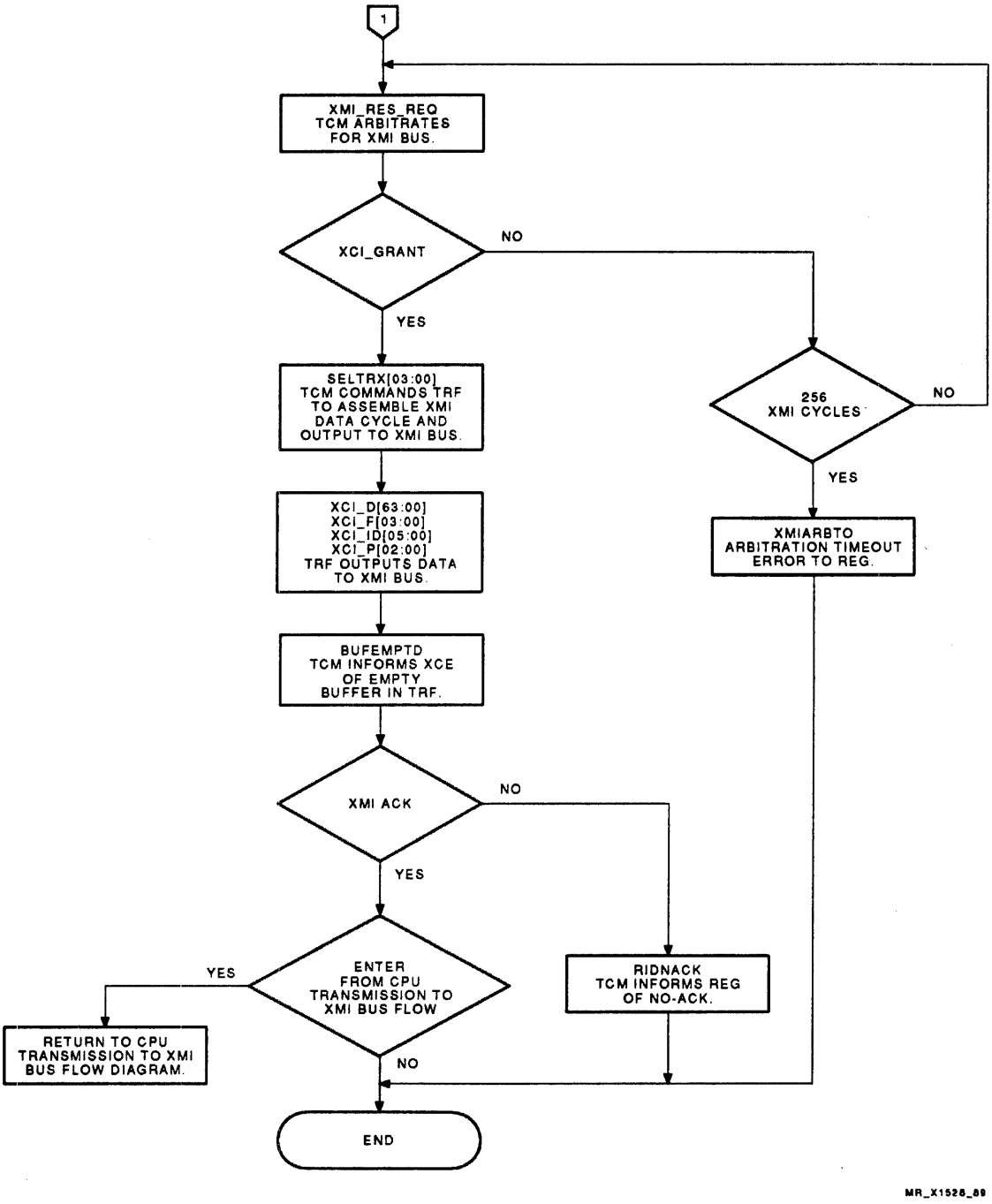


Figure 4–21 Transmission of DMA Read Data/Status Return to XMI Bus

If the TCM receives XCI_GRANT, it commands the select/assembly logic to generate a return read good data bus cycle (Figure 3–1) for the XMI bus. The TCM does this by asserting a SELTRX[03:00] code of 0000 (Table 4–11). The select/assembly logic:

- Selects the first quadword of data from buffer 1 and outputs it as D[63:00],
- Assembles and outputs a 4-bit function code (F[03:00]), specifying good return read data 0,
- Selects the commander ID from the C/A in buffer 1 and outputs it as ID[05:00].

Parity is generated on the function, ID, and data bits and added to the output as P[02:00]. The signals are synchronized with the XMI clock and output to the XMI bus.

For octaword and hexword read-data-return functions, the preceding is repeated with SELTRX[03:00] specifying the read-data-return cycle and the data quadwords being extracted from buffer 1. The function codes from the assembly/select logic (F[03:00]) specify good read data 1, good read data 2, and good read data 3 as required.

When the data packet is placed on the XMI bus, the TCM asserts BUFEMPTD to the XCE, informing it that a TRF buffer is emptied and can be used to accept another data packet from the CBI.

Each cycle of return-read-data transmitted to the XMI bus should receive an acknowledgment (XCI_RCNF). If the TCM does not receive an acknowledgment for a transmitted data cycle, it asserts RIDNACK to REG, which records the error in the XMI bus error register (XBER). The TCM does not retry the transmission.

A portion of Figure 4–21 is used as an extension of the flow diagram of Figure 4–22. This is discussed in Section 4.7.4.

4.7.3 Read Locked Response and Read Error Response

The read locked response and read error response transactions execute basically the same as a DMA read-data-return transaction (Figure 4–21).

The command/address decoder decodes the read locked (or read error) response from the CBI, and notifies the TCM of the cycle type via JXDITYPE[02:00]. The command/address is loaded into the enabled TRF buffer for the purpose of storing the commander ID. When the TCM gains access to the XMI bus, it asserts the read locked (or read error) code (Table 4–11) to the select/assembly logic by using SELTRX[03:00]. The select/assembly logic:

- Zeroes all the data bits (D[63:00])
- Outputs a 4-bit function code (F[03:00]), specifying a read locked (or read error) response (Figure 3–4 or Figure 3–3)
- Selects the commander ID from the C/A longword in buffer 1 and outputs it as ID[05:00]

Parity is generated on the function, ID, and data bits and added to the output as P[02:00]. The signals are synchronized with the XMI clock and output to the XMI bus.

When the locked response (or error response) is placed on the XMI bus, the TCM asserts BUFEMPTD to the XCE, informing it that a TRF buffer is emptied and can be used to accept another data packet from the CBI.

After the response data cycle is transmitted to the XMI bus, an acknowledgment should be received. If the TCM does not receive an acknowledgment (XCI_RCNF), it asserts RIDNACK to REG, which records the error in XBER. The TCM does not retry the transmission.

4.7.4 CPU Read/Write

Figure 4–22 is a flow diagram of a CPU read/write transmission to the XMI bus.

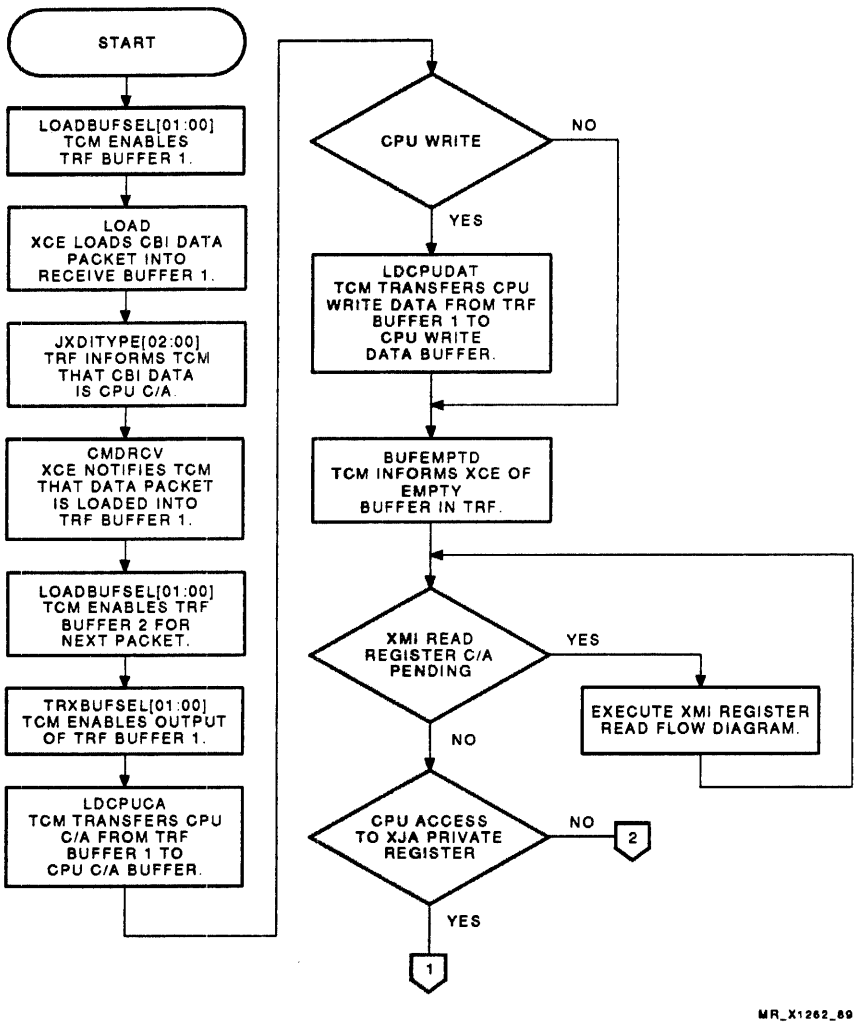


Figure 4–22 (Cont.) Transmission of CPU Read/Write Command/Address Packet to XMI Bus

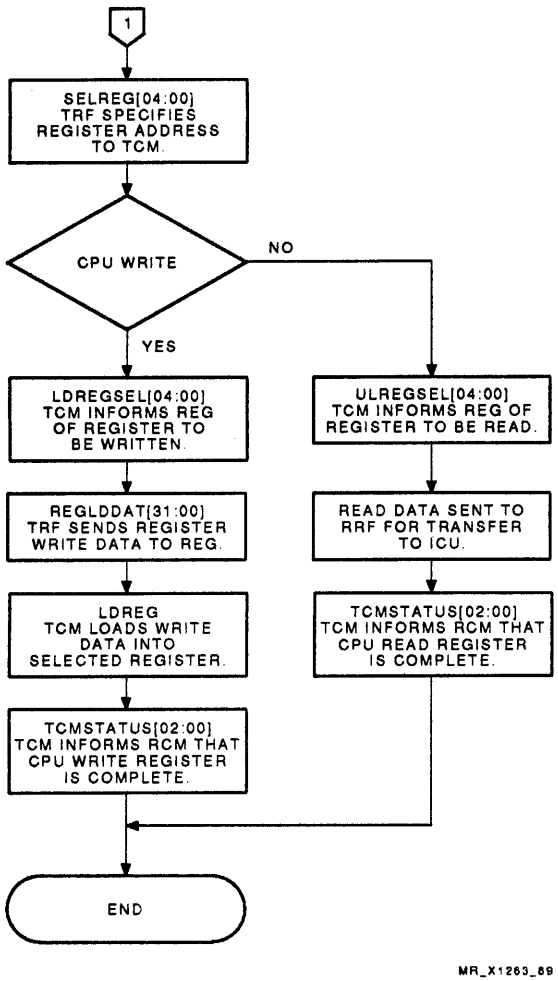


Figure 4–22 (Cont.) Transmission of CPU Read/Write Command/Address Packet to XMI Bus

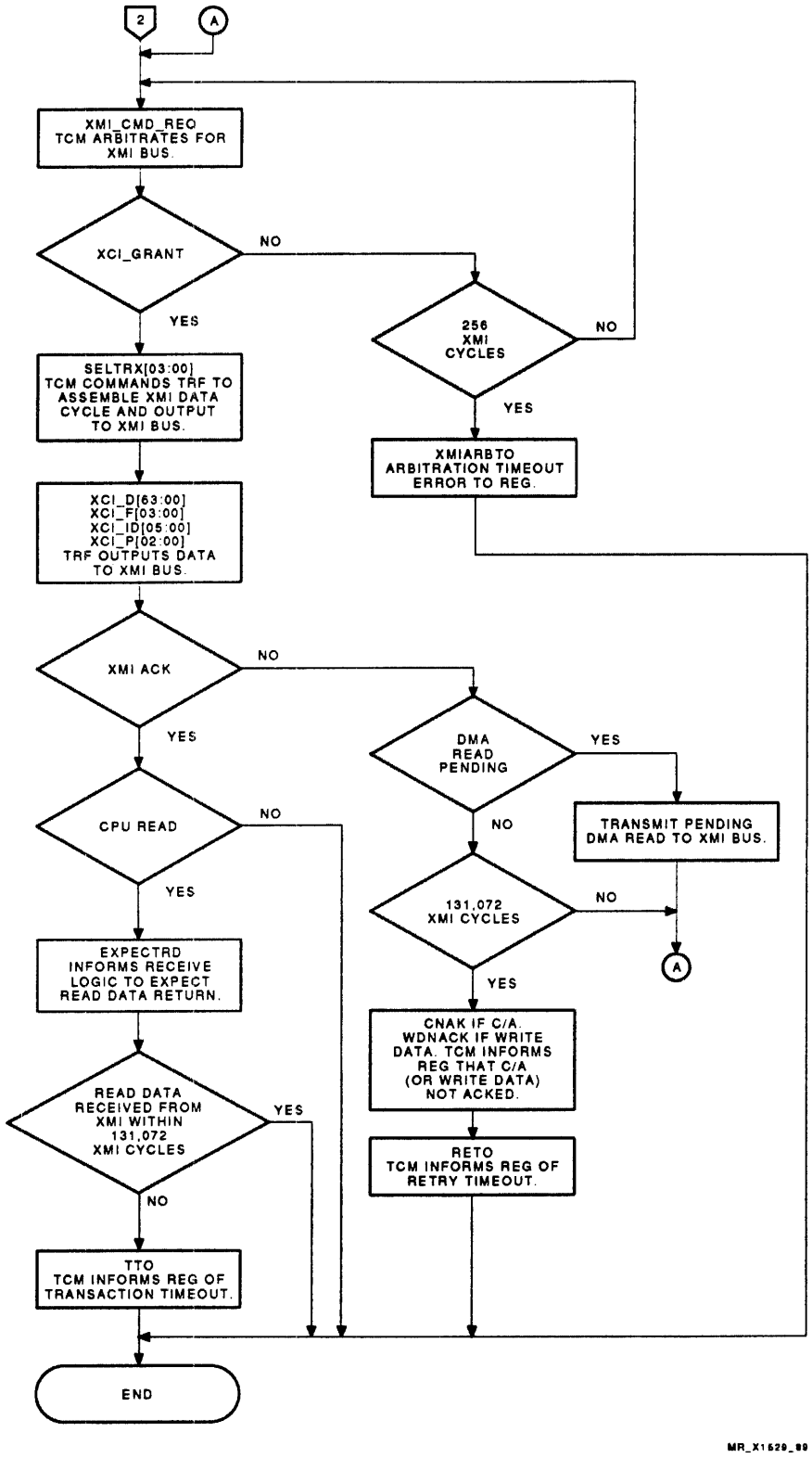


Figure 4–22 Transmission of CPU Read/Write Command/Address Packet to XMI Bus

LOADBUFSEL[01:00] from the TCM enables TRF buffer 1. The XCE loads a CBI data packet into buffer 1 by asserting LOAD. The first data longword (command/address) is decoded by the command decoder, which then informs the TCM of the type of data packet using JXDITYPE[02:00] (Table 4–10).

When the packet is completely loaded into the buffer, the XCE notifies the TCM of this by asserting CMDRCV. The TCM then:

- Increments LOADBUFSEL[01:00] to enable buffer 2 for the next data packet.
- Asserts TRXBUFSEL[01:00], which outputs the data in buffer 1 to the CPU command/address buffer and the CPU write buffer (if the transaction is a CPU write).
- Asserts LDCPUCA to load the CPU command/address from buffer 1 into the CPU command/address buffer.
- If the CPU transaction is a write, the TCM asserts LDCPUDAT to load the CPU write data from buffer 1 into the CPU write buffer. Transferring the CPU command/address and write data to their own buffers frees buffer 1 to accept DMA read data.

The TCM checks to see if a register read request was received from the XMI bus by looking at RCMSTATUS[00] from the RCM. If RCMSTATUS[00] is asserted, an XMI register read transaction is pending. An XMI register read transaction has priority and executes first (Figure 4–23).

The TCM determines if the CPU access is to an XJA private register or to the XMI bus, by looking at SELREG[04:00] from the CPU command/address buffer. If the CPU access is to an XJA private register, SELREG[04:00] specifies the register. If the CPU access is to the XMI bus, the SELREG[04:00] code is all 1s.

4.7.4.1 Access to XJA Private Register

If the CPU read/write is directed toward an XJA private register, the CPU command/address buffer specifies the selected register to the TCM, using SELREG[04:00].

If the CPU function is a write, the TCM asserts LDREGSEL[04:00] to REG to select the register to be written. The write data is transferred to REG from the CPU write buffer as REGLDDAT[31:00]. The TCM loads the write data by asserting LDREG to REG. The TCM informs the RCM that the write operation is complete using TCMSTATUS[02:00] (Table 4–12). The RCM then forces a write complete transaction in the RRF to be sent back to the ICU.

If the CPU function is a read, the TCM asserts ULREGSEL[04:00] to REG to select the register to be read. The read data is sent to the RRF for transmission back to the ICU.

The TCM uses TCMSTATUS[02:00] (Table 4–12) to inform the RCM that a CPU register read operation has occurred. The RCM then functions to force a read data return packet in the RRF. The packet (including the register data) is sent to the ICU.

4.7.4.2 Access to XMI Register

If the CPU read/write is directed toward the XMI bus, the CPU command/address buffer specifies this with a SELREG[04:00] code of all 1s.

The TCM arbitrates for the XMI bus by asserting XMI_CMD_REQ. If the TCM does not receive an XMI grant before 256 XMI cycles have passed (approximately 16.4 μs), it asserts XMIARBTO to REG, indicating that an arbitration timeout error occurred. XMIARBTO sets a bit in the error summary register (ERRS).

If the TCM receives XCI_GRANT, it commands the select/assembly logic to assemble a CPU cycle to the XMI bus by asserting a SELTRX[03:00] code of 1000 (Table 4–11). The select/assembly logic:

- Selects the command/address longword from the CPU C/A buffer and assembles the command longword for the XMI bus as shown in Figure 3–2 for a CPU read or Figure 3–7 for a CPU write. The command longword outputs the assembly/select multiplexer as D[63:00]. If the CPU C/A is directed to BI window space, address bits [28:25] from the CPU C/A buffer are sent to the REG as XBINUM[03:00] to specify which BI window is being addressed. REG responds with XBIID[03:00] which is the XMI node ID of the BI adapter associated with the selected BI window. The select/assembly logic places XBIID[03:00] into address bits [28:25] of the command/address longword.
- Outputs a 4-bit function code (F[03:00]), specifying a command cycle.
- Selects the XMI_NODE_ID[03:00] hardwired input as the commander (XJA) ID. The ID is output to the XMI bus as ID[05:00].

Parity is generated on the function, ID, and data bits and added to the output as P[02:00]. The signals are synchronized with the XMI clock and output to the XMI bus.

If the CPU function is a read, the TRF asserts EXPECTRD to the RCM to inform the receive logic to expect read data return from the XMI bus.

If the CPU function is a write to the XMI bus, the write data (REGLDDAT[31:00]) is transferred from the CPU write buffer to the select/assembly logic where it is selected for the next XMI cycle by the TCM (using SELTRX[03:00]). The select/assembly logic:

- Selects the longword of write data (REGLDAT[31:00]) from the CPU write buffer and outputs it in the lower 32 bits of D[63:00].
- Outputs a 4-bit function code (F[03:00]) specifying a write data cycle.
- Selects the commander ID as the hardwired XMI_NODE_ID[03:00], which specifies the XJA as the commander, and outputs it as ID[05:00].

Parity is generated on the function, ID, and data bits and is added to the output as P[02:00].

After the data cycle(s) is transmitted to the XMI bus, an acknowledgment should be received. If the TCM does not receive an acknowledgment (XCI_RCNF) for the command/address cycle, it checks for a pending DMA read transmission. If there is DMA read data in the TRF to be transmitted to the XMI bus, it has priority and executes (Figure 4–21) before the CPU transmission is retried. A check is made for a pending DMA read transmission before each retry of the CPU transmission. The TCM retries the CPU transmission for 131,072 XMI cycles (approximately 8.4 ms) after which it asserts CNACK (C/A no-ack) and RETO (retry timeout) to REG. If the TCM does not receive an acknowledgment for the write data cycle, it retries the transmission just as for the C/A cycle. If an acknowledgment is not received in 131,072 XMI cycles (approximately 8.4 ms), the TCM asserts WDNACK (write data no-ack) and RETO to REG. CNACK, WDNACK, and RETO set bits in the XMI bus error register (XBER).

In a CPU read XMI operation, the TCM is notified that the XJA received the read data by the assertion of KRESPONSE by the RCM. If the XJA receive logic does not receive the read data after 131,072 XMI bus cycles (approximately 8.4 ms), the TCM asserts TTO (transaction timeout) to REG. TTO sets a bit in the XMI bus error register (XBER).

4.7.5 Read Register Data Return

Figure 4–23 is a flow diagram of the read register data return function to the XMI bus.

RCM informs TCM (by asserting RCMSTATUS[00]) that the XJA receive logic received an XMI read register command from the XMI bus. The XRC identifies the selected register to the TCM using REGADR[03:00]. The TCM responds by asserting ULREGSEL[04:00] to REG, identifying the register to be read. REG unloads the read data from the selected register and transfers it to the TRF as REGULDAT[31:00].

The TCM arbitrates for the XMI bus by asserting XMI_RES_REQ. If the TCM does not receive an XMI grant before 256 XMI cycles has passed (approximately 16.4 µs), it asserts XMIARBTO to REG, indicating that an arbitration timeout occurred. XMIARBTO sets a bit in the error summary register (ERRS).

When the TCM receives XCI_GRANT, it commands the select/assembly logic to assemble a read good data return bus cycle for the XMI bus. It does this by asserting a SELTRX[03:00] code of 0000 (Table 4–11). The select/assembly logic:

- Selects the longword of register read data (REGULDAT[31:00]) from REG and outputs it in the lower 32 bits of D[63:00]
- Outputs a 4-bit function code (F[03:00]) specifying cycle 0 of good read return data.
- Selects the commander ID that was received from the XRC as CMDR_ID[05:00] when the XRC received the read request from the XMI bus. The commander ID was stored in the TRF select/assembly logic for use when the read data is transmitted to the XMI bus.

Parity is generated on the function, ID, and data bits and is added to the output as P[02:00].

After the read data cycle is transmitted to the XMI bus, an acknowledgment should be received. If the TCM does not receive an acknowledgment (XCI_RCNF), it asserts RIDNACK to REG, which records the error. RIDNACK sets a bit in the XMI bus error register (XBER). The TCM does not retry the transmission.

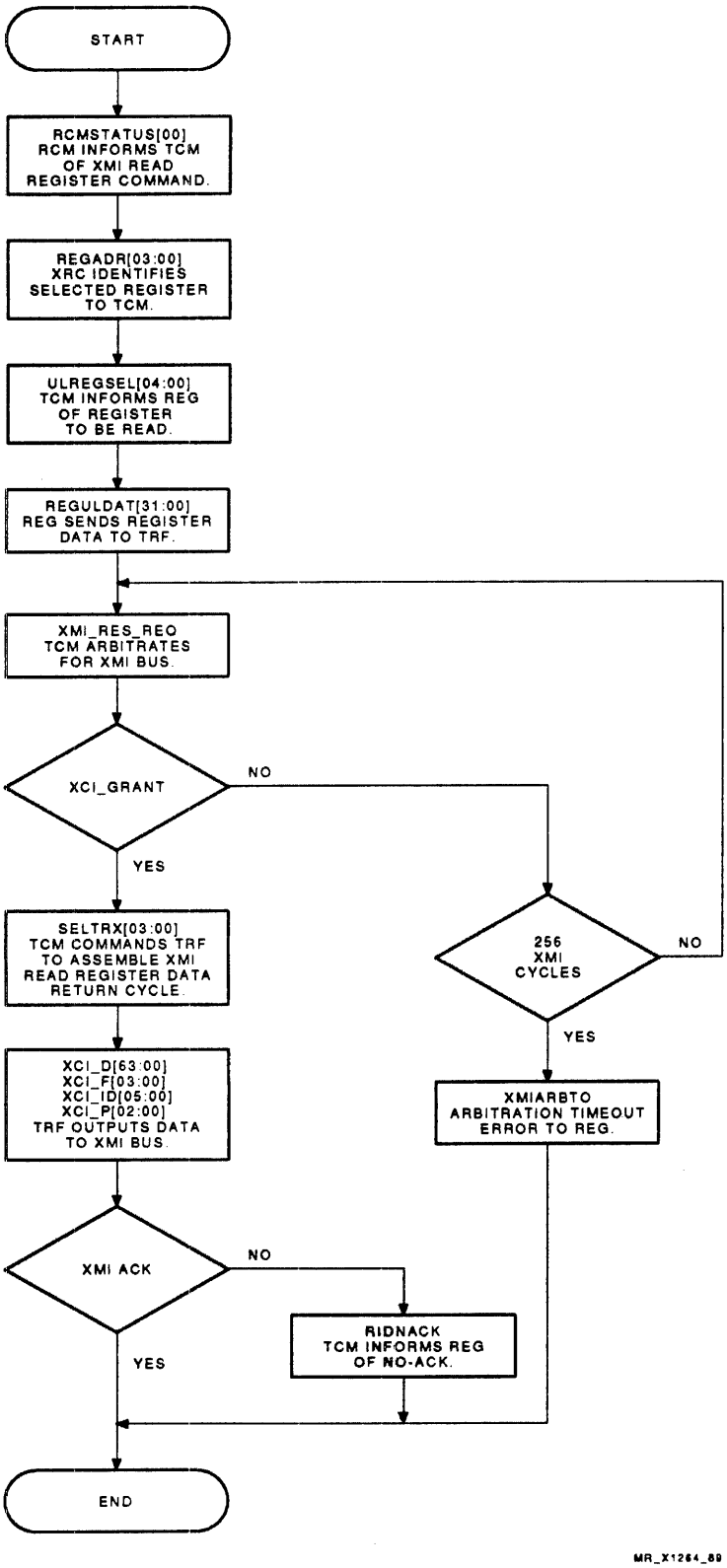


Figure 4–23 Transmission of Read Register Data Return to XMI Bus

4.7.6 IDENT

The RCM informs the TCM of pending interrupts and their IPL using INTRPEND[03:00]. The TCM initiates an IDENT function by asserting a SELTRX[03:00] code to the select/assembly logic, specifying an IDENT command cycle to the XMI at the specified IPL. The select/assembly logic:

- Assembles an IDENT command cycle (Figure 3–8) at the specified IPL. The logic uses IDENTID[15:00] from the RCM as the interrupt source mask in the command longword. The RCM saved the node ID of the XMI interrupting node for this purpose. The assembled IDENT command is output from the select/assembly logic as D[63:00].
- Outputs a 4-bit function code (F[03:00]), specifying a command cycle.
- Selects the commander ID as the hardwired XMI_NODE_ID[03:00], which specifies the XJA as the commander, and outputs it as ID[05:00].

Parity is generated on the function, ID, and data bits and is added to the output as P[02:00].

After the IDENT command cycle is transmitted to the XMI bus, an acknowledgment is received. If the TCM does not receive an acknowledgment (XCI_RCNF), it asserts CNAK to REG and retries the transmission for 131,072 bus cycles (approximately 8.4 ms) after which it asserts RETO (retry timeout) to REG. CNAK and RETO set bits in the XMI bus error register (XBER).

4.8 XJA Registers (REG)

The XJA registers Table 4–14 are used for error reporting and diagnostic functions. All XJA registers ignore masking information on writes. Masked writes to these registers are treated as longword writes. All XJA registers ignore locking information on read locks and write unlocks. No logical locking mechanism is set and these transactions complete as if they were generic reads and writes.

Registers in the XJA can be broken down into two main groups: XMI space registers and XJA private registers.

4.8.1 XMI Space Registers

The XMI space registers (Table 4–14) are those registers that are accessed from the XMI bus. When the system CPU accesses XMI space registers, it places its read/write request on the XMI bus with the XJA as the target node and the location of the XMI space register as the specified address. When running system diagnostics, it is possible to access the XMI space registers directly without going out onto the XMI bus. This feature is useful when the XMI bus is faulty and the system needs the information in the XMI space registers.

4.8.2 XJA Private Registers

The XJA private registers (Table 4–14) contain information that is specific to the XJA. The private registers are only accessible by the system CPU directly in the XJA. They are not accessible from the XMI bus.

Table 4–14 XJA Registers

Register	Mnemonic
XMI Space Registers	
Device	XDEV
Bus error	XBER
Failing address	XFADR
Failing address extension	XFAER
XJA general purpose	XJAGPR
Full system emulation mode control	FAEMC
Add-on self-test status	AOSTS
XJA serial number	SERNUM
XJA Private Registers	
Error summary	ERRS
Force command	FCMD
Interprocessor interrupt source	IPINTRSRC
XJA diagnostic control	DIAG
DMA failing address	DMAFADDR
DMA failing command	DMAFCMD
Error interrupt control	ERRINTR
Configuration	CNF
XBI ID A	XBIIDA
XBI ID B	XBIIDB
Error SCB offset	ERRSCB
SCB offset IPL 14	IDENT4
SCB offset IPL 15	IDENT5
SCB offset IPL 16	IDENT6
SCB offset IPL 17	IDENT7

4.8.3 REG Data Flow

Figure 4–24 shows the XJA registers divided into two groups: those registers loaded from the XMI bus and those not loaded from the XMI bus. The registers loaded from the XMI bus include the eight XMI space registers plus the IPINTRSRC, DMAFCMD, and DMAFADR registers. Although the IPINTRSRC, DMAFCMD, and DMAFADR registers are loaded with XMI data, control of these registers is done from the XJA. They are not accessible from the XMI; that is, an XMI device cannot read or write these three registers.

Figure 4–24 illustrates the various data paths in and out of REG and the loading and unloading controls. These are described in the remainder of this section. A complete description of the registers, their bit fields, and a detailed description of each bit, is provided in Chapter 5.

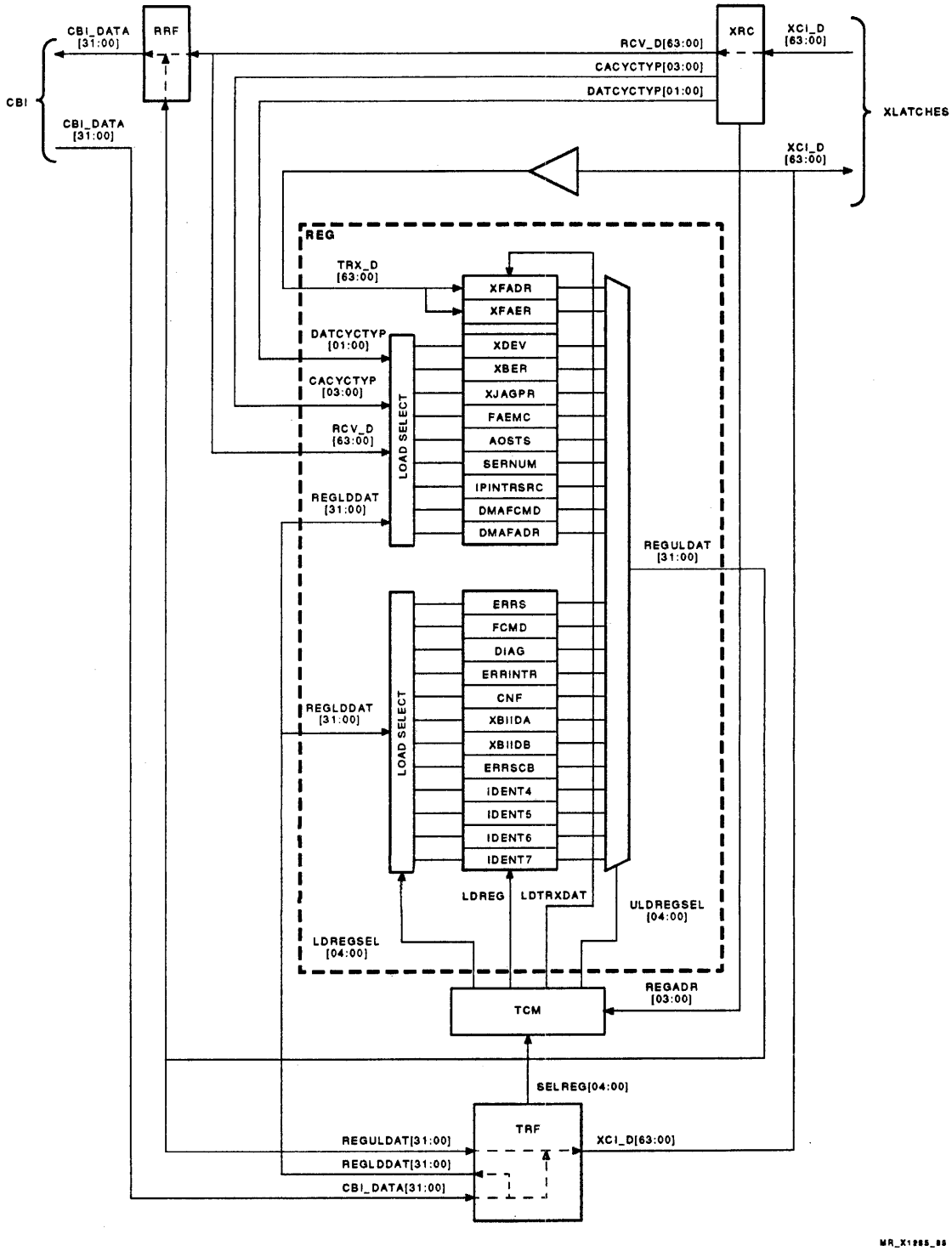


Figure 4–24 REG Data Flow

Reading all the XJA registers is controlled by the TCM. ULDREGSEL[04:00] from the TCM, selects a register and unloads the data onto the REGULDAT[31:00] output lines. Writing the lower 12 registers in Figure 4–24 (those not loaded from the XMI bus) is also controlled by the TCM. LDREGSEL[04:00] from the TCM, selects the register and LDREG loads it with input data.

Writing the upper two registers in Figure 4–24 is controlled by the TCM, which asserts LDTRXDAT each time a command/address is transmitted to the XMI bus. LDTRXDAT loads the command/address into the two registers (XFADR and XFAER). Two 32-bit registers are required to receive the 64-bit command/address.

Writing the nine upper registers remaining in Figure 4–24 is controlled by REG. REG uses CACYCTYP[03:00] and DATCYCTPY[01:00] from XRC to identify the RCV_D[63:00] data cycles from the XRC. When the REG load-select logic detects a write register command/address, it uses the address to select the register to be written. It loads the selected register with write data in the write data cycle that follows.

4.8.3.1 XMI Space Register Read

The XMI read command/address comes from the XLATCHES in the XMI corner as XCI_D[63:00] and enters the XRC. The register address is extracted from the command/address and sent to the TCM as REGADR[03:00]. TCM outputs the register address as ULDREGSEL[04:00]. The read data outputs REG as REGULDAT[31:00], which is sent to the TRF. The TRF outputs the XMI read data to the XLATCHES as the low-order 32 bits of XCI_D[63:00].

4.8.3.2 XMI Space Register Write

The XMI write command/address, followed by the write data, comes from the XLATCHES in the XMI corner as XCI_D[63:00] and enters the XRC. From the XRC, the command/address data is sent to REG along with CACYCTYP[03:00], which identifies the data as the command/address. REG uses the command/address data to select the register to be written with the write data that follows in the next cycle. The write data outputs XRC as the low-order 32 bits of RCV_D[63:00]. RCV_D[63:00] is sent to REG where the low-order bits (RCV_D[31:00]) are loaded into the selected register.

4.8.3.3 CPU Register Read of XJA Private Register

The CPU read command/address comes off the CBI as CBI_DATA[31:00] and enters the TRF. The register address is extracted from the command/address and sent to the TCM as SELREG[04:00]. TCM outputs the register address as ULDREGSEL[04:00]. The read data outputs REG as REGULDAT[31:00], which is sent to the RRF. The RRF outputs the CPU read data onto the CBI as CBI_DATA[31:00].

4.8.3.4 CPU Register Write of XJA Private Register

The CPU write command/address, followed by the write data, comes off the CBI as CBI_DATA[31:00] and enters the TRF. The register address is extracted from the command/address and sent to the TCM as SELREG[04:00]. TCM outputs the register address as LDREGSEL[04:00]. The write data outputs TRF as REGLDDAT[31:00], which is sent to REG and loaded into the selected register by LDREG.

4.8.3.5 XMI Failing Address Register

Another data input to REG is TRX_D[63:00]. TRX_D[63:00] is the command/address being sent to the XMI bus. TRX_D[63:00] is loaded into the XMI failing address register XFADR and failing address extension register XFAER where they can be accessed for error analysis if the XMI transaction fails. LDTRXDAT from TCM loads the data into XFADR and XFAER on each command/address transmission to the XMI bus.

4.8.3.6 DMA Failing Command/Address Registers

A DMA failing command register (DMAFCMD) and DMA failing address register (DMAFADR) are loaded with the command/address data of each DMA packet received from the XMI bus. The REG load-select logic uses CACYCTYP[03:00] from XRC to recognize the RCV_D[63:00] data as being a DMA command/address cycle, and loads the data into DMAFCMD and DMAFADR. Two 32-bit registers are required to receive the 64-bit command/address. The register data is accessed for error analysis in the event the DMA transaction fails.

4.9 Interrupts

There are two basic types of interrupts sent to the system CPU on the JXDI: nonfatal and fatal.

A nonfatal interrupt appears as an interrupt command on the JXDI data lines (XJA_DAT[15:00]). The interrupt command contains the interrupt priority level (IPL) of the interrupt, which can be 14(hex), 15(hex), 16(hex), or 17(hex). During a nonfatal interrupt, operation of the XJA is predictable. The XJA is still capable of responding to CPU requests but transactions can fail.

A fatal interrupt asserts the XJA_FATALERR line on the JXDI. During a fatal interrupt, operation of the XJA is unpredictable. The XJA may not be able to respond to CPU requests.

4.9.1 Nonfatal Interrupts

In nonfatal interrupts, the CPU is interrupted at the IPL contained in the JXDI interrupt command. The CPU responds by initiating a CPU read of the associated IDENT register in the XJA, to obtain the SCB offset (vector). The XJA obtains the vector by various means, according to the source of the interrupt as shown in Table 4–15.

Table 4–15 Nonfatal Interrupts

Source of Interrupt	IPL (Hex)	XJA Gets Vector by
XMI normal interrupt	14–17	Issuing an IDENT
XMI IPINTR	16	Supplying 80(hex) to ICU
Nonfatal error detected by the XJA	17	Unloading contents of ERRSCB to ICU

4.9.1.1 XMI Normal Interrupt

Normal interrupts from the XMI bus are at an IPL of 14(hex), 15(hex), 16(hex), or 17(hex). The interrupt is passed onto the JXDI at the same IPL. The CPU reads the corresponding IDENT register to get the offset vector. The IDENT registers do not physically exist in the XJA. When the CPU reads the pseudo IDENT register, the XJA issues an IDENT command to the interrupting node on the XMI bus. The data returned in the IDENT response is the offset vector. This is passed on to the CPU as a read-data-return transfer.

4.9.1.2 XMI IPINTR (Interprocessor Implied Vector Interrupt)

NOTE

IPINTR interrupts do not occur in the VAX 9000 system as there are currently no CPUs on the XMI bus.

IPINTR interrupts received from the XMI bus cause the XJA to place an interrupt command on the JXDI at an IPL of 16(hex). The CPU responds to the interrupt by reading the IDENT6 register. When the IDENT6 register is read with an IPINTR pending, the XJA returns an interrupt vector of 80(hex) to the CPU.

4.9.1.3 Nonfatal Errors Detected by XJA

Nonfatal errors detected by the XJA, interrupt the CPU by asserting an interrupt command on the JXDI at an IPL of 17(hex). The CPU responds to the interrupt by reading the IDENT7 register. When the IDENT7 register is read with an XJA error interrupt pending, the XJA returns the contents of ERRSCB to the CPU as a read-data-return transfer.

The nonfatal errors detected by the XJA and where they were detected, are shown in Table 4–16.

Table 4–16 Nonfatal Errors Detected by the XJA

Error Detected on	Error	Register/Bit	IPL (Hex)
XMI	Corrected confirmation code	XBER[27]	17
XMI	Parity error	XBER[23]	17
XMI	Corrected read data	XBER[19]	17
XMI	Write sequence error	XBER[22]	17
XMI	Read/IDENT data no-ack	XBER[21]	17
XMI	Write data no-ack	XBER[20]	17
XMI	Reattempt timeout	XBER[18]	17
XMI	Read sequence error	XBER[17]	17
XMI	Read error response	XBER[16]	17
XMI	Command no-ack	XBER[15]	17
XMI	Transaction timeout	XBER[13]	17
XMI	XMI power-up	ERRS[20]	17
XMI	XMI arbitration timeout	ERRS[22]	17
JXDI	Parity error	ERRS[31:30]	17
XJA internal	None	–	–

4.9.2 Fatal Interrupts

Fatal errors detected by the XJA interrupt the CPU by asserting XJA_FATALERR on the JXDI. The SCU interrupts the CPU at an IPL of 1D(hex). The CPU responds by executing a fatal error routine.

The fatal errors detected by the XJA and where they were detected are shown in Table 4–17.

Table 4–17 Fatal Errors Detected by the XJA

Error Detected on	Error	Register/Bit	IPL (Hex)
XMI	Node reset	XBER[30]	1D
XMI	Node halt	XBER[29]	1D
XMI	XMI fault	XBER[26]	1D
XMI	WEI (write error interrupt)	XBER[25]	1D
XMI	XMI power fail (AC or DC LO)	ERRS[21]	1D
JXDI	XCE transmit entry machine error	ERRS[28]	1D
JXDI	ICU buffer count error	ERRS[27]	1D
JXDI	CPU request overrun	ERRS[26]	1D
JXDI	JXDI receive buffer overrun	ERRS[25]	1D
JXDI	JXDI received command error	ERRS[24]	1D
XJA internal	Received CBI parity error	ERRS[23]	1D

4.10 Add-On Self-Test

The add-on self-test (AOST) function provides a way of testing the XJA logic and of checking reliability of all the XJA internal buses. It does this by reading and writing the XJA registers and then checking the results. AOST does not completely check all aspects of the XJA. That is, an XJA that passes AOST is basically functional, but error logic and interaction of the XJA with the CPU is not checked. AOST runs during power-up, console reset, or XMI node reset.

4.10.1 AOST Implementation

Figure 4–25 is a block diagram of the AOST implementation. The AOST logic consists of an 8096 microcontroller, an electrically erasable programmable read-only memory (EEPROM), and an address latch. The EEPROM stores the self-test microcode (EWCLD) used to run the 8096 microcontroller. The address latch is used by the 8096 to extract instructions from the EEPROM. A serial port in the 8096 allows for optional external control from a test terminal. The port is used to input test commands and output test results to the terminal.

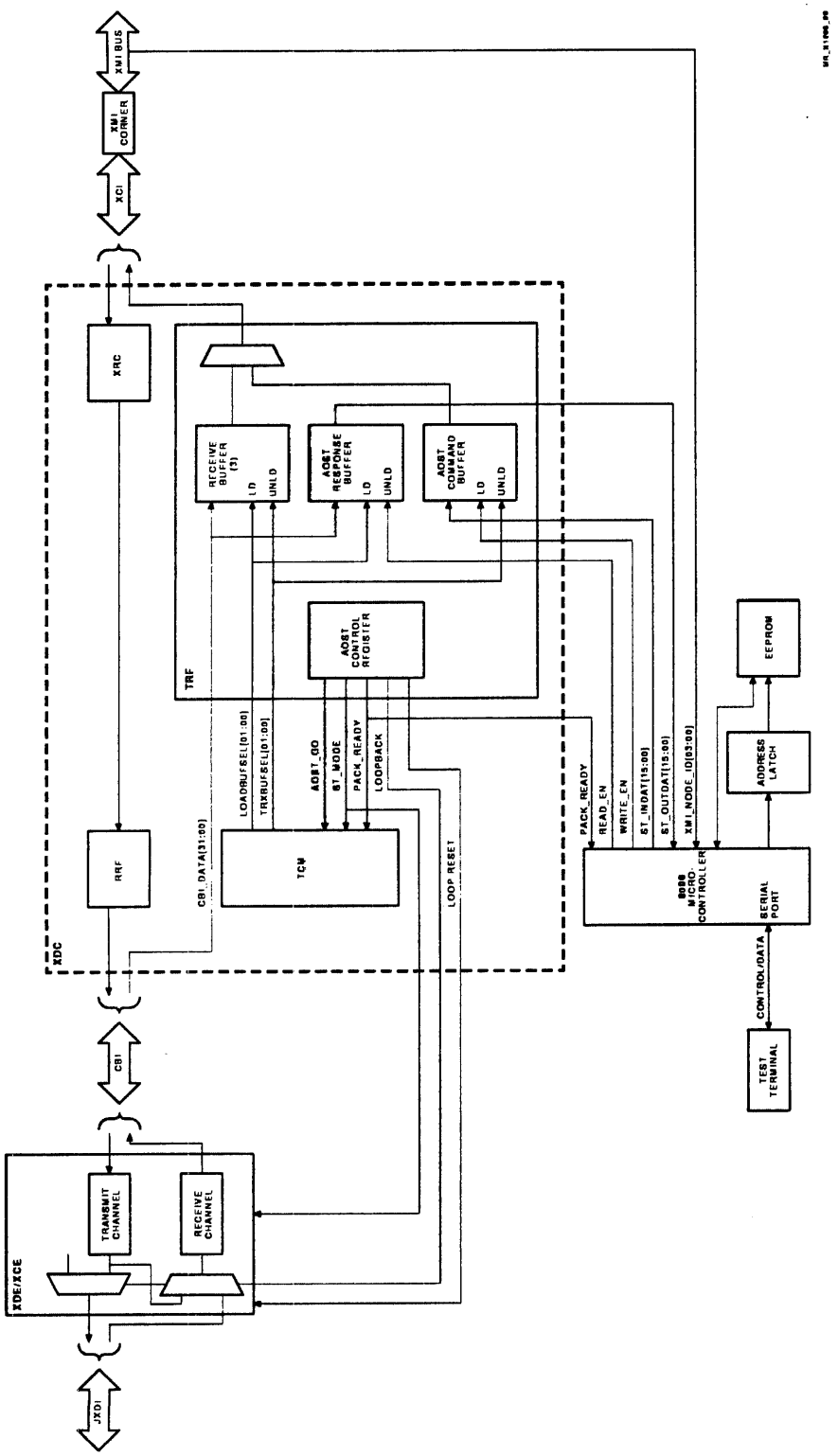


Figure 4–25 Add-On Self-Test Implementation

Within the TRF is an AOST control register, an AOST octaword read-only response buffer, and an AOST octaword write-only command buffer. The control register and the two buffers are associated with the AOST logic. The 8096 microcontroller maps to the control register and the two buffers as shown in Table 4–18.

Table 4–18 8096 Memory Map

Item	Address (Hex)	Contents
Control register	FFDE	AOST control bits
Response buffer (read only)	FFE0	Word 0 (CBI longword 0)
	FFE2	Word 1 (CBI longword 0)
	FFE4	Word 2 (CBI longword 1)
	FFE6	Word 3 (CBI longword 1)
	FFE8	Word 4 (CBI longword 2)
	FFEA	Word 5 (CBI longword 2)
	FFEC	Word 6 (CBI longword 3)
	FFEE	Word 7 (CBI longword 3)
Command buffer (write only)	FFF0	Word 0 (CBI longword 0)
	FFF2	Word 1 (CBI longword 0)
	FFF4	Word 2 (CBI longword 1)
	FFF6	Word 3 (CBI longword 1)
	FFF8	Word 4 (CBI longword 2)
	FFFA	Word 5 (CBI longword 2)
	FFFC	Word 6 (CBI longword 3)
	FFFE	Word 7 (CBI longword 3)

The EWCLD microcode simulates CPU transactions by writing the command buffer with a read or write test command, signaling the XJA to process the test command, and checking for the correct results in the response buffer. The microcode implements these functions by means of the 8096 microcontroller. Figure 4–26 is a flow diagram of AOST transactions. Refer to it in the following discussion.

A self-test mode bit is set in the AOST control register, asserting ST_MODE to the TCM and the XDE/XCE chips. A loopback bit is also set in the control register, asserting LOOPBACK to the XDE/XCE to place the XDEs into the loopback mode required for a self-test transaction. In loopback mode, the output of the XDE transmit channel does not go out to the JXDI, but loops back to a multiplexer where it is selected as the input to the XDE receive channel.

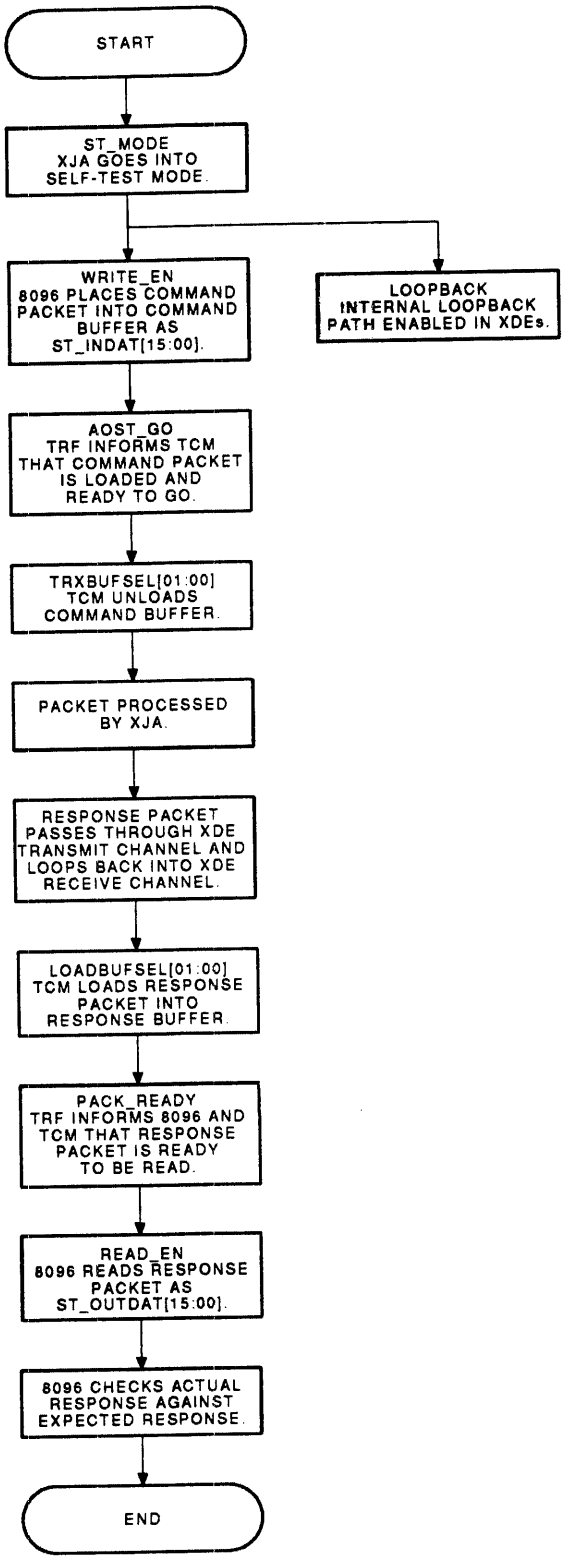


Figure 4–26 Flow Diagram of Add-On Self-Test

The 8096 asserts WRITE_EN to the AOST command buffer to load the command packet on the ST_INDAT[15:00] data lines. The microcode uses the node ID of the XJA in the generation of the address of the XMI space registers. The node ID (XMI_NODE_ID[03:00]) is applied to the 8096 microcontroller from the XMI backplane. When the command packet is loaded, the control register asserts AOST_GO, commanding the TCM to start the test.

The TCM unloads the command buffer using TRXBUFSEL[01:00] and the XJA proceeds to process the test packet as a normal command packet. If the test command is to an XMI space register, the packet goes out to the XMI bus and back in through the XRC. If the test command is to an XJA private register, the packet processes within the XJA.

In either case, a response packet leaves the RRF, passes through the XDE transmit channel, and loops back into the XDE receive channel. From the receive channel, the packet traverses the CBI and enters the TRF where it is loaded into the AOST response buffer by LOADBUFSEL[01:00] from the TCM.

When the response packet is loaded into the response buffer, the AOST control register asserts PACK_READY to the TCM and the 8096. The 8096 then asserts READ_EN to unload the response packet from the response buffer. The response packet is unloaded on the ST_OUTDAT[15:00] data lines. The microcode then checks the response packet and reports the results.

The bits of the AOST control register are shown in Figure 4–27 and described in Table 4–19.

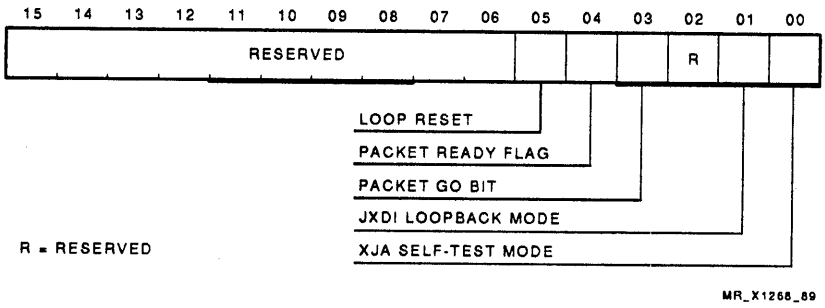


Figure 4–27 AOST Control Register

Table 4–19 AOST Control Register

Bit	Name	Type	Initial State[1]	Description
15–06	–	–	–	Reserved.
05	Loop reset	R/W	0	When set, resets the XCE and XDE gate arrays. This bit must be set each time JXDI loopback mode or self-test mode is changed, to ensure correct timing when the loopback multiplexers are switched.
04	Packet ready flag	R/W 1 to clear	0	When set, indicates that a response packet is loaded into the response buffer and can be read by the AOST firmware. In an AOST XJA register read transaction, the packet ready flag is set when the read data return loops back into the TRF response buffer. In an AOST XJA register write transaction, the packet ready flag is set when the write complete packet loops back into the TRF response buffer. The packet ready flag must be cleared after each read of the XJA response buffer. **NOTE** **When the AOST test is a write of the XJA's force command register, the write complete packet is overwritten in the response buffer by the forced command packet since both loop back into the buffer. The packet ready flag is set only after the forced command packet is loaded into the buffer.**
03	Packet go bit	R/W	0	When set, initiates an AOST test transaction by commanding the TCM to start processing the test packet in the AOST command buffer. This bit is self-clearing.
02	–	–	–	Reserved.
01	JXDI loopback mode	R/W	0	When set, selects internal loopback within the XCE and XDEs. During internal loopback, data passing through the XDE transmit channel and control signals in the XCE transmit logic, are looped back into the XDE receive channel and XCE receive logic, respectively. Signals sent to the XJA on the JXDI are not selected by the XDE loopback multiplexer. The XDEs and XCE allow packet sizes up to an octaword in length to loop back. This translates to a maximum of 8 JXDI word-length cycles. If this bit is cleared, the JXDI operates normally.
00	XJA self-test mode	R/W	0	When set, indicates that the XJA is in AOST mode. This bit must be set for AOST functions to work.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

4.10.2 XJA Register Read Transaction

A CPU read request on the CBI bus has the format shown in part A of Figure 4–28. To match this format, the AOST command buffer must be loaded in the format shown in part B of Figure 4–28.

CBI CYCLE	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
0	A [29:26, 05:02]	R R ID	A [13:06]	R R R R CMD
1	DON'T CARE	M [03:00] A [25:22]	DON'T CARE	A [21:14]

R = RESERVED

A. FORMAT ON CBI

ADDRESS (HEX)	CONTENTS BY BYTE	
FFF0	A [13:06]	, CMD
FFF2	A [29:06, 05:02]	, ID
FFF4	0	, A [21:14]
FFF6	0	, M [03:00] - A [25:22]
FFF8	–	, –
FFFA	–	, –
FFFC	–	, –
FFFE	–	, –

B. FORMAT IN AOST COMMAND BUFFER

MR_X1269_89

Figure 4–28 Format of CPU Read Request

Table 4–20 lists the command codes as they appear on the CBI.

Table 4–20 CBI Command Codes

CBI_DATA[03:00]	Command
0 0 0 0	Read request
0 0 0 1	Read lock request
0 0 1 0	Read data return
0 0 1 1	Read lock data return
0 1 0 0	Write request
0 1 0 1	Write unlock request
0 1 1 0	No-op
0 1 1 1	No-op
1 0 0 0	Interrupt request
1 0 0 1	Read locked status
1 0 1 0	Read error status
1 0 1 1	Write complete
1 1 0 0	No-op
1 1 0 1	No-op
1 1 1 0	No-op
1 1 1 1	No-op

When the read data loops back into the TRF response buffer, the packet ready flag is set in the AOST control register and the response buffer may be unloaded and checked for the correct response.

The format of a CPU read data return packet on the CBI is shown in part A of Figure 4–29. A packet in this format loads into the response buffer in the format shown in part B of Figure 4–29.

An XJA register contains a longword of data, therefore the response packet contains only four bytes of read data. After this data has been read and compared with the expected data, the packet ready flag must be cleared before another test transaction may be started.

CBI CYCLE	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
0	RESERVED	R R ID	RESERVED	R R R R CMD
1	RESERVED	RESERVED	RESERVED	RESERVED
2	RESERVED	RESERVED	BYTE 1	BYTE 0
3	RESERVED	RESERVED	BYTE 3	BYTE 2

R = RESERVED

A. FORMAT ON CBI

ADDRESS (HEX)	CONTENTS BY BYTE
FFE0	0 , COMMAND
FFE2	0 , ID
FFE4	0 , 0
FFE6	0 , 0
FFE8	BYTE 1 , BYTE 0
FFEA	0 , 0
FFEC	BYTE 3 , BYTE 2
FFEE	0 , 0

B. FORMAT IN AOST RESPONSE BUFFER

MR_X1270_89

Figure 4–29 Format of CPU Read Data Return

4.10.3 XJA Register Write Transaction

A CPU write request (plus the write data) packet on the CBI has the format shown in part A of Figure 4–30. To match this format, the AOST command buffer must be loaded in the format shown in part B of Figure 4–30. Table 4–20 lists the command codes as they appear on the CBI.

CBI CYCLE	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
0	A[29:26, 05:02]	R R ID	A[13:06]	R R R R CMD
1	0 0 0 0 0 0 0 0	M [03:00] A [25:22]	BYTE 0	A [21:14]
2	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	BYTE 2	BYTE 1
3	DON'T CARE	0 0 0 0 0 0 0 0	DON'T CARE	BYTE 3

R = RESERVED

A. FORMAT ON CBI

ADDRESS (HEX)	CONTENTS BY BYTE
FFF0	A [13:06] , COMMAND
FFF2	A [29:06, 05:02] , ID
FFF4	BYTE 0 , A [21:14]
FFF6	0 , M [03:00] - A [25:22]
FFF8	BYTE 2 , BYTE 1
FFFA	0 , 0
FFFC	0 , BYTE 3
FFFE	0 , 0

B. FORMAT IN AOST RESPONSE BUFFER

MR_X1271_89

Figure 4–30 Format of CPU Write Request

The response to a CPU write request packet is a write complete packet. When the write complete packet loops back into the TRF response buffer, the packet ready flag is set and the response buffer may be unloaded and checked for the correct response. The format of a CPU write complete packet on the CBI is shown in part A of Figure 4–31. A packet in this format loads into the response buffer in the format shown in part B of Figure 4–31.

After this data is read and compared with the expected data format, the packet ready flag must be cleared before another transaction can be started.

For writes to the XJA's force command register, the write complete packet in the response buffer is overwritten by the forced command.

CBI CYCLE	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16			15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00				
0	DON'T CARE	R	R	ID	DON'T CARE	R	R	R	R	CMD

R = RESERVED

A. FORMAT ON CBI

ADDRESS (HEX)	CONTENTS BY BYTE
FFE0	0 , CMD
FFE2	0 , ID
FFE4	– , –
FFE6	– , –
FFE8	– , –
FFEA	– , –
FFEC	– , –
FFEE	– , –

B. FORMAT IN AOST COMMAND BUFFER

MR_X1272_89

Figure 4–31 Format of CPU Write Complete

5
Register Descriptions

This chapter describes the purpose and the contents of the XJA registers. The register fields are defined, and the action that results from setting the register bits is described. Loading and unloading of the registers, and the associated data paths, are described in Chapter 4.

5.1 XJA Register Overview

The XJA has 23 registers. Eight are XMI space registers (normally accessed from the XMI bus) and fifteen are XJA private registers (cannot be accessed from the XMI bus). Table 5–1 lists the registers and briefly describes their function.

All XJA registers ignore masking information on writes. Masked writes to these registers are treated as longword writes. They ignore locking information on read locks and write unlocks. Read locks and write unlocks execute as basic reads and writes.

Registers in the XJA are divided into two groups: XMI space registers and XJA private registers. The XMI space registers are accessible from the XMI bus and directly within the XJA. The system CPU can access the XMI space registers either directly within the XJA or by executing a transaction on the XMI bus with the target node being the XJA. The method used during normal operation is by using an XMI transaction. Direct access is done in diagnostic mode when the XMI bus may be faulty.

The XJA private registers are accessible only from the system CPU. (See Chapter 1 for I/O space addressing.) They are always accessed directly within the XJA.

Table 5–1 XJA Registers

Register	Mnemonic	Description
XMI Space Registers		
Device	XDEV	Describes the node device.
Bus error	XBER	Contains a summary of the XMI status and errors.
Failing address	XFADR	Saves the low-order four bytes of a failing XMI command/address.
Failing address extension	XFAER	Saves the high-order four bytes of a failing XMI command/address.
XJA general purpose	XJAGPR	Used for diagnostic testing.
Full system emulation mode control	FAEMC	Controls XJA operation in full system emulation mode.
Add-on self-test status	AOSTS	Controls the XJA self-test.
XJA serial number	SERNUM	Contains the manufactuing plant, year and week of manufacture, and serial number of the XJA.
XJA Private Registers		
Error summary	ERRS	Contains a summary of errors detected by the XJA.
Force command	FCMD	Forces XJA transactions for testing purposes.
Interprocessor interrupt source	IPINTRSRC	Identifies the source of interprocessor interrupts.
XJA diagnostic control	DIAG	Controls diagnostic testing of the XJA.
DMA failing address	DMAFADDR	Saves address and length information of a failing DMA or interrupt transaction.
DMA failing command	DMAFCMD	Saves command, mask, and address information of a failing DMA or interrupt transaction.
Error interrupt control	ERRINTR	Disables error interrupts during diagnostic testing.
Configuration	CNF	Contains the XJA number and node ID, and main memory size and starting address.
XBI ID A	XBIIDA	Contains node ID of XBI 7 through XBI 0.
XBI ID B	XBIIDB	Contains node ID of XBI D(hex) through XBI 8.
Error SCB offset	ERRSCB	Contains SCB offset for XJA detected errors at IPL 17(hex).
SCB offset IPL 14	IDENT4	Contains SCB offset for XMI generated interrupts at IPL 14(hex).
SCB offset IPL 15	IDENT5	Contains SCB offset for XMI generated interrupts at IPL 15(hex).
SCB offset IPL 16	IDENT6	Contains SCB offset for XMI generated interrupts at IPL 16(hex).
SCB offset IPL 17	IDENT7	Contains SCB offset for XMI generated interrupts at IPL 17(hex).

5.2 XMI Space Registers

Table 5–2 lists the XMI space registers. Each XMI node is required to implement specific registers contained in specific locations within the node's address space. The first two registers listed in Table 5–2 (XDEV and XBER) are required of all XMI nodes. The next two registers listed (XFADR and XFAER) are required of all XMI commander nodes. The remaining four registers listed are optional XMI space registers used by the XJA.

Table 5–2 XMI Space Registers

Register	Mnemonic	Address[1]	Node Requirements
Device[2]	XDEV	bb + 00	All nodes
Bus error[2]	XBER	bb + 04	All nodes
Failing address[2]	XFADR	bb + 08	Commander nodes only
Failing address extension[2]	XFAER	bb + 0C[3]	Commander nodes only
XJA general purpose	XJAGPR	bb + 10	Optional
Full system emulation mode control	FAEMC	bb + 14	Optional
Add-on self-test status	AOSTS	bb + 18	Optional
XJA serial number	SERNUM	bb + 1C	Optional

[1]The bb refers to the base address of a node (the address of the first location in node space).

[2]XMI bus required.

[3]XFAER is also accessed at address bb + 2C.

5.2.1 Device Register

The device register (XDEV) has two fields containing information about the node device. Both fields are loaded during node initialization. (See Figure 5–1 and Table 5–3.)

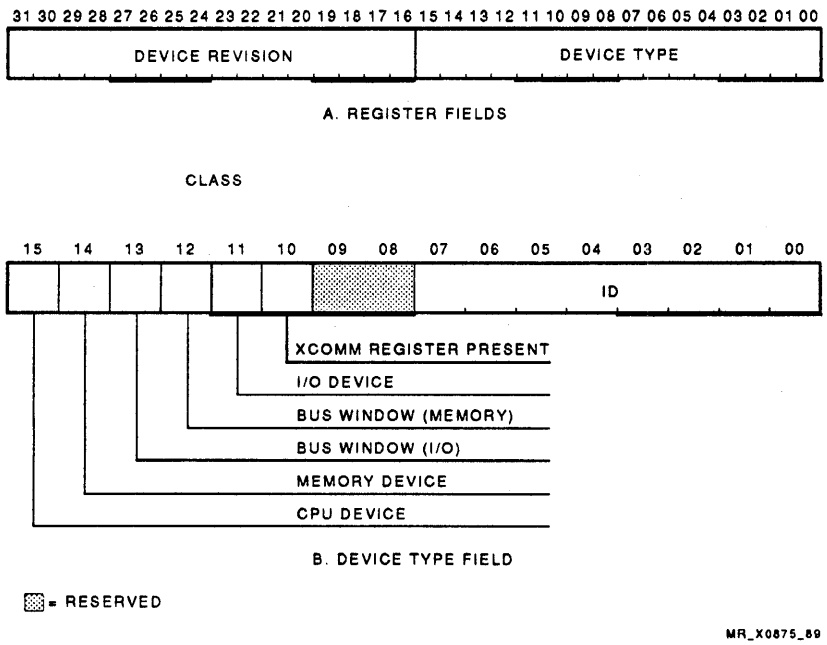


Figure 5–1 Device Register

Table 5–3 Device Register

Bit	Name	Type	Initial State[1]	Description
31:16	Device revision (DREV)	RO	1	Initialized by XJA. Identifies the revision level of the XJA. Because this register is physically contained in the XDC gate array, this field indicates the revision level of the XDC gate array.
15:00	Device type (DTYPE)	RO	1001(hex)	Initialized by XJA. Identifies the type of device on this node. The device type field is broken into two subfields: class and ID. The class field indicates the major category into which the node falls. The currently defined classes are CPU, memory, I/O bus window, memory bus window, and I/O. In addition, bit 10 is used to indicate the presence of an XMI communication (XCOMM) register. The XCOMM register is not used in this system. The ID field uniquely identifies a particular device in a specified class.

[1]Reset or power-up

Legend

RO = Read only

The devices currently supported on the VAX 9000 XMI bus are listed in Table 5–4.

Table 5–4 VAX 9000 XMI Device Types

Device	Description	Adapter Mnemonic	Adapter Module(s)	XDEV[15:00]
KFMSA	DSSI[1] disk interface	DASH	T2036	0810
DEMNA	XMI-to-NI adapter	XNA	T2020	0C03
CIXCD	XMI-to-CI adapter	XCD	T2080	0C05
KDM70	Local DSA disk/tape interface	HSX	T2022 and T2023[2]	0C22
DWMJA	XMI-to-JBox adapter	XJA	T1061	1001
DWMBB	XMI-to-BI adapter	XBI+	T2018 and T1043[3]	2001

[1]Digital storage system interconnect.

[2]The HSX adapter requires two slots in the XMI card cage.

[3]The T1043 module is located in the VAXBI expander cabinet.

5.2.2 XMI Bus Error Register

The XMI bus error register (XBER) contains error information for errors that the XJA detects on XMI bus-related transactions. (See Figure 5–2 and Table 5–5.)

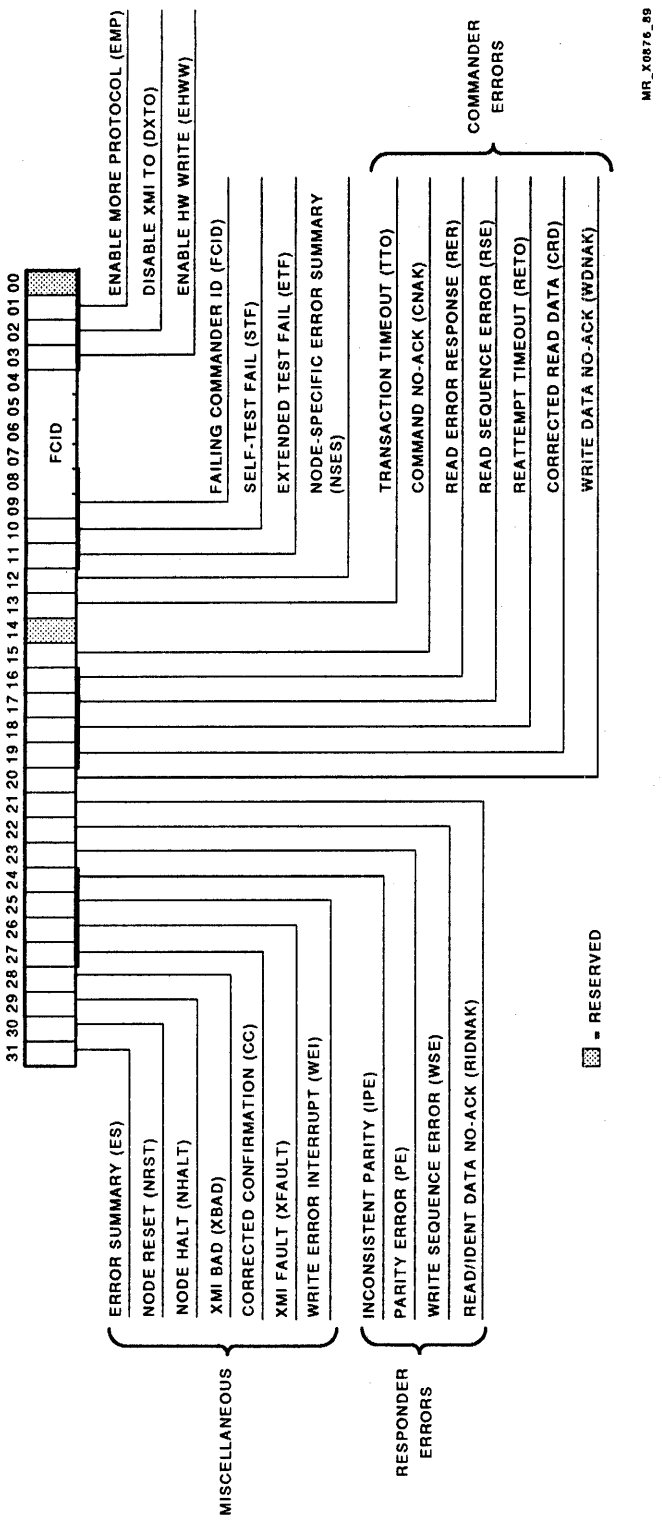


Figure 5–2 XMI Bus Error Register

Table 5–5 XMI Bus Error Register

Bit	Name	Type	Initial State[1]	Description
31	Error summary (ES)	RO	0	Represents the logical-OR of error bits XBER [27:13].
30	Node reset (NRST)	R/W	0	Writing a 1 to this location initiates reset of the XJA and start of the self-test. If FAEMC[31] is set, the XJA will not initiate self-test. When this bit is set, the XJA asserts XJA_FATALERR, requesting an IPL 1D(hex) interrupt of all the system CPUs.
29	Node halt (NHALT)	R/W	0	When this bit is set, the XJA asserts XJA_ FATALERR, requesting an IPL 1D(hex) interrupt of all the system CPUs.
28	XMI BAD (XBAD)	R/W	1	Initialized by SPU/AOST. On reads, this bit indicates the state of the XMI_BAD signal; a 1 indicates XMI_ BAD is asserted. Writes to this location establish the state of the wired-OR XMI_BAD line. Writing a 1 asserts XMI_BAD, while writing a 0 negates it. When the XJA self-test completes successfully, XMI_BAD is reset to 0.
27	Corrected confirmation (CC)	R/W 1 to clear	0	Set when the XJA detects a single-bit confirmation error. (Single-bit confirmation errors are automatically corrected by the XCLOCK chip in the XJA XMI corner.) The XJA requests an IPL 17(hex) error interrupt if ERRINTR [27] is set.
26	XMI_FAULT (XFAULT)	RO	0	When set, indicates the XMI_FAULT backplane signal is asserted for at least one cycle. The XJA asserts XJA_FATALERR, requesting an IPL 1D(hex) interrupt of all the system CPUs.
25	Write error interrupt (WEI)	R/W 1 to clear	0	When set, indicates that the XJA received a write error interrupt transaction. The XJA asserts XJA_ FATALERR, requesting an IPL 1D(hex) interrupt of all the system CPUs.
24	Inconsistent parity error (IPE)	RO as 0	0	The XJA does not implement the IPE bit.
23	Parity error (PE)	R/W 1 to clear	0	When set, indicates that the XJA detected a parity error on an XMI cycle. All XMI nodes ignore XMI cycles that have bad parity. The XJA no-acks an XMI cycle that has bad parity and sets this bit. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [23] is set. The command/address is saved in DMAFADDR [31:00] and DMAFCMD [31:00], and the commander ID is saved in XBER [09:04].

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–5 (Cont.) XMI Bus Error Register

Bit	Name	Type	Initial State[1]	Description
22	Write sequence error (WSE)	R/W 1 to clear	0	When set, indicates that the XJA aborted a DMA write transaction due to a missing data cycle. Write sequence errors occur when the number of data cycles specified in a write command/address cycle, do not follow the command/address cycle on the XMI bus. Upon detection of a write sequence error on a DMA XMI transaction, the XJA no-acks the offending cycle, sets this bit, frees the assigned DMA write buffer, and prevents the transmission of the write data to the ICU. The command/address is saved in DMAFADDR and DMAFCMD, and the commander ID is saved in XBER [09:04]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [22] is set.
21	Read/IDENT data no-ack (RIDNAK)	R/W 1 to clear	0	When set, indicates that a DMA read data cycle (GRDn, CRDn, LOC, or RER) transmitted by the XJA has received a no-ack confirmation. Upon detection of a read/IDENT data no-ack, the XJA sets this bit and clears the return read data buffer. If the no-ack was on return read lock data, the original requester of the data interrupts the system on the no-ack or times out. The system can free the now stuck lock using the physical address stored in the original node's XBER/XFADR. The XJA does not respond to IDENT command transactions from the XMI bus. The command/address currently in DMAFADDR [31:00] and DMAFCMD [31:00], and the commander ID currently in XBER [09:04] are saved. If no intervening transaction is received from the XMI bus, the saved command/address and commander ID is that of the XMI commander that initiated the read transaction. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [21] is set.
20	Write data no-ack (WDNAK)	R/W 1 to clear	0	When set, indicates that a CPU write data cycle transmitted by the XJA received a no-ack confirmation. Upon receipt of a no-ack confirmation code on a write data cycle, the XJA retries the entire transaction until it either completes successfully or a RETO is encountered. In this case, the bit is set. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [20] is set.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–5 (Cont.) XMI Bus Error Register

Bit	Name	Type	Initial State[1]	Description
19	Corrected read data (CRD)	R/W 1 to clear	0	When set, indicates that the XJA received a CRD read response to a CPU read request. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [19] is set.
18	Reattempt timeout (RETO)	R/W 1 to clear	0	When set, indicates that a transaction initiated by the node failed due to a reattempt timeout. The XJA sets this bit when it retries a given transaction and receives a no-ack response on a read/write command/address cycle or a write data cycle for more than 131,072 XMI cycles (approximately 8.4 ms). The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [18] is set. **NOTE** **The function of bit XBER [18] on the VAX 9000 system XMI bus is different from that of other XMI buses where XBER [18] is designated as no read response (NRR) and serves a different purpose.**
17	Read sequence error (RSE)	R/W 1 to clear	0	When set, indicates that an XMI transaction, initiated by the XJA to service a CPU read request, failed due to a read sequence error. The XJA does not reattempt the CPU read transaction if it encounters an RSE on the returned data. The XJA sets this bit if it receives any return read data function code other than GRD0 or CRD0. When the XJA detects a read sequence error, it sends a CPU read error status JXDI transaction back to the ICU. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [17] is set.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–5 (Cont.) XMI Bus Error Register

Bit	Name	Type	Initial State[1]	Description
16	Read error response (RER)	R/W 1 to clear	0	When set, indicates that an XMI transaction, initiated by the XJA to service a CPU read request, failed due to the receipt of a read error response. The XJA does not reattempt the CPU read transaction if it encounters an RER. When the XJA receives an RER, it sends a CPU read error status JXDI transaction back to the ICU. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [16] is set.
15	Command no-ack (CNAK)	R/W 1 to clear	0	When set, indicates that a read/write command cycle transmitted by the XJA received a no-ack confirmation. This confirmation can result from a reference to a nonexistent memory location or a command cycle parity error. The XJA sets this bit when it receives the no-ack confirmation for a given command/address cycle for 131,072 XMI cycles (approximately 8.4 ms). If the attempted transaction is a CPU write type transaction,the XJA will send a CPU write complete JXDI transaction back to the ICU. If the attempted transaction is a CPU read type transaction, the XJA sends a CPU read error response JXDI transaction back to the ICU. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00]. The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [15] is set.
14	–	RO as 0	–	Reserved.
13	Transaction timeout (TTO)	R/W 1 to clear	0	When set, indicates that an XMI read or IDENT transaction, initiated by the XJA, failed due to a transaction timeout. The XJA sets this bit if it fails to receive a response to an acked read or IDENT command cycle within 131,072 XMI cycles (approximately 8.4 ms). The XJA requests an error interrupt at an IPL of 17(hex) if ERRINTR [13] is set. The command/address is saved in XFADR [31:00] and XFAER [31:00], and the CPU ID is saved in ERRS [05:00].
12	Node-specific error summary (NSES)	RO	0	Set when any of ERRS [31:20] are set.
11	Extended test fail (ETF)	RO as 0	0	The XJA does not implement extended self-test and returns 0 for this bit.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–5 (Cont.) XMI Bus Error Register

Bit	Name	Type	Initial State[1]	Description
10	Self-test fail (STF)	R/W	1	When set, indicates that the XJA has not yet passed its self-test. XJA add-on self-test (AOST) logic clears this bit upon the successful completion of the XJA self-test. AOST status is reported in the AOSTS register (Section 5.2.7).
09:04	Failing commander ID (FCID)	RO	0	Used to log the commander ID of a failing DMA or interrupt transaction. Locked when any of the XBER [23:21] bits are set.
03	Enable hexword writes (EHWW)	RO as 0	0	The XJA does not generate hexword length transactions and, therefore, does not implement this bit.
02	Disable XMI timeout (DXTO)	RO as 0	0	The XJA does not implement this bit.
01	Enable MORE protocol (EMP)	RO as 0	0	The XJA does not support the MORE protocol and does not implement this bit.
00	–	RO as 0	–	Reserved.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.2.3 Failing Address Register

The failing address register (XFADR) is used to log the low-order four bytes of an XMI command/address cycle associated with a failing CPU transaction. (See Figure 5–3 and Table 5–6.)

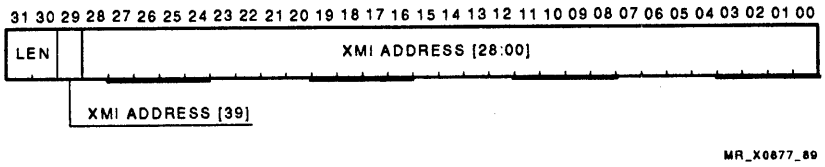


Figure 5–3 Failing Address Register

Table 5–6 Failing Address Register

Bit	Name	Type	Initial State[1]	Description
31:30	Failing length (FLN)	RO	Undefined	Used to log the value of XMI D [31:30] during the command cycle of a failing CPU transaction. Locked when any of the XBER [20:15] bits or XBER [13] is set.
29	Failing address [39]	RO	Undefined	Used to log the value of XMI D [29] during the command cycle of a failing CPU transaction. Locked when any of the XBER [20:15] bits or XBER [13] is set.
28:00	Failing address [28:00]	RO	Undefined	Used to log the value of XMI D [28:00] during the command cycle of a failing CPU transaction. Locked when any of the XBER [20:15] bits or XBER [13] is set.

[1]Reset or power-up

Legend

RO = Read only

5.2.4 Failing Address Extension Register

The failing address extension register (XFAER) is used to log the high-order four bytes of an XMI command/address cycle associated with a failing CPU transaction. (See Figure 5–4 and Table 5–7.)

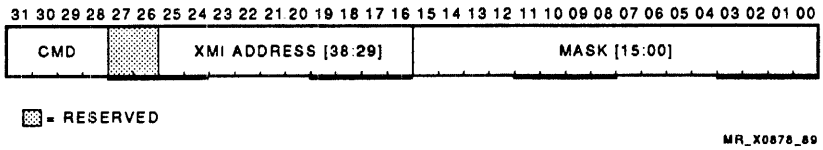


Figure 5–4 Failing Address Extension Register

Table 5–7 Failing Address Extension Register

Bit	Name	Type	Initial State[1]	Description
31:28	Failing command (FCMD)	RO	Undefined	Used to log the command code of a failing CPU transaction. Locked when any of the XBER [20:15] bits or XBER [13] is set.
27:26	–	RO as 0	–	Reserved.
25:16	Failing address [38:29]	RO	Undefined	Used to log the value of XMI D [57:48] during the command cycle of a failing CPU transaction. Locked when any of the XBER [20:15] bits or XBER [13] is set.
15:00	Failing mask	RO	Undefined	Used to log the value of XMI D [47:32] during the command cycle of a failing CPU transaction. XMI D [47:32] are mask bits on an XMI write transaction. This field is locked when any of the XBER [20:15] bits or XBER [13] is set.

[1]Reset or power-up

Legend

RO = Read only

5.2.5 XJA General-Purpose Register

The XJA general-purpose register (XJAGPR) is a longword length register in XJA XMI node space used for diagnostic testing of the XJA data path. (See Figure 5–5 and Table 5–8.)

XJAGPR is a read/write register used as a source for write data when XMI write transactions are initiated using the FCMD register. It is also used to contain the returned read data when XMI read transactions are initiated using the FCMD register (Table 5–14).

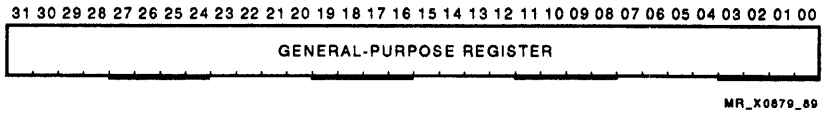


Figure 5–5 XJA General-Purpose Register

Table 5–8 XJA General-Purpose Register

Bit	Name	Type	Initial State[1]	Description
31:00	XJA GPR	R/W	Undefined	Test data.

[1]Reset or power-up

Legend

R/W = Read/write

5.2.6 Full System Emulation Mode Control Register

The full system emulation mode control (FAEMC) register controls operation of the XJA in a special emulation mode; another system is used to simulate the VAX 9000 to the XJA (Figure 5–6). This mode of operation is not described in this manual.

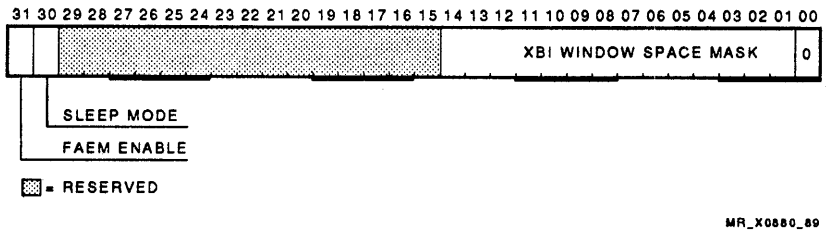


Figure 5–6 Full System Emulation Mode Control Register

5.2.7 Add-On Self-Test Status Register

The add-on self-test status (AOSTS) register (Figure 5–7 and Table 5–9) contains the results of various tests run by the AOST logic in the XJA. The register contents depend on the test just run. See Chapter 4 for a description of the AOST logic.

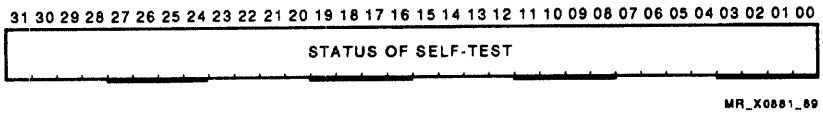


Figure 5–7 Add-On Self-Test Status Register

Table 5–9 Add-On Self-Test Status Register

Bit	Name	Type	Initial State[1]	Description
31:00	AOST status (AOSTS)	R/W	Undefined	Test data.

[1]Reset or power-up

Legend

R/W = Read/write

5.2.8 XJA Serial Number Register

The XJA serial number (SERNUM) register (Figure 5–8 and Table 5–10) is loaded with the contents of a serial number PROM by the AOST logic at the completion of self-test. The serial number PROM is programmed with the serial number of the XJA module using a bar code reader wand.

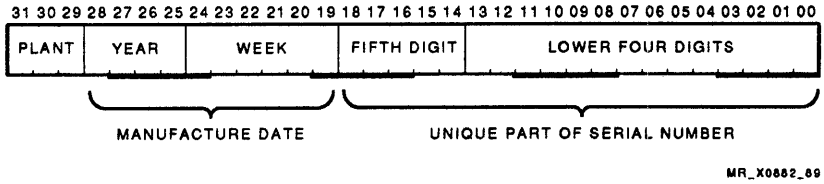


Figure 5–8 XJA Serial Number Register

Table 5–10 XJA Serial Number Register

Bit	Name	Type	Initial State[1]	Description
31:29	Plant code	R/W	Loaded by AOST	Contains a code that corresponds to a specific Digital manufacturing facility as follows:

SERNUM [31:29]	Site Code	Location
0 0 0	ASO	Augusta manufacturing
0 0 1	COM	Caribbean operations manufacturing
0 1 0	BTO	Burlington manufacturing
0 1 1	GAO	Galway manufacturing
Else	–	Reserved

28:25	Year of manufacture	R/W	Loaded by AOST	Contains the last digit of the year of manufacture in binary format.
24:19	Week of manufacture	R/W	Loaded by AOST	Contains the week of manufacture in binary format.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–10 (Cont.) XJA Serial Number Register

Bit	Name	Type	Initial State[1]	Description
18:14	Fifth digit	R/W	Loaded by AOST	Contains the fifth digit of the unique portion of the serial number. The digit could be a number or a letter as shown in the following:

SERNUM [18:14]	Fifth Digit	SERNUM [18:14]	Fifth Digit
00000	0	10000	G
00001	1	10001	H
00010	2	10010	I
00011	3	10011	J
00100	4	10100	K
00101	5	10101	L
00110	6	10110	M
00111	7	10111	N
01000	8	11000	O
01001	9	11001	P
01010	A	11010	Q
01011	B	11011	R
01100	C	11100	S
01101	D	11101	T
01110	E	11110	U
01111	F	11111	V

13:00	Last four digits	R/W	Loaded by AOST	Contains the last four digits of the unique portion of the serial number in binary format.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3 XJA Private Registers

The XJA private registers (Table 5–11) are used for error reporting and diagnostic functions. These registers are accessible only from the system CPU. They cannot be accessed from the XMI bus.

Table 5–11 XJA Private Registers

Register	Mnemonic	Address[1]
Error summary	ERRS	bb + 00
Force command	FCMD	bb + 04
Interprocessor interrupt source	IPINTRSRC	bb + 08
XJA diagnostic control	DIAG	bb + 0C
DMA failing address	DMAFADDR	bb + 10
DMA failing command	DMAFCMD	bb + 14
Error interrupt control	ERRINTR	bb + 18
Configuration	CNF	bb + 1C
XBI ID A	XBIIDA	bb + 20
XBI ID B	XBIIDB	bb + 24
Error SCB offset	ERRSCB	bb + 28
SCB offset IPL 14	IDENT4	bb + 40
SCB offset IPL 15	IDENT5	bb + 44
SCB offset IPL 16	IDENT6	bb + 48
SCB offset IPL 17	IDENT7	bb + 4C

[1]bb = 3 FE00 0000 + (XJA number × 8 0000)

5.3.1 Error Summary Register

The error summary (ERRS) register (Figure 5–9 and Table 5–12) contains information on errors that the XJA detected on JXDI and XMI transactions.

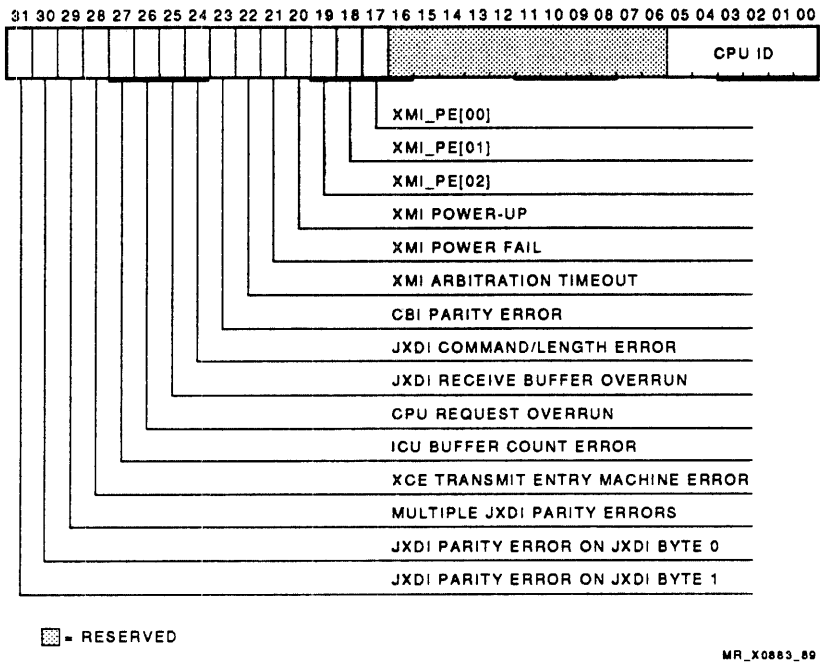


Figure 5–9 Error Summary Register

Table 5–12 Error Summary Register

Bit	Name	Type	Initial State[1]	Description
31	JXDI parity error on JXDI byte 1	R/W 1 to clear	0	When set, indicates that the XJA detected a parity error on received JXDI [15:08]. When this bit is set, ERRS [30] is locked. When this bit is set and ERRINTR [01] is set, the XJA initiates an IPL 17(hex) error interrupt to the system CPU. If ERRS [31] is set, ERRS [30] cannot set. Both ERRS [31:30] can be set at the same time if parity errors occur simultaneously on both JXDI [15:08] and [07:00].

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–12 (Cont.) Error Summary Register

Bit	Name	Type	Initial State[1]	Description
30	JXDI parity error on JXDI byte 0	R/W 1 to clear	0	When set, indicates that the XJA detected a parity error on received JXDI [07:00]. When this bit is set, ERRS [31] is locked. When this bit is set and ERRINTR [01] is set, the XJA initiates an IPL 17(hex) error interrupt to the system CPU. If ERRS [30] is set, ERRS [31] cannot set. Both ERRS [31:30] can be set at the same time if parity errors occur simultaneously on both JXDI [15:08] and [07:00].
29	Multiple JXDI parity errors	R/W 1 to clear	0	When set, indicates that a receive JXDI parity error was detected when either ERRS [31] or ERRS [30] was set. This indicates that JXDI parity errors are occurring at a high rate and further operation of the JXDI interface may be unreliable.
28	XCE transmit entry machine error	R/W 1 to clear	0	When set, indicates that the XCE array detected an illegal state of one of its internal transmit entry machines. Asserts XJA_FATALERR on the JXDI, indicating that further operation of the JXDI interface is unreliable.
27	ICU buffer count error	R/W 1 to clear	0	When set, indicates that the XJA received too many ICU_BUFEMPTD assertions. This means that the ICU indicated it has more than two receive buffers available. Asserts XJA_FATALERR on the JXDI, indicating that further operation of the JXDI interface is unreliable.
26	CPU request overrun	R/W 1 to clear	0	When set, indicates that the XJA received more than one CPU request before it sent completion information (read data, read status, or write complete) to the ICU. Asserts XJA_FATALERR on the JXDI, indicating that further operation of the JXDI interface is unreliable.
25	JXDI receive buffer overrun	R/W 1 to clear	0	When set, indicates that the XJA received more JXDI transactions than it has available buffers. This bit indicates that the ICU is faulty or is seeing too many XJA_BUFEMPTD assertions. Asserts XJA_FATALERR on the JXDI, indicating that further operation of the JXDI interface is unreliable.
24	JXDI command/length error	R/W 1 to clear	0	When set, indicates that the XJA received an illegal JXDI command/length combination with good parity. This indicates either a double-bit JXDI error occurred or the ICU or XJA is faulty. Asserts XJA_FATALERR on the JXDI, indicating that further operation of the JXDI interface is unreliable.
23	CBI parity error	R/W 1 to clear	0	When set, indicates that the XDC gate array detected bad parity on data received from the CBI while the XDE gate arrays detected good parity on the data when received from the ICU. This bit asserts XJA_FATALERR on the JXDI, indicating that further operation of the XJA is unreliable.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–12 (Cont.) Error Summary Register

Bit	Name	Type	Initial State[1]	Description
22	XMI arbitration timeout	**R/W 1 to clear**	0	When set, indicates that the XJA asserted its XMI commander or responder request lines and failed to receive XMI bus grant within 256 XMI cycles (approximately 16.4 µs). The failing command/address is locked in XFADR [31:00] and XFAER [31:00]. An IPL 17(hex) interrupt is initiated to the system CPU if ERRINTR [00] is set. Further CPU requests can be initiated to allow error handling software to access error information. If the failing XMI arbitration request is a DMA read data return transaction, the data is dropped. If the failing transction is a CPU write transaction, the XJA saves the command/address in XFADR [31:00] and XFAER [31:00], and sends a JXDI write complete transaction to the ICU to allow another CPU transaction to be initiated. If the failing transction is a CPU read transaction, the XJA saves the command/address in XFADR [31:00] and XFAER [31:00], and sends a JXDI read error response transaction to the ICU to allow another CPU transaction to be initiated.
21	XMI power fail	**R/W 1 to clear**	0	When set, indicates that the XMI bus, defined by the XJA, is about to lose power. The system CPU is notified by the assertion of XJA_FATALERR on the JXDI. This bit is set upon the assertion of XMI_AC_LO or XMI_DC_LO.
20	XMI power-up	**R/W 1 to clear**	1	When set, indicates that the XMI bus, defined by the XJA, completed the XMI reset sequence. When the self-test has completed successfully, the system CPU is notified by an IPL 17(hex) interrupt.
19:17	XMI parity error [02:00]	**R/W 1 to clear**	0	Represents which parity bit fails when XBER [23] is set.
16:06	–	**RO as 0**	0	Reserved.
05:00	Failing CPU ID	RO	Undefined	Contains the 6-bit CPU ID supplied by the ICU during the most recent CPU type operation. Locked whenever any of XBER [20:15] or XBER [13] are set.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3.2 Force Command Register

The force command (FCMD) register (Figure 5–10 and Table 5–13), together with the DIAG and XJAGPR registers, provides a means by which diagnostic software can exercise the XJA.

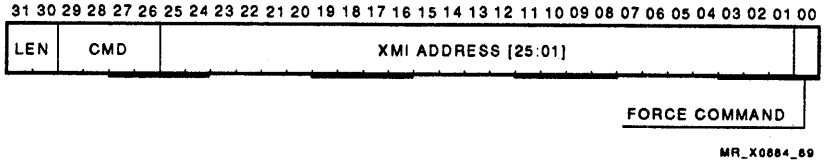


Figure 5–10 Force Command Register

Table 5–13 Force Command Register

Bit	Name	Type	Initial State[1]	Description
31:30	Length	R/W	0	Initialized by diagnostic. Contains the XMI length code for the desired transaction. The XJA asserts this field onto XMI D [31:30] when FCMD [00] is set.
29:26	Command	R/W	0	Initialized by diagnostic. Contains the XMI command code for the desired transaction. The XJA asserts this field onto XMI D [63:60] when FCMD [00] is set. The force command field allows a diagnostic to force any legal or illegal command/address onto the XMI bus. The implementation of the force command field is shown in Table 5–14.
25:01	XMI address	R/W	0	Contains the value that is asserted on XMI D [25:01] during the command/address cycle of the given transaction. XMI D [00] will always be 0.
00	Force command	R/W	0	Initialized by diagnostic. Initiates an XMI command/address cycle for the transaction specified in FCMD [29:26]. The XJA arbitrates for the XMI bus and transmits the given transaction. Upon successful completion of the transaction, the XJA clears this bit. If the system clears this bit before the XJA can complete the transaction, the XJA continues the transaction without interruption.

[1]Reset or power-up

Legend

R/W = Read/write

Table 5–14 Force Command Implementation

FCMD [29:26]	XJA Action
0 0 0 0	Undefined.
0 0 0 1	Initiate XMI read command with address from FCMD [25:01] and length from FCMD [31:30]. If the read address selects the XJA, a read data return longword (as selected by DIAG [10:08]) is loaded into XJAGPR.
0 0 1 0	Initiate XMI read lock command with address from FCMD [25:01] and length from FCMD [31:30]. If the read address selects the XJA, a read data return longword (as selected by DIAG [10:08]) is loaded into XJAGPR.
0 0 1 1	Undefined.
0 1 0 0	Undefined.
0 1 0 1	Undefined.
0 1 1 0	Initiate XMI write unlock command with address from FCMD [25:01] and length from FCMD [31:30]. The value of the 16-bit mask field is derived from the absolute decode of FCMD [04:01]. The data for the required succeeding cycles is derived from XJAGPR as follows:

Cycle	Data [63:32]	Data [31:00]
0	Not (XJAGPR)	XJAGPR
1	Not (XJAGPR sl 1)	XJAGPR sl 1
2	Not (XJAGPR sl 2)	XJAGPR sl 2
3	Not (XJAGPR sl 3)	XJAGPR sl 3

sl = Shift left (toward the MSB) and rotate. (MSB goes to LSB)

0 1 1 1	Initiate XMI write command with address from FCMD [25:01] and length from FCMD [31:30]. The value of the 16-bit mask field is derived from the absolute decode of FCMD [04:01]. The data for the required succeeding cycles is derived from XJAGPR as shown above for the write unlock command.
1 0 0 0	Initiate XMI interrupt command with IPL from FCMD [19:16] and destination from FCMD [15:01].
1 0 0 1	Initiate XMI IDENT command with IPL from FCMD [19:16] and destination from FCMD [15:01]. IDENT commands initiated using the FCMD register do not perform functionally identical to reads of the SCB offset register.
1 0 1 0	Undefined.
1 0 1 1	Undefined.
1 1 0 0	Undefined.
1 1 0 1	Undefined.
1 1 1 0	Undefined.
1 1 1 1	Initiate implied vector interrupt on the XMI bus, with the type of interrupt specified by FCMD [19:16], and the destination specified by FCMD [15:01].

5.3.3 Interprocessor Interrupt Source Register

NOTE
The XJA supports interprocessor interrupts, therefore, the content of this register is described. However, because the VAX 9000 CPU is the only CPU within the entire system configuration, no interprocessor interrupts occur on the XMI bus.

The interprocessor interrupt source (IPINTRSRC) register is shown in Figure 5–11 and described in Table 5–15.

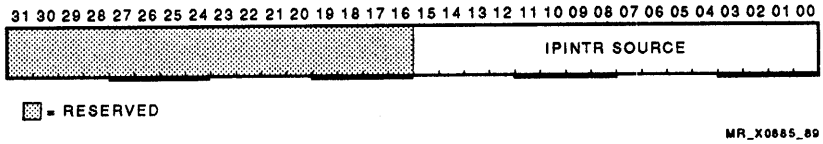


Figure 5–11 Interprocessor Interrupt Source Register

Table 5–15 Interprocessor Interrupt Source Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	0	Reserved.
15:00	IPINTR source	R/W 1 to clear	0	The XJA loads the IPINTR source field with the decoded XMI node ID of any node that sends an interprocessor interrupt to the XJA. The bit corresponding to the interrupting commander's ID is set when an interprocessor interrupt is received whose destination matches the node ID of the XJA. An IPL 16(hex) interrupt is sent to the system CPU only when the IPINTRSRC register transitions from a value of zero to nonzero. This means that after the receipt of an interprocessor interrupt from the XMI bus and the transmission of an IPL 16(hex) interrupt to the CPU, the IPINTRSRC register must be cleared before another IPL 16(hex) interrupt will be accepted by the CPU. This is true regardless of the number of additional interprocessor interrupts that are received in the interim.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3.4 XJA Diagnostic Control Register

The XJA diagnostic (DIAG) control register is shown in Figure 5–12 and described in Table 5–16.

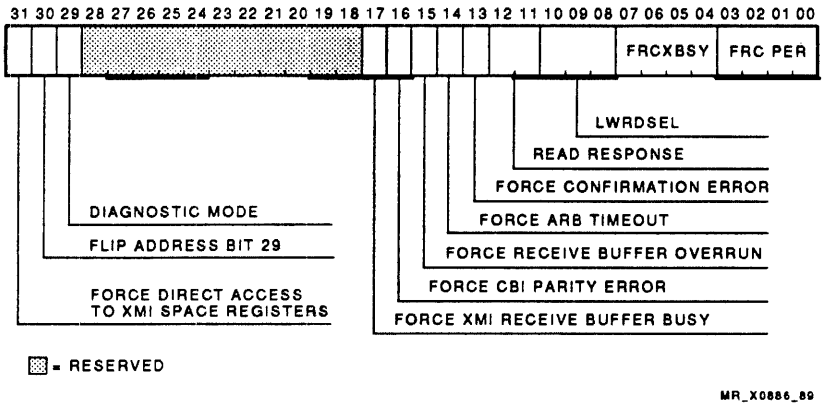


Figure 5–12 XJA Diagnostic Control Register

Table 5–16 XJA Diagnostic Control Register

Bit	Name	Type	Initial State[1]	Description
31	Force local XJA register access	R/W	0	When set, forces all XMI space registers (Table 5–2) to be accessed directly within the XJA instead of by an XMI transaction (Section 5.1). Accessing the XMI space registers directly allows diagnostics and error handling software to gather error information in the event of a faulty XMI bus.
30	Flip address bit [29]	R/W	0	When set, forces the XJA to invert the state of XCI_D [29] and XCI_P [00] during command/address cycles initiated by the XJA. This bit, along with DIAG [10:04], allows diagnostic software to turn I/O space references into memory space references to test the XJA's DMA logic.
29	Diagnostic mode	R/W	0	When set, forces the XJA to load the XJAGPR register with any read return data received from the XMI bus. Used with DIAG [10:08] and the FCMD register to allow a diagnostic program to test the XJA data paths. **NOTE** **During normal operation, this bit must be 0.**

[1]Reset or power-up

Legend

nn = Cycle number
RO = Read only
R/W = Read/write

Table 5–16 (Cont.) XJA Diagnostic Control Register

Bit	Name	Type	Initial State[1]	Description
28:18	–	RO as 0	–	Reserved.
17	Force XMI receive buffer busy	R/W	0	See DIAG [07:04].
16	Force CBI parity error	RO as 0/W	0	When set, causes the XJA to detect a CBI parity error on the next packet received from the ICU. The XJA clears DIAG [16] and asserts XJA_FATALERR to the ICU.
15	Force receive buffer overrun	RO as 0/W	0	When set, forces ERRS [25] to set. The XJA clears DIAG [15] and asserts XJA_FATALERR to the ICU.
14	Force arb timeout	R/W	0	When set, forces a negated XMI_CMD_REQ from the XJA to the XMI bus. CPU reads and writes to the XMI bus will never win the bus, XMI timeout will occur, and ERRS [22] will set.
13	Force CNF error	R/W	0	When set, forces the XJA to assert XCI_TCNF [01:00] so that a single-bit error is induced on the XMI confirmation lines.
12:11	Read response mode	R/W	0	Allows diagnostic software to force the assertion of all the various responses to read commands. When selected, the XJA returns the following response codes when returning read data (DMA read data or register read data):

DIAG [12:11]	XMI Read Response Function Code
0 0	GRDnn (good read data)
0 1	CRDnn (corrected read data)
1 0	Locked response
1 1	Read error response

10:08	Longword read select	R/W	0	Allows diagnostic software to select which longword out of the eight possible longwords will be returned to the XJA when an XMI read transaction is initiated using the FCMD register. Table 5–14 describes how the selected longword is loaded into XJAGPR. **NOTE** **During normal operation, DIAG [10:08] must be 0.**

[1]Reset or power-up

Legend

nn = Cycle number
RO = Read only
R/W = Read/write

Table 5–16 (Cont.) XJA Diagnostic Control Register

Bit	Name	Type	Initial State[1]	Description
07:04	Force XMI receive buffer busy	R/W	0	Allows diagnostic software to test operation of the five XMI command/address buffers in the TRF by setting various buffers to a busy state. If set busy, a command/address buffer is not available for use by the XJA in fielding XMI transactions. When DIAG [17] or any of the DIAG [07:04] bits are set, the associated DMA write buffer is bound to the command/address buffer and is therefore also busy. The coding of this field is as follows:

Force Bit	Busy Buffer
DIAG [17]	C/A buffer 4
DIAG [07]	C/A buffer 3
DIAG [06]	C/A buffer 2
DIAG [05]	C/A buffer 1
DIAG [04]	C/A buffer 0

NOTE

If three command/address buffers are forced busy, all transactions on the XMI bus are suppressed. The system CPUs can still access XMI space registers directly by asserting bit DIAG [31].

[1]Reset or power-up

Legend

nn = Cycle number
RO = Read only
R/W = Read/write

Table 5–16 (Cont.) XJA Diagnostic Control Register

Bit	Name	Type	Initial State[1]	Description
03:00	Force parity error	R/W	0	Allows diagnostic software to force the XJA to assert incorrect parity on the various interfaces that it drives. The coding of this field is as follows:

DIAG [03:00]	Parity Test
0 0 0 0	No parity error is forced.
0 0 0 1	Force parity error, XCI_P[00], C/A cycle.
0 0 1 0	Force parity error, XCI_P[01], C/A cycle.
0 0 1 1	Force parity error, XCI_P[02], C/A cycle.
0 1 0 0	Force parity error, XCI_P[00], data cycle.
0 1 0 1	Force parity error, XCI_P[01], data cycle.
0 1 1 0	Force parity error, XCI_P[02], data cycle.
0 1 1 1	Reserved.
1 0 0 0	Force parity error, JXDI_P[00], cycles 0 and 1.
1 0 0 1	Force parity error, JXDI_P[01], cycles 0 and 1.
1 0 1 0	Force parity error, JXDI_P[00], cycle 2.
1 0 1 1	Force parity error, JXDI_P[01], cycle 2.

[1]Reset or power-up

Legend

nn = Cycle number
RO = Read only
R/W = Read/write

5.3.5 DMA Failing Address Register

The DMA failing address (DMAFADDR) register (Figure 5–13 and Table 5–17) is used to log address and length information associated with a failing DMA or interrupt transaction. The following bit field definitions apply only to DMA transactions. This register is locked if any of the XBER [23:21] bits are set.

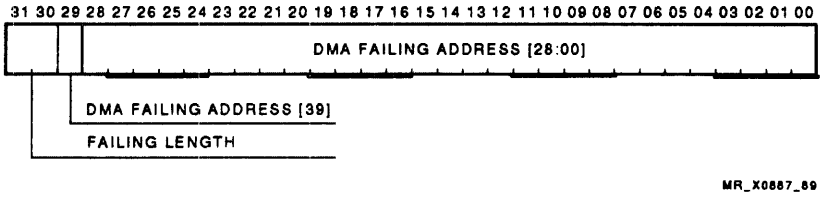


Figure 5–13 DMA Failing Address Register

Table 5–17 DMA Failing Address Register

Bit	Name	Type	Initial State[1]	Description
31:30	DMA failing length	RO	Undefined	Used to log the value of XMI D [31:30] during the command cycle of a failing DMA or interrupt transaction.
29	DMA failing address [39]	RO	Undefined	Used to log the value of XMI D [29] during the command cycle of a failing DMA or interrupt transaction.
28:00	DMA failing address [28:00]	RO	Undefined	Used to log the value of XMI D [28:00] during the command cycle of a failing DMA or interrupt transaction.

[1]Reset or power-up

Legend

RO = Read only

5.3.6 DMA Failing Command Register

The DMA failing command (DMAFCMD) register (Figure 5–14 and Table 5–18) is used to log command information associated with a failing DMA or interrupt transaction, along with the balance of the address field. The following bit field definitions apply only to DMA transactions. This register is locked if any of the XBER [23:21] bits are set.

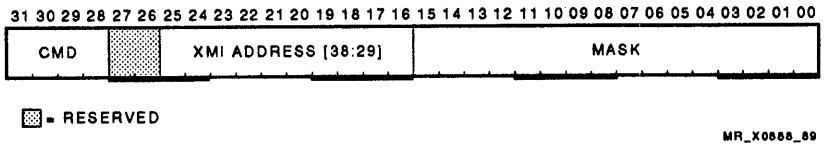


Figure 5–14 DMA Failing Command Register

Table 5–18 DMA Failing Command Register

Bit	Name	Type	Initial State[1]	Description
31:28	DMA failing command	RO	Undefined	Used to log the value of XMI D [63:60] during the command cycle of a failing DMA or interrupt transaction.
27:26	–	RO as 0	–	Reserved.
25:16	Failing address [38:29]	RO	Undefined	Used to log the value of XMI D [57:48] during the command cycle of a failing DMA or interrupt transaction.
15:00	Failing mask	RO	Undefined	Used to log the value of XMI D [47:32] during the command cycle of a failing DMA or interrupt transaction. XMI D [47:32] are mask bits for a DMA write transaction.

[1]Reset or power-up

Legend

RO = Read only

5.3.7 Error Interrupt Control Register

The error interrupt (ERRINTR) control register (Figure 5–15 and Table 5–19) is used to disable specific types of interrupts during diagnostic testing. Disabling an interrupt allows a diagnostic to test the associated error bit without generating an interrupt. In normal operation, all the interrupt enable bits in ERRINTR are set.

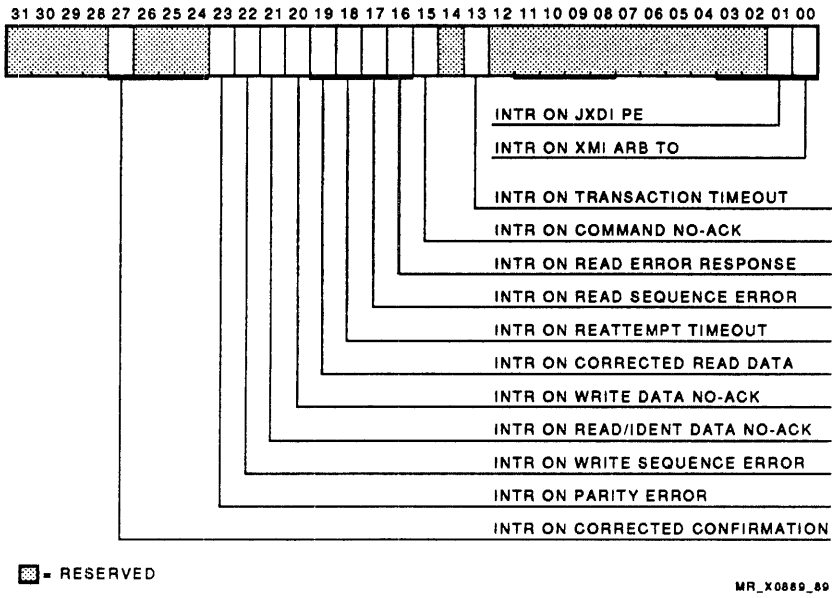


Figure 5–15 Error Interrupt Control Register

Table 5–19 Error Interrupt Control Register

Bit	Name	Type	Initial State[1]	Description
31:28	–	RO as 0	–	Reserved.
27	INTR on corrected confirmation	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the CC bit is set in the XBER register.
26:24	–	RO as 0	–	Reserved.
23	INTR on parity error	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the PE bit is set in the XBER register.
22	INTR on write sequence error	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the WSE bit is set in the XBER register.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

Table 5–19 (Cont.) Error Interrupt Control Register

Bit	Name	Type	Initial State[1]	Description
21	INTR on read/IDENT data no-ack	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the RIDNAK bit is set in the XBER register.
20	INTR on write data no-ack	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the WDNAK bit is set in the XBER register.
19	INTR on corrected read data	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the CRD bit is set in the XBER register.
18	INTR on reattempt timeout	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the RETO bit is set in the XBER register.
17	INTR on read sequence error	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the RSE bit is set in the XBER register.
16	INTR on read error response	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the RER bit is set in the XBER register.
15	INTR on command no-ack	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the CNAK bit is set in the XBER register.
14	–	RO as 0	–	Reserved.
13	INTR on transaction timeout	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if the TTO bit is set in the XBER register.
12:02	–	RO as 0	–	Reserved.
01	INTR on JXDI parity error	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if bit [31] or [30] is set in the ERRS register.
00	INTR on XMI arbitration timeout	R/W	0	Initialized by operating system. When set, causes the XJA to deliver an IPL 17(hex) interrupt to the ICU if bit [22] is set in the ERRS register.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3.8 Configuration Register

The configuration (CNF) register is shown in Figure 5–16 and Table 5–20.

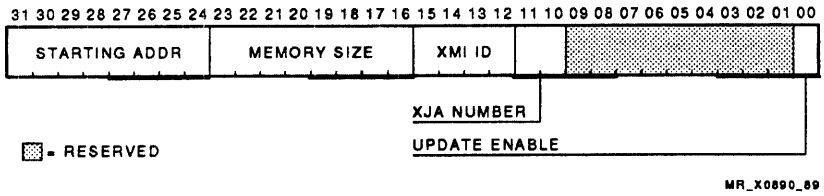


Figure 5–16 Configuration Register

Table 5–20 Configuration Register (CNF)

Bit	Name	Type	Initial State[1]	Description
31:24	Starting address	R/W	0	Initialized by SPU. Defines bits [33:26] of the address of the first location of a block of addresses to be recognized by the XJA for selection as an XMI responder. Addresses received by the XJA that fall below this point are no-acked. Allows the system software to configure the XMI bus connected to this XJA, so that integer multiples of 64 Mbytes of local XMI memory can be used by XMI I/O devices. **NOTE** **Any memory on the XMI bus is not accessible by a VAX 9000 system CPU. The system software is responsible for ensuring the viability of any configuration that implements memory on the XMI bus.**
23:16	Memory size	R/W	0	Initialized by SPU. Contains the number of 64-Mbyte memory segments that are available in the system main memory. Serves as a "number of segments" field so that XMI transactions received by the XJA, whose ADDR [33:26] is greater than CNF [23:16], are no-acked.
15:12	XJA XMI node ID	RO	XJA XMI node ID	Initialized by XJA. Contains the XMI node ID of the XJA as defined by the physical location of the XJA in the XMI card cage. Allows the system CPU to determine the location of the XMI space registers during normal operation.
11:10	XJA number	R/W	Undefined	Initialized by EWCLA/SPU. Identifies to which physical JXDI port this XJA is connected. Loaded by the power-up XJA diagnostic.
09:01	–	RO as 0	–	Reserved.
00	Update enable	RO	[2]	Enables writing of EEPROMs in XMI devices.

[1]Reset or power-up

[2]Initial state is value of XMI update enable as controlled by the console.

Legend

RO = Read only
R/W = Read/write

5.3.9 XBI ID A Register

The XBI ID A (XBIIDA) register (Figure 5–17 and Table 5–21) is used to translate address bits [28:25] of a CPU transaction to BI window space, from the BI window number to the node ID number of the XBI adapter being referenced. Each 4-bit field contains the XMI node ID of the XBI corresponding to the BI window being referenced. (See Section 1.5.5.2.) There can be a total of 14 XBIs in a system, each uniquely identified by its BI window address.

This register, together with XBIIDB, forms a 4-bit wide, 14-location deep, lookup table that allows the XJA to obtain the BI adapter's node ID number using the associated BI window number.

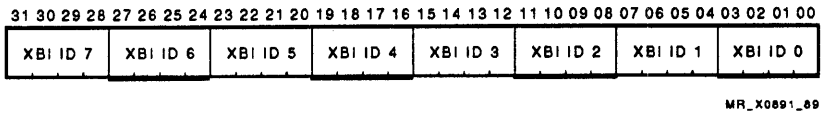


Figure 5–17 XBI ID A Register

Table 5–21 XBI ID A Register

Bit	Name	Type	Initial State[1]	Description
31:00	XBI ID 7 through XBI ID 0 in 4-bit fields	R/W	0	Initialized by the service processor. Contains 4-bit XMI node ID of BI adapters XBI 7 through XBI 0.

[1]Reset or power-up

Legend

R/W = Read/write

5.3.10 XBI ID B Register

The XBI ID B (XBIIDB) register (Figure 5–18 and Table 5–22) is used to translate address bits [28:25] of a CPU transaction to BI window space, from the BI window number to the node ID number of the XBI adapter being referenced. Each 4-bit field contains the XMI node ID of the XBI corresponding to the BI window being referenced. (See Section 1.5.5.2.) There can be a total of 14 XBIs in a system, each uniquely identified by its BI window address.

This register, together with XBIIDA, forms a 4-bit wide, 14-location deep, lookup table that allows the XJA to obtain the BI adapter's node ID number using the associated BI window number.

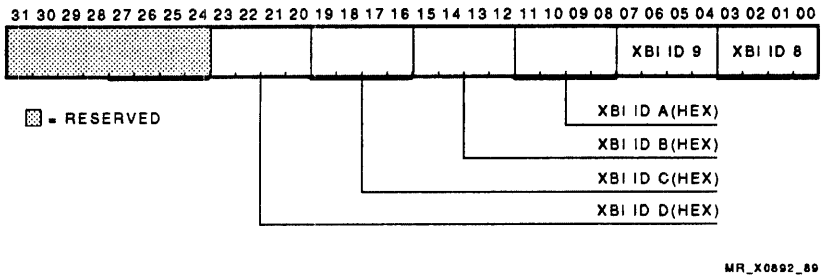


Figure 5–18 XBI ID B Register

Table 5–22 XBI ID B Register

Bit	Name	Type	Initial State[1]	Description
31:24	–	RO as 0	–	Reserved.
23:00	XBI ID D(hex) through XBI ID 8 in 4-bit fields	R/W	0	Initialized by service processor. Contains 4-bit XMI node ID of BI adapters XBI D(hex) through XBI 8.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3.11 Error SCB Offset Register

The error SCB (ERRSCB) offset register is shown in Figure 5–19 and described in Table 5–23.

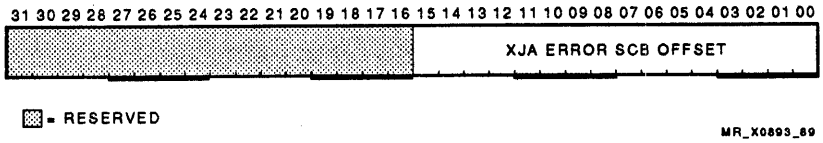


Figure 5–19 Error SCB Offset Register

Table 5–23 Error SCB Offset Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	–	Reserved.
15:00	XJA error SCB offset	R/W	64(hex)	Initialized by operating system. Contains an offset into the system control block that is returned when the XJA has an IPL 17(hex) error interrupt pending and a system CPU reads the IDENT7 register. Loaded by system software at system initialization.

[1]Reset or power-up

Legend

RO = Read only
R/W = Read/write

5.3.12 SCB Offset IPL 14(Hex) Register

The SCB offset IPL 14(hex) (IDENT4) register is shown in Figure 5–20 and described in Table 5–24.

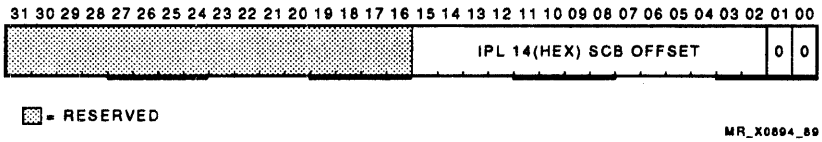


Figure 5–20 SCB Offset IPL 14(Hex) Register

Table 5–24 SCB Offset IPL 14(Hex) Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	–	Reserved.
15:00	IPL 14(hex) SCB offset	RO	0	Read access to this register by the system CPU returns an offset into the system control block corresponding to an IPL 14(hex) interrupt. A read to this register causes the XJA to initiate an XMI IDENT transaction at IPL 14(hex). The XJA returns, to the system CPU, the returned IDENT response from the XMI node that initiated the original interrupt.

[1]Reset or power-up

Legend

RO = Read only

5.3.13 SCB Offset IPL 15(Hex) Register

The SCB offset IPL 15(hex) (IDENT5) register is shown in Figure 5–21 and described in Table 5–25.

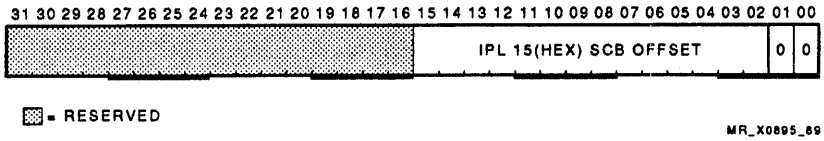


Figure 5–21 SCB Offset IPL 15(Hex) Register

Table 5–25 SCB Offset IPL 15(Hex) Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	–	Reserved.
15:00	IPL 15(hex) SCB offset	RO	0	Read access to this register by the system CPU returns an offset into the system control block corresponding to an IPL 15(hex) interrupt. A read to this register causes the XJA to initiate an XMI IDENT transaction at IPL 15(hex). The XJA returns, to the system CPU, the returned IDENT response from the XMI node that initiated the original interrupt.

[1]Reset or power-up

Legend

RO = Read only

5.3.14 SCB Offset IPL 16(Hex) Register

The SCB offset IPL 16(hex) (IDENT6) register is shown in Figure 5–22 and described in Table 5–26.

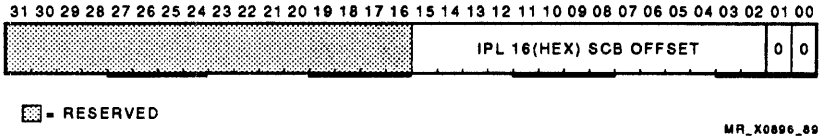


Figure 5–22 SCB Offset IPL 16(Hex) Register

Table 5–26 SCB Offset IPL 16(Hex) Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	–	Reserved.
15:00	IPL 16(hex) SCB offset	RO	0	Read access to this register by a system CPU returns an offset into the system control block corresponding to an IPL 16(hex) interrupt. If the XJA has a pending IPINTR, a read to this register returns 80(hex) to the system CPU. If no IPINTRs are pending, a read to this register causes the XJA to initiate an XMI IDENT transaction at IPL 16(hex). The XJA returns, to the system CPU, the IDENT response returned from the XMI node that initiated the original interrupt.

[1]Reset or power-up

Legend

RO = Read only

5.3.15 SCB Offset IPL 17(Hex) Register

The SCB offset IPL 17(hex) (IDENT7) register is shown in Figure 5–23 and described in Table 5–27.

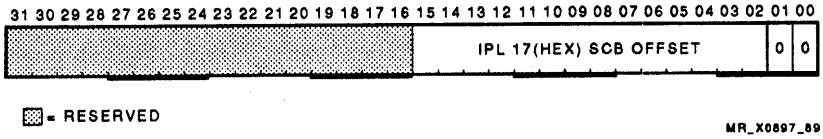


Figure 5–23 SCB Offset IPL 17(Hex) Register

Table 5–27 SCB Offset IPL 17(Hex) Register

Bit	Name	Type	Initial State[1]	Description
31:16	–	RO as 0	–	Reserved.
15:00	IPL 17(hex) SCB offset	RO	0	Read access to this register by the system CPU returns an offset into the system control block corresponding to an IPL 17(hex) interrupt. If the XJA has a pending error interrupt, a read to this register returns the SCB offset contained in ERRSCB [15:00] to the system CPU. If no XJA error interrupt is pending, a read to this register causes the XJA to initiate an XMI IDENT transaction at IPL 17(hex). The XJA returns, to the system CPU, the IDENT response returned from the XMI node that initiated the original interrupt.

[1]Reset or power-up

Legend

RO = Read only

Index

D

E

F

I

J

T

U

W

X

Y