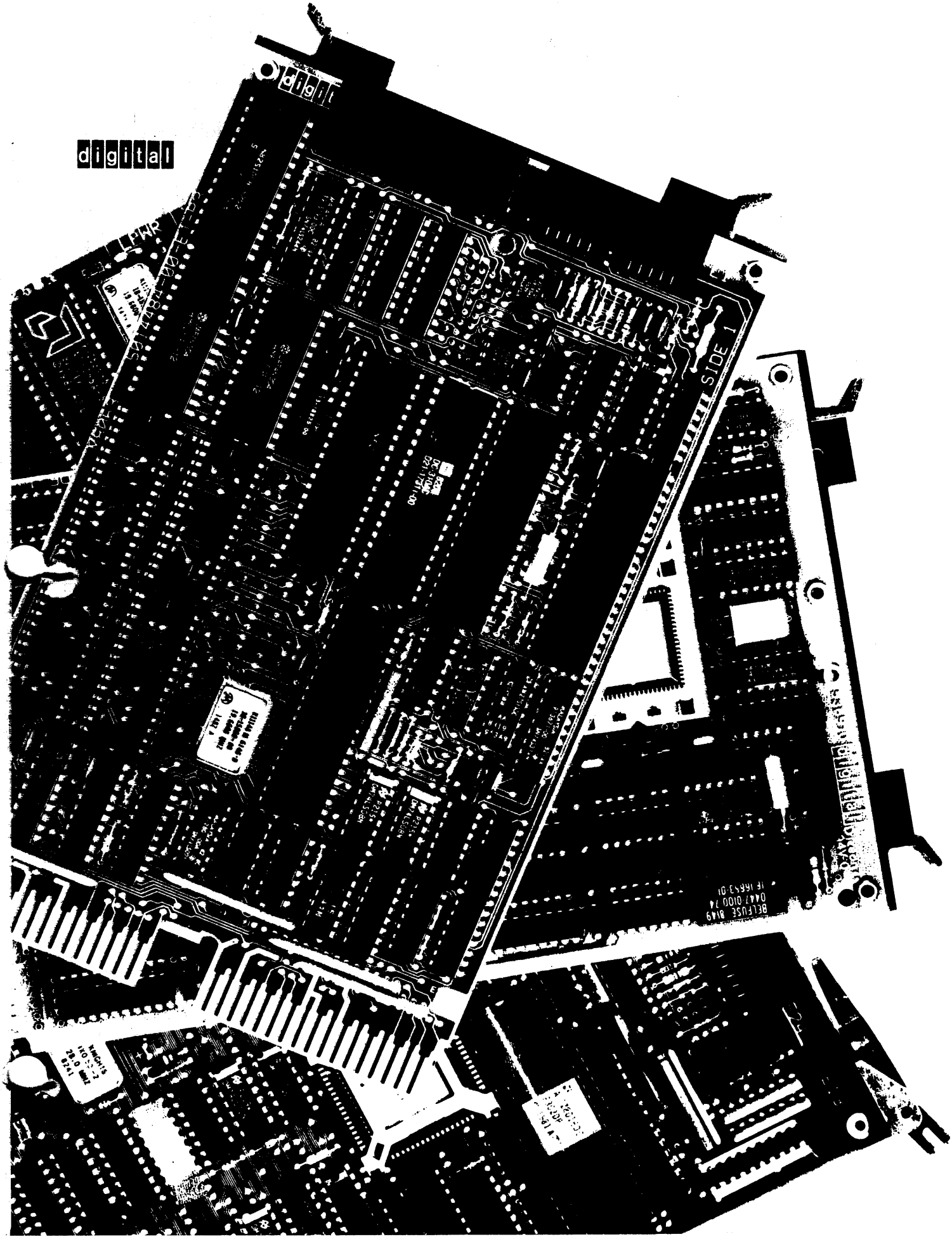

digital
digital
PWR
SIDE 1

digital

Enclosed is the new set of MicroNotes. This set consists of twenty-one new documents relating to some of the latest component products from Digital.

The original set of 111 MicroNotes has been superseded by this new edition. The titles in the original set can be found in Appendix A of the enclosed document. The original MicroNotes (if you do not have them) can still be obtained by writing to the OEM Technical Support Group at:

OEM Micros Technical Support Group
Digital Equipment Corporation
2 Iron Way MR03-3/G20
Marlboro, MA 01752

Attn: Cindy Dorval

Be sure to ask for the original MicroNotes.

If there is someone you know that would like to be added to the MicroNote Distribution List have them fill out the enclosed MicroNote Reservation Form and return it to the address listed above.

The group would appreciate any feedback or suggestions on future MicroNotes these comments can also directed to the above address.

Sincerely,

OEM Micros Technical Support Group

Digital Equipment Corporation
Two Iron Way
Box 1003
Marlboro, Massachusetts 01752-9103
617.467.5111

digital

Your name is on our mailing list. Enclosed is an updated set of MicroNotes which consists of the twenty-one previously published documents plus twenty NEW MicroNotes. The information contained in this set relates to some of the latest component and small system products from Digital.

If someone would like to be added to the MicroNote Distribution List, have them fill out the enclosed MicroNote Reservation Form and return it to the address listed below; attention Cindy Dorval. This form can also be used to make address corrections, noting the date your location changed.

The group would appreciate any feedback or suggestions for future MicroNotes. These comments can also be directed to the address below.

Thank you for your continuing interest.

OEM Technical Support Group
Digital Equipment Corporation
2 Iron Way (MRO3-3/G20)
Marlboro, MA. 01752-9103

## MicroNote Reservation Form

Please fill out this form and return it to:

OEM Micros Technical Support Group
Digital Equipment Corporation
2 Iron Way MRO3-3/G20
Marlboro, MA 0172
Attn: Cindy Dorval

This will add you to the MicroNote Distribution List. MicroNotes are short technical articles written about Digital's component level products. Product highlights, technical descriptions, technical hints-and-kinks not found in the regular documentation, and recent product changes and announcements are discussed in the MicroNotes.

Name: _____

Company: _____

Title: _____

Address: _____

City: _____

State: _____ Zip: _____

## Questionnaire

1. I am:
   - o an OEM
   - o a Distributor
   - o an End-User
   - o Other _____

2. The Industry I Service is (e.g. Medical, Control, Instrumentation, Education):

3. The Application(s) within that Industry is/are (e.g. machine control, IC testers, general purpose computing):

4. I'd like future MicroNotes to Discuss:

TABLE OF CONTENTS

| uNOTE # 001 |
|---|

| Title: MUL,DIV and ASH Instruction for the FALCON and the FALCON-PLUS | Date: 13-APR-82 |
|---|---|
| Originator: Charlie Giorgetti | Page 1 of 4 |

There is no hardware support for the EIS, FIS, or FPP instruction sets. For FALCON SBC-11/21 applications that need the ability to perform the EIS instructions MUL, DIV, and ASH, equivalent software routines can be substituted. These callable routines do not do any form of error checking. A user should be aware that extensive use of these software routines for hardware instructions will have impact on system performance. These routines can be incorporated into an application and called as a subroutine. The calling sequence for the subroutines can be set-up in a macro. The following is a list of each of the subroutines and the macros that are used to set-up and call the software MUL, DIV, and ASH routines.

The following macro and subroutine can be used to perform the MUL instruction in software:

```
        .MACRO  SMUL    A,B,HI,LO

        MOV     A,-(SP)     ; Push a multiplier onto the stack
        MOV     B,-(SP)     ; Push the other multiplier as well
        JSR     PC,$MUL     ; Call the MUL subroutine
        MOV     (SP)+,HI    ; Get the most significant part of
                              the result
        MOV     (SP)+,LO    ; Get the least significant part of
                              the result

        .ENDM

$MUL::  MOV     R0,-(SP)    ; Save some work registers
        MOV     R1,-(SP)
        MOV     10(SP),R1   ; Obtain the value of A from the stack
        MOV     #21,-(SP)   ; Initialize the shift counter
        CLR     R0          ; Initialize the high 16-bit accumulator
  10$:  ROR     R0          ; Perform multiplication
        ROR     R1
        BCC     20$
        ADD     10(SP),R0
        CLC
  20$:  DEC     (SP)        ; Bump the shift counter
        BNE     10$         ; Not done ?
        TST     (SP)+       ; Romove the counter from the stack
        MOV     R1,10(SP)   ; Save the low 16-bit value on the stack
        MOV     R0,6(SP)    ; Save the high 16-bit value on the stack
        MOV     (SP)+,R1    ; Restore the work registers
        MOV     (SP)+,R0
        RTS     PC          ; Return
```

The following macro and subroutine can be used to perform the DIV instruction in software:

```
        .MACRO  SDIV    DIVSOR,DIVHI,DIVLO,REM,QUO

        MOV     DIVSOR,-(SP); Push the divisor onto the stack
        MOV     DIVHI,-(SP) ; Push the upper 16-bits of the dividend
        MOV     DIVLO,-(SP) ; Push the lower 16-bits of the dividend
        JSR     PC,$DIV     ; Call the DIV subroutine
        MOV     (SP)+,REM   ; Get the remainder
        MOV     (SP)+,QUO   ; Get the quotient

        .ENDM

$DIV:: MOV      R5,-(SP)    ; Get some work registers
        MOV     R4,-(SP)
        MOV     R3,-(SP)
        MOV     R0,-(SP)
        MOV     14.(SP),R3  ; Get the divisor from the stack
        MOV     12.(SP),R4  ; Get the high 16-bits of the dividend
        MOV     10.(SP),R5  ; as well as low 16-bits
        CLR     R0          ; Clear an accumulator
        MOV     #32.,-(SP)  ; Shift counter
   1$: ASL      R5          ; Perform the division
        ROL     R4
        ROL     R0
        CMP     R0,R3
        BLO     2$
        SUB     R3,R0
        INC     R5
   2$: DEC      (SP)
        BNE     1$          ; Not done ?
        TST     (SP)+       ; Remove the counter from the stack
        MOV     R0,12.(SP)  ; Store the remainder on the stack
        MOV     R5,14.(SP)  ; Store the quotient as well
        MOV     (SP)+,R0    ; Restore the work registers
        MOV     (SP)+,R3
        MOV     (SP)+,R4
        MOV     (SP)+,R5
        MOV     (SP)+,(SP)  ; Update the return PC
        RTS     PC          ; Return
```

The following macro and subroutine can be used to perform the ASH instruction in software:

```
        .MACRO  SASH    COUNT,VAL

        MOV     COUNT,-(SP) ; Push the shift count
        MOV     VAL,-(SP)   ; Push what is to be shifted
        JSR     PC,$ASH     ; Call the ASH subroutine
        MOV     (SP)+,VAL   ; Get the results of the shift

        .ENDM

$SASH:: MOV     R0,-(SP)    ; Get a couple of work registers
        MOV     R1,-(SP)
        MOV     6(SP),R0    ; R0 = value to be shifted
        MOV     8.(SP),R1   ; R1 = direction and shift count
        BIC     #^C<77>,R1
        BEQ     20$         ; Get out if no shifting
        CMP     R1,#31.     ; What direction is the shift
        BGT     10$         ; go to the corection direction shift
  5$:   ASL     R0
        DEC     R1
        BNE     5$
        BR      20$
 10$:   NEG     R1
        BIC     #^C<77>,R1
 11$:   ASR     R0
        DEC     R1
        BNE     11$
 20$:   MOV     R0,8.(SP)   ; Store the shifted result on the stack
        MOV     (SP)+,R1    ; Restore the work registers
        MOV     (SP)+,R0
        MOV     (SP)+,(SP)  ; Update the return PC
        RTS     PC          ; Return
```

| uNOTE # 002 |
|---|

| Title: Block Mode DMA | Date: 01-JUN-83 |
|---|---|
| Originator: Scott Tincher and Mike Collins | Page 1 of 20 |

## What is Block Mode DMA?

Block Mode DMA is a method of data transfer which increases throughput due to the reduced handshaking necessary over the Q-bus. In order to implement Block Mode DMA both the master and slave devices must understand the block Mode protocol. If either device does not have Block Mode capability the transfers proceed via standard DATI or DATO cycles.

## Conventional Direct Memory Access on the Q-bus

Under conventional DMA operations, after a DMA device has become bus master, it begins the data transfers. This is accomplished by gating an address onto the bus followed by the data being transferred to or from the memory device. If more than one transfer is performed by the temporary bus master, the address portion of the cycle must be repeated for each data transfer.

## Block Mode Direct Memory Access on the Q-bus

Under block Mode DMA operations an address cycle is followed by multiple word transfers to sequential addresses. Therefore data throughput is increased due to the elimination of the address portion of each transfer after the initial transfer.

There are two types of block Mode transfer, DATBI (input) and DATBO (output). An overview of what occurs during each type of block Mode transfer is outlined in figures 1 (DATBI, Block Mode input) and 2 (DATBO, block mode output).

In the following discussion the signal prefix T(Transmit) indicates a bus driver input and the signal prefix R(Receive) indicates a bus receiver output.

## DATBI Bus Cycle

Before a DATBI block mode transfer can occur the DMA bus master device must request control of the bus. This occurs under conventional Q-bus protocol.

- o REQUEST BUS

  The bus master device requests control of the bus by asserting TDMR.

- o GRANT BUS CONTROL

  The bus arbitration logic in the CPU asserts the DMA grant signal TDMGO 0 nsec minimum after TDMR is received and 0 nsec minimum after RSACK negates (if a DMA device was previous bus master).

- o ACKNOWLEDGE BUS MASTERSHIP

  The DMA bus master device asserts TSACK 0 nsec minimum after receiving RDMGI, 0 nsec minimum after the negation of RSYNC and 0 nsec minimum after the negation of RRPLY. The DMA bus master device negates TDMR 0 nsec minimum after the assertion of TSACK.

- o TERMINATE GRANT SEQUENCE

  The bus arbitration logic in the CPU negates TDMGO 0 nsec minimum after receiving RSACK. The bus arbitration logic will also negate TDMGO if RDMR negates or if RSACK fails to assert within 10 usec ('no SACK timeout').

o EXECUTE A BLOCK MODE DATBI TRANSFER

  o ADDRESS DEVICE MEMORY

    a) The address is asserted by the bus master on TADDR<21:00> along with the negation of TWTBT.

    b) The bus master asserts TSYNC 150 nsec minimum after gating the address onto the bus.

  o DECODE ADDRESS

    The appropriate memory device recognizes that it must respond to the address on the bus.

  o REQUEST DATA

    a) The address is removed by the bus master from TADDR<21:00> 100 nsec minimum after the assertion of TSYNC.

    b) The bus master asserts the first TDIN 100 nsec minimum after asserting TSYNC.

    c) The bus master asserts TBS7 50 nsec maximum after asserting TDIN for the first time. TBS7 remains asserted until 50 nsec maximum after the assertion of TDIN for the last time. In each case, TBS7 can be asserted or negated as soon as the conditions for asserting TDIN are met.

    The assertion of TBS7 indicates the bus master is requesting another read cycle after the current read cycle.

  o SEND DATA

    a) The bus slave asserts TRPLY 0 nsec minimum (8000 nsec maximum to avoid a bus timeout) after receiving RDIN.

    b) The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDIN after the current RDIN.

NOTE

Block mode transfers must not cross 16 word boundaries

c) The bus slave gates TDATA<15:00> onto the bus 0 nsec minimum after receiving RDIN and 125 nsec maximum after the assertion of TRPLY.

o TERMINATE INPUT TRANSFER

a) The bus master receives stable RDATA<15:00> from 200 nsec maximum after receiving RRPLY until 20 nsec minimum after the negation of RDIN. (The 20 nsec minimum represents total minimum receiver delays for RDIN at the slave and RDATA<15:00> at the master.)

b) The bus master negates TDIN 200 nsec minimum after receiving RRPLY.

o OPERATION COMPLETED

a) The bus slave negates TRPLY 0 nsec minimum after receiving the negation of RDIN.

b) If RBS7 and TREF are both asserted when TRPLY negates, the bus slave prepares for another DIN cycle. RBS7 is stable from 125 nsec after RDIN is received until 150 nsec after TRPLY negates.

c) If TBS7 and RREF were both asserted when TDIN negated, the bus master asserts TDIN 150 nsec minimum after receiving the negation of RRPLY and continues with timing relationship 'SEND DATA' above. RREF is stable from 75 nsec after RRPLY asserts until 20 nsec minimum after TDIN negates. (The 0 nsec minimum represents total minimum receiver delays for RDIN at the slave and RREF at the master.)

NOTE

The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

- TERMINATE BUS CYCLE

  a) If RBS7 and TREF were not both asserted when TRPLY negated, the bus slave removes TDATA<15:00> from the bus 0 nsec minimum and 100 nsec maximum after negating TRPLY.

  b) If TBS7 and RREF were not both asserted when TDIN negated the bus master negates TSYNC 250 nsec minimum after receiving the last assertion of RRPLY and 0 nsec minimum after the negation of that RRPLY.

- RELEASE THE BUS

  a) The DMA bus master negates TSACK 0 nsec after negation of the last RRPLY.

  b) The DMA bus master negates TSYNC 300 nsec maximum after it negates TSACK.

  c) The DMA bus master must remove RDATA<15:00>, TBS7, and TWTBT from the bus 100 nsec maximum after clearing TSYNC.

- RESUME PROCESSOR OPERATION The bus arbitration logic in the CPU enables processor-generated TSYNC or will issue another bus grant (TDMGO) if RDMR is asserted.

Figure 1 - DATBI CYCLE

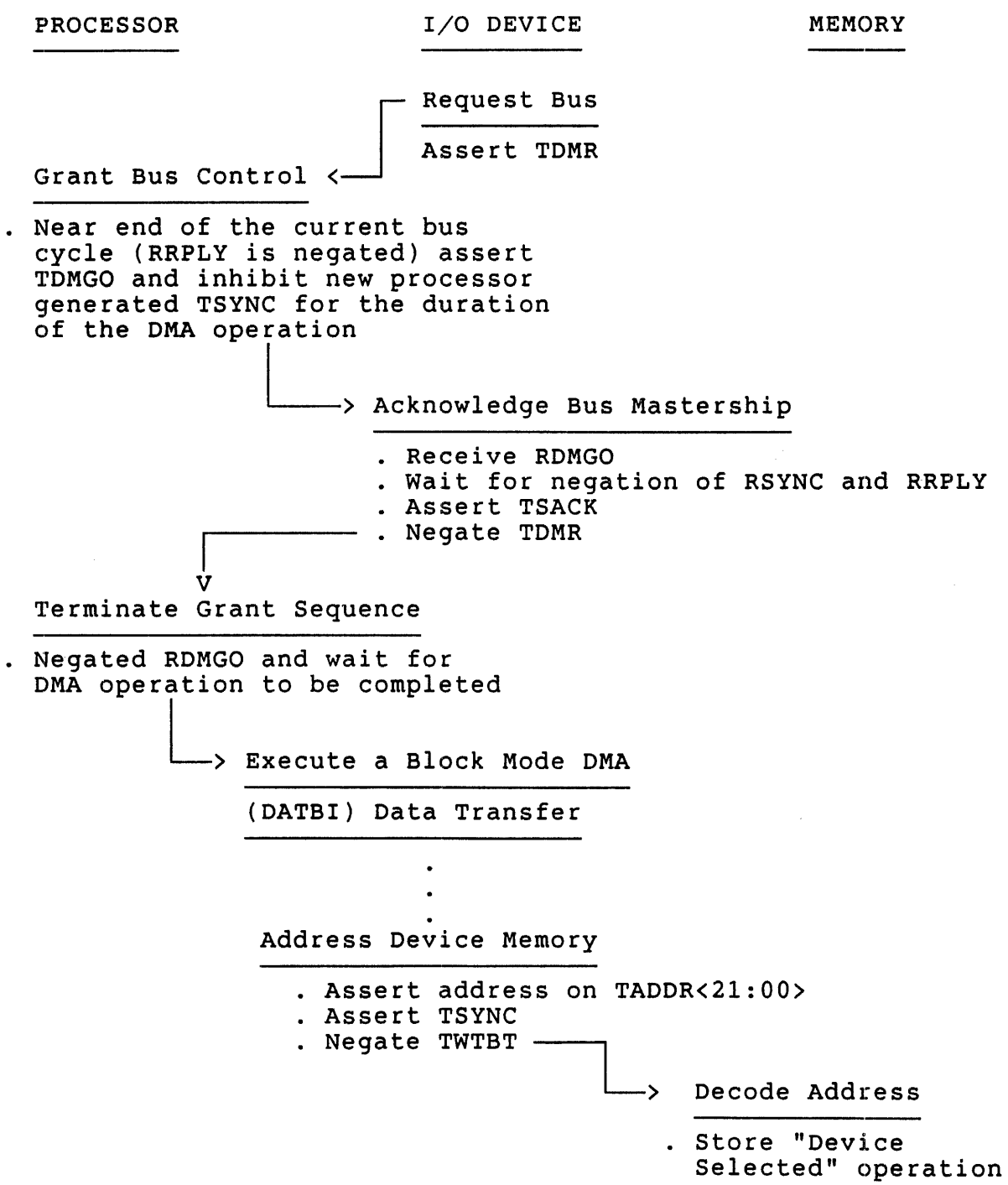


Figure 1 - DATBI CYCLE (continued)

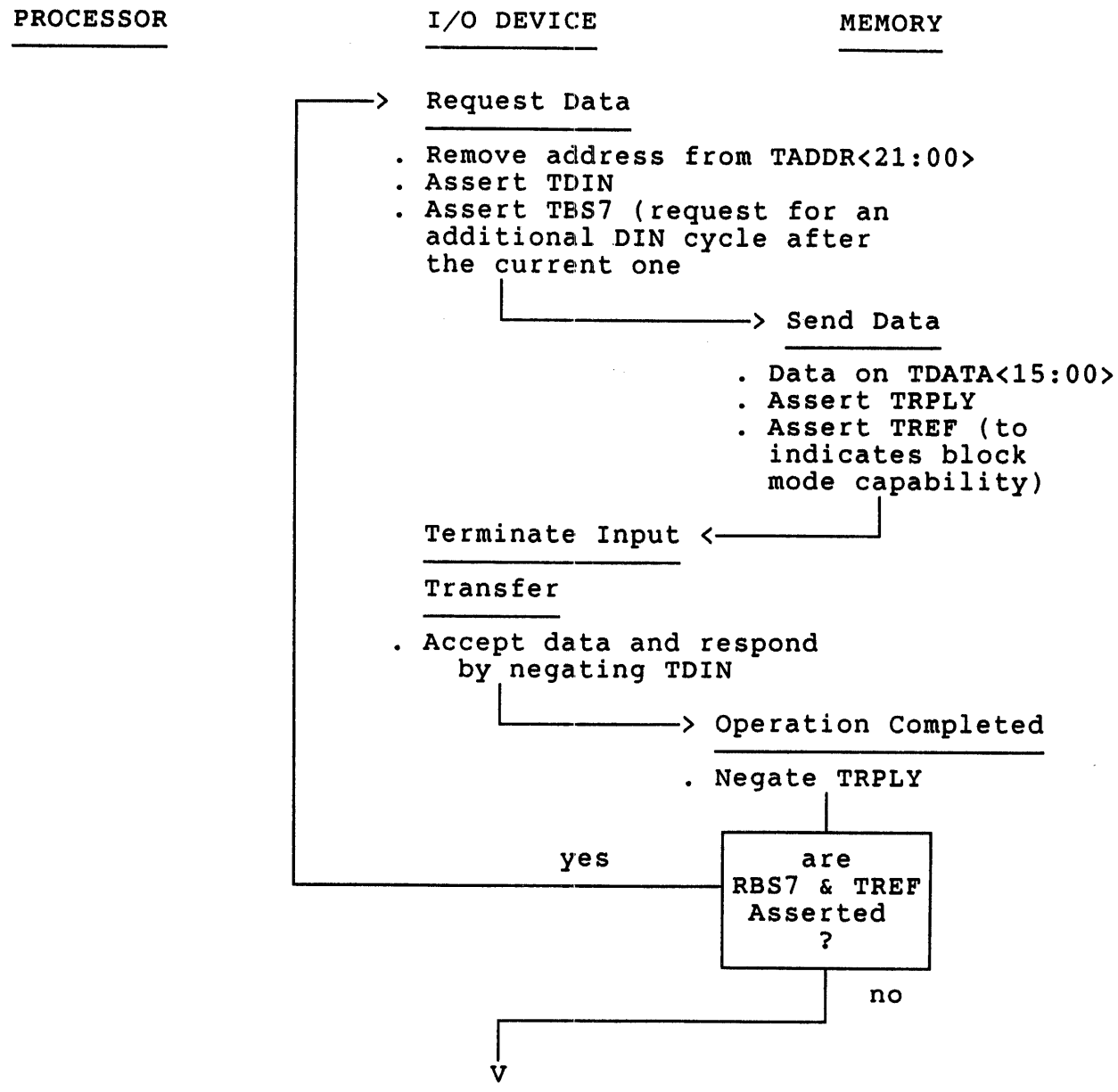
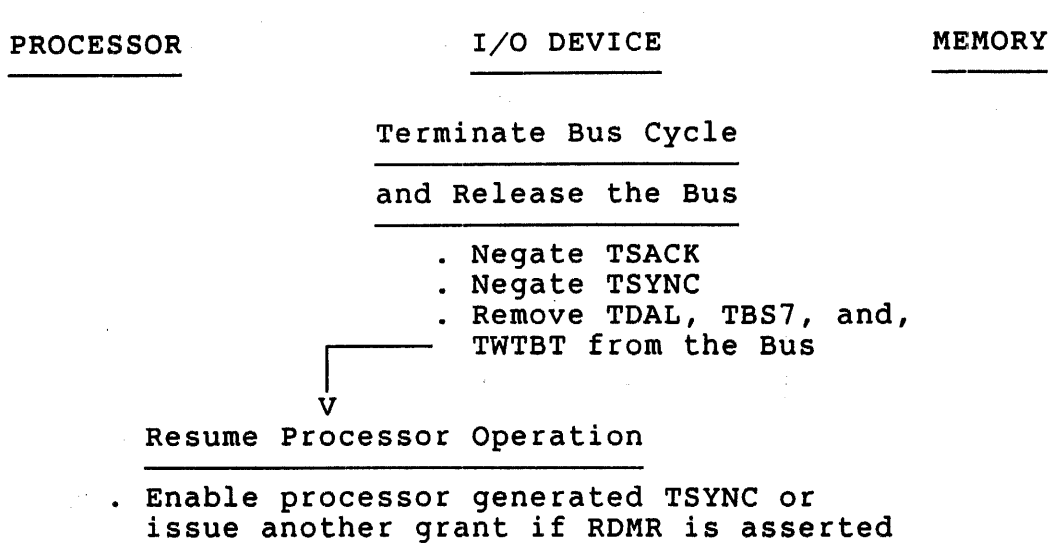
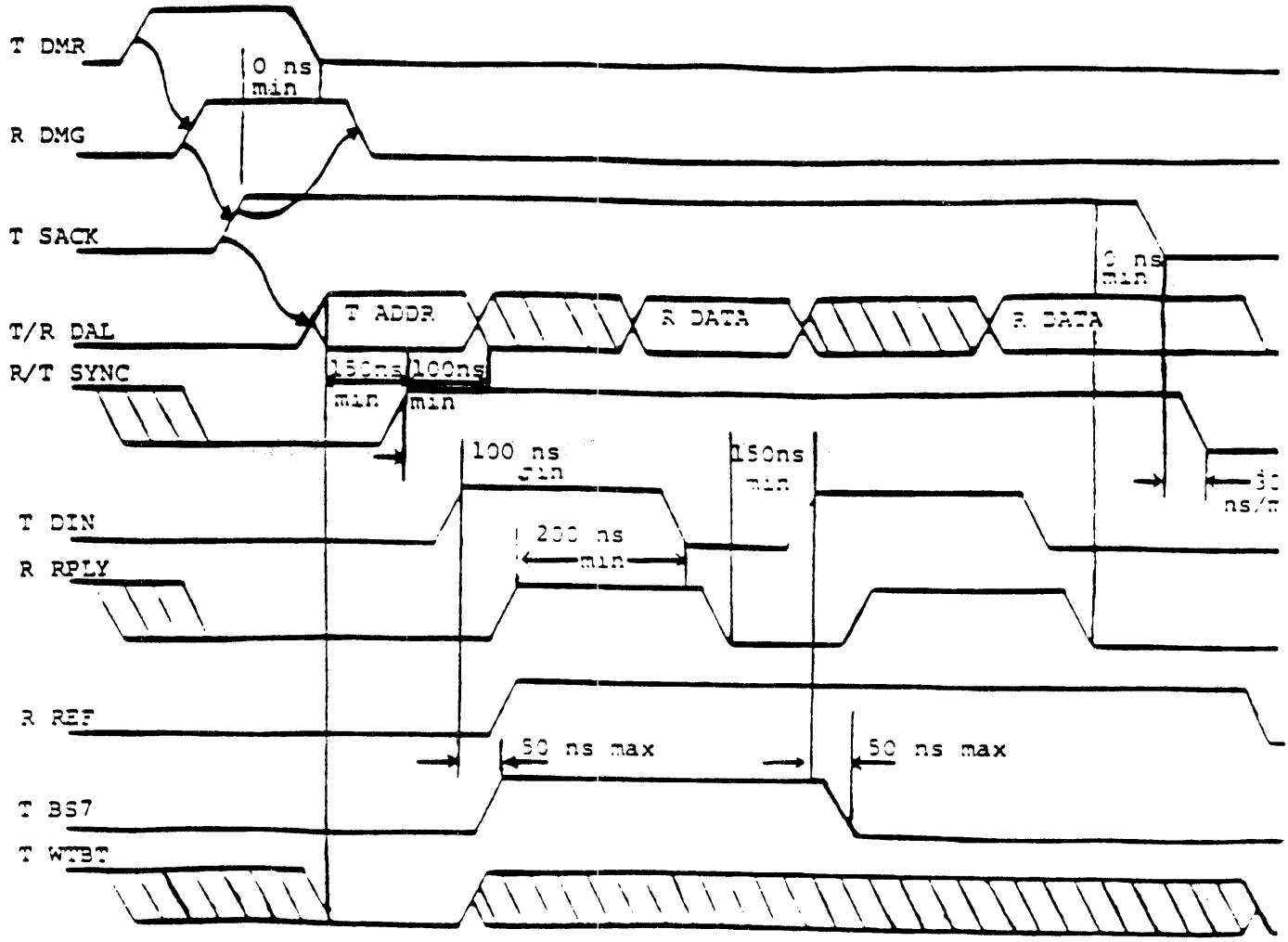


Figure 1 - DATBI CYCLE (continued)

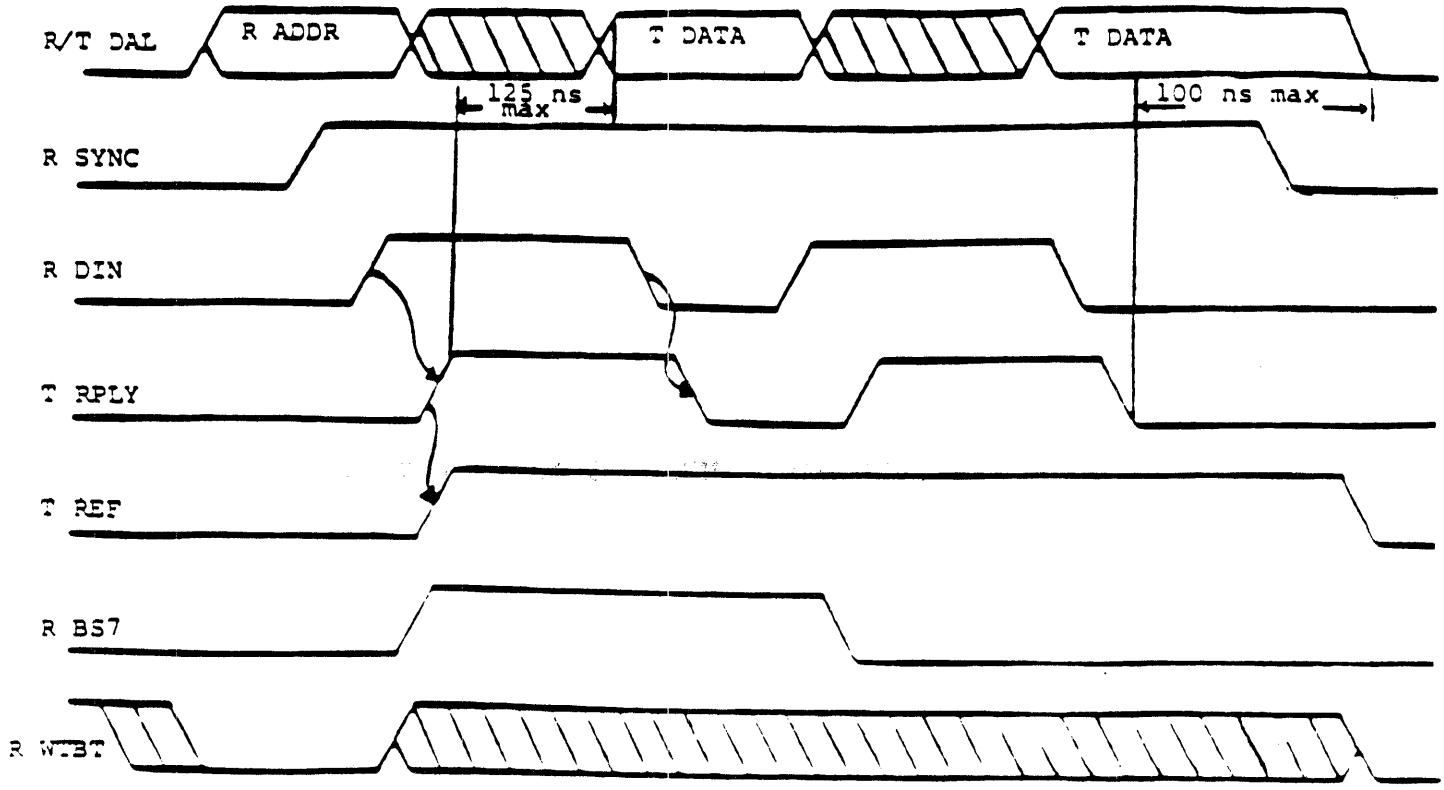





Timing at slave device.
T = bus driver input
R = Bus receiver output

DATBI

THIS PAGE INTENTIONALLY LEFT BLANK




Timing at slave device.
T = bus driver input
R = Bus receiver output

DATE

THIS PAGE INTENTIONALLY LEFT BLANK

## DATBO Bus Cycle

DATBO Bus cycles Before a block mode transfer can occur the DMA bus master device must request control of the bus. This occurs under conventional Q-bus protocol.

- o REQUEST BUS The bus master device requests control of the bus by asserting TDMR.

- o GRANT BUS CONTROL The bus arbitration logic in the CPU asserts the DMA grant signal TDMGO 0 nsec minimum after RDMR is received and 0 nsec minimum after TSACK negates (if a DMA device was previous bus master).

- o ACKNOWLEDGE BUS MASTERSHIP The DMA bus master device asserts TSACK 0 nsec minimum after receiving RDMGI, 0 nsec minimum after the negation of RSYNC and 0 nsec minimum after the negation of RRPLY. The DMA bus master device negates TDMR 0 nsec minimum after the assertion of TSACK.

- o TERMINATE GRANT SEQUENCE The bus arbitration logic in the CPU negates TDMGO 0 nsec minimum after receiving RSACK. The bus arbitration logic will also negate TDMGO if RDMR negates or if RSACK fails to assert within 10 usec ('no SACK timeout').

- o EXECUTE A BLOCK MODE DATBO TRANSFER

  - o ADDRESS DEVICE MEMORY

    a) The address is asserted by the bus master on TADDR<21:00> along with the assertion of TWTBT.

    b) The bus master asserts TSYNC 150 nsec minimum after gating the address onto the bus.

  - o DECODE ADDRESS The appropriate memory device recognizes that it must respond to the address on the bus.

  - o SEND DATA

    a) The bus master gates TDATA<15:00> along with TWTBT 100 nsec minimum after the assertion of TSYNC. TWTBT is negated.

    b) The bus master asserts the first TDOUT 100 nsec minimum after gating TDATA<15:00>.

NOTE

During DATBO cycles TBS7 is undefined

o RECEIVE DATA

a) The bus slave receives stable data on RDATA<15:00> from 25 nsec minimum before receiving RDOUT until 25 nsec minimum after receiving the negation of RDOUT.

b) The bus slave asserts TRPLY 0 nsec minimum after receiving RDOUT.

c) The bus slave asserts TREF concurrent with TRPLY if, and only if, it is a block mode device which can support another RDOUT after the current RDOUT.

NOTE

Blockmode transfers must not cross 16 word boundaries

o TERMINATE OUTPUT TRANSFER The bus master negates TDOUT 150 nsec minimum after receiving RRPLY.

o OPERATION COMPLETED

a) The bus slave negates TRPLY 0 nsec minimum after receiving the negation of RDOUT.

b) If RREF was asserted when TDOUT negated and if the bus master wants to transfer another word, the bus master gates the new data on TDATA<15:00> 100 nsec minimum after negating TDOUT. RREF is stable from 75 nsec maximum after RRPLY asserts until 20 nsec minimum after RDOUT negates. (The 20 nsec minimum represents minimum receiver delays for RDOUT at the slave and RREF at the master).

c) The bus master asserts TDOUT 100 nsec minimum after gating new data on TDATA<15:00> and 150 nsec minimum after receiving the negation of RRPLY. The cycle continues with the timing relationship in 'RECEIVE DATA' above.

NOTE

The bus master must limit itself to not more than eight transfers unless it monitors RDMR. If it monitors RDMR, it may perform up to 16 transfers as long as RDMR is not asserted at the end of the seventh transfer.

o TERMINATE BUS CYCLE

a) If RREF was not asserted when RRPLY negated or if the bus master has no additional data to transfer, the bus master removes data on TDATA<15:00> from the bus 100 nsec minimum after negating TDOUT.

b) If RREF was not asserted when TDOUT negated the bus master negates TSYNC 275 nsec minimum after receiving the last RRPLY and 0 nsec minimum after the the negation of the last RRPLY.

- RELEASE THE BUS

  a) The DMA bus master negates TSACK 0 nsec after negation of the last RRPLY.

  b) The DMA bus master negates TSYNC 300 nsec maximum after it negates TSACK.

  c) The DMA bus master must remove TDATA, TBS7, and TWTBT from the bus 100 nsec maximum after clearing TSYNC.

- RESUME PROCESSOR OPERATION The bus arbitration logic in the CPU enables processor-generated TSYNC or will issue another bus grant (TDMGO) if RDMR is asserted.

Figure 2 - DATBO CYCLE

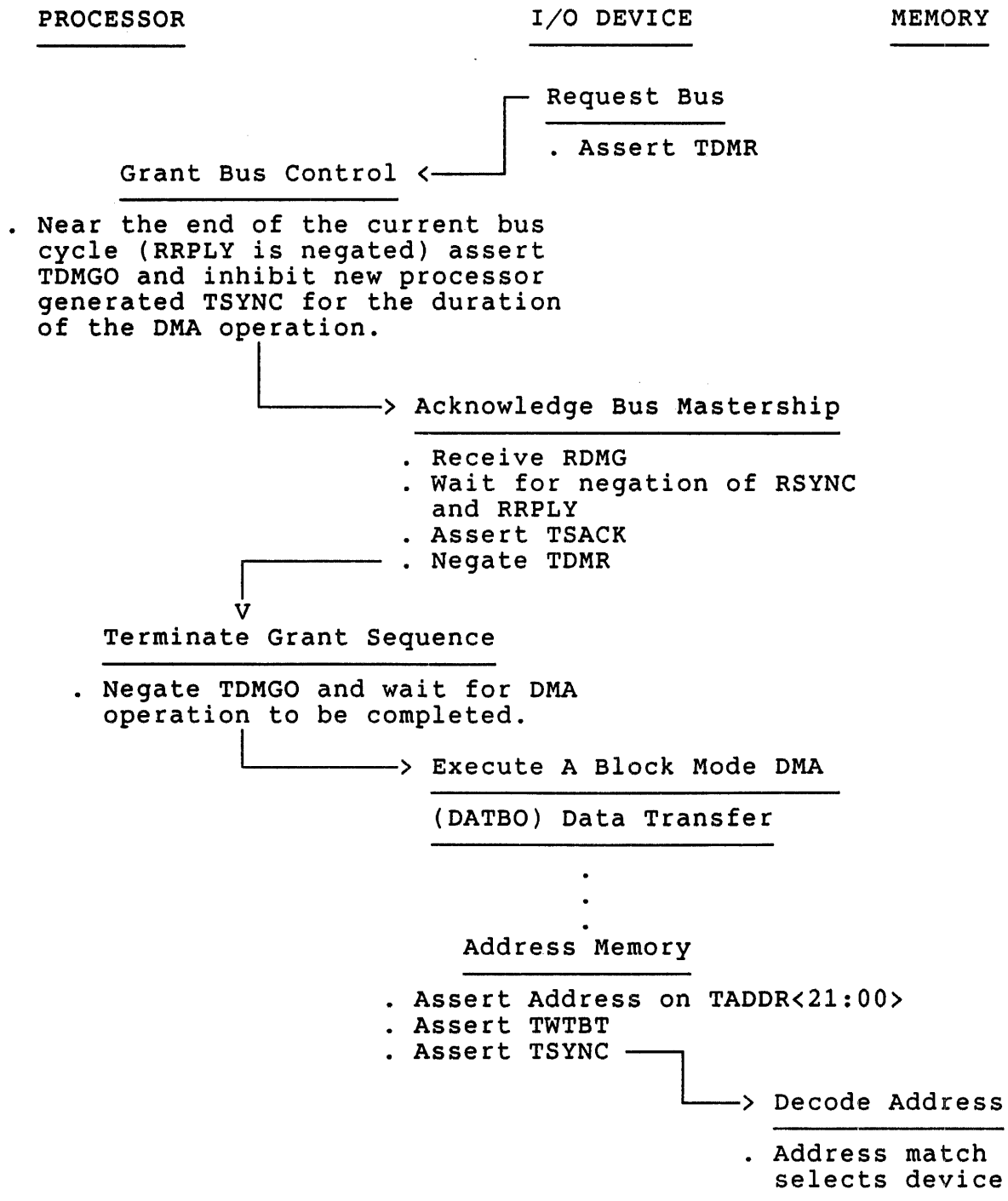
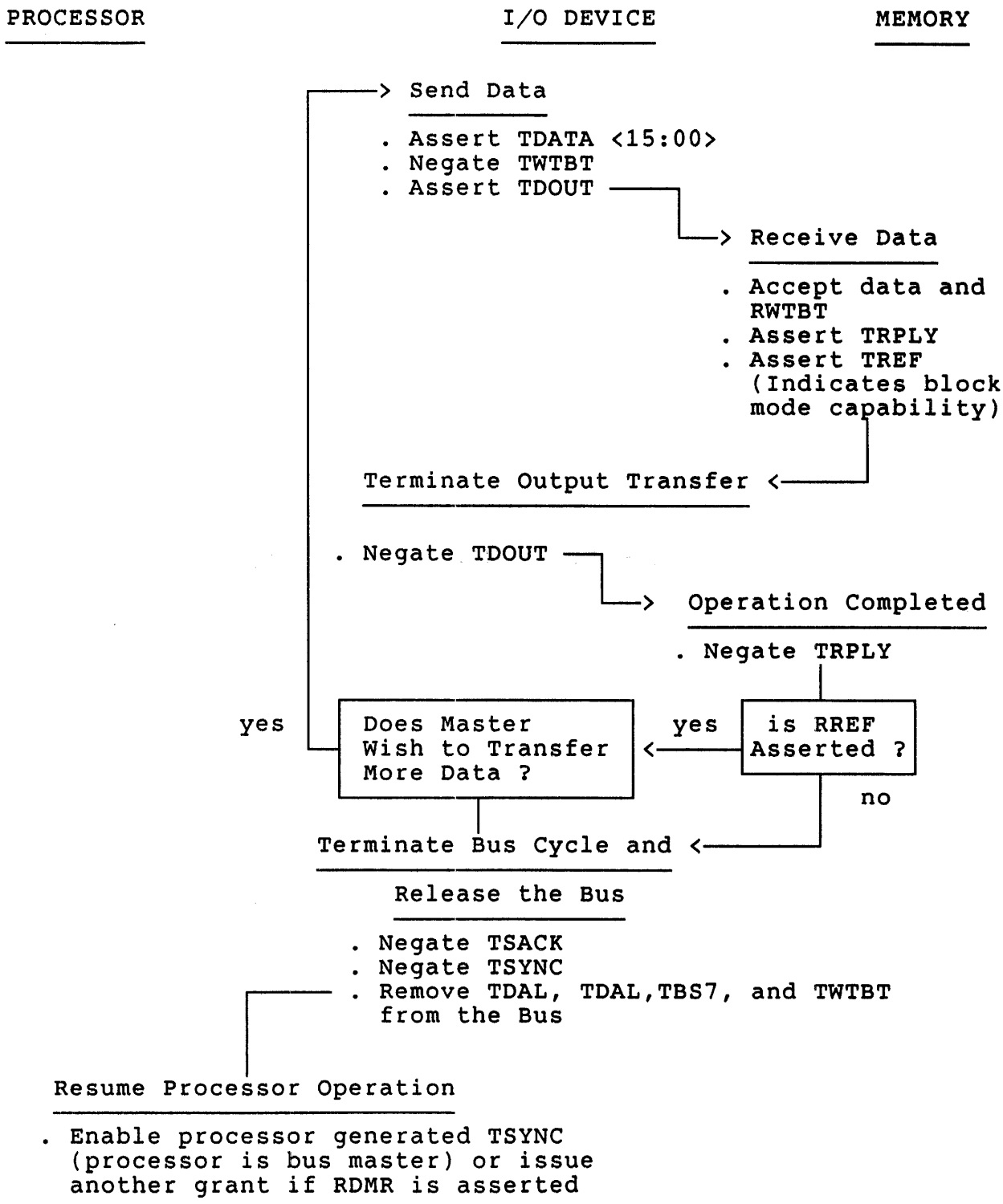
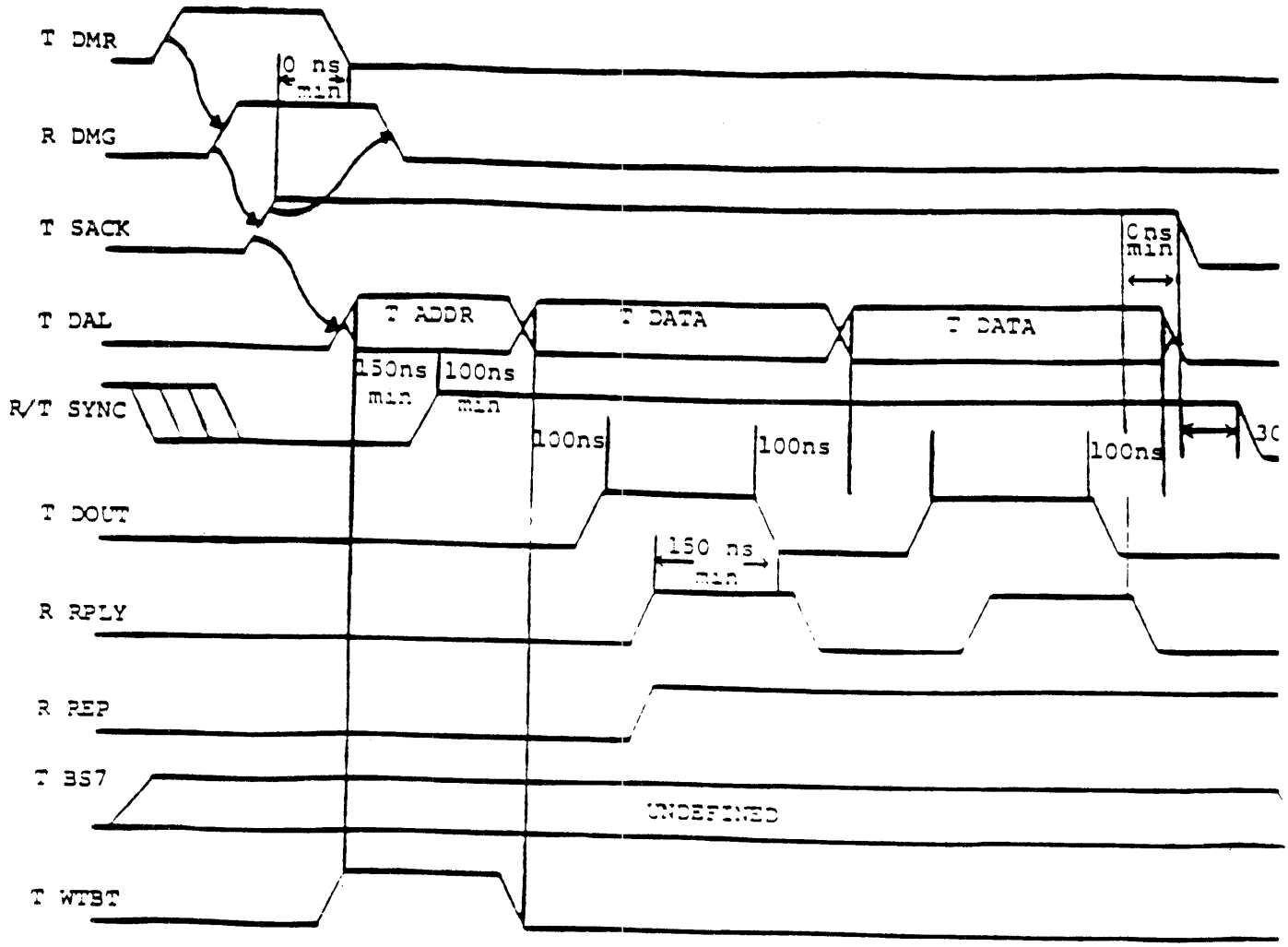


Figure 2 - DATBO CYCLE (continued)

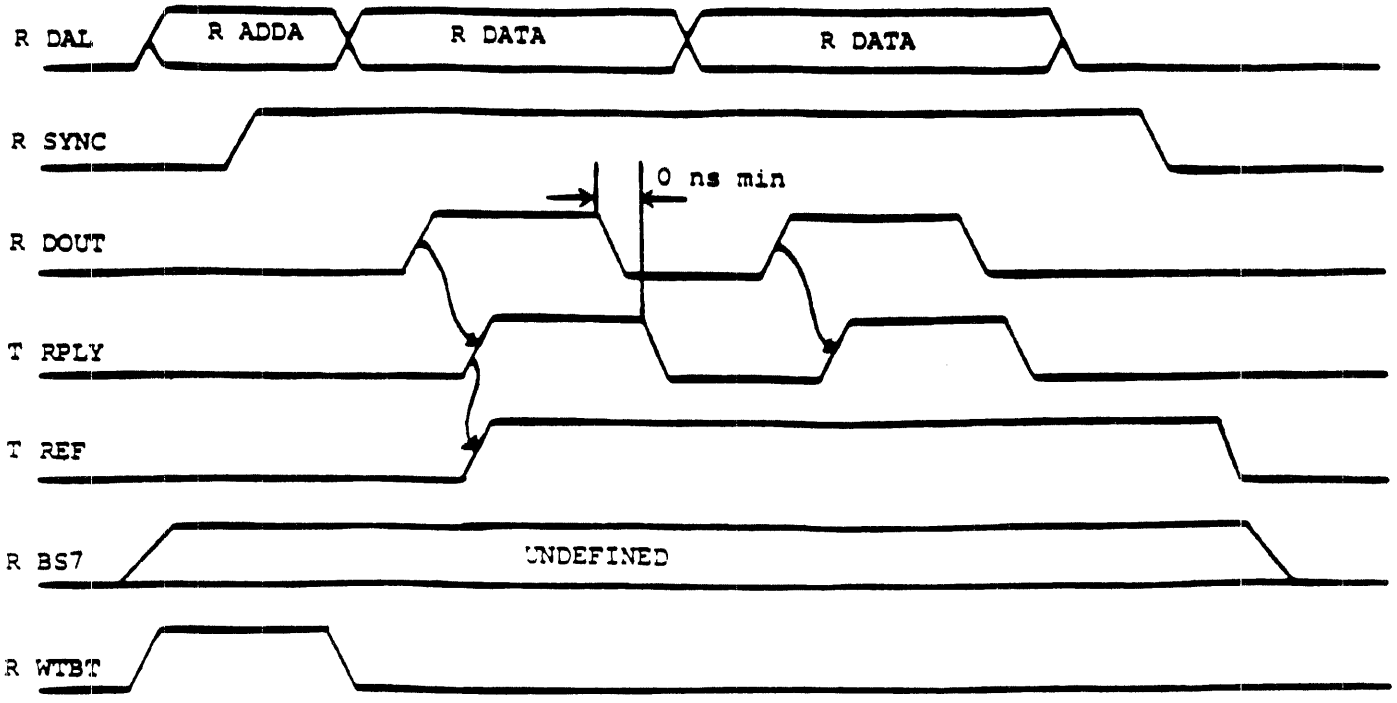


THIS PAGE INTENTIONALLY LEFT BLANK




Timing at master device.
T = Bus driver input
R = Bus receiver output

DATBO




Timing at slave device.
T = Bus driver input
R = Bus receiver output

DATBO

| Title: Compatible Bootstraps for the LSI-11/73 | Date: 28-NOV-83 |
|---|---|
| Originator: Mike Collins | Page 1 of 2 |

The LSI-11/73 (KDJ11-AA) is a high performance CPU for the Q-Bus. It is a CPU only, which means that there is no boot capability on the module itself. Therefore a boot module must be selected to work with the LSI-11/73.

This uNOTE will discuss the bootstrap modules which can be used with the 11/73.

There are 4 possible modules which can be used for bootstrap.
They are : MXV11-BF w/MXV11-B2 boot ROMs
MRV11-D w/MXV11-B2 boot ROMs
MXV11-AA or -AC w/MXV11-A2 boot ROMs
BDV11

For an LSI-11/73 based system to be Field Serviceable the bootstrap code must execute a cache memory diagnostic on power-up. The only boot code which satisfies this requirement is found in the MXV11-B2 boot ROMs. Therefore an LSI-11/73 based, Field Serviceable system must use either the MXV11-BF w/MXV11-B2 ROMs or the MRV11-D w/MXV11-B2 ROMs.

NOTE

The MXV11-B2 ROMs will not work on the MXV11-A module.

## MXV11-BF or MRV11-D w/MXV11-B2 ROMs

The MXV11-BF w/MXV11-B2 ROMs is the preferred choice since this module has 2 asynchronous serial lines as well as 128Kb of dynamic RAM in addition to the boot capability. However, if your application does not need the extra serial lines and RAM, an alternate choice would be the MRV11-D w/MXV11-B2 ROMs.

The MXV11-B2 ROMs will boot the following devices :
RLO1 / RLO2 (DL)
RXO1 / RXO2 (DX,DY)
TU58 (DD)
TSV05 (MS)
MSCP type Devices e.g. RD51, RX50 (DU)
DECnet via DPV11, DLV11-E, DLV11-F, DUV11

NOTE

The MXV11-BF is not supported by RSTS due to its non-parity memory. An alternative configuration would be to use the MRV11-D with the MXV11-B2 boot ROMs, and a DLV11-J or other DLV11 serial line device.

The remaining 2 boot modules do NOT have the necessary cache memory diagnostic code to make an 11/73 based system Field Serviceable.

Below is a list of all of the KNOWN WORKING bootstraps for the 2 remaining boot modules.

MXV11-A w/MXV11-A2 ROMs

Working bootstraps : RLO1 / RLO2
RXO1 / RXO2
TU58 conventional boot
TU58 standalone boot

WARNING

If the MXV11-A is used in a 22 bit system the on-board RAM must be disabled. Refer to uNOTE #106.

BDV11

Working bootstraps : RLO1 / RLO2
RXO2
RK05

WARNING

Disable the processor and memory tests since an odd address trap does occur in each of them. See NOTE below. To disable the CPU test, set switch E15-1 to OFF. To disable the memory test, set switch E15-2 to OFF. (Refer to the Microcomputer and Interfaces Handbook for complete configuration information.)

The 11/73 has an on-board Line Time Clock Register, therefore the BDV11 BEVNT switch E21-5 should be set to the OPEN position. This disables software control of the BEVNT signal via the BDV11 LTC register and allows software control of this signal via the 11/73 LTC register.

If the BDV11 is used in a 22 bit system, it must be CS REV E or later or ECO M8012-ML0005 must be installed.

NOTE

ODD ADDRESS TRAPS. The 11/23 ignores an odd address reference whereas the KDJ11-A will trap to address 4.

| uNOTE #004 |
|---|

| Title: LSI-11/73 Upgrade Paths | Date: 28-NOV-83 |
|---|---|
| Originator: Mike Collins | Page 1 of 6 |

With the announcement of the KDJ11-A CPU module, there will be numerous questions regarding configuring the module into a current system. The purpose of this MicroNote is to address all possible configuration upgrade paths (within reason).

Generally a KDJ11-A will be installed as an upgrade to a system built from components or DEC packaged system.

In the case of a component upgrade it is assumed the processor is a KDF11-A and the boot mechanism is an MXV11-A with the MXV11-A2 Boot ROMs.

System upgrades fall into 2 categories:

1. KDF11-A based systems and
2. KDF11-B based systems (11/23+ and Micro/PDP-11)

There are 3 issues which must be addressed when considering a KDJ11-A upgrade. They are:

1. The Boot mechanism
2. 18 or 22 bit system
3. Single or multiple box system

NOTE

1. In the following upgrade scenarios, the systems have been labeled as being Field Serviceable or not. A system which is Field Serviceable has a bootstrap which meets Field Service requirements. The requirement is that the bootstrap must execute an 11/73 cache memory diagnostic on power-up. There is no guarantee that the overall system will be Field Serviceable or that it will be FCC compliant.

2. Systems using CPU's other than the KDF11-A or KDF11-B (i.e. 11/03 systems) are not considered for upgrade.

CAUTION: It is recommended that the AC and DC loading for the final configuration be checked for conformance with the Q-Bus loading rules.

It is also recommended to check for overloading on the +5 Volt and +12 Volt Power Supplies.

For each system upgrade the following parameters are listed for both the 'Current' system and the 'Upgraded' system:

1. CPU
2. Boot Mechanism
3. System Size
4. Number of Boxes
5. Field Serviceable or not
6. Special Conditions

COMPONENT UPGRADE PATHS:

1. Current System

KDF11-A
MXV11-A
18 Bit System
1 Box

Upgrade 1

KDJ11-A
MXV11-B/MRV11-D with MXV11-B2 Boot ROMs
18 Bit System
1 Box
Field Serviceable

Upgrade 2

KDJ11-A
MXV11-A
18 Bit System
1 Box
NOT Field Serviceable

2. Current System

KDF11-A
MXV11-A
18 Bit System
More than 1 box

Upgrade

See upgrades for category #1

3. Current System

KDF11-A
MXV11-A (Memory Disabled)
22 Bit System
1 Box

Upgrade

See upgrades for category #1

4. Current System

KDF11-A
MXV11-A (Memory Disabled)
22 Bit System
More than 1 box
This system is not currently configureable with DEC equipment.

Upgrade

Not currently configureable with DEC equipment.

PDP 11/23A SYSTEM UPGRADE PATHS:

5. Current System

KDF11-A
BDV11
18 Bit System
1 Box

Upgrade 1

KDJ11-A
MXV11-B/MRV11-D with MXV11-B2 Boot ROMs
18 Bit System
1 Box
Field Serviceable

Upgrade 2

KDJ11-A
BDV11
18 Bit System
1 Box
NOT Field Serviceable
Disable the Processor and Memory tests and also the BEVNT register on the BDV11.

Upgrade 3

KDJ11-A
MXV11-A (with MXV11-A2 boot ROMs)
18 Bit System
1 Box System
NOT Field Serviceable
Check AC loading since termination was removed when the BDV11 was removed from the system.

6. Current System

KDF11-A
BDV11
18 Bit System
More than 1 Box

Upgrade 1

KDJ11-A
MXV11-B/MRV11-D with MXV11-B2 Boot ROMs
18 Bit System
More than 1 box
Field Serviceable
Use BCV1A and BCV1B expansion cables.

Upgrade 2

KDJ11-A
BDV11
18 Bit System
More than 1 Box
NOT Field Serviceable
Disable the Processor and Memory tests and also the BEVNT register on the BDV11.

Use BCV1B cable set between 1st and 2nd box and the BCV1A cable set between the 2nd and 3rd box. Note: If in a 3 box system the expansion cable set lengths must differ by 4 ft.

Upgrade 3

KDJ11-A
MXV11-A (with MXV11-A2 boot ROMs)
18 Bit System
More than 1 Box
NOT Field Serviceable
Use BCV1A and BCV1B expansion cables.

7. Current System

KDF11-A
BDV11
22 Bit System
1 Box
Systems with this configuration were never shipped by DEC.

PDP 11/23 PLUS SYSTEM UPGRADE PATHS:

8. Current System

KDF11-B
Boot is on CPU
22 Bit System
1 Box System

Upgrade 1

KDJ11-A
MXV11-B/MRV11-D with MXV11-B2 Boot ROMs
22 Bit System
1 Box
Field Serviceable

Upgrade 2

KDJ11-A
MXV11-A (with MXV11-A2 boot ROMs)
22 Bit System
1 Box
NOT Field Serviceable
Must disable RAM on MXV11-A.

Upgrade 3

KDJ11-A
BDV11
22 Bit System
1 Box System
NOT Field Serviceable
Must have BDV11 ECO M8012-ML005 installed. Disable the Processor and Memory tests and also the BEVNT register on the BDV11.

9. Current System

KDF11-B
Boot is on CPU
22 Bit System
More than 1 Box

Upgrade 1

Not currently configureable with DEC equipment.

Upgrade 2

Not currently configureable with DEC equipment.

Upgrade 3

Not currently configureable with DEC equipment.

MICRO/PDP-11 SYSTEM UPGRADE PATHS:

10. Current System

Micro/PDP-11
KDF11-BE
Boot is on CPU
22 Bit System
1 Box system

Upgrade

Same as 11/23+ rules, see category #8, Upgrade 1. Upgrades 2 and 3 are not recommended since the MXV11-A and BDV11 cannot boot the 5 1/4" media in the Micro/PDP-11.

11. Current System

Micro/PDP-11
KDF11-BE
Boot is on CPU
22 Bit System
More than 1 box

Upgrade

Same as 11/23+ rules, see upgrades for category #9.

NOTE

It is not currently possible to expand out of the Micro/PDP-11 while maintaining FCC compliance.

11/23 PLUS and Micro/PDP-11 system upgrades will require an EXTRA backplane slot to accomodate the additional boot module (i.e. MXV11-A,-B or BDV11).

## 11/23-S SYSTEM UPGRADE SOLUTIONS:

| 12. Current System | Upgrade |
|---|---|
| KDF11-BA<br>Boot is on CPU<br>18 Bit System<br>1 Box system | See upgrades for category #5. |

| 13. Current System | Upgrade |
|---|---|
| KDF11-BA<br>Boot is on CPU<br>18 Bit system<br>More than 1 box | See upgrades for category #6. |

NOTE

It is not currently possible to expand out of the 11/23-S while maintaining FCC compliance.

| uNOTE # 005 |
|---|

| Title: Q22 Compatible Options | Date: 23-Apr-84 |
|---|---|
| Originator: Charlie Giorgetti | Page 1 of 6 |

This is a list of Q22 compatible options. A Q22 compatible option is defined as a Q-bus option that will work without restriction in an extended Q-bus system, that is a 22-bit Q-bus system. This list also includes options that are not compatible in Q22 systems and the reason for the restriction.

The requirements for a device to be Q22 compatible are the following:

1. Processors, memories, and DMA devices must all be capable of 22-bit addressing.

2. Devices must use backplane pins BC1, BD1, BE1, BF1 and DC1, DD1, DE1, DF1, for BDAL18-21 only.

Processors, memories, or DMA devices which are not capable of 22-bit addressing may generate or decode erroneous addresses if they are used in systems which implement 22-bit addressing. Memory and memory-addressing devices which implement only 16 or 18-bit addressing may be used in a 22-bit backplane, but the size of the system memory must be restricted to the address range of those devices (64 KB for systems with 16-bit devices and 256 KB for systems with an 18-bit devices).

Any device which uses backplane pins BC1, BD1, BE1, BF1 or DC1, DD1, DE1, DF1, for purposes other than BDAL18-21 is electrically incompatible with the 22-bit bus and may not be used without modification.

NOTE

Eighteen or sixteen bit DMA devices can potenitially work in Q22 systems by buffering I/O in the 18- or 16-bit address space.

I. Fully Compatible Options

Options in this category meet both of the requirements mentioned above and may be used in any Q-bus configuration.

A. Processors

| | | |
|---|---|---|
| KD32-A | M7135/M7136 | MicroVAX I CPU Module |

| | | |
|---|---|---|
| KDF11-A | M8186 | LSI-11/23 CPU<br>(Etch Rev. C or later) |
| KDF11-B | M8189 | LSI-11/23B CPU |
| KDJ11-A | M8192 | LSI-11/73 CPU |
| KDJ11-B | M8190 | MicroPDP-11/73 CPU |
| KXT11-C | M8377 | Q-bus Perpherial I/O Processor |
| KMV11-A | M7500 | Q-bus Perpherial Communication Processor |

B. Backplanes/Boxes

| | | |
|---|---|---|
| | H9270-Q | 4 X 4 Q22/Q22 Backplane |
| | H9281-QA | 2 X 4 Q22 Dual-height Backplane |
| | H9281-QB | 2 X 8 Q22 Dual-height Backplane |
| | H9281-QC | 2 X 12 Q22 Dual-height Backplane |
| | H9275 | 4 X 9 Q22/Q22 Backplane |
| BA11-S | H9276 | 4 X 9 Q22/CD Backplane |
| Micro/PDP-11 | H9278 | 4 X 3 Q22/CD and 4 X 5 Q22/Q22 Backplane |

C. Memory

| | | |
|---|---|---|
| MCV11-D | M8631 | CMOS Non-volatile Memory |
| MSV11-L | M8059 | MOS Memory (either 128 KB or 256 KB) |
| MSV11-P | M8067 | MOS Memory (either 256 KB or 512 KB) |
| MSV11-Q | M7551 | MOS Memory (1 MB) |
| MXV11-B | M7195 | Multifunction Module |
| MRV11-D | M8578 | PROM/ROM Module |

D. Options

| | | |
|---|---|---|
| AAV11-C | A6006 | D/A Converter |
| ADV11-C | A8000 | A/D Converter |
| AXV11-C | A0026 | D/A and A/D Combination Converter |
| BDV11 | M8012 | Bootstrap, Terminator, Diagnostic<br>(CS Rev. E or later, ECO M8012-ML005 installed) |

| | | |
|---|---|---|
| DEQNA | M7504 | Ethernet Controller |
| DLV11 | M7940 | Asynchronous Serial Line Interface |
| DLV11-E | M8017 | Asynchronous Serial Line Interface |
| DLV11-F | M8028 | Asynchronous Serial Line Interface |
| DLV11-J | M8043 | Four Asynchronous Serial Line Interfaces (CS Rev. E or later, ECO M8043-MR002 installed) |
| DHV11 | M3104 | 8-line Asynchronous EIA Multiplexer |
| DMV11-AD | M8053-MA | Synchronous Communications Interface |
| DMV11-AF | M8064-MA | Synchronous Communications Interface |
| DPV11 | M8020 | Programmable Synchronous EIA Line |
| DRV11 | M7941 | 32 line Parallel Interface |
| DRV11-J | M8049 | 64 line Parallel Interface |
| DRV11-W | M7651 | General Purpose DMA Interface (dual) |
| DUV11 | M7951 | Programmable Synchronous EIA Line |
| DZQ11 | M3106 | 4-line Asynchronous EIA Multiplexer (dual) |
| DZV11 | M7957 | 4-line Asynchronous EIA Multiplexer (quad) |
| FPF11 | M8188 | Floating Point Processor |
| IBV11-A | M7954 | IEEE Instrument Bus Interface |
| IEQ11 | M8634 | DMA IEEE Instrument Bus Interface |
| KLESI-QA | M7740 | LESI Bus Adaptor (RC25 Interface) |
| KPV11-A | M8016 | Power-fail and LTC Generator (KPV11-B and -C are not compatible) |
| KWV11-C | A4002 | Programmable Real-time Clock |
| LAV11 | M7949 | LA180 Line Printer Interface |
| LPV11 | M8027 | LA180/LP05 Printer Interface |
| RLV12 | M8061 | RL01/2 Controller |
| RQDX1 | M8639 | Controller for 5.25" Floppy and Winchester |

| | | |
|---|---|---|
| RXV11 | M7946 | RX01 Floppy Disk Interface |
| TQK25 | M7605 | Streaming Cartridge Tape Controller |
| TSV05 | M7196 | Magnetic Tape Controller |

E. Bus Cable-Cards

| | |
|---|---|
| M9404 | Cable Connector |
| M9404-YA | Cable Connector with 240-Ohm Terminators |
| M9405 | Cable Connector |
| M9405-YA | Cable Connector with 120-Ohm Terminators |

II. Restricted Compatibility Options

Options in this category do not meet one or both of the requirements for use in a 22-bit system. These options are incompatible with some or all 22-bit systems.

A. Processors

| | | |
|---|---|---|
| KDF11-A | M8186 | LSI-11/23 CPU<br>(Prior to etch rev. C, 18-bit addressing only, and use of BC1,BD1,BE1,BF1 for purposes other than BDAL18-21) |
| KD11-HA | M7270 | LSI-11/2 CPU<br>(16-bit addressing only, and use of BC1,BD1, BE1,BF1 for purposes other than BDAL18-21) |
| KD11-F | M7264 | LSI-11 CPU<br>(16-bit addressing only, and use of DC1,DB1, DE1,DF1 for purposes other than BDAL18-21) |
| KXT11-A | M8063 | SBC-11/21 CPU<br>(16-bit addressing only) |

B. Backplanes/Boxes

| | | |
|---|---|---|
| DDV11-B | | 6 X 9 Backplane<br>(18-bit addressing only) |
| BA11-M | H9270 | 4 X 4 Backplane<br>(18-bit addressing only) |
| BA11-N | H9273-A | 4 X 9 Backplane<br>(18-bit addressing only) |

| | | |
|---|---|---|
| BA11-VA | H9281-A,B,C | 2 X n Dual-height Backplane n = 4, 8, and 12<br>BA11-VA used the H9281-A<br>(18-bit addressing only) |
| VT103 | | 4 X 4 Backplane (part number: 54-14008)<br>(18-bit addressing only) |

C. Memories

| | | |
|---|---|---|
| MMV11-A | G653 | 8 KB Core Memory<br>(16-bit addressing only, Q-bus required on C/D backplane connectors) |
| MRV11-AA | M7942 | ROM Module<br>(16-bit addressing only) |
| MRV11-BA | M8021 | UV PROM-RAM<br>(16-bit addressing only) |
| MRV11-C | M8048 | PROM/ROM Module<br>(18-bit addressing only) |
| MSV11-B | M7944 | 8 KB bus refreshed RAM<br>(16-bit addressing only) |
| MSV11-C | M7955 | 32 KB RAM<br>(18-bit addressing only) |
| MSV11-D,E | M8044/M8045 | 8 KB, 16 KB, 32 KB, 64 KB RAM<br>(18-bit addressing only) |
| MXV11-A | M8047 | Multifunction Module<br>(18-bit addressing only on memory, the memory can be disabled) |

D. Options

| | | |
|---|---|---|
| AAV11 | A6001 | D/A Converter<br>(Use of BC1 for purposes other than BDAL18) |
| ADV11 | A012 | A/D Converter<br>(Use of BC1 for purposes other than BDAL18) |
| BDV11 | M8012 | Bootstrap/Terminator<br>(CS Revision E or earlier 18 bits only) |
| DLV11-J | M8043 | Serial Line Interface<br>(CS Rev. E or earlier incompatible with KDF11-A and KDF11-B) |
| DRV11-B | M7950 | General Purpose DMA Interface (quad)<br>(18-bit DMA only) |

| | | |
|---|---|---|
| KPV11-B,C | M8016-YB,YC | Power-fail/line-time clock/terminator<br>(Termination for 18-bits only) |
| KUV11 | M8018 | Writable Control Store<br>(For use with KD11-F processor only) |
| KWV11-A | M7952 | Programmable real-time clock<br>(Use of BC1 for purposes other than BDAL18) |
| REV11 | M9400 | Terminator, DMA refresh, bootstrap<br>(Bootstrap for use with KD11-F and KD11-HA processors only.<br>Termination for 18-bits only.<br>DMA refresh may be used in any system.) |
| RKV11-D | M7269 | RK05 Controller Interface<br>(16-bit DMA only) |
| RLV11 | M8013<br>M8014 | RL01,2 Controller<br>(18-bit DMA only, use of BC1 and BD1 for purposes other than BDAL18 and BDAL19) |
| RXV21 | M8029 | RX02 Floppy Disk Interface<br>(18-bit DMA only) |
| TEV11 | M9400-YB | 120-Ohm Bus Terminator<br>(Termination for 18-bits only) |
| VSV11 | M7064 | Graphics Display<br>(18-bit DMA only) |

E. Bus Cable-Cards

| | |
|---|---|
| M9400-YD | Cable Connector<br>(18-bit bus only) |
| M9400-YE | Cable Connector with 240-Ohm Terminators<br>(18-bit bus only) |
| M9401 | Cable Connector<br>(18-bit bus only) |

| uNOTE #006 |
|---|

| Title: Differences Between the LSI-11/73 and LSI-11/23 | Date: 23-APR-84 |
|---|---|
| Originator: Mike Collins | Page 1 of 8 |

This uNOTE identifies and discusses the differences between the LSI-11/23 (KDF11-AA) and the LSI-11/73 (KDJ11-AA). The following table lists these differences. Following the table are individual discussions on these differences.

Some of these differences are discussed from the point of view of an 11/23 to 11/73 upgrade.

Table 1 LSI-11/73 versus LSI-11/23

| FEATURE | 11/73 | 11/23 |
|---|---|---|
| Odd Address Traps | Yes | No |
| Micro ODT | 22 Bit | 18 Bit |
| Illegal Halt | Traps to 4 | Traps to 10 |
| Processor Modes | 3 | 2 |
| I & D Space | Yes | No |
| General Purpose Reg Sets | 2 | 1 |
| Floating Point Inst. Set | Standard | Option |
| Line Time Clock Reg. | Yes | No |
| On-board Cache Memory | Yes | No |
| Pipelined Processing | Yes | No |
| UBMap Signal on the Q-bus | Not Available | Available |
| Additional Instructions Available | CSM, TSTSET, WRTLCK | Not Available |

cont'd

Table 1 cont'd LSI-11/73 versus LSI-11/23

| FEATURE | 11/73 | 11/23 |
|---|---|---|
| Additional CPU Registers | CPU Error Register<br>Memory System Error Reg<br>Cache Control Reg<br>Hit/Miss Reg<br>Program Interrupt Req Reg<br>Line Time Clock Reg<br>Maintenance Reg | Not Available |
| Processor Speed | A discussion of processor speed can be found in the respective user guides<br>User Guide Part # EK-KDJ1A-UG | User Guide Part # EK-KDF11-UG |

## ODD ADDRESS TRAPS

The 11/73 processor will trap to 4 when it encounters an odd address reference. i.e. whenever an address begins on an odd byte boundary (least significant bit = 1). The 11/23 ignores odd address references and simply treats the LSB as a zero, effectively 'forcing' all addresses to begin on even byte boundaries. Odd address traps do not occur frequently, however it is possible for code to run on an 11/23 and NOT run on an 11/73 because of them. Fixes for these errors are straightforward.

## MICRO ODT (Octal Debugging Technique)

Both the 11/23 and the 11/73 implement ODT in their microcode. The 11/23 can use ODT to examine main memory locations from 0 to 256 Kbytes, but no further. On the other hand, the 11/73 ODT can examine the full 4 Mbyte range of main memory. When accessing addresses in the I/O page with an 11/73, a full 22 bit address must be specified.

Example: To look at the first instruction of the bootstrap code with an 11/73 it is necessary to type:

```
@17773000/
or  @7777777777773000/

NOT @773000/      This is NOT enough because only
                  18 bits have been specified.
```

## ILLEGAL HALT

The 11/23 and the 11/73 respond differently when detecting a halt instruction in user or supervisor mode. The 11/23 traps to address 10 whereas the 11/73 traps to address 4. The 11/73 also sets the Illegal Halt Bit in the CPU ERROR Register to indicate an Illegal Halt occurred.

## PROCESSOR MODES

The 11/23 has two processor modes, KERNEL and USER. The 11/73 has three KERNEL, SUPERVISOR and USER.

## I and D SPACE

The concept of I and D space is used in mapping information into separate physical memory segments, depending on whether the information is considered instructions (I) or data (D). The use of I and D space allows programs to exist in two virtual segments and effectively doubles the address available to the user from 64 Kbytes to 128 Kbytes.

The 11/73 has the capability for I and D space whereas the 11/23 does not. To implement this feature, many more PAR/PDR pairs are necessary. The 11/73 has 48 PAR/PDR pairs, the 11/23 has only 16 PAR/PDR pairs.

## GENERAL PURPOSE REGISTER SETS

The 11/23 and all previous LSI-11 processors have 1 set of general purpose registers, R0 thru R7. Some of these are used for special purposes. R7 is used as the program counter and R6 is used as the stack pointer. Internal to the 11/23 are 2 registers used for stack pointers, one for each processor mode). There are 5 additional registers R0 thru R5.

The 11/73 has two sets of general purpose registers, listed in the table below. Only eight are visible to the user at any given time. There are two groups of six registers (R0 thru R5 and R0' thru R5'). The group currently being used is selected by bit 11 in the Processor Status Word (PSW). Only one stack pointer is visible to the user at any one time and is determined by bits 14 and 15 in the PSW.

| Register Number | Designation | | |
|---|---|---|---|
| 0 | R0 | R0' | |
| 1 | R1 | R1' | |
| 2 | R2 | R2' | |
| 3 | R3 | R3' | |
| 4 | R4 | R4' | |
| 5 | R5 | R5' | |
| 6 | KSP | SSP | USP |
| 7 | PC | | |

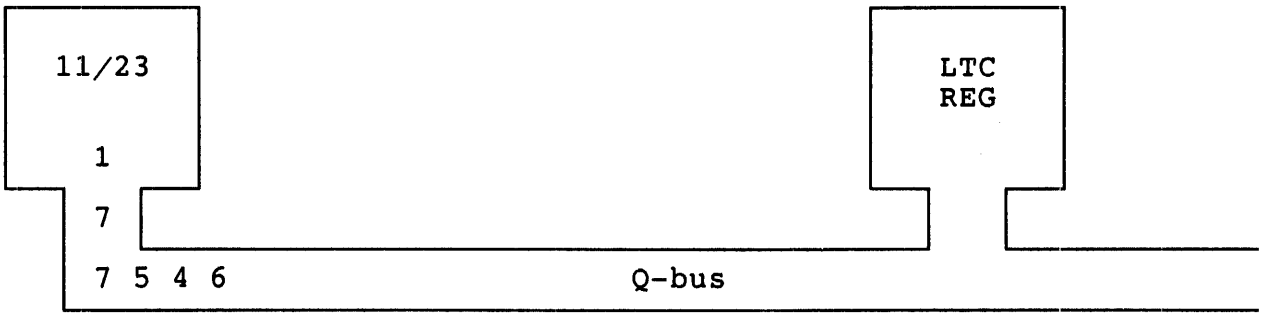
KSP = Kernel Stack Pointer
SSP = Supervisor Stack Pointer
USP = User Stack Pointer

## FLOATING POINT INSTRUCTION SET

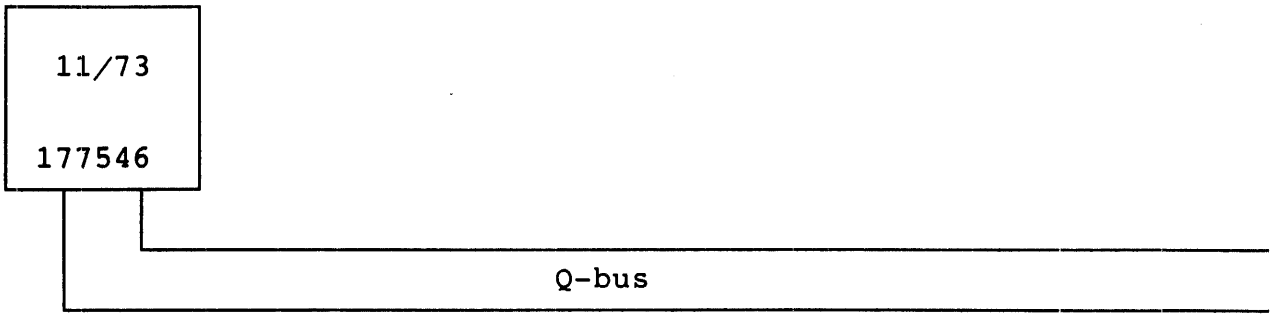
Both the 11/23 and the 11/73 use the FP11 Floating Point Instruction Set. The FP11 Instruction Set is an option for the 11/23 (choice of either the KEF11 chip or the FPF11 floating point accelerator). The FP11 instruction set is part of the J11 microprocessor microcode and is therefore a standard feature of the 11/73.

## LINE TIME CLOCK REGISTER

The original dual height 11/23 CPU does not have an LTC (Line Time Clock) register on the board. In 11/23 based systems the BDV11 boot module contains the LTC reg. In order to enable or disable LTC interrupts under software control, the 11/23 must write to this register over the Q-bus.




The 11/73 has an LTC register on the CPU board. This means that whenever the 11/73 wants to enable or disable LTC interrupts under software control it writes to this on-board register. The address of the LTC register (location 177546) is 'trapped' on the board and NEVER goes out onto the Q-bus. When the 11/73 is used in a system with a BDV11, it is recommended that software control over the LTC interrupts be disabled on the BDV11 (see uNOTE #114).




## ON-BOARD CACHE MEMORY

Cache memory systems are designed to increase CPU performance. The

cache maintains copies of portions of main memory in very high-speed RAM and thus reduces access times significantly.

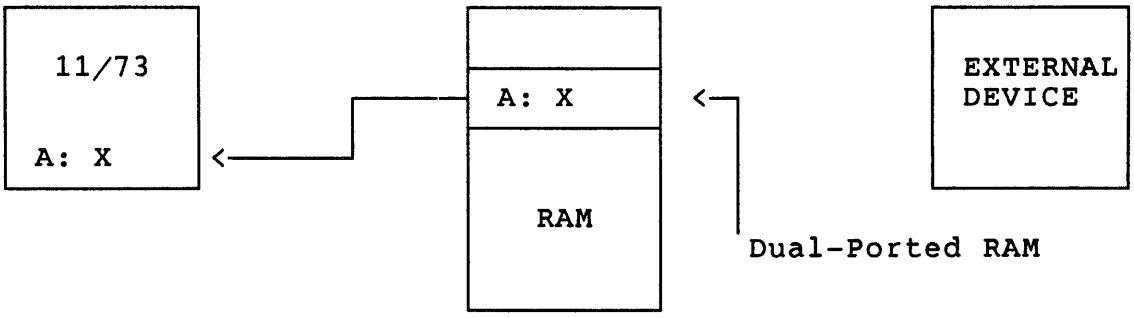
The 11/73 is the first Q-bus processor to implement a cache memory system. The cache is automatically enabled on power-up and its operation is transparent to software. However software can enable or disable the cache by writing to the Cache Control Register (CCR).

When the cache is enabled, any information fetched from main memory will be 'cached' i.e. placed in the high-speed RAM. Information fetched from an I/O device will NOT be 'cached' (i.e. information fetched from an address in the I/O page).

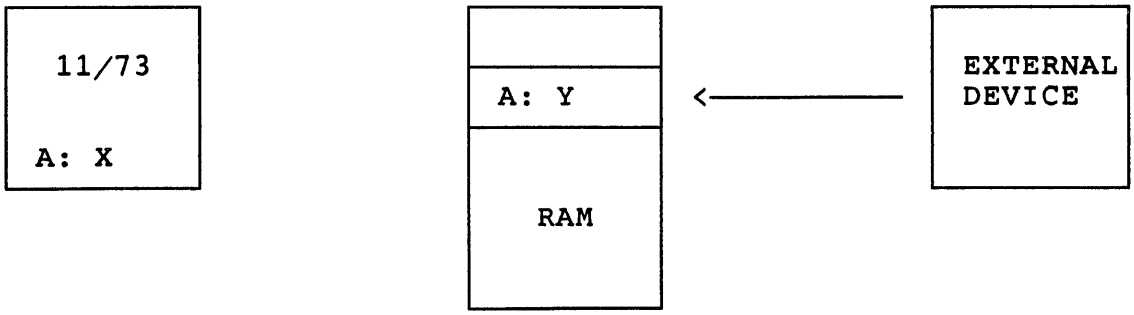
CAUTION : Digital Equipment Corporation does not support a system which uses shared or dual-ported memory on the Q-bus. However there are applications and non-DEC add-on hardware which do support such configurations. Consider the following:

The system below uses an 11/73, has a certain amount of main memory as well as dual-ported memory. The cache is enabled and the following sequence of events occur:

1. The 11/73 reads a word from the dual-ported RAM at address A which contains the value X. The value is 'cached'.




2. The external device writes a new value, Y, into location A.




3. The 11/73 references location A again, but finds that it

is in the cache and therefore uses the 'old' value of X. But this is incorrect since the external device updated location A with the new value Y.

This anomaly can be corrected in a number of ways.

A. Put the dual ported RAM somewhere in the I/O page since any I/O page reference always bypasses cache. If the amount of dual-ported RAM is large this may not be practical.

B. The memory management unit contains several Page Descriptor Registers (PDRs). The PDRs contain information relative to page expansion, page length and access control. Bit 15 of each PDR is the Bypass Cache bit. If the PDR accessed during a relocation operation has this bit set the reference will go directly to main memory. Hits on reads or writes will result in invalidation of the accessed cache location.

Enabling this bit in each PDR associated with the dual-ported memory will force these references to bypass cache.

C. Whenever the processor reads from the dual-ported RAM, add extra code which will simply turn off the cache prior to the read and turn the cache back on after the read is complete. Turning the cache on and off can be done by setting the appropriate bits in the Cache Control Register.

## PIPELINED PROCESSING

The 11/73 gets much of its performance by implementing a prefetch and predecode mechanism. The major benefit of this is that memory references are overlapped with internal operations which results in faster program execution. The 11/23 does not implement such a mechanism.

CAUTION : This implementation is completely compatible with DEC hardware and software. However there are applications and non-DEC add-on hardware which may be confused. Such situations are easily corrected.

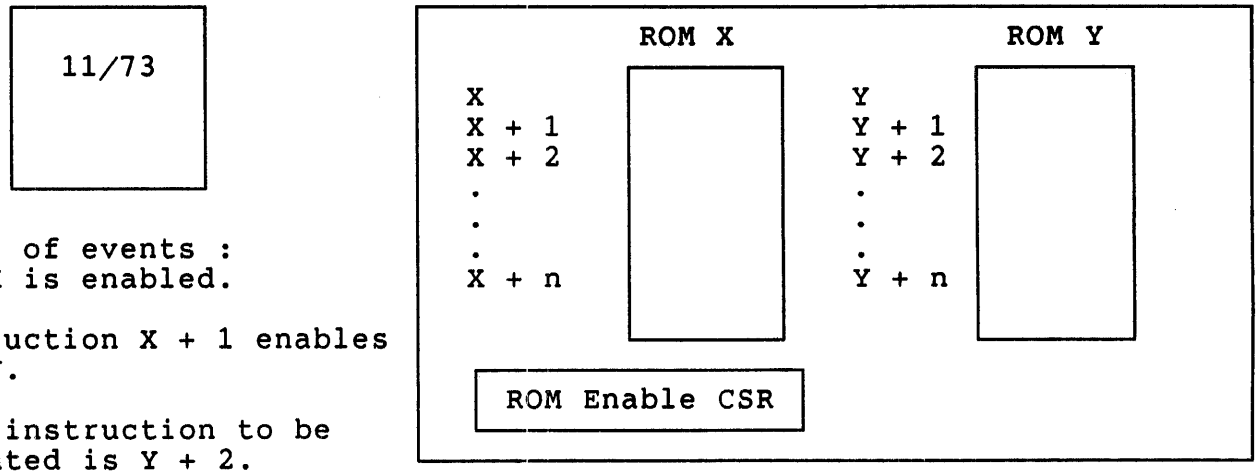
The prefetch mechanism assumes sequential program flow; one instruction immediately follows the next. Whenever the program flow is not sequential (i.e. the PSW, CCR, PC or any memory management register is written) the pipeline is 'flushed'. If a non-DEC device does its own 'macro' memory management, the instruction flow may be confused.

For example :

A non-DEC device utilizes 2 ROM sets for boot code. Both ROM sets map over the same addresses but only 1 set is enabled at any instant in

time. This is done via a ROM set enable CSR. Assume the boot code in ROM set X is executing. Instruction X + 1 is a MOV instruction to the ROM enable CSR to transfer program control to ROM set Y. The intent of the ROM code is for statement Y + 2 in ROM set 2 to be executed next. This will work OK with an 11/23. However because of the prefetch mechanism of the 11/73, the 11/73 will execute instruction X + 2 of ROM set 1, NOT ROM set 2.

This particular boot method effectively does its own memory mapping and since it is done at the 'macro' level and external to the J11 cpu, the pipeline is not 'flushed'. A simple solution is to include a NOP instruction after the instruction which updates the ROM enable CSR, or to use a branch instruction which effectively changes the PC and causes the pipeline to be 'flushed'.




Sequence of events :
1. ROM X is enabled.
2. Instruction X + 1 enables ROM Y.
3. Next instruction to be executed is Y + 2.

## MAP SIGNAL ON THE Q-BUS (UBMAP L)

The 11/23 outputs the signal UBMAP onto the Q-bus. Some non-DEC add-on equipment may use this signal for special purposes. This signal is not defined by the Q-bus specification and for design reasons was not included on the 11/73. Therefore, the 11/73 may not work with non-DEC devices which expect to see this signal.

## ADDITIONAL INSTRUCTIONS AVAILABLE

These 3 instructions are part of the 11/73 instruction set but are NOT found in the 11/23:

| | |
|---|---|
| CSM | Call to Supervisor Mode |
| TSTSET | Test Destination and Set Low Bit |
| WRTLCK | Read/Lock Destination Write/Unlock R0 into Destination |

## ADDITIONAL CPU REGISTERS AVAILABLE

The following CPU registers are part of the DCJ11 chip or implemented on the 11/73 module and are not found on the 11/23.

CPU Error Register
Memory System Error Register
Cache Control Register
Hit/Miss Register
Additional PAR/PDR's necessary to implement I & D Space
Program Interrupt Request Register
Line Time Clock Register (previously discussed)
Maintenance Register

## PROCESSOR SPEED

The 11/73 executes instructions significantly faster than the 11/23. Software which bases delays on instruction loops may not work on the 11/73 because the instructions, and therefore the delay loop, are completed much faster than they were when executed on the 11/23. Software which uses this method of implementing delays produces code which is not processor independent. However instances do appear every so often and it is important to be aware of this possibility.

| uNOTE # 007 |
|---|

| Title: User Defined Memory Maps For The Falcon And The Falcon Plus | Date: 01-MAY-84 |
|---|---|
| Originator: Jack Toto | Page 1 of 13 |

The memory maps on the Falcon, and Falcon Plus are defined by the FPLA (Field Programmable Logic Array) that is located on board in socket XE41. This micronote explains how to redefine the existing maps.

The equipment needed to custom build the FPLA is as follows:

1. Signetics NS82S100N FPLA.
2. Signetics NS82S100N FPLA worksheet.
3. PROM programmer that supports Signetics NS82S100N FPLA.
4. User written routine to convert binary into hex form, needed for the PROM programmer.

The Falcon or Falcon Plus can be configured to any one of four standard memory maps. When you custom build the memory map FPLA, you may keep this ability to select any one of four memory map schemes, however it is not nessecary to build in the four maps, you may build an FPLA with only one map for the Falcon. Further the addresses of the SLUs and Parallel I/O Port can be be changed to some other addresses on the module, or they can be relocated off the module to some other device such as an MXV11-A/B, DLV11-J, DRV11-J, etc.

The memory map is defined by:

1. Selecting a range of addresses, for example addresses 077777 - 037777. This includes selecting the addresses of the boards SLUs and PIO.
2. Asserting or deasserting the bits which define that range.
3. Defining which one of four memory maps the selected range of addresses will respond.
4. Selecting an enable bit (A7 -A0), which tells the T-11 where to find that range of addresses: on the Q-bus, on board the Falcon Plus in one of the two sockets, or in the local RAM.

When mapping an FPLA to be used with the KXT11-A2 or KXT11-A5 ODT ROMs, you must map addresses 177600 - 177777 to the local RAM that comes on board the Falcon or Falcon Plus. This space is used for a scratch pad area by Macro ODT. If Macro ODT is not going to be used than this address range can be left to be selected by the user. Beyond this all addresses and functions can be mapped off board.

Below is an example of an FPLA worksheet that was completed specifying one map (map 0) which maps all of memory to the Q-bus, and changes the address of SLU2 from 176540 - 176547, to 176500 - 176507. This format is simalar to the Signetics FPLA worksheet. For each address range shown, the combination of bits set (H), cleared (L), or in the don't care state (-) define that range. Further each address range has associated with it an output enable bit (A) that identifies the location of the memory containing that range. It is possible to define larger or smaller ranges then the ones shown here, if needed. More examples are provided in the Falcon Plus User's Guide Appendix H.

THIS IS AN FPLA MAP FOR
COMPANY: XYZ

| | | I | I | I | I | I | I | I | I | I | I | I | I | I | I | I | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | | | | | | |
| | | 7 | 8 | 6 | 9 | 5 | 0 | 4 | 1 | 3 | 2 | 2 | 3 | 1 | 5 | 0 | 4 | F | F | F | F | F | F | F | |
| ADDRESS | X | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RANGE | X | 5 | 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |
| 177600-177777 | RAM | - | - | - | H | H | H | H | H | H | - | - | - | - | L | L | H | A | | | | | | | 00 |
| 177570-177577 | QB | - | - | - | H | H | H | H | H | L | H | H | H | H | L | L | H | | | | | | A | | 01 |
| 177560-177567 | DL0 | - | - | - | H | H | H | H | H | L | H | H | H | L | L | L | H | | | | A | | | | 02 |
| 177540-177557 | QB | - | - | - | H | H | H | H | H | L | H | H | L | - | L | L | H | | | | | | A | | 03 |
| 177500-177537 | QB | - | - | - | H | H | H | H | H | L | H | L | - | - | L | L | H | | | | | | A | | 04 |
| 177400-177477 | QB | - | - | - | H | H | H | H | H | L | L | - | - | - | L | L | H | | | | | | A | | 05 |
| 177000-177377 | QB | - | - | - | H | H | H | H | L | - | - | - | - | - | L | L | H | | | | | | A | | 06 |
| 176600-176777 | QB | - | - | - | H | H | H | L | H | H | - | - | - | - | L | L | H | | | | | | A | | 07 |
| 176560-176577 | QB | - | - | - | H | H | H | L | H | L | H | H | H | - | L | L | H | | | | | | A | | 08 |
| 176550-176557 | QB | - | - | - | H | H | H | L | H | L | H | H | L | H | L | L | H | | | | | | A | | 09 |
| 176540-176547 | QB | - | - | - | H | H | H | L | H | L | H | H | L | L | L | L | H | | | | | | A | | 10 |
| 176510-176537 | QB | - | - | - | H | H | H | L | H | L | H | L | - | H | L | L | H | | | | | | A | | 11 |
| 176500-176507 | DL1 | - | - | - | H | H | H | L | H | L | H | L | L | L | L | L | H | | | | | | | A | 12 |
| 176400-176477 | QB | - | - | - | H | H | H | L | H | L | L | - | - | - | L | L | H | | | | | | A | | 13 |
| 176300-176377 | QB | - | - | - | H | H | H | L | L | H | H | - | - | - | L | L | H | | | | | | A | | 14 |
| 176240-176277 | QB | - | - | - | H | H | H | L | L | H | L | H | - | - | L | L | H | | | | | | A | | 15 |
| 176220-176237 | QB | - | - | - | H | H | H | L | L | H | L | L | H | - | L | L | H | | | | | | A | | 16 |
| 176210-176217 | QB | - | - | - | H | H | H | L | L | H | L | L | L | H | L | L | H | | | | | | A | | 17 |
| 176200-176207 | PIO | - | - | - | H | H | H | L | L | H | L | L | L | L | L | L | H | | | | A | | | | 18 |
| 176000-176177 | QB | - | - | - | H | H | H | L | L | L | - | - | - | - | L | L | H | | | | | | A | | 19 |
| 174000-175777 | QB | - | - | - | H | H | L | - | - | - | - | - | - | - | L | L | H | | | | | | A | | 20 |
| 170000-173777 | QB | - | - | - | H | L | - | - | - | - | - | - | - | - | L | L | H | | | | | | A | | 21 |
| 160000-167777 | QB | - | - | - | L | - | - | - | - | - | - | - | - | - | L | L | H | | | | | | A | | 22 |
| 140000-157777 | QB | H | H | L | - | - | - | - | - | - | - | - | - | - | L | L | L | | | | | | A | | 23 |
| 100000-137777 | QB | H | L | - | - | - | - | - | - | - | - | - | - | - | L | L | L | | | | | | A | | 24 |
| 000000-077777 | QB | L | - | - | - | - | - | - | - | - | - | - | - | - | L | L | L | | | | | | A | | 25 |
| | | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Once the address ranges have been coded, they need to be converted into the formatted file shown directly below. This file will be used as the input file to a program (see end of Micronote) that will output the FPLA terms in a format that is usable by the PROM Programmer itself. When converting the above worksheet to FPLA terms, the following should be considered:

1. The area at the beginning of the file may be used as a comment area as long as the word "START" is not used in the comments. "START" is the key word to begin the formatting process while the word "END" is the key word used to signal the formatter that the processing must be terminated. Both words MUST appear in capital letters.

2. The layout of the map must be as follows:

```
                                                                  _____
I15 I14 I13 I12 I11 I10 I09 I08 I07 I06 I05 I04 I03 I02 I01 I00 | TERM1
O7  O6  O5  O4  O3  O2  O1  O0                                    |
                                                                  _____
                                                                  _____
I15 I14 I13 I12 I11 I10 I09 I08 I07 I06 I05 I04 I03 I02 I01 I00 | TERM2
O7  O6  O5  O4  O3  O2  O1  O0                                    |
                                                                  _____
```

   Where I15 is input 15 on the FPLA ,I14 is input 14 on the PLA etc. further O7 is output 7 on the FPLA and O6 is output 6 on the PLA.

3. "—" indicates a "DON'T CARE ON THAT INPUT" AND * indicates a "DON'T CARE ON THIS OUTPUT".

The following is the formatted FPLA map for the worksheet shown above. Notice that bus address bit 00 is translated to FPLA bit I14, that bus address 01 is translated to FPLA bit I00, that is, the bus address bits are not directly translated to the FPLA bits, however the worksheet activity (output) bits F7 - F0 translate directly to FPLA bits A7 - A0.

```
START
LH——HHH———HHH——L
A*******
LHHHHHH———HHLHHL
*****A**
LHHHHHH———HHLHLL
**A*****
LHLHHHH———HHLH-L
*****A**
LH-HHHH———HHLL-L
*****A**
LH-LHHH———HHL——L
*****A**
LH——LHH———HH———L
*****A**
LH——HHH———HLH——L
```

```
*****A**
LHHHHHH----HLLH-L
*****A**
LHLHHHH----HLLHHL
*****A**
LHLHHHH----HLLHLL
*****A**
LH-HHHH----HLLLHL
*****A**
LHLHHHH----HLLLLL
******A*
LH-LHHH----HLL--L
*****A**
LH-HLHH----HLH--L
*****A**
LH-LLHH----HLHH-L
*****A**
LHHLLHH----HLHL-L
*****A**
LHLLLHH----HLHLHL
*****A**
LHLLLHH----HLHLLL
***A****
LH--LHH----HLL--L
*****A**
LH---LH----H-----L
*****A**
LH-----H----L-----L
*****A**
LH-----L---------L
*****A**
LL------HHL------L
*****A**
LL------LH-------L
*****A**
LL-------L-------L
*****A**
```

Translation: The following is a translation of FPLA terms and FPLA activity outputs, and the actual pin name on the FPAL chip.

I15....D15 (02 on worksheet above) used for high order bit of map decode.
I14....LBS7 (00 on worksheet above) asserted for all I/O page addresses.
I13....AD04
I12....AD06
I11....AD08
I10....AD10
I09....AD12
I08....AD14
I07....AD15

I06....AD13
I05....AD11
I04....AD09
I03....AD07
I02....AD05
I01....AD03
I00....DO(01 on worksheet above) used for the low order bit of map decode.

A7.....F7 on worksheet above, used for local RAM enable.
A6.....F6 on worksheet above, used for socket A enable.
A5.....F5 on worksheet above, used for DL0 enable.
A4.....F4 on worksheet above, used for PIO enable.
A3.....F3 on worksheet above, not used.
A2.....F2 on worksheet above, used for all Q-bus enables.
A1.....F1 on worksheet above, used for DL1 enable.
A0.....F0 on worksheet above, used for socket B enable.

The output file shown below is an example of one that would be loadable into a PROM Programmer in order to blast the FPLA chip. An example of a program that will produce the formatted output file shown below can be found at the end of this micronote.

```
START
$A0000,   ( ADDRESS )
FF        ( FIRST 8 BITS )
$A100,    ( ADDRESS )
FF
00
FD
FB
F1
EF
7E
C1
80
7F
00
FF
FF
FF
FF
FF
FF
FF
.
.         ( OTHER TERMS IN HEX BYTES )
.
.
$A200     ( NEXT ADDRESS BOUNDARY )
.
.         ( TERMS )
.
```

```
$A300    ( ADDRESS )
 .
 .        ( TERMS )
 .
 $A400    ( ADRESSS )
 .
 .        ( TERMS )
 .
 ETC.
```

Once the FPLA is built, it is placed into socket XE41, the memory will be configured to whatever the FPLA terms look like. The Falcon Plus needs to be strapped for a start address, a memory map (1 of 4) that matches the one called out in the FPLA, and other user configurable options.

FPLA FORMAT PROGRAM **********

```
1 REM THIS PROGRAM IS FOR NORMAL PLA MAP INPUTS
2 PRINT "INPUT DEVICE AND FILE NAME";H1$
4 INPUT H1$
5 OPEN H1$ FOR INPUT AS FILE #1
7 PRINT "DO YOU WISH OUTPUT COMPARE FILE ON LINE PRINTER,DEVICE,BOTH OR
  NONE ? "
8 PRINT "DEFAULT IS NONE (LP,DEV,B,N) ";N1$
10 INPUT N1$
12 IF N1$-"LP" THEN  GO TO 30
16 IF N1$-"DEV" THEN  GO TO 36
20 IF N1$-"B" THEN  GO TO 42
22 LET N$-"N"
23 LET K1$-"N"
24 GO TO 50
30 OPEN "LP:" FOR OUTPUT AS FILE #2
33 LET N$-"Y"
34 LET K1$-"N"
35 GO TO 50
36 PRINT "WHAT IS DEVICE AND FILE NAME";U1$ \ INPUT U1$
37 OPEN U1$ FOR OUTPUT AS FILE #3
38 LET N$-"N"
39 LET K1$-"Y"
40 GO TO 50
42 OPEN "LP:" FOR OUTPUT AS FILE #2
45 LET N$-"Y"
46 PRINT "WHAT IS DEVICE AND FILE NAME";U1$ \ INPUT U1$
47 OPEN U1$ FOR OUTPUT AS FILE #3
48 LET K1$-"Y"
50 PRINT "WHAT DEVICE AND FILE NAME FOR OUTPUT PROM PROGRAMMER FILE";W$
52 INPUT W$
55 OPEN W$ FOR OUTPUT AS FILE #4
60 LET Y1=1
62 LET H1$=CHR$(2)
64 LET H2$=CHR$(3)
```

```
66 LET Z$=CHR$(10)+CHR$(13)
68 LET L1$-" "
70 INPUT #1,A$
71 IF N$-"N" THEN  GO TO 73
72 PRINT #2,A$
73 IF K1$-"N" THEN  GO TO 76
74 PRINT #3,A$
75 PRINT #4,A$
76 IF A$-"START" THEN  GO TO 80
78 GO TO 70
80 LET S1$-"$A0000,"
82 PRINT #4,H1$;S1$
84 PRINT #4,"FF "
86 LET S1$-"$A100,"
88 PRINT #4,S1$
90 GO TO 92
91 PRINT #4,S1$
93 INPUT #1,I1$
92 IF I1$-"END" THEN  GO TO 120
94 INPUT #1,O1$
95 INPUT #1,I2$
96 IF I2$-"END" THEN  GO TO 130
97 INPUT #1,O2$
98 INPUT #1,I3$
99  IF I3$-"END" THEN  GO TO 140
100 INPUT #1,O3$
101 INPUT #1,I4$
102 IF I4$-"END" THEN  GO TO 150
103 INPUT #1,O4$
104 INPUT #1,I5$
105 IF I5$-"END" THEN  GO TO 160
106 INPUT #1,O5$
107 INPUT #1,I6$
108 IF I6$-"END" THEN  GO TO 170
109 INPUT #1,O6$
110 INPUT #1,I7$
111 IF I7$-"END" THEN  GO TO 180
112 INPUT #1,O7$
113 INPUT #1,I8$
114 IF I8$-"END" THEN  GO TO 190
115 INPUT #1,O8$
117 GO TO 200
120 LET I1$-"0000000000000000" \ LET O1$-"AAAAAAAA"
121 LET I2$-"0000000000000000" \ LET O2$-"AAAAAAAA"
122 LET I3$-"0000000000000000" \ LET O3$-"AAAAAAAA"
123 LET I4$-"0000000000000000" \ LET O4$-"AAAAAAAA"
124 LET I5$-"0000000000000000" \ LET O5$-"AAAAAAAA"
125 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
126 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
127 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
128 LET E1=1
129 GO TO 200
```

```
130 LET I2$-"0000000000000000" \ LET O2$-"AAAAAAAA"
132 LET I3$-"0000000000000000" \ LET O3$-"AAAAAAAA"
133 LET I4$-"0000000000000000" \ LET O4$-"AAAAAAAA"
134 LET I5$-"0000000000000000" \ LET O5$-"AAAAAAAA"
135 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
136 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
137 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
138 LET E1=1
139 GO TO 200
140 LET I3$-"0000000000000000" \ LET O3$-"AAAAAAAA"
143 LET I4$-"0000000000000000" \ LET O4$-"AAAAAAAA"
144 LET I5$-"0000000000000000" \ LET O5$-"AAAAAAAA"
145 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
146 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
147 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
148 LET E1=1
149 GO TO 200
150 LET I4$-"0000000000000000" \ LET O4$-"AAAAAAAA"
154 LET I5$-"0000000000000000" \ LET O5$-"AAAAAAAA"
155 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
156 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
157 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
158 LET E1=1
159 GO TO 200
160 LET I5$-"0000000000000000" \ LET O5$-"AAAAAAAA"
165 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
166 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
167 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
168 LET E1=1
169 GO TO 200
170 LET I6$-"0000000000000000" \ LET O6$-"AAAAAAAA"
176 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
177 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
178 LET E1=1
179 GO TO 200
180 LET I7$-"0000000000000000" \ LET O7$-"AAAAAAAA"
187 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
188 LET E1=1
189 GO TO 200
190 LET I8$-"0000000000000000" \ LET O8$-"AAAAAAAA"
198 LET E1=1
200 PRINT I1$,O1$,Z$,I2$,O2$,Z$,I3$,O3$,Z$,I4$,O4$
210 PRINT I5$,O5$,Z$,I6$,O6$,Z$,I7$,O7$,Z$,I8$,O8$
215 IF N$-"N" THEN  GO TO 218
216 PRINT #2,I1$,O1$,Z$,I2$,O2$,Z$,I3$,O3$,Z$,I4$,O4$
217 PRINT #2,I5$,O5$,Z$,I6$,O6$,Z$,I7$,O7$,Z$,I8$,O8$
218 IF K1$-"N" THEN  GO TO 221
219 PRINT #3,I1$,O1$,Z$,I2$,O2$,Z$,I3$,O3$,Z$,I4$,O4$
220 PRINT #3,I5$,O5$,Z$,I6$,O6$,Z$,I7$,O7$,Z$,I8$,O8$
221 LET X=16
235 LET D1$=SEG$(I1$,X,X)
240 IF D1$-"H" GO TO 260
```

```
245 IF D1$="L" GO TO 270
250 IF D1$="-" GO TO 280
252 IF D1$="0" GO TO 256
255 GO TO 1340
256 LET A(1,0)=1 \ LET A(1,1)=1
257 GO TO 285
260 LET A(1,0)=1 \ LET A(1,1)=0
265 GO TO 285
270 LET A(1,0)=0 \ LET A(1,1)=1
275 GO TO 285
280 LET A(1,0)=0 \ LET A(1,1)=0
285 LET D2$=SEG$(I2$,X,X)
290 IF D2$="H" GO TO 315
300 IF D2$="L" GO TO 325
305 IF D2$="-" GO TO 335
306 IF D2$="0" GO TO 309
307 GO TO 1340
309 LET A(1,2)=1 \ LET A(1,3)=1
310 GO TO 340
315 LET A(1,2)=1 \ LET A(1,3)=0
320 GO TO 340
325 LET A(1,2)=0 \ LET A(1,3)=1
330 GO TO 340
335 LET A(1,2)=0 \ LET A(1,3)=0
340 LET D3$=SEG$(I3$,X,X)
345 IF D3$="H" GO TO 365
350 IF D3$="L" GO TO 375
355 IF D3$="-" GO TO 385
357 IF D3$="0" GO TO 359
358 GO TO 1340
359 LET A(1,4)=1 \ LET A(1,5)=1
360 GO TO 390
365 LET A(1,4)=1 \ LET A(1,5)=0
370 GO TO 390
375 LET A(1,4)=0 \ LET A(1,5)=1
380 GO TO 390
385 LET A(1,4)=0 \ LET A(1,5)=0
390 LET D4$=SEG$(I4$,X,X)
400 IF D4$="H" GO TO 420
405 IF D4$="L" GO TO 430
410 IF D4$="-" GO TO 440
412 IF D4$="0" GO TO 414
413 GO TO 1340
414 LET A(1,6)=1 \ LET A(1,7)=1
415 GO TO 445
420 LET A(1,6)=1 \ LET A(1,7)=0
425 GO TO 445
430 LET A(1,6)=0 \ LET A(1,7)=1
435 GO TO 445
440 LET A(1,6)=0 \ LET A(1,7)=0
445 LET A8=A(1,0)*1+A(1,2)*2+A(1,4)*4+A(1,6)*8
450 LET A9=A(1,1)*1+A(1,3)*2+A(1,5)*4+A(1,7)*8
```

```
455 LET A8=15-A8 \ LET A9=15-A9
460 LET D5$=SEG$(I5$,X,X)
465 IF D5$="H" GO TO 485
470 IF D5$="L" GO TO 495
475 IF D5$="-" GO TO 505
476 IF D5$="0" GO TO 479
477 GO TO 1340
479 LET B(1,0)=1 \ LET B(1,1)=1
480 GO TO 510
485 LET B(1,0)=1 \ LET B(1,1)=0
490 GO TO 510
495 LET B(1,0)=0 \ LET B(1,1)=1
500 GO TO 510
505 LET B(1,0)=0 \ LET B(1,1)=0
510 LET D6$=SEG$(I6$,X,X)
515 IF D6$="H" GO TO 535
520 IF D6$="L" GO TO 545
525 IF D6$="-" GO TO 555
526 IF D6$="0" GO TO 529
527 GO TO 1340
529 LET B(1,2)=1 \ LET B(1,3)=1
530 GO TO 560
535 LET B(1,2)=1 \ LET B(1,3)=0
540 GO TO 560
545 LET B(1,2)=0 \ LET B(1,3)=1
550 GO TO 560
555 LET B(1,2)=0 \ LET B(1,3)=0
560 LET D7$=SEG$(I7$,X,X)
565 IF D7$="H" GO TO 585
570 IF D7$="L" GO TO 595
575 IF D7$="-" GO TO 605
576 IF D7$="0" GO TO 579
577 GO TO 1340
579 LET B(1,4)=1 \ LET B(1,5)=1
580 GO TO 610
585 LET B(1,4)=1 \ LET B(1,5)=0
590 GO TO 610
595 LET B(1,4)=0 \ LET B(1,5)=1
600 GO TO 610
605 LET B(1,4)=0 \ LET B(1,5)=0
610 LET D8$=SEG$(I8$,X,X)
615 IF D8$="H" GO TO 635
620 IF D8$="L" GO TO 645
625 IF D8$="-" GO TO 655
626 IF D8$="0" GO TO 629
627 GO TO 1340
629 LET B(1,6)=1 \ LET B(1,7)=1
630 GO TO 660
635 LET B(1,6)=1 \ LET B(1,7)=0
640 GO TO 660
645 LET B(1,6)=0 \ LET B(1,7)=1
650 GO TO 660
```

```
655 LET B(1,6)=0 \ LET B(1,7)=0
660 LET B8=B(1,0)*1+B(1,2)*2+B(1,4)*4+B(1,6)*8
665 LET B9=B(1,1)*1+B(1,3)*2+B(1,5)*4+B(1,7)*8
670 LET B8=15-B8 \ LET B9=15-B9
675 IF B8<10 THEN LET P1=0
680 IF B8<10 THEN  GO TO 720
685 IF B8=10 THEN LET B8$-"A"
690 IF B8=11 THEN LET B8$-"B"
695 IF B8=12 THEN LET B8$-"C"
700 IF B8=13 THEN LET B8$-"D"
705 IF B8=14 THEN LET B8$-"E"
710 IF B8=15 THEN LET B8$-"F"
715 LET P1=1
720 IF A8<10 THEN LET P0=0
725 IF A8<10 THEN  GO TO 765
730 IF A8=10 THEN LET A8$-"A"
735 IF A8=11 THEN LET A8$-"B"
740 IF A8=12 THEN LET A8$-"C"
745 IF A8=13 THEN LET A8$-"D"
750 IF A8=14 THEN LET A8$-"E"
755 IF A8=15 THEN LET A8$-"F"
760 LET P0=1
765 IF P1=0 THEN  GO TO 780
770 IF P0=0 THEN  GO TO 805
775 GO TO 795
780 IF P0=1 THEN  GO TO 815
785 PRINT #4,STR$(B8);STR$(A8);L1$
790 GO TO 820
795 PRINT #4,B8$;A8$;L1$
800 GO TO 820
805 PRINT #4,B8$;STR$(A8);L1$
810 GO TO 820
815 PRINT #4,STR$(B8);A8$;L1$
820 IF B9<10 THEN LET P3=0
825 IF B9<10 THEN  GO TO 865
830 IF B9=10 THEN LET B9$-"A"
835 IF B9=11 THEN LET B9$-"B"
840 IF B9=12 THEN LET B9$-"C"
845 IF B9=13 THEN LET B9$-"D"
850 IF B9=14 THEN LET B9$-"E"
855 IF B9=15 THEN LET B9$-"F"
860 LET P3=1
865 IF A9<10 THEN LET P2=0
870 IF A9<10 THEN  GO TO 910
875 IF A9=10 THEN LET A9$-"A"
880 IF A9=11 THEN LET A9$-"B"
885 IF A9=12 THEN LET A9$-"C"
890 IF A9=13 THEN LET A9$-"D"
895 IF A9=14 THEN LET A9$-"E"
900 IF A9=15 THEN LET A9$-"F"
905 LET P2=1
910 IF P3=0 THEN  GO TO 925
```

```
915 IF P2=0 THEN  GO TO 945
920 GO TO 937
925 IF P2=1 THEN  GO TO 955
930 PRINT #4,STR$(B9);STR$(A9);L1$
935 GO TO 960
937 PRINT #4,B9$;A9$;L1$
940 GO TO 960
945 PRINT #4,B9$;STR$(A9);L1$
950 GO TO 960
955 PRINT #4,STR$(B9);A9$;L1$
960 LET X=X-1
965 IF X>0 THEN  GO TO 235
970 LET Y=8
975 LET C1$=SEG$(O1$,Y,Y)
980 IF C1$="A" THEN LET A1=1
985 IF C1$="*" THEN LET A1=0
990 LET C2$=SEG$(O2$,Y,Y)
995 IF C2$="A" THEN LET A2=1
1000 IF C2$="*" THEN LET A2=0
1005 LET C3$=SEG$(O3$,Y,Y)
1010 IF C3$="A" THEN LET A3=1
1015 IF C3$="*" THEN LET A3=0
1020 LET C4$=SEG$(O4$,Y,Y)
1025 IF C4$="A" THEN LET A4=1
1030 IF C4$="*" THEN LET A4=0
1035 LET C5$=SEG$(O5$,Y,Y)
1040 IF C5$="A" THEN LET A5=1
1045 IF C5$="*" THEN LET A5=0
1050 LET C6$=SEG$(O6$,Y,Y)
1055 IF C6$="A" THEN LET A6=1
1060 IF C6$="*" THEN LET A6=0
1065 LET C7$=SEG$(O7$,Y,Y)
1070 IF C7$="A" THEN LET A7=1
1075 IF C7$="*" THEN LET A7=0
1080 LET C8$=SEG$(O8$,Y,Y)
1085 IF C8$="A" THEN LET F8=1
1090 IF C8$="*" THEN LET F8=0
1095 LET O9=(A1*1)+(A2*2)+(A3*4)+(A4*8)
1100 LET P9=(A5*1)+(A6*2)+(A7*4)+(F8*8)
1105 LET O9=15-O9 \ LET P9=15-P9
1110 IF P9<10 THEN LET T1=0
1115 IF P9<10 THEN  GO TO 1155
1120 IF P9=10 THEN LET P9$="A"
1125 IF P9=11 THEN LET P9$="B"
1130 IF P9=12 THEN LET P9$="C"
1135 IF P9=13 THEN LET P9$="D"
1140 IF P9=14 THEN LET P9$="E"
1145 IF P9=15 THEN LET P9$="F"
1150 LET T1=1
1155 IF O9<10 THEN LET T0=0
1160 IF O9<10 THEN  GO TO 1200
1165 IF O9=10 THEN LET O9$="A"
```

```
1170 IF O9=11 THEN LET O9$-"B"
1175 IF O9=12 THEN LET O9$-"C"
1180 IF O9=13 THEN LET O9$-"D"
1185 IF O9=14 THEN LET O9$-"E"
1190 IF O9=15 THEN LET O9$-"F"
1195 LET T0=1
1200 IF T1=0 THEN  GO TO 1215
1205 IF T0=0 THEN  GO TO 1240
1210 GO TO 1230
1215 IF T0=1 THEN  GO TO 1255
1220 PRINT #4,STR$(P9);STR$(O9);L1$
1225 GO TO 1260
1230 PRINT #4,P9$;O9$;L1$
1235 GO TO 1260
1240 PRINT #4,P9$;STR$(O9);L1$
1250 GO TO 1260
1255 PRINT #4,STR$(P9);O9$;L1$
1260 LET Y=Y-1
1265 IF Y>0 THEN  GO TO 975
1270 PRINT #4,Z$
1275 LET Y1=Y1+1
1280 IF Y1=2 THEN LET S1$-"$A200,"
1285 IF Y1=3 THEN LET S1$-"$A300,"
1290 IF Y1=4 THEN LET S1$-"$A400,"
1295 IF Y1=5 THEN LET S1$-"$A500,"
1300 IF Y1=6 THEN LET S1$-"$A600,"
1305 PRINT #4,Z$
1310 IF Y1=7 THEN  GO TO 1320
1315 IF E1=0 THEN  GO TO 91
1320 PRINT #4,H2$
1330 CLOSE
1335 GO TO 1345
1340 PRINT "INPUT LINE IN ERROR"
1345 END
```

| uNOTE # 008 |
|---|

| Title: Memory Management and the LSI-11/73 | Date: 22-Jun-84 |
|---|---|
| Originator: Dave Smith | Page 1 of 11 |

This micronote explains memory management as it applies to the LSI-11/73. It includes descriptions of what memory management is, what it does, and how it works.

Simply stated, memory management is a method of mapping virtual addresses to physical addresses. The virtual address space is the view of memory as seen by a process running. The physical address space is the actual physical memory as seen by the entire system. Since ALL memory references must be mapped, this translation is done in hardware by using a Memory Management Unit (MMU). Various schemes (dependent upon the architecture) have been developed to accomplish this, but most memory management systems provide the following services:

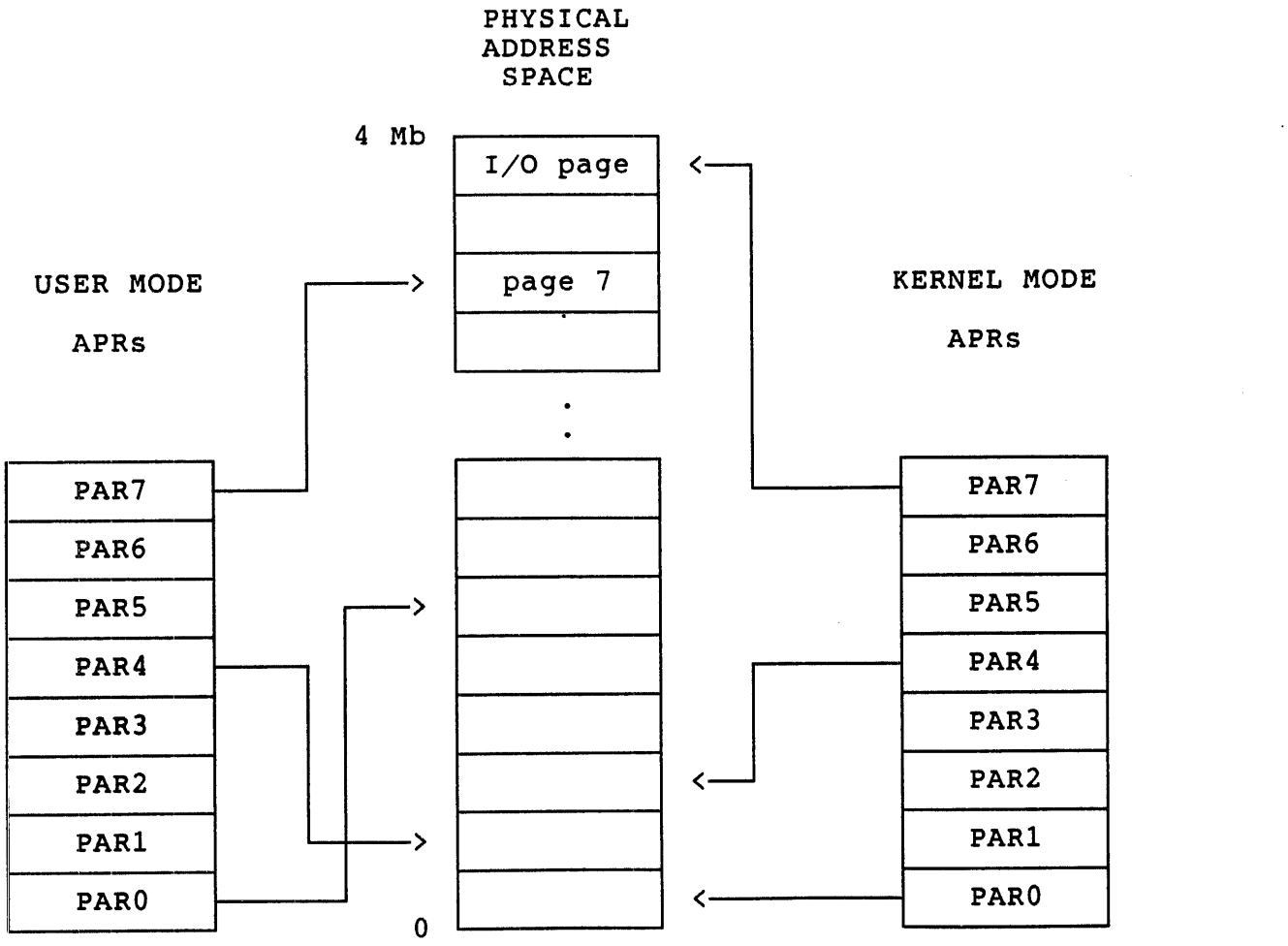
1) Protection
2) Relocation
3) Segmentation

The first two are of great importance in a multiprogramming system since memory management provides the mechanism for assigning memory areas to user programs and for preventing users from accessing areas that have been assigned to other users. This protects the operating system executive as well as users from accidental or willful memory accesses outside of a user's assigned memory. Relocation is also important in a multiprogramming environment since the executive must be able to relocate the user's program to a free area in physical memory.

The LSI-11/73 Memory Management Unit provides the hardware for complete memory management by providing all of the above services. It is software compatible with the larger UNIBUS PDP-11s and other Q-bus processors. Since the LSI-11/73 has the PDP-11 architecture and a 16-bit program counter, all 16-bit addresses access a 64Kb virtual address space. On the LSI-11/73 this addressing limitation can be eased by using separate sets of memory management registers for instructions (I-space) and for data (D-space). By utilizing separate I- and D-space, the address space can be segmented into two 64Kb segments, effectively doubling the virtual address space. The LSI-11/73 and the Q-bus allow 4 Mb of memory to be referenced. The MMU is necessary to provide the mapping from the 64Kb virtual address space to the 4 Mb physical address space.

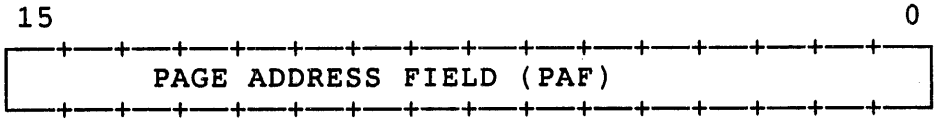
When the MMU is activated, a 16-bit virtual address is mapped to an 18-bit or 22-bit physical address. The MMU uses registers known as Active Page Registers (APRs). An APR consists of two 16-bit registers which are called the Page Address Register (PAR) and the Page Descriptor Register (PDR). PARs are used in the actual address translation while PDRs contain access and other information. Since the LSI-11/73 is functionally equivalent to the PDP-11/70, it can operate in one of three modes: Kernel, Supervisor, or User and it can use I- and D- space. This means that the MMU must provide separate sets of registers for each mode and within each mode, sets of registers for I-space and for D-space. A set of registers consists of eight pairs of PDRs and PARs. Thus the LSI-11/73 MMU has a total of three sets of 32 16-bit registers.

Mapping is always done in pages of 8Kb (4K words) in length or less. In order to map the largest possible virtual address space (64 Kb), the address space is divided into eight pages of 8Kb each. One APR is used for each of the eight pages, numbered 0 to 7. The uppermost page in physical memory is called the I/O page and is usually mapped by a Kernel Mode APR since it is a privileged area.



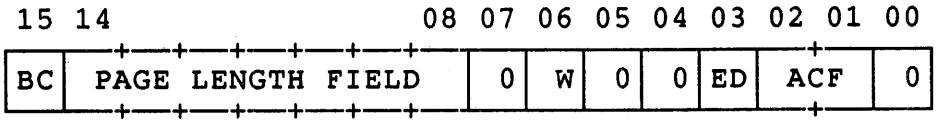
The Page Address Register consists solely of the Page Address Field (PAF). If 22-bit addressing is enabled, then all 16 bits of the PAR are used as the PAF. If only 18-bit addressing is desired then the 12 lower order bits are the PAF and the 4 higher order bits are unused. It may be thought of as a base register containing a base address or as a relocation register containing a relocation constant.

PAGE ADDRESS REGISTER (PAR)




The Page Descriptor Register contains information about access, page length, and expansion direction:

PAGE DESCRIPTOR REGISTER (PDR)




PDR <15> BC (Bypass Cache)

Set if memory accessing is wished to be without utilizing the cache. Useful with dual-ported memories.

PDR <08:14> PLF (Page Length Field)

Specifies the authorized length of the page in 32 word (or 64 byte) groups.

0 <—> 32 word page
.
.
177 (8) <—> 4096 word page

PDR <06> W (Written Into)

Useful in determining whether or not a page can simply be erased or must be saved to be brought back into memory.

0 <—> Page has NOT been written into
1 <—> Page has been written into

PDR <03> ED (Expansion Direction)

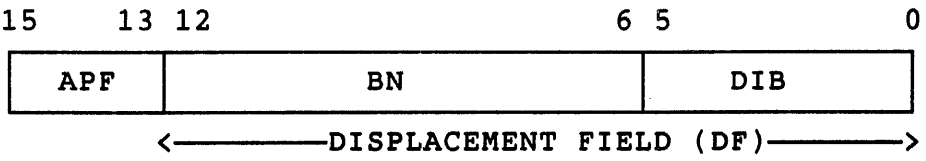
0 <—> Expands to higher addresses (Normally used)
1 <—> Expands to lower addresses (Can be used for stack segments)

PDR <01:02> ACF (Access Control Field)

00 <—> Nonresident
01 <—> Resident - Read Only
10 <—> Not Used
11 <—> Resident - Read/Write

The 16-bit virtual address is divided into three fields:

VIRTUAL ADDRESS (VA)




- o APF or Active Page Field
  These three bits determine which of the eight APRs are selected.

- o BN or Block Number
  The PAF determines an 8 Kb page in memory. The BN is that offset that is added to the base of the page determined by the PAF to obtain the block within the page to map.

o DIB or Displacement In the Block
Tells exactly which one of the 32 words is being mapped to. The low order 6 bits (DIB) are never relocated.

The BN and DIB fields are collectively referred to as the Displacement Field (DF).

If the MMU is not activated then 16-bit virtual addresses are also 16-bit physical addresses and linearly map the 64Kb address space. When the MMU is activated the 16-bit virtual address (VA) is no longer interpreted as a physical address Instead, the physical address (PA) is constructed using the VA and the PAF (Page Address Field) of the selected PAR.

The translation of virtual addresses to 22-bit physical addresses is accomplished as follows:

1) A set of registers is selected. This is determined by the space being referenced (Instructions or Data) and by the mode bits of the Processor Status Word (PSW <15:14>).

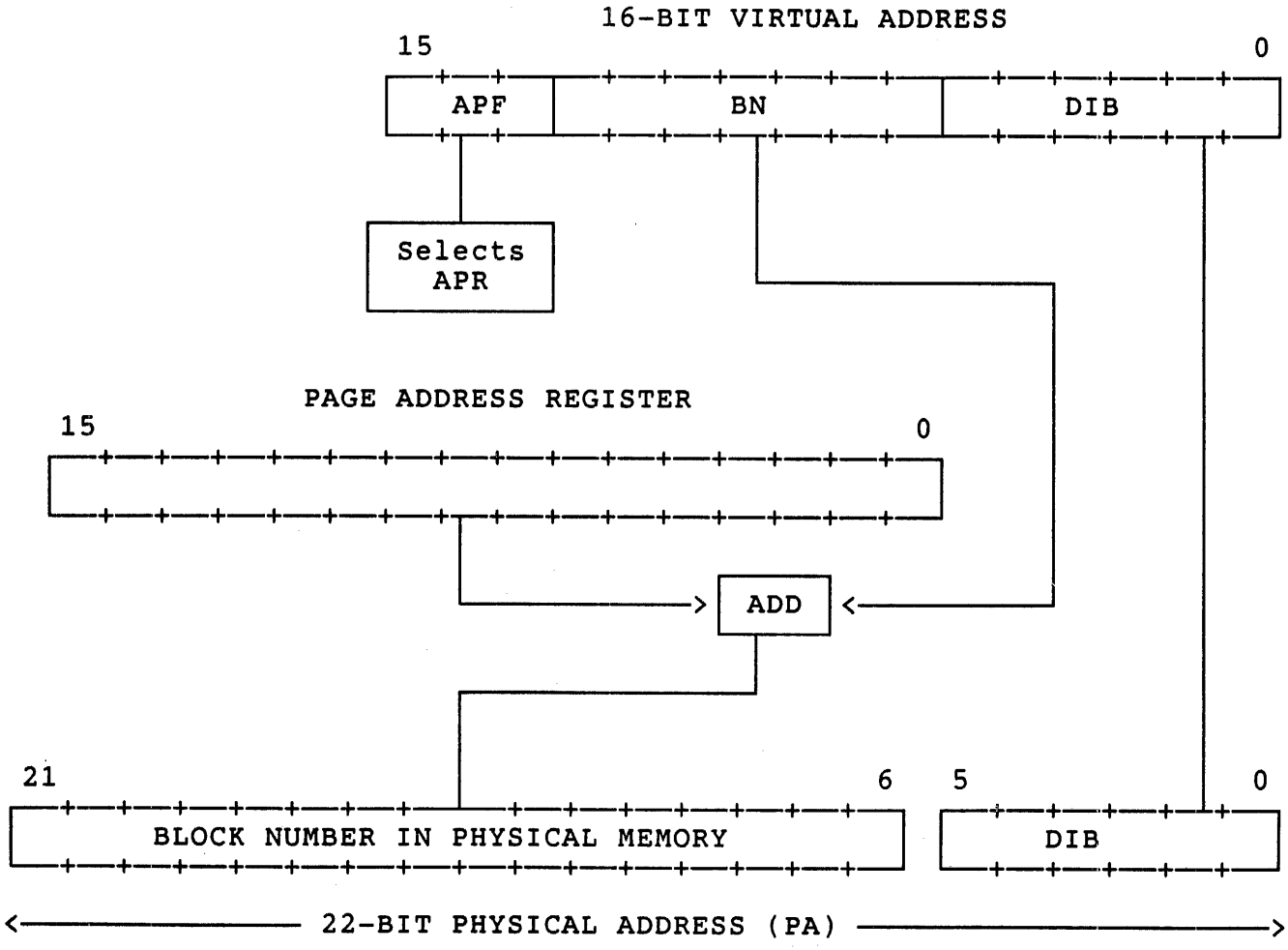
2) The APF field of the virtual address determines which of the eight pairs in the selected set will be used for the mapping.

3) The PAF of the selected register pair contains the starting address of the currently active page as a block number.

4) The block number from the virtual address and the block number from the PAF are added together. The block number from the PAF is shifted left by six bits in order to perform this addition. The result is the actual physical block number (bits <21:06> of the physical address).

5) The DIB of the virtual address is carried along unchanged as bits <05:00> of the translated address.

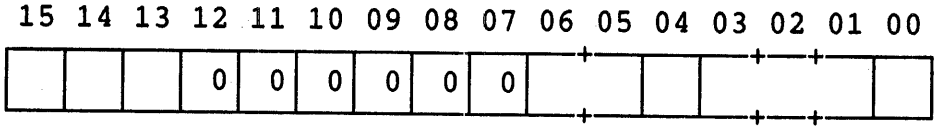
VIRTUAL TO PHYSICAL ADDRESS TRANSLATION



There are four memory management registers that are used for memory fault recovery and abort and status information as well as control of the MMU. They are called MMR0, MMR1, MMR2, and MMR3.

Memory Management Register 0 (MMR0) is the main control and status register.

MEMORY MANAGEMENT REGISTER 0 (MMR0)




MMR0 <15> NONRESIDENT ABORT

Set if an attempt is made to access a page with an ACF of 0 or 2 or if the PSW indicates mode 2.

MMR0 <14> PAGE LENGTH ABORT

Set if an attempt is made to access a location whose block number is outside the area authorized by the PLF of the PDR for that page.

MMR0 <13> READ ONLY ABORT

Set if an attempt is made to write to a page with an ACF of 1 (Read Only).

MMR0 <06:05> PROCESSOR MODE

Copy of the PSW <15:14> when abort occurred.

MMR0 <04> PAGE SPACE

1 <—> D-space
0 <—> I-space

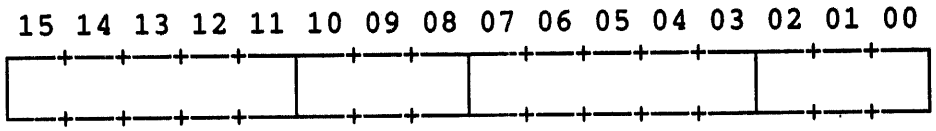
MMR0 <03:01> PAGE NUMBER

Page Number of page causing the abort.
Copy of APF of virtual address.

| | | | |
|---|---|---|---|
| MMR0 <00> | ENABLE RELOCATION | | |
| | 1 | <—> | ALL addresses are relocated (MMU activated) |
| | 0 | <—> | NO addresses are relocated (MMU disabled) |

Memory Management Register 1 (MMR1) is called the Instruction Backup Register. It records any autoincrement or autodecrement of any general purpose register. The lower byte is used for source operands and the destination operand may be in either byte, dependent upon the instruction.

MEMORY MANAGEMENT REGISTER 1 (MMR1)

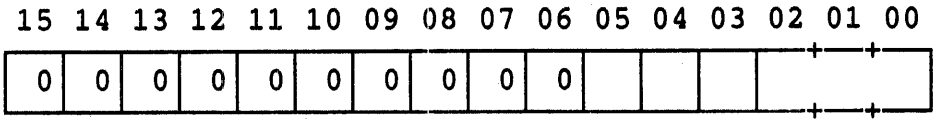



| | |
|---|---|
| MMR1 <15:11> | AMOUNT CHANGED (2's complement) |
| MMR1 <10:08> | REGISTER NUMBER |
| MMR1 <07:03> | AMOUNT CHANGED (2's complement) |
| MMR1 <02:01> | REGISTER NUMBER |

Memory Management Register 2 (MMR2) is known as the Last Virtual Program Counter. It is loaded with the value of the Program Counter at the beginning of each instruction fetch and is used in instruction fault recovery.

Memory Management Register 3 (MMR3) is used to select 18-bit or 22-bit address mapping and is used to enable/disable the data space for any of the processor modes.

MEMORY MANAGEMENT REGISTER 3 (MMR3)




MMR3 <05> UNINTERPRETED

May be set or cleared but is not interpreted.

MMR3 <04> ENABLE 22-BIT MAPPING

1 <—> Mapping is to 22-bit space.
0 <—> Mapping is to 18-bit space.

MMR3 <03> ENABLE CSM (Call Supervisor Mode) INSTRUCTION

1 <—> CSM is recognized.
0 <—> CSM is not recognized.

MMR3 <02> KERNEL DATA SPACE

1 <—> Enable data space mapping in Kernel Mode
0 <—> Disable data space mapping in Kernel Mode

MMR3 <01> SUPERVISOR DATA SPACE

1 <—> Enable data space mapping in Supervisor Mode
0 <—> Disable data space mapping in Supervisor Mode

MMR3 <00> USER DATA SPACE

1 <—> Enable data space mapping in User Mode
0 <—> Disable data space mapping in User Mode

Summary:

The LSI-11/73 Memory Management Unit provides a powerful, general purpose tool for memory management. It can be used to expand memory in a simple way and it can be used in a multiprogramming system to provide all the services necessary for an efficient and secure environment.

The following is a simple MACRO-11 program which illustrates the method of setting up the registers and performing some mappings. It writes a known value to an unmapped location and sets up the MMU registers and turns on the unit. It then writes another known value to the same virtual address which is now mapped. Then it turns the MMU off. At this point the two known values are in different memory locations.

```
        .TITLE  MMU

;
;       Program to demonstrate setting up MMU registers
;       and to illustrate the mapping that takes place
;

        PSW     =       177776  ; Processor Status Word
        MMR0    =       177572  ; Memory Management Register 0
        PDR0    =       172300  ; Page Descriptor Register 0
        PAR0    =       172340  ; Page Address Register 0
        PAR1    =       172342  ; Page Address Register 1
        PAR7    =       172356  ; Page Address Register 7

; INSURE THAT MMU IS NOT ACTIVATED

START:: CLR     @#MMR0          ; CLEAR MMR0

; STORE A KNOWN VALUE (152152) IN UNMAPPED LOCATION 20000

        MOV     #152152,@#20000

; SET PSW TO KERNEL MODE

        BIC     #140000,@#PSW

; SET THE PARS SO THAT EACH PAGE MAPS TO ITSELF

        MOV     #PAR0,R5        ; SET UP PAR POINTER
        MOV     #10,R4          ; SET UP PAR COUNTER
        CLR     R3              ; SET UP PAR OFFSET VALUE

LOOP1:  MOV     R3,(R5)+        ; SET EACH RELOCATION CONSTANT TO
                                ; MAP TO ITSELF
        ADD     #200,R3         ; UPDATE OFFSET
        SOB     R4,LOOP1        ; DO ALL OF THEM
```

```
; SET PAR1 TO MAP PAGE 1 TO PAGE 2

        MOV     #400,@#PAR1

; SET PAR7 TO MAP THE I/O PAGE TO THE TOP OF MEMORY

        MOV     #7600,@#PAR7    ; 172356 POINTS TO 760000

;
; SET UP THE PDRS
;
; 77406 (8) = 0 111 111 100 000 110 (2)
;
;       RESIDENT - READ/WRITE
;       UPWARD EXPANDING
;       NOT WRITTEN INTO
;       8KB PAGE SIZE
;       DON'T BYPASS CACHE
;

        MOV     #PDR0,R5        ; SET UP PDR POINTER
        MOV     #10,R4          ; SET UP PDR COUNTER

LOOP2:  MOV     #77406,(R5)+    ; SET EACH PDR
        SOB     R4,LOOP2        ; DO THEM ALL

; ENABLE THE MMU

        INC     @#MMR0          ; SET BIT 0 OF MMR0

; WRITE A VALUE TO LOCATION 20000. THIS SHOULD BE MAPPED.

        MOV     #107010,@#20000 ; WRITE 107010 TO MAPPED 2000

; DISABLE THE MMU

        DEC     @#MMR0          ; CLEAR BIT 0 OF MMR0

; AT THIS POINT IN THE PROGRAM, EXAMINING PHYSICAL
; ADDRESS 20000 SHOWS THE VALUE OF 152152 WHICH WAS PLACED THERE
; ADDRESS 40000 (WHICH 20000 MAPPED TO) HOLDS THE VALUE 107010

        HALT
        .END    START
```

| uNOTE # 009 |
|---|

| Title: Cache Concepts and the LSI-11/73 | Date: 02-JUL-84 |
|---|---|
| Originator: Charlie Giorgetti | Page 1 of 6 |

The goal is to introduce the concept of cache and its particular implementation on the LSI-11/73 (KDJ11-A). This is not a detailed discussion of the different cache organizations and their impact on system performance.

## What Is A Cache ?

The purpose of having a cache is to simulate a system having a large amount of moderately fast memory. To do this the cache system relies on a small amount of very fast, easily accessed memory (the cache), a larger amount of slower, less expensive memory (the backing store), and the statistics of program behavior.

The goal is to store some of the data and its associated addresses in the cache and all of the data at its usual addresses (including the currently cached data) in the backing store. If it can be arranged that most of the time when the processor needs data it is located in fast memory, then the program will execute more quickly, slowing down only occasionally for main memory operations. The placement of data in the cache should not be a concern to the programmer but is a consequence of how the cache functions.

Figure 1 is an example of a memory organization showing a cache with backing store. If the data needed by the microprocessor (uP) can be found in the cache then it is accessed much faster due to the local data path and faster cache memory than by having to access the backing store on the slower system bus.

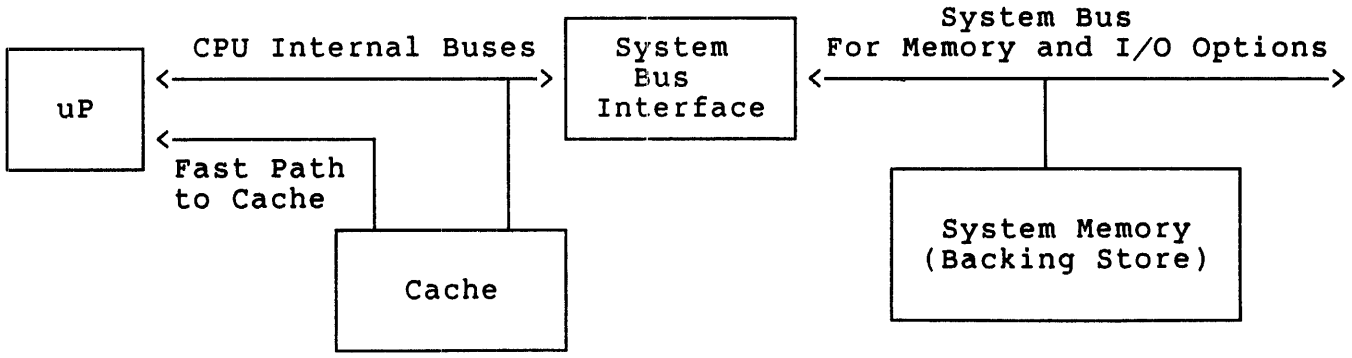



Figure 1 - An Example Memory System with Cache

A cache memory system can only work if it can successfully predict most of the time what memory locations the program will require. If a program accessed data from memory in a completely random fashion, it would be impossible to predict what data would be needed next. If this was the case a cache would operate no better then a conventional memory system.

Programs rarely generate random addresses. In many cases the subsequent memory address referenced is often very near the current address accessed. This is the principle of program locality. The next address generated is in the neighborhood of the current address. This behavior helps makes cache systems feasible.

The concept of program locality is not always adhered to, but is a statement of how many programs behave. Many programs execute code in a linear fashion or in loops with predictable results in next address generation. Jumps and context switching give the appearance of random address generation. The ability to determine what word a program will reference next is never completely successful and therefore the correct "guesses" are a statistical measure of the size and organization of the cache, and the behavior of the program being executed.

The measure of a cache performance is a statistical evaluation of the number of memory references found versus not found in cache. When memory is referenced and the address is found in the cache this is known as a hit. When it is not it is termed a miss. Cache performance is usually stated in terms of the hit ratio or the miss ratio where these are defined as:

$$\text{Hit Ratio} = \frac{\text{Number of Cache Hits}}{\text{Total Number of Memory References}}$$

$$\text{Miss Ratio} = 1 - \text{Hit Ratio}$$

## The LSI-11/73 Cache Implementation

The cache organization chosen must be one that can be implemented within the physical and cost constraints of the design.

The LSI-11/73 implements a direct map cache. A direct map organization has a single unique cache location for a given address and this is where the associated data from backing store are maintained. This means an access to cache requires one address comparison to determine if there is a hit. The significance of this is that a small amount of circuitry is

required to perform the comparison operation. The LSI-11/73 has an 8 KByte cache. This means that there are 4096 unique address locations each of which stores two bytes of information.

The cache not only maintains the data from backing store but it also includes other information that is needed to determine if its content is valid. These are parity detection and valid entry checking. The following diagram shows the logical layout of the cache and what each field and its associated address in the cache is used for.

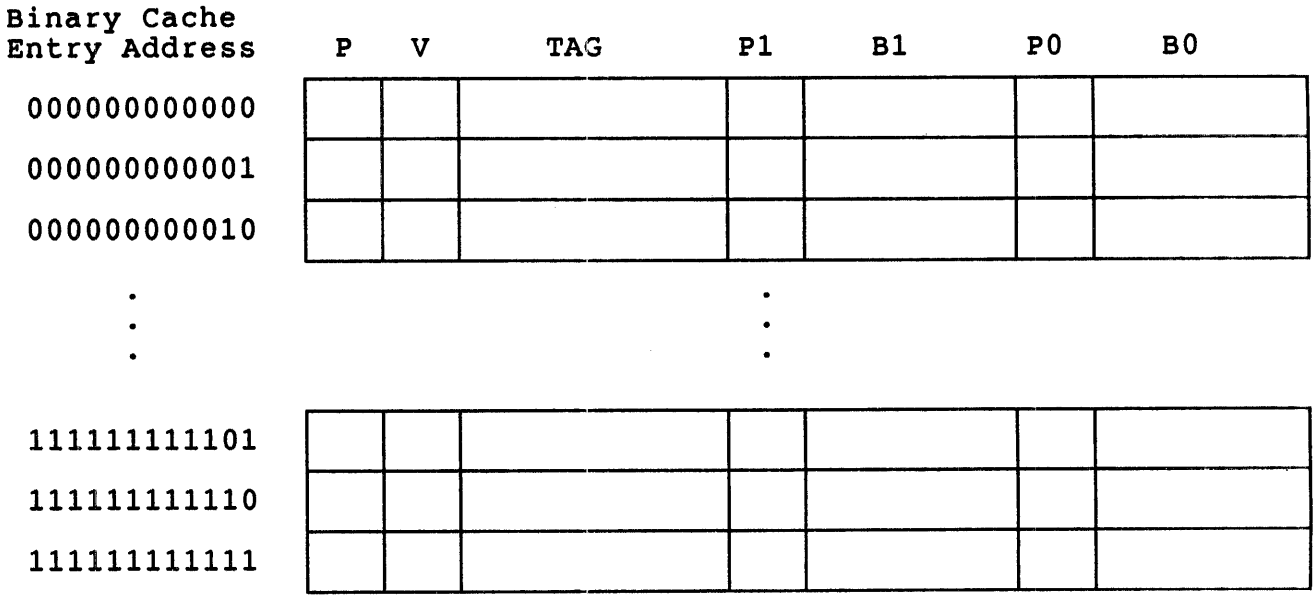



Figure 2 - LSI-11/73 Cache Layout

The Cache Entry Address is the address of one of 4096 entries within the cache. This value has a one-to-one relationship with a field in each address that is generated by the processor (described in the next section on how the physical address accesses cache).

Each field has the following meaning:

Tag (TAG) - This nine bit field contains information that is compared to the address label, described in the next section on how the physical address accesses cache. When the physical address is generated, the address label is compared to the tag field. If there is a match it can be considered a hit provided that there is entry validation and no parity errors.

Cache Data (B0 and B1) - These two bytes are the actual data stored in cache.

Valid Bit (V) - The valid bit indicates whether the information in B0 and B1 is usable as data if a cache hit occurs. The valid bit is set when the entry is allocated during a cache update which occurs as a result of a miss.

Tag Parity Bit - (P) - Even parity calculated for the value stored in the tag field.

Parity Bits (P0 and P1) - P0 is even parity calculated for the data byte B0 and P1 is odd parity calculated for the data byte B1.

When the processor generates a physical address, the on-board cache control logic must determine if there is a hit by looking at the unique location in cache. To determine what location to check, the cache control logic considers each address generated as being made up of three unique parts. The following are the three fields of a 22-bit address (in an unmapped or 18-bit environment the label field is six or four bits less respectfully):

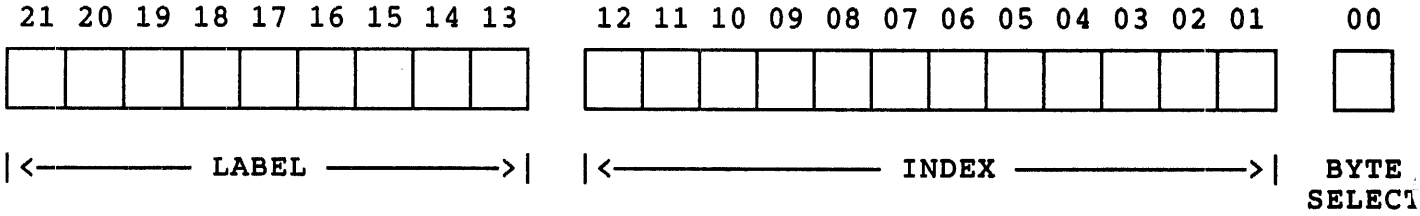



Figure 2 - Components of a 22-bit Address For Cache Address Selection

Each field has the following meaning:

Index - This twelve bit field determines which one of the 4096 cache data entries to compare with for a cache hit. The index field is the displacement into the cache and corresponds to the Cache Entry Address.

Label - Once the location in the cache is selected, the nine bit label field is compared to the tag field stored in the cache entry under consideration. If the address label and the tag field match, the valid bit is set, and there is no parity error, then a hit has occurred.

Byte Select Bit - This bit determines if the reference is on an odd or even byte boundary. All Q-bus reads are word only so this bit has no effect on a cache read. Q-bus writes can access either words or bytes. If there is a word write the cache will be updated if there is a hit. If there is a miss a new cache entry will be made. If there is a byte write, the cache will only be updated if there is a hit. A miss will not create a new entry on a byte write.

The LSI-11/73 direct map cache must update the backing store on a memory write. The LSI-11/73 uses the write through method. With this technique, writes to backing store occurs concurrently with cache writes. The result is that the backing store always contains the same data as the cache.

## Features Of The LSI-11/73 Cache

The LSI-11/73 direct map cache has a number of features that assist in the performance of the overall system in addition to the speed enhancement as a result of faster memory access. These features consist of the following:

- o Q-bus DMA monitoring
- o I/O page reference monitoring
- o Memory management control of cache access
- o Program control of cache parameters
- o Statistical monitoring of cache performance

The LSI-11/73 cache control logic monitors the Q-bus during DMA transactions. When an address that has its data stored in cache is accessed during DMA, the cache and backing store contents might no longer be the same. This is an unacceptable situation. The cache control logic invalidates a cache entry if the address is used during DMA. This also includes addresses used during Q-bus Block Mode DMA transfers.

Memory references to the I/O page are not cached since that data is volatile, meaning its contents can change without a Q-bus access. Since the cache could end up with stale data, I/O references are not cached.

There are situations for which using the cache to store information for faster access is not desirable. An example is a device that resides in the I/O page, and is true in other instances as well. One situation is a device that does not reside in the I/O page but can change its contents without a bus reference, such as dual ported memory.

Another situation is partitioning and tuning an application for instruction code execution versus data being manipulated. In this case the instruction stream may execute many times over for different data values. Speed enhancement can be obtained if the instructions are cached while the data is not cached. By forcing the data never to be cached it cannot replace instructions in the cache.

The memory management unit (MMU) of the LSI-11/73 can assist in this situation. Pages of memory allocated for data can be marked to bypass the cache and therefore not effect instructions that loop many times. The cache and the MMU work together to achieve the goal of increased system performance.

The dynamics of cache operation are under program control through use of the Cache Control Register (CCR), an LSI-11/73 on-board register. This

register can "turn" the cache on or off, force cache parity errors for diagnostic testing, and invalidate all cache entries. The details of the CCR are described in the KDJ11-A CPU Module User's Guide (part number EK-KDJ1A-UG-001).

During system design or at run-time the performance enhancements provided by the cache system can be monitored under program control. This is accomplished by using another LSI-11/73 on-board register the Hit/Miss Register (HMR). This register tracks the last six memory references and indicates if a hit or miss took place. The details of the HMR are also described in the KDJ11-A CPU Module User's Guide.

## Summary

Caches are a mechanism that can help improve overall system performance. The dynamics of a given cache are dictated by the organization and the behavior of the programs running on the machine. The LSI-11/73 cache is designed to be flexible in its use, simple in implementation, and enhance application performance.

More detailed discussions on how caches work and other cache organizations can be found in computer architecture texts that have a discussion of memory hierarchy.

| uNOTE # 010 |
|---|

| Title: MicroVAX I/O Programming | Date: 27-JUL-84 |
|---|---|
| Originator: Peter Jonhson | Page 1 of 5 |

The Qbus MicroVax implements the full Vax memory management, so virtual addresses are translated to physical addresses - just as they are for the Vax minicomputers. When memory management is enabled, system and process space virtual addresses are translated into physical addresses and sent onto the Qbus. Normally, the programmmer need not concern himself with this translation as this is completely handled by the operating system; however, when accessing locations in the I/O space directly the programmer must concern himself with the mapping since specific information must be supplied by the programmer to MicroVMS in order for it to successfully map the user's virtual addresses into the I/O space. The process of mapping a virtual address to a physical address must start by determining the physical address that you wish to access. In our case this means that we must calculate a Qbus MicroVax physical address given that we know the address that the Qbus board is configured to. In order to do this we must know a few key facts about the Qbus MicroVax architecture.

1) The Qbus MicroVax I/O space begins at physical 20000000 (hex)

2) The I/O space of a Qbus MicroVax is largely empty containing only Q22 bus I/O space which is 8K bytes long

Given these two pieces of information we now know that the I/O space for the Qbus MicroVax starts at physical 20000000 (hex) and extends to 20001FFF (hex). This I/O space directly corresponds to the configurable addresses on Qbus option boards 160000-177777 (oct). In order to convert any Qbus option address to a Qbus MicroVax address one simply subtracts 160000 from the boards configured address to get its offset into the I/O space and then add this value to the base of the Qbus MicroVax I/O space. For example, let us consider a board which has been configured to - 166540. In order to calculate its equivalent Qbus MicroVax address we would do the following:

1) Subtract off the I/O base address for this board (160000)

```
    166540 (oct)
-   160000 (oct)
-------------
      6540 (oct) ---> D60 (hex)
```

2) Add the board's address offset to the Qbus MicroVax I/O space base address

```
    20000000 (hex)
+         D60 (hex)
-------------------
    20000D60 (hex)
```

This addition results in the physical address on the Qbus MicroVax system that this board would answer to.

Now we have calculated the physical address for the board in the Qbus MicroVax environment. This value, however, is in the raw state and is still not usable by the uVMS software to perform virtual to physical address translation. In order for this address to be useful to the software it must now be converted from a physical address to a page frame number. The page is the basic unit of memory mapping and protection. A page is 512 contiguous byte locations. A page frame number (PFN) is the address of the first byte of a page in physical memory. This means that the lsb of a PFN has the resolution of 1 page or 512 bytes. It is a simple matter now to convert a physical address to a PFN. Since the lsb of a PFN is 1 page to convert a physical address to PFN just shift right the physical address 9 bits, i.e. shift off the 9 least significant bits.

physcial 20000D60 (hex) shift right 9 ---> PFN 100006 (hex)

The PFN value which we have calculated is sufficient to allow the system to map a physcial page of addresses into your virtual address space. The address of the Qbus option resides somewhere in the page which we have mapped. It is the responsibility of the programmer to correctly offset from the beginning of the page in order to access the board i.e. the programmer must displace from the base of the mapped page with the correct virtual address offset. To determine the offset from the beginning of the mapped page look to bits 0-9 of the configured address of the Qbus option board - this is the offset. In our example the offset would be:


This offset would be used by the programmer to access the device registers on the board. Failure to use the offset will usually result in an attempt to access non-esxistent locations (analagous to memory time-outs) which will result in an access violation error being returned to the user. A sample program follows which illustrates the principles which have been discussed. It uses the same board address discussed earlier so that one can see the code needed to acutally implement the previous example.

The following program illustrates, in a raw fashion, how one might actually access the I/O space with software. It is not meant to illustrate good or recommended programming practice but rather to show the mechanics of accessing the I/O space of a Qbus on MicroVax I.

```
.title  map_io_space

;This program will allow a user with suitable privilege to access
;device registers in the I/O space of a uVax.  This example shows
;code which is used to extract data from a DRV11 - a general purpose
;parallel interface which resides in I/O space.  It is sufficiently
;general to allow the concept to be used in other situations where
;access to the I/O space from a user process is desireable.


;This portion of the program is responsible for creating the
;virtual to physical mapping required to access registers in
;the I/O space.  For this example the device registers are assumed to
;start at 166540.  In order for virtual to physical mapping to occur
;the  user must calculate the physcial page frame that the device
;registers overlap into.  (See accompanying text for how to do this)



;Create and map section directive does the actual work of mapping.
;In order for this system service to work correctly the
;use must have PFNMAP privilege

.entry  start,^m<>

$crmpsc_s -
inadr = maprange, -             ;Input virtual address range
retadr = actadr, -              ;Virtual address range acutally
                                ;mapped
pagcnt = #1, -                  ;Non zero required for page frame
                                ;section
flags -#<sec$m_expreg!sec$m_pfnmap!sec$m_wrt>, -
vbn = pfn_number                ;Actual page frame of I/O space
                                ;page which contains device registers

blbs    r0,continue             ;check the return status of create
                                ;and map - if successful branch

pushl   r0                      ;put status onto stack
calls   #1,g^lib$stop           ;tell him what happened and
                                ;stop program
```

```
;At this point one page of virtual addresses in the user's process
;space is now mapped into one page of the I/O space.  Now the user
;can access device registers by using move instructions.
;...... ** Please note that only certain instructions are allowed when
;doing physical I/O to the bus.  For example a MOVL instruction is
;NOT legal to the I/O space.


continue:

movl    actadr, r6              ;Get starting virtual address from


                                ;system service and put into R6
addl    #^x160, r6              ;Add the displacement into this page
                                ;to access the device registers

;Access data from the parallel interface

tstb    (r6)                    ;Test for data transfer request
bgew    10$                     ;If no data exit
movw    4(r6), data_in_buffer   ;Data is present - store it


10$:
$exit_s -                       ;exit gracefully
code = #1


;data for program ......

maprange:       .long   ^x400
                .long   ^x800

control_status  .word   0
data_in_buffer  .word   0

actadr:         .blkl   2               ;holds returned virtual
                                        ; address range
pfn_number:     .long   ^x100006        ;actual pfn of I/O page to
                                        ;be mapped

.end    start
```

| uNOTE # 011 |
|---|

| Title: LSI-11/73 ADVANCED MEMORY MANAGEMENT | Date: 04-Sep-84 |
|---|---|
| Originator: Art Bigler | Page 1 of 11 |

This micronote examines the advanced memory management features available on the DCJ11 based LSI-11/73 series processors (KDJ11-A, KDJ11-B). These features include the standard virtual address relocation within the physical address space and the kernel and user execution modes, all of which are currently available as options on the mid-range LSI-11/23 (KDF11-A, KDF11-B) processors. In addition to these features, the DCJ11 based processors also provide instruction and data space (I/D space) memory management and the supervisor execution mode. The following discussion is intended to further clarify these features. For information pertaining to address relocation the reader is referred to micronote 008, MEMORY MANAGEMENT AND THE LSI-11/73.

## 1.0 INSTRUCTION/DATA SPACE MEMORY MANAGEMENT

I/D space memory management is utilized in the DCJ11 based processors IN ADDITION TO the relocation of virtual addresses within the physical address space. This provides the ability to place multiple program images in physical memory while at the same time providing an increased virtual address space of 128 kb or 64 kw by mapping instructions and data to separate areas of physical memory. The means by which I/D space memory management is attained involves both hardware and software as described in the following paragraphs.

### 1.1 I/D SPACE HARDWARE

The hardware required to implement I/D space addressing is integrated into the memory management unit and is standard on all DCJ11 based processors. This includes the following:

1. Eight (8) additional active page registers (APR's) per execution mode (more about execution modes later). These APR's are used to map to the data space when I/D space memory management is enabled.

1. Additional control and status bits in memory management registers 0 and 3 (MMR0, MMR3) which are used to control the enabling and disabling of data space addressing. INSTRUCTION SPACE ADDRESSING IS ALWAYS ENABLED.

### 1.1.1 ACTIVE PAGE REGISTERS

The hardware provides a total of sixteen (16) APR's per execution mode, eight (8) instruction space registers and eight (8) data space registers. THE APR's are further divided into page descriptor registers (PDR's) and page address registers (PAR's) as described in micronote 008. The physical addresses for these registers are contained in the I/O page and are as follows:

| MODE | PAR's | PDR's | PAGE |
|---|---|---|---|
| KERNEL I SPACE | 17772340 | 17772300 | 0 |
| | 17772342 | 17772302 | 1 |
| | 17772344 | 17772304 | 2 |
| | 17772346 | 17772306 | 3 |
| | 17772350 | 17772310 | 4 |
| | 17772352 | 17772312 | 5 |
| | 17772354 | 17772314 | 6 |
| | 17772356 | 17772316 | 7 |
| KERNEL D SPACE | 17772360 | 17772320 | 0 |
| | 17772362 | 17772322 | 1 |
| | 17772364 | 17772324 | 2 |
| | 17772366 | 17772326 | 3 |
| | 17772370 | 17772330 | 4 |
| | 17772372 | 17772332 | 5 |
| | 17772374 | 17772334 | 6 |
| | 17772376 | 17772336 | 7 |

TABLE 1a
KERNEL MODE APR'S

| MODE | PAR's | PDR's | PAGE |
|---|---|---|---|
| SPVSR I SPACE | 17772240 | 17772200 | 0 |
| | 17772242 | 17772202 | 1 |
| | 17772244 | 17772204 | 2 |
| | 17772246 | 17772206 | 3 |
| | 17772250 | 17772210 | 4 |
| | 17772252 | 17772212 | 5 |
| | 17772254 | 17772214 | 6 |
| | 17772256 | 17772216 | 7 |
| SPVSR D SPACE | 17772260 | 17772220 | 0 |
| | 17772262 | 17772222 | 1 |
| | 17772264 | 17772224 | 2 |
| | 17772266 | 17772226 | 3 |
| | 17772270 | 17772230 | 4 |
| | 17772272 | 17772232 | 5 |
| | 17772274 | 17772234 | 6 |
| | 17772276 | 17772236 | 7 |

TABLE 1b
SUPERVISOR MODE APR'S

| MODE | PAR's | PDR's | PAGE |
|---|---|---|---|
| USER I SPACE | 17777640 | 17777600 | 0 |
| | 17777642 | 17777602 | 1 |
| | 17777644 | 17777604 | 2 |
| | 17777646 | 17777606 | 3 |
| | 17777650 | 17777610 | 4 |
| | 17777652 | 17777612 | 5 |
| | 17777654 | 17777614 | 6 |
| | 17777656 | 17777616 | 7 |
| USER D SPACE | 17777660 | 17777620 | 0 |
| | 17777662 | 17777622 | 1 |
| | 17777664 | 17777624 | 2 |
| | 17777666 | 17777626 | 3 |
| | 17777670 | 17777630 | 4 |
| | 17777672 | 17777632 | 5 |
| | 17777674 | 17777634 | 6 |
| | 17777676 | 17777636 | 7 |

TABLE 1c
USER MODE APR'S

### 1.1.2 MEMORY MANAGEMENT REGISTER 0

Memory management register 0 (MMR0) contains control and status information for the memory management unit (MMU). This register is discussed completely in micronote 008, to which the reader is again refferred for information on those functions which are not directly applicable to I/D space and supervisor mode.

MMR0 contains three (3) status bits which are used in the implementation of I/D space memory addressing. These bits, 04 through 06, yield MMU status information whenever a MMU abort occurs and are used in

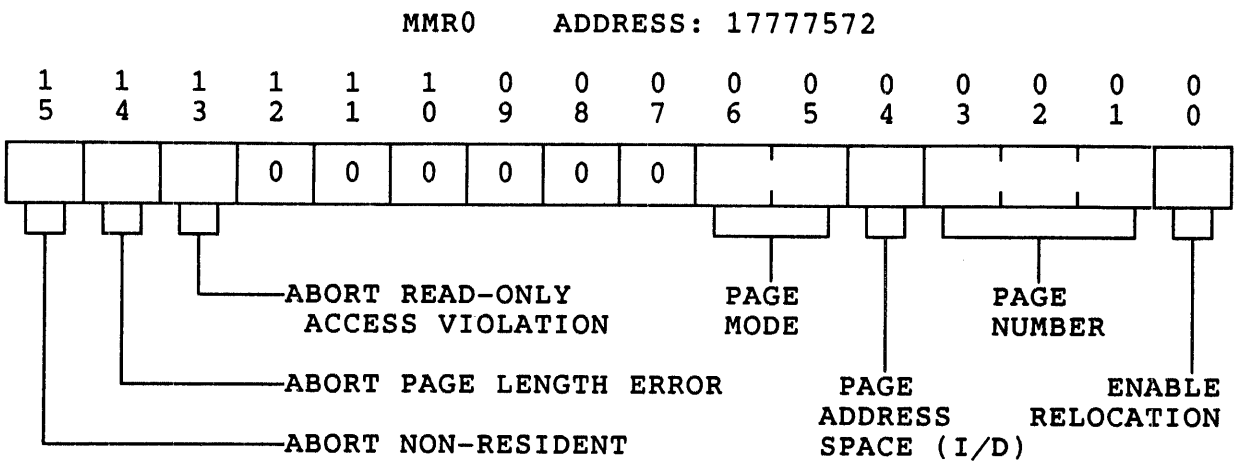
conjunction with MMR0 bits 01 through 03 and 13 through 15 to provide complete execution mode and I/D space status for the page causing the abort. See figure 1.

Bit 04, the page address space status bit, indicates the address space associated with the aborted page and is equal to a zero (0) for an instruction space page and a one (1) for a data space page whenever I/D space addressing is enabled. If I/D space addressing is not enabled this bit always reflects a zero (0).

Bits 05 and 06, the processor mode status bits, indicate the processor execution mode associated with the page causing the abort. These bits are coded as follows:

| BIT 06 | BIT 05 | EXECUTION MODE |
|---|---|---|
| 0 | 0 | KERNEL |
| 0 | 1 | SUPERVISOR |
| 1 | 0 | ILLEGAL (causes an abort with bit 15 set) |
| 1 | 1 | USER |

For more information on MMU aborts see micronote 008.




| BIT # | | DESCRIPTION |
|---|---|---|
| <15> | - | ABORT READ-ONLY ACCESS VIOLATION (R ONLY) |
| <14> | - | ABORT PAGE LENGTH ERROR (R ONLY) |
| <13> | - | ABORT NON-RESIDENT (R ONLY) |
| <12:07> | - | NOT USED (R ONLY) |
| <06:05> | - | PAGE MODE (R ONLY) |
| <04> | - | PAGE ADDRESS SPACE (I/D) (R ONLY) |
| <03:01> | - | PAGE NUMBER (R ONLY) |
| <00> | - | ENABLE RELOCATION (R/W) |

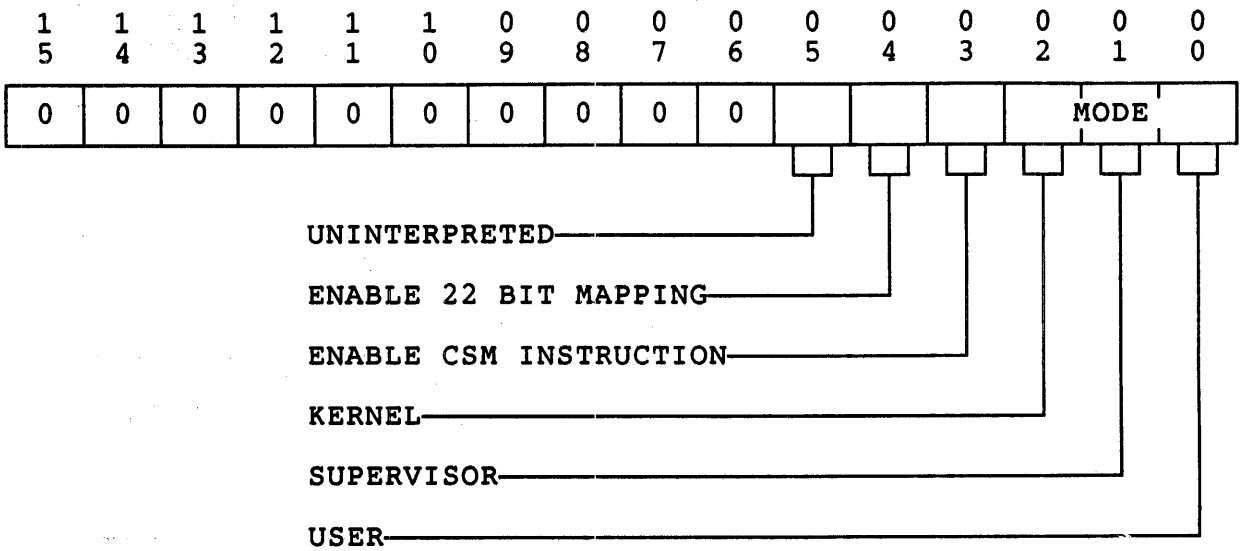
FIGURE 1
MEMORY MANAGEMENT REGISTER 0 (MMR0)

### 1.1.3 MEMORY MANAGEMENT REGISTER 3

Memory management register 3 (MMR3) contains control and status information for data space addressing, 22 bit mapping, and the call to supervisor mode (CSM) instruction. This register, once again, is discussed in detail in micronote 008.

MMR3 contains three (3) control bits which are used in the implementation of I/D space addressing. These bits, 00 through 02, individually enable data space addressing for each of the execution modes. Bit 00 enables data space addressing for the USER mode, bit 01 enables it for SUPERVISOR mode, and bit 02 enables it for KERNEL mode. The desired bits are set to a one (1) whenever data space addressing is desired. MMR3 is cleared during power-up, console restart, and the execution of the RESET instruction. See figure 2.

MMR3 REGISTER ADDRESS: 177725$_{16}$




| BIT # | | DESCRIPTION |
|---|---|---|
| <15:06> | - | NOT USED (R ONLY) |
| <05> | - | UNINTERPRETED (R/W) |
| <04> | - | ENABLE 22 BIT MAPPING (R/W) |
| <03> | - | ENABLE CSM INSTRUCTION (R/W) |
| <02> | - | KERNEL DATA SPACE (R/W) |
| <01> | - | SUPERVISOR DATA SPACE (R/W) |
| <00> | - | USER DATA SPACE (R/W) |

FIGURE 2
MEMORY MANAGEMENT REGISTER 3

### 1.1.4 I/D SPACE ADDRESS MAPPING

When I/D space addressing has been enabled the MMU hardware performs the address mapping (IN ADDITION TO ADDRESS RELOCATION WHICH IS PERFORMED USING THE APPROPRIATE SET OF APR'S) as follows:

1. The current instruction is ALWAYS fetched from the instruction space.

2. The operands are mapped according to table 2.

| OPERAND ADDRESSING MODE | REGISTER USED | TYPE OF ADDRESSING | I OR D SPACE USED |
|---|---|---|---|
| 000 | ANY | REGISTER | I |
| 001 | ANY | REGISTER DEFERRED | D |
| 010 | 0 THROUGH 6 | AUTOINCREMENT | D |
| | 7 | IMMEDIATE | I |
| 011 | 0 THROUGH 6 | AUTOINCREMENT DEFERRED | D (A)<br>D (D) |
| | 7 | ABSOLUTE | I (A)<br>D (D) |
| 100 | 0 THROUGH 6 | AUTODECREMENT | D |
| | 7 | DO NOT USE !! | |
| 101 | 0 THROUGH 6 | AUTODECREMENT DEFERRED | D (A)<br>D (D) |
| | 7 | DO NOT USE !! | |
| 110 | ANY | INDEX | I (A)<br>D (D) |
| 111 | ANY | INDEX DEFERRED | I (A)<br>D (A)<br>D (D) |

(A) = INDIRECT OR INDEX ADDRESS
(D) = DATA

TABLE 2
OPERAND ADDRESSING WITH DATA SPACE ENABLED

All address mapping is performed using the I space APR's when data space addressing is not enabled.

The most difficult example showing data space addressing is the index deferred type of addressing.

```
CLR     @1000(R3)
```

1. The instruction is fetched from the instruction space at location PC.

2. The base address 1000 is fetched from the instruction space at location PC+2. The index in R3 is added to the base address forming the address of the indirect address.

3. The indirect address is fetched from the data space using the address calculated in step 2.

4. The data is fetched from the data space using the address calculated in step 3.

## 1.2 I/D SPACE SOFTWARE

At the present time I/D space addressing is supported by two (2) Digital supplied operating systems, RSX-11M-PLUS and ULTRIX-11.

RSX-11M-PLUS provides linking of tasks which utilize I/D space addressing via the task builder (TKB) utility. Those programs which include the data PSECTs in their object files may be task built using the /ID switch. It should be noted that the task may not make use of the entire 32kw data space because RSX-11M-PLUS requires that the stack and the task header be placed in data space. Other restrictions may apply, consult the task builder manual for further information.

When using I/D space with other operating systems or in standalone programs, the user must do all the mapping within the program. This implies that the mapping of the operating system must be attended to by the user program if operating system features are to be utilized.

To make use of data space addressing the program must:

1. Separate the instruction space from the data space. (ie. create different regions in memory for instructions and data)

2. Load the instruction space and data space APR's with the appropriate relocation information.

3. Enable I/D space mapping by setting the MMR3 bit associated with the execution mode under which the program will run.

The following restrictions apply to I/D space programs:

1. The instruction space can only contain instructions, immediate operands, absolute addresses, and index words. This is reflected in table 2.

2. The stack page must be mapped into both instruction and data

space if the MARK instruction is used because it is executed off the stack.

3. Instruction space-only pages cannot contain subroutine parameters which are data. This precludes the mapping of any pages containing standard PDP-11 calling sequences entirely into an instruction space page.

4. The trap catcher technique of putting .+2 in the trap vector followed by a halt must be mapped into both instruction and data space.

For further information on I/D space addressing under RSX-11M-PLUS and ULTRIX-11 consult the appropriate documentation set.

## 2.0 SUPERVISOR MODE

The DCJ11 based processors provide three (3) execution modes: KERNEL, SUPERVISOR, and USER. They provide for various forms of memory and processor protection and permit additional features to be implemented in multiprogramming environments. Each mode has its own set of mapping registers.

KERNEL mode is the most privileged of the modes, allowing the execution of any instruction and the modification of any area in memory including the I/O page.

USER mode prohibits the execution of privileged instructions such as HALT and RESET and the modification of areas in memory that the KERNEL program does not provide access to.

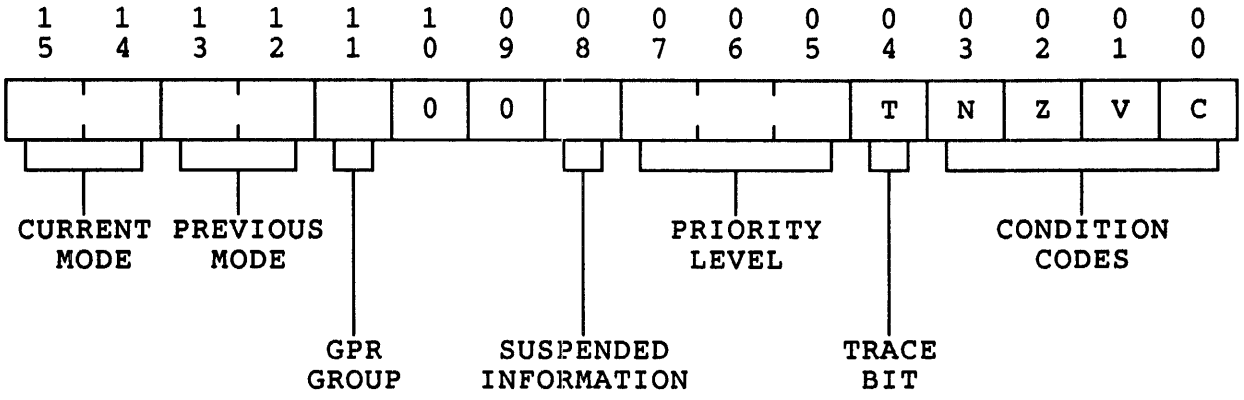
SUPERVISOR mode has the same privileges as USER mode with its own set of mapping registers, thus providing another level of protection. SUPERVISOR mode is intended for use in the mapping and execution of programs to be shared by users while still providing protection from them. Examples of this are command line processors which are required for use by all users on a system, while necessitating write protection from them.

The execution mode is controlled by the state of bits 14 and 15 in the processor status word (PSW). These bits are changed by the execution of traps and interrupts, pushing and popping of old PSW's to and from the stack, and, when in KERNEL mode, the direct manipulation by the program. Bits 12 and 13 reflect the execution mode which existed prior to the event which placed the processor in the current mode. See figure 3.

The current and previous mode PSW bits are coded as follows:

| BIT 15 14 13 12 | EXECUTION MODE |
|---|---|
| 0 0 | KERNEL |
| 0 1 | SUPERVISOR |
| 1 0 | ILLEGAL |
| 1 1 | USER |

PROCESSOR STATUS WORD (PSW) ADDRESS: 17777776




| BIT # | DESCRIPTION |
|---|---|
| <15:14> - | CURRENT MODE (R/W) |
| <13:12> - | PREVIOUS MODE (R/W) |
| <11> - | GENERAL PURPOSE REGISTER SET (R/W) |
| <10:09> - | NOT USED (R ONLY) |
| <08> - | SUSPENDED INFORMATION (R/W) |
| <07:05> - | PROCESSOR PRIORITY LEVEL (R/W) |
| <04> - | TRACE BIT (R/W) |
| <03> - | NEGATIVE CONDITION CODE (R/W) |
| <02> - | ZERO CONDITION CODE (R/W) |
| <01> - | OVERFLOW CONDITION CODE (R/W) |
| <00> - | CARRY CONDITION CODE (R/W) |

FIGURE 3
PROCESSOR STATUS WORD

| uNOTE # 012 |
|---|

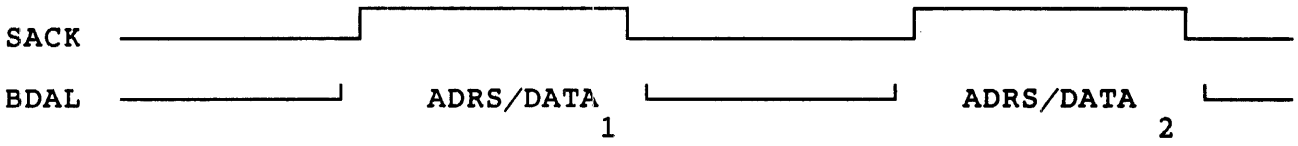
| Title: DMA on The Q-bus | Date: 06-SEP-84 |
|---|---|
| Originator: Jack Toto | Page 1 of 4 |

This Micronote explains the various types of DMA on the Q-bus; Single Cycle Mode, Burst Mode and Block Mode.

## SINGLE CYCLE MODE:

Single cycle mode DMA like all DMA on the Q-bus requires that the DMA device gain control of the bus through an arbitration cycle. During the arbitration cycle the DMA device becomes bus master by first asserting a DMA request (BDMR). When the arbiter acknowledges this request it issues a DMA grant (BDMGO). In the event that there is more than one DMA device in the backplane the grant signal is daisy chained from device to device. Eventually the device that issued the DMA request will latch the grant signal and take control of the bus, and proceed with the DMA transfer.

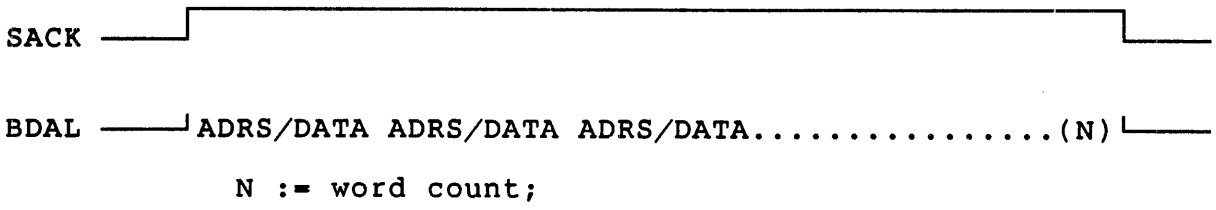
Once becoming bus master the device asserts BSACK and is allowed to do one word tranfer to or from memory, during which time the CPU is idle. Certain processors such as the KDJ11-B have a cache memory with dual tag store which allows it to process data while DMA transfers are occurring. Regardless of which processor type is used only one transfer is allowed in single cycle mode. If the device must perform additional transfers, it must go through the bus arbitration cycle again.




In single cycle mode, the theoretical transfer rate across the Q-bus is 1.66 Mbytes/sec (833Kw/sec) A device such as the DRV11-B or the newer 22-bit compatible DRV11-W can transfer data at a rate of 250KW/sec while in single cycle mode.

## BURST MODE:

Burst mode DMA can be performed by certain devices such as the DRV11-B. Once the DMA controller becomes bus master (through the arbitration routine described in the single cycle section and it has asserted BSACK, the DMA tranfers can begin. Each data word that is transfered is accompanied by an address that the data word is targeted for. In burst mode loading an octal value into the 16 bit word count register (WCR) allows for that number of words (64Kb max) to be transfered under one sack. This differs from single mode, in that the word count register can be loaded with the same value, but each single word transfer will require a new arbitraion cycle, i.e in order to transfer 64Kb of data it would require 65,536 arbitration cycles.



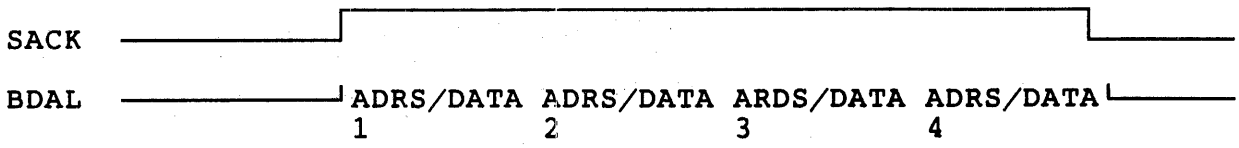

N := word count;

The theoretical transfer rate across the Q-bus in burst mode remains at 1.66 Mbytes/sec, however a device such as the DRV11-B operating in burst mode can transfer data at a rate twice that of a DRV11-B operating in single mode, or 500Kw/sec. In burst mode the DMA bus master maintains control of the bus until it has transferred all of the required data. Burst mode has the advantage of moving large blocks of memory across the bus with no delay. The caution here is that no other device (including the CPU) has access to the bus during that time. This can have severe impact on system performance.

## DMA COMPROMISE:

Since Single Cycle Mode requires a rearbitration for every data transfer and Burst Mode can adverseley impact system performance in some cases DIGITAL EQUIPMENT CORP. has made some compromises with certain DMA controllers. Most DIGITAL devices will do a limited Burst Mode operation. These controllers (for example the RXV21 and RLV12) are

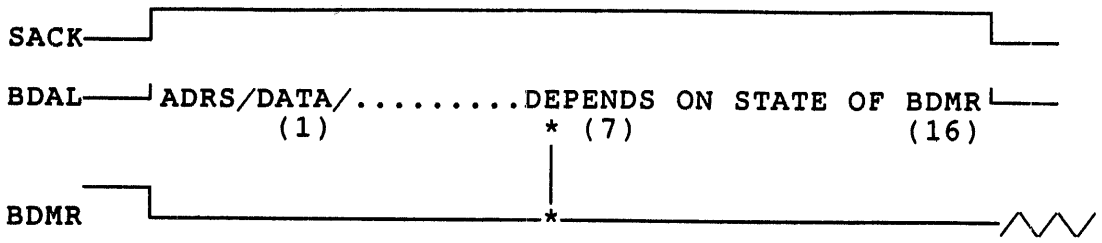
allowed up to do four words of data tranfer. Each word of transfer is preceeded on the bus by an address that the data word is targeted for. This allows data to move across the bus with a minimum of rearbitraton. However, when a group of four transfers is finished, the DMA devices must again go through the arbitration cycle in order to allow other devices the opportunity to use the bus. If no other bus requests are pending at a higher priority, then bus mastership will be returned to the device for the next set of data transfers.




BLOCK MODE:

For increased throughput, Block Mode DMA may be implemented on a device for use with memories that support this type of transfer. Block Mode DMA devices are only block mode when operating. They may not operate as a Single or Burst Mode device. They may, however, appear to operate like a single mode device if they are only doing a single word transfer, and they will always look like a Single Cycle Mode DMA device when used with non-Block Mode memory.

Once a Block Mode device has arbitrated for the bus, the starting memory address is asserted, then data for that address, followed by data for consecuetive addresses. By eliminating the assertion of the address for each data word, the transfer rate is almost doubled. The DMA device should monitor the BDMR line. If the line is not asserted after the seventh transfer than the device can continue. This allows a maximum of 16 data transfers for one abitration cylce. If the BDMR line is not monitored by the DMA device than a maximum data tranfer of 8 words is allowed after completing one bus arbitration cycle. Block Mode DMA transactions can be described as two types, a DATBI (block mode data in) and DATBO (block mode data out). Both of these cycles are explained in depth in Micronote #002. When reading the appropriate micronote special attention should be paid to the use of BREF and BBS7 signals when performing a DATBO.




Block Mode devices such as the DEQNA, RQDX1 and the MSV11-P memories can transfer data across the bus at rates that approach twice that of DMA devices in Burst Mode. The actual rate is dependent upon the device itself. The technical manuals for each of these devices should be checked for actual performance figures.

| Title: Run-time System Performance Evaluation Using MicroPower/Pascal V 1.5 | Date: 09-Oct-84 |
|---|---|
| Originator: Herbert Maehner | Page 1 of 5 |

In real-time programming, the performance of the run-time system and the compiler together govern the overall power of the application. The performance of the MicroPower/Pascal compiler has been extensively discussed by R.Billig/R.Cronk [1].

The performance of the run-time executive of MicroPower/Pascal is measured in this MicroNote using different LSI-11 CPU-boards and the KXT11-CA I/O processor. Data was obtained using MicroPower/Pascal version 1.5.

## Test Conditions

Results obtained through a lab experiment are only as precise as the test environment and may only be referenced giving the exact test conditions.

The goal was to measure the elapsed time of a given primitive execution on the Pascal process level, i.e. how long it takes to call/execute a kernel primitive from a Pascal program.

Generally, the following procedure was used to obtain the elapsed time, where in some cases more than one output bit has been used to obtain the desired pulse-width.

```
WHILE Condition = TRUE DO
BEGIN
  Out_port.bit0 := TRUE;

    { here call/execute given primitive }

  Out_port.bit0 := FALSE;
END;
```

The whole test was done in a loop as long as the condition was true. Here, Condition is a boolean variable, which is set FALSE by a high priorty process waiting on a terminal input (READLN).

The Outport.bit0 is bit0 of a parallel device. The parallel device was

either a DRV11 (using LSI 11/23 and LSI 11/73) or the on-board parallel device (using the FALCON plus SBC 11/21 and KXT11-CA).

The bit0 pulse is used as the input to an oscilloscope which has the capabilities to measure and display time differences and frequencies. The elapsed time required to execute the various primitives was obtained. In addition to the primitive requests some math-functions times were obtained. The results are shown in Table 1 at the of this MicroNote.

## Interrupt-Test Conditions

A Pascal-program with an embedded interrupt service routine needs DRIVER privileges in a mapped environment. The test program either connects to a "normal" ISR or to a prio7 ISR. The program executed a simple loop like:

```
WHILE TRUE DO
BEGIN
 Out_port.Bit1 := TRUE;
 Out_port.Bit1 := FALSE;
END;
```

The Outport is the parallel device of the type mentioned above. This is used to monitor process execution behavior. Testing the interrupt response time, we used a square wave generator which triggered an interrupt on that parallel device. The ISR was coded as

```
            .ENABL  GBL             ; Enable global symbols
            .MCALL  MACDF$,PURE$    ; Set-up pure/impure area
            .MCALL  IMPUR$

            .GLOBL  INPORT,OUTPRT   ; port A,B of PPI

            MACDF$                  ; MACDF$ must be called before the
            PURE$                   ; two assembly directives

            .DSABL AMA

PPIINT::
            BISB    #1,@#OUTPRT     ; set bit 0 output port   ----------
            MOVB    @#INPORT,@#Temp ; dummy read                | ISR time
            MOVB    @#INPORT,@#Temp ; dummy read                | measured
            BICB    #1,@#OUTPRT     ; set bit 0 output port   ----------
            RTS     PC              ; normal ISR return
            RTS     R4              ; prio 7 ISR return

            IMPUR$
TEMP:       .WORD 0                 ; reserve one word
            .END
```

Depending upon the ISR-type either the RTS PC or the RTS R4 must be used to exit the ISR. The first MACRO-statement within the ISR signaled bit0 of the parallel port. The resulting interrupt dispatch time was defined as the pulse width given by the square wave generator edge and the signaled output port. This includes the hardware ISR dispatch time as well. The ISR execution time was given by the pulse width indicated within ISR source above. Again, all pulses were measured using an oscilloscope. The maximum interrupt rate was determined by increasing the square frequency (which in turn increases the outpt square wave of the ISR) until the system lost interrupts.

Using CONNECT SEMAPHORE the interrupt performance was similarly measured. In this case only a dynamic process was waiting on the semaphore to be signaled. The results are shown in table 2.

References

1. Rich Billig and Randy Cronk, A System/Architecture Approach to Microcomputer Benchmarking, DIGITAL Equipment Corporation, Sept. 1982, EZ-12053-03/82

2. MicroPower/Pascal Newsletter, Volume 1, No. 1, March 1984, p. 23, DIGITAL Equipment Corporation, Order Number AV-B067A-TK

| Operation | LSI-11/23 w/o FPU u | LSI-11/23 w/o FPU m | LSI-11/23 w/ FPU u | LSI-11/23 w/ FPU m | LSI-11/73 u | LSI-11/73 m | SBC 11/21+ | KXT11 -CA |
|---|---|---|---|---|---|---|---|---|
| Process | | | | | | | | |
| creation | 3.1 | 5.48 | 3.56 | 5.93 | 1.84 | 2.64 | 3.96 | 2.71 |
| deletion | 2.4 | 4.19 | 2.55 | 4.35 | 1.14 | 1.92 | 2.96 | 2.06 |
| Schedule + Context Switch | 0.56 | 0.97 | 0.82 | 1.27 | 0.38 | 0.57 | 0.69 | 0.49 |
| Ring_buffer | | | | | | | | |
| 1 character | | | | | | | | |
| get | 0.55 | 0.96 | 0.55 | 0.99 | 0.27 | 0.42 | 0.67 | 0.47 |
| put | 0.50 | 0.93 | 0.51 | 0.95 | 0.25 | 0.39 | 0.63 | 0.43 |
| 2 characters | | | | | | | | |
| get | 0.61 | 1.05 | 0.61 | 1.05 | 0.29 | 0.43 | 0.72 | 0.51 |
| put | 0.58 | 1.01 | 0.58 | 1.02 | 0.28 | 0.42 | 0.70 | 0.49 |
| 4 characters | | | | | | | | |
| get | 0.73 | 1.20 | 0.73 | 1.18 | 0.36 | 0.50 | 0.88 | 0.62 |
| put | 0.71 | 1.17 | 0.71 | 1.15 | 0.35 | 0.49 | 0.86 | 0.60 |
| Signal Semaphr | | | | | | | | |
| by descriptor | 0.35 | 0.64 | 0.35 | 0.65 | 0.16 | 0.25 | 0.44 | 0.30 |
| by name | 0.61 | 0.93 | 0.61 | 0.94 | 0.27 | 0.37 | 0.75 | 0.52 |
| fast named | 0.36 | 0.65 | 0.36 | 0.66 | 0.17 | 0.29 | 0.44 | 0.30 |
| Get_status | 0.42 | 0.84 | 0.42 | 0.85 | 0.20 | 0.33 | 0.50 | 0.35 |
| Send + Receive (by value) | | | | | | | | |
| - 1 Byte | 1.17 | 2.18 | 1.18 | 2.20 | 0.59 | 0.89 | 1.45 | 1.01 |
| - 34 Bytes | 1.46 | 2.50 | 1.46 | 2.51 | 0.73 | 1.11 | 1.79 | 1.24 |
| Send + Receive (by reference) | | | | | | | | |
| - 10 Bytes | 1.26 | 2.28 | 1.26 | 2.31 | 0.67 | 0.96 | 1.54 | 1.08 |
| - 100 Bytes | 1.59 | 2.64 | 1.59 | 2.67 | 0.84 | 1.14 | 1.94 | 1.35 |
| - 500 Bytes | 3.09 | 4.32 | 3.09 | 4.35 | 1.74 | 2.11 | 3.78 | 2.63 |
| TAN | 6.84 | 7.74 | 1.64 | 1.68 | 0.35 | 0.36 | 9.00 | 6.24 |
| SIN | 5.13 | 5.82 | 1.75 | 1.78 | 0.33 | 0.34 | 6.80 | 4.69 |
| COS | 6.19 | 7.00 | 1.94 | 1.98 | 0.39 | 0.39 | 8.15 | 5.64 |
| EXP | 5.04 | 5.69 | 1.45 | 1.48 | 0.32 | 0.33 | 6.55 | 4.59 |
| LN | 5.34 | 6.01 | 1.27 | 1.31 | 0.28 | 0.29 | 6.95 | 4.86 |

Table 1: MicroPower/Pascal V1.5 Runtime System (millisec)

Notes for Table 1

- u = without MMU and m = with MMU
- FPU = with floating point unit (KEF11)
- Send/Receive without context-switch
- SBC-11/21+ using on-board memory only

| Operation | LSI-11/23 u | LSI-11/23 m | LSI-11/73 u | LSI-11/73 m | SBC 11/21+ | KXT11 -CA |
|---|---|---|---|---|---|---|
| ISR: | | | | | | |
| Interrupt dispatch-time (usec) | 62 | 91 | 42 | 54 | 81 | 61 |
| ISR execution time (usec) | 20 | 23 | 13.4 | | 21 | 16 |
| Maximal interrupt-frequency (kHz) | 7.0 | 5.1 | 12.9 | 10.9 | 4.8 | 7.5 |
| PRIO7 ISR: | | | | | | |
| Interrupt dispatch-time (usec) | 28.5 | 60 | 22 | 38 | 32 | 28 |
| ISR execution time (usec) | 20 | 24 | 13.4 | | 21 | 16 |
| Maximal interrupt-frequency (kHz) | 17.8 | 9.3 | 26 | 16 | 12.8 | 16.5 |
| CONNECT SEMAPHORE: (one process waiting on that semaphore) | | | | | | |
| Interrupt dispatch + context-switch time (msec) | 0.88 | 1.36 | 0.49 | 0.63 | 1.15 | 0.82 |
| Maximal interrupt-frequency (kHz) | 0.67 | 0.39 | 1.20 | 0.83 | 0.41 | 0.75 |

Table 2: Interrupt Performance

Note for Table 2

Additionally, the system had to service the clock interrupt at a rate of 50 Hz without the clock driver implemented, i.e. the interrupt dispatcher discarded the interrupt. The clock interrupt was enabled because realistically most systems have the clock enabled.

| Title: Using Fortran Routines In A VAXELN-Pascal Environment | Date: 16-Oct-84 |
|---|---|
| Originator: Herbert F. Maehner | Page 1 of 4 |

This Micronote discusses the VAXELN interface to VAX-11 Fortran. The following topics are covered are discussed:

1. The VAX-11 Procedure Calling Standard

2. Establishing a COMMON-data area between a VAXELN program and Fortran routines

## VAXELN Procedure Calling Standard

VAXELN Pascal (EPascal) does conform to the VAX Procedure Calling Standard. The standard allows for three methods of parameter passing : value, reference, and descriptor, and requires that values be no longer than a longword. EPascal does not explicitly support descriptors as parameters, and other languages may not treat conformant parameters as EPascal does, but EPascal does nothing to violate the calling standard.

All routines can be described according to the conventions described in the summary of run-time library entry points [1]. There are principle differences in passing parameters in Pascal and Fortran:

In Pascal, you may pass parameters as

- values, e.g PROCEDURE Pass_it (What: INTEGER);

  i.e. the value of the parameter will be copied into the procedure's stack frame with no implications for the source variable. This is termed pass by value.

or as

- variable, e.g. PROCEDURE Pass_it (VAR What: INTEGER);

  i.e. the parameter will be referenced through its address. An assignment to the parameter within the procedure will directly affect the source variable. This is termed pass by reference.

In Fortran, you pass parameters as

- values, e.g. SUBROUTINE Passit( What)
  INTEGER*4 What

  i.e. with no implications for the source. It uses a common data area to pass variables to the main program. The main difference to Pascal is, that Fortran uses pass by reference, although it is actually a value.

Calling a Fortran routine with parameter passing from a Pascal environment, you have to declare the parameters as VAR parameters in Pascal ( Figure 1 and Figure 2).

## COMMON Data Area

As mentioned before, Fortran uses a COMMON data area to pass variables from procedures to the main part of the program. In VAX-11 Pascal the [COMMON] attribute enables the linker to establish the common data section. VAXELN- Pascal has no such attribute and wouldn't overlay data sections for common areas. To overcome this restriction you must use the [EXTERNAL] attribute in VAXELN-Pascal to declare the prospective data as externally declared and use a MACRO-32 declaration to assign the Fortran common part to the "global" data area (Figure 3).

References:

1. VMS RUN TIME LIBRARY USER'S GUIDE (Summary of Run Time Library Entry Points)

2. VAXELN Encyclopedia, Procedures and Functions,

3. VMS MACRO Language Reference Manual

```
MODULE Fortran_TO_Pascal;

{ This module is a simple example on, how to use Fortran
  routines in VAXELN. }

CONST
  Max = 50;

TYPE
  Array_type = ARRAY[1..Max] OF INTEGER;

VAR
  AA: [EXTERNAL] Array_type;

PROCEDURE Valaccess (VAR What:INTEGER); EXTERNAL;

FUNCTION Double_it (VAR What: INTEGER):INTEGER; EXTERNAL;

PROGRAM FORTEST(INPUT,OUTPUT);

VAR
  What,I,J,K:  INTEGER;
  Twenty: [READONLY] INTEGER:=20;

BEGIN
  WRITELN('Program start ');
  FOR I:=1 TO Max DO AA[I] := 0; { initialize array }
  Valaccess(Twenty);        { call Fortran routine Valaccess }
  FOR I:=21 TO Max DO
  BEGIN
    What := I;
    AA[I] := Double_it(What);     { use Fortran Function to
  END;                                double array value }

   { formated output to screen  }

  K := 1;
  FOR I:=1 TO 10 DO
  BEGIN
    FOR J:=1 TO 5 DO
    BEGIN
      WRITE(AA[K]:4,'  ');
      K := K+1
    END;
    WRITELN;
  END;
END;
END; { Bend of module Fortran_to_Pascal }
```

Figure 1 : VAXELN main program module

```
C
C Fortran SUBROUTINE TO SET THE INDEXED ARRAY VALUE
C THE MAXIMAL INDEX IS PASSED AS A PARAMETER
C
      SUBROUTINE VALACCESS(WHAT)
      IMPLICIT INTEGER*4 (A-Z)
C
      COMMON  /XX$AA/ AA(50)
C
C
      DO 10 I=1,What
      AA(I) =  What-I
10    CONTINUE
C
      RETURN
C
      END
C
C
C
C INTEGER FUNCTION TO DOUBLE THE VALUE PASSED AS A PARAMETER
C
      INTEGER FUNCTION DOUBLE IT(WHAT)
      IMPLICIT INTEGER*4 (A-Z)
      DOUBLE IT = WHAT + WHAT
      RETURN
C
      END
```

Figure 2: External Fortran routines used

```
;
; definition file for common array AA to be accessed by Fortran
; subroutine VALACCESS
; the P-section name XX$AA must be the same as the one used in the
; Fortran routine
;
      .TITLE COMDAT
      .PSECT XX$AA,LONG,PIC,USR,OVR,REL,GBL,SHR,NOEXE,RD,WRT,NOVEC
AA::  .LONG 50
      .END
```

Figure 3: MACRO definition module to define the common array

| uNOTE # 015 |
|---|

| Title: Q-Bus Hardware Bootstraps | Date: 16-Oct-84 |
|---|---|
| Originator: Dave Smith | Page 1 of 4 |

The purpose of this micronote is to provide a comprehensive list of Q-Bus hardware bootstraps and the devices they support.

The tables on the next two pages are organized as follows. There is a row for each of the currently supported bootable devices. There is a column for each hardware bootstrap. The columns span both pages. In the heading for each bootstrap will be found any ordering information and/or references to notes which follow the tables. When two order numbers are given, both must be ordered since the boot code is divided into high byte and low byte ROMs.

The bootstrap devices listed are:

| BOOT DEVICE | DESCRIPTION |
|---|---|
| BDV11 | Bus Terminator, Bootstrap & Diagnostic ROM used primarily with older LSI-11 configurations |
| MXV11-A2 | Bootstrap ROM set designed for MXV11-A board |
| MXV11-B2 | Bootstrap ROM set designed for MXV11-BF & MRV11-D |
| KDF11-BA | Bootstrap ROM on board PDP-11/23+ systems |
| KDF11-BE | Bootstrap ROM on board MicroPDP-11/23 systems |
| KDF11-BF | New Bootstrap ROM for PDP-11/23+ and MicroPDP-11/23 |
| KXT11-A2 | Bootstrap ROM on board Falcon |
| KXT11-A5 | Bootstrap ROM on board Falcon-Plus |
| KDJ11-B | Bootstrap ROM on board MicroPDP-11/73 CPU |
| uVAX I | Bootstrap ROM on board MicroVAX I CPU |

BOOTSTRAP DEVICE SUPPORT

| DEVICE | BDV11 Rev A see Note 1 | MXV11-A2 | MXV11-B2 see Note 2 | KDF11-BA part no 23-339E2 23-340E2 | KDF11-BE part no. 23-157E4 23-158E4 |
|---|---|---|---|---|---|
| RX01 | X | X | X | X | X |
| RX02 | X | X | X | X | X |
| TU58 | see Note 1 | X | X | X | X |
| RL01/2 | X | X | X | X | X |
| MRV11-C | | X | | X | |
| MRV11-D | | | | | |
| RK05 | X | X | | | |
| RX50 | | | X | | X |
| RD51 | | | X | | X |
| RD52 | | | | | |
| TSV05 | | | X | | |
| TK25 | | | | | |
| RC25 | | | | | |
| DEQNA | | | | | |
| DLV11-E | X | | X | X | |
| DLV11-F | X | | X | X | |
| DUV11 | X | | X | X | |
| DPV11 | | | X | | |

BOOTSTRAP DEVICE SUPPORT

| DEVICE | KDF11-BF<br><br>part no<br>23-183E4<br>23-184E4 | KXT11-A2 | KXT11-A5 | KDJ11-B<br><br>available on CPU board only | uVAX I<br><br>available on CPU board only |
|---|---|---|---|---|---|
| RX01 | X | X | X | X | |
| RX02 | X | X | X | X | |
| TU58 | X | X | X | X | |
| RL01/2 | X | | X | X | |
| MRV11-C | | | | | |
| MRV11-D | | | | | X |
| RK05 | | | | X | |
| RX50 | X | | X | X | X |
| RD51 | X | | X | X | X |
| RD52 | X | | | X | X |
| TSV05 | X | | | | |
| TK25 | X | | | X | |
| RC25 | X | | | See note 3 | See note 3 |
| DEQNA | X | | | X | X |
| DLV11-E | | | | X | |
| DLV11-F | | | | X | |
| DUV11 | | | | X | |
| DPV11 | | | | | |

NOTES:

(1) The information in the BDV11 column refers to the Rev A chips. There were also Rev O chips and an additional TU58 chip that can be added to the board:

Rev O:

Part numbers 23-010E2, 23-011E2

Does NOT support:
DLV11-F, RX02 as bootable devices

TU58 ROM:

Part number 23-126F3

Inserted into socket XE40. Other ROM must be Rev A. Allows use of the TU58 DECtape II as a bootable device.

(2) The MXV11-B2 Bootstrap ROMs can be used with the MXV11-BF multifunction module as well as the MRV11-D ROM module. It will not work with the MXV11-A module.

(3) The RC25 adapter board must be configured at the the DEC standard base address for the first MSCP controller (772150). Other MSCP controllers may also reside but may not be booted.

| uNOTE # 016 |
|---|

| Title: KXT11-CA Software Development Tools | Date: 16-OCT-84 |
|---|---|
| Originator: Scott Tincher | Page 1 of 8 |

The KXT11-CA is a single board computer (SBC) which executes the PDP-11 instruction set. It may be utilized as a stand-alone SBC or interfaced to the Q-bus as a peripheral processor or as an intelligent I/O processor (IOP). This article will describe the software tools available to develop applications in either the stand-alone mode or the IOP mode.

The KXT11-CA features:

- T11 Microprocessor which implements the PDP-11 instruction set
- 32K bytes of on-board static RAM
- Two 28-pin sockets for up to 16K bytes of additional RAM or 32K bytes of ROM
- Three serial line units:
  - One asynchronous DL compatible line (RS232)
  - One synch/asynch line with modem control (RS449)
  - One synch/asynch with data and timing only (RS449)
- 20 programmable parallel I/O lines
- Three 16-bit programmable interval timers
- 2-channel DMA controller
- Q-bus interface
- Four diagnostic LEDs

The Q-bus interface of the KXT11-CA allows up to 14 KXT11-CAs to be added to a traditional Q-bus system. As a slave device the KXT11-CA offloads the arbiter CPU's processing activities by providing real-time I/O data buffering, preprocessing, and high speed communications.

The KXT11-CA is especially suited for applications with critical interrupt latency requirements or applications that must service a high frequency of interrupts. The KXT11-CA may also be used as a computational engine in applications where it is possible to partition the application to run in parallel.

The software development environment for systems which utilize KXT11-CAs is slightly different from that of the traditional Q-bus system. The system programmer must develop application programs for each KXT11-CA in the system in addition to the application code which runs on the arbiter CPU. Different software application tools are available for the arbiter and "onboard" environments. (When used in stand-alone mode only the onboard environment need be considered.)

## THE ARBITER ENVIRONMENT

The arbiter system may run under any of the following operating systems:

- MicroPower/Pascal
- RT-11
- RSX-11M
- RSX-11M-PLUS

Each of these operating systems offers a device handler for the two-port RAM of the KXT11-CA as well as a utility for loading application programs across the Q-bus into the KXT11-CA. A MicroPower/Pascal application may be coded in Pascal and MACRO-11. RT-11 and RSX-11 applications will be coded in MACRO-11 or a high level language, such as FORTRAN, which is capable of issuing programmed requests (RT-11) or QIO directives (RSX-11).

## USING A MICROPOWER/PASCAL ARBITER SYSTEM

If the arbiter system controlling the application is running in a MicroPower/Pascal environment there are KXT11-CA specific functions available to aid in program development. The first component is the KX device handler. This handler provides the arbiter-side interface to the two-port RAM of the KXT11-CA. The KX handler supports up to 14 KXT11-CAs on the Q-bus. Two functions are supplied which simplify the interface between the application program and the KX handler. These functions are:

- KX_write_data: Transfer data from an arbiter buffer to a KXT11-CA process and return a completion-status value.

- KX_read_data: Transfer data from a KXT11-CA process to an arbiter buffer and return a completion-status value.

MicroPower/Pascal also provides a function which transfers a MicroPower/Pascal .MIM file from the arbiter to a KXT11-CA. This function, KXT_LOAD, reads a .MIM file from the arbiter and initiates a DMA transfer using the DTC of the KXT11-CA to transfer the file to the KXT11-CA's local memory. This procedure may be called at any time by the arbiter's application program - not necessarily at system startup time.

MicroPower/Pascal also supplies the symbolic debugger PASDBG. PASDBG supplies the following features:

- A set of debugger commands and qualifiers that allow for control of an executing program.
- Access to the symbol table generated by the Pascal compiler, providing symbolic (Pascal language) referencing and variable access.
- Access to process control variables and structures.
- Control of an application system not configured for terminal I/O.
- A method for user control after an execution error.
- A method for loading a program on the application system while PASDBG is running on a host computer.

## USING AN RT-11 OR RSX-11 ARBITER SYSTEM

If the arbiter system controlling the application is running in a RT-11 or RSX-11 environment there are tool kits available to aid in program development. They are the KXT11-C/RT-11 Peripheral Processor Tool Kit (QJV51) and the KXT11-C/RSX-11 Peripheral Processor Tool Kit (QJV52).

There are two major components in each of these tool kits. They are the KX device handler and the KUI utility program.

The KX device handler manipulates the two-port RAM of the KXT11-CA so that it appears to be a standard Q-bus I/O device. The KUI utility program allows programs to be loaded into a peripheral processor from the arbiter, performs debugging operations, starts execution of KXT11-C programs, and initiates KXT11-CA selftests.

The KX handler supplied with the RT-11 tool kit supports up to four KXT11-CAs where each KXT11-CA appears as two logical units. More than four KXT11-CAs may be supported by editing, renaming, and rebuilding the KX handler.

The following RT-11 programmed requests are supported by the KX handler:

- .OPEN - associates a user-specified channel number with a logical unit number of the KXT11-CA.
- .CLOSE - frees a previously opened channel for use with another device or file.
- .READ - transfers data from a peripheral processor to an arbiter buffer. (.READ, .READW, .READC)
- .WRITE - transfers data from an arbiter buffer to a peripheral processor. (.WRITE, .WRITEW, .WRITEC).

The KX handler supplied with the RSX-11 tool kit supports up to 14 KXT11-CAs. The KX handler assigns a unit number for each data channel in each KXT11-CA two-port RAM. This handler supplies the following RSX-11 I/O requests:

- IO.RVB - Read a virtual block of data from the device unit unit specified in the macro call.
- IO.WVB - Write a virtual block of data to a physical device unit.
- IO.ATT - Attach a physical device to the control of the task which issued the request.
- IO.DET - Detach a physical device from the control of the task which issued the request.

Included in the RT-11 and RSX-11 tool kits is the KUI (KXT11-CA User Interface) utility program. The KUI program has several commands which supply the following functions:

- @ - Process commands from the specified indirect command file.
- CLOSE - Close the file specified in the LOG command.

- EXECUTE - Start a program on the specified KXT11-CA.

- EXIT - Exit the KUI utility and return to the operating system.

- LOAD - Load a program from the arbiter's mass storage to arbiter memory. Then perform a DMA operation to transfer the image to the specified peripheral processor's memory. KUI under RT-11 supports the transfer of .SAV, .LDA, and .MIM files. KUI under RSX-11 supports the transfer of .TSK and .MIM files.

- LOG - Record all commands, status information, and messages during this terminal session in the specified file. The CLOSE command terminates the logging session.

- ODT - Executes the octal debugging tool (ODT). This tool allows the arbiter system to examine and modify the contents of registers and memory local to a KXT11-CA. ODT may also be used to start or halt a program.

- REINIT - Reinitialize the specified peripheral processor and reboot it's application.

- RESUME - Causes a SUSPENDed command file to continue execution.

- SELFTEST - Causes one or more of several diagnostic programs to execute.

- SET - Specifies a peripheral processor as the target for subsequent commands.

- SHOW - Displays information about the state of the peripheral processor.

- SUSPEND - Used in an indirect command file to halt execution of the file. The RESUME command can return control to the command file.

- TRAP - performs a trap emulation so that a trap handling routine can be tested.

THE ONBOARD PROGRAMMING ENVIRONMENT

The KXT11-CA may be programmed in either MACRO-11 or MicroPower/Pascal. MicroPower/Pascal provides the ability to program the onboard devices in a high-level language, Pascal. In particular MicroPower/Pascal provides the following device handlers:

- DD: This handler supports the TU58 tape drive. It allows the TU58 to be interfaced to any of the asynchronous I/O channels.
- KK: This handler manipulates the two-port RAM from the KXT11-CA side in the KX/KK protocol. This protocol allows the KK handler to pass variable length messages to the arbiter system by emulating a traditional Q-bus slave device. Two functions are supplied which simplify the interface between the user's application code and the KK handler. These functions are:
  - KK_read_data: transfer data from the arbiter to a KXT11-CA buffer and return a completion-status value.
  - KK_write_data: transfer data from a KXT11-CA buffer to the arbiter and return a completion-status value.
- QD: This handler supports the two-channel DMA transfer controller (DTC). The QD handler enables the DTC to move data from source to destination without the aid of the CPU. One location, source or destination, must be local to the KXT11-CA. The QD handler may be used for the following functions:
  - Transfer data to and from Q-bus memory.
  - Transfer data to and from local memory.
  - Search for data.
  - Transfer to and from local I/O devices.
  - Access the Q-bus I/O page.
  - Assure access to a DMA Channel.
- XL: Supports asynchronous communications on the three serial ports of the KXT11-CA. The first port is a standard DL device. The second port is channel A of the multiprotocol chip. This

channel is supported with full modem controls. The third port is channel B of the multiprotocol chip. This channel is supported as though it were a standard DL device. All three channels may be operated simultaneously.

- XS: Supports synchronous operation of channel A of the multiprotocol chip. The handler provides the following bit-oriented communication procedures:
  - Synchronization (Flag detection)
  - Transparency (Bit stuffing)
  - Invalid frame detection
  - Frame abortion detection
  - Frame check sequence (FCS) checking/calculation

  The handler can be used by user-written software as a component in performing bit-oriented protocols such as X.25, HDLC, SDLC, and others.

- YK: Supports the parallel I/O port and the three counter-timers. The handler provides the functions of read, write, pattern recognition, DMA read, DMA write, counter-timer set, counter-timer read, and counter-timer clear. Typical parallel port operations are:
  - Transferring a series of bytes or words through a port with handshake protocol.
  - Setting or reading the bits of external state lines.
  - Generating a time base to software.
  - Generating a waveform for external output.
  - Counting pulses from an external input.

These MicroPower/Pascal device handlers do not support all of the functions of the onboard devices of the KXT11-CA. For this reason, or because of preference, the application code for the KXT11-CA may also be written in MACRO-11.

RELATED DOCUMENTS

For further information pertaining to the KXT11-CA and it's software development tools please reference the following materials:

| | |
|---|---|
| KXT11-CA Single-Board Computer User's Guide | EK-KXTCA-UG |
| KXT11-C Peripheral Processor Software User's Guide | AA-Y615A-TK |

| uNOTE # 017 |
|---|

| Title: LSI 11/23 ECO History | Date: 19-NOV-84 |
|---|---|
| Originator: Bob Hessinger | Page 1 of 8 |

This micronote documents the ECO and etch revision history of the KDF11-A (LSI 11/23) module. A quick verify has been included so that the status of a module may be determined by a visual check.

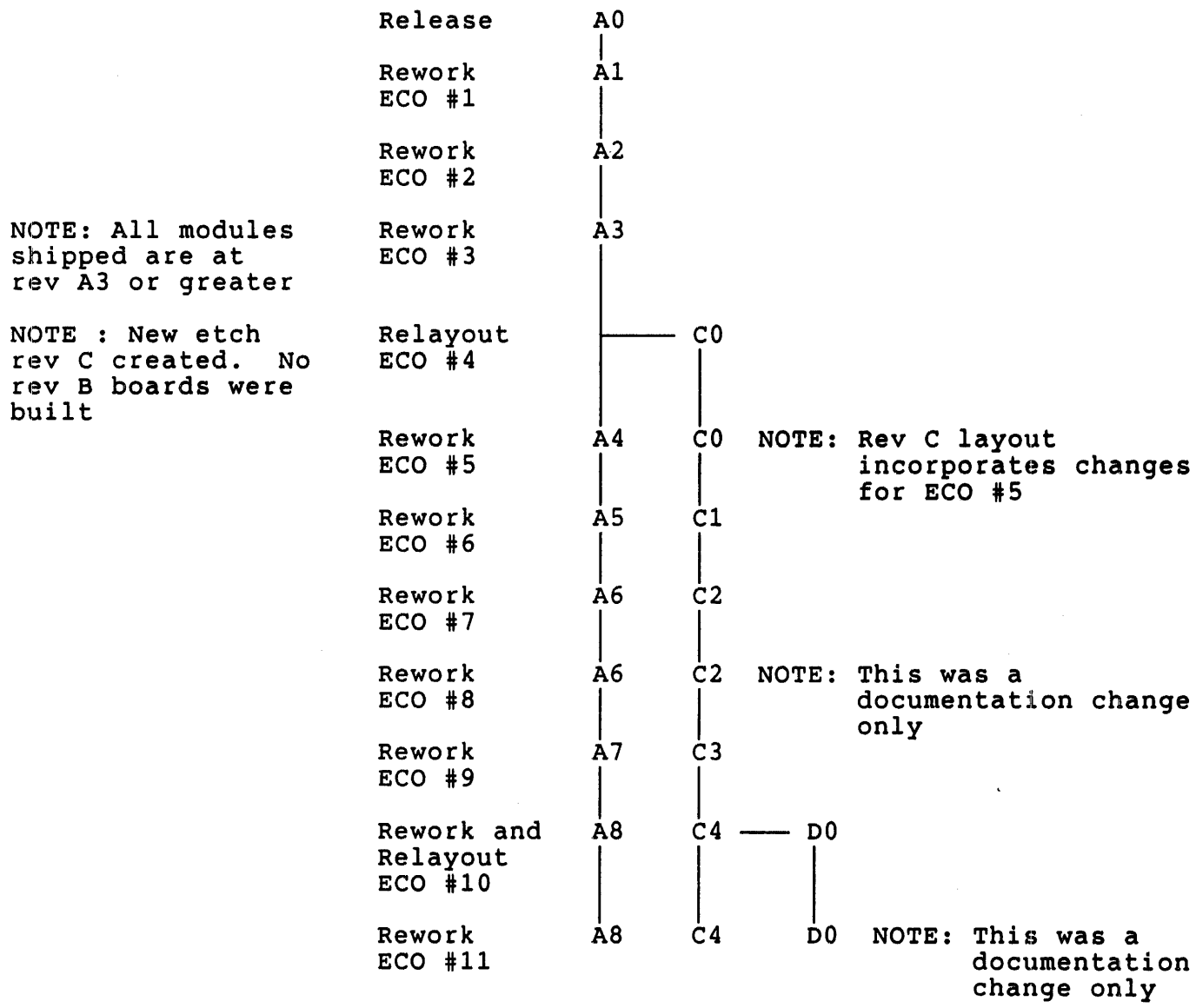
For the M8186, the revision identifier is a two field alphanumeric designation stamped on the reverse side of the module handle. The first field indicates the etch revision. The second field indicates the modifications to this etch.

Hardware revision notation :

```
                    A 0
Identifies etch level    Identifies modifications
```

The M8186 began as hardware revision "A0", as shown above. That is, etch revision "A" with no modifications or rework. As ECO's were released calling for rework, the hardware revision level was updated to "A1", then "A2", etc. Periodically new etch revisions were released, incorporating previous modifications. When these occurred the etch revision field was updated from "A" to "C", and later from "C" to "D". For the M8186, no etch revision "B" was released.

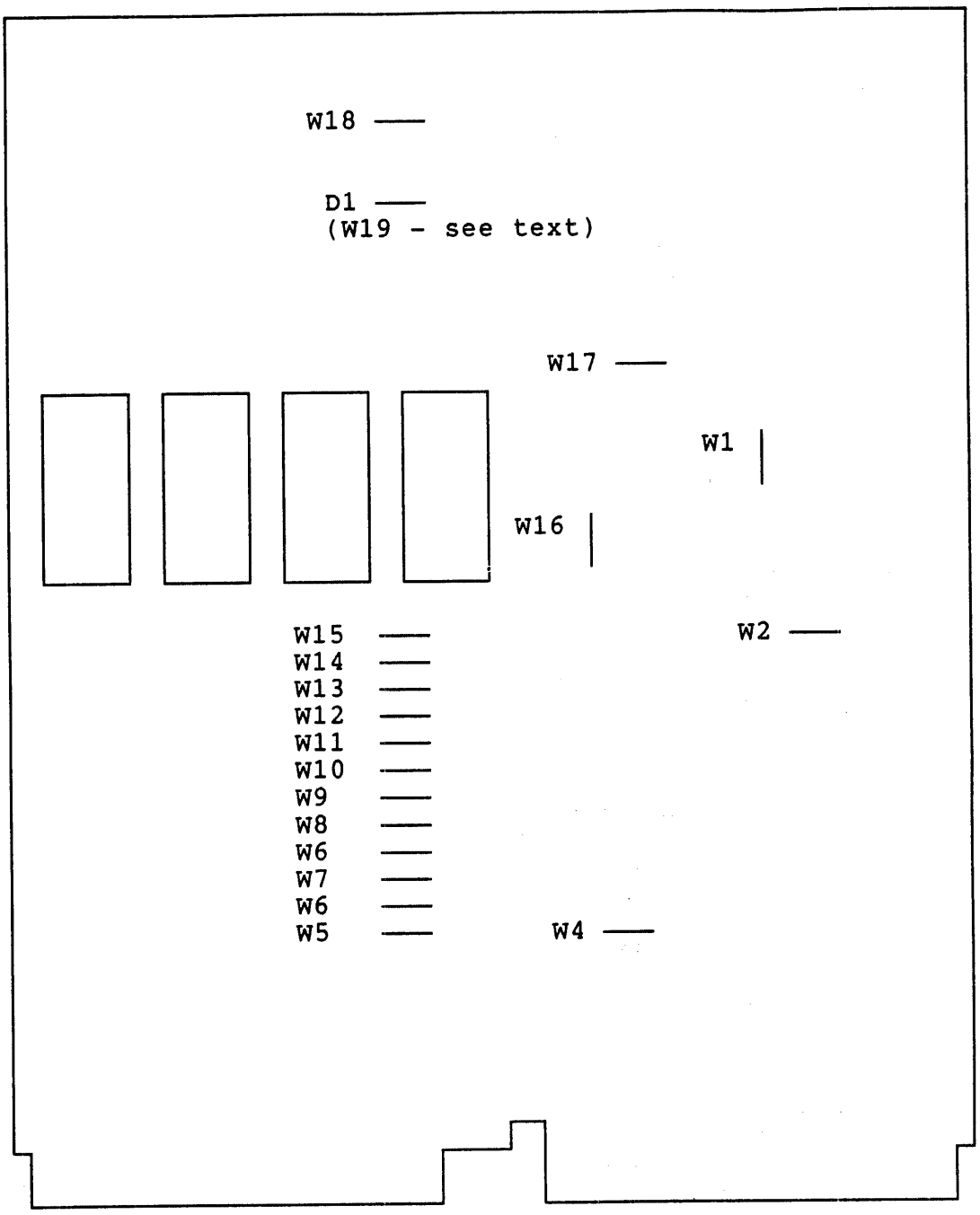
The hardware revision history of the M8186 is shown below:



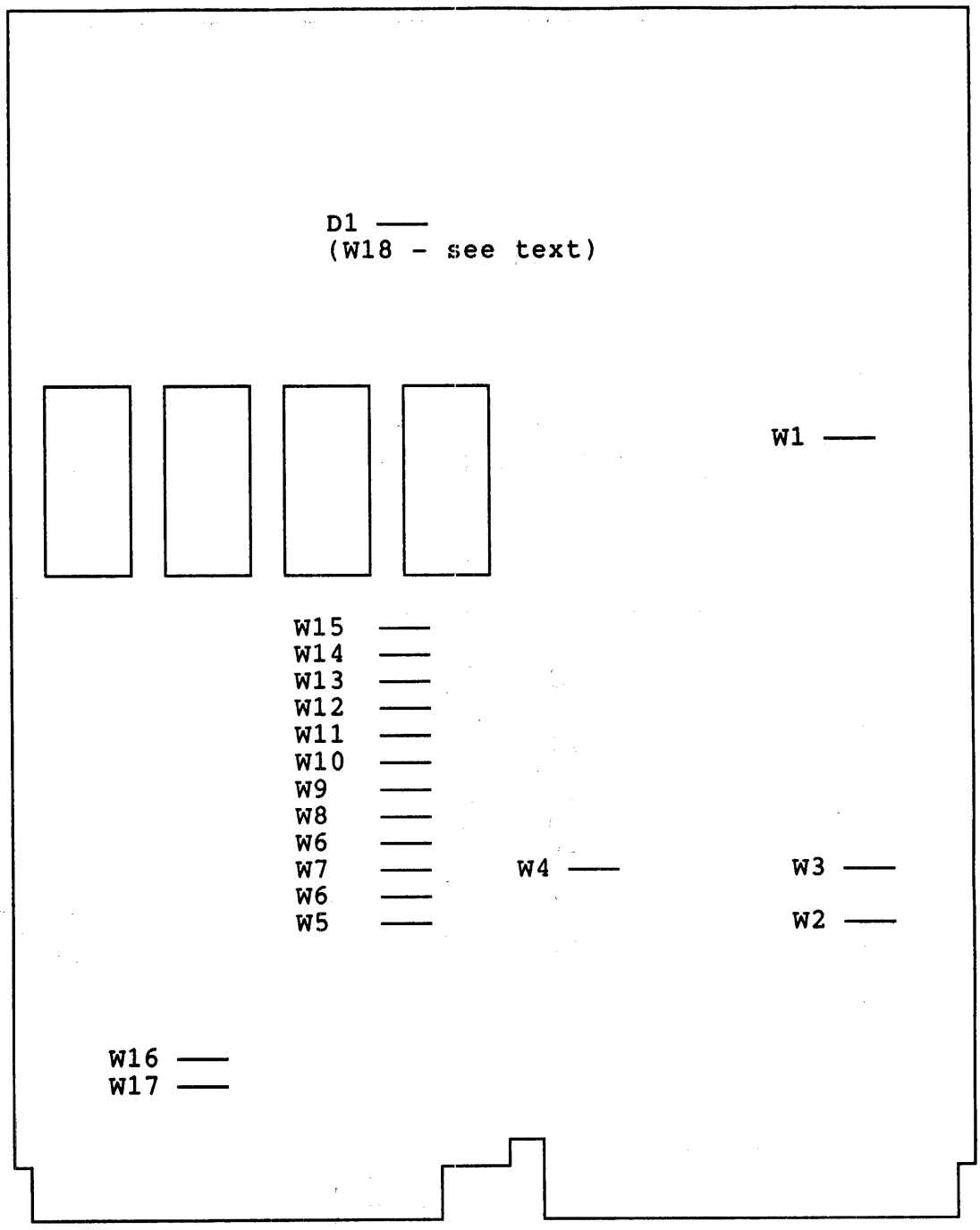
Jumper Functions on Etch Revision "A", "C" and "D" Modules

| Jumper | Function | Description | Shipped |
|---|---|---|---|
| W1 | Master Clock | In = Enabled, do not remove | In |
| W2,W3 | DEC Reserved | Factory configured, do not change (see note 1) | W2=Out<br>W3=In |
| W4 | BEVENT | Out = Enabled | In |
| W5,W6 | Power-up Mode | Mode W5 W6<br>0 - PC=24,PS=26 Out Out<br>1 - Console ODT In Out<br>2 - Bootstrap Out In<br>3 - Reserved In In | Mode 1<br>W5=In<br>W6=Out |
| W7 | Halt/Trap Option | In = Trap to 10 on Halt<br>Out = Enter Console ODT on Halt | Out |
| W8 | Bootstrap Address | In = Boot to 17773000<br>Out = Bootstrap address specified by jumpers W9-W15 | In |
| W9-W15 | User Bootstrap Address | W9-W15 correspond to address bits 9-15 respectively.<br>In = logic 1, Out = logic 0 | In |
| W16,W17 | DEC Reserved | Factory configured, do not change | In |
| | Etch A only | | |
| W18 | DEC Reserved | Factory configured, do not change | In |
| W19 | Wake-up Circuit | Out = enabled<br>This jumper is a red wire across diode D1 | In |
| | Etch C and D only | | |
| W18 | Wake-up Circuit | Out = enabled<br>This jumper is a red wire across diode D1 | In |

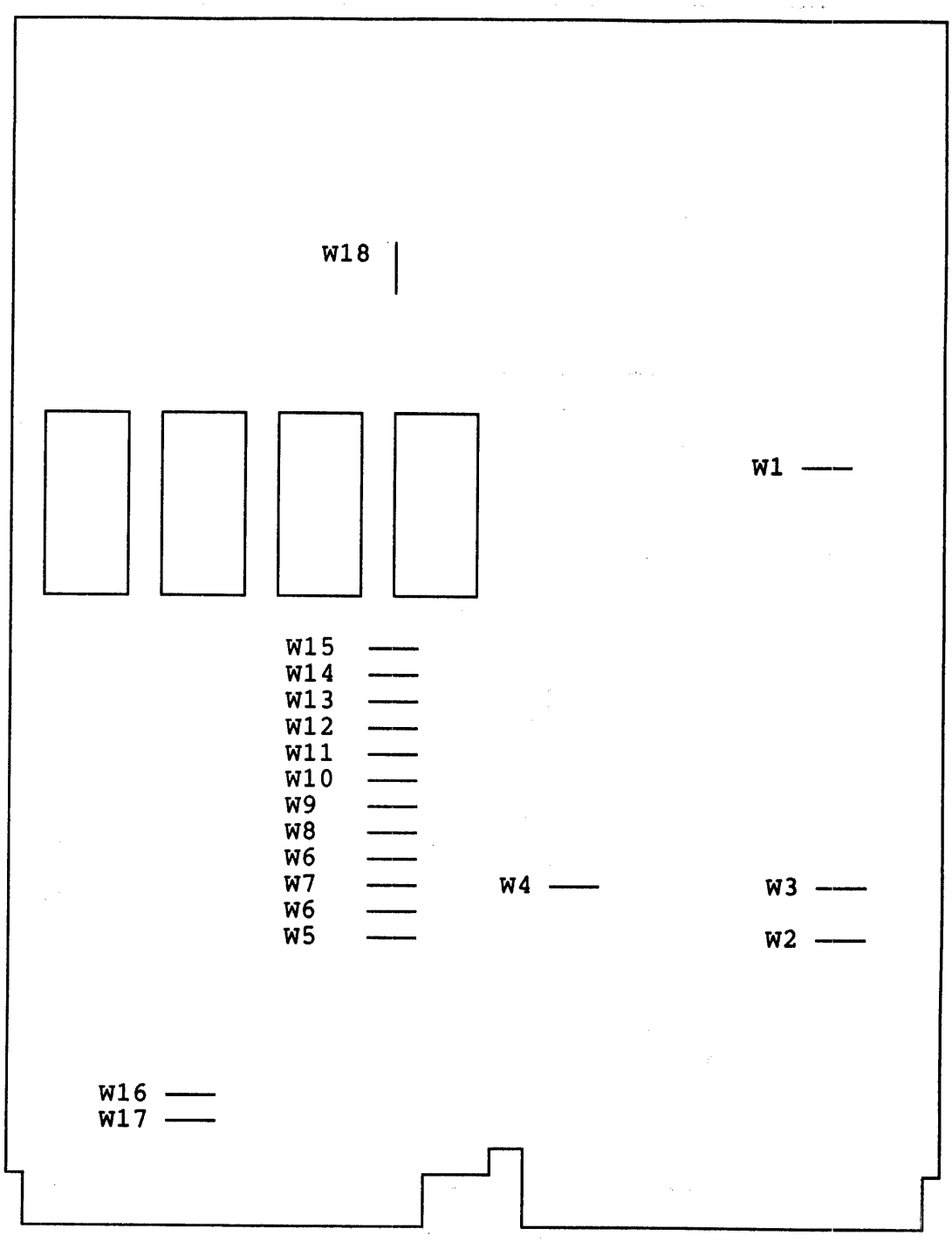
note 1 : W3 on etch A modules consists of a jumper from E2 pin 5 to E2 pin 15.




KDF11-A REV A Jumper Layout




KDF11-A REV C Jumper Layout




KDF11-A REV D Jumper Layout

The following table details the ECO's issued since the M8186 began to ship to the field. These ECO's are coded "M8186-ML0XX", where "XX" is the ECO number shown below :

| ECO # | Problem | Quick Verify |
|---|---|---|
| 04 | Too many wires and etch cuts, new etch needed. Note that the jumper locations change for etch Revision C. Note also that etch A boards bring only 18 bits of addressing from the MMU to the Q-BUS, while etch C boards bring all 22 bits of addressing from the MMU to the Q-BUS. | Module handle will be stamped "Cn" where n indicates modifications. |
| 05 | I/O page addressing scheme differs from LSI-11/2 processor. | Check for etch cut to E7 pins 16 and 18 (rev A boards only) |
| 06 | The internal wake-up circuit defeats the power sequencing provided by standard DEC power supplies. | Red jumper wire is installed in parallel with D1. |
| 07 | CTL/DAT hybrid (57-00000-00) and MMU IC (21-15542-00) were not compatible with KEF11-AA floating point option. The FP registers in the MMU were inaccessible, and the CTL/DAT data path caused intermittent errors in floating point instructions. | CTL/DAT should be 57-00000-01 or higher and MMU should be 21-15542-01 or higher for floating point compatibility. Coordinate with ECO M8186-ML009 |
| 08 | MMU (21-15542-01) was included as part of the M8186 module. This documentation change removes the MMU and makes it an option which is ordered separately. | Some modules may or not have MMUs, depending on the options ordered. |

| ECO # | Problem | Quick Verify |
|---|---|---|
| 09 | 1) No jumper table in print set (doc only)<br>2) Crystal oscillator may short to adjacent components<br>3) Possibility of worst case MMU timing violations. Change configuration of W2 and W3 to adjust timing. This ECO must be installed :<br>A) When ECO m8186-ML007 is installed<br>B) When the KEF11 or FPF11 is installed<br>C) When one of the F-11 chips is replaced<br>D) Whenever unexplained system crashes occur | 1) Table added<br>2) Manufacturing includes nylon spacer<br>3) Module will have W2 removed and W3 in. On rev A modules W3 is a wire from E2 pin 5 to E2 pin 15. |
| 10 | 1) Heavily loaded systems lock up during worst case timing between DMA and interrupt arbitration. Symptoms usually occur with DMA options not manufactured by DEC.<br>2) Too many wires and etch cuts, new etch needed. Note W18 now uses jumper posts. | 1) Rev A and Rev C modules will have wires on E2 pins 2 and 4. Rework included in Rev D.<br>2) Module handle will be stamped "Dn" where n indicates modifications. |
| 11 | Documentation updated. | Documentation only. |

| uNOTE # 018 |
|---|

| Title: Programming the KXT11-CA DMA controller | Date: 28-DEC-84 |
|---|---|
| Originator: Scott Tincher | Page 1 of 24 |

The KXT11-CA intelligent I/O processor contains several user programmable devices. One of these devices is a DMA transfer controller (DTC). This article will describe the features of the DTC and provide some programming examples. This article is intended for use by individuals interested in programming the DTC using MACRO-11. DIGITAL supplies a DTC device driver for those programmers using MicroPower/Pascal. A working knowledge of MACRO-11 and of the KXT11-CA is assumed.

## FEATURES/CAPABILITIES

The DTC is addressable by the local T-11 microprocessor as an I/O device. It is capable of performing DMA transfers between any of the following addresses:

1) A 16-bit local address to a 16-bit local address

2) A 16-bit local address to a 22-bit global address

3) A 22-bit global address to a 16-bit local address

4) A 22-bit global address to a 22-bit global address

5) To/From channel A of the multiprotocol SLU

6) To/From the PIO chip

Word, high byte, and low byte transfers are supported locally. Only word transfers are supported across the Q-bus.

The operations of the DTC are controlled by several internal registers. It was designed with the capabilty of loading these registers directly from memory thereby minimizing the amount of processor intervention necessary to perform a DMA transaction. The area of memory where the parameters for the DTC are stored is referred to as the chain table. The local microprocessor need only load the address of the chain table into the DTC and give a "start" command to initiate a DMA transfer.

DMA transactions may be initiated locally by the T11 or by the arbiter CPU. If the transfer is initiated by the arbiter the command words and transfer parameters are placed in the command registers of the two-port RAM file. The local CPU will then initiate the DMA transaction using the parameters supplied by the arbiter.

The DTC consists of two identical channels. DMA transfers may be interleaved between these two channels or interleaved between the DTC and the T-11. It is also possible to select a "hog mode" that allows the DMA transfer to run to completion without interruption.

The DTC supports three types of operations: Transfer, Search, and Transfer-and-Search. As the name implies, Transfer operations move data from a source to a destination. Search operations read data from a source and compare the data to the pattern register. A mask register allows the user to declare "don't care" bits. The Transfer-and-Search operation combines the features of the Transfer and Search functions. In this type of operation data is transferred between a source and destination until the data transferred meets the match condition specified in the Channel Mode register.

The DTC is capable of performing multiple DMA transactions without processor intervention. This can be accomplished in two ways: base-to-current reloading or chaining. Base-to-current reloading allows the DTC to reload a portion of its registers before initiating a DMA transfer. The reload operation occurs between internal registers so there are no memory access related delays. This type of operation is only practical in applications where data is continuously transferred between the same addresses. Chaining allows all of the applicable registers of the DTC to be reloaded from a new chain table. Therefore this is a slower but more flexible alternative.

Upon completion of a DMA transfer the DTC may perform any combination of the following options: Interrupt the local processor, perform base-to-current reloading, or perform a chain reload. It may also choose to take no action.

## DTC REGISTERS

Among the internal registers of the DTC are two chip-level registers, the Master Mode register and the Command register. These registers control both channels of the DTC. In addition, each channel of the DTC is controlled by several channel-level registers. For the sake of completeness a brief description of these registers will be included here. For a detailed description refer to the KXT11-CA Single Board Computer User's Guide (EK-KXTCA-UG-001).

## CHIP-LEVEL REGISTERS

### Master Mode Register

The Master Mode register controls the chip-level interfaces. It is used to:

- Enable/disable the DTC
- Select DTC/CPU interleaving
- Enable/disable asynch operation
- Enable/disable counter/timer interrupt request
- Enable/disable interrupt save vector

### Command Register

The command register is used to issue commands to the DTC channels such as: Reset, Start Chain, etc.

## CHANNEL-LEVEL REGISTERS

(Each of the following registers is present in each channel of the DTC)

### Current Address Registers A and B (CARA, CARB)

CARA and CARB consist of two words, the segment/tag and the offset. The segment/tag is used to indicate:

- Address bits <21:16> of the source (or destination)
- If the source (or destination) resides on the Q-bus
- Whether the source (or destination) address should be incremented, decremented, or held constant
- Whether wait states should be included

The offset is used to indicate:

- Address bits <15:00>

### Base Address Registers A and B (BARA, BARB)

BARA and BARB are identical to CARA and CARB. They are used to reload CARA and CARB if base-to-current reloading is selected after a DMA operation has terminated.

Current Operation Count Register (COPC)

This 16-bit register is used to specify the number of words (or bytes) to be transferred during a DMA operation. The maximum word count is obtained by programming this register with a zero.

Base Operation Count Register (BOPC)

This register is identical to the COPC register. It is used to reload the COPC register when base-to-current reloading is selected.

Pattern and Mask Registers

The Pattern and Mask registers are used during Search and Transfer-and-Search operations. The contents of the Pattern register are compared to the read data to generate a "match" condition. The Mask register is used to generate "don't care" bits. Setting a bit to '1' in the Mask register specifies that the bit always matches.

Status Register

The status register is a 16-bit read-only register which returns the status of the following fields: Interrupts status, DTC status, Hardware interface status, and Completion status.

Interrupt Vector and Interrupt Save Registers

The Interrupt Vector register contains the vector that is output during an interrupt acknowledge cycle. When an interrupt occurs the contents of the Interrupt Vector register and a part of the Status register are stored in the Interrupt Save register. This allows a new vector to be loaded during chaining so that a new DMA operation can be performed before an interrupt acknowledge cycle occurs.

Channel Mode Register

The Channel Mode register consists of two words, channel mode high and channel mode low. Channel mode low is used to indicate:

- The operation type (transfer, search, transfer-and-search,bytes,words)
- Whether CARA (or CARB) defines the source (or destination)
- Transfer type (single, hog mode, interleaved)

- Completion options (interrupt CPU, base-to-current reload, chain reload)

Channel mode high is used to:

- indicate match conditions
- mask the hardware requests for DMA operations
- cause the channel to request the bus for a DMA operation

Chain Address Register

The chain address register consists of two words, the segment/tag and the offset. This register is used to point to the reload word, the first word in a chain table. The segment/tag is used to indicate:

- Whether the reload word resides in Q-bus memory
- Whether the reload word resides in the Q-bus I/O page
- Address bits <21:16>

The offset is used to indicate:

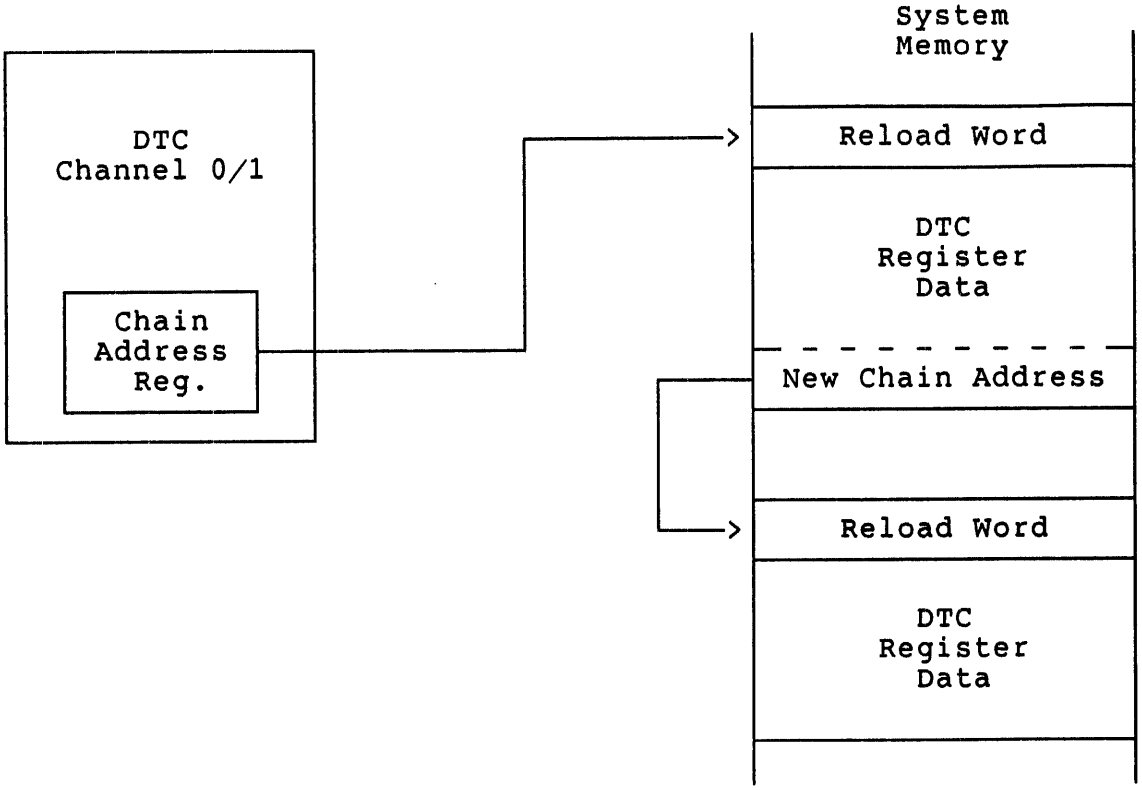
- Address bits <15:00>

## PROGRAMMING THE DTC

Programming the DTC consists of three phases: Chip Initialization, Data Transfer (or Search), and Termination. This section will provide a general description of these phases.

## CHIP INITIALIZATION

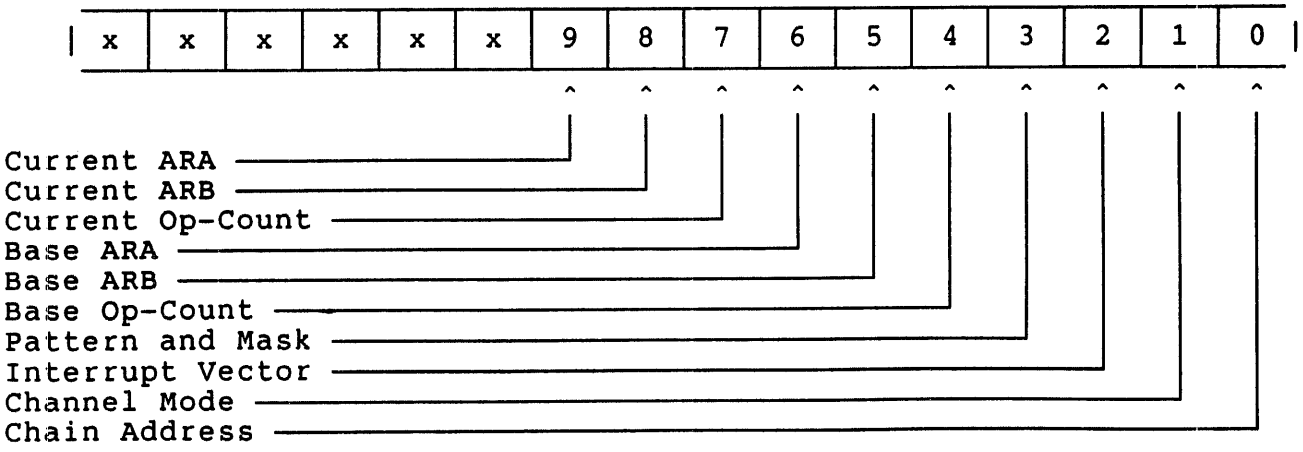
The Reset instruction is used to place the DTC in a known state. A reset will clear the CIE, IP, SIP and WFB bits and set the CA and NAC bits in the Channel Status registers. The Master Mode register will also be cleared. Before a DMA operation is initiated the local CPU loads the Master Mode register and the Chain Address register of the appropriate channel of the DTC. The DTC fetchs any other parameters that are necessary from a table located in system memory referred to as the chain table. This minimizes the amount of CPU intervention necessary to perform a DMA operation. The relationship of the Chain Address register to the chain table is shown in Figure 1.




- Figure 1 -

The first word in the chain table is the reload word. The reload word is used to specify which registers are to be loaded for the pending DMA operation. Bits <9:0> of the reload word correspond to the registers of the DTC as shown in figure 2. Bits <15:10> are not used.

Reload Word




- Figure 2 -

Therefore if a bit in the reload word is set then the corresponding register is to be reloaded from the chain table. Since all of the registers are not applicable to each DMA operation the chain table may be of variable length. ( i.e. The pattern and mask registers would not be used in DMA operations that do not search the data.) It is NOT correct to select a register in the reload word and subsequently load that register with a dummy argument such as zero. The following are examples of the relationship between the reload word and the chain table.

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Current ARA Segment/Tag | | | | | | | | | |
| Current ARA Offset | | | | | | | | | |
| Current ARB Segment/Tag | | | | | | | | | |
| Current ARB Offset | | | | | | | | | |
| Current Op-Count | | | | | | | | | |
| Channel Mode High | | | | | | | | | |
| Channel Mode Low | | | | | | | | | |

| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| Current ARA Segment/Tag | | | | | | | | | |
| Current ARA Offset | | | | | | | | | |
| Current Op-Count | | | | | | | | | |
| Pattern Register | | | | | | | | | |
| Mask Register | | | | | | | | | |
| Channel Mode High | | | | | | | | | |
| Channel Mode Low | | | | | | | | | |
| Chain Address Segment/Tag | | | | | | | | | |
| Chain Address Offset | | | | | | | | | |

The DTC has been properly initialized once the chain table(s) have been created and the Master Mode register and Chain Address Register for the selected channel have been loaded.

DATA TRANSFER

The DTC may perform a DMA operation once it has been properly initialized. A DMA operation may be initiated in one of four ways: by software request, by hardware request, by loading a set software request bit in the Channel Mode register during chaining, or as the result of a command from the arbiter.

Software Request: The local CPU may initiate a DMA operation by writing a 'software request' command followed by a 'start chain' command to the Command register. The 'software request' command sets the software request bit in the channel's Mode register. If either the SIP (second interrupt pending) bit or the NAC (no auto-reload or chain) bit is set in the channel's status register the DMA operation will not begin. The SIP bit will be cleared when the channel receives an interrupt acknowledge. The NAC bit will be cleared when the channel receives a 'start chain' command. The 'start chain' command initiates the DMA operation after the registers of the selected channel are loaded from the chain table. The 'start chain' command is ignored if the SIP bit or the CA (Chain Abort) bit are set in the channel's status register. The SIP bit was described above. The CA bit is cleared when the channel's chain address register is reloaded.

Hardware Request: DMA operations may be started by applying a 'low' on the channel's DREQ input. No details about this type of request will be provided since they fall beyond the scope of this note.

Starting After Chaining: If the software request bit of the channel's Mode register is loaded during chaining the channel will perform the DMA operation at the end of chaining.

Arbiter Request: The arbiter may interrupt the local CPU to request a DMA operation. This is accomplished by passing parameters to load the chain address register of channel 0 via the two-port RAM. The arbiter loads register 2 of the TPR with the offset of the chain address register and register 3 of the TPR with the segment/tag of the chain address register. The DMA operation is then initiated by setting the DMA Load bit (bit 1) in the TPR command register (register 0). Error conditions will be returned in TPR register 1.

Information in the channel's Mode register determines what type of DMA operation will be performed. The Channel Mode register consists of two words, Channel Mode High and Channel Mode Low.

Bits <3:0> of the Channel Mode Low register select the type of DMA operation. These bits determine whether the data should be transferred,

searched, or transferred-and-searched. Bit 4 is the flip bit. It is used to determine which set of current address registers (CARA, CARB) points to the source.

Bits <6:5> determine the transfer type. The types of DTC transfers are: single transfer, demand dedicated with bus hold, demand dedicated with bus release, and channel-to-channel demand interleave. Single transfer is used with devices which transfer data at irregular intervals. A single DMA transaction will occur each time a 'software request' command is issued or the DREQ input is asserted. Demand dedicated with bus hold is a software hog mode. This mode allows the DMA transaction to run to completion as long as there is a valid op count and the DREQ input is asserted. If the DREQ input is not asserted no DMA operations will occur but the channel will retain bus control. Demand dedicated with bus release is similar to demand dedicated with bus hold in that a DMA transaction is allowed to run to completion if DREQ is asserted. If DREQ is not asserted the DTC must release the bus thus allowing other devices to obtain the bus. The operation performed by a channel-to-channel demand interleave request depends on the state of bit 2 in the Master Mode register. If MM bit 2 is clear then control may be passed between each channel of the DTC without the need to release the bus. If MM bit 2 is set then the DTC must share the bus with the local processor. The DTC will release the bus and then re-request it after every DMA iteration.

Bits <1:0> of the Channel Mode High register are used to determine the type of match control in Search and Transfer-and-Search operations. The DTC is capable of generating a termination condition based on 'No Match', 'Word Match', and 'Byte Match'.

Bit <4> of the Channel Mode High register causes the channel to request the bus and perform transfers when it is set by a 'Software Request Command' or a chain reload.

TERMINATION OPTIONS

Bits <15:7> of the Channel Mode Low register control the termination options. A DTC operation may be terminated in a number of ways. If the Current Operation Count Register goes to zero then a Terminal Count (TC) termination is generated. External logic may assert the End Of Process (EOP) input of the DTC to generate an EOP termination at any time. In addition, during a Search or Transfer-and-Search operation a match condition may occur which generates a MC termination. Bits <15:7> allow the DTC to perform a chain reload, a base-to-current reload, or to interrupt the local processor if a TC, EOP, or MC termination condition is encountered. If bits <15:7> are cleared then no special action is initiated when a TC, EOP, or MC condition is encountered.

EXAMPLES

The following example programs were developed on a PDP-11/23+ system with 256KB of memory using the RT-11 (version 5.1) operating system with the KXT11-C Peripheral Processor Software Toolkit. These examples assume the programmer is familiar with MACRO-11 and the KXT11-C Peripheral Processor Toolkit.

```
        .TITLE EXAM1.MAC

; This program transfers data from local KXT11-C addresses to other
; local KXT11-C addresses.  This program should be compiled and linked

; on the development system and then downloaded into the KXT11-C using

; the KXT11-C Software Toolkit.  Once the program has been compiled and

; linked use the following KUI commands to execute it and verify its
; successfullness.
;
;       .KUI
;       KUI>SET n       ! Where n is the appropriate KXT11-C
;       KUI>LOAD EXAM1
;       KUI>ODT         ! Use KUI ODT to verify that the destination
;            .            addresses are cleared
;            .
;       ODT>^C
;       KUI>EXECUTE     ! Execute EXAM1
;       KUI>ODT         ! Use KUI ODT to verify that the transfer was
;            .            successful
;            .
;       ODT>^C
;       KUI>EXIT
;

;       SET UP REGISTER ASSIGNMENTS

        MMREG   =       174470  ; MASTER MODE REGISTER
        CMDREG  =       174454  ; COMMAND REGISTER
        CASTF0  =       174446  ; CHAN 0 CHAIN ADDRESS SEG/TAG FIELD
        CAOF0   =       174442  ; CHAN 0 CHAIN ADDRESS OFFSET FIELD


START:  MOVB    #130,MMREG      ; LOAD MASTER MODE REG TO DISABLE DTC

        CLRB    CMDREG          ; RESET THE DTC

        MOV     #0,CASTF0       ; LOAD THE CHAIN ADDRESS REG SEG/TAG
        MOV     #RELOAD,CAOF0   ; LOAD THE CHAIN ADDRESS REG OFFSET

        MOVB    #131,MMREG      ; LOAD MASTER MODE REG TO ENABLE DTC

        MOVB    #102,CMDREG     ; SET SOFTWARE REQUEST CHANNEL 0

        MOVB    #240,CMDREG     ; START CHAIN CHANNEL 0

        BR      .               ; STAY HERE WHILE THE USER VERIFIES
                                ; THAT THE PROGRAM WAS SUCCESSFUL

        ; CHAIN LOAD REGION

RELOAD: .WORD   001602  ; RELOAD WORD <Select CARA,CARB,COPC,CM>
```

```
            .WORD   000000  ; CURRENT ADDRESS REGISTER A SEG/TAG
            .WORD   SOURCE  ; CURRENT ADDRESS REGISTER A OFFSET
                            ; <This local address is the source>

            .WORD   000000  ; CURRENT ADDRESS REGISTER B SEG/TAG
            .WORD   DESTNT  ; CURRENT ADDRESS REGISTER B OFFSET
                            ; <This local address is the destination>

            .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

            .WORD   000000  ; CHANNEL MODE REGISTER HIGH
            .WORD   000040  ; CHANNEL MODE REGISTER LOW
                            ; <No match conditions, do nothing upon
                            ;  completion, transfer type = Demand Dedicated
                            ;  w/Bus Hold, CARA = source, word transfers>

SOURCE:     .WORD   1,2,3,4,5,6,7,6,5,4,3,2,1

DESTNT:     .BLKW   13.

            .END START
```

```
        .TITLE EXAM2.MAC

; This program transfers data from local KXT11-C addresses to
; global Q-bus addresses. This program should be compiled and linked on

; the development system and then downloaded into the KXT11-C using the

; KXT11-C Software Toolkit.  Once the program has been compiled and
; linked use the following commands to execute it and verify its
; successfullness.
;
; <HALT the development machine so that locations may be examined
;  with Q-bus ODT>
; @600000/xxxxxx          ! Examine the destination locations and clear
;          .              ! them if necessary
;          .
; @600030/xxxxxx
; @P                      ! Use the 'P' command to return to the
;                         ! system prompt
;
; .KUI
; KUI>SET n               ! Where n is the appropriate KXT11-C
; KUI>LOAD EXAM2
; KUI>EXECUTE
; KUI>EXIT
;
; <HALT the development machine so that locations may be examined
;  with Q-bus ODT>
; @600000/xxxxxx          ! Examine the destination locations to verify
;          .              ! the success of the transfer
;          .
; @600030/xxxxxx
;

;       SET UP REGISTER ASSIGNMENTS

        MMREG   =       174470  ; MASTER MODE REGISTER
        CMDREG  =       174454  ; COMMAND REGISTER
        CASTF0  =       174446  ; CHAN 0 CHAIN ADDRESS SEG/TAG FIELD
        CAOF0   =       174442  ; CHAN 0 CHAIN ADDRESS OFFSET FIELD


START:  MOVB    #130,MMREG      ; LOAD MASTER MODE REG TO DISABLE DTC

        CLRB    CMDREG          ; RESET THE DTC

        MOV     #0,CASTF0       ; LOAD THE CHAIN ADDRESS REG SEG/TAG
        MOV     #RELOAD,CAOF0   ; LOAD THE CHAIN ADDRESS REG OFFSET

        MOVB    #131,MMREG      ; LOAD MASTER MODE REG TO ENABLE DTC

        MOVB    #102,CMDREG     ; SET SOFTWARE REQUEST CHANNEL 0

        MOVB    #240,CMDREG     ; START CHAIN CHANNEL 0
```

```
          BR      .                 ; STAY HERE WHILE THE USER VERIFIES
                                    ; THAT THE PROGRAM WAS SUCCESSFUL

          ; CHAIN LOAD REGION

RELOAD:   .WORD   001602  ; RELOAD WORD <Select CARA,CARB,COPC,CM>

          .WORD   000000  ; CURRENT ADDRESS REGISTER A SEG/TAG
          .WORD   SOURCE  ; CURRENT ADDRESS REGISTER A OFFSET
                          ; <This local address is the source>

          .WORD   101400  ; CURRENT ADDRESS REGISTER B SEG/TAG
          .WORD   000000  ; CURRENT ADDRESS REGISTER B OFFSET
                          ; <This global Q-bus address is the destination>
                          ; <This corresponds to address 600000 on the
                          ;  Q-bus - the DTC uses physical addresses only>

          .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

          .WORD   000000  ; CHANNEL MODE REGISTER HIGH
          .WORD   000040  ; CHANNEL MODE REGISTER LOW
                          ; <No match conditions, do nothing upon
                          ;  completion,transfer type = Demand Dedicated
                          ;  w/Bus Hold, CARA = source, word transfers>

SOURCE:   .WORD   1,2,3,4,5,6,7,6,5,4,3,2,1

          .END START
```

```
        .TITLE EXAM3.MAC

; This program transfers data from global Q-bus addresses to local
; KXT11-C addresses.  This program should be compiled and linked on
; the development system and then downloaded into the KXT11-C using the

; KXT11-C Software Toolkit.  Once the program has been compiled and
; linked use the following commands to execute it and verify its
; successfullness.
;
; <Use Q-bus ODT to deposit values in locations 600000(8) --> 600030(8).

;  These values will be the source for this operation>
;
; @600000/000001          ! Deposit source values
;          .
;          .
; @600030/000001
; @P              ! Use the 'P' command to return to the system prompt
;
; .KUI
; KUI>SET n       ! Where n is the appropriate KXT11-C
; KUI>LOAD EXAM3
; KUI>EXECUTE
; KUI>ODT         ! Use KUI ODT to examine the destination locations to
;                    verify the transfer was successful
; ODT> .
;      .
;      .
; ODT>^C
; KUI>EXIT
;

;       SET UP REGISTER ASSIGNMENTS

        MMREG   =       174470  ; MASTER MODE REGISTER
        CMDREG  =       174454  ; COMMAND REGISTER
        CASTF0  =       174446  ; CHAN 0 CHAIN ADDRESS SEG/TAG FIELD
        CAOF0   =       174442  ; CHAN 0 CHAIN ADDRESS OFFSET FIELD


START:  MOVB    #130,MMREG      ; LOAD MASTER MODE REG TO DISABLE DTC

        CLRB    CMDREG          ; RESET THE DTC

        MOV -   #0,CASTF0       ; LOAD THE CHAIN ADDRESS REG SEG/TAG
        MOV     #RELOAD,CAOF0   ; LOAD THE CHAIN ADDRESS REG OFFSET

        MOVB    #131,MMREG      ; LOAD MASTER MODE REG TO ENABLE DTC

        MOVB    #102,CMDREG     ; SET SOFTWARE REQUEST CHANNEL 0

        MOVB    #240,CMDREG     ; START CHAIN CHANNEL 0
```

```
          BR      .                 ; STAY HERE WHILE THE USER VERIFIES
                                    ; THAT THE PROGRAM WAS SUCCESSFUL

          ; CHAIN LOAD REGION

RELOAD:   .WORD   001602  ; RELOAD WORD <Select CARA,CARB,COPC,CM>

          .WORD   000000  ; CURRENT ADDRESS REGISTER A SEG/TAG
          .WORD   DESTNT  ; CURRENT ADDRESS REGISTER A OFFSET
                          ; <This local address is the destination>

          .WORD   101400  ; CURRENT ADDRESS REGISTER B SEG/TAG
          .WORD   000000  ; CURRENT ADDRESS REGISTER B OFFSET
                          ; <This global Q-bus address is the source>
                          ; <This corresponds to address 600000 on the
                          ;  Q-bus - the DTC uses physical addresses only>

          .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

          .WORD   000000  ; CHANNEL MODE REGISTER HIGH
          .WORD   000060  ; CHANNEL MODE REGISTER LOW
                          ; <No match conditions, do nothing upon
                          ;  completion, transfer type = Demand Dedicated
                          ;  w/Bus Hold, CARB = source, word transfers>
                          ; <Notice how similar this reload table is to
                          ;  the one in EXAM2.  By utilizing the flip bit
                          ;  in the CM Reg Low no further changes were
                          ;  necessary to use this table in this example>

DESTNT:   .BLKW   13.

          .END START
```

```
        .TITLE EXAM4.MAC

; This program transfers data from global Q-bus addresses to other
; global Q-bus addresses.  This program should be compiled and linked
; on the development system and then downloaded into the KXT11-C using
; the KXT11-C Software Toolkit.  Once the program has been compiled and

; linked use the following commands to execute it and verify its
; successfullness.
;
; <Use Q-bus ODT to deposit values in locations 600000(8) --> 600030(8).

;  These values will be the source for this operation>
;
; @600000/000001        ! Deposit source values
;          .
;          .
; @600030/000001
; @P            ! Use the 'P' command to return to the system prompt
;
; .KUI
; KUI>SET n     ! Where n is the appropriate KXT11-C
; KUI>LOAD EXAM4
; KUI>EXECUTE
; KUI>EXIT
;
; <Use Q-bus ODT to examine the destination locations to verify that
;  the operation was sucessful>
; @610000/xxxxxx
;          .
;          .
; @610030/xxxxxx
; @P            ! Return to system prompt

;       SET UP REGISTER ASSIGNMENTS

        MMREG   =       174470  ; MASTER MODE REGISTER
        CMDREG  =       174454  ; COMMAND REGISTER
        CASTF0  =       174446  ; CHAN 0 CHAIN ADDRESS SEG/TAG FIELD
        CAOF0   =       174442  ; CHAN 0 CHAIN ADDRESS OFFSET FIELD


START:  MOVB    #130,MMREG      ; LOAD MASTER MODE REG TO DISABLE DTC

        CLRB    CMDREG          ; RESET THE DTC

        MOV     #0,CASTF0       ; LOAD THE CHAIN ADDRESS REG SEG/TAG
        MOV     #RELOAD,CAOF0   ; LOAD THE CHAIN ADDRESS REG OFFSET

        MOVB    #131,MMREG      ; LOAD MASTER MODE REG TO ENABLE DTC

        MOVB    #102,CMDREG     ; SET SOFTWARE REQUEST CHANNEL 0

        MOVB    #240,CMDREG     ; START CHAIN CHANNEL 0
```

```
          BR                        ; STAY HERE WHILE THE USER VERIFIES
                                    ; THAT THE PROGRAM WAS SUCCESSFUL

          ; CHAIN LOAD REGION

RELOAD:   .WORD    001602   ; RELOAD WORD <Select CARA,CARB,COPC,CM>

          .WORD    101400   ; CURRENT ADDRESS REGISTER A SEG/TAG
          .WORD    000000   ; CURRENT ADDRESS REGISTER A OFFSET
                            ; <This global Q-bus address is the source>
                            ; <This corresponds to Q-bus address 600000(8)>

          .WORD    101400   ; CURRENT ADDRESS REGISTER B SEG/TAG
          .WORD    010000   ; CURRENT ADDRESS REGISTER B OFFSET
                            ; <This global Q-bus address is the destination>
                            ; <This corresponds to Q-bus address 610000(8)>

          .WORD    000013.  ; CURRENT OPERATION COUNT <Transfer 13 words>

          .WORD    000000   ; CHANNEL MODE REGISTER HIGH
          .WORD    000040   ; CHANNEL MODE REGISTER LOW
                            ; <No match conditions, do nothing upon
                            ;  completion, transfer type = Demand Dedicated
                            ;  w/Bus Hold, CARA = source, word transfers>

          .END START
```

```
        .TITLE EXAM5.MAC

; This program demonstrates how chaining is implemented using the
; DTC.  A local to local transfer will be initiated under program
; control.  Then, using the chaining feature of the DTC, a local to
; global transfer will be performed followed by a global to global
; transfer and finally a global to local transfer.  The following
; diagram illustrates these transfers.
;
;                KXT11-C Memory                    Q-bus Memory
;
;
;
;
;Transfer #1
;                                  Transfer #2
;
;
;                                  Transfer #4                    Xfer
;                                                                  #3
;
;
; This program should be compiled and linked on the development system
; and then downloaded into the KXT11-C using the KXT11-C Software
; Toolkit.  Once the program has been compiled and linked use the
; following commands to execute it and verify its successfullness.
;
; <Use Q-bus ODT to clear the memory locations 600000(8) --> 600030(8)
;  and 6100000(8) --> 610030(8) before executing the program>
;
; .KUI
; KUI>SET n      ! Where n is the appropriate KXT11-C
; KUI>LOAD EXAM5
; KUI>EXECUTE
; KUI>ODT        ! Use KUI ODT to verify that the destination contents
; ODT> .           are accurate
; ODT> .
; ODT>^C
; KUI>EXIT
;
; <Use Q-bus ODT to examine the contents of the intermediate
;  destinations to verify their accuracy>

;       SET UP REGISTER ASSIGNMENTS

        MMREG   =       174470  ; MASTER MODE REGISTER
        CMDREG  =       174454  ; COMMAND REGISTER
        CASTF0  =       174446  ; CHAN 0 CHAIN ADDRESS SEG/TAG FIELD
        CAOF0   =       174442  ; CHAN 0 CHAIN ADDRESS OFFSET FIELD
```

```
START:  MOVB    #130,MMREG      ; LOAD MASTER MODE REG TO DISABLE DTC

        CLRB    CMDREG          ; RESET THE DTC

        MOV     #0,CASTF0       ; LOAD THE CHAIN ADDRESS REG SEG/TAG
        MOV     #LOAD1,CAOF0    ; LOAD THE CHAIN ADDRESS REG OFFSET

        MOVB    #131,MMREG      ; LOAD MASTER MODE REG TO ENABLE DTC

        MOVB    #102,CMDREG     ; SET SOFTWARE REQUEST CHANNEL 0

        MOVB    #240,CMDREG     ; START CHAIN CHANNEL 0

        BR      .               ; STAY HERE WHILE THE USER VERIFIES
                                ; THAT THE PROGRAM WAS SUCCESSFUL

        ; CHAIN LOAD REGION

LOAD1 : .WORD   001603  ; RELOAD WORD <Select CARA,CARB,COPC,CM,CA>

        .WORD   000000  ; CURRENT ADDRESS REGISTER A SEG/TAG
        .WORD   AREA1   ; CURRENT ADDRESS REGISTER A OFFSET
                        ; <This local address is the source of transfer
                        ;  #1>

        .WORD   000000  ; CURRENT ADDRESS REGISTER B SEG/TAG
        .WORD   AREA2   ; CURRENT ADDRESS REGISTER B OFFSET
                        ; <This local address is the destination of
                        ;  transfer #1>

        .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

        .WORD   000000  ; CHANNEL MODE REGISTER HIGH
        .WORD   100040  ; CHANNEL MODE REGISTER LOW
                        ; <No match conditions, chain reload upon
                        ;  completion, transfer type = Demand Dedicated
                        ;  w/Bus Hold, CARA = source, word transfers>

        .WORD   000000  ; CHAIN ADDRESS REGISTER SEG/TAG
        .WORD   LOAD2   ; CHAIN ADDRESS REGISTER OFFSET
                        ; <This address points to the new chain table>

LOAD2 : .WORD   001603  ; RELOAD WORD <Select CARA,CARB,COPC,CM,CA>

        .WORD   000000  ; CURRENT ADDRESS REGISTER A SEG/TAG
        .WORD   AREA2   ; CURRENT ADDRESS REGISTER A OFFSET
                        ; <This local address is the source of xfer #2>

        .WORD   101400  ; CURRENT ADDRESS REGISTER B SEG/TAG
        .WORD   000000  ; CURRENT ADDRESS REGISTER B OFFSET
                        ; <This global address is the destination of
```

```
                        ;  transfer #2 - 600000(8)>

        .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

        .WORD   000000  ; CHANNEL MODE REGISTER HIGH
        .WORD   100040  ; CHANNEL MODE REGISTER LOW
                        ; <No match conditions, chain reload upon
                        ;  completion, transfer type = Demand Dedicated

                        ;  w/Bus Hold, CARA = source, word transfers>

        .WORD   000000  ; CHAIN ADDRESS REGISTER SEG/TAG
        .WORD   LOAD3   ; CHAIN ADDRESS REGISTER OFFSET
                        ; <This address points to the new chain table>

LOAD3 : .WORD   001603  ; RELOAD WORD <Select CARA,CARB,COPC,CM,CA>

        .WORD   101400  ; CURRENT ADDRESS REGISTER A SEG/TAG
        .WORD   000000  ; CURRENT ADDRESS REGISTER A OFFSET
                        ; <This global address is the source of xfer #3>
                        ;  <600000(8)>

        .WORD   101400  ; CURRENT ADDRESS REGISTER B SEG/TAG
        .WORD   010000  ; CURRENT ADDRESS REGISTER B OFFSET
                        ; <This global address is the destination of
                        ;  transfer #3 - 610000(8)>

        .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>

        .WORD   000000  ; CHANNEL MODE REGISTER HIGH
        .WORD   100040  ; CHANNEL MODE REGISTER LOW
                        ; <No match conditions, chain reload upon
                        ;  completion, transfer type = Demand Dedicated

                        ;  w/Bus Hold, CARA = source, word transfers>

        .WORD   000000  ; CHAIN ADDRESS REGISTER SEG/TAG
        .WORD   LOAD4   ; CHAIN ADDRESS REGISTER OFFSET
                        ; <This address points to the new chain table>

LOAD4 : .WORD   001602  ; RELOAD WORD <Select CARA,CARB,COPC,CM>

        .WORD   101400  ; CURRENT ADDRESS REGISTER A SEG/TAG
        .WORD   010000  ; CURRENT ADDRESS REGISTER A OFFSET
                        ; <This global address is the source of sfer #4>
                        ;  <610000(8)>

        .WORD   000000  ; CURRENT ADDRESS REGISTER B SEG/TAG
        .WORD   AREA3   ; CURRENT ADDRESS REGISTER B OFFSET
                        ; <This local address is the destination of
                        ;  transfer #4>

        .WORD   000013. ; CURRENT OPERATION COUNT <Transfer 13 words>
```

```
            .WORD   000000  ; CHANNEL MODE REGISTER HIGH
            .WORD   000040  ; CHANNEL MODE REGISTER LOW
                            ; <No match conditions, do nothing upon
                            ;  completion, transfer type = Demand Dedicated
                            ;  w/Bus Hold, CARA = source, word transfers>

AREA1 :     .WORD   1,2,3,4,5,6,7,6,5,4,3,2,1

AREA2 :     .BLKW   13.

AREA3 :     .BLKW   13.

            .END START
```

```
        .TITLE  EXAM6.MAC

; This program demonstrates how to initiate a DTC operation from the
; arbiter CPU.  This program will tranfer a block of data from Q-bus
; memory to KXT11-C memory.  All of the information necessary for the
; transfer will reside in Q-bus memory (chain table, source data)
; This program should be compiled, linked, and run on the arbiter
; development system.  After the program executes use the following
; KUI commands to verify the transfer
;
; .KUI
; KUI>SET n     ! Where n is the appropriate KXT11-C
; KUI>ODT
; ODT>5000/xxxxxx  ! Examine locations 5000 —> 5030 to verify that
;       .                  the data was transfered correctly
;       .
; ODT>5030/xxxxxx
; ODT>^C
; KUI>EXIT
;
;       Two-port RAM register definitions

        TPR0=160100
        TPR1=160102
        TPR2=160104
        TPR3=160106

        .MCALL  .EXIT

START:  MOV     #100000,TPR3 ; Place Chain Address Reg Seg/Tag in TPR3

        MOV     #LOAD,TPR2   ; Place Chain Address Reg Offset in TPR2

;       *  NOTE!!  *
;
;       The KXT11-C User's Guide contains an error which instructs the

;       programmer to place the CA register Seg/Tag in TPR2 and the CA

;       register Offset in TPR3.  This information is reversed and is
;       correct as stated above.

        BIS     #2,TPR0 ; Issue DMA Load cmd to the command register

        .EXIT

LOAD  : .WORD   001602  ; RELOAD WORD <Select CARA,CARB,COPC,CM>

        .WORD   100000  ; CARA SEG/TAG <Select Q-bus address as source>
        .WORD   SOURCE  ; CARA OFFSET

        .WORD   000000  ;CARB SEG/TAG <Select KXT address 5000 as dest.
        .WORD   005000  ;CARB OFFSET

        .WORD   000013. ; COPC <Op-count = 13 words>
```

```
        .WORD   000000  ; CM High
        .WORD   000040  ; CM Low <select no termination options,
                        ;         software hog-mode, CARA = source,
                        ;         word transfers>

SOURCE: .WORD   1,2,3,4,5,6,7,6,5,4,3,2,1

        .END    START
```

RELATED DOCUMENTS

For further information concerning the KXT11-CA and the DTC please consult the following sources:

| | |
|---|---|
| KXT11-CA Single-Board Computer User's Guide | EK-KXTCA-UG-001 |
| AmZ8016 DMA Transfer Controller User's Guide | 01924C |

For further information concerning the KXT11-CA Peripheral Processor Software Toolkit please consult:

| | |
|---|---|
| KXT11-C Peripheral Processor Software User's Guide | AA-Y615A-TK |
| KXT11-CA Software Toolkit/RT Reference Manual | AA-AU63A-TC |
| KXT11-CA Software Toolkit/RSX Reference Manual | AA-AU64A-TC |

| uNOTE # 019 |
|---|

| Title: Disabling RAM on the MXV11-BF | Date: 10-JAN-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 3 |

This uNOTE describes how to disable the on-board memory of the MXV11-BF multifunction module. There is also a comparison of features between the MXV11-BF (with RAM disabled) and the combination of the MRV11-D and DLV11-J modules.

## DISABLING RAM ON THE MXV11-BF MODULE

WARNING!

The procedure outlined below for disabling the RAM requires physical changes to the circuit board. Implementing this change will void the warranty of the MXV11-BF as well as voiding any field service contract for the board.

The procedure requires cutting one etch and adding one wire.

PROCEDURE (see figure below)

1. The etch cut can be achieved two ways.

   Cut and bend up pin 13 of chip E17.

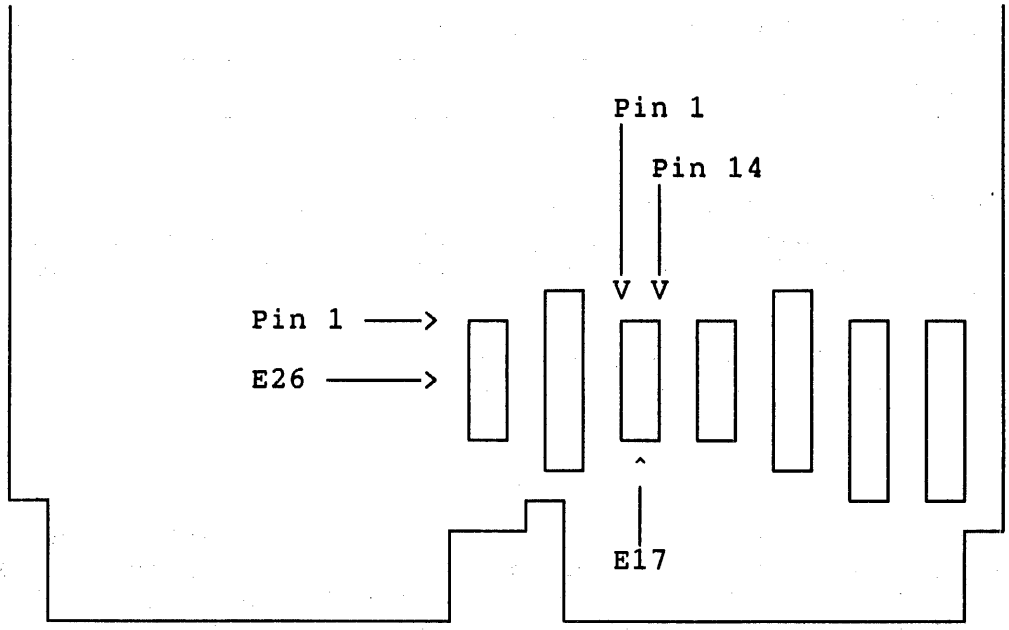
   OR

   On side 2 of module (the non-component side), cut etch which connects E17 pin 13 to E26 pin 2.

2. Add wire between E17 pin 13 and E17 pin 14 (+5 Volts).

3. The on-board RAM is now disabled.

MXV11-BF (M7195)




## COMPARISON OF THE MXV11-BF VERSUS THE COMBINATION OF MRV11-D AND DLV11-J

The following table lists several features common to both the MXV11-BF with the RAM disabled and the MRV11-D / DLV11-J combination.

| FEATURE | MXV11-BF w/o RAM | MRV11-D and DLV11-J |
|---|---|---|
| RAM | None | None |
| Boot ROMs | MXV11-B2 | MXV11-B2 |
| SLU's | 2 DLARTs<br>4 baud rates<br>No format options<br>(Start, stop bits, parity, # of bits, etc.) | 4 UARTs<br>9 baud rates<br>Many format options |
| Miscellaneous features | 4 Red LED's<br>1 Green LED | 14 unused ROM/RAM sockets |
| Cabinet kit | Currently N/A | Available |
| +5 V | 3.4A | 2.6A |
| Heat diss. | 16.65W | 16W |

| FEATURE | MXV11-BF w/o RAM | MRV11-D and DLV11-J |
|---|---|---|
| Slots | 1 dual | 2 duals |
| AC loads | 2.3 | 4.0 |
| DC loads | 0.5 | 1.5 |
| +12 V | 0.1A | 0.25A |

When comparing the features on page 2, the MRV11-D/DLV11-J combination works out better than the MXV11-BF. All of the features on page 3 show the advantages of the MXV11-BF.

The MXV11-BF is the best choice in situations where backplane space and loading is a primary concern. In terms of backplane space, if the number of slots available is limited, the MXV11-BF is preferable since it takes up one dual slot whereas the MRV11-D/DLV11-J combination requires two dual slots.

In a large system where AC loading may be a concern, the MXV11-BF also has an advantage over the MRV11-D/DLV11-J combination. The number of AC loads the MXV11-BF presents to the bus is fewer than the two board combination.

The most attractive use for the MXV11-BF is in a minimal yet powerful two-board system i.e. KDJ11-A and the MXV11-BF. The advantages of such a system are the size or form factor (only two dual height modules) 128Kb of RAM, 2 serial lines, bootstrap capability and the high performance 11/73 CPU.

| uNOTE # 020 |
|---|

| Title: Differences between the MXV11-A and MXV11-B | Date: 10-JAN-85 |
|---|---|
| Originator: Dave Smith | Page 1 of 2 |

This MicroNote illustrates the features to consider when upgrading from MXV11-AA/-AC to MXV11-BF multifunction boards or when choosing one in a new configuration.

| MXV11-A | MXV11-BF |
|---|---|
| o Dual height form factor<br>o 18-bit compatible only<br>o 2 SLU's (DLV11-J type)<br>o RAM:<br>8 KB non-parity (MXV11-AA)<br>32 KB non-parity (MXV11-AC)<br>o ROM: 2 sockets (24-pin)<br>(for MXV11-A2 boot ROMS<br>- cannot use MXV11-B2 ROMS) | o Dual height form factor<br>o 18 or 22-bit compatible<br>o 2 SLU's (DLARTs)<br>o RAM:<br>128 KB non-parity<br>o ROM: 2 sockets (28-pin)<br>(for MXV11-B2 boot ROMS<br>- cannot use MXV11-A2 ROMS) |

Notes:

The MXV11-B2 Boot ROMs are required for an LSI-11/73 based system to be maintained under a DEC Field Service contract because these ROMs contain cache diagnostics.

For a comprehensive list of Bootstraps and device support, refer to MicroNote #15, entitled "Q-bus Hardware Bootstraps"

Since the ROM sockets on these boards may be used for user-written application code as well as boot code, it is important to note what is different on the two boards. Besides the sockets being of different size, the ROM on the two boards has another difference. The ROM on the MXV11-BF is window mapped and may be used under application control to address up to 8 KB. The MXV11-A ROM is direct mapped and can only address 256 bytes. Thus, an upgrade would require new ROMs and possible application recoding.

For more detailed information concerning MXV11-BF issues, refer to MicroNote 19.

For more detailed information concerning MXV11-A issues, refer to the archived MicroNotes listed in appendix A.

Summary:

When upgrading from the MXV11-A to the MXV11-BF or choosing the MXV11-BF over the MXV11-A, three important features are obtained:

1) 22-bit compatibility - allows use in large memory systems
   (The MXV11-BF memory can only be configured to start in the lower 512 KB. It does, however, decode all 22 lines of address which makes it usable in systems with up to 4 MB.)

2) More memory (although its non-parity feature creates some software support issues if used with RSX or RSTS)

3) Support for boot ROMs that will boot MSCP devices such as the RX50 and RD51 and will provide cache diagnostics for the LSI-11/73.

| uNOTE # 021 |
|---|

| Title: Floating Point Considerations on MicroVAX I | Date: 10-Jan-85 |
|---|---|
| Originator: Christopher DeMers | Page 1 of 2 |

The MicroVAX architecture implements a subset of VAX data types. Four that are part of the VAX architecture but not part of the MicroVAX architecture are the D_floating, F_floating, G_floating and H_floating data types. Floating point instructions that use these data types are likewise not part of the architecture.

The MicroVAX I system implements a proper superset of the MicroVAX architecture; that is, MicroVAX I uses the MicroVAX architecture, but implements a few items that are not defined as part of the MicroVAX architecture. Even though the architecture specifies emulation support for floating point, the following data types and instructions are supported in the MicroVAX I microcode:

- F_float
- D_float
- G_float

The F_float data type and instructions are standard on the MicroVAX I. In addition, the user may specify EITHER D_float or G_float as the double precision instruction set. The decision to use D_float or G_float depends on the application.

Both D_float and G_float are double precision floating point data types/instructions. D_float is compatible with the PDP-11 format and is the double precision floating point default for many of Digital's compilers including FORTRAN, PL/I, BASIC and Pascal. Therefore, if the application has been compiled, say, on a VAX-11/780, with the default (D_float) and is not to be re-compiled before being run on the MicroVAX I, then choose the D_float option. The compilers mentioned above have a switch that allows the generation of G_float instructions. If you wish to choose the G_float option for your MicroVAX I, the program needs to be re-compiled with the G_float switch set.

If Macro programs use a specific data type such as G_float, then the MicroVAX I will need to have the G_float option unless the program is modified so that the floating point instructions match the option chosen for the MicroVAX I.

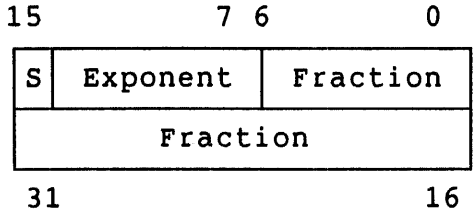
Note that even though a program uses one type of double precision floating point and the MicroVAX I has the other as an option, the program will run. A feature of the MicroVAX architecture is that all instructions are executed. The floating point instructions not chosen as a microcode option are emulated in software. An instruction/data type mismatch could result in severe performance degradation.

Another reason for choosing one double precision format over another is the size and accuracy of the data. Both formats are 64 bits long. The D_float range is 2.9E-37 to 1.7E38. This type gives approximately sixteen decimal digits precision. G_float "steals" three bits from the fraction to give the exponent a larger range. The G_float range is .56E-308 to .9E308 and gives approximately fifteen decimal digits precision. The range of the number is increased significantly while only reducing the precision by one decimal digit.

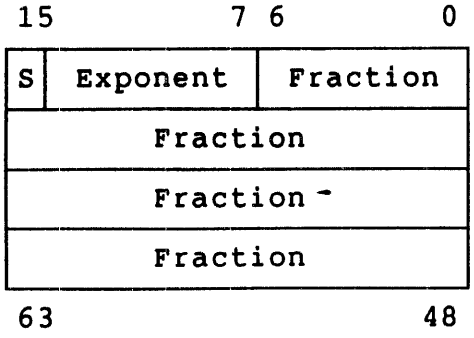
H_float, the only other floating data type in the VAX architecture, is 128 bits long with a range of close to 10E5000 and a precision of thirty-three decimal digits. H_float is not implemented as part of the MicroVAX architecture. It is, however, emulated in software.

Data Type Representations

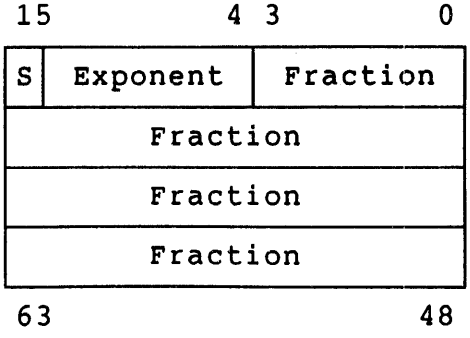
F_Floating




D Floating




G_Floating



| uNOTE # 022 |
|---|

| Title: Differences between the MicroVAX I and the MicroVAX II CPUs | Date: 28-APR-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 13 |

This MicroNote identifies the differences between the MicroVAX I processor and the MicroVAX II processor. The term 'MicroVAX 630' is sometimes used in this and other documentation when referring to the CPU module of the MicroVAX II system. Table 1 below contains a summary of these differences and following the table are detailed discussions of each.

Table 1 - MicroVAX II versus the MicroVAX I

| FEATURE | MicroVAX II | MicroVAX I |
|---|---|---|
| CPU Technology | MicroVAX 78032 | Custom VLSI |
| Memory System | Local Memory System | Uses Q-bus Memory |
| Floating Point | MicroVAX 78132 | Microcode |
| Q-bus I/O Map | YES | NO |
| Addressable Physical Memory | 16 MBytes | 4 MBytes |
| On-Board Memory | 256KB or 1MB | None |
| Performance | See Performance Section | |
| Console Serial Line | YES | YES |
| Boot & Diagnostic ROMs | YES | YES |
| Form Factor | 1 Quad | 2 Quads |
| Configuration Set via | Cabinet Kit | Cabinet Kit & On-board Switches |
| RQDX Controller Type | RQDX2 | RQDX1 or RQDX2 |

Table 1 - MicroVAX II versus the MicroVAX I (cont'd)

| | | |
|---|---|---|
| TOY Clock w/ Battery Backup | YES | NO |
| Multicomputing Hooks | YES | NO |
| Instruction Set Differences | See MicroNote #24 | |
| Power Requirements | +5V - 6.2 Amps<br>+12V - 0.14 Amps | +5V - 14 Amps<br>+12V - 0.5 Amps |
| AC Loads | 2.7 | 2 |
| DC Loads | 1 | 1 |
| Termination | 240 Ohms | 240 Ohms |

The MicroVAX II is both faster and smaller than the MicroVAX I. Two major features of the MicroVAX II are responsible for these enhancements: the MicroVAX 78032 microprocessor and the memory system.

## CPU TECHNOLOGY

The technologies used to design the two processors are different. This is why the MicroVAX II CPU is half the size and three times the speed of the MicroVAX I.

The MicroVAX I CPU was designed using a custom VLSI chip, several ROMs and off-the-shelf parts to implement the ALU, memory management unit and the microcode. In contrast, these same features are implemented in the MicroVAX 78032 microprocessor. The MicroVAX 78032 is the first VAX on a chip. Two custom gate arrays were also designed to reduce the size of the MicroVAX II CPU to one quad module: the Q-bus interface gate array and the MicroVAX 78032 interface gate array.

## MEMORY SYSTEM

The significant difference in performance is due in part to the different memory systems used by the two processors. The MicroVAX II memory system uses a high speed interconnect between the MicroVAX 78032 microprocessor and RAM. The MicroVAX I uses standard Q-bus memories which are not as fast but are also less expensive and usable by both the MicroVAX I and other 16-bit processors (such as the MicroPDP-11). The MicroVAX I also uses a cache memory scheme.

The MicroVAX II CPU communicates to memory over a local memory interconnect. This was done to increase the performance of the system in two ways. First, the local memory system provides a fast connection between the processor and memory (400 nsec read and write cycles). Second, since memory fetches occur over the local memory system, the Q-bus is now a dedicated I/O bus. The diagrams below contrast previous Q-bus processors and the MicroVAX II.

The local memory system allows for 2 expansion memory boards. (See next section on Addressable Physical Memory)

Configuration Used by Previous Q-bus Processors:
This approach is used by most Q-bus processors such as the 11/23, LSI-11/73 and the MicroVAX I. In this design, the Q-bus bandwidth is shared by the processor and I/O devices when accessing memory.

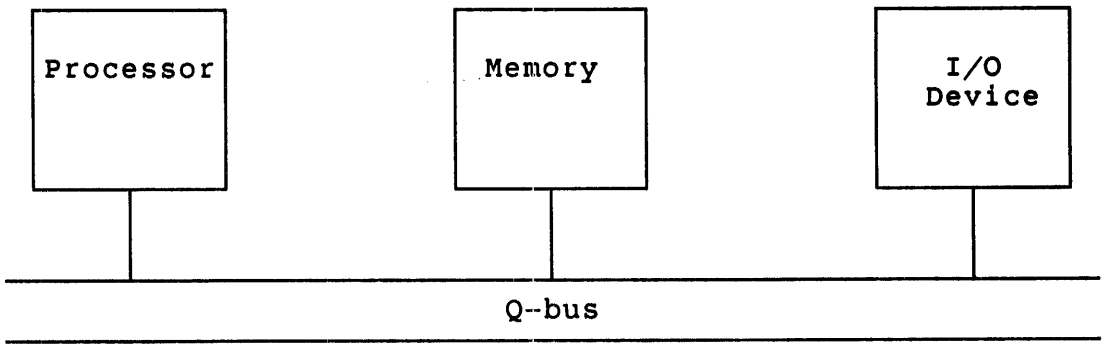



Figure 1 - Memory Design Used by Previous Q-bus Processors

MicroVAX II Design:
The MicroVAX II uses a local memory system for all memory references, allowing the Q-bus to be a dedicated I/O bus. The local memory system is implemented using the C-D interconnect of the backplane and an over-the-top cable. The memory system has a bandwidth of 10 MBytes/sec which will support the MicroVAX 78032 microprocessor running at full speed (approx. 6 MB/sec) with simultaneous Q-bus DMA transfers (3.3 MB/sec).

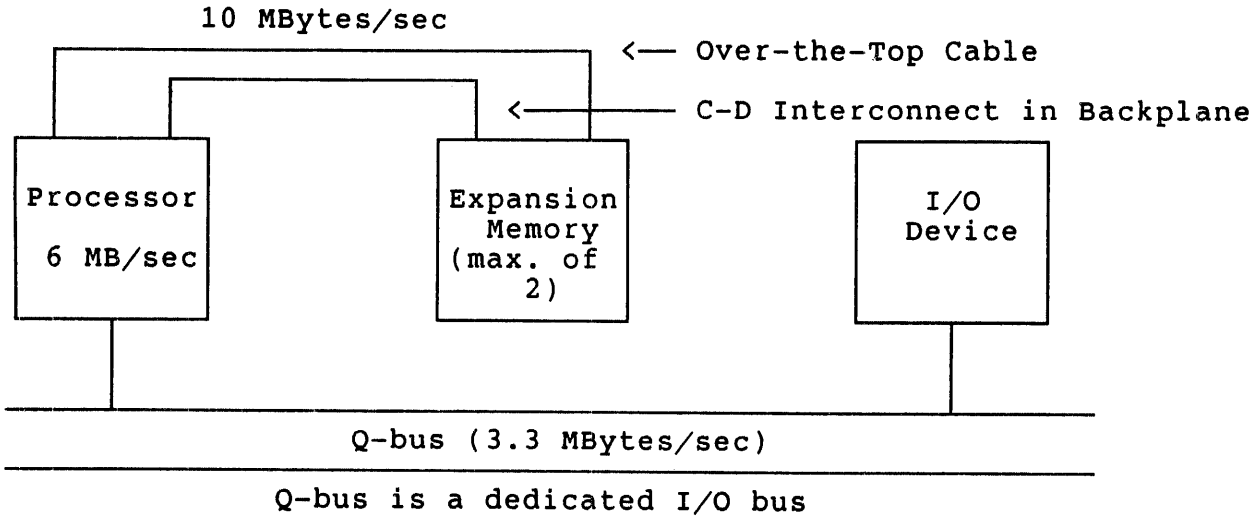



Figure 2 - MicroVAX II Memory Design

## FLOATING POINT

The MicroVAX architecture specifies that the floating point instruction set need not be implemented in the hardware of a MicroVAX processor. For those processors that fall into this category, there is an emulator in the software which guarantees that the instructions are still executable. However the MicroVAX architecture does not restrict a MicroVAX design from having floating point instructions implemented in hardware.

This is what was done on the MicroVAX I. The MicroVAX I does have the F_floating and D_floating or the F_floating and G_floating instructions in microcode. (see MicroNote #21 "Floating Point Considerations on MicroVAX I")

The MicroVAX II uses the MicroVAX 78132 floating point unit to implement floating point instructions. It is a high performance coprocessor for the MicroVAX 78032 and eliminates the need to emulate floating-point instructions in software.

The MicroVAX 78132 handles the F_floating (single precision), D_floating (double-precision) and the G_floating (extended range double-precision) floating point data types and instructions. The FPU also accelerates the execution of integer multiply and divide operations.

The H_floating instructions are emulated in both the MicroVAX I and the MicroVAX II.

## Q-bus I/O MAP (SCATTER/GATHER MAP)

The concept of a scatter/gather map has been used on the large UNIBUS VAXes since the VAX 11/780. The same concept is used on the MicroVAX II. Simply stated, the scatter/gather mechanism maps addresses from one bus to another bus. This mapping is necessary because the address length is not the same for the two buses.

The term 'scatter/gather' is a description of what operations are performed through the scatter/gather map. A VAX with mapping enabled is a virtual machine and manages data in 512-byte pages which may be discontiguous in physical memory. When a DMA write operation occurs, pages of data may be 'scattered' into physical memory. During a DMA read operation from memory to an I/O device, such as a disk, these pages of data must be 'gathered' from throughout memory and transferred to the device.

The MicroVAX I does not require a Q-bus I/O map. Since all I/O devices and memory share the same bus, the Q-bus, there is no mismatch in address lengths and no I/O map is necessary. Reading and writing data to and from memory in 512 byte pages becomes the responsibility of the I/O device or the system software.

In the case of the MicroVAX II CPU, the scatter/gather map provides the mapping mechanism to allow DMA devices on the Q-bus, with a 4 MByte physical address range, to gain access to all of main memory on the local memory system, a 16 MByte physical address range.

Figure 3 below illustrates the mapping process.

The high order 13 bits of the Q-bus address select one of 8192 mapping registers. Each register translates an address for a range covering one page of 512 bytes; thus the mapping registers cover the entire Q-bus address space of 4MB.

The lower 15 bits of the mapping register will have been previously loaded with the correct information, which when appended to the lower 9 bits of the Q-bus address selects the appropriate byte in main memory.

The high order 15 bits of the physical address select which page in main memory has the correct address and the low order 9 bits select the byte within a page.

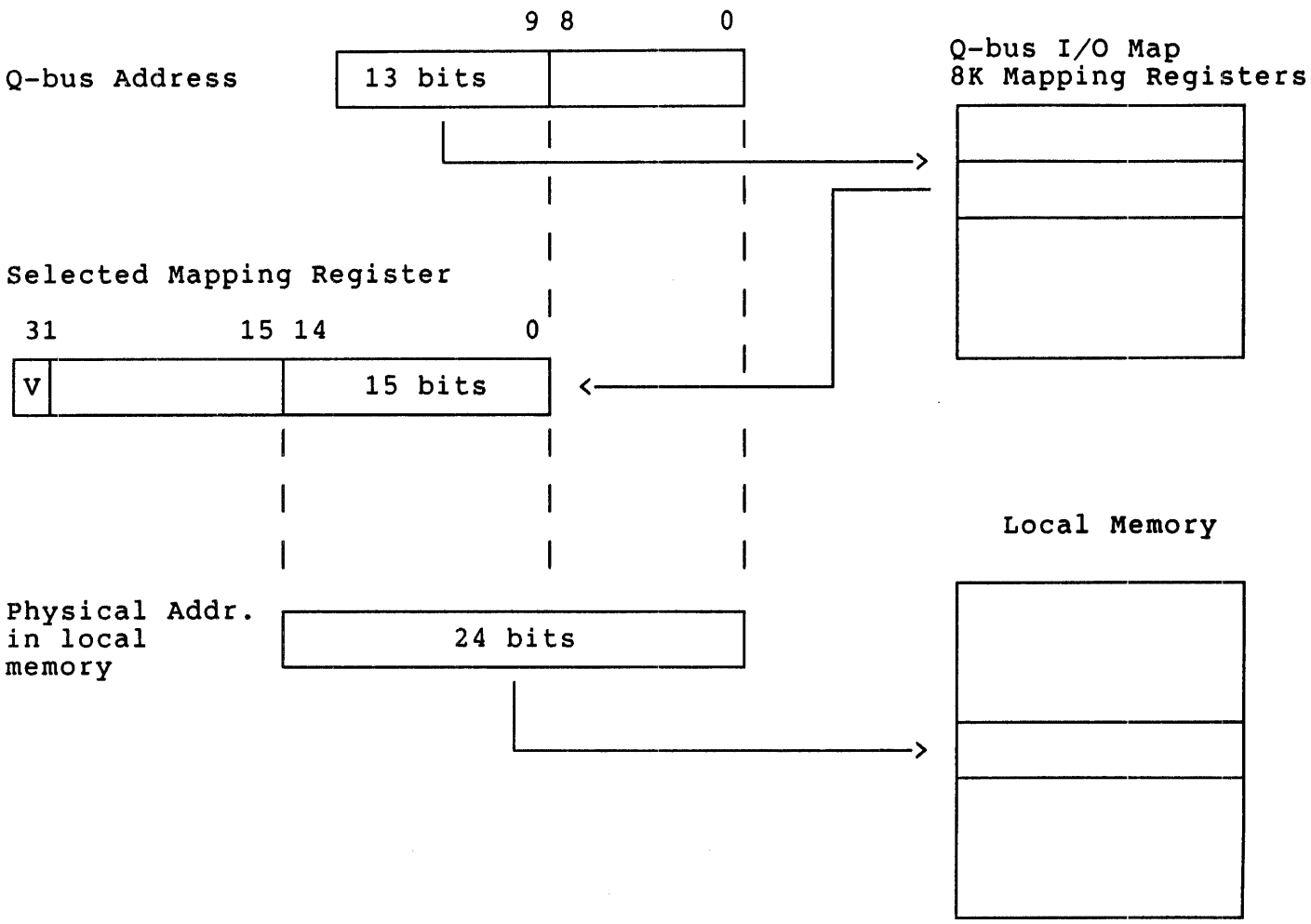



Figure 3 - Q-bus to Private Memory Mapping Process

## ADDRESSABLE PHYSICAL MEMORY

The total amount of addressable physical memory for each CPU is different because of the memory system designs.

The MicroVAX I uses standard Q-bus memories for main memory and therefore has a lower physical addressing limit than the MicroVAX II. This limit is constrained by the number of address lines on the Q-bus. There are 22 address lines on the Q-bus, limiting maximum memory to 4 MB.

The MicroVAX II uses a 24 bit address when accessing the local memory system. Therefore the maximum amount of addressable memory is 16 MBytes.

When the system is first available, 9 Mbytes will be the maximum because the 16 MByte total depends on the availability of 1 Mbit RAM parts. The 9 MByte total includes a MicroVAX II CPU (KA630-AA) which includes 1 MByte on-board plus two 4 MByte expansion modules.

The memory module capacities are listed in the table 3 below.

Table 3 - Expansion Memory Options

| MEMORY | MEMORY SIZE | FORM FACTOR |
|---|---|---|
| MS630-AA | 1 Mbyte | Dual |
| MS630-BA | 2 Mbytes | Quad |
| MS630-BB | 4 Mbytes | Quad |

## PERFORMANCE

The relative performance between the processors is of course application dependent. However, as a general rule of thumb, the MicroVAX II CPU will be approximately 3 times faster than the MicroVAX I. The results of standard, compute bound floating point benchmarks indicate that the MicroVAX II (with the MicroVAX 78132 FPU) will run 4 to 6.6 times faster than the MicroVAX I.

## CONSOLE SERIAL LINE

The MicroVAX II and the MicroVAX I processors both have one dedicated serial line for the console terminal. There are two differences between the two serial line units. First, the number of selectable baud rates is different. The possible baud rates are listed in table 4.

Table 4 - Baud Rate Selections

| MicroVAX II | MicroVAX I |
|---|---|
| 300 | 300 |
| 600 | 1200 |
| 1200 | 9600 |
| 2400 | 19200 |
| 4800 | |
| 9600 | |
| 19200 | |
| 38400 | |

Second, the console device connector on the patch panel for the MicroVAX I is an RS232 25-pin D-subminiature connector. The connector on the MicroVAX II patch panel is a 9-pin D-subminiature connector. Because the connectors are different, different cables will be used to connect to the console terminal.

## FORM FACTOR

All of the features and functions of the MicroVAX II processor are contained on one quad module. The MicroVAX I processor is made up of two quad modules, a data path module ('DAP'), and a memory controller ('MCT'). The two modules of the MicroVAX I communicate over the C-D backplane interconnect and an over-the-top ribbon cable.

## BOOT AND DIAGNOSTIC ROMs

Both processors have boot/diagnostic ROMs. The ROMs on the MicroVAX I are NOT the same as those on the MicroVAX II CPU, therefore there are differences in their operation.

Both ROMs perform the following functions :
- Initialize the machine into a known state.
- Perform diagnostic tests.
- Allow for the automatic restart or bootstrap of the system following a processor halt or initial power-up.
- Provide bootstrap capability from a variety of devices.
- Provide the VAX console command language.

Perform diagnostic tests.
When either machine is powered on, their respective diagnostic tests are executed.

Microverify is the name given to the diagnostics executed by the MicroVAX I system. There are three LEDs on the DAP module and a seven segment display on the patch panel which will isolate the problem to one of the two CPU modules. Error codes are defined in table 10-1 on page 10-4 of the MicroVAX I Owner's Manual.

A similar function is performed by the MicroVAX II CPU diagnostics. As each test is run, a code is displayed on the processor LEDs, the patch panel and the console terminal. These codes count down in hexadecimal from F to 0. There are potentially 16 steps which may be executed, therefore the MicroVAX 630 has 4 LEDs. The definitions for these codes may be found in the MicroVAX 630 CPU Module User's Guide (P/N CK-KA630-UG).

Power-up modes.
A major difference between the two processors is what occurs at power-up. The MicroVAX I will either attempt to do a restart, bootstrap or halt and enter console mode (the appropriate option is selected via two switches on the DAP module).

The MicroVAX II can also be configured to try a restart, perform a bootstrap or halt. Before it attempts these options however, it does a language inquiry. All console messages may be output in one of eleven different languages; one must be selected. There are options such as defaulting to English or defaulting to the language which was selected previously for unassisted bootstrap or always prompting for the language at power-up. These options are described in the MicroVAX 630 CPU Module User's Guide.
Bootstrap Devices.
The list of devices which each processor can boot from is slightly different. Table 5 lists the supported boot devices for each processor. The devices are listed in the order by which they are searched by the processor.

Table 5 - Bootstrap Devices

| MicroVAX II | MicroVAX I |
|---|---|
| RQDX, KDA50 or RC25 | RQDX |
| TK50 | |
| MRV11-D | MRV11-D |
| DEQNA | DEQNA |

VAX Console Command Language.
Both processors implement the VAX console command language in their boot ROMs. Most of the frequently used commands are implemented by both machines. However, there are a few commands which are only implemented by one or the other. Table 6 lists the commands and indicates which are available on each processor.

Table 6 - VAX Console Commands

| COMMAND | MicroVAX II | MicroVAX I |
|---|---|---|
| BREAK | X | X |
| INITIALIZE | X | X |
| START | X | X |
| CONTINUE | X | X |
| HALT | X | X |
| BOOT | X | X |
| UNJAM | X | X |
| EXAMINE | X | X |
| DEPOSIT | X | X |
| X | X | X |
| FIND | X | |
| REPEAT | X | |
| TEST | X | X |
| ! (Comment) | X | |
| N (Next) | | X |
| CTRL/U | X | X |
| CTRL/S | X | |
| CTRL/Q | X | |
| CTRL/O | X | |
| CTRL/R | X | |
| CTRL/C | X | |

## CONFIGURATION

Both CPUs have a cabinet kit through which certain features are configured. However the cabinet kits are different. The cabinet kit for the MicroVAX I will not work with the MicroVAX II CPU and vice versa.

In addition to the cabinet kit are eight switches on one of the MicroVAX I CPU boards used for setting such functions as the recovery action, break detect enable and the baud rate selection. The cabinet kit for the MicroVAX I is a set of two cables and a patch panel insert for FCC system integration. It performs three functions:

1. Console terminal connector
2. Baud rate rotary switch
3. Two digit LED display

The MicroVAX II CPU has no switches or jumpers on the module itself. All options are set via the patch panel insert of the cabinet kit or an optional configuration card used in place of the cabinet kit. The patch panel insert has the following functions :

1. Halt enable switch
2. Power-Up mode rotary switch
3. Baud rate rotary switch
4. Hexadecimal display
5. Console terminal connector
6. Batteries for battery backup

For MicroVAX II applications which do not need the cabinet kit, Digital designed a special configuration card. This card allows the user to configure the same functions as the cabinet kit but it is much smaller and requires no cables. The configuration card is 1.5" x 2.5" and plugs into two connectors on the CPU module. Simple DIP switches are used to select the appropriate functions.

## TIME-OF-YEAR CLOCK WITH BATTERY BACKUP

The MicroVAX I does not have the capability for keeping time after a system power-down or power failure. Each time the system is brought back up the time must be reentered manually (This limitation can be bypassed under MicroVMS for example, but the system time will be incorrect until set at a later time).

The MicroVAX II has the capability for a battery backed up time-of-year (TOY) clock. A low-power TOY clock chip was designed onto the CPU module. After a system power-down the chip keeps track of the correct time and the system can reboot without needing an operator to enter the time and date.

During normal operation the system keeps track of time using a 10 msec interval timer internal to the MicroVAX 78032. When power is lost this interval timer will stop but the TOY clock chip will continue to operate. The TOY clock chip has a resolution of 1 second.

The batteries for the time-of-year clock chip are located on the patch panel insert of the cabinet kit.

The battery backup capability is not present if a system is using the configuration card instead of the cabinet kit in order to configure the system. However there are pins on the configuration card which can be used to connect external batteries to provide the same capability. It is the user's responsibility to provide the batteries and cabling for battery backup when using the configuration card.

## RQDX CONTROLLER TYPE

Initial MicroVAX I systems use the RQDX1 disk controller for the RX50 floppy disks and the RD51 / RD52 winchester disks. This controller is required to be the last module in a system because it does not pass interrupt acknowledge (IACK) nor the DMA grant (BDMG) signals to options which come after it in the backplane.

The RQDX2 disk controller was designed to work with and is shipped with initial MicroVAX II systems. The RQDX2 will also work with other Q-bus processors such as the MicroVAX I and MicroPDP-11/73.

IMPORTANT! The RQDX1 is incompatible with the MicroVAX II CPU therefore the two should never be configured into the same system.

## MULTICOMPUTING HOOKS

The MicroVAX II CPU has been designed to allow a maximum of four processors to coexist in the same system. There will always be an arbiter and up to 3 auxiliaries. This feature is NOT supported by any Digital operating system and is not an off-the-shelf multiprocessing solution. For more information on the specifics of the multicomputing feature, reference MicroNote #26 and the MicroVAX 630 CPU Module User's Guide.

## INSTRUCTION SET DIFFERENCES

The MicroVAX architecture specifies which instructions must be implemented in the hardware/microcode, all other instructions will then be emulated in software. Depending on the particular instruction, the emulation is either entirely in software or in software with a hardware assist. Any specific MicroVAX implementation will meet this minimum requirement but may choose to include in its microcode some instructions which would otherwise be emulated. MicroNote #24 contains a table which lists all of the instructions in the VAX architecture and indicates for each MicroVAX processor whether or not the instruction is in the hardware or emulated.

## POWER REQUIREMENTS, AC LOADS, DC LOADS AND TERMINATION

Table 7 below summarizes the pertinent information necessary to configure one of these processors into a system.

Since the MicroVAX II CPU is only one quad module it is not surprising that the amount of current drawn from the +5 volt supply is substantially less than that required by the MicroVAX I CPU.

Table 7 - Specifications

| SPECIFICATION | MicroVAX II | MicroVAX I |
|---|---|---|
| Power Requirements<br>+5 Volt (amps)<br>+12 Volt (amps) | <br>6.2<br>0.14 | <br>14.0<br>0.5 |
| AC loads | 2.7 | 2 |
| DC loads | 1.0 | 1 |
| Termination (ohms) | 240 | 240 |

| uNOTE # 023 |
|---|

| Title: MicroVAX I to MicroVAX II Upgrade Issues | Date: 28-APR-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 5 |

This MicroNote describes the hardware issues in upgrading from a MicroVAX I to a MicroVAX II.

The Add-On and Upgrades group within Digital Equipment Corporation announced a formal upgrade kit at the time of the MicroVAX II announcement. For more information about this program contact a local DEC sales representative. This MicroNote is intended for those users who wish understand the issues in an upgrade.

Reference MicroNote #22 for a detailed description of the differences between the processors of the MicroVAX I and MicroVAX II systems.

Upgrading a MicroVAX I to a MicroVAX II is straightforward because the two systems share many common components. First of all the BA23 enclosure is used by both systems, this includes the backplane, power supply, control panel, and I/O patch panel. The 5 1/4" disk devices used on a MicroVAX I are also compatible with the MicroVAX II. Since I/O devices for both machines are Q-bus devices they are directly compatible with the new processor. The upgrade process does however involve replacing the processor, the memory, the 5 1/4" disk controller and the processor cabinet kit. Each of these upgrade issues is addressed in detail.

An upgrade example is shown at the end of this MicroNote.

THE PROCESSOR UPGRADE:
The MicroVAX I CPU is made up of two quad modules, the memory controller ('MCT') and the data path ('DAP'). These two modules occupy slots 1 and 2 in the backplane. Both of these modules are replaced with a single quad module, the MicroVAX II CPU which must be placed in slot 1. The MicroVAX II CPU module includes 1 MB of memory and the MicroVAX 78132 floating point unit. Slot 2 previously used by the MicroVAX I is now available for expansion memory or other options.

THE MEMORY UPGRADE:
The memory system designed for the MicroVAX II is unique to that processor, therefore the Q-bus memories used in the MicroVAX I must be removed and replaced by the new type.

If the MicroVAX I being upgraded has only 1 MB of main memory then the processor and memory can both be replaced with the MicroVAX II CPU module. Additional memory may be added to a MicroVAX II CPU if necessary. Besides the 1 MB of on-board memory, the MicroVAX II CPU will support a maximum of 2 memory expansion boards. These additional boards must be placed immediately after the processor in slots 2 and 3 of the backplane. The memory modules communicate with the CPU over the C-D interconnect of the backplane and an over-the-top cable. A cable is shipped with each memory module.

Figure 1 is an example of how the processor and memory boards are configured into a system. If the dual height memory expansion module is used (MS630-AA) it must be placed in the C-D side of the backplane. The memory modules may be placed in any order in slots 2 and 3. Table 2 lists the memory expansion options.

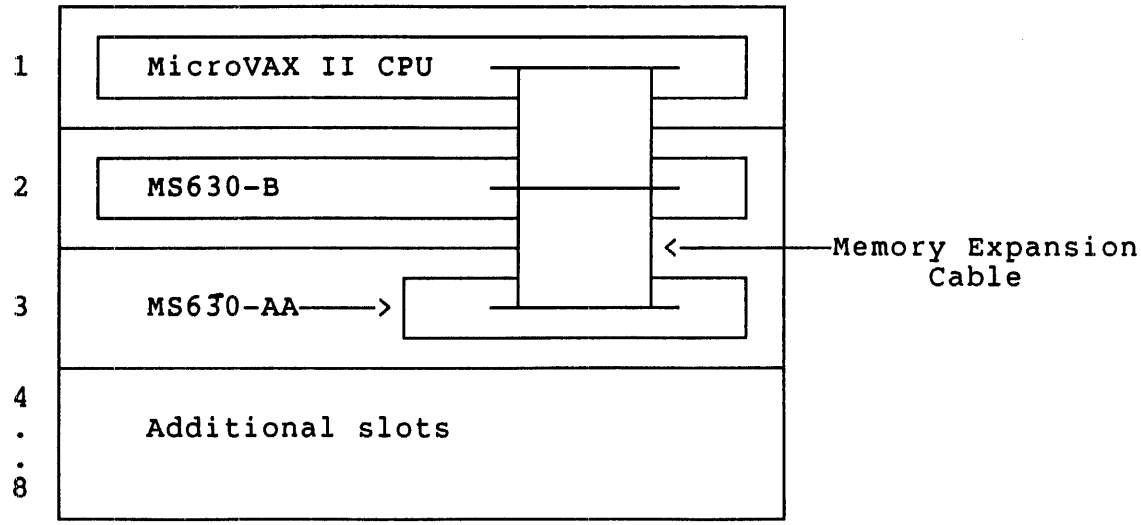
WARNING

Both the MicroVAX II CPU and MS630 memory expansion modules must be placed in backplane slots with the C-D interconnect. Damage may result if these modules are placed in backplanes with Q-bus signals on both the A/B and C/D sides.

Table 2 - MS630 Memory Expansion Modules

| MEMORY | MEMORY SIZE | FORM FACTOR |
|---|---|---|
| MS630-AA | 1 Mbyte | Dual |
| MS630-BA | 2 Mbytes | Quad |
| MS630-BB | 4 Mbytes | Quad |

Figure 1 - Processor and Memory Configuration



THE RQDX DISK CONTROLLER UPGRADE:
The RQDX1 5 1/4" disk controller used in MicroVAX I systems must be removed and replaced with the RQDX2 controller. This upgrade is necessary because the RQDX2 provides features not available on the RQDX1 and because the RQDX1 is incompatible with the MicroVAX II.

The RQDX1 does not pass the interrupt acknowledge (IACK) or DMA grant signal (DMGO) to other modules which may be located after it in the backplane. Therefore it must always be the last module in a system. This restriction has been removed with the RQDX2, which does pass both of those signals.

Both RQDX modules are capable of controlling 4 logical units. However the RQDX1 can only control a maximum of 2 fixed winchesters (RD5n type drives). Therefore the largest amount of storage that can be connected to an RQDX1 would be a combination of 2 RD5n type drives and an RX50, which counts as two logical units because it actually handles two 400KB floppies. The RQDX2 also controls 4 logical units but it knows how to control any combination of RDs and RX50s. Therefore the maximum amount of storage that can be connected to an RQDX2 would be four RD5n type drives.

The RQDX1 is compatible with RX50 floppies (800KB), RD51 (11MB) and RD52 (31MB) disk drives. The RQDX2 is compatible with those drives but is also capable of controlling RD53 (71MB) drives.

THE CABINET KIT UPGRADE:
The MicroVAX I cabinet kit is incompatible with the MicroVAX II, therefore the cabinet kit used in the MicroVAX II, BA23 based system must be used. The part number is CK-KA630-AB.

Both cabinet kits have one serial line connector in order to communicate with the VAX console device. The MicroVAX I cabinet kit uses the standard 25-pin, D-subminiature RS232 connector. The MicroVAX II cabinet kit uses a different type of connector. It is a 9-pin D-subminiature connector and will therefore require a different cable from the patch panel to the console device. The appropriate cable is part number BCC08.

OPERATING SYSTEM UPGRADE:
Once the hardware has been upgraded it is also necessary to have the appropriate system software which will support the MicroVAX II. Table 3 lists the initial versions of MicroVMS, ULTRIX-32m and VAXELN which support the MicroVAX II.

Table 3 - System Software Support

| System Software | Version with MicroVAX II Support |
|---|---|
| MicroVMS | V4.1m or later |
| ULTRIX-32m | V1.1 or later |
| VAXELN | V2.0 or later |

Many software layered products support the MicroVAX II. Consult the appropriate Software Product Description (SPD) to verify if the current version of the layered product is supported.

UPGRADE EXAMPLE:
The following example illustrates the changes made to a MicroVAX I system to upgrade it to a MicroVAX II system. The diagrams represent the modules placed in the BA23 backplane.

Both systems have a CPU, 2MB of memory, a DHV11 8-line asynchronous multiplexer, a DEQNA Ethernet controller and an RQDX 5 1/4" disk controller.

MicroVAX I System

| | |
|---|---|
| 1 | KD32 CPU - MCT |
| 2 | KD32 CPU - DAP |
| 3 | MSV11-QA (1MB) |
| 4 | MSV11-QA (1MB) |
| 5 | DHV11 |
| 6 | DEQNA \| G7272 |
| 7 | RQDX1 |
| 8 | |

MicroVAX II System

| | | |
|---|---|---|
| 1 | KA630-AA | |
| 2 | DEQNA | MS630-AA |
| 3 | RQDX2 | |
| 4 | DHV11 | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |

Comments:

The 2 modules of the MicroVAX I CPU and the two quad memory boards (MSV11-QAs) were replaced by the KA630-AA and one MS630-AA.

The DHV11 and DEQNA are compatible with both processors, therefore they were not replaced and remained in the system.

The RQDX1 controller is incompatible with the MicroVAX II, therefore it is replaced with the RQDX2. Any 5 1/4" disk (RX50, RD51 and RD52) used on the MicroVAX I is compatible with the RQDX2 on the MicroVAX II.

The RQDX1 controller must occupy the last slot in a system. The RQDX2 controller is located before the DHV11 in the upgraded system to emphasize the point that it is not required to be the last module in the system.

The MicroVAX I system required a G7272 grant continuity card in slot 6 in order to make the system work properly. This was not necessary in the upgraded MicroVAX II system because there was an empty dual-height slot next to the MS630-AA memory module. This illustrates the point that the dual height slot next to an MS630-AA is a valid Q-bus slot and can be occupied by any compatible, dual-height option.

The cabinet kits for the two systems are not shown in the diagrams, but the one used for the MicroVAX I must be replaced by the cabinet kit for the MicroVAX II, part number CK-KA630-AB. The serial line cable from the patch panel to the terminal must also change. The cable of the MicroVAX I system must be replaced by a BCC08 cable.

| Title: MicroVAX Instruction Set Differences | Date: 28-APR-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 11 |

The MicroVAX architecture specifies that the full VAX instruction set need not be implemented in the hardware of a MicroVAX processor. For those processors that fall into this category, there is a software emulator which guarantees that the instructions are still executable.

This MicroNote lists all of the instructions of the VAX architecture and for each MicroVAX processor indicates which instructions are implemented in hardware/microcode, those that are emulated and those that are present in a floating point unit.

The instructions are listed in alphabetical order by instruction mnemonic. The following designations are used to indicate where an instruction will be executed.

CPU - Instructions marked with 'CPU' are implemented in the hardware of the particular MicroVAX processor.

EMA - Instructions marked with 'EMA' are emulated with microcode assist.

E - Instructions marked with an 'E' are emulated entirely in software.

Processor specific designations:
MicroVAX II
FPU - Instructions marked with 'FPU' are implemented in the hardware only if an external floating point unit is present, otherwise they are emulated.

MicroVAX I
Hx - Instructions marked with 'H', 'HF', 'HD' or 'HG' are implemented in hardware even though the MicroVAX architecture specifies that these instructions are emulated.

There are two versions of the MicroVAX I, one with F_ and D_floating and the other with F_ and G_floating instructions in microcode. There are no MicroVAX I processors with all 3 of these floating point instruction types in the hardware. The F_floating instructions are identified by 'HF', the D_floating by 'HD' and the G_floating by 'HG'.

Reference MicroNote #21, 'Floating Point Considerations on MicroVAX I'.

The following table lists statistics about the distribution of instructions based on where they are executed. There are 304 instructions in the VAX instruction set.

Table 1 - MicroVAX Instruction Distribution

| Description | MicroVAX Architecture | MicroVAX I | MicroVAX II |
|---|---|---|---|
| Percentage executed in CPU | 57.6 | 74.3 | 57.6 |
| Percentage executed in FPU | N/A | N/A | 23.0 |
| Percentage emulated with microcode assist | 8.9 | 7.3 | 8.9 |
| Percentage emulated entirely in software | 33.5 | 18.4 | 10.5 |

Table 2 - Instruction Set Differences

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| ACBB | Add compare and branch byte | CPU | CPU | CPU |
| ACBD | Add compare and branch D_floating | E | HD | FPU |
| ACBF | Add compare and branch F_floating | E | HF | FPU |
| ACBG | Add compare and branch G_floating | E | HG | FPU |
| ACBH | Add compare and branch H_floating | E | E | E |
| ACBL | Add compare and branch longword | CPU | CPU | CPU |
| ACBW | Add compare and branch word | CPU | CPU | CPU |
| ADAWI | Add aligned word interlocked | CPU | CPU | CPU |
| ADDB2 | Add byte 2-operand | CPU | CPU | CPU |
| ADDB3 | Add byte 3-operand | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| ADDD2 | Add D_floating 2-operand | E | HD | FPU |
| ADDD3 | Add D_floating 3_operand | E | HD | FPU |
| ADDF2 | Add F_floating 2_operand | E | HF | FPU |
| ADDF3 | Add F_floating 3_operand | E | HF | FPU |
| ADDG2 | Add G_floating 2_operand | E | HG | FPU |
| ADDG3 | Add G_floating 3_operand | E | HG | FPU |
| ADDH2 | Add H_floating 2_operand | E | E | E |
| ADDH3 | Add H_floating 3-operand | E | E | E |
| ADDL2 | Add longword 2-operand | CPU | CPU | CPU |
| ADDL3 | Add longword 3-operand | CPU | CPU | CPU |
| ADDP4 | Add packed 4-operand | E | EMA | EMA |
| ADDP6 | Add packed 6-operand | E | EMA | EMA |
| ADDW2 | Add word 2-operand | CPU | CPU | CPU |
| ADDW3 | Add word 3-operand | CPU | CPU | CPU |
| ADWC | Add with carry | CPU | CPU | CPU |
| AOBLEQ | Add one and branch on less or equal | CPU | CPU | CPU |
| AOBLSS | Add one and branch on less | CPU | CPU | CPU |
| ASHL | Arithmetic shift longword | CPU | CPU | CPU |
| ASHP | Arithmetic shift and round packed | E | EMA | EMA |
| ASHQ | Arithmetic shift quad | CPU | CPU | CPU |
| BBC | Branch on bit clear | CPU | CPU | CPU |
| BBCC | Branch on bit clear and clear | CPU | CPU | CPU |
| BBCCI | Branch on bit clear and clear interlocked | CPU | CPU | CPU |
| BBCS | Branch on bit clear and set | CPU | CPU | CPU |
| BBS | Branch on bit set | CPU | CPU | CPU |
| BBSC | Branch on bit set and clear | CPU | CPU | CPU |
| BBSS | Branch on bit set and set | CPU | CPU | CPU |
| BBSSI | Branch on bit set and set interlocked | CPU | CPU | CPU |
| BCC | Branch on carry clear | CPU | CPU | CPU |
| BCS | Branch on carry set | CPU | CPU | CPU |
| BEQL | Branch on equal | CPU | CPU | CPU |
| BEQLU | (=BEQL) Branch on equal unsigned | CPU | CPU | CPU |
| BGEQ | Branch on greater or equal | CPU | CPU | CPU |
| BGEQU | (=BCC) Branch on greater or equal unsigned | CPU | CPU | CPU |
| BGTR | Branch on greater | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAX |
|---|---|---|---|---|
| BGTRU | Branch on greater unsigned | CPU | CPU | CPU |
| BICB2 | Bit clear byte 2-operand | CPU | CPU | CPU |
| BICB3 | Bit clear byte 3-operand | CPU | CPU | CPU |
| BICL2 | Bit clear longword 2-operand | CPU | CPU | CPU |
| BICL3 | Bit clear longword 3-operand | CPU | CPU | CPU |
| BICPSW | Bit clear processor status word | CPU | CPU | CPU |
| BICW2 | Bit clear word 2-operand | CPU | CPU | CPU |
| BICW3 | Bit clear word 3-operand | CPU | CPU | CPU |
| BISB2 | Bit set byte 2-operand | CPU | CPU | CPU |
| BISB3 | Bit set byte 3-operand | CPU | CPU | CPU |
| BISL2 | Bit set longword 2-operand | CPU | CPU | CPU |
| BISL3 | Bit set longword 3-operand | CPU | CPU | CPU |
| BISPSW | Bit set processor status word | CPU | CPU | CPU |
| BISW2 | Bit set word 2-operand | CPU | CPU | CPU |
| BISW3 | Bit set word 3-operand | CPU | CPU | CPU |
| BITB | Bit test byte | CPU | CPU | CPU |
| BITL | Bit test longword | CPU | CPU | CPU |
| BITW | Bit test word | CPU | CPU | CPU |
| BLBC | Branch on low bit clear | CPU | CPU | CPU |
| BLBS | Branch on low bit set | CPU | CPU | CPU |
| BLEQ | Branch on less or equal | CPU | CPU | CPU |
| BLEQU | Branch on less or equal unsigned | CPU | CPU | CPU |
| BLSS | Branch on less | CPU | CPU | CPU |
| BLSSU | (=BCS) Branch on less unsigned | CPU | CPU | CPU |
| BNEQ | Branch on not equal | CPU | CPU | CPU |
| BNEQU | (=BNEQ) Branch on not equal unsigned | CPU | CPU | CPU |
| BPT | Break point fault | CPU | CPU | CPU |
| BRB | Branch with byte displacement | CPU | CPU | CPU |
| BRW | Branch with word displacement | CPU | CPU | CPU |
| BSBB | Branch to subroutine with byte displacement | CPU | CPU | CPU |
| BSBW | Branch to subroutine with word displacement | CPU | CPU | CPU |
| BVC | Branch on overflow clear | CPU | CPU | CPU |
| BVS | Branch on overflow set | CPU | CPU | CPU |
| CALLG | Call with general argument list | CPU | CPU | CPU |
| CALLS | Call with argument list on stack | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| CASEB | Case byte | CPU | CPU | CPU |
| CASEL | Case longword | CPU | CPU | CPU |
| CASEW | Case word | CPU | CPU | CPU |
| CHME | Change mode to executive | CPU | CPU | CPU |
| CHMK | Change mode to kernel | CPU | CPU | CPU |
| CHMS | Change mode to supervisor | CPU | CPU | CPU |
| CHMU | Change mode to user | CPU | CPU | CPU |
| CLRB | Clear byte | CPU | CPU | CPU |
| CLRD (=CLRQ) | Clear D_floating | CPU | CPU | CPU |
| CLRF (=CLRL) | Clear F_floating | CPU | CPU | CPU |
| CLRG (=CLRQ) | Clear G_floating | CPU | CPU | CPU |
| CLRH (=CLRO) | Clear H_floating | CPU | CPU | CPU |
| CLRL | Clear longword | CPU | CPU | CPU |
| CLRO | Clear ocatword | E | E | E |
| CLRQ | Clear quadword | CPU | CPU | CPU |
| CLRW | Clear word | CPU | CPU | CPU |
| CMPB | Compare byte | CPU | CPU | CPU |
| CMPC3 | Compare character 3-operand | E | H | EMA |
| CMPC5 | Compare character 5-operand | E | EMA | EMA |
| CMPD | Compare D_floating | E | HD | FPU |
| CMPF | Compare F_floating | E | HF | FPU |
| CMPG | Compare G_floating | E | HG | FPU |
| CMPH | Compare H_floating | E | E | E |
| CMPL | Compare longword | CPU | CPU | CPU |
| CMPP3 | Compare packed 3-operand | E | EMA | EMA |
| CMPP4 | Compare packed 4-operand | E | EMA | EMA |
| CMPV | Compare field | CPU | CPU | CPU |
| CMPW | Compare word | CPU | CPU | CPU |
| CMPZV | Compare zero-extended field | CPU | CPU | CPU |
| CRC | Calculate cyclic redundancy check | E | EMA | EMA |
| CVTBD | Convert byte to D_floating | E | HD | FPU |
| CVTBF | Convert byte to F_floating | E | HF | FPU |
| CVTBG | Convert byte to G_floating | E | HG | FPU |
| CVTBH | Convert byte to H_floating | E | E | E |
| CVTBL | Convert byte to longword | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| CVTBW | Convert byte to word | CPU | CPU | CPU |
| CVTDB | Convert D_floating to byte | E | HD | FPU |
| CVTDF | Convert D_floating to F_floating | E | HD | FPU |
| CVTDH | Convert D_floating to H_floating | E | E | E |
| CVTDL | Convert D_floating to longword | E | HD | FPU |
| CVTDW | Convert D_floating to word | E | HD | FPU |
| CVTFB | Convert F_floating to byte | E | HF | FPU |
| CVTFD | Convert F_floating to D_floating | E | HF | FPU |
| CVTFG | Convert F_floating to G_floating | E | HG | FPU |
| CVTFH | Convert F_floating to H_floating | E | E | E |
| CVTFL | Convert F_floating to longword | E | HF | FPU |
| CVTFW | Convert F_floating to word | E | HF | FPU |
| CVTGB | Convert G_floating to byte | E | HG | FPU |
| CVTGF | Convert G_floating to F_floating | E | HG | FPU |
| CVTGH | Convert G_floating to H_floating | E | E | E |
| CVTGL | Convert G_floating to longword | E | HG | FPU |
| CVTGW | Convert G_floating to word | E | HG | FPU |
| CVTHB | Convert H_floating to byte | E | E | E |
| CVTHD | Convert H_floating to D_floating | E | E | E |
| CVTHF | Convert H_floating to F_floating | E | E | E |
| CVTHG | Convert H_floating to G_floating | E | E | E |
| CVTHL | Convert H_floating to longword | E | E | E |
| CVTHW | Convert H_floating to word | E | E | E |
| CVTLB | Convert longword to byte | CPU | CPU | CPU |
| CVTLD | Convert longword to D_floating | E | HD | FPU |
| CVTLF | Convert longword to F_floating | E | HF | FPU |
| CVTLG | Convert longword to G_floating | E | HG | FPU |
| CVTLH | Convert longword to H_floating | E | E | E |
| CVTLP | Convert longword to packed | E | EMA | EMA |
| CVTLW | Convert longword to word | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| CVTPL | Convert packed to longword | E | EMA | EMA |
| CVTPS | Convert packed to leading separate | E | EMA | EMA |
| CVTPT | Convert packed to trailing | E | EMA | EMA |
| CVTRDL | Convert rounded D_floating to longword | E | HD | FPU |
| CVTRFL | Convert rounded F_floating to longword | E | HF | FPU |
| CVTRGL | Convert rounded G_floating to longword | E | HG | FPU |
| CVTRHL | Convert rounded H_floating to longword | E | E | E |
| CVTSP | Convert leading separate to packed | E | EMA | EMA |
| CVTTP | Convert trailing to packed | E | EMA | EMA |
| CVTWB | Convert word to byte | CPU | CPU | CPU |
| CVTWD | Convert word to D_floating | E | HD | FPU |
| CVTWF | Convert word to F_floating | E | HF | FPU |
| CVTWG | Convert word to G_floating | E | HG | FPU |
| CVTWH | Convert word to H_floating | E | E | E |
| CVTWL | Convert word to longword | CPU | CPU | CPU |
| DECB | Decrement byte | CPU | CPU | CPU |
| DECL | Decrement longword | CPU | CPU | CPU |
| DECW | Decrement word | CPU | CPU | CPU |
| DIVB2 | Divide byte 2-operand | CPU | CPU | CPU |
| DIVB3 | Divide byte 3-operand | CPU | CPU | CPU |
| DIVD2 | Divide D_floating 2-operand | E | HD | FPU |
| DIVD3 | Divide D_floating 3-operand | E | HD | FPU |
| DIVF2 | Divide F_floating 2-operand | E | HF | FPU |
| DIVF3 | Divide F_floating 3-operand | E | HF | FPU |
| DIVG2 | Divide G_floating 2-operand | E | HG | FPU |
| DIVG3 | Divide G_floating 3-operand | E | HG | FPU |
| DIVH2 | Divide H-floating 2-operand | E | E | E |
| DIVH3 | Divide H-floating 3-operand | E | E | E |
| DIVL2 | Divide longword 2-operand | CPU | CPU | CPU |
| DIVL3 | Divide longword 3-operand | CPU | CPU | CPU |
| DIVP | Divide packed | E | EMA | EMA |
| DIVW2 | Divide word 2-operand | CPU | CPU | CPU |
| DIVW3 | Divide word 3-operand | CPU | CPU | CPU |
| EDITPC | Edit packed to character string | E | EMA | EMA |
| EDIV | Extended divide | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXI |
|---|---|---|---|---|
| EMODD | Extended modulus D_floating | E | HD | FPU |
| EMODF | Extended modulus F_floating | E | HF | FPU |
| EMODG | Extended modulus G_floating | E | HG | FPU |
| EMODH | Extended modulus H_floating | E | E | E |
| EMUL | Extended multiply | CPU | CPU | CPU |
| EXTV | Extract field | CPU | CPU | CPU |
| EXTZV | Extract zero_extended field | CPU | CPU | CPU |
| FFC | Find first clear bit | CPU | CPU | CPU |
| FFS | Find first set bit | CPU | CPU | CPU |
| HALT | Halt (kernel mode only) | CPU | CPU | CPU |
| INCB | Increment byte | CPU | CPU | CPU |
| INCL | Increment longword | CPU | CPU | CPU |
| INCW | Increment word | CPU | CPU | CPU |
| INDEX | Index calculation | CPU | CPU | CPU |
| INSQHI | Insert at head of queue, interlocked | CPU | CPU | CPU |
| INSQTI | Insert at tail of queue, interlocked | CPU | CPU | CPU |
| INSQUE | Insert into queue | CPU | CPU | CPU |
| INSV | Insert field | CPU | CPU | CPU |
| JMP | Jump | CPU | CPU | CPU |
| JSB | Jump to subroutine | CPU | CPU | CPU |
| LDPCTX | Load process context (only legal on interrupt stack) | CPU | CPU | CPU |
| LOCC | Locate character | E | H | EMA |
| MATCHC | Match characters | E | EMA | EMA |
| MCOMB | Move complemented byte | CPU | CPU | CPU |
| MCOML | Move complemented longword | CPU | CPU | CPU |
| MCOMW | Move complemented word | CPU | CPU | CPU |
| MFPR | Move from processor register (kernel mode only) | CPU | CPU | CPU |
| MNEGB | Move negated byte | CPU | CPU | CPU |
| MNEGD | Move negated D_floating | E | HD | FPU |
| MNEGF | Move negated F_floating | E | HF | FPU |
| MNEGG | Move negated G_floating | E | HG | FPU |
| MNEGH | Move negated H_floating | E | E | E |
| MNEGL | Move negated longword | CPU | CPU | CPU |
| MNEGW | Move negated word | CPU | CPU | CPU |
| MOVAB | Move address of byte | CPU | CPU | CPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| MOVAD (=MOVAQ) | Move address of D_floating | CPU | CPU | CPU |
| MOVAF (=MOVAL) | Move address of F_floating | CPU | CPU | CPU |
| MOVAG (=MOVAQ) | Move address of G_floating | CPU | CPU | CPU |
| MOVAH (=MOVAO) | Move address of H_floating | CPU | CPU | CPU |
| MOVAL | Move address of longword | CPU | CPU | CPU |
| MOVAO | Move address of ocatword | E | E | E |
| MOVAQ | Move address of quadword | CPU | CPU | CPU |
| MOVAW | Move address of word | CPU | CPU | CPU |
| MOVB | Move byte | CPU | CPU | CPU |
| MOVC3 | Move character 3-operand | CPU | CPU | CPU |
| MOVC5 | Move character 5-operand | CPU | CPU | CPU |
| MOVD | Move D_floating | E | HD | FPU |
| MOVF | Move F_floating | E | HF | FPU |
| MOVG | Move G_floating | E | HG | FPU |
| MOVH | Move H_floating | E | E | E |
| MOVL | Move longword | CPU | CPU | CPU |
| MOVO | Move octaword | E | E | E |
| MOVP | Move packed | E | EMA | EMA |
| MOVPSL | Move processor status longword | CPU | CPU | CPU |
| MOVQ | Move quadword | CPU | CPU | CPU |
| MOVTC | Move translated characters | E | EMA | EMA |
| MOVTUC | Move translated until character | E | EMA | EMA |
| MOVW | Move word | CPU | CPU | CPU |
| MOVZBL | Move zero-extended byte to longword | CPU | CPU | CPU |
| MOVZBW | Move zero-extended byte to word | CPU | CPU | CPU |
| MOVZWL | Move zero-extended word to longword | CPU | CPU | CPU |
| MTPR | Move to processor register (kernel mode only) | CPU | CPU | CPU |
| MULB2 | Multiply byte 2-operand | CPU | CPU | CPU |
| MULB3 | Multiply byte 3-operand | CPU | CPU | CPU |
| MULD2 | Multiply D_floating 2-operand | E | HD | FPU |
| MULD3 | Multiply D_floating 3-operand | E | HD | FPU |
| MULF2 | Multiply F_floating 2-operand | E | HF | FPU |
| MULF3 | Multiply F_floating 3-operand | E | HF | FPU |
| MULG2 | Multiply G_floating 2-operand | E | HG | FPU |
| MULG3 | Multiply G_floating 3-operand | E | HG | FPU |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXI |
|---|---|---|---|---|
| MULH2 | Multiply H_floating 2-operand | E | E | E |
| MULH3 | Multiply H_floating 3-operand | E | E | E |
| MULL2 | Multiply longword 2-operand | CPU | CPU | CPU |
| MULL3 | Multiply longword 3-operand | CPU | CPU | CPU |
| MULP | Multiply packed | E | EMA | EMA |
| MULW2 | Multiply word 2-operand | CPU | CPU | CPU |
| MULW3 | Multiply word 3-operand | CPU | CPU | CPU |
| NOP | No operation | CPU | CPU | CPU |
| POLYD | Evaluate polynomial D_floating | E | HD | FPU |
| POLYF | Evaluate polynomial F_floating | E | HF | FPU |
| POLYG | Evaluate polynomial G_floating | E | HG | FPU |
| POLYH | Evaluate polynomial H_floating | E | E | E |
| POPR | Pop registers | CPU | CPU | CPU |
| PROBER | Probe read access | CPU | CPU | CPU |
| PROBEW | Probe write access | CPU | CPU | CPU |
| PUSHAB | Push address of byte | CPU | CPU | CPU |
| PUSHAD | (=PUSHAQ) Push address of D_floating | CPU | CPU | CPU |
| PUSHAF | (=PUSHAL) Push address of F_floating | CPU | CPU | CPU |
| PUSHAG | (=PUSHAQ) Push address of G_floating | CPU | CPU | CPU |
| PUSHAH | (=PUSHAO) Push address of H_floating | CPU | CPU | CPU |
| PUSHAL | Push address of longword | CPU | CPU | CPU |
| PUSHAO | Push address of ocatword | E | E | E |
| PUSHAQ | Push address of quadword | CPU | CPU | CPU |
| PUSHAW | Push address of word | CPU | CPU | CPU |
| PUSHL | Push longword | CPU | CPU | CPU |
| PUSHR | Push registers | CPU | CPU | CPU |
| REI | Return from exception or interrupt | CPU | CPU | CPU |
| REMQHI | Remove from head of queue, interlocked | CPU | CPU | CPU |
| REMQTI | Remove from tail of queue, interlocked | CPU | CPU | CPU |
| REMQUE | Remove from queue | CPU | CPU | CPU |
| RET | Return from procedure | CPU | CPU | CPU |
| ROTL | Rotate longword | CPU | CPU | CPU |
| RSB | Return from subroutine | CPU | CPU | CPU |
| SBWC | Subtract with carry | CPU | CPU | CPU |
| SCANC | Scan for character | E | H | EMA |

| Mnemonic | Description | MicroVAX Architecture | MicroVAXI | MicroVAXII |
|---|---|---|---|---|
| SKPC | Skip character | E | H | EMA |
| SOBGEQ | Subtract one and branch on greater or equal | CPU | CPU | CPU |
| SOBGTR | Subtract one and branch on greater | CPU | CPU | CPU |
| SPANC | Span characters | E | H | EMA |
| SUBB2 | Subtract byte 2-operand | CPU | CPU | CPU |
| SUBB3 | Subtract byte 3-operand | CPU | CPU | CPU |
| SUBD2 | Subtract D_floating 2-operand | E | HD | FPU |
| SUBD3 | Subtract D_floating 3-operand | E | HD | FPU |
| SUBF2 | Subtract F_floating 2-operand | E | HF | FPU |
| SUBF3 | Subtract F_floating 3-operand | E | HF | FPU |
| SUBG2 | Subtract G_floating 2-operand | E | HG | FPU |
| SUBG3 | Subtract G_floating 3-operand | E | HG | FPU |
| SUBH2 | Subtract H_floating 2-operand | E | E | E |
| SUBH3 | Subtract H_floating 3-operand | E | E | E |
| SUBL2 | Subtract longword 2-operand | CPU | CPU | CPU |
| SUBL3 | Subtract longword 3-operand | CPU | CPU | CPU |
| SUBP4 | Subtract packed 4-operand | E | EMA | EMA |
| SUBP6 | Subtract packed 6-operand | E | EMA | EMA |
| SUBW2 | Subtract word 2-operand | CPU | CPU | CPU |
| SUBW3 | Subtract word 3-operand | CPU | CPU | CPU |
| SVPCTX | Save process context (kernel mode only) | CPU | CPU | CPU |
| TSTB | Test byte | CPU | CPU | CPU |
| TSTD | Test D_floating | E | HD | FPU |
| TSTF | Test F_floating | E | HF | FPU |
| TSTG | Test G_floating | E | HG | FPU |
| TSTH | Test H_floating | E | E | E |
| TSTL | Test longword | CPU | CPU | CPU |
| TSTW | Test word | CPU | CPU | CPU |
| XFC | Extended function call | CPU | CPU | CPU |
| XORB2 | Exclusive OR byte 2-operand | CPU | CPU | CPU |
| XORB3 | Exclusive OR byte 3-operand | CPU | CPU | CPU |
| XORL2 | Exclusive OR longword 2-operand | CPU | CPU | CPU |
| XORL3 | Exclusive OR longword 3-operand | CPU | CPU | CPU |
| XORW2 | Exclusive OR word 2-operand | CPU | CPU | CPU |
| XORW3 | Exclusive OR word 3-operand | CPU | CPU | CPU |

| uNOTE # 025 |
|---|

| Title: FPJ11-AA Compatibility with the LSI-11/73 (KDJ11-A) | Date: 28-APR-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 2 |

Early LSI-11/73 (KDJ11-A) modules are incompatible with the FPJ11-AA floating point accelerator. This MicroNote describes how to identify those modules which are compatible and those which are not compatible with the FPJ11-AA.

The following information refers to two identifying numbers. They are the module variation and the module revision.

The module variation number is stamped onto the end of the plastic handles. It will be 'M8192', 'M8192-YB' or 'M8192-YC'.

The module revision number can be found on side two of the module (noncomponent side) stamped into the plastic handle, or written with indelible ink. This number is updated as the board is ECO'd. The designation is a number followed by the module revision. e.g. 441 C1, indicates module revision C1.

The first LSI-11/73 module is called a KDJ11-AA and is incompatible with the FPJ11-AA. The module variation for the KDJ11-AA is M8192, and the module revision number is C1. The KDJ11-AA can be upgraded to be FPJ11-AA compatible. Contact the local DEC office for more information concerning the upgrade procedure.

Two new options have been created. The KDJ11-AB is a module which is fully compatible with the FPJ11-AA. The module variation for the KDJ11-AB is M8192-YB, and the module revision number is A1 or A2.

The third option is called a KDJ11-AC and is used for fully compatible modules which have FPJ11-AAs installed onto the board by Digital. It has a module variation of M8192-YC and the module revision number is A1 or A2. Therefore a KDJ11-AB becomes a KDJ11-AC by simply installing an FPJ11-AA.

The following table summarizes the above information and should be used as a 'quick check' for FPJ11-AA compatibility.

| OPTION | MODULE VARIATION | MODULE REVISION | FPJ11 COMPATIBLE? |
|---|---|---|---|
| KDJ11-AA | M8192 | C1 | NO (But Upgradeable) |
| KDJ11-AB | M8192-YB | A1 or A2 | YES |
| KDJ11-AC | M8192-YC | A1 or A2 | YES (FPJ11-AA Installed) |

CAUTION

An FPJ11-AA installed on a KDJ11-AA may APPEAR to work but cache errors and parity errors may occur.

An MXV11-BF must be REV C to be compatible with a KDJ11-A with an FPJ11-AA (KDJ11-AC). Earlier versions of the MXV11-BF are incompatible with KDJ11-ACs. KDJ11-As without FPJ11-AAs are compatible with all versions of the MXV11-BF.

The MCV11-D is incompatible with KDJ11-ACs. KDJ11-As without FPJ11-AAs are compatible with all versions of the MCV11-D.

| Title: The MicroVAX II CPU Multicomputing Capability | Date: 28-JUN-85 |
|---|---|
| Originator: Mike Collins | Page 1 of 14 |

The MicroVAX II CPU may be configured as an arbiter CPU or as one of three auxiliary CPU's. Therefore it is possible to configure a Q-bus system with multiple MicroVAX II CPUs. This 'multicomputing' capability is the topic of this MicroNote.

## Multicomputing Introduction

The multicomputing capability is a hardware design feature of the MicroVAX II. It allows a maximum of four MicroVAX II CPUs to reside on the same Q-bus. Figure 1 below is a block diagram which shows how 2 of a possible 4 CPUs may be configured on the bus. One of the processors will be the arbiter and the others will be auxiliaries.

NOTE

> There is no shared memory between processors, therefore this system should not be considered as symmetrical multiprocessing.

Each processor may use expansion memory modules provided that there are sufficient Q/CD slots in the backplane. See the configuration section for more detail on the configuration rules.

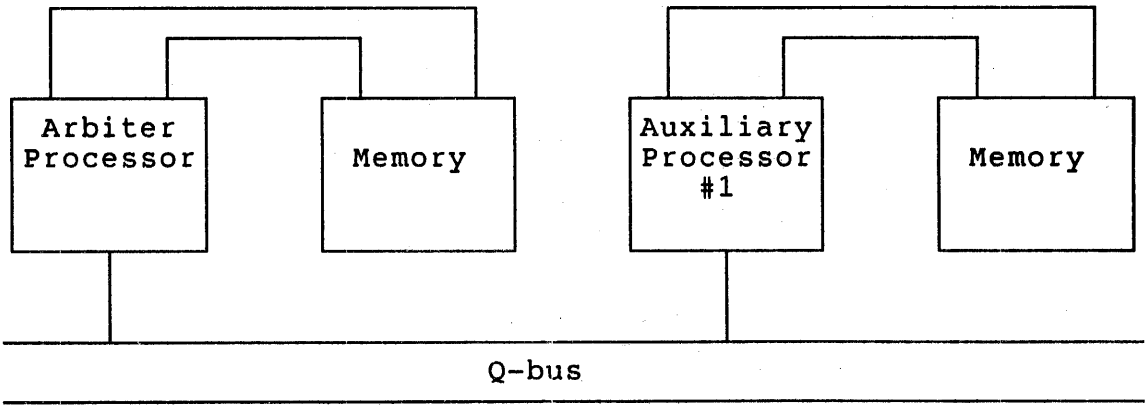



Figure 1 - Multicomputing Configuration

CAUTION

Digital's 32-bit operating systems DO NOT support the multicomputing feature. Users who wish to customize their systems to take advantage of the multicomputing capability may do so but are responsible for designing the software system.

The following topics which are pertinent to the multicomputing design will be described in detail:

1. MicroVAX II Memory System
2. Interprocessor Communications Register
3. Arbiter/auxiliary Processor Differences
4. Bootstrapping an Auxiliary Processor
5. Configuration Rules

REFERENCES

The MicroVAX 630 CPU Module User's Guide (P/N EK-KA630-UG) and MicroNote #22 have more detailed information on all aspects of the MicroVAX II CPU and memory system.

## MicroVAX II Memory System

Before describing the details of the multicomputing feature it is necessary to understand the memory system of a MicroVAX II.

The memory system of the MicroVAX II is unique. Unlike previous Q-bus processors, the memory modules do not communicate to the CPU over the Q-bus. Instead, the communication and transfers occur over a dedicated, closely coupled interconnect. This design allows for very fast reads and writes to memory (400 nsec). Not only are the transfers very fast, but none of this CPU-memory activity appears on the Q-bus. Therefore the Q-bus is strictly for I/O.

Since memory is not directly on the Q-bus it is necessary to allow DMA devices on the Q-bus to access memory. This feature is called the Q-bus I/O map. It is a mechanism which maps Q-bus addresses (22-bit physical address) to the local memory system of a MicroVAX II (24-bit physical address). This same concept is used on larger UNIBUS based VAXes.

When one or more auxiliary processors are added to a system, remember that each has its own local memory system. Therefore it is intended that they will each operate out of their own memory. This design is

well suited for applications which can be easily partitioned between multiple processors such that only messages are passed between them.

It would be possible for a processor to operate out of memory on the Q-bus but this would be unconventional from a systems standpoint and would decrease system performance significantly. Digital's operating systems do not support the use of standard Q-bus memories in a MicroVAX II.

With only one processor on the Q-bus, the system software has complete control over DMA transfers. The software controls the contents of the Q-bus map registers. First, it determines whether the register is enabled and second, if it is enabled, where the DMA transfers are directed in the local memory system.

In a multiple processor configuration (since each has its own set of Q-bus I/O map registers) it is now possible for multiple map registers to respond to a single Q-bus address. It is the responsibility of the system software running on each processor to cooperate and assure that one and only one map register will respond to any Q-bus address. Otherwise multiple memory locations may respond and the results are indeterminate (i.e. multiple BRPLYs on the Q-bus for a single address).

A similar situation occurs if standard Q-bus memories are added to a MicroVAX II system. The system software is again responsible for assuring that for any address on the Q-bus which is 'shared' by a Q-bus memory board and a Q-bus I/O map register, the map register is disabled to allow only a Q-bus memory reply.

Q-bus memories would not be added to 'typical' systems. However there are some special applications which may require this capability. For example, a graphics system where the Q-bus memory is the bit map of a graphics display.

On power-up, the MicroVAX II boot ROM will check for the presence of Q-bus memory and clear the valid bits of the corresponding mapping registers. This will prevent the above situation from occurring as long as the system software does not alter the state of these valid bits at a later time.

## Interprocessor Communications Register

The interprocessor communications register (ICR) is the primary mechanism of the MicroVAX II CPU which enables multiple processors to cooperate and reside on the same Q-bus. Figure 3 describes the ICR and the bit definitions.

The ICR provides the following four functions:

1. Any processor on the bus may interrupt another without using the Q-bus interrupt lines. This is done by setting the appropriate bit in the ICR of the second processor. The CPU will then interrupt at IPL 14 (HEX) with an interrupt vector of 204 (HEX).

2. The ICR also has a bit which controls external access to local memory via the Q-bus map. When set this bit allows external devices to access the local memory system. When reset, this bit prevents the local memory system from responding to any Q-bus address references.

3. If a processor is an auxiliary then the ICR provides a mechanism which allows it to be halted by other CPUs. This feature is used at power-up to coordinate the bootstrap process.

4. Parity errors are identified when one occurs during accesses to the MicroVAX II's local memory.

The address for the ICR is located in the Q-bus I/O page address space and may be accessed by any device which can become Q-bus master. The ICR is byte-addressable.

Since it is possible to put a maximum of 4 MicroVAX II CPUs on one Q-bus they each have their own unique ICR. Table 1 lists each processor and the associated ICR address.

Table 1 - Interprocessor Communication Register Address Assignments

| Processor | 22-bit Octal Address | 32-bit Hexadecimal Address |
|---|---|---|
| Arbiter | 17 777 500 | 2000 1F40 |
| Auxiliary #1 | 17 777 502 | 2000 1F42 |
| Auxiliary #2 | 17 777 504 | 2000 1F44 |
| Auxiliary #3 | 17 777 506 | 2000 1F46 |

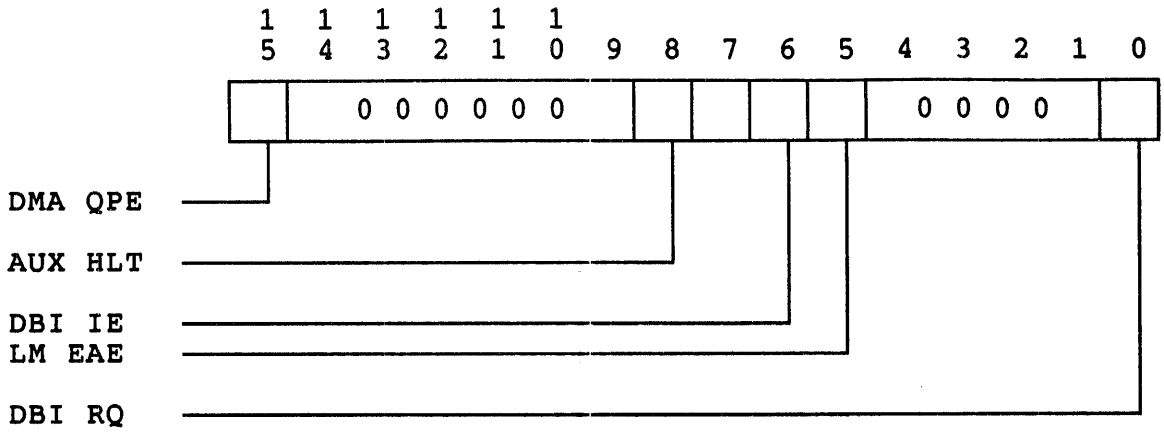



Figure 3 - Interprocessor Communications Register

| Bit(s) | Mnemonic | Meaning |
|---|---|---|
| <15> | DMA QPE | DMA Q22-Bus Address Space Parity Error. This read-only bit is set if Memory System Error Register bit <04> (DMA QPE) is set. The DMA QPE bit indicates that a parity error occurred when an external device (or CPU) was accessing the MicroVAX II CPU local memory. |
| <14:09> | - | Unused. Read as zeros. |
| <08> | AUX HLT | Auxiliary Halt. On an auxiliary MicroVAX, AUX HLT is a read-write bit. When set, typically by the arbiter CPU, it causes the on-board CPU to transfer program control to the Halt Mode ROM Code. On an arbiter MicroVAX II, AUX HLT is a read-only bit which always reads as zero. It has no effect on arbiter CPU operation. |
| <07> | - | Unused. Read as zero. |
| <06> | DBI IE | Doorbell Interrupt Enable. This bit, when set, enables interprocessor doorbell interrupt requests via ICR <00>. When the on-board CPU is Q22-Bus master, DBI IE is a read-write bit. When an external device (or CPU) is bus master, DBI IE is a read-only bit. DBI IE is cleared by power up, by the negation of DCOK and by writes to the Bus Initialize Register. |
| <05> | LM EAE | Local Memory External Access Enable. This bit, when set, enables external access to |

| | | |
|---|---|---|
| | | local memory (via the Q22-Bus map). When the on-board CPU is Q22-Bus master, LM EAE is a read-write bit. When an external device (or CPU) is bus master, LM EAE is a read-only bit. LM EAE is cleared by power up, by the negation of DCOK and by writes to the Bus Initialize Register. |
| <04:01> | - | Unused. Read as zeros. |
| <00> | DBI RQ | Doorbell Interrupt Request. If ICR <06> (DBI IE) is set, writing a "1" to DBI RQ sets DBI RQ, thus requesting a doorbell interrupt. If ICR <06> is clear, writing a "1" to DBI RQ has no effect. Writing a "0" to DBI RQ has no effect. DBI RQ is cleared when the CPU grants the doorbell interrupt request. DBI RQ is held clear whenever DBI IE is clear. |

When a processor is interrupted via its ICR the interrupt vector is 204 (Hex). This vector is used when the interrupting device is the arbiter, auxiliary #1, auxiliary #2, auxiliary #3, or any device which may become Q-bus master. Therefore it is the responsibility of the interrupt service routine to determine which device caused the interrupt since the same vector is used no matter what device set the DBI RQ bit. This interrupt occurs at the same interrupt priority level (IPL) as IRQ4 on the Q-bus.

NOTE

Following such an interrupt, the MicroVAX II sets the IPL to 14 (Hex). This is different from what happens after a standard Q-bus interrupt. When a Q-bus interrupt occurs on IRQ4, the processor raises the IPL to 17 (Hex) and it is the responsibility of the interrupt service routine to lower the IPL later on.

## Arbiter/Auxiliary Processor Differences

There are several differences between arbiter and auxiliary processors. It is important to understand these differences when configuring multiple processors into a system.

1. The arbiter MicroVAX II arbitrates bus mastership in accordance with the Q22-Bus DMA protocol. The arbitration logic is disabled on an auxiliary MicroVAX II.

2. Both the arbiter and auxiliary MicroVAX II request bus mastership via the Q22-Bus DMA Request protocol.

   a. They both assert BDMR on the Q22-Bus.

   b. The arbiter MicroVAX II receives DMGI from its arbitration logic. But an auxiliary receives DMGI from its Q22-Bus BDMGI pin.

   c. Only the auxiliary MicroVAX II actually asserts BSACK on the Q22-Bus.

3. The arbiter MicroVAX II asserts the Q22-Bus BINIT signal when DCOK is negated and when its CPU software writes to its Bus Initialize Register. An auxiliary MicroVAX II never asserts Q22-Bus BINIT, but receives BINIT and uses it to initialize the MicroVAX chip and to clear all internal registers which are cleared by the negation of DCOK.

4. The physical address of the Interprocessor Communication Register is different for each of the four MicroVAX II arbiter/auxiliary configurations.

5. An auxiliary MicroVAX II can be halted by setting bit <08> (AUX HLT) of its Interprocessor Communication Register. On an arbiter MicroVAX II this feature is disabled and AUX HLT is a read-only bit which always reads as zero.

6. The CPU halts are controlled by the external connector HLT ENB input. However, the external halts which are affected differ somewhat for the arbiter and auxiliary MicroVAX II modules. If the HLT ENB signal is asserted (low) the a MicroVAX II CPU will halt and enter the console program if:

   a. The program executes a halt instruction in kernel mode.

   b. The console detects a break character.

   c. Arbiter CPU only - the Q-bus halt line is asserted.

   d. Auxiliary CPU only - the Interprocessor Communication Register AUX HLT bit is set.

   If the HLT ENB signal is negated then:

   a. The halt line and break character are ignored and the ROM program responds to a halt instruction by restarting or rebooting the system.

b. If the MicroVAX CPU is an auxiliary, the ROM program responds to the assertion of the ICR AUX HLT bit by rebooting.

The state of the HLT ENB bit can be read by software via the Boot and Diagnostic Register.

7. Each arbiter or auxiliary MicroVAX II module can field interrupt requests from its interval timer, from its console device, and from its interprocessor doorbell. Only the arbiter MicroVAX II can field interrupts from Q22-Bus interrupt request lines IRQ7-4.

8. The arbiter asserts BIAKO on the Q22-Bus when it responds to a Q22-Bus interrupt request. An auxiliary asserts BIAKO on the Q22-Bus when it receives the assertion of BIAKI in order to pass it through to devices after it.

9. Although both the arbiter and auxiliary MicroVAX II modules contain the same time-of-year clock and battery back-up circuitry, it is assumed that the auxiliary will be configured without batteries and that its clock will never actually be enabled. This configuration will ensure a single time base for the system. If an auxiliary needs to set its time-of-year clock it can do so by referencing the arbiter's clock.

10. An arbiter processor can bootstrap from a variety of devices but an auxiliary will always boot using the ROM bootstrap protocol. See the following section for a detailed description of the bootstrap process.

## Bootstrapping an Auxiliary Processor

Since there are distinct differences between the operation and capabilities of arbiter and auxiliary processors, it is necessary to bootstrap them differently. An arbiter processor bootstraps in the conventional manner from one of several devices but an auxiliary boots using only one method.

On power-up both processors initialize themselves and perform some self tests. At this point the primary bootstrap (VMB) begins to execute. An arbiter processor may bootstrap from any one of the following devices:

1. DU type device (RX50, RDxx, RC25 or RAxx)
2. TK50 tape
3. ROM bootstrap protocol
4. Ethernet (DEQNA)

An auxiliary processor will not attempt to boot from disk, tape or ethernet. It will always boot via the ROM bootstrap protocol, which is described below.

In order to synchronize the bootstrap process, after power-up and initialization an auxiliary processor will not boot until it is allowed to do so by the arbiter. The controlling device is not required to be the arbiter, it could be another auxiliary or DMA device which is bus master, but it is the most logical point of control.

The following steps summarize the bootstrap procedure for a system with an arbiter and one (or more) auxiliaries:

1. Both types of processors initialize themselves after power-up.
2. Self-diagnostics execute to check out major sections of the CPU.
3. The arbiter boots from one of the four device types listed above.
    a. While the arbiter is booting the auxiliary completes its diagnostics and waits to start its bootstrap process.
    b. The auxiliary clears the valid bit in each of its Q-bus map registers to prevent accidental transfers to or from its local memory.
    c. The auxiliary loops doing nothing until the AUX HLT bit in its ICR is cleared. When this bit is finally cleared, the auxiliary boots via the ROM bootstrap protocol.
4. Somewhere in the system the appropriate data structure has been created to allow the auxiliary to boot via the ROM bootstrap protocol. This can be done several ways:

a. The bootstrap code is actually in ROM on an MRV11-D module.

b. The bootstrap code is loaded into non-volatile RAM.

c. The arbiter CPU loads the bootstrap code into its own local memory then sets up some Q-bus map registers so that this data can be seen by the auxiliary CPU over the Q-bus. This method is the most flexible since the bootstrap code can be changed easily.

5. When the arbiter is ready and knows auxiliary boot code is available, it allows the auxiliary CPU to bootstrap by clearing the AUX HLT bit in its ICR.

ROM Bootstrap Protocol.
To locate a PROM bootstrap, VMB searches the Q22-bus address range from high to low addresses by page (512 bytes per page) looking for readable memory. If the first six longwords of any such page contains a valid PROM "signature block" (see figure 4), VMB passes control directly to the bootstrap code in the PROM, it does not copy the PROM code to local memory for execution as it does for all other secondary bootstraps.

Note that while defined as an MRV11 PROM or equivalent bootstrap, VMB does not actually require that the signature block or the bootstrap code be in PROM. It could be in ROM, nonvolatile RAM or it could be loaded into another MicroVAX II's RAM and mapped to the Q22-bus thus enabling an auxiliary to see it.

RB:

| | | | | |
|---|---|---|---|---|
| + 0: | check byte | any value | 0 | 18 (hex) |
| + 4: | any value | | 1 | 0 |
| + 8: | size of PROM in pages | | | |
| + 12: | must be zero | | | |
| + 16: | offset into PROM to start execution | | | |
| + 20: | sum of previous three longwords | | | |

Figure 4: PROM Bootstrap Memory Format

RB + 0: This byte must be 18 (hex).

RB + 1: This byte must be 0.

RB + 2: This byte may be any value.

| | |
|---|---|
| RB + 3: | This byte must be the ones complement of the sum of the previous three bytes. |
| RB + 4: | This byte must be zero. |
| RB + 5: | This byte must be 1. |
| RB + 6: | These two bytes may be any value. |
| RB + 8: | This longword contains the size in pages of the PROM. |
| RB + 12: | This longword must be zero. |
| RB + 16: | This longword contains the byte offset into the PROM where execution is to begin. |
| RB + 20: | This entry is a longword containing the sum of the previous three longwords. |

## Configuration Rules

In order to configure multiple MicroVAX II CPUs onto one bus, 4 issues must be addressed:

1. Q-bus slot requirements for the CPU and MS630 memory boards.
2. Compatible enclosures or boxes.
3. Bus termination.
4. Using a PDP-11 CPU as the Arbiter.

1. Q-bus slot requirements.

To operate properly both the MicroVAX II CPU and all versions of the MS630 memory modules MUST be inserted in Q-bus slots which feature Q-bus signals on the A/B side and the CD interconnect on the C/D side. The MicroVAX II CPU and any expansion memory modules must all be adjacent in the backplane.

WARNING

The MicroVAX II CPU WILL be damaged if inserted into a slot which has Q-bus signals on the C/D side.

2. Compatible enclosures or boxes.

Since Q/CD slots are required, there are only 4 enclosures which are compatible with these modules. They are:

a. BA23 (8 slots total, first 3 are Q/CD)

b. BA123 (12 slots total, first 4 are Q/CD)

c. BA11-S (9 slots total, all 9 are Q/CD)

d. BA11-N (9 slots total, all 9 are Q/CD) This enclosure is electrically compatible but since it is older it is only 18-bit compatible and has not been tested for FCC compliance in a system package. Since the BA11-S is the 22-bit, updated version, the BA11-N enclosure is not recommended.

3. Bus termination.

Each MicroVAX II processor has 240 ohm termination. When multiple CPUs are placed into one system, take care to abide by the Q-bus specification regarding termination and configuring multiple box systems.

REFERENCE

Reference MicroNote #29, entitled 'Q-bus Expansion Concepts' and appendix A of the Microsystems Handbook (P/N EB-26085-41) for more information concerning Q-bus configuration rules.

The BA11-S enclosures are best suited for multiple CPUs because they have backplanes with the Q22/CD configuration in all slots. Therefore multiple CPUs and expansion memory can be accommodated most easily. The other potential boxes, the BA23 and BA123, are not as flexible because they only have 3 and 4 Q22/CD slots respectively.

When using multiple MicroVAX II CPUs in one system the goal should be to configure each backplane with 120 ohm termination if the number of backplanes used is one or two. In the three backplane case, only the first and third backplanes should have 120 ohm termination. This implies that no MicroVAX II CPUs should be configured into the middle backplane of a three backplane system (three backplanes is the maximum number allowed for the Q-bus).

The following configurations assume the BA11-S enclosures will be used.

Two CPU System:

A two CPU system is the easiest case to configure. As mentioned earlier, each CPU has 240 ohm termination. With two CPUs in the same

backplane, the proper overall termination of 120 ohms is achieved. The first CPU has 240 ohms in parallel with 240 ohms on the second CPU which is 120 ohms, assuming there is no termination on the backplane (which is true for the BA11-S enclosure).

Three CPU System:

A three CPU system should be configured in 2 backplanes to maintain 120 ohm termination in each. The first 2 CPUs should be configured in the first box which terminates that backplane properly as in the case above. The third CPU should be placed in the second backplane. The modules of the expansion cable set must be terminated differently. The module of the expansion cable set in the first backplane should add no additional termination to the system while the module in the second backplane should be terminated with 240 ohms. Therefore the second backplane is also terminated in 120 ohm (240 on the processor in parallel with 240 on the expansion module).

Four CPU System:

A four CPU system should also be configured in 2 backplanes. The first backplane will have 2 CPUs for 120 ohm termination and the second backplane will also have 2 CPUs for 120 ohm termination. But the expansion cable set connecting the two backplanes will not be the same as in the three CPU system. In this case there should be no termination on either of the expansion modules.

These same basic rules should be followed if the BA23 enclosure is used. However the configurations become more constrained because of the relatively few number of Q/CD slots. The configuration also becomes more complicated because the BA23 has built in termination on the backplane (termination resistors are socketed and therefore removable).

4. Using a PDP-11 CPU as the Arbiter.

The arbiter can be a Q-bus PDP-11 CPU as well as a MicroVAX II CPU. In this configuration only the 3 auxiliary MicroVAXes could be added to the system. Although there are tremendous architectural differences between a PDP-11 and MicroVAX CPU, they both perform the arbiter function on the Q-bus and therefore such a configuration is possible. Two issues arise from this 'mixed' configuration.

1. All Q-bus PDP-11s to-date have their main memory on the Q-bus. In order to talk to an auxiliary MicroVAX CPU's local memory, part of the Q-bus physical address space must be reserved for mapping to that local memory.

2. The PDP-11 processor will not have an interprocessor communication register because that feature is unique to the MicroVAX II CPU. Therefore, if it is necessary to have the capability for an auxiliary CPU to interrupt the arbiter, an additional device must be added to the system. This device will look like the interprocessor

communication register for an arbiter and will convert and ICR interrupt into a conventional Q-bus interrupt (there is no capability which allows an auxiliary to interrupt the arbiter via the standard Q-bus interrupt protocol).

| uNOTE # 027 |
|---|

| Title: USING MESSAGES WITH VAXELN | Date: 28-JUN-85 |
|---|---|
| Originator: Christopher DeMers | Page 1 of 5 |

This Micronote discusses how VAXELN uses messages for inter-job communication. Included is an overview of the message architecture, benchmark data from MicroVAX II configurations and a sample program illustrating the message handling technique.

In a VAXELN application, every job(Pascal,C or Macro-32 program) has a unique and protected virtual address space. Within a single processor, the kernel separates each job's virtual address space using the VAX memory management hardware. Within a network, each job's virtual address space is separated by virtue of the fact that the jobs may exist in the memories of different computer systems.

One of the principal reasons for dividing an application into separate jobs is to aid the migration and distribution of the jobs within the network. In order for the jobs to work together on the same application, they must be able to share data, but the only way of moving data from the memory of one processor to another in a network is by packaging the data in a message and directing the network hardware to move the message to the destination memory.

To make the network movement of data between jobs the same as in the non-network case, message passing is the principal means of inter-job communication. The kernel provides a number of facilities to make an efficient and transparent means of communication.

The VAXELN MESSAGE object describes a block of memory that can be moved from one job's virtual address space to another's. The block of memory is called 'message data' and is allocated dynamically by the kernel from physically contiguous, page-aligned blocks of memory. A MESSAGE object and its associated message data are both created by calling the CREATE_MESSAGE kernel service.

Message data is mapped into a job's P0 virtual address space, so it is potentially accessible to all the processes in the job. If a message is sent to a job on the sending job's local node, the kernel unmaps the

message data from the sending job's virtual address space and remaps it into the receiver's space. Instead of moving the data, Page Table Entries are remapped, resulting in very fast message transmission time. If a message is sent to a remote node, the kernel again unmaps the message data, but it remaps it into the appropriate network device driver job to send the message to the remote system. The reverse operations then cause the message data to be remapped in the receiver's space.

The code necessary to send a message local to a machine or over the network is identical in both cases. A port name table parameter (NAME$LOCAL or NAME$UNIVERSAL) allows the programmer to define local or remote ports. The kernel maintains the local name list, while the name table for network-wide names is maintained by the name server. Names known throughout the network are guaranteed to be unique. Thus, jobs can initially be executed on one processor and later be segmented to run on multiple processors without modification of the program.

## Datagrams and Circuits

Messages can be used two ways to transmit data:

- One process can obtain the value of a port anywhere in the system (including other jobs) or in a different system running on a different Ethernet node. The process can send the port a message with the SEND procedure. This is called the datagram method.

- Any two ports (usually in different jobs) can be bound together to form a circuit. In this case, having established the circuit, the sending process has one port of its own bound to another port, which usually is in a different job or on a different network node. The sender sends the message to its own port, and it is routed automatically to the other port in the circuit. Processes can both send to and receive from a circuit port.

In the datagram method, as well as in the circuit method, a process can use the WAIT procedures to wait for the receipt of a message on the specified port.

The datagram method requires no connection sequence as in circuits, but cannot guarantee that a message is actually received at the destination. However, it does guarantee that received messages are correct.

At the cost of setting up and handling a circuit connection, circuits offer several advantages over the basic datagram method:

- Guaranteed delivery. The circuit method guarantees that either the message arrives at the destination port regardless of its location, or that the sender is notified that the message could not be delivered.

- Flow control. You can prevent a sending process from sending too many messages to a slower receiving process. If a port is connected in a circuit and is full, the sender is put into the waiting state until the port is no longer full (you can override this default). The implicit waiting performed by the SEND procedure evens the flow of messages between the transmitting process and the receiving process without having to explicitly program for the condition. Furthermore, circuits guarantee the sequence of message delivery as no other unconnected port can send a message to a connected port.

- Segmentation. Message can have any length (datagrams are limited to approx. 1500 bytes), and, if the transmission is across the network, the network services will divide the message into segments of the proper length, transmit the segments and reassemble them at the destination node.

- User interface via Pascal I/O routines. The OPEN procedure permits you to 'open' a circuit as if it were a file and to use the I/O routines such as READ and WRITE to transmit messages.

There is no performance penalty with circuits for messages transmitted over the same network node and very little over the network. For full generality, programs should assume that the sending and receiving jobs may be distributed on different nodes in a network. Circuits are the preferred means of sending message in almost every practical case.

## Expedited Messages

An expedited message bypasses the normal flow-control mechanism and can be sent even if the receiving port already has its maximum number of messages. The message is received by the port before any normal data messages. The size of an expedited message is limited to a maximum of 16 bytes.

## Monitoring Network Integrity

Circuits can be used as a method for monitoring the status of a network connection. In addition to the circuit used to send data, another process could establish a secondary circuit with a process in a remote node. Neither process sends data, they only WAIT on their respective ports. When a node fails, the circuit is broken and the WAIT is satisfied. At this point exception handling routines could be invoked to handle this condition. This allows the programmer to separate the error handling code from the message transmission code.

## Connecting with VAX/VMS nodes

The VAXELN procedure CONNECT_CIRCUIT can be used to request a connection with a VAX/VMS program on the same DECnet network. Instead of using a port as the destination for the connection, a VMS node and an object name (program name on the VMS node) is used. The program on the VMS node completes the connection by opening the "file" SYS$NET.

A VAX/VMS node may also request a connection by specifying a node name and port name in the file portion of an OPEN statement. The VAXELN node complete the connection by issuing an ACCEPT_CIRCUIT statement.

## Performance Data

MicroVAX II, 5MB memory, connection via circuit
All times in microseconds (us).
Throughput in thousand bytes/second(KB/s).

| Message Size (bytes) | Create Time | Delete Time | Delivery One Machine Time | Delivery One Machine Throughput | Delivery Two Machines Time | Delivery Two Machines Throughput |
|---|---|---|---|---|---|---|
| 0 | 131 | 144 | 1094 | N/A | 4479 | N/A |
| 512 | 361 | 255 | 1611 | 318 | 5271 | 97 |
| 2048 | 450 | 274 | 1885 | 1087 | 8675 | 236 |
| 8192 | 481 | 394 | 2139 | 3830 | 27538 | 298 |

```
PROGRAM msgsend(INPUT,OUTPUT);
{++
{ This program connects a circuit to the global name 'msgrecv',
{ sends a message to the receiver and waits for a reply.
{--}
VAR
     send_port,reply_port: PORT;
     msg,reply:            MESSAGE;
     msg_data_ptr:         ^VARYING_STRING(20);
     reply_data_ptr:       ^VARYING_STRING(10);

BEGIN
  CREATE_PORT(send_port);
  CONNECT_CIRCUIT(send_port, destination_name := 'msgrecv');
  CREATE_MESSAGE(msg,msg_data_ptr);
  msg_data_ptr^ := 'message from msgsend';
  SEND(msg,send_port);
  WAIT_ANY(send_port);
  RECEIVE(reply,reply_data_ptr,send_port);
  DELETE(reply);
  DISCONNECT_CIRCUIT(send_port);
END; { program msgsend }


PROGRAM msgrecv (INPUT,OUTPUT);
{++
{ This module runs a receive program that accepts a circuit,receives
{ a message and sends back a reply.
{--}

VAR
     reply_port:     PORT;
     nam:            NAME;
     msg,reply:      MESSAGE;
     msg_data_ptr:   ^VARYING_STRING(20);
     reply_data_ptr: ^VARYING_STRING(20);

BEGIN
  CREATE_PORT(reply_port);
  CREATE_NAME(nam,'msgrecv',reply_port,table := name$universal);
  INITIALIZATION_DONE;
  ACCEPT_CIRCUIT(reply_port);
  WAIT_ANY(reply_port);
  RECEIVE(msg,msg_data_ptr,reply_port);
  DELETE(msg);
  CREATE_MESSAGE(reply,reply_data_ptr);
  reply_data_ptr^ := 'reply';
  SEND(reply,reply_port);
END; { program msgrecv }
```

| uNOTE # 028 |
|---|

| Title: MSV11-Q/M/J MEMORY COMPARISONS | Date: 28-JUN-85 |
|---|---|
| Originator: JACK TOTO | Page 1 of 3 |

This MicroNote will compare three of Digital Equipment Corporation's newest memory modules. Following the description of each memory module, is a chart which, list the differences between these memories.

MSV11-Q

The MSV11-Q has three versions, the MSV11-QA, the MSV11-QB and the MSV11-QC. The MSV11-QA (M7551-A) is a quad size Q-bus memory module and contains 1MB of MOS RAM using 64K bit parts. The MSV11-QA has two etch revisions which must be checked when configuring the module into a system. The two revisions are the C rev and the A rev, both of these revisions are discussed in the Users Guide (EK-MSV1Q-UG) and in Micronote # 030. The second version of the MSV11-Q is the MSV11-QB (M7551-B) which is a half populated quad size Q-bus memory module containing 2MB of MOS RAM using 256K bit parts. The last version of the MSV11-Q is the MSV11-QC which is a fully populated MSV11-QB quad size Q-bus memory module containing 4MB of MOS RAM using 256K bit parts. Both the MSV11-QB and the MSV11-QC use the MSV11-QA printed circuit board, however that board is ECOed to etch level C.

MSV11-J

The MSV11-J has four versions, the MSV11-JB and MSV11-JC which are used in the PDP-11/84 UNIBUS systems and the MSV11-JD and MSV11-JE are used in either the MicroPDP-11/83 Q-bus systems, or the PDP-11/84 UNIBUS systems. All four modules use ECC memory for error correction, as well as using 256K bit MOS RAM parts on either a half or fully populated quad size module.

NOTE:

NONE OF THE FOUR MSV11-J MODULES CAN BE PLACED IN A Q/Q BACKPLANE SLOT. IF THIS IS ATTEMPTED PERMANENT DAMAGE WILL BE DONE TO THE BOARDS AND TO THE SYSTEM.

The MSV11-JB (M8637-BA) is a half populated quad size PMI memory module containing 1MB of memory. The second version of the MSV11-J is the MSV11-JC (M8639-CA), this is a fully populated MSV11-JB quad size PMI memory module containing 2MB of memory. These two modules can not be used in a Q-bus system due to gate array incompatibilities, and can only

be used in PDP-11/84 systems which use the UNIBUS/PMI bus interface (KTJ11-A). The third version of the MSV11-J is the MSV11-JD (M8639-DA) which is a half populated quad size PMI memory module containing 1MB of memory. The last version of the MSV11-J is the MSV11-JE, (M8639-EA) which is a fully populated MSV11-JD quad size PMI memory module containing 2MB of memory. These last two modules can be used with either the MicroPDP-11/83 system which uses the Q-bus/PMI bus interface or the PDP-11/84 system which was mentioned above. For details reference the PMI protocol Micronote # xxx.

Although the MSV11-JD and MSV11-JE are PMI memories they can be used in two other Q-bus configurations.

1. If either of these two memories are used in slots 1 and/or 2 of a Q/CD backplane such as the H9276 (BA11-S box) or in the MicroPDP-11 BA23 backplane, and with the standard 15MHZ KDJ11-B cpu (non-fpj compatible) in slot 3, the system will perform PMI memory cycles. In the case of BA23 backplanes, a maximum of two memory modules may be used in slots ahead of the processor, with a minimum of one memory module in front of the processor still functioning as PMI memory. In the case of the H9276 backplane a number of MSV11-JD/JE memory modules may be used ahead of the processor bringing the system to a full 4MB of PMI memory.

2. If in a Q/CD or BA23 backplane the memories reside in slots 2&3 with either of the KDF11 or KDJ11 processors in the slot 1 the MSV11-J memories will respond as standard Q-bus memories, performing normal Q-bus and block mode memory cycles. This use of the MSV11-J memories is also true for the MicroVAX I cpu. However the fact that the MicroVAX I cpu is a two board set requires a slightly different configuration. Slots 1&2 will be used for the MicroVAX I cpu boards with only one memory card used, that being located in slot 3 for the BA23 backplane and one or more memory cards being used for other Q/CD backplanes. The only constraint with either of the second configurations is that in the BA23 or other Q/CD backplanes no module may be placed adjacent to the MSV11-J that uses pins in the CD connector. Instead leave an empty slot between the MSV11-J and this option. An option which does not use the CD interconnect may be placed adjacent to the MSV11-J.

MSV11-M

The MSV11-M has two versions; the MSV11-MA and the MSV11-MB. The MSV11-MA (M7506-AA) is a half populated dual sized Q-bus memory module which contains 0.5MB of MOS RAM using 256K parts. The second version the MSV11-MB (M7506-BA) is a fully populated dual size Q-bus memory module containing 1MB of MOS RAM using 256K parts.

## MEMORY COMPARISON TABLE

| ATTRIBUTE | MSV11-QA/QB/QC | MSV11-MA/MB | MSV11-JB/JC/JD/JE |
|---|---|---|---|
| MEMORY SIZE | 1MB, 2MB, 4MB | 0.5MB, 1MB | 1MB, 2MB |
| FORM FACTOR | QUAD | DUAL | QUAD |
| SYSTEM SIZE | Q18/Q22 | Q18/Q22 | PMI/CD ONLY |
| BLOCK MODE | YES | YES | YES (JD/JE ONLY) |
| BLOCK MODE CONFIGURABLE | YES FOR REV-A<br>NO FOR REV-C | NO<br>(DEFAULT OPTION) | NO<br>(DEFAULT OPTION) |
| AC BUS LOADS | 2.4 | 2.0 | 2.5 |
| DC BUS LOADS | 1.0 | 1.0 | 0.5 |
| AMPS @ +5 VDC<br><br>(ACTIVE ONLY) | TYP MAX<br>QA 2.4 3.99<br>QB 2.3 3.59<br>QC 2.5 4.30 | TYP MAX<br>MA 2.19 3.11<br>MB 2.22 3.34 | TYP MAX<br>JB/JD 1.5 3.94<br>JC/JE 1.7 4.29 |
| AMPS @ +12VDC | NO +12V SUPPLY NEEDED | | |
| WATTS<br><br>(ACTIVE ONLY) | TYP MAX<br>QA 12.0 20.95<br>QB 11.5 18.85<br>QC 12.5 22.58 | TYP MAX<br>MA 11.0 16.3<br>MB 11.1 17.5 | TYP MAX<br>JB/JD 7.5 19.7<br>JC/JE 8.5 21.5 |
| MEMORY ERROR CORERECTION | PARITY | PARITY | ECC |
| BATTERY BACKUP SUPPORT | YES FOR REV-C<br>NO FOR REV-A | YES | YES |
| DIAGNOSTICS | CVMSAA FOR PDP11s<br>EHXMS FOR MICROVAX | SAME<br>SAME | CVMJA0 |
| MAINTENANCE PRINTSET | MP01931 | MP02053 | MP02056 |
| USER GUIDE | EK-MSV1Q-UG | EK-MSV1M-UG | EK-MSV1J-UG |

| uNOTE # 029 |
|---|

| Title: Q-bus Expansion Concepts | Date: 28-Jun-85 |
|---|---|
| Originator: Charlie Giorgetti | Page 1 of 5 |

This MicroNote discusses the expansion (multiple backplanes) characteristics of a Q-bus system. Understanding this topic is critical when configuring a system. The loading, impedance, and single backplane characteristics of the Q-bus and some assumptions and definitions are discussed prior to defining the expansion rules. The specific products used in expansion are not discussed here.

## Viewing the Q-bus for Electrical Analysis

When analyzing the Q-bus from a configuration rule standpoint the bus is treated as a transmission line. The reasons for this:

- The Q-bus has voltage sources at both ends of a conductor.
- When one of these voltage sources (typically a processor) changes state (a control/data signal transitioning) its effect is not seen instantaneously at the other end, but after some propagation delay. The propagation delay could result in signal reflections on the bus if it is not properly terminated or expanded.

## Loading Definitions

The Q-bus specification defines two loading parameters used when configuring a system. These parameters, AC and DC loading, indicate the load presented to the system by individual elements on the Q-bus. A system element is either a Q-bus module or a backplane. The definition of AC and DC loads are:

- AC loading is the capacitive loading added to a Q-bus system by a Q-bus module or by the backplane itself. Capacitive loading will cause bus reflections and impact signal rise and fall times. This is measured at the time the module or backplane is being designed. An AC load is 9.35 pf/signal line.
- DC loading is the amount of leakage current presented to the Q-bus by an undriven signal line on a Q-bus module. This information is obtained from the specification data for Q-bus drivers and receivers. A DC load is defined as 210 uA.

The number of AC and DC loads allowed in a configuration is dictated by the number of backplanes and the termination used. This will be discussed in later sections of this MicroNote.

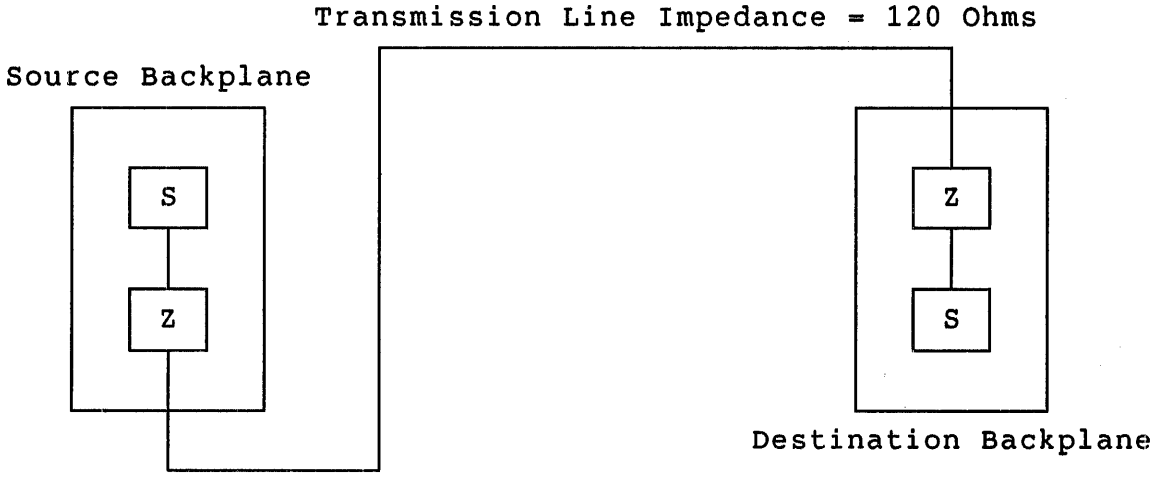
The AC and DC values for Q-bus modules and backplanes can be found in either the Microcomputer Products Handbook (#EB-26078-41) or the Microcomponents Configuration Guide (#EB-27318-68).

## Backplane Configurations

The rules that govern Q-bus system implementation must be viewed in light of the backplane arrangement used. The two supported Q-bus configurations are: single backplane or multiple backplanes. How the Q-bus is treated as a transmission line varies for these two configurations and is the foundation for the implementation rules.

## Impedance and Termination Characteristics

The characteristic impedance of the Q-bus is approximately 120 Ohms. Therefore, when implementing a system (single or multiple backplane) the basic configuration is:




Z - Bus Termination with 120 Ohms Characteristic Impedance
S - Voltage Source

The transmission line in this diagram could be a single backplane or multiple backplanes connected with expansion cables.

Figure 1 - General Q-bus Configuration

## Q-bus Configuration - Single Backplane

For the single backplane case the transmission line is the length of the etch runs on the Q-bus connector blocks and the backplane printed circuit board. This orientation has a signal generator at one end (the processor) and potentially a terminator at the far end of the bus. The length of the etch runs cannot exceed 14 inches (35.56 cm). In figure 1 the transmission line is the backplane itself.

A single backplane system does not require termination if there are less than 20 AC loads. In this case the signals do not lose their integrity because the reflections, caused by the mismatched impedances, are not significant enough to disrupt bus activity. However, in a high ambient electrical noise environment, system integrity may be further insured by proper termination.

The single backplane configuration requires termination if the number of AC loads is 20 or greater. The number of allowable AC loads in this case is dictated by the termination on the processor. A 120 Ohm processor can have up to 45 AC loads. A 240 Ohm processor can have up to 35 AC loads.

## Q-bus Configuration - Multiple Backplanes

For the multiple backplane case (where the multiples are two or three backplanes) the transmission line is the cables used to interconnect the multiple backplanes. The expansion cable set consists of:

- A module in the source backplane
- A module in the destination backplane
- Cables to connect the two modules

The maximum length of the cables is 16.0 feet (4.88 meters). The length of these cables is by comparison significantly longer then the length of the Q-bus connector blocks and the backplane etch used in the single backplane case. Therefore, only the interconnect cables are considered for configuration purposes. This arrangement has a signal generator at one end and requires termination at the far end.

The far end termination must reside in the last backplane of the configuration. The location of the far end termination can be any place in the last backplane, since the backplane etch runs do not enter into transmission line considerations. The lump sum termination must be 120 Ohms.

The termination in the source box must also be 120 Ohms. If the processor is 240 Ohms then the expansion cable set module or the backplane printed circuit board must have 240 Ohm termination to achieve the 120 Ohms for the lump sum load.

Lump sum implies that the 120 Ohms can be achieved by one or more expansion module or backplane printed circuit board mounted terminators and its location is position independent in a given backplane. Figure 2 shows an example of how such a lump sum load can be accomplished.

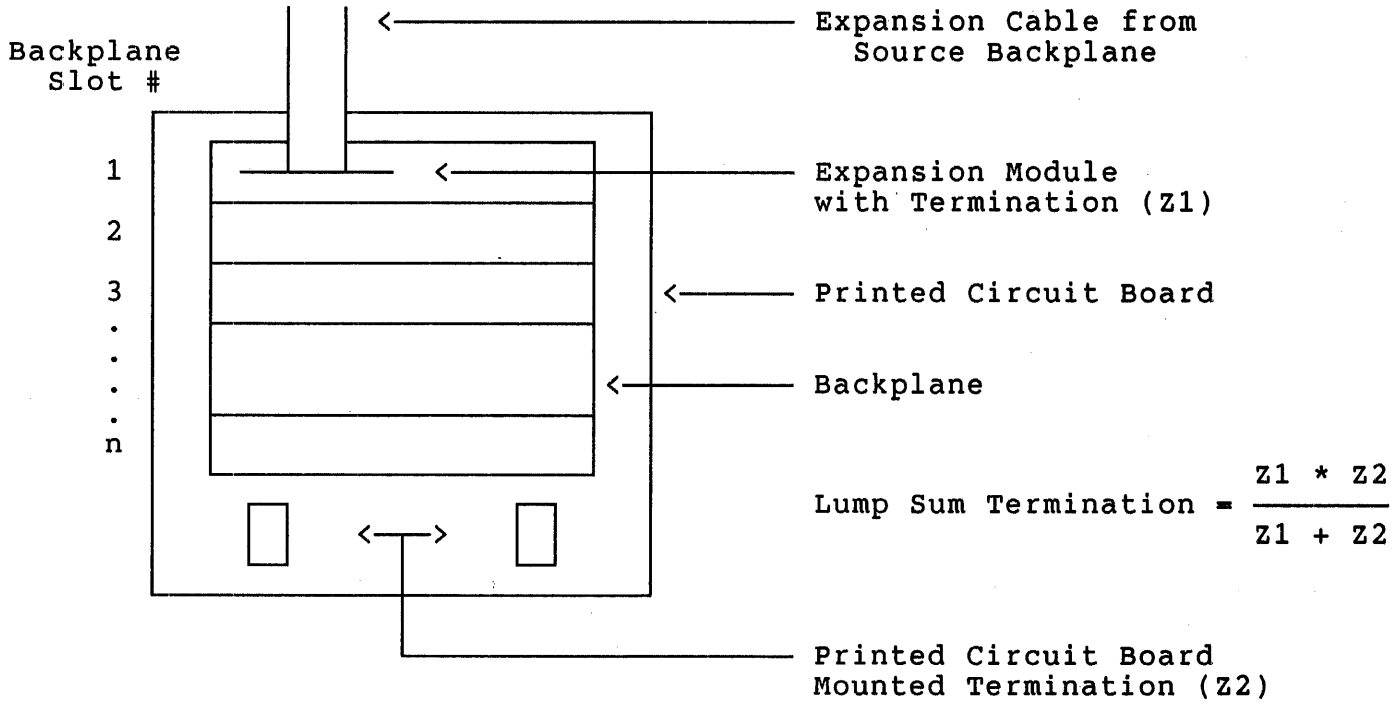



Figure 2 - Example of Lump Sum Termination in an Expanded Backplane

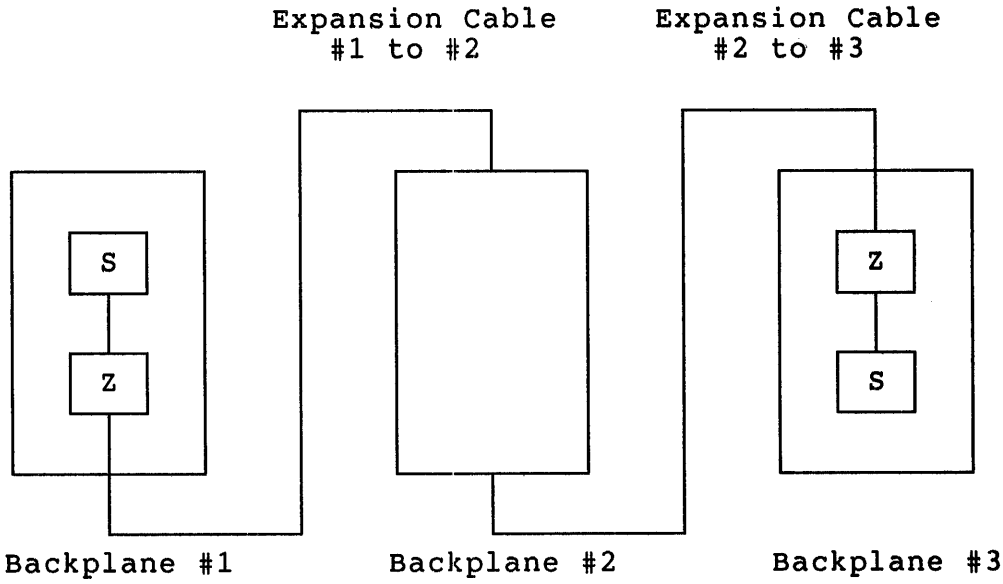
Figure 1 shows the double backplane configuration where the expansion cable set is considered the transmission line. The far end termination is required. Figure 3 shows the three backplane configuration. Backplane #2 in figure 3 for all practical purposes is part of the expansion cable set when looking at it from an expansion point of view.

The lengths of the expansion cables in multiple box configurations are strictly specified. As mentioned the maximum length of the overall cable is 16.0 feet. The minimum length is 2.0 feet (0.61 meters). Therefore, in a two backplane configuration the expansion cable must be between 2.0 and 16.0 feet.

In the three backplane configuration the maximum cable length is still 16.0 feet. One of the two interconnect cables must be between 2.0 feet and 6.0 feet (1.83 meters) in length. The other interconnect cable must be at least 4.0 feet (1.22 meters) but not longer than 10 feet (3.05 meters). The difference in the two cable lengths must be 4.0 feet.

The cable lengths are specified to insure that any reflections occur in the expansion cables and not in the backplane (if they happen).

The etch runs on the backplane printed circuit board used in a multiple backplane configuration must be no longer than 10 inches (25.4 cm). Not all backplanes used in single configurations can be used in multiple backplane configurations.




Z - Bus Termination with a Characteristic Impedance
S - Voltage Source

Figure 3 - Three Backplane Q-bus Configuration

Multiple backplane configurations allow 22 AC loads/backplane. Therefore, it is 44 AC loads in a two or 66 AC loads in a three backplane configuration. To avoid lumping too many AC loads together the total number of AC loads should be distributed as evenly as possible over the two or three backplanes. The entire configuration cannot exceed 20 DC loads.

In summary, following the expansion rules insures proper system operation. The set of rules to be followed are dictated by the single or multiple configuration chosen and the arrangement of the termination in the system.

| uNOTE # 030 |
|---|

| Title: The Private Memory Interconnect between the KDJ11-B and the MSV11-J | Date: 28-Jun-85 |
|---|---|
| Originator: Peter Kent | Page 1 of 9 |

## Purpose

This MicroNote describes the Private Memory Interconnect on the MicroPDP-11/83 system. It is not intended to be a design guide for PMI, since no devices other than the CPU and memory will make use of it.

## General Description

A MicroPDP-11/83 system consists of the KDJ11-B CPU and one or more MSV11-J memories in a Q-bus backplane. The slots used for the CPU and memory use the CD interconnect. In a MicroPDP-11/83 configuration, the first 1 or 2 slots in a BA23 backplane (an 8 slot backplane with the first 3 slots Q/CD) are reserved for MSV11-J memory. The CPU is put in the third slot. The mechanical design of the signal pathways between the CPU and memory were designed to prevent accidental interference between the PMI and other devices that might be placed adjacent to the CPU and memory. It is possible to have a single 2 Mb board in slot 1 followed by the CPU in slot 2. Putting the CPU after the memory ends the PMI because the PMI signals from the CD side of the CPU board are only on the component side of the CPU board.

"Private Memory Interconnect" is the addition of 14 unique signals to the Q-bus. These new signals use the CD part of the backplane in a Q-bus system for communications between the KDJ11-B CPU and one or two MSV11-J memories. Only the CPU and memory may communicate over this bus (hence PRIVATE). The Q-bus Data/Address lines are used for passsing data and addresses between the CPU and memory. All other Q-bus transactions proceed as before.

The PDP-11/84 Unibus system uses the KTJ11-B Unibus Adaptor, KDJ11-B CPU, and one or two MSV11-J memories. No Q-bus devices may be configured with the PDP-11/84 system. Five of the PMI signals are used only with Unibus systems. All communications between Unibus devices and the KTJ11-B occur according to the Unibus protocol. The KTJ11-B provides the interface between PMI and Unibus protocols. This MicroNote does not explain the details of the Unibus and PMI interaction.

## What is PMI?

To understand PMI it is necessary to describe some of the bus cycles used by PMI and compare them with ordinary Q-bus cycles. There are 4 PMI cycles used in the MicroPDP-11/83 system:

DATI - Data word input. This cycle is used to read one or more 16 bit words from memory by the CPU.

DATIP - Data word input pause. This cycle is used to read one or more 16 bit words from memory by the CPU. It is often used to perform a read/modify/write cycle.

DATO - Data word ouput. This cycle is used to write a 16 bit word to memory by the CPU.

DATOB - Data byte output. This cycle is used to write a byte to memory by the CPU.

The KDJ11-B does not perform Block Mode reads or writes with memory. Certain other Q-bus devices (such as the RDRX1 and RQDX2 controllers and RQC25) perform Block Mode DMA with MSV11-P,MSV11-Q,AND MSV11-M memories. The CPU monitors Block Mode transactions to keep its cache in order, but it has no control over such transfers once it has relinquished control of the bus to those devices.

### REFERENCES

PMI signal definitions are listed at the end of this MicroNote. All other signal definitions are as given in the Microcomputer Products Handbook (EB 26078-41) or Microsystems Handbook (EB 26085-41).

TWTBT, a Q-bus signal, is used somewhat differently during a PMI transaction than during a normal Q-bus transaction. See the definitions section for an explanation of this signal.

### NOTE

Few timing relationships are given here because it is not necessary for a basic understanding of PMI. The relationships that are given are a comparison to Q-bus only. The prefix "T" refers to bus driver input and "R" refers to bus receiver output. This helps to distinguish between the device which issues a particular signal and that of the receiver of the signal. If the prefix "B" is used, it denotes a general term for the signal on the bus without regard for its timing relation with respect to sender or receiver (i.e. BSYNCH).

## Address part of cycle

The address portion of the MicroPDP-11/83 PMI cycles is the same for all 4 PMI cycles. It is listed here first and the description of the other cycles follow.

1. For the first part of the cycle, the CPU gates ADDR, BBS7 (if the I/O page is referenced), TWTBT, and TPBYT onto the Q-bus. The combination of TWTBT and TPBYT (PMI signal) determine what type of PMI transaction will take place - see Table 1 below. These signals are asserted for a brief period after the assertion of TPBCYC.

Table 1

| BWTBT L | PBYT L | Description |
|---|---|---|
| H | H | DATI or DATBI Cycle |
| H | L | DATIP Cycle |
| L | H | DATO Cycle |
| L | L | DATOB Cycle |

As noted above, Q-bus signals such as TWTBT are used differently during a PMI cycle. TWTBT is not normally used during a DATI cycle, for example.

2. Next, memory issues TPSSEL after receiving RADDR and RBS7. The CPU receives RPSSEL. This part of the cycle indicates that the memory is responding.

3. If the CPU was the previous bus master and the previous cycle was a Q-bus cycle, then the CPU must not assert TPBCYC or TSYNC until after the negation of TSYNC and after the negation of RRPLY. If the cycle was an interrupt service cycle, then the CPU must wait until after the negation of RRPLY. This stipulation allows for other Q-bus devices equal access to the bus. Also, if another Q-bus device was previously bus master, then the CPU must not asssert TPBCYC or TSYNC until after the negation of RSYNC - this is to make sure RRPLY is negated long enough as per Q-bus protocol.

4. Now, if RPSSEL is asserted and if RPUBMEM is negated, the PMI master proceeds with a PMI cycle. The negation of RPUBMEM indicates no that no Unibus memory is responding - in other words a Q-bus PMI cycle. Here is where the diffe- rence between Q-bus cycles and PMI cycles becomes visible.

5. The CPU asserts TPBCYC after gating TADDR, TBS7, TPBYT, and TWTBT onto the bus. The PMI master continues to gate TADDR, TBS7, and TWTBT after the assertion of TPBCYC.

6. If at this point RPSSEL is negated, the system will revert to a normal Q-bus cycle.

This describes the address portion of the PMI cycle for Q-bus only systems.

## DATI

When the CPU (KDJ11-B) is accessing memory for a PMI Data In Cycle, it transfers 2 words. This takes advantage of the KDJ11-B restart overhead to load a second 16-bit word into the cache on the CPU module. Listed below equivalent Q-bus cycles are compared with PMI cycles under "Timimg Comparisons". For the read cycle, 2 data words (one after another) are latched into the MSV11-J data gate array and both words are placed on the bus. Interestingly enough, either word may be selected to be placed on the bus first. If the odd word is placed on the bus first, it is followed by the preceding even word. For example, if a word at address 17362 is selected to be placed on the bus first, the next word transferred will be from address 17360. If the even word is selected to be placed on the bus first, the next odd word is then transferred second. For example, if a word at address 17360 is selected to be placed on the bus first, the next odd word is then transferred second. For example, if a word at address 17360 is selected to be placed on the bus, the next word transferred will be at address 17362.

Using the MSV11-J memory as a normal Q-bus memory (disabling the PMI by putting it after the CPU in the bus) and performing a read cycle, 2 words will be latched into the data gate array. However, only one word is placed on the Q-bus.

1. At this point the address portion of the cycle is over. The memory gates TDATA onto the bus after the assertion of RPBCYC.

2. Immediately after RPBCYC the memory enables the parity signals TPHBPAR and TPLBPAR.

3. Memory asserts TPRDSTB immediately after the reception of RPBCYC.

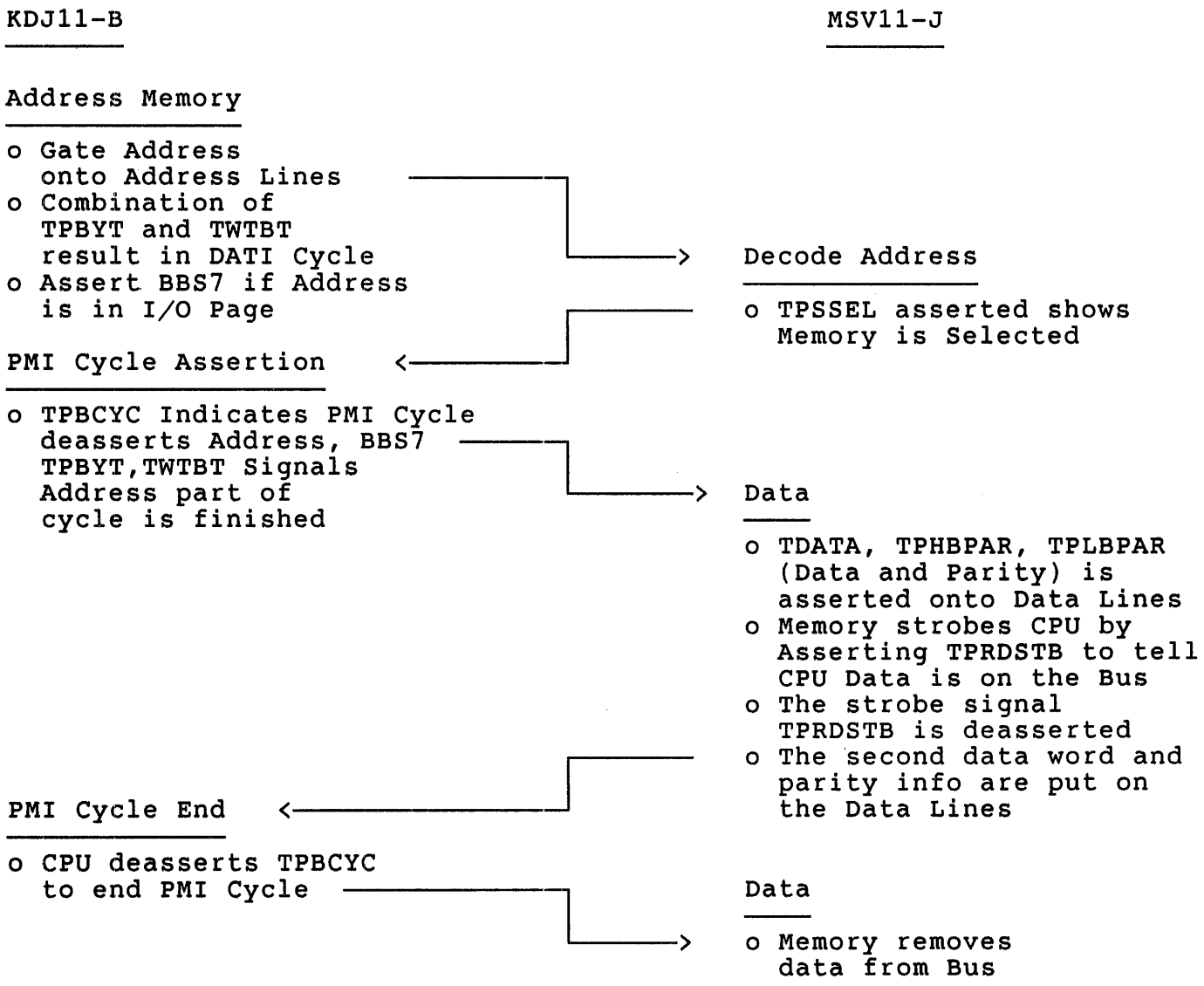
4. The strobe signal TPRDSTB is negated by memory after the reception of RPBCYC.

5. The memory gates the second data word onto the bus after TPRDSTB is negated. Shortly after the parity in- formation is sent. Another advantage is realized here: the second data word is transmitted without the need of any further signals between the CPU and memory.

6. If the CPU has read 2 words, it negates TPBCYC after latching the second data word.

7. After the negation of RPBCYC memory removes TDAT from the bus.

It becomes evident that there is an essential difference between normal Q-bus transactions and PMI transactions. Q-bus transactions are all based on handshaking. During a DATI Q-bus transaction, the memory responds with BRPLY after BDIN from the processor. There must be the signal between each device that the transaction has taken place. During a PMI DATI transaction, there is only a strobe signal from the memory saying that it is ready for the data. The only further communication between the memory and CPU is the actual data transfer. Only upon the transmission of the second possible data word is parity information sent along with the second data word. No other memory-CPU signalling occurs.



## Timing Comparisons

At this point it would be interesting to make some timing comparisons between Q-bus and PMI DATI transactions. Considering a Q-bus DATI transaction, the cycle time (timing from BSYNCH to TRPLY ignoring addressing time) is 510 ns for MSV11-M. The MSV11-M was chosen because its cycle time does not include the ECC overhead of the MSV11-J. Add to this 320 ns access time and this totals 830 ns to transfer 1 word from memory to CPU. Now, if 2 words are to be transferred in this manner, add another 300 ns due to delay between RRPLY and TSYNCH in order for other Q-bus devices access to the bus. This results in 1130 ns total for a 2 word transfer using MSV11-M parity memory.

The MSV11-J memory access time is 417 ns. This is the time from RPBCYC to PRDSTB. Fifty eight ns later, at the trailing edge of PRDSTB, the CPU receives the second data word. This means that PMI is approx. 2.5 times faster than Q-bus on 2 word reads from memory to CPU. Remember, this also accounts for the time the ECC requires to do its modified Hamming code versus the MSV11-M parity check. For each 18 bits of MSV11-J memory there are 6 bits used for ECC. This accounts for the space needed on the MSV11-J for 2 Mb of memory whereas 4 Mb is possible on the MSV11-Q on the same size board.

## DATO (DATOB)

The CPU uses DATO to transfer a single word (or byte for a DATOB cycle) to memory. The address portion of the cycle is the same as for the DATI and is described above.

1. The CPU determines what type of cycle (DATO or DATOB) by logically combining TWTBT and TPBYT. During the address portion of the cycle, these signals were used to indicate which type of PMI cycle was selected.

2. Memory asserts TRPLY after RPBCYC.

3. After the assertion of TPBCYC, data is gated onto the bus by the CPU.

4. The CPU asserts TPWTSTB after data is gated onto the bus.

5. Memory asserts TPSBFUL after RPWTSTB.

6. The CPU deasserts TPBCYC after negating TWTSTB.

7. The memory waits before it can accept another PMI cycle (or Q-bus cycle) - then it deasserts TRPLY.

8. Memory negates TPSBFUL before it can accept another PMI (or Q-bus cycle).

KDJ11-B

MSV11-J

Address Memory

- Gate Addresses onto Address Lines
- Combination of TPBYT and TWTBT Result in DATO Cycle
- Assert BBS7 if Address is in the I/O Page

→ Decode Address

- TPSSEL Shows Memory is address selected

PMI Cycle Assertion

- Assertion of TPBCYC indicates PMI Cycle
- CPU deasserts Address, BBS7, TPBYT, TWTBT Signals - Address Portion of Cycle is finished

→ Memory Responding

- After reception of RPBCYC, memory sends TRPLY to CPU

Data

- TDATA, TPHBPAR, TPLBPAR (Data and Parity) is put onto the Data Lines
- CPU then strobes the Memory by asserting TPWTSTB

→ Reception of Data

- Memory asserts TPSBFUL after reception of RPWTSTB

PMI Cycle End

- CPU deasserts TPBCYC to indicate current PMI Cycle is finished

→ Memory Cycle End

- Memory waits for other devices to claim Bus before it can accept another PMI Cycle - then deasserts TRPLY
- Memory must negate TPSBFUL before another PMI Cycle can begin

## Timing Comparisons

Here is a comparison of a Q-bus DATO cycle with a PMI DATO cycle. Looking at a Q-bus DATO transaction, the cycle time (timing from RSYNCH to TRPLY) for MSV11-M is 550 ns. Comparing this to the MSV11-J DATO cycle, the result of 223 ns is obtained. This figure is arrived at as follows: 38 ns access time for the memory + 80 ns TPBCYC to TDATA + 75 ns TPWTSTB after data is on the bus + 30 ns to hold TPWTSB. Again the speed advantage of PMI transactions over normal Q-bus transactions is about 2.5 to 1.

## DATIP

The PMI Data In Pause cycle is identical to the DATI cycle except that TPBYT is asserted with TADDR to indicate that the next cycle (immediately following the current cycle) will be a data out cycle to the same address.

## PMI and Q-bus signal definitions

Eight of the PMI signals are used in an MicroPDP-11/83 system, therefore only those will be defined here. One Q-bus signal, BWTBT, is defined here because it is used differently than during normal Q-bus transactions.

PBYT L — PMI Byte. When the CPU gates address onto the bus, it asserts this signal with BWTBT to indicate the type of bus cycle (see Table 1 above).

PBCYC L — PMI bus cycle. The CPU asserts this signal at the start of a PMI cycle and negates it at the end of that cycle.

PRDSTB L — PMI read strobe. Memory asserts and negates this signal to control data transfers during DATI cycles. The CPU latches the received first word data on the negating edge of this signal. The second word is latched after that without further signalling.

PWTSTB L — PMI Write Strobe. The CPU asserts this signal after gating data onto the bus. The memory latches the data into its write buffer after the leading edge of this pulse.

PSSEL L — PMI slave selected. Memory asserts this signal whenever it decodes its address on the Q-bus.

PHBPAR L PMI high byte data parity. This signal is generated by PMI memory during DATI cycles and provides odd parity for the high byte data on the Q-bus.

PLBPAR L PMI low byte data parity. This signal is generated by PMI memory during DATI cycles and provides even parity for the low byte data.

PSBFUL L PMI memory buffer full. Memory asserts this signal during a write cycle indicating that its write buffer is full and that it cannot respond to another cycle request.

## Q-bus signal

BWTBT L Write byte (PMI write indication). In Q-bus systems, this signal is used for Q-bus write cycles. For PMI transactions, the CPU gates this signal with PBYT to indicate the type of PMI cycle. See Table 1 above.

## References


| | |
|---|---|
| Microcomputer Products Handbook | EB-26078-41 |
| Microsystems Handbook | EB-26085-41 |
| MSV11-M User Guide | EK-MSV1M-UG-001 |
| MSV11-Q User Guide | EK-MSV1Q-UG-002 |
| MSV11-J User Guide | EK-MSV1J-UG-001 |

| uNOTE # 031 |
|---|

| Title: MSV11-QA Revision Differences | Date: 28-JUN-85 |
|---|---|
| Originator: Jack Toto | Page 1 of 9 |

Digital Equipment Corporation recently announce the addition of three memories for the Q-bus space; the MSV11-QA quad module at 1MB, the MSV11-QB quad module at 2MB, and the MSV11-QC quad module at 4MB.

Earlier versions of the MSV11-QA were shipped using a revision A etch and the documentation correctly showed the jumper configurations. However early in 1985 the MSV11-QA/QB/QC started shipping with revision C etch. The documentation at that point did not show how to properly configure the C etch. It is the intent of this MicroNote to point out the differences between the two modules and to explain the configuration of both the etch revisions.

The two modules can be identified in any one of the three ways listed in the table below;

1. The location and value of the modules serial number.
2. The location of CSR and address jumpers and switches.
3. The module's designation number, stamped on the insertion handles will be different.
   - 3A. Revision A modules will be labeled M7551-AA.
   - 3B. Revision C modules will be labeled M7551-AA.

## SERIAL NUMBER LOCATION

The two modules can be identified by the location of its serial number. The revision A module will have its serial number located in the upper right hand corner of the module as you hold it with the fingers pointing down or at you and the component side facing you or up as it would lay on a work bench. This serial number will be 5017547A1. The location of the serial number for revision C module will be on the component side also, but in the upper left hand corner. This serial number will be 5017547-01-C1. The attached diagrams of the revision A and revision C modules show the location of these numbers.

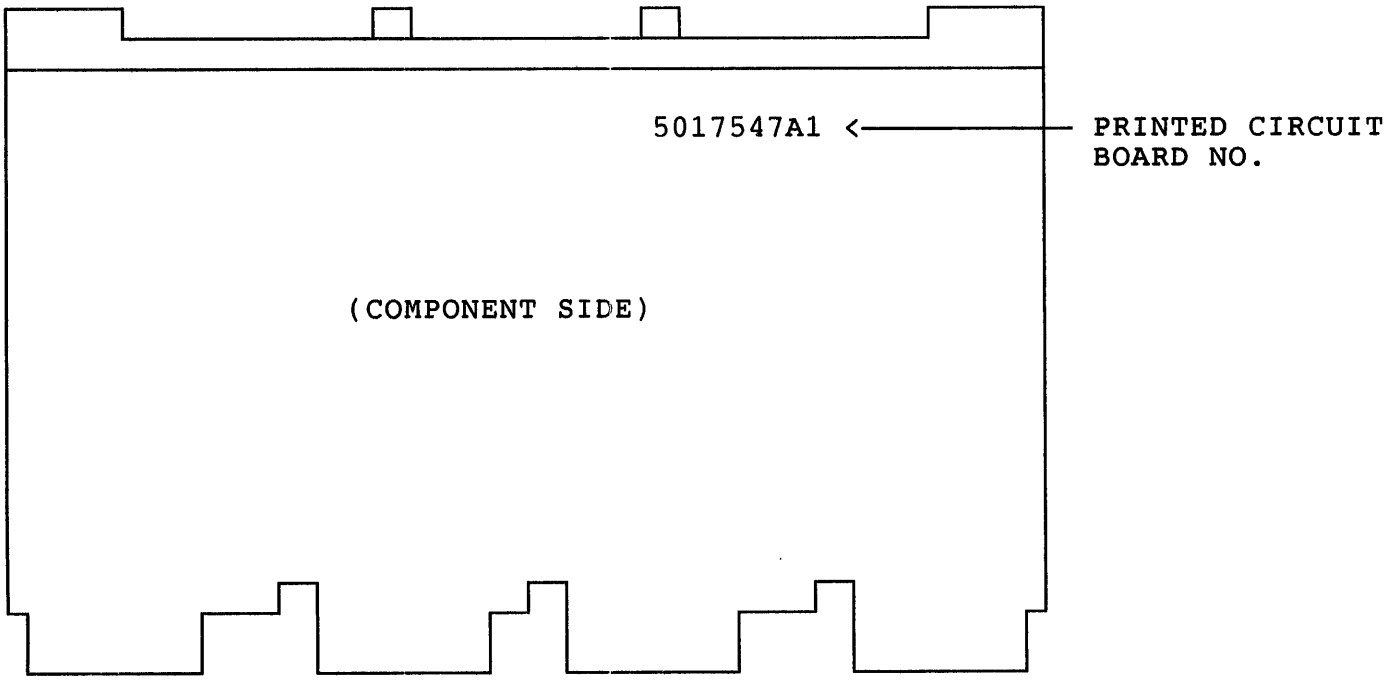
## JUMPER LOCATION

The MSV11-QA revision A module allows for the selection of Block Mode DMA transfers, parity detection, and extended address parity error detection as well as CSR and address selection. The address selection includes two switches, one for the modules starting address and one for the modules ending address. The ending address switch is used to control the decode logic on the module as the same etch is used for 2MB and 4MB boards as well, each with different value memory chips to reach there designed capacity. The location and function of these switches and jumpers if detailed in the diagrams and configuration tables attached to the end of this MicroNote.

The MSV11-QA revision C module does not allow for the selection of Block Mode DMA or for the selection of parity detection, these features are built into the module itself and can be used or not used through hardware/software compatibility. When used in a system that supports Block Mode the MSV11-QA will perform as a Block Mode device, when used in a non Block Mode system it will perform as a normal memory module without any decrease in that system's performance. When used with software such as RSX11 and using mixed parity/non-parity memories parity detection can be disabled through sysgen. The only functions that a user is required to configure to use the MSV11-QA revision C is CSR selection and starting and ending address. It should be pointed out that the two switches for starting and ending addresses have been flip flopped from the positions in which they were located on the revision A module. Once again refer to the diagrams and configuration tables attached to this MicroNote for location and function of these switches and jumpers.

The following pages contain diagrams and tables for configuring the two versions of the MSV11-QA

## MSV11-QA revision A Module Identification




(A) MSV11-QA (ETCH revision A ONLY)

## MSV11-QA revision A CSR SELECTION

| NUMBER OF MEMORY MODULE | JUMPER POSITION R | P | N | M | CSR REGISTER ADDRESS |
|---|---|---|---|---|---|
| 1ST | IN | IN | IN | IN | 17772100 |
| 2ND | OUT | IN | IN | IN | 17772102 |
| 3RD | IN | OUT | IN | IN | 17772104 |
| 4TH | OUT | OUT | IN | IN | 17772106 |
| 5TH | IN | IN | OUT | IN | 17772110 |
| 6TH | OUT | IN | OUT | IN | 17772112 |
| 7TH | IN | OUT | OUT | IN | 17772114 |
| 8TH | OUT | OUT | OUT | IN | 17772116 |
| 9TH | IN | IN | IN | OUT | 17772120 |
| 10TH | OUT | IN | IN | OUT | 17772122 |
| 11TH | IN | OUT | IN | OUT | 17772124 |
| 12TH | OUT | OUT | IN | OUT | 17772126 |
| 13TH | IN | IN | OUT | OUT | 17772130 |
| 14TH | OUT | IN | OUT | OUT | 17772132 |
| 15TH | IN | OUT | OUT | OUT | 17772134 |
| 16TH | OUT | OUT | OUT | OUT | 17772136 |

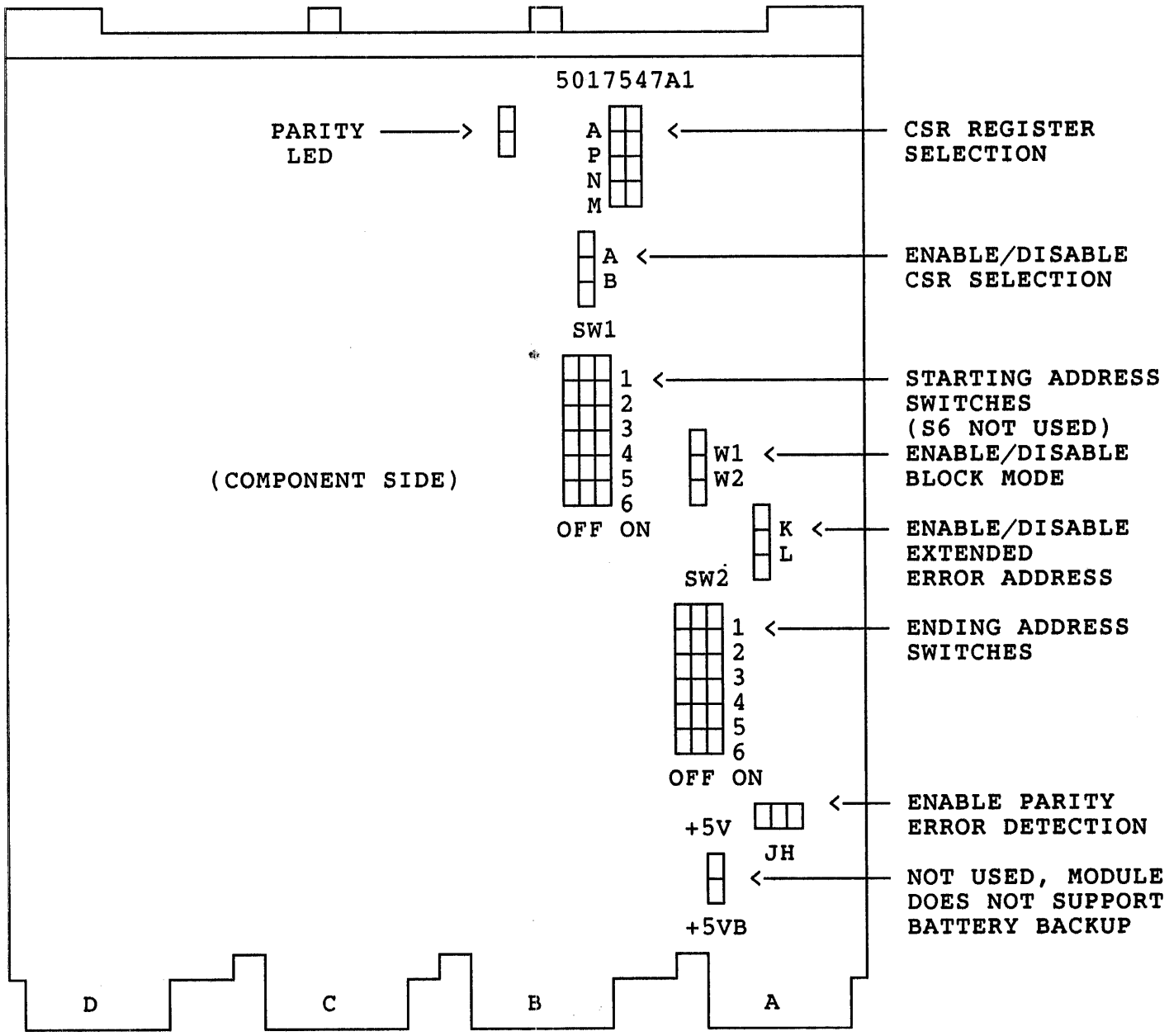
MSV11-QA revision A STARTING AND ENDING ADDRESS

| DESIRED STARTING ADDRESS | SW1 SWITCH POSITION | SW2 SWITCH POSITION | STARTING ENDING ADDRESS | ENDING SWITCH POSITION |
|---|---|---|---|---|
| IN KBYTE | 12345 | 6 | IN KBYTE | 12345 |
| 0 | 00000 | 0 | 128 | 11111 |
| 128 | 11111 | 1 | 256 | 01111 |
| 256 | 01111 | 1 | 384 | 10111 |
| 384 | 10111 | 1 | 512 | 00111 |
| 512 | 00111 | 1 | 640 | 11011 |
| 640 | 11011 | 1 | 768 | 01011 |
| 768 | 01011 | 1 | 896 | 10011 |
| 896 | 10011 | 1 | 1024 (1MB) | 00011 |
| 1024 (1MB) | 00011 | 1 | 1152 | 11101 |
| 1152 | 11101 | 1 | 1280 | 01101 |
| 1280 | 01101 | 1 | 1408 | 10101 |
| 1408 | 10101 | 1 | 1536 | 00101 |
| 1536 | 00101 | 1 | 1664 | 11001 |
| 1664 | 11001 | 1 | 1792 | 01001 |
| 1792 | 01001 | 1 | 1920 | 10001 |
| 1920 | 10001 | 1 | 2048 (2MB) | 00001 |
| 2048 (2MB) | 00001 | 1 | 2176 | 11110 |
| 2176 | 11110 | 1 | 2304 | 01110 |
| 2304 | 01110 | 1 | 2432 | 10110 |
| 2432 | 10110 | 1 | 2560 | 00110 |
| 2560 | 00110 | 1 | 2688 | 11010 |
| 2688 | 11010 | 1 | 2816 | 01010 |
| 2816 | 01010 | 1 | 2944 | 10010 |
| 2944 | 10010 | 1 | 3072 (3MB) | 00010 |
| 3072 (3MB) | 00010 | 1 | 3200 | 11100 |
| 3200 | 11100 | 1 | 3328 | 01100 |
| 3328 | 01100 | 1 | 3456 | 10100 |
| 3456 | 10100 | 1 | 3584 | 00100 |
| 3584 | 00100 | 1 | 3712 | 11000 |
| 3712 | 11000 | 1 | 3840 | 01000 |
| 3840 | 00010 | 1 | 3968 | 10000 |
| 3968 | 10000 | 1 | 4096 (4MB) | 00000 |

1 = off position (open)
0 = on position (closed)

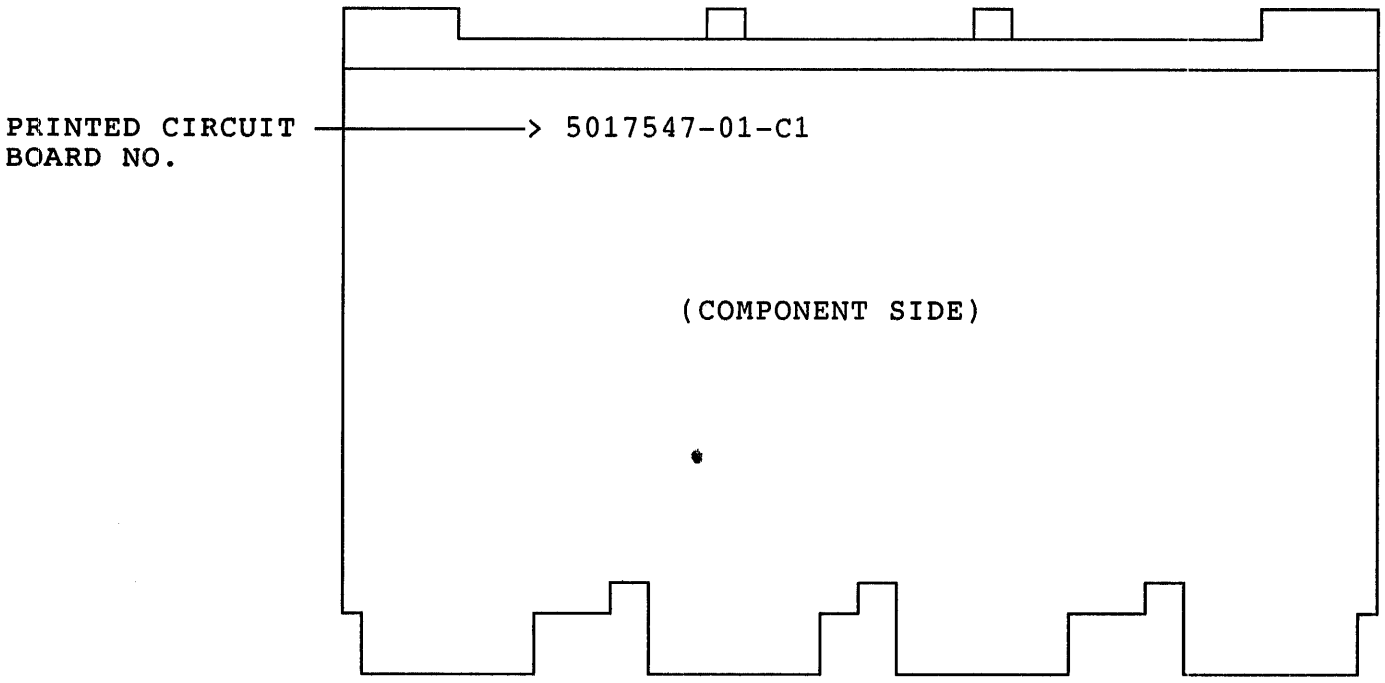
NOTE: Switch S6 of SW1 is not used. For a memory starting address of 0, switch S6 of SW2 should be set to 0 (on). For all other starting addresses S6 of SW2 should be off (1).

MSV11-QA revision A




Jumper and Switch Locations
(Drawing not to scale)

MSV11-QA revision C Module Identification




(B) MSV11-QA (ETCH revision C OR LATER)
MSV11-QB AND MSV11-QC

(Drawings not to scale)

MSV11-QA revision C CSR SELECTION

| NUMBER OF MEMORY MODULE | JUMPER POSITION J5 | J7 | J9 | J11 | CSR REGISTER ADDRESS |
|---|---|---|---|---|---|
| 1ST | IN | IN | IN | IN | 17772100 |
| 2ND | OUT | IN | IN | IN | 17772102 |
| 3RD | IN | OUT | IN | IN | 17772104 |
| 4TH | OUT | OUT | IN | IN | 17772106 |
| 5TH | IN | IN | OUT | IN | 17772110 |
| 6TH | OUT | IN | OUT | IN | 17772112 |
| 7TH | IN | OUT | OUT | IN | 17772114 |
| 8TH | OUT | OUT | OUT | IN | 17772116 |
| 9TH | IN | IN | IN | OUT | 17772120 |
| 10TH | OUT | IN | IN | OUT | 17772122 |
| 11TH | IN | OUT | IN | OUT | 17772124 |
| 12TH | OUT | OUT | IN | OUT | 17772126 |
| 13TH | IN | IN | OUT | OUT | 17772130 |
| 14TH | OUT | IN | OUT | OUT | 17772132 |
| 15TH | IN | OUT | OUT | OUT | 17772134 |
| 16TH | OUT | OUT | OUT | OUT | 17772136 |

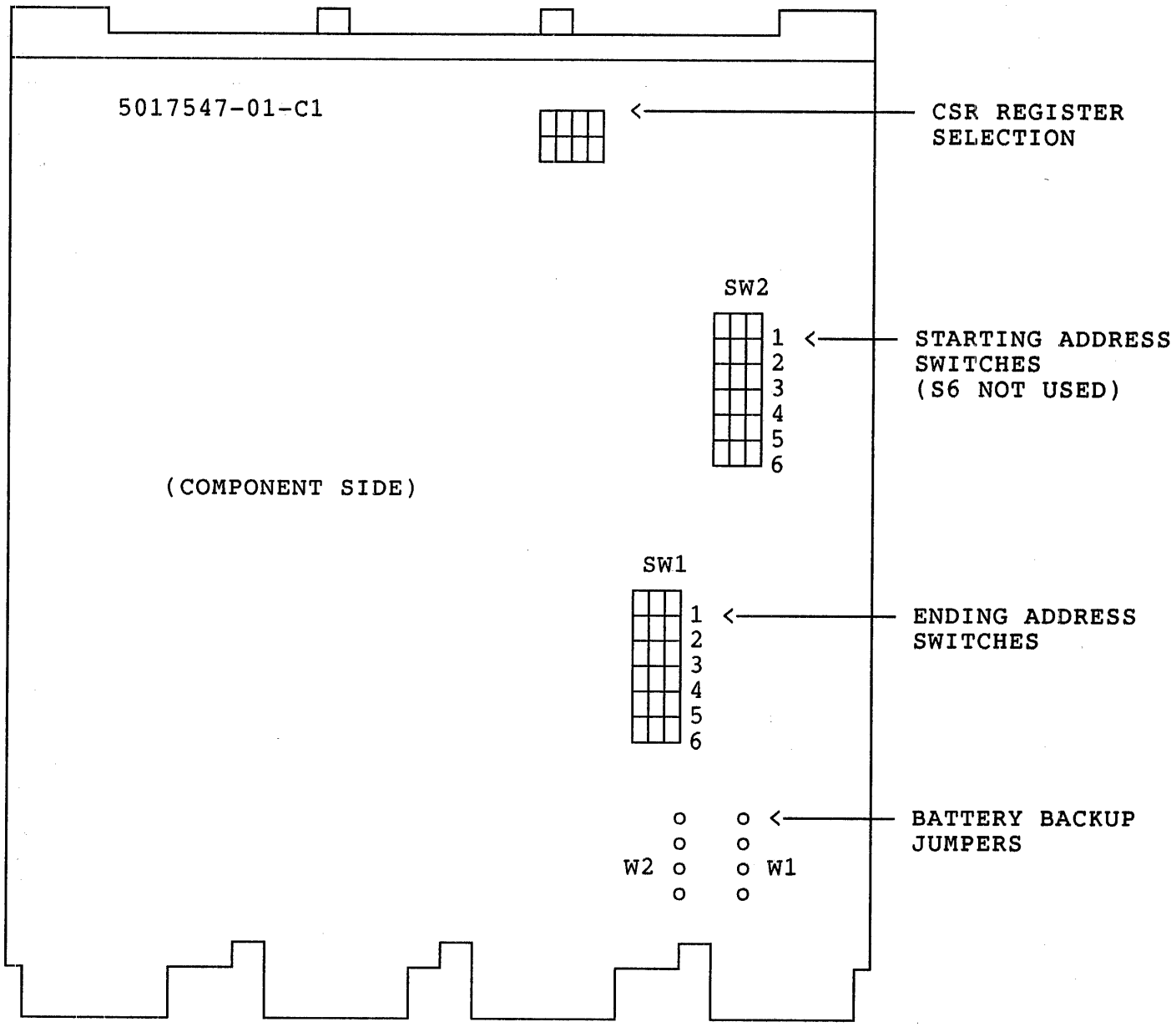
MSV11-QA revision C STARTING AND ENDING ADDRESS

| DESIRED STARTING ADDRESS | SW2 SWITCH POSITION | SW1 SWITCH POSITION | STARTING ENDING ADDRESS | ENDING SWITCH POSITION |
|---|---|---|---|---|
| IN KBYTE | 12345 | 6 | IN KBYTE | 12345 |
| 0 | 00000 | 0 | 128 | 11111 |
| 128 | 11111 | 1 | 256 | 01111 |
| 256 | 01111 | 1 | 384 | 10111 |
| 384 | 10111 | 1 | 512 | 00111 |
| 512 | 00111 | 1 | 640 | 11011 |
| 640 | 11011 | 1 | 768 | 01011 |
| 768 | 01011 | 1 | 896 | 10011 |
| 896 | 10011 | 1 | 1024 (1MB) | 00011 |
| 1024 (1MB) | 00011 | 1 | 1152 | 11101 |
| 1152 | 11101 | 1 | 1280 | 01101 |
| 1280 | 01101 | 1 | 1408 | 10101 |
| 1408 | 10101 | 1 | 1536 | 00101 |
| 1536 | 00101 | 1 | 1664 | 11001 |
| 1664 | 11001 | 1 | 1792 | 01001 |
| 1792 | 01001 | 1 | 1920 | 10001 |
| 1920 | 10001 | 1 | 2048 (2MB) | 00001 |
| 2048 (2MB) | 00001 | 1 | 2176 | 11110 |
| 2176 | 11110 | 1 | 2304 | 01110 |
| 2304 | 01110 | 1 | 2432 | 10110 |
| 2432 | 10110 | 1 | 2560 | 00110 |
| 2560 | 00110 | 1 | 2688 | 11010 |
| 2688 | 11010 | 1 | 2816 | 01010 |
| 2816 | 01010 | 1 | 2944 | 10010 |
| 2944 | 10010 | 1 | 3072 (3MB) | 00010 |
| 3072 (3MB) | 00010 | 1 | 3200 | 11100 |
| 3200 | 11100 | 1 | 3328 | 01100 |
| 3328 | 01100 | 1 | 3456 | 10100 |
| 3456 | 10100 | 1 | 3584 | 00100 |
| 3584 | 00100 | 1 | 3712 | 11000 |
| 3712 | 11000 | 1 | 3840 | 01000 |
| 3840 | 00010 | 1 | 3968 | 10000 |
| 3968 | 10000 | 1 | 4096 (4MB) | 00000 |

1 = off position (open)
0 = on position (closed)

NOTE: Switch S6 of SW2 is not used. For a memory starting address of 0, switch S6 of SW1 should be set to 0 (on). For all other starting addresses S6 of SW1 should be off (1).

MSV11-QA revision C




Jumpers and Switches
(Drawing not to scale)

## COMPARISON OF SIMILAR FUNCTION JUMPERS

| JUMPER FUNCTION | SELECTABILITY REV-A | SELECTABILITY REV-A | ENABLED CONFIGURATION | DISABLED CONFIGURATION |
|---|---|---|---|---|
| DISABLE BLOCK MODE | YES | NO | W1 IN (J11 TO J12)<br>W2 OUT | W2 IN (J11 TO J10)<br>W1 OUT |
| DISABLE PARITY MEMORY | YES | NO | JUMPER B IN (J14 TO 13)<br>JUMPER A OUT | JUMPER A IN (J14 TO J5)<br>JUMPER B OUT |
| DISABLE PARITY DETECTION | YES | NO | JUMPER H IN (J2 TO J3)<br>JUMPER G OUT | JUMPER G IN (J1 TO J2)<br>JUMPER H OUT |
| DISABLE 22-BIT ADDRESSING | YES | NO | JUMPER L IN (J8 TO J7)<br>JUMPER K OUT | JUMPER K IN (J8 TO J9)<br>JUMPER L OUT |
| BATTERY BACKUP SUPPORT | NO | YES | W3 AND W1 IN WITH W4 AND W1 OUT (revision C ONLY) | W4 AND W1 IN WITH W3 AND W1 OUT (revision C ONLY) |

BATTERY BACKUP CONFIGURATION

NOTE: JUMPERS W4 AND W2 ARE LOCATED BETWEEN E21 AND E13 IN THE LOWER RIGHT HAND CORNER OF THE COMPONENT SIDE OF THE MODULE. FOR THE ACTUAL LOCATION OF THESE AND ALL OTHER JUMPERS PLEASE REFER TO THE USERS GUIDE.

| uNOTE # 032 |
|---|

| Title: KXT11-C Parallel I/O Programming | Date: 28-JUN-85 |
|---|---|
| Originator: Scott Tincher | Page 1 of 42 |

The KXT11-CA is a single board computer that provides the user with flexible I/O programming options. One of the onboard programmable devices is a 20 line parallel I/O port (PIO). This note will describe the operation of the PIO and will provide some programming examples. Since the DIGITAL operating system MicroPower/Pascal provides a device handler for the PIO, the programming examples included in this note will be written in MACRO-11 for the user who wishes to program the PIO in MACRO-11. The example programs assume the user is familiar with the KXT11-C Software Toolkit for RT-11 or RSX.

## FEATURES/CAPABILITIES

The PIO of the KXT11-C supplies the following features:

- Two 8-bit, double buffered, bidirectional I/O ports
- A 4-bit special purpose I/O port
- Four handshake modes
- REQUEST signal for utilizing the DMA controller
- Pattern recognition logic
- Three independent 16-bit counter/timers

The two 8-bit ports (A and B) are identical except that Port B can provide external access to Counter/Timers 1 and 2. Each port may be configured under program control as a single or double buffered port with handshake logic or as a bit port for control applications. Pattern recognition logic is also included in each port. This logic allows interrupt generation whenever a specific pattern is recognized. Ports A and B may be linked to form a 16-bit port with handshake.

When Port A or B is used as a port with handshake the control lines are supplied by a special 4-bit port (Port C). If no handshake lines are required then Port C may be used as a bit port. Port C also provides external access to Counter/Timer 3 and a REQUEST line that allows the PIO to utilize the DMA controller when transfering data.

The PIO supplies three identical 16-bit Counter/Timers. These Counter/Timers operate at a frequency of 2 MHz which provides a resolution of 500 ns. Each Counter/Timer may operate with one of three output duty cycles: pulse, one-shot, or square-wave. In addition, each unit may operate as retriggerable or non-retriggerable.

## REGISTER DESCRIPTION

The following section provides a brief description of the registers of the PIO.

## MASTER CONTROL REGISTERS

There are two registers that control the overall function of the PIO, the Master Interrupt Control Register and the Master Configuration Control Register.

### Master Interrupt Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Address = 177000

Bit 7: Master Interrupt Enable (MIE)

0 - Inhibits this device from requesting an interrupt or responding to an interrupt acknowledge.
1 - Allows interrupt logic to operate normally.

Bits 6,5,4,3,2,1: These bits must be programmed to zero.

Bit 0: Reset

0 - Clears the reset bit and allows the other registers to function properly.
1 - Resets the device. While this bit is 1 reads of other registers will be 0 and writes to other registers will be ignored. This bit is cleared only by writing a 0 to the RESET bit.

Master Configuration Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Address = 177002

Bit 7: Port B Enable (PBE)

0 - Inhibits Port B from issuing an interrupt request and forces tth Port B I/O lines into a high impedance state.
1 - Allows Port B to operate normally.

Bit 6: Counter/Timer 1 Enable (CT1E)

0 - Inhibits Counter/Timer 1 from issuing an interrupt request and clears the Count In Progress (CIP) flag. All trigger inputs are ignored.
1 - Allows Counter/Timer 1 to operate normally.

Bit 5: Counter/Timer 2 Enable (CT2E)

Provides the same functions for Counter/Timer 2 that CT1E does for Counter/Timer 1.

Bit 4: Port C and Counter/Timer 3 Enable (PCE) and (CT3E)

Provides the same functions for Port C that PBE does for Port B and the same functions for Counter/Timer 3 that CT1E does for Counter/Timer 1.

Bit 3: Port Link Control (PLC)

0 - Allows Ports A and B to operate independently.
1 - Links Ports A and B to form a 16-bit port. In this mode Port A's handshake and command and status registers are used. Port B is specified as a bit port. This bit must be set before the ports are enabled.

Bit 2: Port A Enable (PAE)

Provides the same functions for Port A that PBE provides for Port B.

Bits 1,0: Counter/Timer Link Controls

These two bits specify how Counter/Timers 1 and 2 are linked according to the following table:

| Bit 1 | Bit 0 | |
|---|---|---|
| 0 | 0 | Counter/Timers are independent |
| 0 | 1 | C/T 1's output (inverted) gates C/T 2 |
| 1 | 0 | C/T 1's output (inverted) triggers C/T 2 |
| 1 | 1 | C/T 1's output (inverted) is C/T 2's count input |

The Counter/Timers must be linked before they are enabled.

PORT SPECIFICATION REGISTERS

Ports A and B both utilize the following port specification registers:

Port Mode Specification Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Port A = 177100
Port B = 177120

A RESET forces all of these bits to 0. All bits are read/write.

Bits 7,6: Port Type Select

These two bits specify the port type as defined by the following table:

| Bit 7 | Bit 6 | |
|---|---|---|
| 0 | 0 | Bit port (No handshake) |
| 0 | 1 | Input port with handshake |
| 1 | 0 | Output port with handshake |
| 1 | 1 | Bidirectional port with handshake |

Bit 5: Interrupt on Two Bytes (ITB)

0 - Indicates that Interrupt Pending (IP) should be set when one byte of data is available for transfer. For an input port IP is set when the Input Data Register is full. For an output port IP is set when the Output Data Register is empty.

1 - Indicates that IP should be set when two bytes of data are available for transfer. For an input port IP is set when both the Input Data Register and the Input Buffer Register are full. For an output port IP is set when both the Output Data Register and the Output Data Buffer are empty.

This bit must be set to zero for ports specified as bit ports, single-buffered ports, or bidirectional ports.

Bit 4: Single Buffered (SB)

0 - Indicates that the port is double-buffered.
1 - Indicates the the port is single-buffered.

This bit is always 0 for bit ports.

Bit 3: Interrupt on Match Only (IMO)

0 - Port operates normally.
1 - An interrupt is generated when the data moved into the Input Data Register or out of the Output Data Register matches the pattern specification.

Bits 2,1: Pattern Mode Specification Bits

These bits define the operation of the pattern recognition logic as shown by the following table:

| Bit 2 | Bit 1 | |
|---|---|---|
| 0 | 0 | Disable Pattern Match |
| 0 | 1 | AND Mode |
| 1 | 0 | OR Mode |
| 1 | 1 | OR-Priority Encoded Vector Mode |

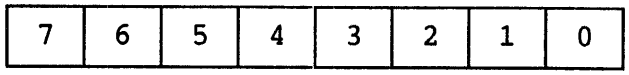
Bit 0: Latch on Pattern Match (LPM) or Deskew Timer Enable (DTE)

When a port is used as a bit port the LPM function is used. When a port with handshake is used the DTE function is used.

LPM:
0 - Pattern matches are detected but the data read from the port follows the port pins.
1 - When a pattern match is detected the input data at the port is latched.

DTE:
0 - The deskew timer is not activated.
1 - The deskew timer is activated to perform delay functions as set in the Port Handshake Specification Register.

Port Handshake Specification Register




Port A = 177102
Port B = 177122

These bits are ignored if the port is a bit port. A RESET forces all bits to 0. All bits are read/write.

Bits 7,6: Handshake Type Specification Bits

These bits define the type of handshake a port will use as shown by the following table:

| Bit 7 | Bit 6 | |
|---|---|---|
| 0 | 0 | Interlocked Handshake |
| 0 | 1 | Strobed Handshake |
| 1 | 0 | Pulsed Handshake |
| 1 | 1 | 3-Wire Handshake |

The Pulsed and 3-Wire Handshakes must not be specified for bidirectional ports. Only one port at a time may use the Pulsed Handshake. If one port uses the 3-Wire Handshake the other port must be specified as a bit port.

Bits 5,4,3: REQUEST/WAIT Specification Bits (RWS)

The WAIT function is not implemented on the KXT11-C. These bits define the utilization of the REQUEST line as shown by the following table:

| Bit 5 | Bit 4 | Bit 3 | |
|---|---|---|---|
| 0 | 0 | 0 | REQUEST disabled |
| 0 | 0 | 1 | Not supported |
| 0 | 1 | 1 | Not supported |
| 1 | 0 | 0 | Special REQUEST |
| 1 | 0 | 1 | Output REQUEST |
| 1 | 1 | 1 | Input REQUEST |

Only Port A may use the REQUEST capability - Port B must be programmed as a bit port.

Bits 2,1,0: Deskew Time Specification Bits

These bits specify the amount of deskew time to be provided for output data. They define the minimum number of Peripheral Clock (PCLK) cycles of delay between the output of a new byte of data and the handshake logic indicating that new data is available. PCLK = 250 ns. 0 PCLK cycles are chosen by setting DTE=0 in the Port Mode Specification Register.

| Bit 2 | Bit 1 | Bit 0 | PCLK cycles |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 4 |
| 0 | 1 | 0 | 6 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 10 |
| 1 | 0 | 1 | 12 |
| 1 | 1 | 0 | 14 |
| 1 | 1 | 1 | 16 |

Port Command and Status Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Port A = 177020
Port B = 177022

A RESET forces ORE to 1 and all other bits to 0. All bits are readable and four are writeable.

Bit 7: Interrupt Under Service (IUS)

0 - Cleared to indicate that the port is not servicing an interrupt.
1 - Indicates that that the port has been recognized by an interrupt acknowledge sequence.

Bit 6: Interrupt Enable (IE)

0 - Interrupt logic disabled. The port is unable to request an interrupt or to respond to an interrupt acknowledge.
1 - Interrupt logic operates normally.

Bit 5: Interrupt Pending (IP)

0 - Cleared to indicate that the port does not require service.
1 - Set to indicate the port needs service because of a pattern match, a handshake, or an error.

Bits 7, 6, and 5 are written using the following codes:

| Bit 7 | Bit 6 | Bit 5 | |
|---|---|---|---|
| 0 | 0 | 0 | Null code |
| 0 | 0 | 1 | Clear IP and IUS |
| 0 | 1 | 0 | Set IUS |
| 0 | 1 | 1 | Clear IUS |
| 1 | 0 | 0 | Set IP |
| 1 | 0 | 1 | Clear IP |
| 1 | 1 | 0 | Set IE |
| 1 | 1 | 1 | Clear IE |

Bit 4: Interrupt Error (ERR)

This bit is set to 1 when using a a bit port with pattern match enabled if a second match occurs before the previous match is acknowledged. This is a read-only bit.

Bit 3: Output Data Register Empty (ORE)

A status bit that indicates when an output port's Output Data Register is empty. This bit can only be cleared by writing to the data register. This is a read-only bit.

Bit 2: Input Data Register Full (IRF)

A status bit that indicates if an input port's Input Data Register is full. This bit can only be cleared by reading the Input Data Register. This is a read-only bit.

Bit 1: Pattern Match Flag (PMF)

If the port pattern match logic is enabled this bit will indicate when a match is detected. This bit is read-only.

Bit 0: Interrupt on Error (IOE)

0 - Disables the generation of an interrupt if an error occurs within the pattern match logic.
1 - Enables the generation of an interrupt if an error occurs within the pattern match logic.

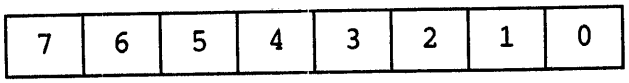
This bit is valid only for bit ports with pattern match logic enabled. It is ignored by ports with handshake and should be programmed to 0.

BIT PATH DEFINITION REGISTERS

Each port has a set of these registers. Only the four least significant bits are valid in the port C registers.

## Data Path Polarity Registers

These registers define whether the bits in a port are inverting or non-inverting. These bits are cleared by a RESET and are read/write.

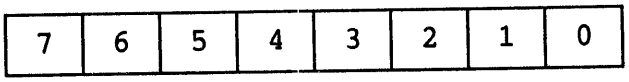



Port A = 177104
Port B = 177124
Port C = 177012

0 = Non-inverting
1 = Inverting

## Date Direction Registers

These registers are ignored by ports with handshake. For bit ports they define the data direction for each bit. These bits are cleared by a RESET and are read/write.

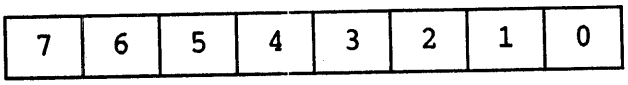



Port A = 177106
Port B = 177126
Port C = 177014

0 = Output bit
1 = Input bit

## Special I/O Control Registers

These registers supply special characteristics to the port's data paths. A RESET clears all bits to 0. All bits are read/write.



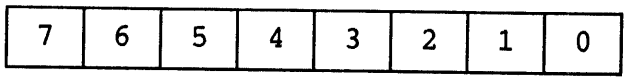

Port A = 177110
Port B = 177130
Port C = 177016

0 = Normal Input or Output
1 = Input with 1's catcher

## PATTERN DEFINITION REGISTERS

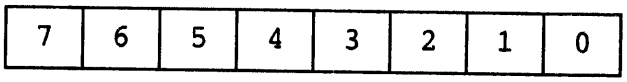
These registers are used collectively to specify the match pattern for a port. A RESET clears all bits to 0. All bits are read/write.

### Pattern Polarity Registers (PPR)



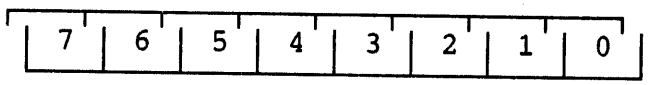

Port A = 177112
Port B = 177132

### Pattern Transition Registers (PTR)




Port A = 177114
Port B = 177134

### Pattern Mask Registers (PMR)




Port A = 177116
Port B = 177136

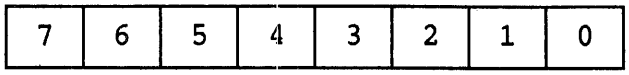
The pattern specification for each bit is shown in the following table:

| PPR | PTR | PMR | |
|---|---|---|---|
| 0 | 0 | 0 | Bit masked off |
| 0 | 1 | 0 | Any transition |
| 1 | 0 | 0 | Zero |
| 1 | 0 | 1 | One |
| 1 | 1 | 0 | One to zero transition |
| 1 | 1 | 1 | Zero to one transition |

## PORT DATA REGISTERS

Ports A and B have a data path that consists of three registers: an Input Data Register, and Output Data Register, and a Buffer Register. The Buffer Register is used to buffer the input or output data of a port with handshake. It is also used by bit ports to latch data when pattern matching is enabled.

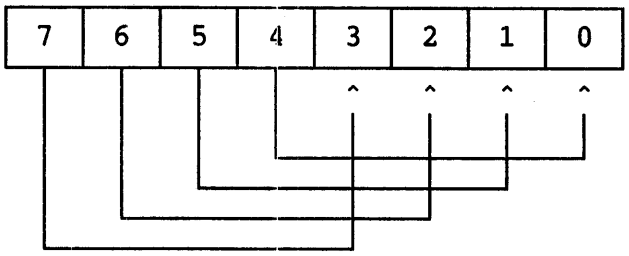
Port A and B Data Registers




Port A = 177114
Port B = 177134

The Port C data register consists of two registers: an Input Data register and an Output Data register. Because Port C is only 4 bits wide the least significant four bits of the data register are used for the data path. The four most significant bits are used as a write-protect mask for the four least significant bits.

Port C Data Register




Bits 7,6,5,4: 0 = Writing of corresponding LSB enabled
1 = Writing of corresponding LSB inhibited

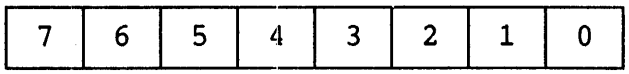
Port C = 177036

COUNTER/TIMER CONTROL REGISTERS

Each counter/timer has a set of Counter/Timer Control registers to specify the operation of the counter/timers.

Counter/Timer Mode Specification Registers

These registers define the mode of operation for the counter/timers and specify the external control and status lines to provide for it. A RESET clears all bits to 0. All bits are read/write.




Counter/Timer 1 = 177070
Counter/Timer 2 = 177072
Counter/Timer 3 = 177074

Bit 7: Continuous/Single Cycle

0 - When the counter reaches 0 the countdown sequence is terminated.
1 - When the counter reaches 0 the time constant value is reloaded and the countdown sequence is repeated.

Bit 6: External Output Enable (EOE)

0 - No external access.
1 - The output of the counter/timer is available on the I/O pin associated with that counter/timer. (See table 2 for pin assignments.)

Bit 5: External Count Enable (ECE)

0 - No external access.
1 - The I/O line of the port associated with the counter/timer is used as an external counter input. (See table 2 for pin assignments.)

Bit 4: External Trigger Enable (ETE)

0 - No external access
1 - The I/O line of the port associated with the counter/timer is used as a trigger input to the counter/timer. (See table 2 for pin assignments.)

Bit 3: External Gate Enable (EGE)

0 - No external access
1 - The I/O line of the port associated with the counter/timer is used as an external gate input to the counter/timer. This allows the external line to suspend/continue the countdown in progress by toggling the line. (See table 2 for pin assignments.)

Bit 2: Retrigger Enable Bit (REB)

0 - Triggers (external or internal) that occur during a countdown sequence are ignored.
1 - Triggers that occur during a countdown sequence cause a new countdown to begin.

Bits 1,0: Output Duty Cycle Selects

These two bits select the output duty cycle as shown in the following table:

| Bit 1 | Bit 0 | |
|---|---|---|
| 0 | 0 | Pulse Output |
| 0 | 1 | One-Shot Output |
| 1 | 0 | Square Wave Output |
| 1 | 1 | Do not use |

See figure 2 for a description of each output duty cycle.

## Counter/Timer Command and Status Registers

Each counter/timer contains a command and status register for controlling the operation of the counter/timer. A RESET clears all bits to 0.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Counter/Timer 1 = 177024
Counter/Timer 2 = 177026
Counter/Timer 3 = 177030

Bit 7: Interrupt Under Service (IUS)

The operation of the this bit is the same as the IUS bit described on page *.

Bit 6: Interrupt Enable (IE)

The operation of the this bit is the same as the IE bit described on page *.

Bit 5: Interrupt Pending (IP)

This bit is set to 1 to indicate that the counter/timer needs to be serviced. It is automatically set to 1 each time the counter/timer reaches its terminal count.

The IUS, IE, IP bits are written by using the codes shown in the following table:

| Bit 7 | Bit 6 | Bit 5 | |
|---|---|---|---|
| 0 | 0 | 0 | Null code |
| 0 | 0 | 1 | Clear IP and IUS |
| 0 | 1 | 0 | Set IUS |
| 0 | 1 | 1 | Clear IUS |
| 1 | 0 | 0 | Set IP |
| 1 | 0 | 1 | Clear IP |
| 1 | 1 | 0 | Set IE |
| 1 | 1 | 1 | Clear IE |

Bit 4: Interrupt Error (ERR)

This bit is set to indicate that the counter/timer has reached a terminal count before the previous terminal count has been serviced.

Bit 3: Read Counter Control (RCC)

Writing this bit to a 1 causes the contents of the Counter/Timer Current Count Register (CCR), which normally follows the down-counter, to be frozen until the least-significant byte of the CCR is read.

Bit 2: Gate Command Bit (GCB)

0 - Halts the countdown sequence.
1 - Starts or resumes the countdown sequence.

Bit 1: Trigger Command Bit (TCB)

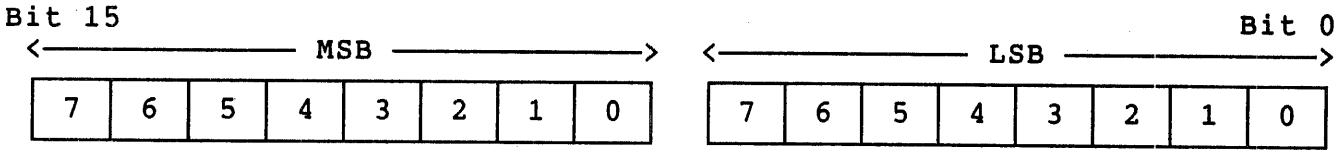
When written with a 1, this bit causes the down-counter to be loaded with the time constant value and a countdown sequence to be initiated.

Bit 0: Count in Progress (CIP)

This status bit is set to 1 to indicate that a countdown sequence is in progress. It is automatically set to 0 when the down-counter reaches 0.

Counter/Timer Time Constant Registers

These registers contain the time constant value that is loaded into the down-counter when a trigger is detected. These registers are 16 bits wide and are accessed as two 8-bit registers. (Bit 7 of the most-significant byte is bit 15 of the Time Constant register). A RESET does not effect these registers.

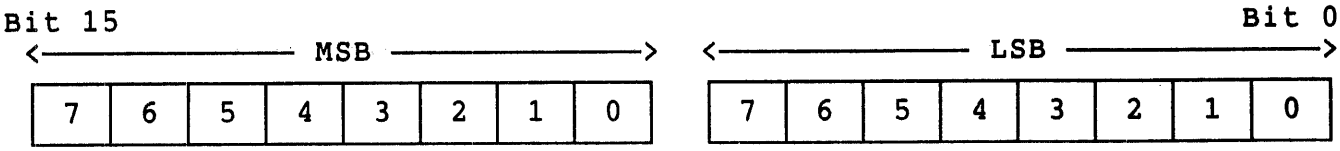



Counter/Timer 1 MSB = 177054
Counter/Timer 2 MSB = 177060
Counter/Timer 3 MSB = 177064

Counter/Timer 1 LSB = 177056
Counter/Timer 2 LSB = 177062
Counter/Timer 3 LSB = 177066

Counter/Timer Current Count Registers

These 16-bit registers follow the contents of the appropriate down-counter until a 1 is written into the RCC register. At that time the contents of the CCR are frozen until the least-significant byte of the CCR is read. A RESET forces the CCR to follow the down-counter again.




Counter/Timer 1 MSB = 177040
Counter/Timer 2 MSB = 177044
Counter/Timer 3 MSB = 177050

Counter/Timer 1 LSB = 177042
Counter/Timer 2 LSB = 177046
Counter/Timer 3 LSB = 177052

INTERRUPT RELATED REGISTERS

These registers contain the interrupt vectors output during an interrupt acknowledge sequence. Registers are provided for Port A, Port B, one to be shared by the Counter/Timers. Another register is provided to indicate which devices need service in a polled environment.

Interrupt Vector Registers

These vectors contain the vector output when the source of an interrupt is acknowledged. If Master Interrupt Enable = 1 then the vector register returns status when read according to the following table:

Port Vector Status
OR-Priority Encoded Vector Mode:

| Bit 3 | Bit 2 | Bit 1 | |
|---|---|---|---|
| x | x | x | Encodes the number of the highest-priority bit with a match |

All other modes:

| Bit 3 ORE | Bit 2 IRF | Bit 1 PMF | |
|---|---|---|---|
| 0 | 0 | 0 | Normal |
| | | | Error |

Counter/Timer Status

| Bit 2 | Bit 1 | |
|---|---|---|
| 0 | 0 | Counter/Timer 3 |
| 0 | 1 | Counter/Timer 2 |
| 1 | 0 | Counter/Timer 1 |
| 1 | 1 | Error |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Port A = 177004
Port B = 177006
Counter/Timers = 177010

The native firmware of the KXT11-C initializes these interrupt vectors to the following values:

Port A = 200
Port B = 204
Counter/Timers = 210

Current Vector Register

When read, this register returns the interrupt vector that would have been output by the device during an interrupt acknowledge cycle if its IEI input had been high. The vector returned corresponds to the highest priority IP independent of IUS. The order of priority is: Counter/Timer 3, Port A, Counter/Timer 2, Port B, Counter/Timer 1. If no enabled interrupts are pending, a pattern of ones is returned. This is useful in a polled environment.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Address = 177076

I/O BUFFER CONTROL REGISTER

The PIO chip is protected from the connector by a set of buffers. These buffers comply with the IEEE 488 electrical standards. The buffers allow the ports to configured as inputs or outputs. They also allow the ports to be configured as open collectors or active pull-ups.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Address = 177140

Bit 15: PCTT (Cleared on RESET)

0 - Configures the Port C drivers for open collector
1 - Configures the Port C drivers for active pull-up

Bit 14: PABTT (Cleared on RESET)

0 - Configures the Port A and B drivers for open collector
1 - Configures the Port A and B drivers for active pull-up

Bits 13:10: PC DIR (Cleared on RESET)

0 - Port C bit is a receiver
1 - Port C bit is a driver

Bit 9: PAHN DIR (Cleared on RESET)

0 - Port A high nibble bits (4:7) are receivers
1 - Port A high nibble bits are drivers

Bit 8: PALN DIR (Cleared on RESET)

0 - Port A low nibble bits (0:3) are receivers
1 - Port A low nibble bits are drivers

Bits 7:0: PB DIR (Cleared on RESET)

0 - Port B bit is a receiver
1 - Port B bit is a driver

## PROGRAMMING THE I/O PORTS

This section will describe how to program the I/O ports and provide example programs. In particular this section will describe how to use the I/O ports in the following modes: as bit ports, as ports with handshake, in 16-bit linked mode, and with the DMA controller. The use of the pattern recognition logic will also be discussed.

### Programming the I/O Ports as Bit Ports

Using the I/O ports as bit ports provides up to 20 lines for control and status. Each bit in ports B and C may be independently configured to be an input or an output. Port A must be configured on a nibble (4-bit) basis.

Programming the PIO as a bit port is straight-forward. First, the Port Mode Specification Register is used to select the port as a bit port with/without pattern matching. Then the Bit Path Definition Registers are used to determine the polarity, direction, and special characteristics of the bits of the port. If pattern recognition is enabled the Pattern Definition Registers must also be initialized. It is then a simple matter to write to the output data buffer to provide the correct control signals and to read the input data buffer to monitor status.

The following program provides an example for using the PIO in the bit mode:

```
        .TITLE  PIO1.MAC
;+
;  This program provides an example of how to program the PIO's
;  I/O ports as bit ports.  This program utilizes the PIO
;  loopback connector (Part #H3021 or 54-16227) which makes the
;  following connections:
;
;                       A0  --  B0
;                       A1  --  B1
;                           .
;                           .
;                       A7  --  B7
;                       C0  --  C3
;                       C1  --  C2
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD PIO1.SAV
;  EXECUTE
;  !ODT
;  !
;  !001152
;  !R2/000000
;  !1154/041101
;  !001156/042103
;  !001160/043105
;  !001162/177507
;  !001164/041101
;  !001166/042103
;  !001170/043105
;  !001172/000107
;  !001174/000000
;  !
;  EXIT
;
;  A non-zero result in R2 indicates that an error has occurred.  (Try
;  running the test without the loopback connector).  Location 1154 is
;  the beginning of the output buffer.  Location 1164 is the beginning
;  of the input buffer.
;-
        ; Register Assignments

        MIC     --  177000
        MCC     --  177002
```

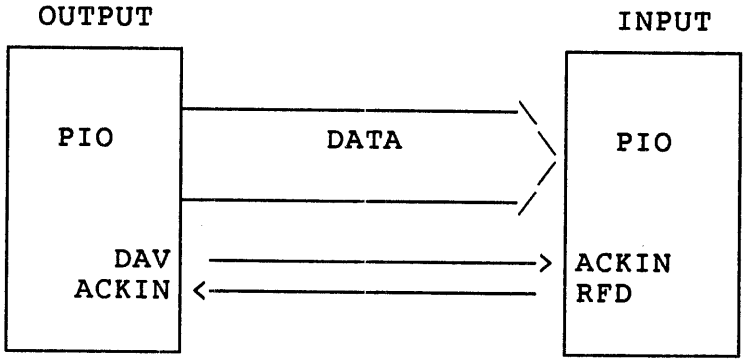
```
        PAMODE  =:  177100
        PAPOL   =:  177104
        PADDIR  =:  177106
        PASIO   =:  177110
        PADATA  =:  177032

        PBMODE  =:  177120
        PBPOL   =:  177124
        PBDDIR  =:  177126
        PBSIO   =:  177130
        PBDATA  =:  177034

        IOCNTL  =:  177140

START::
        MTPS    #340            ; Inhibit recognition of
                                ;  interrupts

        ;  Initialize PIO
        MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                ;  requests
        CLRB    MIC             ; Enable device (interrupts still
                                ;  inhibited)

        ;  Set-up Port A
        CLRB    PAMODE          ; Port A:  bit port, no pattern match
        CLRB    PAPOL           ; Port A bits are non-inverting
        CLRB    PADDIR          ; Port A bits are output bits
        CLRB    PASIO           ; Normal output

        ;  Set-up Port B
        CLRB    PBMODE          ; Port B:  bit port, no pattern match
        CLRB    PBPOL           ; Port B bits are non-inverting
        MOVB    #377,PBDDIR     ; Port B bits are input bits
        CLRB    PBSIO           ; Normal input

        ;  Set-up the PIO buffers
        MOV     #1400,IOCNTL    ; configure the PIO buffers for
                                ;  A=output and B=input

        ;  Initialize GPRs
        MOV     #OUTBUF,R0      ; Point to data to be output
        MOV     #INBUF,R1       ; Point to input data buffer
        CLR     R2              ; R2 will indicate error status

        ;  Flush input buffer
        TSTB    PBDATA

        ;  Enable Ports A and B and send the data
        MOVB    #204,MCC        ; Enable ports A and B
```

```
1$:     MOVB    (R0)+,PADATA    ; Move data out of Port A
        NOP                     ;  .
                                ;  .
        MOVB    PBDATA,(R1)+    ; and into Port B

        ;  Test to see if done
        TSTB    (R0)            ; IF (R0) is positive
        BPL     1$              ; THEN transfer another byte
                                ; ELSE check if data is valid

        ;  Compare original data with received data
        MOV     #OUTBUF,R0      ; Point to output data buffer
        MOV     #INBUF,R1       ; Point to input data buffer

        ;  Test to see if done
2$:     TSTB    (R0)            ; IF (R0) is negative
        BMI     3$              ;  THEN done comparing
                                ;  ELSE do another compare
        CMPB    (R0)+,(R1)+     ; Compare bytes
        BEQ     2$              ; IF bytes are equal
                                ;  THEN test another pair
                                ;  ELSE indicate error
        INC     R2              ; A non-zero value of R2 indicates
                                ;  an error
3$:     BR                      ; Branch here upon completion

OUTBUF: .BYTE   101,102,103,104,105,106,107,-1
        .EVEN
INBUF:  .BLKB   7

        .END    START
```

Programming the I/O Ports as Ports with Handshake

Ports A and B may be configured as ports with handshake to facilitate transferring data on a byte-by-byte basis. Port C is used to provide the handshake lines. In addition, Port C may use the REQUEST line to utilize a DMA controller to transfer the data. See table 1 for a description of the Port C handshake lines. Figure 1 shows how two PIOs can be connected directly together to transfer data and the handshake lines that are utilized.

PIO Handshake Lines




- Figure 1 -

The handshakes that are available are: Interlocked, Strobed, Pulsed, and 3-Wire. A short description of each handshake type follows:

When using the Interlocked Handshake any action by the PIO must be acknowledged by the external device before the next action can take place. In other words, an output port does not indicate that it has new data available until the external device indicates that it is ready for data. Likewise, and input port does not indicate that it is ready for new data until the external device indicates that the previous byte of data is no longer available, thereby acknowledging the input port's acceptance of the last byte.

The Strobed Handshake uses external logic to "strobe" data into or out of a port. In contrast to the Interlocked handshake, the signal indicating that the port is ready for another data transfer operates independently of the ACKIN input. External logic must ensure the data transfers at the appropriate speed.

The Pulsed Handshake is used to interface to mechanical devices which require data to be held for relatively long periods of time in order to be gated in or out of the device. The logic is the same as the Interlocked Handshake except that Counter/Timer 3 is linked to the handshake logic to add the appropriate delays to the handshake lines.

The 3-Wire Handshake may be used so that one output port can communicate to several input ports simultaneously. This is essentially the same as the Interlocked Handshake except that two individual lines are used to indicate when an input port is ready for data (RFD) and when it has accepted data (DAC). Because this handshake requires three lines only one port can use the 3-Wire Handshake at a time.

Port C Handshake Lines

| | Port C Bits | | | |
|---|---|---|---|---|
| Port A/B Configuration | Pin C3 | Pin C2 | Pin C1 | Pin C0 |
| Ports A & B = Bit Ports | Bit I/O | Bit I/O | Bit I/O | Bit I/O |
| Port A = Input or Output (Interlocked, Strobed, or Pulsed Handshake)* | RFD or DAV | ACKIN | REQUEST or Bit I/O | Bit I/O |
| Port B = Input or Output (Interlocked, Strobed, or Pulsed Handshake)* | REQUEST or Bit I/O | Bit I/O | RFD or DAV | ACKIN |
| Port A or B = Input Port (3-Wire Handshake) | RFD (Output) | DAV (Input) | REQUEST or Bit I/O | DAC (Output) |
| Port A or B = Output Port (3-Wire Handshake) | DAV (Output) | DAC (Input) | REQUEST or Bit I/O | RFD (Input) |
| Port A or B = Bidirectional (Interlocked or Strobed Handshake) | RFD or DAV | ACKIN | REQUEST or Bit I/O | IN/OUT |

* Both Ports A & B may be specified as input or output ports with the Interlocked, Strobed, or Pulsed Handshakes at the same time if neither uses REQUEST. Only one port can use the Pulsed Handshake at a time.

- Table 1 -

When Ports A and B are configured as ports with handshake they must also be configured as single- or double-buffered. Double-buffering a port allows more time for the interrupt service routine to respond to a data transfer. A second byte of data is input to or output from the port before the interrupt for the first byte is serviced. A single-buffered port is used where it is important to have byte-by-byte control over the transfer or where it is important to enter the interrupt service routine in a fixed amount of time after the data has been accepted/output.

The REQUEST line may also be used by ports with handshake. This control line enables the PIO to signal the DMA controller of the KXT11-C that the port wishes to transfer data without CPU intervention. The operation of the REQUEST line is dependent on the Interrupt on Two Bytes

(ITB) bit in the Port Mode Specification Register. If ITB = 0 then the REQUEST line goes active anytime a byte is available to transfer. If ITB = 1 then the REQUEST line does not assert until two bytes are available to transfer. The implementation of the PIO on the KXT11-C requires that only Port A be used for DMA transfers. Since the REQUEST line utilizes one of the Port C bits Port B must be programmed as a bit port when Port A uses the REQUEST facility.

The following example programs display the capabilities of the PIO used as a port with handshake:

```
        .TITLE  PIO2.MAC

;  This program demonstrates the ability of the PIO to transfer data
;  on a byte-by-byte basis.  The program uses the Interlocked
;  Handshake to transfer data from Port A to Port B.  Both ports are
;  configured as single-buffered.  The PIO loopback connector (part
;  #H3022 or 54-16227) or a functional equivalent is required to
;  successfully run this program.
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD PIO2.SAV
;  EXECUTE
;  !ODT
;  !
;  !001214
;  !1262/065151
;  !001264/066153
;  !001266/067155
;  !001270/070157
;  !001272/000377
;  !001274/065151
;  !001276/066153
;  !001300/067155
;  !001302/070157
;  !001304/000000
;  !
;  EXIT
;
;  This verifies that the contents of the output buffer (location 1262
;  were successfully transferred to the input buffer (location 1274).
;

        ; Register Assignments

        MIC     —   177000
```

```
            MCC     —   177002

            PAVEC   —   177004
            PASTAT  —   177020
            PADATA  —   177032
            PAMODE  —   177100
            PAHDSH  —   177102
            PAPOL   —   177104
            PASIO   —   177110

            PBVEC   —   177006
            PBSTAT  —   177022
            PBDATA  —   177034
            PBMODE  —   177120
            PBHDSH  —   177122
            PBPOL   —   177124
            PBSIO   —   177130

            PCPOL   —   177012
            PCDDIR  —   177014

            IOCNTL  —   177140

START::
            MTPS    #340            ; Inhibit recognition of interrupts

            MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                    ;  requests from the PIO
            CLRB    MIC             ; Enable device (interrupts still
                                    ;  inhibited)

            MOVB    #200,PAVEC
            MOV     #OUT,@#200      ; Set up Port A interrupt vector
            MOV     #340,@#202      ; ... and PSW

            MOVB    #204,PBVEC
            MOV     #IN,@#204       ; Set up Port B interrupt vector
            MOV     #340,@#206      ; ... and PSW

            ; Set-up Port A
            MOVB    #220,PAMODE     ; Port A: Output Port, single-buffered
            CLRB    PAHDSH          ; Use interlock handshake
            CLRB    PAPOL           ; Port A bits are non-inverting
            CLRB    PASIO           ; Normal output
            MOVB    #300,PASTAT     ; Enable Port A interrupts

            ; Set-up Port B
            MOVB    #120,PBMODE     ; Port B:  Input Port, single-buffered
            CLRB    PBHDSH          ; Use interlock handshake
            CLRB    PBPOL           ; Port B bits are non-inverting
            CLRB    PBSIO           ; Normal input
            MOVB    #300,PBSTAT     ; Enable Port B interrupts
```

```
        ; Set-up the Port C handshake lines.
        ;  All handshake lines are configured as inputs - even
        ;  if they aren't!
        MOVB    #377,PCDDIR     ; Port C bits are inputs

        ; Set-up the PIO buffers
        MOV     #165400,IOCNTL  ; configure the PIO buffers for A=out
                                ;  B=input, C0,C2=input, C1,C3=output

        ; Set-up data areas
        MOV     #OUTBUF,R0      ; Point to Output Buffer
        MOV     #INBUF,R1       ; Point to Input Buffer

        ; Enable Interrupts
        MOVB    #224,MCC        ; Enable ports A, B, and C
        MOVB    #200,MIC        ; Enable MIC
        MTPS    #0              ; Enable recognition of interrupts

        ; Start the first transfer
        MOVB    #200,PASTAT     ; Set IP to initiate a transfer

        BR                      ; Wait here for the interrupts

OUT::
        TSTB    (R0)            ; IF (R0) are negative
        BMI     1$              ;  THEN transfers are complete
                                ;  ELSE transfer another byte
        MOVB    (R0)+,PADATA    ; Move byte to the Port A output data
                                ;  register
        BR      2$
1$:     MOVB    #240,PASTAT     ; Clear IP when done
2$:     MOVB    #140,PASTAT     ; Clear IUS on each pass
        RTI

IN::
        MOVB    PBDATA,(R1)+    ; Move byte from Port B input data
                                ;  register
        MOVB    #140,PBSTAT     ; Clear IUS on each pass
        RTI

OUTBUF: .BYTE   151,152,153,154,155,156,157,160,-1
        .EVEN
INBUF:  .BLKB   10

        .END    START
```

```
        .TITLE  PIO3.MAC

;  This program is basically the same as PIO2.MAC with the
;  with the exception that the ports are double-buffered.
;  The PIO loopback connector (part #H3022 or 54-16227) or a
;  functional equivalent is required to successfully run this program.
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD PIO3.SAV
;  EXECUTE
;  !ODT
;  !
;  !001214
;  !1272/065151
;  !001274/066153
;  !001276/067155
;  !001300/070157
;  !001302/000377
;  !001304/065151
;  !001306/066153
;  !001310/067155
;  !001312/070157
;  !001314/000000
;  !
;  EXIT
;
;  This verifies that the contents of the output buffer (location 1272
;  were successfully transferred to the input buffer (location 1304).
;

        ; Register Assignments

        MIC     —  177000
        MCC     —  177002

        PAVEC   —  177004
        PASTAT  —  177020
        PADATA  —  177032
        PAMODE  —  177100
        PAHDSH  —  177102
        PAPOL   —  177104
        PASIO   —  177110

        PBVEC   —  177006
        PBSTAT  —  177022
        PBDATA  —  177034
        PBMODE  —  177120
```

```
PBHDSH  —  177122
PBPOL   —  177124
PBSIO   —  177130

PCPOL   —  177012
PCDDIR  —  177014

IOCNTL  —  177140

START::
        MTPS    #340            ; Inhibit recognition of interrupts

        MOVB    #1,MIC          ; Reset device and inhibit interrupt
        CLRB    MIC             ;  requests from the PIO
                                ; Enable device (interrupts still
                                ;  inhibited)

        MOVB    #200,PAVEC
        MOV     #OUT,@#200      ; Set up Port A interrupt vector
        MOV     #340,@#202      ; ... and PSW

        MOVB    #204,PBVEC
        MOV     #IN,@#204       ; Set up Port B interrupt vector
        MOV     #340,@#206      ; ... and PSW

        ; Set-up Port A
        MOVB    #240,PAMODE     ; Port A: Output Port, double-buffered
        CLRB    PAHDSH          ; Use interlock handshake
        CLRB    PAPOL           ; Port A bits are non-inverting
        CLRB    PASIO           ; Normal output
        MOVB    #300,PASTAT     ; Enable Port A interrupts

        ; Set-up Port B
        MOVB    #140,PBMODE     ; Port B:  Input Port, double-buffered
        CLRB    PBHDSH          ; Use interlock handshake
        CLRB    PBPOL           ; Port B bits are non-inverting
        CLRB    PBSIO           ; Normal input
        MOVB    #300,PBSTAT     ; Enable Port B interrupts

        ; Set-up the Port C handshake lines.
        ;  All handshake lines are configured as inputs - even
        ;  if they aren't!
        MOVB    #377,PCDDIR     ; Port C bits are inputs

        ; Set-up the PIO buffers
        MOV     #165400,IOCNTL  ; configure the PIO buffers for A=out
                                ;  B=input, C0,C2=input, C1,C3=output

        ; Set-up data areas
        MOV     #OUTBUF,R0      ; Point to Output Buffer
        MOV     #INBUF,R1       ; Point to Input Buffer
```

```
        ; Enable Interrupts
        MOVB    #224,MCC        ; Enable ports A, B, and C
        MOVB    #200,MIC        ; Enable MIC
        MTPS    #0              ; Enable recognition of interrupts

        ; Start the first transfer
        MOVB    #200,PASTAT     ; Set IP to initiate a transfer

        BR                      ; Wait here for the interrupts

OUT::
        TSTB    (R0)            ; IF (R0) are negative
        BMI     1$              ;  THEN transfers are complete
                                ;  ELSE transfer another byte
        MOVB    (R0)+,PADATA    ; Move 1st byte to the Port A output
                                ;  data register
        MOVB    (R0)+,PADATA    ; Move 2nd byte to the Port A buffer
                                ;  register
        BR      2$
1$:     MOVB    #240,PASTAT     ; Clear IP when done
2$:     MOVB    #140,PASTAT     ; Clear IUS on each pass
        RTI

IN::
        MOVB    PBDATA,(R1)+    ; Move 1st byte from Port B input data
                                ;  register
        MOVB    PBDATA,(R1)+    ; Move 2nd byte from Port B buffer
                                ;  register
        MOVB    #140,PBSTAT     ; Clear IUS on each pass
        RTI

OUTBUF: .BYTE   151,152,153,154,155,156,157,160,-1
        .EVEN
INBUF:  .BLKB   10

        .END    START
```

```
;  This example shows something a little more practical - one
;  KXT11-C transferring data to another.  Two programs follow:
;  one accepts data through Port B using the double-buffered
;  mode (PIO4I.MAC); the second one sends data out of Port A
;  using the double buffered mode (PIO4O.MAC).  In order to
;  successfully run these programs the KXT11-Cs must be connected
;  by a "straight-thru" ribbon cable which is given a half twist.
;  In other words, it should make the same connections that the
;  PIO loopback connector does. (A1-B1,A2-B2,...A7-B7,C0-C3,C1-C2).
;
;  Each program should be assembled and linked separately on the
;  development machine.  Then use the KUI utility of the KXT11-C
;  Software Toolkit to load the programs into the KXT11-Cs to execute
;  as shown in this example:
;
;  SET 3
;  LOAD PIO4I.SAV
;  EXECUTE
;  SET 2
;  LOAD PIO4O.SAV
;  EXECUTE
;  SET 3
;  !ODT
;  !
;  !001130
;  !1152/065151
;  !001154/066153
;  !001156/067155
;  !001160/070157
;  !001162/000000
;  !
;  EXIT
;
;  This verifies that the data was successfully transferred to
;  the input buffer of KXT11-C #3.
;
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
        .TITLE  PIO4I.MAC

        ; Register Assignments

        MIC     —  177000
        MCC     —  177002

        PBVEC   —  177006
        PBSTAT  —  177022
        PBDATA  —  177034
        PBMODE  —  177120
        PBHDSH  —  177122
        PBPOL   —  177124
        PBDDIR  —  177126
```

```
        PBSIO   —  177130

        PCDDIR  —  177014

        IOCNTL  —  177140

START::
        MTPS    #340            ; Inhibit recognition of interrupts
        MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                ;  requests from the PIO
        CLRB    MIC             ; Enable device (interrupts still
                                ;  inhibited)

        MOVB    #204,PBVEC
        MOV     #IN,@#204       ; Set up Port B interrupt vector
        MOV     #340,@#206      ; ... and PSW

        MOVB    #140,PBMODE     ; Port B:  Input Port, double-buffered
        CLRB    PBHDSH          ; Use interlock handshake
        CLR     PBPOL           ; Port B bits are non-inverting
        CLR     PBSIO           ; Normal input
        MOVB    #300,PBSTAT     ; Enable Port B interrupts

        MOVB    #377,PCDDIR     ; Port C bits are inputs

        MOV     #165400,IOCNTL  ; configure the PIO buffers for A=out
                                ;  B=input, C0,C2=input, C1,C3=output

        MOV     #INBUF,R1       ; Point to input data buffer

        MOVB    #220,MCC        ; Enable ports B and C
        MOVB    #200,MIC        ; Enable MIC

        MTPS    #0              ; Enable recognition of interrupts
        BR                      ; Wait here for the interrupts

IN::
        MOVB    PBDATA,(R1)+    ; Move 1st byte from Port B input data
                                ;  register
        MOVB    PBDATA,(R1)+    ; Move 2nd byte from Port B buffer
                                ;  register
        MOVB    #140,PBSTAT     ; Clear IUS on each pass
        RTI

INBUF:  .BLKB   10

        .END    START
```

```
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------

        .TITLE  PIO40.MAC

        ; Register Assignments

        MIC     —  177000
        MCC     —  177002

        PAVEC   —  177004
        PASTAT  —  177020
        PADATA  —  177032
        PAMODE  —  177100
        PAHDSH  —  177102
        PAPOL   —  177104
        PADDIR  —  177106
        PASIO   —  177110

        PCPOL   —  177012
        PCDDIR  —  177014

        IOCNTL  —  177140

START::
        MTPS    #340            ; Inhibit recognition of interrupts
        MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                ;  requests from the PIO
        CLRB    MIC             ; Enable device (interrupts still
                                ;  inhibited)

        MOVB    #200,PAVEC
        MOV     #OUT,@#200      ; Set up Port A interrupt vector
        MOV     #340,@#202      ; ... and PSW

        MOVB    #240,PAMODE     ; Port A: Output Port, double-buffered
        CLRB    PAHDSH          ; Use interlock handshake
        CLR     PAPOL           ; Port A bits are non-inverting
        CLR     PASIO           ; Normal output
        MOVB    #300,PASTAT     ; Enable Port A interrupts

        MOVB    #377,PCDDIR     ; Port C bits are inputs

        MOV     #165400,IOCNTL  ; configure the PIO buffers for A=out
                                ;  B=input, C0,C2=input, C1,C3=output

        MOV     #OUTBUF,R0      ; Point to output data buffer

        MOVB    #24,MCC         ; Enable ports A  and C
        MOVB    #200,MIC        ; Enable MIC
        MTPS    #0              ; Enable recognition of interrupts
```

```
            MOVB    #200,PASTAT     ; Set IP to initiate a transfer
            BR                      ; Wait here for the interrupts

OUT::
            TSTB    (R0)            ; IF (R0) are negative
            BMI     1$              ;  THEN all data has been transferred
                                    ;  ELSE do another transfer
            MOVB    (R0)+,PADATA    ; Move 1st byte to the Port A output
                                    ;  data register
            MOVB    (R0)+,PADATA    ; Move 2nd byte to the Port A buffer
                                    ;  register
            BR      2$
1$:         MOVB    #240,PASTAT     ; Clear IP when done
2$:         MOVB    #140,PASTAT     ; Clear IUS on each pass
            RTI

OUTBUF::.BYTE   151,152,153,154,155,156,157,160,-1

            .END    START
```

```
;  The following two programs demonstrate how the DTC may be used
;  to transfer data from the PIO to KXT11-C local memory.  DTC
;  transfers may only be accomplished using Port A of the PIO.
;  It is not possible to properly connect two PIOs with a ribbon
;  cable because the handshake lines will not align correctly when
;  connecting Port A to Port A.  Therefore it is necessary to build
;  a cable that makes the following connections:
;
;              Input Port A                    Output Port A
;                   A0       <----------->         A0
;                   A1       <----------->         A1
;                    .                             .
;                    .                             .
;                   A7       <----------->         A7
;
;                   C2       <----------->         C3
;                   C3       <----------->         C2
;
;  It is also necessary to place a jumper between posts M48 and M49
;  so that the REQUEST line from the PIO may signal the DTC.  For more
;  information about programming the DTC please refer to Micronote 18.
;
;  After each program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the programs into a KXT11-C to execute as
;  shown in this example:
;
;  SET 3
;  LOAD PIO5I.SAV
;  EXECUTE
;  SET 2
;  LOAD PIO5O.SAV
;  EXECUTE
;  SET 3
;  !ODT
;  !
;  !001140
;  !1140/000777
;  !001142/065151
;  !001144/066153
;  !001146/067155
;  !001150/070157
;  !001152/001602
;  !
;
;  Examining the contents of the input buffer (location 1142)
;  verifies that the data was successfully transferred.
;
;-------------------------------------------------------------------
;
;-------------------------------------------------------------------
        .TITLE  PIO5I.MAC
```

```
        ; This program transfers data from Port A of the PIO
        ; to local memory by utilizing the DTC

        ; Register Assignments

        MMREG   —  174470
        CMDREG  —  174454
        CASTF1  —  174444
        CAOF1   —  174440

        MIC     —  177000
        MCC     —  177002

        PAVEC   —  177004
        PASTAT  —  177020
        PADATA  —  177032
        PAMODE  —  177100
        PAHDSH  —  177102
        PAPOL   —  177104
        PADDIR  —  177106
        PASIO   —  177110

        PCPOL   —  177012
        PCDDIR  —  177014

        IOCNTL  —  177140

START::
        MTPS    #340            ; Inhibit recognition of interrupts

        ; Initialize the DTC - for more information on the DTC
        ; refer to Micronote #18.
        MOVB    #154,MMREG      ; Load Master Mode Reg to Disable DTC
        CLRB    CMDREG          ; Reset the DTC
        MOV     #0,CASTF1       ; Load the CH1 Register SEG/TAG
        MOV     #RELOAD,CAOF1   ; Load the CH1 Register Offset
        MOVB    #155,MMREG      ; Load Master Mode Reg to Enable DTC
        MOVB    #241,CMDREG     ; Start Chain Channel 1

        ; Initialize the PIO
        MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                ;  requests from the PIO
        CLRB    MIC             ; Enable device (interrupts still
                                ;  inhibited)

        ; Set-up Port A
        MOVB    #120,PAMODE     ; Port A:  Input Port, single-buffered
        MOVB    #70,PAHDSH      ; Use interlock handshake, input
                                ;  REQUEST
        CLR     PAPOL           ; Port A bits are non-inverting
        CLR     PASIO           ; Normal input
```

```
        MOVB    #2,PCPOL        ; Invert pin C1 - this is the line
                                ;  that is used for the REQUEST signal
        MOVB    #377,PCDDIR     ; Port C bits are inputs

        MOV     #164377,IOCNTL  ; configure the PIO buffers for A=in
                                ;  B=output, C0,C2=input, C1,C3=output

        MOV     #INBUF,R1       ; Point to input data buffer

        MOVB    #24,MCC         ; Enable ports A and C

        BR                      ; Wait here while the DMA transfers
                                ;  take place

INBUF:  .BLKB   10

; Chain Load Region

RELOAD: .WORD   001602  ; Reload Word <Select CARA,CARB,COPC,CM>

        .WORD   000020  ; Current Address Register A Seg/Tag
        .WORD   padata+1; Current Address Register A Offset
                        ; <This local address is the source,
                        ;  its address is held constant, since
                        ;  the DTC is doing byte transfers specify
                        ;  the source address high byte>

        .WORD   000000  ; Current Address Register B Seg/Tag
        .WORD   inbuf   ; Current Address Register B Offset
                        ; <This local address is the destination>

        .WORD   000010  ; Current Operation Count <Transfer 8 words>

        .WORD   000000  ; Channel Mode Register High
        .WORD   000001  ; Channel Mode Register Low
                        ; <No match conditions, do nothing upon
                        ;  completion, transfer type = Single Transfer
                        ;  CARA = source, byte transfers>

        .END    START
```

;————————————————————————————————————————————————————————————————

;

;————————————————————————————————————————————————————————————————

```
        .TITLE  PIO5O.MAC

        ;  This program transfers data out of Port A of the PIO
        ;  utilizing the DTC

        ; Register Assignments
```

```
MMREG   —  174470
CMDREG  —  174454
CASTF1  —  174444
CAOF1   —  174440

MIC     —  177000
MCC     —  177002

PAVEC   —  177004
PASTAT  —  177020
PADATA  —  177032
PAMODE  —  177100
PAHDSH  —  177102
PAPOL   —  177104
PADDIR  —  177106
PASIO   —  177110

PCPOL   —  177012
PCDDIR  —  177014

IOCNTL  —  177140

START::
        MTPS    #340            ; Inhibit recognition of interrupts

        ; Initialize the DTC
        MOVB    #154,MMREG      ; Load Master Mode Reg to Disable DTC
        CLRB    CMDREG          ; Reset the DTC
        MOV     #0,CASTF1       ; Load the CH1 Register SEG/TAG
        MOV     #RELOAD,CAOF1   ; Load the CH1 Register Offset
        MOVB    #155,MMREG      ; Load Master Mode Reg to Enable DTC
        MOVB    #241,CMDREG     ; Start Chain Channel 1

        ; Initialize the PIO
        MOVB    #1,MIC          ; Reset device and inhibit interrupt
                                ;  requests from PIO
        CLRB    MIC             ; Enable device (interrupts still
                                ;  inhibited)

        ; Set-up Port A
        MOVB    #220,PAMODE     ; Port A: Output Port, single-buffered
        MOVB    #050,PAHDSH     ; Use interlock handshake, output
                                ;  REQUEST
        CLR     PAPOL           ; Port A bits are non-inverting
        CLR     PASIO           ; Normal output

        MOVB    #2,PCPOL        ; Pin C1 must be inverted - this is
                                ;  the line used to signal the DTC
        MOVB    #377,PCDDIR     ; Port C bits are inputs

        MOV     #165400,IOCNTL  ; configure the PIO buffers for A=out
                                ;  B=input, C0,C2=input, C1,C3=output
```

```
        MOV     #OUTBUF,R0          ; Point to output data buffer

        MOVB    #24,MCC             ; Enable ports A and C

        BR                          ; Wait here while the DMA transfers
                                    ;  complete

OUTBUF::.BYTE   151,152,153,154,155,156,157,160,-1
        .EVEN

; CHAIN LOAD REGION

RELOAD: .WORD   001602  ; Reload Word <Select CARA,CARB,COPC,CM>

        .WORD   000000  ; Current Address Register A Seg/Tag
        .WORD   outbuf  ; Current Address Register A Offset
                        ; <This local address is the source>

        .WORD   000020  ; Current Address Register B Seg/Tag
        .WORD   padata+1; Current Address Register B Offset
                        ; <This local address is the destination,
                        ;  Hold the address, must specify high byte
                        ;  for byte transfer>

        .WORD   000010  ; Current Operation Count <Transfer 8 words>

        .WORD   000000  ; Channel Mode Register High
        .WORD   000001  ; Channel Mode Register Low
                        ; <No match conditions, do nothing upon
                        ;  completion, transfer type = Single Transfer
                        ;  CARA = source, byte transfers>

        .END    START
```

## PROGRAMMING THE COUNTER/TIMERS

This section will describe how to program the Counter/Timers and provide example programs demonstrating their capabilities.

Each of the three Counter/Timers provides up to four lines for external access. If these external lines are used the corresponding port pins must be available and programmed in the proper direction. The following table displays which port pins correspond to the Counter/Timer external access lines:

Counter/Timer External Access Lines

| Function | C/T 1 | C/T 2 | C/T 3 |
|---|---|---|---|
| Counter/Timer Output | Port B4 | Port B0 | Port C0 |
| Counter Input | Port B5 | Port B1 | Port C1 |
| Trigger Input | Port B6 | Port B2 | Port C2 |
| Gate Input | Port B7 | Port B3 | Port C3 |

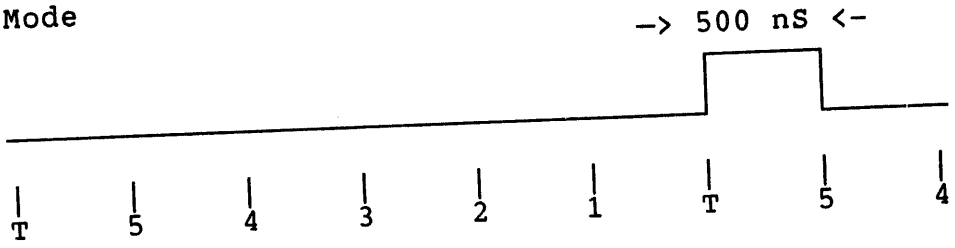
- Table 2 -

The first step in programming a Counter/Timer is to specify which (if any) external lines are to be used, the output duty cycle, and whether the cycle is continuous or single-cycle. The following figures display the available output duty cycles:

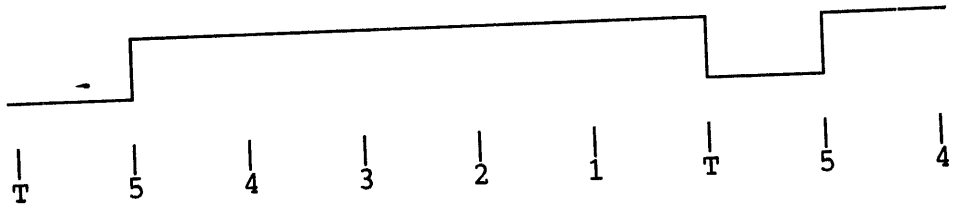
Output Duty Cycles

If Time Constant Value = 5

Pulse Output Mode

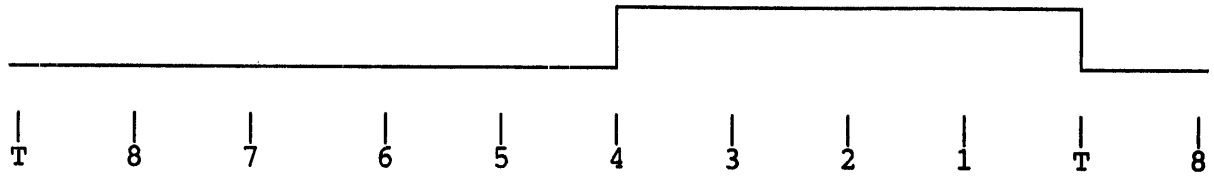



One-Shot Mode



If Time Constant Value = 8

Square Wave Mode




- Figure 2 -

Next, the Time Constant Registers must be loaded. Each Counter/Timer contains two of these registers which are used to form the 16-bit value that is loaded into the down-counter when the Counter/Timer is triggered.

If external lines are to be used then the corresponding port pins should be programmed as bit ports with the correct data direction. Finally, the Counter/Timer enable bit for that port must be enabled in the Master Configuration Control Register.

The down-counter is loaded and the countdown sequence is initiated when the Counter/Timer is triggered. This trigger may occur because the Trigger Command Bit (TCB) in the Command and Status Register is set or because an external trigger input was asserted. Once the countdown is initiated it will continue towards the terminal count as long as the Gate Command Bit (GCB) in the Command and Status Register is set and the Gate Input is held asserted (if it is enabled). If a trigger occurs during a countdown sequence the action taken is determined by the Retrigger Enable Bit (REB). If REB = 0 then the trigger is ignored, but if REB = 1 then the down-counter is reloaded and a new countdown is initiated.

When the terminal count is reached the state of the Continuous/Single Cycle bit (C/SC) in the Mode Specification Register is examined. If C/SC = 0 then the countdown sequence stops. If C/SC = 1 then the time constant is reloaded and a new countdown is initiated. If the Interrupt Enable Bit (IE) is set an interrupt request is generated when the down-counter reaches its terminal count. If a terminal count occurs while the Interrupt Pending Bit (IP) then an error is indicated by the Interrupt Error (ERR) bit.

The following program provides an example of how to program the Counter/Timers:

```
        .TITLE  CT1.MAC

;  This program demonstrates how to utilize one of the Counter/Timers
;  on the KXT11-C.  Counter/Timer 1 will be used in this program.
;  This counter/timer is clocked at a 500 ns rate.  The time constant
;  used for the counter is 50,000.  Therefore the countdown sequence
;  will take 25 ms.  (500 ns x 50,000 = 25,000,000 ns = 25 ms).  The
;  interrupt service routine waits until the countdown sequence has
;  completed 40 times and then outputs an 'A' out of the console
;  port.  This should happen approximately one time a second.  (25 ms
;  x 40 = 1 s).
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD CT1.SAV
;  EXECUTE
;  EXIT
;
;  Notice that the 'A's keep on coming after you exit KUI!

        ; Register Assignments

        MIC     —  177000
        MCC     —  177002
        CTVEC   —  177010
        CT1CON  —  177024
        CT1HI   —  177054
        CT1LO   —  177056
        CT1MOD  —  177070

START::
        MTPS    #340            ; Disable recognition of interrupts

        MOVB    #1,MIC          ; Reset PIO
        CLRB    MIC             ; Enable PIO (Interrupts disabled)

        MOVB    #210,CTVEC      ;
        MOV     #ISR,@#210      ; Initialize Counter/Timer vector
        MOV     #340,@#212      ;  and ISR address

        CLR     R1              ; Used as a counter

        MOVB    #200,CT1MOD     ; Select continuous mode, no external
                                ;  access, pulse output
        MOVB    #203,CT1HI      ; CT1HI and CT1LO combine to form
        MOVB    #120,CT1LO      ;  141520(8) = 50000(10)

        MOVB    #100,MCC        ; Enable Counter/Timer 1
```

```
        MOVB    #200,MIC        ; Enable PIO interrupts
        MTPS    #0              ; Enable recognition of interrupts

        BISB    #306,CT1CON     ; Set IE,GCB,TCB - this starts the
                                ;  countdown
        BR                      ; Wait here for the interrupts

ISR:
        INC     R1              ; Increment the counter
        CMP     R1,#40.         ; IF this is not the 40th time
        BNE     2$              ;  THEN count again
        CLR     R1              ;  ELSE clear the counter and...
        MOVB    #101,@#177566   ;   send an 'A' to the console
;+
;  The console in this case is the KXT11-C console - NOT the
;  development system console.  Therefore you'll have to hook a
;  terminal up  to SLU1 to see the 'A's pop out.
;-
2$:     MOVB    #44,CT1CON      ; Clear IUS and IP but don't bother
                                ;  GCB
        RTI

        .END    START
```

RELATED DOCUMENTS

For further information about the KXT11-C and the PIO please consult the following sources:

KXT11-C Single-Board Computer User's Guide (EK-KXTCA-UG)

AmZ8036 Counter/Timer, Parallel I/O Technical Manual*

* This manual may be obtained from Advanced Micro Devices

| Title: System Configuration of DL-type Devices | Date: 28-Jun-85 |
|---|---|
| Originator: Art Bigler | Page 1 of 14 |

DL-type devices are single, asynchronous serial line unit (SLU) interfaces used on Digital's Q-bus and UNIBUS processors. On PDP-11s the DL-type SLU is required as the console terminal interface for use with console octal debugging technique (ODT), diagnostics, and operating system consoles. They are also used to provide interfacing to a wide variety of peripherals on both PDP-11 and VAX processors.

This MicroNote examines the proper use and configuration of DL-type SLUs in system environments. It first discusses their characteristics, and then, some of the applications in which they are used. Lastly, the configuration of DL-type SLUs in the system environment is discussed and a number of recommendations for their use are presented.

It should be noted that the examination of a system configuration and any resulting recommendations are based upon the particular application of the system. The configuration guidelines presented here are not necessarily suitable for all application environments. The system designer must determine the extent to which this information is appropriate.

## 1.0 CHARACTERISTICS OF THE DL-TYPE SLU

The DL-type SLU is an interface to asynchronous serial peripherals (e.g. terminals) consisting of a host bus interface (to the Q-bus or UNIBUS), a receiver and transmitter (usually an industry-standard UART), and an electrical interface to the peripheral (20 ma. current loop, EIA RS-232-C, RS-422, etc.). Additional features (e.g. the line time clock register on the UNIBUS DL11 or modem control on the Q-bus DLVE1) may or may not be present on a DL-type SLU. These features have nothing to do with the basic function of the SLU which is transmit and receive asynchronous serial data (i.e. Modem control is a extension of the capabilities of the interface, not a basic function). A block diagram of a DL-type SLU is presented in figure 1.

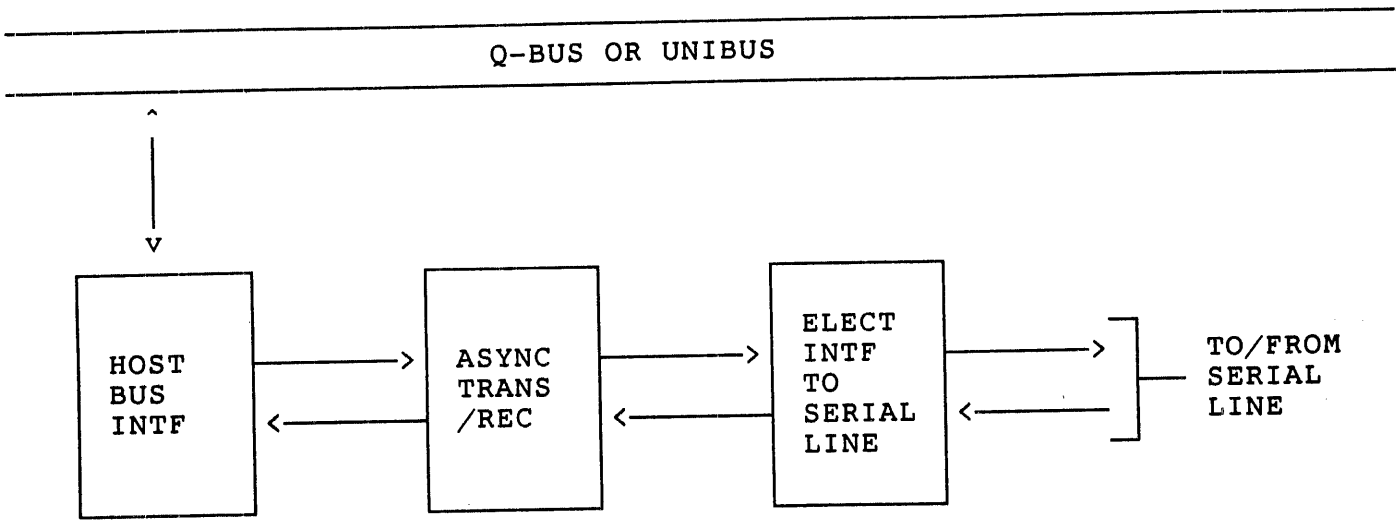



Figure 1
Block Diagram, DL-type SLU

## 1.1 DL-TYPE SLU REGISTERS

All DL-type SLUs must implement four registers: two control and status registers (CSRs), and two data buffer registers (BUFs). The same base configuration of bits within these registers must also be implemented. These registers and their functions are listed below:

1. The receiver control and status register (RCSR) provides a minimum of control and status of the receive interrupt enable (bit 06) and status of the receiver done flag (bit 07). The receiver done flag is set by the SLU whenever a character is received. It may be polled by software or it may generate an interrupt if the receive interrupt enable has been set. The receiver done flag is reset by an initialization from the host bus or by reading the character from the receiver data buffer register.

2. The receiver data buffer register (RBUF) provides a minimum of received character data (bits 00 through 07) and received data error information (bits 12 through 15). The received data is valid whenever the receiver done flag is set in the RCSR. The error bits are implemented in all DL- type SLUs except the DLV11 and provide status information for the received data. All DL-type SLUs implement framing error (bit 13), overrun error (bit 14), and error summary (bit 15). Additionally, all DL- type SLUs, except those

implemented with the DC319-AA DLART, implement parity error status (bit 12). Error summary is set whenever any of the other error bits are set. Overrun error is set when a newly received character is received before the receiver done flag is reset. Framing error is set whenever a character is received without a valid stop bit, usually by a break character. Parity error is set whenever a character is received with the incorrect parity, if the interface has been configured to check parity.

3. The transmitter control and status register (TCSR) provides a minimum of control and status of the transmit interrupt enable (bit 06) and the transmit break enable (bit 00), and status of the transmitter ready flag (bit 07). The transmitter ready flag is set by the SLU whenever the transmitter is ready to transmit another character, or by an initialization from the host bus. It may be polled by a program or it may generate an interrupt if the transmit interrupt enable has been set. The transmitter ready flag is reset by transferring another character to the transmitter data buffer register. The transmit break enable forces the transmitter to generate a continuous break character (continuous space level) while it is set.

4. The transmitter data buffer register (TBUF) holds a minimum of the last character to be transmitted (bits 00 through 07). Transferring a character to this register resets the transmitter ready flag in the TCSR.

Transfers to these registers are performed by programmed I/O, the result of the execution of an instruction by the host processor. The only other bus operation a DL-type SLU may initiate is an interrupt from the transmitter or receiver.

Figure 2 provides graphic representations of these registers.

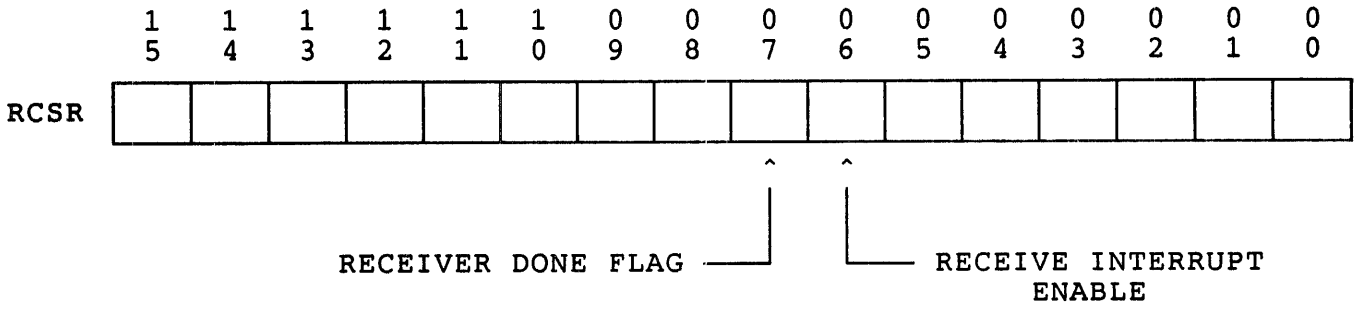



Figure 2
DL-type SLU registers

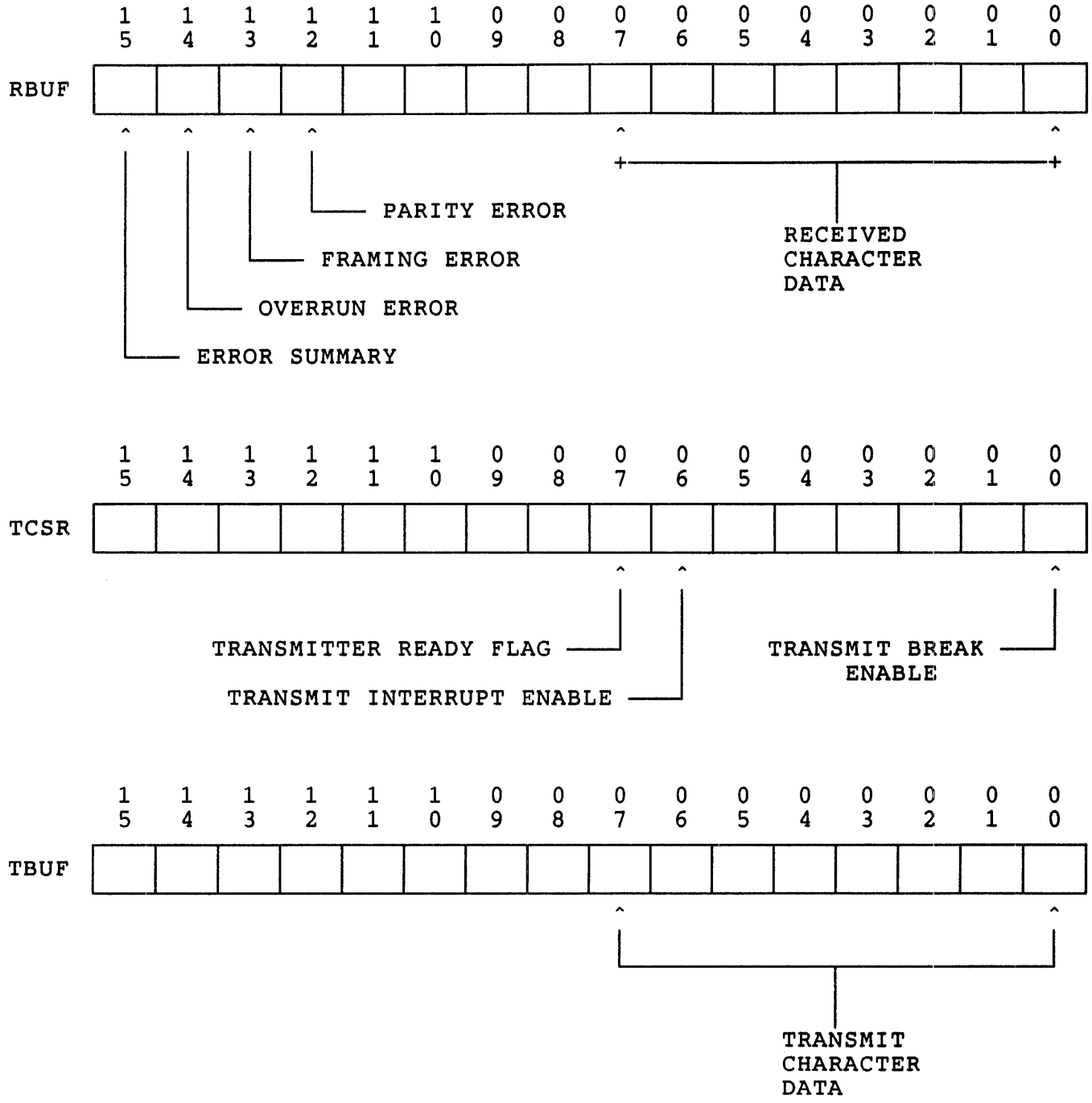



Figure 2, continued
DL-type SLU registers

## 1.2 DL-TYPE SLU ADDRESSES AND VECTORS

DL-type SLUs must implement their addresses and vectors in the following manner:

1. The four registers, RCSR, RBUF, TCSR, and TBUF, occupy four contiguous word addresses in the I/O page. The first address is called the base address.

    A. RCSR is assigned base address +00.

    B. RBUF is assigned base address +02.

    C. TCSR is assigned base address +04.

    D. TBUF is assigned base address +06

2. The receive and transmit interrupt vectors occupy two contiguous vectors with the receive vector always first.

For the console SLU on a PDP-11 processor the base address is always 17777560 (octal) and the receive vector is at 60 (octal). DL-type SLUs used for applications other than the console SLU are usually assigned address and vectors in the floating address and vector spaces.

## 1.3 DL-TYPE SLU PROGRAMMING CONSIDERATIONS

There are two methods of programming DL-type SLUs; polled I/O and interrupt driven I/O.

1. In polled I/O the host software polls, or tests, the receiver done flag in the RCSR or the transmitter ready flag in the TCSR to determine if a character has been received or is ready to be transmitted. If the flag is set the software executes an instruction to move the character from the RBUF or to the TBUF. The polling is then resumed. Polled I/O is used by PDP-11 diagnostics because it is the simplest method and requires the least amount of hardware working in a system, a definite advantage when attempting to service a system which is operating improperly.

2. In interrupt driven I/O the software sets the receive interrupt enable in the RCSR or the transmit interrupt enable in the TCSR. If the receiver done flag or the transmitter ready flag is set, an interrupt is generated through the appropriate vector. Execution is transferred to an interrupt service routine (ISR) and an instruction is

executed to move the character from the RBUF or to the TBUF. A return from interrupt (RTI) instruction then returns control back to the main program. All Digital operating systems and most user software utilize interrupt driven I/O because processor time is optimized.

When programming a DL-type SLU, several restrictions must be taken into account. Most of these are due to the design of the asynchronous receiver and transmitter and its implementation in the SLU. The important restrictions are detailed below.

1. There is only a one character buffer in the receive side of the SLU. Therefore, any character which has been received must be moved from the RBUF prior to the next character being assembled in the receiver. If characters are actually being received at the maximum character rate as defined by the bit data rate of the serial line, the RBUF must be read within one character time or an overflow error will occur on the SLU and data will be lost.

2. The transmitter also has only a one character buffer. However, while this restriction may result in the transmission of characters at a rate lower than the maximum character rate, the transmitter can wait a theoretically infinite time for the next character. No data will be lost and no errors will occur.

3. Direct memory access (DMA) I/O has precedence over interrupt I/O. Therefore, if a DMA device and DL-type SLU wishing to interrupt the processor are both requesting use of the bus, the DMA device will be granted it. This does not normally cause problems in systems which make use of single-cycle DMA transfers. An exception to this is when the system is heavily loaded with a number of active DMA devices. However, the use of burst-mode DMA or, on the Q-bus, block-mode DMA could conceivably lock out the interrupts from the SLU for an appreciable length of time. This should have little effect on the transmitter since it can wait indefinitely for interrupts, but it could have a disastrous effect on the receiver, resulting in the loss of data and the generation of overrun errors.

## 1.4 DIGITAL'S DL-TYPE SLUs

Digital manufactures a variety of DL-type SLUs for both the Q-bus and the UNIBUS. The following list is compilation of those currently available.

1. Q-bus compatible, DL-type SLUs

   A. DLV11 - a single line, DL-type SLU on one dual height module. Data rate from 50 to 9,600 bits per second, full duplex, data leads only (no modem control). RS-232-C and 20 ma. current loop serial line interfaces.

   B. DLVE1 (formerly DLV11-E) - a single line, DL-type SLU on one dual height module. Data rate from 50 to 19,200 bits per second, full duplex with limited modem control. RS-232-C serial line interface.

   C. DLVJ1 (formerly DLV11-J) - four individual, DL-type SLUs on one dual height module. Data rate from 150 to 38,400 bits per second, full duplex, data leads only (no modem control). RS-232-C serial line interface.

2. UNIBUS compatible, DL-type SLUs

   A. DL11 - single line, DL-type SLU on one quad height small peripheral controller (SPC) module. Data rate from 50 to 9600 bits per second, full duplex. A number of variations are available some including modem control and line time clock interfaces. RS-232-C and 20 ma. current loop available.

3. Digital produces two multifunction modules for the Q-bus, the MXV11-A and the MXV11-B, which contain DL-type SLUs. These are intended for the board-level applications where RAM and ROM memory and SLUs in a single compact board are required.

4. Additionally, Digital manufactures a DL-type SLU on a chip, the DC319-AA DLART. The DLART is used in several of Digital's systems as the console terminal SLU for processor modules such as the KDJ11-B.

## 2.0 DL-TYPE SLU APPLICATIONS

The following are examples of the potential uses for DL-type SLUs in systems.

1. In general, DL-type serial line units are required for the console device interface on all PDP-11 processors for three functions:

   A. Console ODT - the console ODT routines in these processors can communicate with DL-type interfaces only.

   B. Diagnostics - diagnostics which run under XXDP and DEC/X11 assume the use of a DL-type console interface.

   C. Operating system console - the operating systems which run on PDP-11 processors require DL-type SLUs for the system console. Even if the console interface device can be redirected to another serial line, as is possible in the RSX family of operating systems, the crash and panic dumps routines usually require the use of a DL-type SLU.

   The reason DL-type interfaces are used for these functions is that they are the simplest, most reliable serial line unit available. If a system is down, having a DL-type SLU as the console interface will almost always eliminate the possibility that the problem is with the console interface.

2. DL-type SLUs are also used in some systems to provide interactive terminal support. Their use for this application is not recommended for reasons which are discussed later in this document.

3. Applications requiring the use of a serial hardcopy terminal as a low-cost system printer may use the DL-type SLU as the interface.

4. Applications which require immediate attention to a serial data stream may require a DL-type SLU as the interface. The serial data stream is most often from an instrumentation device, such as a thermocouple, in a process control environment.

The next sections examine the configuration of DL-type SLUs in system environments.

## 3.0 SYSTEM CONFIGURATION OF DL-TYPE SLUS

There are several items which must be considered when configuring DL-type SLUs into systems.

1. The data rate at which the SLU is expected to transfer data to and from the serial line, and the impact on the host processor's I/O bus.

2. The aggregate data rate expected from all devices on the host processor's I/O bus. This includes the amount and mode of DMA activity on the bus and its potential effect on the processor's response to the DL-type SLU.

3. The interrupt priority of the SLU on the host processor's bus.

In the following discussions, we use an example based upon the configuration of a DLVJ1 quad SLU in a Q-bus system. With the exception of block-mode DMA (which does not exist on the UNIBUS) the example is, in general, valid for both Q-bus and UNIBUS PDP-11 and VAX processors. It must be remembered, however, that the configuration is largely dependent upon the system application which may alter some of the rules presented.

## 3.1 DLVJ1 DESCRIPTION AND CONSIDERATIONS FOR USE

The DLVJ1 is a dual height Q-bus module consisting of four individually programmed RS-232-C/RS-449/RS-423 compatible DL-type SLUs. Although they share common Q-bus transceivers and device selection and data gating logic, the four SLUs each have their own sets of serial line interfaces circuits, Universal Asynchronous Receiver Transmitter (UART) circuits, and DC003 interrupt controller circuits. This, in effect, gives the system designer four individual DL-type serial line units each with its own set of CSRs and data buffer registers and data rates to 38.4K bits per second per channel.

The four DC003 interrupt controllers are configured and wired in Q-bus compatible fashion for bus request level four only (BR4) devices. That is, BIAKI comes in from the bus, through the first DC003, and out as BIAKO which then goes to the BIAKI input of the second DC003. The second DC003 then routes the BIAK signal to the third DC003 and the third to the fourth which then routes its BIAKO signal out to the Q-bus for continuation of the daisy chain. The Q-bus specification provides for two methods of establishing device priority:

1. Distributed arbitration - priority levels are implemented in hardware on the device. Devices must monitor all interrupt requests with higher priority than their own and pass through the interrupt acknowledge if one exists. When devices of equal priority request an interrupt simultaneously , priority is given to the device electrically closest to the processor.

2. Position-defined arbitration - priority is determined solely by electrical position on the bus. The closer a device is to the processor, the higher its priority is. Devices which use position-defined arbitration must be placed in descending bus request order after any devices which implement distributed arbitration.

Digital has produced only one Q-bus device with multiple interrupt request levels, the TSV05. All other devices produced to date have implemented BR4 only, and depend upon position-defined arbitration for their priority. This is largely due to the standard Q-bus interface chips' lack of circuitry required to perform the higher level bus request monitoring. Due to the implementation of the bus, UNIBUS devices do not have this restriction. For further information on the subjects of interrupt priorities and device placement consult the PDP-11 Architecture Handbook (EB-23657) and the Microcomputer Products Handbook (EB-26078).

Receiver done and transmitter done interrupt requests are wired to the DC003's so that the four receiver done interrupts have higher priority than the four transmitter done interrupts. Elevating the receivers' done interrupt priorities over the transmitters' done interrupts helps reduce the potential for lost input characters due to receiver data overrun errors. Transmitter operation is essentially unaffected, except potentially in throughput, since the transmitters will hold their interrupt requests indefinitely. This configuration is actually preferable to that of four individual DL-type interfaces (DLVE1, etc. on the Q-bus or DL11 on the UNIBUS) which would have the transmitter done interrupts placed between the receiver done interrupts.

The device address selection and vector address generation circuits allow the DLVJ1 to be configured at a number of different base addresses and vectors. Channel three may be configured for use as the system console device interface and it is this capability, not its cost, that has made the DLVJ1 an extremely popular option.

The serial interfaces are capable of data-leads-only interfacing (i.e. transmit data, receive data and ground) to external equipment. That is, there is no modem control capability in the DLVJ1. This is usually of no consequence since the largest use of this device is for local console device and terminal interfacing. If the user requires modem control, the DLVE1 is available. If the interface is not being used for the console terminal, asynchronous multiplexers with modem control such as the DZV11 and DHV11 are available and are preferable in almost all applications.

## 3.2 SLU DATA RATE CONSIDERATIONS

The UART used in the DLVJ1 is the 6402 which is a member of a generic UART family in wide use in a large number of Digital and third party vendor products, including the DLVE1 and DLV11 Q-bus options and the DL11 UNIBUS option. They are very similar to Digital's DLART interface chip which is used on the MXV11-B multifunction module, the MicroPDP-11/23 and PDP-11/23-PLUS processor board (KDF11-Bx), and the MicroPDP-11/73 processor board (KDJ11-Bx).

The 6402 provides one level of buffering on input and output data with the implementation of a receiver data holding register and a transmitter data holding register respectively. While the level of buffering is relatively unimportant to the transmitter due to its ability to wait indefinitely for additional characters, it is extremely important to the receiver. To illustrate this the following example is used:

> The 6402 provides a receiver register which holds the current, incoming character. The data is transferred into this register at the bit data rate at which the serial line is running. The time required to fill the receiver register can be calculated as follows:
>
> t = 1 / (bit data rate) * (number of bits per character)
>
> Where t is the character time and the number of bits per character includes the start bit, all data bits, parity bit if used, and all stop bits.

Picking some common numbers for this example:

> Bit data rate = 9600. bits per second.
> Number of data bits = 8.
> Number of parity bits = 0.
> Number of stop bits = 1.

The character time is therefore:

> t = 1/9600*(1+8+0+1)
>
> t = 1.041 msec. per character

Therefore, at a 9600 bits per second data rate, the time from the start of transmission of the character to the time the character is loaded into the receiver buffer register is approximately one millisecond.

If the next character starts transmission immediately, the processor has at least one character time before the receiver buffer register must be emptied so that it can accept the current input character, in this case one millisecond. This implies that the receiver done interrupt must be serviced within that time also. Provided that the combination of software and hardware is capable of servicing the interrupt within a character time the receive data should never overrun.

Therefore, limiting the serial data rate will help reduce the amount of system resources required to service the SLU and optimize processor time. Specifically it increases the allowable interrupt response latency before data overrun would occur.

## 3.3 HOST PROCESSOR AGGREGATE DATA RATE CONSIDERATIONS

The above example does not take into account any other bus interrupt or DMA activity which will impact the speed at which the DLVJ1 interrupt is serviced. To minimize the effects of the host processor's aggregate data rate two configuration rules should be followed:

1. The data rate on the DLVJ1 devices should be kept to a reasonable limit which is usually considered to be at or below 1200 bits per second. This allows additional time for the processor to service the receiver done interrupt and will keep the the amount of bus activity due to serial line interrupts reasonable.

2. Careful attention should be given to the amount and mode (i.e. single cycle, burst, or block) of DMA activity on the host processor's bus. Large quantities of DMA traffic mean that the processor has less time to service interrupts which have a lower priority than DMA operations. Devices doing burst and block mode DMA operations may hold the bus for long periods of time, causing interrupts to be blocked for the duration of the DMA operation. This results in data from the receiver being lost or data to the transmitter being postponed.

## 3.4 INTERRUPT PRIORITY OF THE DL-TYPE SLU

The DL-type SLU should be configured as the highest priority interrupting device on the bus if at all possible. In most cases this will result in the SLU being the first device on the bus, with the possible exception of the line time clock or BEVNT line. This is

consistent with the MicroPDP-11's, PDP-11/23-PLUS's , and the UNIBUS processors, which have their console device interfaces very close to the beginning of the bus.

By placing the DL-type SLUs as close to the beginning of the bus as possible, the interrupt priority is elevated above that of the other devices on the bus. This allows interrupt requests to serviced before the rest of the devices and immediately after the DMA activity, resulting in a reduced possibility of data loss.

## 4.0 SUMMARY AND RECOMMENDATIONS

The DL-type SLUs are required by PDP-11 systems as their console interface. There are several options which may be used to provide this function. Theses include the DLVxx series options on the Q-bus and DLxx series options on the UNIBUS. Additionally, several multifunction options and processors have DL-type interfaces integrated into them. These include the MXV11-A, MXV11-B, KDF11-B and KDJ11-B.

The use of all DL-type SLUs should be limited in most systems (special applications may have valid uses of DL-type devices). They should, in most circumstances, be used only for the console device interface and serial printer applications. The use of an asynchronous multiplexer is recommended for all other serial line requirements, excepting special applications.

The use of any DL-type SLU for an interactive terminal on a multiuser system is indicative of a poor system configuration. In this case, the proper configuration would include asynchronous multiplexers such as DZV11's, DZQ11's, or DHV11's (or their UNIBUS counterparts) to provide efficient handling of terminal interfaces with the minimum of bus activity and operating system overhead.

The use of a DL-type SLU as a low cost printer port is exempt from the above statement because it incurs approximately the same overhead as the LPV11 or LP11-type line printer controller. However, the use of an asynchronous multiplexer with DMA output, such as the DHV11, may provide a more efficient method of handling printers, depending upon the implementation.

The data rate at which any DL-type SLU operates should be kept to a minimum, particularly when receiving data, to ensure that the amount of bus activity is as low as possible since these interfaces are interrupt intensive. This holds for any interrupt intensive device (parallel I/O devices such as DRV11s, etc.) in general. The problem is not that of interface limitations but of the amount of bus bandwidth available to support the aggregate bus data rate.

The DL-type SLU should be placed in as high an interrupt priority position as possible to insure adequate servicing by software. This will not affect DMA activity since it holds priority over bus interrupts. The performance of most systems will not be affected by the increased priority level of the maximum one or two DL-type interfaces which are recommended to be active at one time.

Never place any device, including a DL-type interface, on the bus after a RQDX1 disk controller. The RQDX1 does not pass through the DMA and interrupt grant lines properly, thus producing unpredictable results and often times "hung" devices.

Recommendations for the DL-type SLUs, and any other interrupt intensive devices, are:

1. Limit their use whenever possible. Use asynchronous multiplexers or other more advanced interfaces whenever possible. The cost savings of a DLVJ1 over a DZQ11 must not overshadow the performance issues.

2. Limit the rate at which they operate. For DL-type interfaces a good limit appears to be 1200 bits per second or less although in some applications a higher rate may be possible.

3. Place them so that they have as high an interrupt priority as possible, in most cases the highest priority.

4. If a DL-type SLU is chosen for use, perform the calculations as in section 3.2. Make a rational decision as to whether the configuration can handle the data rates imposed.

Following these few recommendations will result in improved system performance when configuring DL-type SLUs into systems. For additional information consult the appropriate technical manuals, user guides, and systems manuals for the specific devices under consideration.

| Title: Programming the KXT11-C Multiprotocol SLU | Date: 19-Jul-85 |
|---|---|
| Originator: Scott Tincher | Page 1 of 24 |

The KXT11-CA single board computer provides a two-channel multiprotocol serial line unit. The SLU is implemented with an NEC uPD7201 chip. This Micronote will explain the operation of this SLU and provide example programs which display its capabilities. The example programs will be written in Macro-11 so it is assumed that the programmer is familiar with Macro-11 and either the RT-11 or RSX KXT11-C Software Toolkits. It should be noted that the DIGITAL operating system MicroPower/Pascal provides a device handler for the uPD7201 chip.

## FEATURES/CAPABILITIES

The multiprotocol SLU supplies the KXT11-C with the following features and capabilities:

- Two full duplex channels
  - Channel A provides full modem control
  - Channel B provides data and timing leads only
- Each channel may be operated in one of three modes:
  - Asynchronous
    - 5, 6, 7, or 8 Data bits
    - 1, 1-1/2, or 2 Stop bits
    - Odd, Even, or No Parity
    - Break generation and detection
    - Interrupt on Parity, Overun, or Framing Errors
  - Character-oriented synchronous

    - Monosync, Bisync, and External Sync Operations
    - Software Selectable Sync Characters
    - Automatic Sync Insertion
    - CRC Generation and Checking
  - Bit-oriented synchronous
    - HDLC and SDLC Operations
    - Abort Sequence Generation and Detection
    - Automatic Zero Insertion and Detection
    - Address Field Recognition
    - CRC Generation and Checking
    - I-Field Residue Handling
- Programmable Baud Rates
- Double Buffered Transmitted Data
- Quadruply Buffered Received Data
- Programmable CRC Algorithm
- Channel A may utilize the DMA controller to transfer data

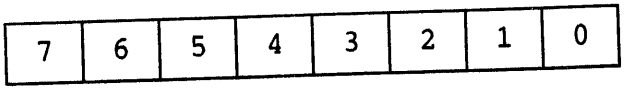
## REGISTER DESCRIPTION

The multiprotocol SLU is controlled by manipulating the registers of the uPD7201 chip as well as registers in support chips that provide access to the baud rate generator and the modem control signals. This section will provide a brief description of the registers necessary to program the multiprotocol SLU.

### uPD7201 Registers

This section will describe the registers found in the uPD7201 itself. These registers are found in both channels of the uPD7201 unless otherwise indicated.

Control Register 0



Bits 0,1,2: Register Pointer

These bits are used to specify which register will be affected by the next Control Register Write or Status Register Read. After a reset the register pointer is set to 0. When the register pointer is set to a value other than 0 the next control or status access is to the specified register, then the pointer is reset to 0.

Bits 3,4,5: Command

These bits are used to select the command to be sent to the uPD7201. A list of these commands follows:

NULL (000)
This command has no effect and is used when setting the register pointer or issuing a CRC command.

SEND ABORT (001)
When operating in the SDLC mode, this command causes the uPD7201 to transmit the SDLC abort code.

RESET EXTERNAL/STATUS INTERRUPTS (010)
Clears any pending external interrupts and reenables the latches so that new interrupts may be detected.

CHANNEL RESET (011)
After issuing a reset command the receivers and transmitters are disabled, the transmitters are set in the marking (high) state, and the modem control outputs are set high. In addition, all interrupts are disabled, and all interrupt and DMA requests are cleared. All control registers must be rewritten after a reset. One NOP instruction must be issued before writing a new command.

ENABLE INTERRUPT ON NEXT CHARACTER (100)
When operating in Interrupt on First Character mode this command is issued to re-enable the interrupt logic for the next received character.

RESET PENDING TRANSMITTER INTERRUPT/DMA REQUEST (101)
Issue this command to reset a pending Transmitter Buffer Becoming Empty interrupt or DMA request without sending another character.

ERROR RESET (110)
This command resets a Special Receive Condition interrupt. It also re-enables the Parity and Overrun Error latches that allow you to check for these errors at the end of a message.

END OF INTERRUPT (111) (Channel A only)

When an interrupt request has been issued by the uPD7201 all lower priority interrupts are blocked to permit the current interrupt to be serviced. At some point in the interrupt service routine this command must be issued to re-enable the daisy chain and permit any pending lower priority interrupt requests to occur.

Bits 6,7: CRC Control Commands

These commands control the operation of the CRC generator/checker logic.

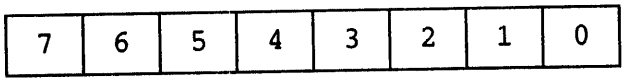
NULL (00)
This command has no effect and is used when issuing other commands or setting the register pointer.

RESET RECEIVER CRC CHECKER (01)
This command resets the CRC checker to 0 when the channel is in a synchronous mode and resets to all ones when is SDLC mode.

RESET TRANSMITTER CRC GENERATOR (10)
This command resets the CRC generator to 0 when the channel is in a synchronous mode and resets to all ones when is SDLC mode.

RESET IDLE/CRC LATCH (11)
This command resets the Idle/CRC latch so that when a transmitter underrun condition occurs, the transmitter enters the CRC phase of operation and begins to send the 16-bit CRC character calculated up to that point. The latch is then set so that if the underrun conition persists, idle characters are sent following the CRC. After a hardware or software reset, the latch is in the set state.

Control Register 1




Bit 0: External/Status Interrupt Enable

When this bit is set to one, the uPD7201 issues an interrupt whenever any of the following occur:

- o transition of DCD input

- o transition of CTS input
- o transition of synch input
- o entering or leaving synchronous Hunt Phase break detection or termination
- o SDLC abort detection or termination
- o Idle/CRC latch becoming set (CRC being sent)

Bit 1: Transmitter Interrupt Enable

When this bit is set to one, the uPD7201 issues an interrupt when:

- o The character currently in the transmitter buffer is transferred to the shift register (Transmitter Buffer Becoming Empty)
- o The transmitter enters Idle Phase and begins transmitting sync or flag characters

Bit 2: Status Affects Vector (Channel B only)

This bit must always be programmed to one so that the fixed vector programmed into Control Register 2B is modified to indicate the cause of the interrupt.

Bits 3,4: Receiver Interrupt Mode

This field determines how the uPD7201 handles the character receive conditions.

RECEIVER INTERRUPTS/DMA REQUEST DISABLED (00)
The uPD7201 does not issue an interrupt or DMA request when a character has been received. (Polled Mode).

INTERRUPT ON FIRST RECEIVED CHARACTER ONLY (01)
The uPD7201 issues an interrupt for the first character received after an Enable Interrupt on First Character Command (CR0) has been given. If the channel is in DMA mode then a DMA request is issued for each character received including the first.

INTERRUPT ON ALL RECEIVED CHARACTERS (10)
An interrupt is issued whenever a character is present in the receive buffer. A DMA request is issued if the channel is in DMA mode. A parity error is considered a special receive condition.

INTERRUPT ON ALL RECEIVED CHARACTERS (11)
This mode is the same as above except that a parity error is not treated as a special receive condition.

The following are always considered special receive conditions:

- o Receiver Overrun Error
- o Asynchronous Framing Error
- o SDLC End of Message

Bits 5,6,7: These bits should always be programmed to 0.

Control Register 2 (Channel A)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bits 0,1: DMA Mode Select

These bits determine the mode in which channels A and B operate. If a channel operates in a non-DMA mode it my perform transfers in either polled or interrupt mode.

| Bit 1 | Bit 0 | Channel A | Channel B |
|---|---|---|---|
| 0 | 0 | Non-DMA | Non-DMA |
| 0 | 1 | DMA | Non-DMA |
| 1 | 0 | Illegal | Illegal |
| 1 | 1 | Illegal | Illegal |

Bit 2: Priority

This bit is used to select the appropriate internal priorities for interrupts. The Channel A receiver always has a higher priority than the Channel A transmitter when Channel A is in DMA mode.

If Channels A and B are both in Interrupt Mode:

0 - RxA > TxA > RxB > TxB > extA > extB
1 - RxA > RxB > TxA > TxB > extA > extB

If Channel A is in DMA mode and Channel B is in Interrupt Mode:

0,1 - RxA > RxB > TxB > extA > extB

Bits 3,5,6,7: These bits must be programmed to 0.

Bit 4: This bit must always be programmed to 1.

Control Register 2 (Programmed in Channel B for both channels)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bits 0..7: Interrupt Vector

The native firmware of the KXT11-C initializes the uPD7201 interrupt vector to 70(8). All interrupts use this vector. In order to determine the cause of the interrupt the uPD7201 must be operated with Condition Affects Vector enabled. (Control Register 1 - Bit 2). When this bit is set the vector is modified to reflect the cause of the interrupt. This modified vector is read from Status Register 2.

Control Register 3

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bit 0: Receiver Enable

0 - Disables the receiver
1 - Enables the receiver

Bit 1: Sync Character Load Inhibit

In a synchronous mode, this bit inhibits the transfer of sync characters to the receiver buffer. When using the uPD7201's CRC checking capabilities this feature should only be used to strip leading sync characters preceding a message since the load inhibit does not exclude sync characters embedded in the message from the CRC calculations. Synchronous protocols using other types of block checking such as checksum or LRC are free to strip embedded sync characters with this bit.

Bit 2: Address Search Mode

In SDLC mode, setting this bit places the uPD7201 in Address Search mode where character assembly does not begin until the 8-bit character (secondary address field) following the starting flag of a message matches either the address programmed into CR6 or the global address 11111111.

Bit 3: Receiver CRC Enable

This bit enables (enable = 1) the CRC checker in COP mode to allow the selective inclusion or exclusion of characters form the CRC calculation.

Bit 4: Enter Hunt Phase

The uPD7201 automatically enters Hunt Phase after a reset. Setting this bit to 1 causes the uPD7201 to re-enter the Hunt Phase.

Bit 5: Auto Enables

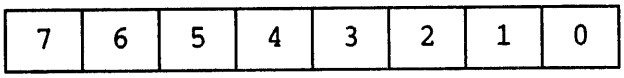
Setting this bit to 1 causes the DCD and CTS inputs to act as enable inputs to the receiver and transmitter, respectively.

Bits 6,7: Number of Received Bits/Character

This field specifies the number of data bits per received character:

| Bit 7 | Bit 6 | Bits/Character |
|---|---|---|
| 0 | 0 | 5 |
| 0 | 1 | 7 |
| 1 | 0 | 6 |
| 1 | 1 | 8 |

Control Register 4




Bit 0: Parity Enable

Setting this bit to 1 adds an extra data bit containing parity information to each transmitted cahracter. Each received character is expected to contain this extra bit and the receiver parity checker is enabled.

Bit 1: Parity Even/Odd

0 - Odd parity generation and checking
1 - Even parity generation and checking

Bits 2,3: Number of Stop Bits/Sync Mode

This field specifies whether the channel is used in a synchronous mode or in asynchronous mode. In asynchronous mode, this field also specifies the number of stop bits used by the transmitter. The receiver always checks for one stop bit.

| Bit 3 | Bit 2 | Mode |
|---|---|---|
| 0 | 0 | Synchronous mode |
| 0 | 1 | Asynch Mode, 1 stop bit |
| 1 | 0 | Asynch Mode, 1-1/2 stop bits |
| 1 | 1 | Asynch Mode, 2 stop bits |

Bits 4,5: Sync Mode

These bits select the synchronous protocol to use if the channel has been programmed in a synchronous mode.

| Bit 5 | Bit 4 | Mode |
|---|---|---|
| 0 | 0 | Monosync |
| 0 | 1 | Bisync |
| 1 | 0 | SDLC |
| 1 | 1 | External Synchronization |

Bits 6,7: Clock Rate

These bits specify the relationship between the transmitter and receiver clock inputs and the actual data rate. When operating in synchronous mode the clock rate must be specified as 1X the data rate.

| Bit 7 | Bit 6 | Clock Rate |
|---|---|---|
| 0 | 0 | Clock Rate = 1X Data Rate |
| 0 | 1 | Clock Rate = 16X Data Rate |
| 1 | 0 | Clock Rate = 32X Data Rate |
| 1 | 1 | Clock Rate = 64X Data Rate |

Control Register 5

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bit 0: Transmitter CRC Enable

A 1 enables the CRC generator calculation. By setting or resetting this bit just before loading the next character, it and subsequent characters are included or excluded from the calculation.

Bit 1: RTS

In synchronous and SDLC modes, setting this bit to 1 causes the RTS pin to go low while a 0 causes it to go high. In asynchronous mode, setting this bit to 0 does not cause RTS to go high until the transmitter is completely empty.

Bit 2: CRC Polynomial Select

A 0 selects the CRC-CCITT Polynomial (X**16 + X**12 + X**5 + 1). A 1 selects the CRC-16 Polynomial (X**16 + X**15 + X**2 +1). The CRC-CCITT polynomial must be selected when in SDLC mode.

Bit 3: Transmitter Enable

After a reset the transmitted data output is held high (marking) and the transmitter is disabled until this bit is set.

Bit 4: Send Break

Setting this bit to 1 forces the transmitter output low (spacing).

Bits 5,6: Transmitted Bits/Character

These bits specify the number of data bits per transmitted character.

| Bit 6 | Bit 5 | Bits/Character |
|---|---|---|
| 0 | 0 | 5 (or less) |
| 0 | 1 | 7 |
| 1 | 0 | 6 |
| 1 | 1 | 8 |

Bit 7: DTR

When this bit is 1, the DTR output is active.

Control Register 6

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bits 0..7: Sync Byte 1

Sync byte 1 is used in the following modes:

| | |
|---|---|
| Monosync: | The 8-bit character transmitted during the Idle Phase. |
| Bisync: | The least significant 8 bits of the 16-bit transmit and receive sync character. |
| External Sync: | Sync character transmitted during the Idle Phase. |
| SDLC: | Secondary address value matched to the Secondary Address field of the SDLC frame when the uPD7201 is in Address Search Mode. |

Control Register 7

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bits 0..7: Sync Byte 2

Sync Byte 2 is used in the following modes:

| | |
|---|---|
| Monosync: | 8-bit sync character matched by the receiver. |
| Bisync: | Most significant 8 bits of the 16-bit transmit and receive sync characters. |
| SDLC: | Must contain the flag value, 01111110, for flag matching by the uPD7201 receiver. |

Status Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bit 0: Received Character Available

When this bit is set, it indicates that one or more characters are available in the receiver buffer. Once all of the available characters have been read, the uPD7201 resets this bit until a new character is received. By utilizing this bit the programmer my run at higher data rates than normal because it will be possible to capture more that one character per interrupt service routine.

Bit 1: Interrupt Pending (Channel A only)

The interrupt pending bit is used with the interrupt vector register (status register 2) to make it easier to determine the uPD7201's interrupt status. In Non-vectored

Interrupt mode, interrupt pending is set when status register 2B is read. If status affects vector is enabled and interrupt pending is set, the vector read from SR2 contains valid condition information.

Bit 2: Transmitter Buffer Empty

This bit is set whenever the transmitter buffer is empty, except during the transmission of the CRC. After a reset, the buffer is considered empty and transmitter buffer empty is set.

Bit 3: DCD (Data Carrier Detect)

This bit reflects the inverted state of the DCD input. When DCD is low, the DCD status bit is high. Any transition of this bit causes an External/Status Interrupt request.

Bit 4: Sync Status

The bit assumes different meanings depending on the operating mode of the uPD7201.

Asynch mode: Sync Status reflects the inverted state fo the Sync input. Any transition of this bit causes an External/Status interrupt request.

External Sync mode: Sync Status operates in the same manner as asynch mode. A low-to-high transition indicates that synchronization has been achieved and character assembly begins.

Monosync, Bisync, SDLC modes: Sync Status indicates whether the receiver is in the Sync Hunt (bit =1) or the Receive Data Phase (bit = 0) of operation.

Bit 5: CTS (Clear to Send)

This bit reflects the inverted state of the CTS input. Any transition of this bit causes an External/Status interrupt request.

Bit 6: Idle/CRC

This bit indicates the state of the Idle/CRC latch used in synchronous and SDLC modes.

Bit 7: Break/Abort

In async mode, this bit indicates that the detection of a break sequence that occurs when the input is held low

(spacing) for more than one character time. This bit is reset when the input returns high (marking).

In SDLC mode, Break/Abort indicates the detection of an abort sequence when 7 or more ones are received in sequence.

Any transition of the Break/Abort bit causes an External/Status interrupt.

Status Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bit 0: All Sent

In async mode, this bit is set when the transmitter is empty and reset when a character is present in the transmitter buffer or shift register. In synchronous and SDLC modes, this bit is always 1.

Bits 1,2,3: SDLC Residue Code

The data portion of an SDLC message can consist of any number of bits and not necessarily an integral number of characters. Special logic determines and reports when the End of Frame flag has been received, the boundary between the data field, and the CRC character in the last few data characters that were read.

Bit 4: Parity Error

This bit is set when parity is enabled and the received parity bit does not match the sense calculated from the data bits.

Bit 5: Receiver Overrun Error

This error occurs when the receiver buffer already contains three characters and a fourth character is completely received, overwriting the last character in the buffer.

Bit 6: CRC/Framing Error

In Async modem a framing error is flagged when no sop bit is detected at the end of a character.

In sync and SDLC modes, this bit indicates the result of the comparison between the current CRC result and the appropriate check value.

Bit 7: End of SDLC Frame

This flag is used in SDLC mode to indicate that the End of FRame flag has been received and that the CRC error flag and residue code is valid.

Status Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bits 0..7: Interrupt Vector (Channel B only)

Reading Status Register 2B returns the interrupt vector that was programmed into Control Register 2B. If Condition Affects Vector is enabled the value of the vector is modified as follows:

Condition Affects Vector Modifications

| Bit 2 | Bit 1 | Bit 0 | Condition |
|---|---|---|---|
| 1 | 1 | 1 | No interrupt pending |
| 0 | 0 | 0 | Channel B Transmitter Buffer Empty |
| 0 | 0 | 1 | Channel B External/Status Change |
| 0 | 1 | 0 | Channel B Received Character Available |
| 0 | 1 | 1 | Channel B Special Receive Condition |
| 1 | 0 | 0 | Channel A Transmitter Buffer Empty |
| 1 | 0 | 1 | Channel A External/Status Change |
| 1 | 1 | 0 | Channel A Received Character Available |
| 1 | 1 | 1 | Channel A Special Receive Condition |

- Table 1 -

Code 111 has two meanings: either Channel A Special Receive Condition or no interrupt pending. In order to distinguish between the two, the Interrupt Pending bit (SR0) must be examined.

Baud Rate Generator Registers

Programmable baud rates for channels A and B are supplied by an Intel 8254-2 timing controller chip with two counters operating at 9.8304 MHz. A third counter that operates at 800 Hz is available for general use. This general purpose counter issues a level 6 interrupt request to the T-11 via vector 104.

Programming these counters is straightforward. First, a Timer Control register is initialized to provide the proper counting mode. Then a divider ratio is loaded into a Timer Data register to obtain the desired baud rate. The divider ratio is obtained from the following calculations:

For synchronous transmission,

Synchronous bit rate = 9830.4K / divider ratio

Therefore,

divider ratio = 9830.4K / synchronous bit rate

A few examples,

| Bit Rate | Ratio |
|---|---|
| 1200 | 8192 |
| 9600 | 1024 |
| 38.4K | 256 |
| 76.8K | 128 |

For asynchronous transmission (assuming that the clock rate is divided by 16),

Asynchronous bit rate = 9830.4K (1/16) / divider ratio

Therefore,

divider ratio = 614.4K / asynchronous bit rate

A few examples,

| Bit Rate | Ratio |
|---|---|
| 1200 | 512 |
| 9600 | 64 |
| 38.4K | 16 |
| 76.8K | 8 |

Timer Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Bit 0: BCD or Binary

0 - Use 16-bit binary counter
1 - Use BCD counter with four decades

Bits 1,2,3: Mode Select

The mode of the counter is selected with these bits:

| Bit 3 | Bit 2 | Bit 1 | Mode |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt on Terminal Count |
| 0 | 0 | 1 | Not supported |
| 0 | 1 | 0 | Rate Generator |
| 0 | 1 | 1 | Square Wave Generator |
| 1 | 0 | 0 | Software Triggered Strobe |
| 1 | 0 | 1 | Not supported |
| 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | Reserved |

Bits 4,5: Read/Write Sequence Selection

The Timer Data registers are programmed on a byte basis. These bits determine the sequence in which the Timer Data registers interpret the data.

| Bit 5 | Bit 4 | Sequence |
|---|---|---|
| 0 | 0 | Counter Latch Command |
| 0 | 1 | Read/Write least significant byte only |
| 1 | 0 | Read/Write most significant byte only |
| 1 | 1 | Read/Write least significant byte first, then most significant byte |

Bits 6,7: Counter Select

These bits select which counter is being programmed.

| Bit 7 | Bit 6 | Counter |
|---|---|---|
| 0 | 0 | Counter 0 |
| 0 | 1 | Counter 1 |
| 1 | 0 | Counter 2 |
| 1 | 1 | Read-back command |

KXT Control/Status Register A

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

This register contains some of the control lines for the uPD7201.

Bit 0: SYNCM B

0 - Channel B receives its clock from the onboard baud rate generator
1 - Channel B receives its clock from an external source

Bit 1: SLU2B R EN

0 - Party line receive data enabled (Board must be properly configured)
1 - Party line receive data disabled

Bit 2: SYNCM A

0 - Channel A receives its clock from the onboard baud rate generator
1 - Channel A receives its clock from an external source

Bit 3: Data Terminal Ready (DTR)

0 - DTR is not asserted
1 - DTR is asserted

Bit 4: Terminal in Service (Busy)

0 - Terminal in Service is not asserted
1 - Terminal in Service is asserted

Bit 5: Diagnostic Prom Enable

This bit allows two different 1K portions of the onboard firmware to be visible at addresses 160000-163777.

Bit 6: Real Time Clock Enable

0 - The RTC interrupt is inhibited
1 - The RTC interrupt is enabled

Bit 7: Counter 2 Interrupt Enable

0 - Counter 2 interrupts are inhibited
1 - Counter 2 interrupts are enabled

The following table lists the registers that have been described and their addresses:

| | | |
|---|---|---|
| KXT Control/Status Register A | 177520 | Read/Write |
| Timer Control Register | 175736 | Write only |
| Timer 2 Data Register | 175734 | Write only |
| Timer 1 Data Register | 175732 | Write only |

| | | |
|---|---|---|
| Timer 0 Data Register | 175730 | Write only |
| Timer 2 Data Register | 175724 | Read only |
| Timer 1 Data Register | 175722 | Read only |
| Timer 0 Data Register | 175720 | Read only |
| | | |
| Channel B Transmitter | 175716 | Write only |
| Channel B Control Register | 175714 | Write only |
| Channel B Receiver | 175712 | Read only |
| Channel B Status Register | 175710 | Read only |
| Channel A Transmitter | 175706 | Write only |
| Channel A Control Register | 175704 | Write only |
| Channel A Receiver | 175702 | Read only |
| Channel A Status Register | 175700 | Read only |

- Table 2 -

PROGRAMMING EXAMPLES

The following programs provide 'skeletons' on which to base user application programs.

```
          .TITLE  SLU1.MAC

;  This program utilizes the uPD7201 to transfer serial data.  The
;  data will be transfered out of Channel A and received by Channel
;  A so a loopback connector is required (Part #H3022 or 54-16229-01)
;  This example transfers the data in asynchronous mode using
;  interrupts.
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD SLU1.SAV
;  EXECUTE
;  !ODT
;  !
;  !001206
;  !001302/041101
;  !001304/042103
;  !001306/043105
;  !001310/044107
;  !001312/041101
;  !001314/042103
;  !001316/043105
;  !001320/044107
;  !001322/000000
;  !R4/000000
;  !
```

```
;  EXIT
;
;  This verifies that the data was successfully transfered.  1302 is
;  the address of the transmit buffer and 1312 is the address of the
;  receive buffer.  R4=0 verifies that no external or special
;  condition interrupts were received.
;

;  Register Definitions

        STATA  — 175700         ; Channel A status register
        RBUFA  — 175702         ; Channel A receiver
        CNTRLA — 175704         ; Channel A Control register
        TBUFA  — 175706         ; Channel A transmitter

        STATB  — 175710         ; Channel B status register
        CNTRLB — 175714         ; Channel B control register

        TIMREG — 175736         ; Timer control register
        TIMER0 — 175730         ; Timer 0 data register

START::

;  This section initializes the KXT11-C system environment

        MTPS    #340            ; Disable recognition of interrupts

        MOV     #ISR,@#70       ; SLU2 interrupts at location 70
        MOV     #340,@#72       ; Let the ISR run at priority 7

        CLR     R0              ; This is the transmit char counter

        MOV     #TBUF,R2        ; R2 points to the transmit buffer
        MOV     #RBUF,R3        ; R3 points to the receive buffer

        CLR     R4              ; This counter keeps track of external
                                ;  status changes and special receive
                                ;  receive conditions

;  This section initializes the bit rate generator

        MOVB    #26,TIMREG      ; Select timer 0, low byte only,
                                ;  mode 3, binary
        MOVB    #64.,TIMER0     ; This divider selects 9600 bps

;  This section initializes the 7201 for asynch operation

        MOVB    #30,CNTRLA      ; Reset Channel A
        NOP                     ; Wait for reset to complete

        MOVB    #30,CNTRLB      ; Reset Channel B
        NOP                     ; Wait for reset to complete
```

```
            MOVB    #2,CNTRLA       ; Point to CR2A
            MOVB    #24,CNTRLA      ; Setup bus interface options:
                                    ; No DMA, RxA>RxB>TxA..., Non-Vectored

            MOVB    #4,CNTRLA       ; Point to CR4
            MOVB    #104,CNTRLA     ; Set operation mode:
                                    ; No parity, asynch mode, 1 stop bit,
                                    ;  clock rate = 16x data rate

            MOVB    #3,CNTRLA       ; Point to CR3
            MOVB    #301,CNTRLA     ; Enable receiver, char length = 8

            MOVB    #5,CNTRLA       ; Point to CR5
            MOVB    #152,CNTRLA     ; Enable transmitter, Char length = 8

            CLRB    CNTRLA          ; Point to CR0
            MOVB    #20,CNTRLA      ; Reset External/Status Interrupts

            MOVB    #1,CNTRLA       ; Point to CR1
            MOVB    #36,CNTRLA      ; Transmit IE, Interrupt on all
                                    ;  received chars, enable condition
                                    ;  affects vector

MAIN::
            MTPS    #0              ; Enable recognition of interrupts
            MOVB    (R2)+,TBUFA     ; Send first character
            BR      .               ; Stay here while the interrupts occur

ISR::
            MOVB    #2,CNTRLB       ; Point to SR2B
            MOVB    STATB,-(SP)     ; Store the condition affects vector
                                    ;  on the stack

            ;  This section inspects the condition affects vector to
            ;  determine the cause of the interrupt

            ROR     (SP)            ; Rotate bit 0 into the carry bit
            BCS     EXT             ; If this bit was set then the
                                    ;  interrupt was caused by a special
                                    ;  receive condition or an external/
                                    ;  status change
            ROR     (SP)            ; Rotate bit 1 into the carry bit
            BCS     RCV             ; If this bit was set then the
                                    ;  interrupt was caused by a received
                                    ;  character
            ;+
            ;  If neither of the above conditions was
            ;  satisfied then the interrupt must have
            ;  been caused by the transmitter buffer
            ;  going empty
            ;-
XMIT::
```

```
        INC     R0              ; Increment the xmit char counter
        CMP     R0,#8.          ; IF this is the eight char
        BEQ     1$              ;  THEN branch to 1$
        MOVB    (R2)+,TBUFA     ;  ELSE send another char
        BR      IDONE           ;   and return
1$:     MOVB    #50,CNTRLA      ;  reset pending xmit interrupt
        BR      IDONE           ;   request - then return

RCV::   MOVB    RBUFA,(R3)+     ; Store this character
        BR      IDONE           ;  and return

EXT::   ;  This program does not take any special action if an
        ;  External/Status interrupt or Special Receive Condition
        ;  occurs.  Just note that it occurred (there shouldn't be
        ;  any) and continue.

        INC     R4              ; Increment the counter
                                ;  and return

IDONE:: TST     (SP)+           ; Fix the stack
        MOVB    #70,CNTRLA      ; Issue end of interrupt command
        RTI                     ;  and return to main program

TBUF::  .BYTE   101,102,103,104,105,106,107,110
RBUF::  .BLKB   8.

        .END    START



        .TITLE  SLU2.MAC

;  This example program for the uPD7201 transfers serial data via
;  a loopback connector (part #H3022 or 54-16229) between Channel
;  A's transmit and receive using the DMA controller.  No ISR is
;  included in this example as it is meant to show how the uPD7201
;  and the DTC may work together.  A 'real-life' program should
;  include an ISR which monitors any External or Special Receive
;  condition interrupts.  For more information regarding the
;  programming of the DTC please refer to MicroNote #018.
;
;  After this program has been assembled and linked on the
;  development machine use the KUI utility of the KXT11-C Software
;  Toolkit to load the program into the KXT11-C to execute as
;  shown in this example:
;
;  SET 2
;  LOAD SLU2.SAV
;  EXECUTE
;  !ODT
;  !
;  !001234
```

```
;   !1276/041101
;   !001300/042103
;   !001302/043105
;   !001304/044107
;   !001306/041101
;   !001310/042103
;   !001312/043105
;   !001314/044107
;   !001316/000000
;   !
;   EXIT
;
;   This verifies that the data was tranfered successfully.  The
;   transmit buffer begins at address 1276 and the receive buffer
;   begins at address 1306.
;
;   Register Assignments

        MMREG   —   174470      ; Master Mode Register
        CMDREG  —   174454      ; Command Register
        CASTF0  —   174446      ; Chan 0 Chain Address Seg/Tag Field
        CAOF0   —   174442      ; Chan 0 Chain Address Offset Field
        CASTF1  —   174444      ; Chan 1 Chain Address Seg/Tag Field
        CAOF1   —   174440      ; Chan 1 Chain Address Offset Field

        STATA  — 175700         ; Channel A status register
        RBUFA  — 175702         ; Channel A receiver
        CNTRLA — 175704         ; Channel A Control register
        TBUFA  — 175706         ; Channel A transmitter

        STATB  — 175710         ; Channel B status register
        CNTRLB — 175714         ; Channel B control register

        TIMREG — 175736         ; Timer Control register
        TIMER0 — 175730         ; Timer 0 Data register

START::

;   This section initializes the KXT11-C system environment

        MTPS    #340            ; Disable recognition of interrupts

        MOV     #TBUF,R2        ; R2 points to the transmit buffer
        MOV     #RBUF,R3        ; R3 points to the receive buffer

;   This section initializes the bit rate generator

        MOVB    #26,TIMREG      ; Select timer 0, low byte only,
                                ;  mode 3, binary
        MOVB    #64.,TIMER0     ; This divider selects 9600 bps

;   This section initializes the 7201 for asynch operation
```

```
        MOVB    #30,CNTRLA      ; Reset Channel A
        NOP                     ; Wait for reset to complete

        MOVB    #30,CNTRLB      ; Reset Channel B
        NOP                     ; Wait for reset to complete

        MOVB    #2,CNTRLA       ; Point to CR2A
        MOVB    #25,CNTRLA      ; Setup bus interface options:
                                ; Chan A DMA, RxA>RxB>TxA...,
                                ; Non-Vectored

        MOVB    #4,CNTRLA       ; Point to CR4
        MOVB    #104,CNTRLA     ; Set operation mode:
                                ; No parity, asynch mode, 1 stop bit,
                                ;  clock rate = 16x data rate

        MOVB    #3,CNTRLA       ; Point to CR3
        MOVB    #301,CNTRLA     ; Enable receiver, char length = 8
                                ;
        MOVB    #5,CNTRLA       ; Point to CR5
        MOVB    #152,CNTRLA     ; Enable transmitter, Char length = 8
                                ;
        CLRB    CNTRLA          ; Point to CR0
        MOVB    #20,CNTRLA      ; Reset External/Status Interrupts
                                ;
        MOVB    #1,CNTRLA       ; Point to CR1
        MOVB    #16,CNTRLA      ; Transmit IE, Interrupt on 1st
                                ;  received char and issue DMA request
                                ;  enable condition affects vector

;  This section initializes the DMA controller

        CLRB    CMDREG          ; Reset the DTC

        MOV     #0,CASTF0       ; Load Chain Address Register Seg/Tag
        MOV     #LOAD0,CAOF0    ; Load Chain Address Register Offset
        MOV     #0,CASTF1       ; Load Chain Address Register Seg/Tag
        MOV     #LOAD1,CAOF1    ; Load Chain Address Register Offset

        MOVB    #115,MMREG      ; Load Master Mode Reg to Enable DTC

        MOVB    #240,CMDREG     ; Start Chain Channel 0
        MOVB    #241,CMDREG     ; Start Chain Channel 1

MAIN::
        BR                      ; Stay here while the DMA transfers
                                ;  occur

        ; Chain Load Region

LOAD1:  .WORD   001602  ; Reload Word <Select CARA,CARB,COPC,CM>
```

```
            .WORD   000000  ; Current Address Register A Seg/Tag
            .WORD   TBUF    ; Current Address Register A Offset
                            ; <This local address is the source>

            .WORD   000020  ; Current Address Register B Seg/Tag
            .WORD   TBUFA+1 ; Current Address Register B Offset
                            ; <This local address is the destination>

            .WORD   000010  ; Current Operation Count <Transfer 8 bytes>

            .WORD   000020  ; Channel Mode Register High
            .WORD   000001  ; Channel Mode Register Low
                            ; <No match conditions, do nothing upon
                            ;  completion, transfer type = single transfer
                            ;  CARA = source, byte transfers>

LOAD0:      .WORD   001602  ; Reload Word <Select CARA,CARB,COPC,CM>

            .WORD   000020  ; Current Address Register A Seg/Tag
            .WORD   RBUFA+1 ; Current Address Register A Offset
                            ; <This local address is the source>

            .WORD   000000  ; Current Address Register B Seg/Tag
            .WORD   RBUF    ; Current Address Register B Offset
                            ; <This local address is the destination>

            .WORD   000010  ; Current Operation Count <Transfer 8 bytes>

            .WORD   000000  ; Channel Mode Register High
            .WORD   000001  ; Channel Mode Register Low
                            ; <No match conditions, do nothing upon
                            ;  completion, transfer type = single transfer
                            ;  CARA = source, byte transfers>

TBUF::      .BYTE   101,102,103,104,105,106,107,110
RBUF::      .BLKB   10

            .END    START
```

RELATED DOCUMENTS

For further information about the KXT11-C and the uPD7201 please consult the following sources:

KXT11-C Single-Board Computer User's Guide (EK-KXTCA-UG)

uPD7201 Technical Manual *

* This manual may be obtained from NEC

| Title: Backplane Expansion/Termination | Date: 19-Jul-85 |
|---|---|
| Originator: Jack Toto | Page 1 of 8 |

The following MicroNote discusses the termination and expansion configurations. These configurations will deal with 18 and 22-bit Q-bus processors, backplanes and enclosures. Not all cases presented in this MicroNote meet FCC regulations, and only those that do are so marked. The MicroNote is partitioned as follows:

1. System configurations
2. Single Box expansion/termination rules.
3. Multiple box expansion/termination rules.
4. Configuration/case reference chart.
5. Supported single box configuration cases.
6. Supported multiple box configuration cases.

## 1. SYSTEM CONFIGURATION

The following is a list of single and multiple backplane termination rules which must be followed when termination is required. Further explanation of these rules can be found in MicroNote # 029, the Microcomputers Products Handbook (EB-26078-41), the Microcomputer Products Configuration Guide (EB-27318-68), and generally the user guide for any of the CPUs.

The LSI-11 Bus system can be divided into two types:

1. Systems containing one backplane.
2. Systems containing multiple backplanes

Before configuring any system, module/system characteristics must be known. These characteristics are:

1. Power consumption. The +5 Vdc and +12 Vdc current requirements.

2. AC bus loading. The amount of capacitance that a module presents to a bus signal line. AC loading is expressed in terms of ac loads where one ac load equals 9.35 pf of capacitance.

3. DC bus loading. The amount of dc leakage current a module presents to a bus signal when the line is high (undriven). DC loading is expressed in terms of dc loads where one dc load equals 210 ma nominal.

4. Total backplane loading must include ac and dc loads and the power consumption of the processor, modules, terminator module, and backplane.

5. Processor termination, class as either 120 ohms or 240 ohms, as follows:

| OPTION | TERMINATION | MODEL NAME |
|---|---|---|
| A. KDF11-A | 240 OHMS | LSI 11/23 |
| B. KDF11-B | 120 OHMS | LSI 11/23 + |
| C. KDJ11-A | 240 OHMS | LSI 11/73 |
| D. KDJ11-B | 120 OHMS | PDP 11/73 |
| E. MicroVAX I | 240 OHMS | MicroVax I CPU |
| F. MicroVax II | 240 OHMS | MicroVax II CPU |

Power consumption, ac loading, and dc loading specifications for each module can be found in sources mentioned earlier.

## 2. SINGLE BACKPLANE TERMINATION RULES

1. When using a processor with 240 ohms termination, the bus can accommodate up to 20 ac loads (total) before additional termination is required. If more than 20 ac loads are included, the far end of the bus must be terminated with 120 ohms, although termination of 240 ohms is optimum. Following the addition of at least the minimum termination up to 35 ac loads may be present in a single backplane.

2. When using a processor with 120 ohms termination, up to 35 ac loads (total) may be present before additional termination is required. If more than 35 ac loads are included, the far end of the bus must be terminated with 120 ohms. When this has been done up to 45 ac loads may be present.

3. The bus can accommodate up to 20 (total) dc loads. This is true in all cases.

4. The bus signal lines on the backplane can be up to 35.6 cm (14 in) long.

## 3. MULTIPLE BACKPLANE TERMINATION RULES

1. Up to three backplanes maximum can be configured in a multiple backplane system.

2. The signal lines on each backplane can be up to 25.4 cm (10 in) in length.

3. Terminated multiple backplane systems can accommodate up to 44 ac loads, for two backplane systems, and 66 ac loads for three backplane systems. In multiple backplane systems no more than 22 ac loads may be present in any one backplane, nor may any unused ac loads from one backplane be added to the next backplane. It is best to load each backplane equally, but if not possible, then the first and second backplanes should have the highest number of ac loads.

4. DC loading for all modules in all backplanes cannot exceed 20 loads (total).

5. Both ends of the bus must be terminated with 120 ohms. This means that the first and last backplanes must have an impedance of 120 ohms. To achieve this, each backplane must be lumped together as a single point. The resistive termination may be provided by combining two of the modules in the backplane; the processor providing 240 ohms to ground in parallel with an expansion module providing 240 ohms to give the needed 120 ohms termination. Alternately a processor with 120 ohms termination would require no additional termination on the expansion module to provide 120 ohms in the first box. The 120 ohms termination in the last box may be provided in three ways. The termination resistors may reside either on the bus expansion module, or on a bus terminator module such as a BDV11, or on the backplane itself as in the case of the H9275 and H9278 (BA23-A enclosure) backplanes.

6. The cable lengths connecting the first and second backplane are 61 cm (2 ft) or greater.

7. The cables connecting the second and third backplane are 122 cm (4 ft) longer or shorter than the cables connecting the first and second backplanes.

8. The combined length of the cables can not exceed 4.88 m or 16 feet.

9. The cables must have a characteristic impedance of 120 ohms.

## 4. CONFIGURATION/CASE REFERENCE CHART

The chart below is designed to be a quick reference to a specific CPU and system combination. The actual configurations are listed after the chart.

To use the chart below, find the CPU that is in the system and the number of backplanes or enclosures that you will be using. The intersection of the two parameters will give you the case/variation number that is valid for that configuration. Ex. case 1.2 represents case 1, with variation 2.

SYSTEM CONFIGURATION CHART

| PROCESSOR | SINGLE BOX | TWO BOX | THREE BOX |
|---|---|---|---|
| KDF11-A 240 OHMS | CASE 1<br>CASE 2 | CASE 4.1<br>CASE 4.2<br>CASE 4.3 | 18-BIT SYSTEMS ONLY |
| KDF11-B 120 OHMS | CASE 1<br>CASE 2 | CASE 3.1<br>CASE 3.2<br>CASE 3.3<br>CASE 3.4 | 18-BIT SYSTEMS ONLY |
| KDJ11-A 240 OHMS | CASE 1<br>CASE 2 | CASE 4.1<br>CASE 4.2<br>CASE 4.3 | 18-BIT SYSTEMS ONLY |
| KDJ11-B 120 OHMS | CASE 1<br>CASE 2 | CASE 3.1<br>CASE 3.2<br>CASE 3.3<br>CASE 3.4 | 18-BIT SYSTEMS ONLY |
| MICROVAX I 240 OHMS | CASE 2 | CASE 3.1<br>CASE 3.2<br>CASE 3.3<br>CASE 3.4 | NOT APPLICABLE |
| MICROVAX II 240 OHMS | CASE 2 | CASE 4.1<br>CASE 4.2<br>CASE 4.3 | NOT APPLICABLE |

## 5. SINGLE BOX SYSTEM CONFIGURATION CASES

Single box 18 or 22 bit system configurations can be terminated the following two ways. The two configuration cases presented in this section will give optimum bus termination respectively to 120 ohm and 240 ohm processor based systems.

CASE 1. Use an unterminated enclosure/backplane with a termination card such as the BDV11 in the first unused slot. This card should be ECO'd to etch revision E, when used in 22-bit systems . This card should also have the on board processor and memory diagnostics disabled if it is going to be used to terminate a system with the KDJ11-A or as the CPU. (refer to MicroNote # 003) The following enclosures and backplanes are unterminated:

| | OPTION | SYSTEM SIZE |
|---|---|---|
| A. | BA11-SA | 18/22 BIT |
| B. | BA11-M | 18 BIT |
| C. | BA11-N | 18 BIT |
| D. | H9270-Q | 18/22 BIT |
| E. | H9281-QA | 18/22 BIT |
| F. | H9273-A | 18 BIT |

CASE 2. Use an enclosure/backplane which is already terminated. All but one of Digital's backplanes are terminated with 120 ohms, and will meet the minimum termination required for additional ac loading beyond the capabilities of an unterminated backplane. The one backplane that is not terminated at 120 ohms is the one found inside of the BA23-A enclosure. This option is terminated at 240 ohms. This enclosure is the only option that will provide optimum termination for 240 ohm CPUs. The following table list all of the terminated enclosures and backplanes available from Digital Equipment Corporation:

| | OPTION | SYSTEM SIZE | | TERMINATION |
|---|---|---|---|---|
| A. | BA23-A | 18/22 bit | | 240 OHMS |
| B. | H9275-A | 22 BIT | (not expandable) | 120 OHMS |
| C. | H9281-QB | 18/22 BIT | (not expandable) | 120 OHMS |
| D. | H9281-QC | 18/22 BIT | (not expandable) | 120 OHMS |

## 6. MULTIPLE BOX SYSTEM CONFIGURATION CASES

Multiple box configurations can be up to three boxes maximum. However currently only 18-bit three box systems can be configured and terminated properly. Therefore cases 3 and 4 described below will deal only with two box 22-bit system configurations using CPUs of either impedance as

18-bit systems are sufficiently documented as noted below.

NOTE

FOR 18-BIT MULTIPLE BOX SYSTEMS USING A CPU CONTAINING EITHER 120 OR 240 OHMS OF IMPEDANCE THE PROCEDURE FOR EXPANDING FROM A ONE BOX SYSTEM TO A TWO BOX SYSTEM IS DOCUMENTED IN SEVERAL TECHNICAL RESOURCES, SUCH AS THE EXPANSION PRODUCTS HANDBOOK (EB24836-75/68) AND THE BA11-N TECHNICAL MANUAL (EK-BA11N-TM-001). A PARTICULARLY GOOD RESOURCE FOR 18-BIT MULTIPLE BACKPLANE EXPANSION AND TERMINATION GUIDELINES IS THE LSI SYSTEM SERVICES MANUAL (EK-LSIFS-SV-005).

CASE 3. This case deals with a 120 ohm CPU. The 120 ohms of impedance on the CPU does not have to matched in the first box, but does have to be matched at the far end of the bus which will be located in the second box. This will generate four variations to the case dealing with 120 ohm CPUs. All four of these variations will have in common the BCV2A expansion assembly. This option contains two paddle cards (M9404-00 at 0 ohms and the M9405-YA at 120 ohms) and the BC02D-03 interconnect cable. The card for expanding the bus out of the first box (M9404) will be installed in the first unused slot of the first backplane, with the cable connected to it the bus will be carried to the second backplane. Here the bus is terminated by installing the termination card (M9405) in the first slot of the second backplane.

VARIATION 1: Use two unterminated enclosures such as the BA11-SA master box and the BA11-SE expansion box, connected with the BCV2A. This configuration is not FCC compliant and places the task of FCC compliance on the user. FCC compliance can be obtained by rack mounting these two enclosures in an H9642 cabinet and using the H349 distribution panel to make connections from the system to the outside environment, using the appropriate option cabinet kits. This cabinet system has been tested by Digital Equipment Corporation for FCC compliance.

NOTE

THE NEXT TWO VARIATIONS MAY BE MADE FCC COMPLIANT BY RACK MOUNTING BOTH BOXES IN AN H9642 CABINET THAT HAS THE H9544-AJ SIDE PANELS. THESE SIDE PANELS ALLOW FOR THE SIDE TO SIDE AIR FLOW FOR THE BA23 ENCLOSURE. ALSO INCLUDED IN THIS CABINET CONFIGURATION IS THE H3490 PATCH PANEL WHICH IS USED FOR MAKING CONNECTIONS FROM THE SYSTEM TO THE OUTSIDE ENVIRONMENT VIA THE APPROPRIATE OPTION MODULE CABINET KITS.

VARIATION 2: Use the BA23 enclosure as the primary enclosure and the BA11-SE as the expansion chassis, again using the BCV2A as the interconnect for the two enclosures. The termination that exists on the BA23 backplane must be removed because the that the CPU has 120 ohms of impedance in the first box and does not require any additional termination at this point. The BA11-SE has not been tested in this configuration for FCC compliance, however using the information in the above NOTE MAY produce FCC compliance.

VARIATION 3: Use two BA23 enclosures. When using two BA23-A enclosures and the BCV2A expansion assembly option the termination from both backplanes must be removed. This is due to the fact that the 120 ohms of CPU impedance does not have to matched in the first backplane of a multiple backplane system and that the BCV2A will put the required termination into the last backplane of this configuration.

VARIATION 4: A final variation to the 120 ohm CPU system would be to follow the same scenario as in the first three variations, but using a mix of some terminated and unterminated backplanes as opposed to system enclosures. These backplanes and their termination states are listed in cases one and two.

CASE 4. This case deals with the 240 ohm CPUs. As stated in the termination rules for 240 ohms CPUs, the processors impedance must be matched in the first box. This would bring the total impedance in the first box to 120 ohms which is the ideal impedance. This 120 ohms from the first box, would be matched at the far end of the bus which will be located in the second box. Configurations with 240 ohm CPUs have three variations. All of the case 4 variations will have in common the BCV2A expansion assembly. The installation of this option is explained above, in the section introducing two box systems.

VARIATION 1: This case variation uses the BA23 enclosure as the primary box and expands into a BA11-SE. Using this configuration requires that the termination on the backplane of the BA23 be left in. This will provide for an optimum impedance match in the first box. The bus will be terminated at the far end in the second box via the expansion assembly termination card (M9405-YA). This configuration as is will not be FCC compliant however following the guide lines from the CASE 3 variations FCC compliance MAY possibly be achieved.

VARIATION 2: The two enclosures used here will be the BA23 system box and the BA23 expansion box. While this configuration resembles case 4 with variation 1, the only change will be the removal of any termination from the second (expansion) backplane. Interconnect between the two boxes and FCC compliance can be achieved as described.

NOTE

WHEN CONFIGURING MULTIPLE BACKPLANE SYSTEMS USING THE BA23-A BOX WITH THE RQDX1 RD/RX CONTROLLER INSTALLED, CONSIDERATION SHOULD BE GIVEN TO THE PLACEMENT OF THE CONTROLLER AND ITS RELATIONSHIP TO THE DEVICES THEMSELVES. FURTHER WHEN USING MULTIPLE BA23-A ENCLOSURES IT BECOMES POSSIBLE TO HAVE THE BEVNT LINE FROM BOTH OF THE POWER SUPPLIES TO BE ACTIVE AT THE SAME TIME. THERE SHOULD ALWAYS BE ONLY ONE BEVNT LINE ACTIVE AT ANYTIME, THEREFORE CARE MUST TAKEN TO AVOID THIS CONFLICT. THIS PROBLEM IS AVOIDED WHEN USING A BA23-C AS THE SECOND BOX.

VARIATION 3: This final case 4 variation deals with the use of terminated and unterminated backplanes rather than enclosures. Using a mix of these products the configurations would resemble the first two for case 4, and would follow the same rules for proper termination.

| uNOTE # 036 |
|---|

| Title: MicroVMS Revealed | Date: 19-Jul-85 |
|---|---|
| Originator: Edward P. Luwish | Page 1 of 25 |


ABSTRACT

This MicroNote explains the contents of the MicroVMS distribution kit as well as listing the "Full" VMS files not included in said kit. The description is of MicroVMS V4.1M and VAX/VMS V4.1, but will apply, with minor changes, to later revisions of VMS V4.


## DESCRIPTION OF THE COMPONENTS OF THE MICROVMS KIT

The MicroVMS kit comes in three parts - a standalone BACKUP piece (currently three diskettes), the BASE system (which is copied to a blank, formatted hard disk by the standalone BACKUP) and a collection of additional pieces which are added to the system (using VMSINSTAL) as though they were layered products. These pieces are labeled UTIL, USER, PROG and SYSP. Another piece, described here but purchasable separately, is NET (DECnet). Note that the tape distribution contains all these components (except DECnet) on a single volume - even so, the partitioning is the same.

The installation procedure for MicroVMS is simple - first boot the standalone BACKUP volume, use it to copy the BASE system to the hard disk, then boot and log into the system thus built. In many cases, no other files need be included on the hard disk. If necessary, additional options can be added to the system using VMSINSTAL and the remaining pieces of the distribution kit. The following sections describe the components, starting with the Base system.

The files are divided into classes according to their usefulness in a turnkey runtime environment. Class I was established experimentally, as described in MicroNote # 37 - "In Search of NanoVMS". The remaining classes represent the author's opinion, rather than defining a hierarchy, and were based on the author's experiences with minimum runtime environments. Your environment may be somewhat different.

FILES INCLUDED IN THE BASE SYSTEM KIT

Class I files -
The following BASE SYSTEM files are required if a system is to boot up at all. The assumption is that the boot disk is an RQDX or other MSCP device.

| | |
|---|---|
| sys$system:DCL.EXE | - Command language interpreter (Executes startup command file) |
| sys$system:DUDRIVER.EXE | - Class (protocol) driver for MSCP devices |
| sys$system:F11BXQP.EXE | - File structure and volume structure (Extended QIO Processor) |
| sys$system:FPEMUL.EXE | - Emulate floating point instructions |
| sys$system:INSTALL.EXE | - Utility that installs known images |
| sys$system:JOBCTL.EXE | - Job controller/symbiont manager (Creates detached process for LOGINOUT) |
| sys$system:LOGINOUT.EXE | - Login/logout utility (Needed for response to unsolicited input from non-logged-in terminals) |
| sys$system:MTAAACP.EXE | - Magnetic tape ancillary control process (Required if system is booted from TK50) |
| sys$system:PDDRIVER.EXE | - Pseudo-disk driver for bootstrap (Required if system is booted from TK50) |
| sys$system:PUDRIVER.EXE | - Port (physical) driver for MSCP devices |
| sys$system:RMS.EXE | - Record Management Services |
| sys$system:RUNDET.EXE | - Runs detached images (Needed to run JOBCTL.EXE) |
| sys$system:SCSLOA.EXE | - Loadable routines used by System Communication Services (needed by MSCP, etc..) |
| sys$system:SET.EXE | - Processes many SET commands (Needed frequently by STARTUP.COM) |
| sys$system:SETP0.EXE | - Processes SET MESSAGE command (Needed frequently by STARTUP.COM) |
| sys$system:STARTUP.COM | - System-startup DCL command procedure (Creates a standard VMS environment) |
| sys$system:SYS.EXE | - Operating System image file |
| sys$system:SYSBOOT.EXE | - System bootstrap utility (Sets up system parameters prior to invocation of STARTUP.COM) |
| sys$system:SYSGEN.EXE | - System customization utility (Loads drivers, sets system parameters) |
| sys$system:SYSINIT.EXE | - Operating System Initialization image |
| sys$system:SYSLOAUV1.EXE | - MicroVAX I-specific initialization |
| sys$system:SYSLOAUV2.EXE | - MicroVAX II-specific initialization (Pick only one of the above two files) |
| sys$system:TTDRIVER.EXE | - Terminal driver (including console) |
| sys$system:TUDRIVER.EXE | - Class (protocol) driver for TMSCP tapes (Required if system is booted from TK50) |
| sys$system:VAXEMUL.EXE | - Emulate VAX instructions not in uVAX arch. |
| sys$system:VAXVMSSYS.PAR | - System parameter file (Used by SYSGEN.EXE and SYSBOOT.EXE) |

Class I files - (continued)

| | |
|---|---|
| sys$system:VMOUNT.EXE | - Volume mount utility (Needed to mount system and user disks) |
| sys$library:DCLTABLES.EXE | - Command-parsing tables (Required by DCL.EXE) |
| sys$library:LBRSHR.EXE | - Runtime shareable library for Librarian (Required by INSTALL.EXE) |
| sys$library:LIBRTL.EXE | - Runtime shareable library of common system support routines (Required by Job Controller) |
| sys$library:LIBRTL2.EXE | - Runtime shareable library of common system support routines (Part 2) (Required by Job Controller) |
| sys$library:MOUNTSHR.EXE | - MOUNT shareable image (Required by VMOUNT) |
| sys$library:SCRSHR.EXE | - V3.x screen management package (Required by SYSGEN.EXE) |
| sys$manager:VMSIMAGES.DAT | - Data file for installing known images |
| sys$message:SYSMSG.EXE | - System error messages (required for boot up) |
| sys$message:SYSMGTMSG.EXE | - ACC, EDIT/ACL, BACKUP, INSTALL, MONITOR and AUTHORIZE error message file (required for boot up) |

Class IA files -
These files are required for proper system initialization and orderly shutdown, but will not prevent a system from booting if absent:

| | |
|---|---|
| sys$system:DISMOUNT.EXE | - Volume dismount utility (Required for orderly system shutdown or disk changing) |
| sys$system:OPCCRASH.EXE | - System shutdown utility (Required for orderly system shutdown) |
| sys$system:SHUTDOWN.COM | - System shutdown DCL command procedure (Required for orderly system shutdown) |
| sys$system:UVSTARTUP.COM | - Processor-specific startup commands (Required for orderly system startup) |
| sys$library:DISMNTSHR.EXE | - DISMOUNT shareable image |
| sys$library:MTHRTL.EXE | - Math support runtime shareable library (required by an image invoked by SHUTDOWN.COM) |
| sys$library:UVMTHRTL.EXE | - MicroVAX version of MTHRTL |
| sys$manager:SYSTARTUP.COM | - Site-specific startup commands (Required for orderly system startup) |
| sys$manager:SYCONFIG.COM | - Required for orderly system startup |
| sys$manager:SYSHUTDWN.COM | - Site-specific shutdown commands (Required for orderly system shutdown) |

Class IB Files -
These files are required by optional hardware:

| | |
|---|---|
| sys$system:ANALYZBAD.EXE | - ANALYZE/MEDIA image (Required only for non-MSCP disk support) |
| sys$system:BADBLOCK.EXE | - Dynamic bad block Files-11 ACP subprocess (Required only for non-MSCP disk support) |
| sys$system:DLDRIVER.EXE | - RL02 Disk Driver |
| sys$system:DZDRIVER.EXE | - DZV11 Serial Interface Driver |
| sys$system:MTAAACP.EXE | - Magnetic tape ancillary control process (Required for tape support) |
| sys$system:SMGMAPTRM.EXE | - TERMTABLE global section - runs at system startup. (Required for video terminal support using VMS screen management package) |
| sys$system:SYSLOAWS1.EXE | - Graphics display initialization (Required for VAXstation I support only) |
| sys$system:SYSLOAWS2.EXE | - Graphics display initialization (Required for VAXstation II support only) |
| sys$system:TERMTABLE.EXE | - Compiled terminal definitions file (Required for terminal support) |
| sys$system:TUDRIVER.EXE | - Class (protocol) driver for TMSCP tapes (Required for TK50 support only) |
| sys$system:YFDRIVER.EXE | - DHV11 Serial Interface Driver |

Class II files -
These files MAY be required by user applications or layered products, since they depend on which high-level language or operating system features are used: [Note that C and ADA are absent - their runtime licenses are separate from, and in addition to, that of VMS]

| | |
|---|---|
| sys$library:BASRTL.EXE | - Runtime shareable library - BASIC |
| sys$library:BASRTL2.EXE | - Runtime shareable library - BASIC |
| sys$library:CDDSHR.EXE | - Required by layered products using the Common Data Dictionary - such as Datatrieve |
| sys$library:COBRTL.EXE | - Runtime shareable library - COBOL |
| sys$library:ENCRYPSHR.EXE | - Dummy encryption module (Required by layered products that can optionally use DES data encryption) |
| sys$library:FORRTL.EXE | - Runtime shareable library - FORTRAN |
| sys$library:PASRTL.EXE | - Runtime shareable library - PASCAL |
| sys$library:PLIRTL.EXE | - Runtime shareable library - PL/I |
| sys$library:RPGRTL.EXE | - Runtime shareable library - RPG II |
| sys$library:SMGSHR.EXE | - VMS screen management package |
| sys$library:VMSRTL.EXE | - Old-format VMS runtime library (Required by V3.x applications) |
| sys$message:CLIUTLMSG.EXE | - ANALYZE/MEDIA, EXCHANGE, MAIL, PHONE, PRINT, SUBMIT, RUN, SET, SHOW and SEARCH error messages. |
| sys$message:FILMNTMSG.EXE | - ANALYZE/OBJECT, ANALYZE/IMAGE, EDIT/FDL, ANALYZE/DISK error messages |
| sys$message:PASMSG.EXE | - PASCAL language error messages |
| sys$message:PLIMSG.EXE | - PL/I language error messages |
| sys$message:RPGMSG.EXE | - RPG language error messages |
| sys$message:SHRIMGMSG.EXE | - CONVERT, DCX (library de/compression utility), FDL, SORT, SMGSHR and EDT error messages |

Class IIA files -
These are user utilities that can be called by application programs even though there may exist no way to invoke them from the terminal by DCL command:

| | |
|---|---|
| sys$library:CRFSHR.EXE | - Cross-Reference shareable image (Required by compilers & linker if cross-reference option is invoked) |
| sys$library:DCXSHR.EXE | - Data de/compression support |
| sys$library:EDTSHR.EXE | - Callable editor (Required by EDT.EXE) |
| sys$library:FDLSHR.EXE | - File Description Language parsing shareable image (Required by CREATEFDL.EXE and EDF.EXE) |
| sys$library:SORTSHR.EXE | - VAX Sort/Merge Runtime library (Required by SORTMERGE.EXE) |

Class III files -
These are often used to diagnose or maintain systems in the field. They can be used to adapt systems to changing user need on an interactive basis. Some applications call these as part of their operation.

| | |
|---|---|
| sys$system:BACKUP.EXE | - Backup utility |
| sys$system:CHECKSUM.EXE | - Used during installation of VAX/VMS updates |
| sys$system:ERRFMT.EXE | - Error logging facility |
| sys$system:LINK.EXE | - Linker (Development/upgrade utility) |
| sys$system:MODPARAMS.DAT | - Site modifactions to sysgen parameters - Used by AUTOGEN.COM |
| sys$system:PATCH.EXE | - Image patching utility (Required for system updates/bug fixes) |
| sys$system:SUMSLP.EXE | - Batch-oriented source file editor (Required for system updates/bug fixes) |
| sys$library:SECURESHR.EXE | - Rights database (RIGHTSLIST.DAT) service routines. (Required by BACKUP.EXE) |
| sys$library:SUMSHR.EXE | - Shareable image required by SUMSLP |
| sys$update:AUTOGEN.COM | - System tuning utility |
| sys$update:LIBDECOMP.COM | - Library decompression utility |
| sys$update:REMOVE.COM | - Update utility |
| sys$update:SWAPFILES.COM | - System Tuning utility |
| sys$update:VMSINSTAL.COM | - Update utility |

Optional files for disk support:

| | |
|---|---|
| sys$system:INIT.EXE | - Volume initialization utility |
| sys$system:UNLOCK.EXE | - For reopening improperly closed files |
| sys$system:VERIFY.EXE | - For error-correction of disks |

Class IV files -
These set up and maintain a multi-user environment even if interactive DCL is not supported by the turnkey system:

| | |
|---|---|
| sys$system:AUTHORIZE.EXE | - User authorization utility (Required only for login password support) |
| sys$system:CVTNAF.EXE | - Convert NETUAF.DAT utility (Required only for V3.5+ system upgrades) |
| sys$system:CVTUAF.EXE | - Convert SYSUAF.DAT utility (Required only for V3.5+ system upgrades) |
| sys$system:NOTICE.TXT | - Announcement file for logged-in users |
| sys$system:OPCOM.EXE | - Operator communications facility (For systems with a human operator) |
| sys$system:REPLY.EXE | - Message broadcasting utility (For systems with a human operator /multi-user/secure. Required by OPCOM) |
| sys$system:SETSHOACL.EXE | - SET and SHOW ACCESS CONTROL LIST commands (For secure systems) |
| sys$system:SYSALF.DAT | - Automatically logged-in terminal data file (Required only if this feature is used) |
| sys$system:SYSUAF.DAT | - User authorization data file (Required only for login password support) |
| sys$library:SECURESHR.EXE | - Rights database (RIGHTSLIST.DAT) service routines. (Required for multi-user systems) |
| sys$manager:ADDUSER.COM | - For maintaining multi-user systems |
| sys$manager:ALFMAINT.COM | - For managing automatically logged-in terminals |
| sys$manager:EDTINI.EDT | - Editor initialization script for system manager's account |
| sys$manager:LOGIN.COM | - Command file executed when system manager logs into account |
| sys$manager:SUCCESS.TXT | - Not required |
| sys$manager:SYLOGIN.COM | - Command file executed when any user logs in (prior to user's own LOGIN.COM) (Required for multi-user systems) |
| sys$manager:WELCOME.TXT | - Not required |
| sys$update:BACKUSER.COM | - Back up user directories before performing system update Update utility (multi-user systems) |
| sys$update:CVTNAF.COM | - Convert NETUAF.DAT utility (Required only for V3.5+ system upgrades) |
| sys$update:CVTUAF.COM | - Convert SYSUAF.DAT utility (Required only for V3.5+ system upgrades) |
| sys$update:RESTUSER.COM | - Restore user directories after performing system update Update utility (multi-user systems) |

Class V files -
These are user utilities that exist for the convenience of interactive users. Some of these are of interest primarily to programmers. None of the Class V files normally become part of a turnkey application.

| | |
|---|---|
| sys$system:CDU.EXE | - For defining new DCL commands (User-oriented utility) |
| sys$system:CONVERT.EXE | - Converts RMS files to new formats and organizations (User-oriented utility) |
| sys$system:COPY.EXE | - User-oriented file copying utility |
| sys$system:CREATE.EXE | - User-oriented file and directory creation utility |
| sys$system:CREATEFDL.EXE | - CREATE/FDL image |
| sys$system:DELETE.EXE | - User-oriented file deletion/purge utility |
| sys$system:DIRECTORY.EXE | - User-oriented directory utility |
| sys$system:EDT.EXE | - Interactive text editor |
| sys$system:LIBRARIAN.EXE | - Librarian (Development utility) |
| sys$system:RECLAIM.EXE | - Used to recover free space in ISAM RMS files (User-oriented utility) |
| sys$system:RENAME.EXE | - User-oriented file renaming utility |
| sys$system:SHOW.EXE | - SHOW command processor |
| sys$system:SORTMERGE.EXE | - SORT and MERGE command processor |
| sys$system:TYPE.EXE | - Utility for typing text files on terminal (User-oriented utility) |
| sys$system:VMSHELP.EXE | - Interactive help utility |
| sys$library:CONVSHR.EXE | - Shareable image required by CONVERT.EXE and RECLAIM.EXE |
| sys$library:DBGSSISHR.EXE | - DEBUG system service intercept handler (Development utility) |
| sys$library:TRACE.EXE | - Error traceback facility (Development utility) |
| sys$help:EDTHELP.HLB | - Help library for EDT |
| sys$help:HELPLIB.HLB | - Main help library (small version) |
| sys$help:UAFHELP.HLB | - Help library for AUTHORIZE |
| sys$message:DBGTBKMSG.EXE | - DEBUG, TRACE messages (Development utility) |
| sys$message:PRGDEVMSG.EXE | - CDU, DIFF, DUMP, Librarian, Linker, MACRO, MESSAGE, PATCH, ANALYZE/SYSTEM, and ANALYZE/CRASH messages (Development utility) |
| sys$update:SPKITBLD.COM | - Utility for building software kits (Development utility) |
| sys$update:VMSKITBLD.DAT | - Data for VMSKITBLD.COM - for building MicroVMS binary distribution kits (Development utility) |

Class VI files -

These are files required by options which are not included in the BASE System:

sys$message:NETWRKMSG.EXE - DECnet error messages

## FILES INCLUDED IN THE OPTIONAL KITS

The optional kits include UTIL, USER, PROG, SYSP and (at additional cost) NET. The files of these kits are described in the next sections of the MicroNote. First, a word of explanation about the format: These kits are in VMSINSTAL format - this means they consist of a number of BACKUP savesets. Thus the UTIL option consists of the UTILxxx.A, UTILxxx.B, etc. savesets, where "xxx" is a number giving the version and revision level. Each saveset is listed separately, and denoted as "UTIL_A", "UTIL_B", etc. The savesets are described individually because they can be added to your system (or not) depending on your replies to the interactive VMSINSTAL.COM procedure. The "_A" saveset contains the installation data tables and any default files (if any) needed by ALL the other savesets, so it will often appear empty in the tables below.

### NETWORK DEVICE DRIVERS

In order to permit the use of DECnet communication devices as synchronous serial lines, the hardware drivers are included in the UTIL option, and do not require a DECnet license. These same files are also included in the DECnet kit so as to simplify network installation.

This section lists the files of the UTIL option:

UTIL_A — KITINSTAL.COM

UTIL B — MAIL utility

```
sys$system:MAIL.COM          — Command procedure used by DECnet mail
sys$system:MAIL.EXE          — Mail Utility
sys$system:MAILEDIT.COM      — Default MAIL editing command procedure
sys$help:MAILHELP.HLB        — Mail Utility help file
```

UTIL_C — SEARCH utility

```
sys$system:SEARCH.EXE        — File search utility
```

UTIL_D — DIFF utility

```
sys$system:DIFF.EXE          — File compare utility
```

UTIL_E — DUMP utility

```
sys$system:DUMP.EXE          — File dump utility
```

UTIL_F — RUNOFF utility

```
sys$system:DSRTOC.EXE        — RUNOFF/CONTENTS image
sys$system:DSRINDEX.EXE      — RUNOFF/INDEX image
sys$system:RUNOFF.EXE        — Text formatting utility
```

UTIL_G — PHONE utility

```
sys$system:PHONE.COM         — PHONE startup procedure
sys$system:PHONE.EXE         — Phone utility
sys$help:PHONEHELP.HLB       — Phone utility help file
```

UTIL_H — MicroVMS HELP library

```
sys$help:HELPLIB.HLB         — Full default (DCL) help file
```

UTIL_I — Remote terminal support via SET HOST/DTE

```
sys$system:RTPAD.EXE         — Remote terminal command interface
sys$library:DTE_DF03.EXE     — SET HOST/DTE support for
                               DF03 dialer
```

UTIL_J — Drivers for network communication devices

```
sys$system:NODRIVER.EXE      — Asynchronous DECnet driver
sys$system:XDDRIVER.EXE      — DECnet DMV11 datalink driver
sys$system:XQDRIVER.EXE      — DEQNA Ethernet interface driver
```

(UTIL option, continued)

UTIL_K — LAT-11 terminal server support (via Ethernet)

| | |
|---|---|
| sys$system:LATCP.EXE | — LAT-11 Control Program |
| sys$system:LTDRIVER.EXE | — LAT-11 Driver |
| sys$system:XQDRIVER.EXE | — DEQNA Ethernet interface driver |
| sys$help:LATCP.HLB | — LAT-11 Control Program help file |
| sys$manager:LTLOAD.COM | — Command procedure to load and start LAT |

UTIL_L — Stand-alone backup on system disk support

| | |
|---|---|
| sys$system:STABACCOP.EXE | — Copy program for building standalone BACKUP kit |
| sys$system:STABACKUP.EXE | — Standalone BACKUP utility |
| sys$system:STANDCONF.EXE | — Standalone BACKUP configure image |
| sys$system:STASYSGEN.EXE | — Standalone SYSGEN utility |
| sys$update:STABACKIT.COM | — Command procedure that builds standalone BACKUP to media |

UTIL_M — MicroVAX-I bootstrap that works for any MSCP system device

| | |
|---|---|
| sys$system:VMBUVAX1P.EXE | — Image which boots disks inaccessible from boot ROM or console command |
| sys$update:VMBUVAX1.COM | — Command procedure to build RX50 console floppy to boot other disks |

UTIL_N — Error Log Report Generator utility

| | |
|---|---|
| sys$system:ERF.EXE | — ANALYZE/ERROR image |
| sys$system:ERFBRIEF.EXE | — ANALYZE/ERROR brief report generator |
| sys$system:ERFBUS.EXE | — ANALYZE/ERROR bus display generator |
| sys$system:ERFDISK.EXE | — ANALYZE/ERROR disk display generator |
| sys$system:ERFINICOM.EXE | — ANALYZE/ERROR initialize routines |
| sys$system:ERFPROC1.EXE | — ANALYZE/ERROR processing routines |
| sys$system:ERFPROC2.EXE | — ANALYZE/ERROR processing routines |
| sys$system:ERFPROC3.EXE | — ANALYZE/ERROR processing routines |
| sys$system:ERFPROC5.EXE | — ANALYZE/ERROR processing routines |
| sys$system:ERFSUMM.EXE | — ANALYZE/ERROR summary display routines |
| sys$system:ERFUVAX.EXE | — ANALYZE/ERROR uVAX-specific routines |
| sys$library:ERFCOMMON.EXE | — ANALYZE/ERROR common data structures |
| sys$library:ERFCTLSHR.EXE | — ANALYZE/ERROR shareable image |
| sys$library:ERFLIB.TLB | — ANALYZE/ERROR device descriptions |
| sys$library:ERFSHR.EXE | — ANALYZE/ERROR common routines |

This section lists the files of the USER option:

USER_A — Default files

USER_B — File Access Control List utilities

```
sys$system:ACLEDT.EXE      — Access Control List (ACL) Editor
sys$library:ACLEDIT.INI    — ACL Editor initialization file
sys$help:ACLEDT.HLB        — ACL Editor help file
```

USER_C — Disk Quota utility

```
sys$system:DISKQUOTA.EXE  — Disk Quota Utility
sys$help:DISKQUOTA.HLB    — Disk Quota Utility help file
```

USER_D — Print and Batch Queue utilities

```
sys$system:LPDRIVER.EXE   — Line printer driver
sys$system:PRTSMB.EXE     — Print symbiont
sys$system:QUEMAN.EXE     — Queue managing utility
sys$system:REQUEST.EXE    — Operator request facility
sys$system:SUBMIT.EXE     — Batch job submission utility
sys$library:SMBSRVSHR.EXE — Print symbiont service routines
```

USER_E — Input Queue Symbiont

```
sys$system:INPSMB.EXE     — Card reader input symbiont
```

USER_F — Accounting Log Report Generator utility

```
sys$system:ACC.EXE        — Accounting Utility
```

This section lists the files of the PROG option:

PROG_A — KITINSTAL.COM

PROG B — Debugger utility

```
sys$library:DEBUG.EXE      — Symbolic debugger
sys$help:DEBUGHLP.HLB      — Debugger help file
```

PROG_C — Image Dump utility

```
sys$system:ANALIMDMP.EXE   — ANALYZE/PROCESS DUMP image
sys$library:IMGDMP.EXE     — Image dump procedures
```

PROG_D — RMS Analyze and FDL Editor utilities

```
sys$system:ANALYZRMS.EXE   — ANALYZE/RMS FILE image
sys$system:EDF.EXE         — File Definition Language editor
sys$help:ANLRMSHLP.HLB     — ANALYZE/RMS FILE help file
sys$help:EDFHLP.HLB        — FDL Editor help file
```

PROG_E — Message utility

```
sys$system:MESSAGE.EXE     — Message compiler
```

(PROG option, continued)

PROG_F — Object and Shareable Image libraries

```
    sys$library:IMAGELIB.OLB  — System default shareable image library
    sys$library:STARLET.OLB   — System object and runtime library
```

PROG_G — Macro libraries

```
    sys$library:LIB.MLB       — Operating system macro library
    sys$library:STARLET.MLB   — System macro library
```

PROG_H — Macro assembler

```
    sys$system:MACRO32.EXE    — VAX MACRO assembler
```

PROG_I — SDL intermediary form of STARLET.MLB

```
    sys$system:SDLNPARSE.EXE  — SDL compiler (for installing
                                optional software)
    sys$library:STARLETSD.TLB — Text library of STARLET definitions
                                Used during layered product installations
```

PROG_J — FORTRAN require files

```
    sys$library:FORDEF.FOR    — FORTRAN INCLUDE file: FOR$ sysmbols
    sys$library:FORIOSDEF.FOR — FORTRAN INCLUDE file: IOSTAT error codes
    sys$library:LIBDEF.FOR    — FORTRAN program utility INCLUDE files
    sys$library:MTHDEF.FOR    — FORTRAN INCLUDE file: MATH$ symbols
    sys$library:SIGDEF.FOR    — FORTRAN program utility INCLUDE files
    sys$library:XFDEF.FOR     — Definitions available for programs
                                using DR780 support routines
```

This section lists the files of the SYSP option:

SYSP_A — Default files

```
sys$help:INSTALHLP.HLB     — Install Utility help file
sys$help:PATCHHELP.HLB     — Patch Utility help file
sys$help:SYSGEN.HLB        — Sysgen Utility help file
```

SYSP_B — Files-11 ODS1 ACP and EXCHANGE utility

```
sys$system:EXCHANGE.EXE    — RT-11/DOS file transfer utility
sys$system:F11ACP.EXE      — Files-11 Structure Level 1 ACP
sys$help:EXCHNGHLP.HLB     — Exchange Utility help file
```

SYSP_C — Monitor utility

```
sys$system:MONITOR.EXE     — Monitor utility
sys$help:MNRHELP.HLB       — Monitor Utility help file
```

SYSP_D — Analyze Object File utility

```
sys$system:ANALYZOBJ.EXE   — ANALYZE/IMAGE and ANALYZE/OBJECT image
```

SYSP_E — Delta debugger (for drivers and other privileged code)

```
sys$library:DELTA.EXE      — DELTA multimode debugging tool image
sys$library:DELTA.OBJ      — Alternate debugging tool
```

SYSP_F — System Dump Analyzer utility

```
sys$system:SDA.EXE         — System Dump Analyzer
sys$help:SDA.HLB           — System Dump Analyzer help file
```

SYSP_G — System Symbol Table file

```
sys$system:SYS.STB         — Global symbol table of operating system
```

SYSP_H — Misc Symbol Table files

```
sys$system:SYSDEF.STB      — Global definitions for
                             executive structures
```

SYSP_I — System map

```
sys$system:SYS.MAP         — Map of the operating system
```

SYSP_J — Connect-to-Interrupt Driver

```
sys$system:CONINTERR.EXE   — Connect-to-Interrupt Driver
```

This section consists of the files included in the DECnet Kit. In addition to these, a license disk is included, which unlocks the end-node or the full routing node functionality in these files:

NET_A — Default files

| | |
|---|---|
| sys$system:EVL.COM | — Command file used by DECnet error logging |
| sys$system:EVL.EXE | — DECnet event logging program |
| sys$system:NCP.EXE | — Network control program |
| sys$system:NDDRIVER.EXE | — DECnet pseudo-datalink driver |
| sys$system:NETACP.EXE | — DECnet ancillary control process |
| sys$system:NETDRIVER.EXE | — DECnet logical link driver |
| sys$system:NETSERVER.COM | — Network server DECnet command procedure |
| sys$system:NETSERVER.EXE | — Network server image |
| sys$system:NICONFIG.COM | — Ethernet configurator DECnet command procedure |
| sys$system:NICONFIG.EXE | — Ethernet configurator image |
| sys$system:NML.COM | — NML server startup procedure |
| sys$system:NML.EXE | — DECnet network manager listener |
| sys$system:NODRIVER.EXE | — Asynchronous DECnet driver |
| sys$system:XDDRIVER.EXE | — DECnet DMV11 datalink driver |
| sys$system:XQDRIVER.EXE | — DEQNA Ethernet interface driver |
| sys$library:NMLSHR.EXE | — DECnet management listener shareable image |
| sys$help:NCPHELP.HLB | — Network Command Program help file |
| sys$manager:LOADNET.COM | — DCL procedure to create network ACP process |
| sys$manager:NETCONFIG.COM | — DCL procedure to configure network database |
| sys$manager:STARTNET.COM | — DECnet startup procedure |

NET_B — Incoming Remote File Access files

| | |
|---|---|
| sys$system:FAL.COM | — FAL startup procedure |
| sys$system:FAL.EXE | — DECnet File Access Listener |

NET_C — Incoming Remote Terminal files

| | |
|---|---|
| sys$system:CTDRIVER.EXE | — CTERM Driver |
| sys$system:REMACP.EXE | — Remote device ACP |
| sys$system:RTTDRIVER.EXE | — Remote terminal driver |
| sys$system:STOPREM.EXE | — Stop REMACP utility |
| sys$manager:RTTLOAD.COM | — Remote terminal loader |

(NET option, continued)

NET_D — Outgoing Remote Terminal files

```
sys$system:RTPAD.EXE        — Remote terminal command interface
sys$library:DTE_DF03.EXE    — SET HOST/DTE support for DF03 dialer
```

NET_E — Network Test files

```
sys$system:DTR.COM          — DTRECV.EXE server initiating procedure
sys$system:DTRECV.EXE       — DTSEND server
sys$system:DTSEND.EXE       — DECnet logical links test program
sys$system:MIRROR.COM       — MIRROR startup procedure
sys$system:MIRROR.EXE       — DECnet node loopback server
sys$system:MOM.COM          — Maintenance operations module
                              DECnet command procedure
sys$system:MOM.EXE          — Maintenance operations module image
```

NET_F — Remote Task Loading

```
sys$system:HLD.COM          — Command procedure used by HLD.EXE
sys$system:HLD.EXE          — Downline task loading program
```

FILES WHICH ARE INCLUDED IN VMS DISTRIBUTIONS BUT NOT IN MicroVMS:

Files which are specific to larger CPUs:

Console-device files for 730, 750, 780:

| | | | |
|---|---|---|---|
| VMB.EXE | BOOT58.EXE | BOOTBLOCK.EXE | CONSCOPY.COM |
| BOOTBLDR.COM | BOOTUPD.COM | RTB.EXE | WRITEBOOT.EXE |
| DXCOPY.COM | SETDEFBOO.COM | 780CNSL.DAT | 750CNSL.DAT |
| 730CNSL.DAT | CONSOLBLD.COM | | |

Part of system startup:

| | | |
|---|---|---|
| SYSLOA730.EXE | — | CPU-specific initialization (11/730) |
| SYSLOA750.EXE | — | CPU-specific initialization (11/750) |
| SYSLOA780.EXE | — | CPU-specific initialization (11/780,782,785) |
| SYSLOA790.EXE | — | CPU-specific initialization (VAX 8600) |
| CONFIGURE.EXE | — | Dynamic device configure process |

System-bus attachments:

| | | |
|---|---|---|
| CVDRIVER.EXE | — | 8600 Console Disk Controller (RL02) |
| DQDRIVER.EXE | — | RB730 11/730 Integrated Disk Controller (R80/RL02) |
| PADRIVER.EXE | — | CI780 Port Driver |
| STACONFIG.EXE | — | HSC System Disk Configurator |
| XFDRIVER.EXE | — | DR750, DR780 Ultra-high-speed Parallel Interface |
| XFLOADER.EXE | — | DR750, DR780 Microcode loader |

Omitted from MicroVMS for reasons of size or performance:

DES Encryption:

| | | |
|---|---|---|
| ENCRYPFAC.EXE | — | ENCRYPT command image |

Related to new Screen Management handler for terminals:

| | | |
|---|---|---|
| SMGBLDTRM.EXE | — | Compiler for TERMTABLE definition file |
| SMGTERMS.TXT | — | ASCII source file for DEC terminal definitions |
| TERMTABLE.TXT | — | Terminal definitions source file |

Related to SORT/MERGE:

| | | |
|---|---|---|
| SRTTRN.EXE | — | SORT specification file translator image |

VAXCluster support:

| | | |
|---|---|---|
| FILESERV.EXE | — | File system cache flush server |
| CLUSTRLOA.EXE | — | Loadable VAXcluster support code |
| CSP.EXE | — | Cluster server process image |
| MSCP.EXE | — | MSCP server |
| CNDRIVER.EXE | — | CI DECnet Protocol Driver |
| MAKEROOT.COM | — | Add new roots to cluster common system disk |
| CLUSTRLOA.MAP | — | Link map of loadable VAXcluster support code |
| SHWCLSTR.EXE | — | SHOW CLUSTER command |
| SHWCLHELP.HLB | — | SHOW CLUSTER help file |

11/782 Shared Memory support:

| | | |
|---|---|---|
| MP.EXE | — | VAX 11-782 multiprocessing code |
| MP.MAP | — | Link map of multiprocessing code |
| MPSHWPFM.EXE | — | Multiprocessing utility |
| MPCLRPFM.EXE | — | Multiprocessing utility |
| MBXDRIVER.EXE | — | Shared memory mailbox driver |

PDP-11 Compatibility-mode images:

| | | |
|---|---|---|
| TECO.EXE | — | TECO text editor and programming language |
| TECO.HLB | — | TECO help file |

MASSBUS drivers:

| | | |
|---|---|---|
| DBDRIVER.EXE | — | RP05, RP06 Disks |
| DRDRIVER.EXE | — | RM03, RM05, RM80, RP07 Disks |

LPA11-K Laboratory Peripheral Accelerator support:

| | | |
|---|---|---|
| LADRIVER.EXE | — | LPA11 Laboratory Peripheral Accelerator driver |
| LALOAD.EXE | — | Sends requests to LALOADER.EXE |
| LALOADER.EXE | — | Loads LPA11 microcode |
| LPA11STRT.COM | — | LPA11 site-specific startup command file |

UNIBUS drivers:

| | | |
|---|---|---|
| CRDRIVER.EXE | — | CR11 Card Reader |
| DDDRIVER.EXE | — | TU58 Cartridge Tape |
| DMDRIVER.EXE | — | RK611 (RK06, RK07) Disks |
| DXDRIVER.EXE | — | RX01 Floppy Diskette |
| DYDRIVER.EXE | — | RX02 Floppy Diskette |
| DZDRIVER.EXE | — | DZ11 Asynchronous Serial Multiplexer (NOT the same as MicroVMS DZDRIVER.EXE) |
| LCDRIVER.EXE | — | DMF32 Line Printer Port |
| TFDRIVER.EXE | — | TU78 Magnetic Tape |
| TMDRIVER.EXE | — | TE16, TU45, TU77 Magnetic Tape |
| TSDRIVER.EXE | — | TS11, TS05, TU80 Magnetic Tape (TSV05 will be supported) |
| TUDRIVER.EXE | — | TA81, TU81 Magnetic Tape (NOT the same as MicroVMS TUDRIVER.EXE) |
| XADRIVER.EXE | — | DR11-W High-speed Parallel Interface |
| XEDRIVER.EXE | — | DEUNA Ethernet Interface |
| XGDRIVER.EXE | — | DMF32 Synchronous Port |
| XMDRIVER.EXE | — | DMC11 Synchronous Communications Adapter |
| XWDRIVER.EXE | — | DUP11 Synchronous Serial Line Interface |
| YCDRIVER.EXE | — | DMF32, DMZ32, CPI32 Asynchronous Serial Multiplexers |

Files from sys$examples:

| | | |
|---|---|---|
| ADDRIVER.MAR | — | Example device driver for AD11-K |
| CONNECT.COM | — | Command procedure that connects device for LABIO system |
| DOD_ERAPAT.MAR | - | Example loadable erase pattern generator |
| DRCOPY.PRM | — | Parameter file for DRCOPY routines |
| DRCOPYBLD.COM | — | Command procedure to build DRCOPY.EXE |
| DRMAST.MAR | — | VAX RMS interface for DRMASTER.FOR |
| DRMASTER.FOR | — | Master subroutines for DRCOPY |
| DRSLAVE.FOR | — | Slave subroutines for DRCOPY |
| DRSLV.MAR | — | VAX RMS interface for DRSLAVE.FOR |
| DTE_DF03.MAR | — | SET HOST/DTE dialer support |
| GBLSECUFO.MAR | — | Opens file used as global section for LABIO system |
| LABCHNDEF.FOR | — | Defines information associated with each A/D for LABIO system |
| LABIO.OPT | — | Linker options file for linking modules to be used in LABIO |
| LABIOACQ.FOR | — | Acquires data for LABIO system |
| LABIOCIN.MAR | — | Contains connect-to-interrupt call for LABIO system |
| LABIOCIN.OPT | — | Linker options file for linking LABIO_DATA_ACQ |
| LABIOCOM.FOR | — | Attaches a LABIO user program to the LABIO system modules of the LABIO system |
| LABIOCOMP.COM | — | Command procedure to compile and assemble the modules of the LABIO system |
| LABIOCON.FOR | — | Handles user requests and modifies the database for LABIO system |
| LABIOLINK.COM | — | Command procedure to link LABIO system |
| LABIOPEAK.FOR | — | Samples channel for peak data in LABIO system |
| LABIOSAMP.FOR | — | Samples channel in intervals, reporting date, time and average value on logical device for LABIO system |
| LABIOSEC.FOR | — | Places LABIO_SECTION on page boundary |
| LABIOSTAT.FOR | — | Displays A/D channel status for LABIO system |
| LABIOSTRT.COM | — | Command procedure to start LABIO system |
| LABMBXDEF.FOR | — | Defines mailbox block for LABIO system |
| LBRDEMO.COM | — | Command procedure to create Librarian DEMO.EXE |
| LBRDEMO.FOR | — | Librarian demo (first part) |
| LBRMAC.MAR | — | Librarian demo (second part) |
| LPATEST.FOR | — | LPA11-K test program |
| LPMULT.B32 | — | Example program for line printer |
| MAILCOMPRESS.COM | | Sample procedure to compress mail files |
| MAILCVT.COM | — | Sample procedure to convert V3.x mail files |
| MAILUAF.COM | — | Sample procedure to manipulate sys$system:VMSMAIL.DAT |

Files from sys$examples, continued:

| | | |
|---|---|---|
| MSCPMOUNT.COM | — | Example cluster disk mount procedure |
| PEAK.FOR | — | Peak selection routine in LABIO system |
| SCRFT.MAR | — | Optional screen package (SCR$... in RTL) extension to handle foreign terminals |
| SYSGTTSTR.MSG | — | Sample SYSGEN TERMINAL/ECHO message file |
| TDRIVER.MAR | — | Template for user-written driver |
| TESTLABIO.FOR | — | Tests LABIO system |
| USSDISP.MAR | — | Sample user system service dispatch and service examples |
| USSLNK.COM | — | Link command procedure for USSDISP |
| USSTEST.MAR | — | Sample program to invoke one of the example user services implemented in USSDISP |
| USSTSTLNK.COM | — | Link command procedure for USSTEST |
| XADRIVER.MAR | — | DR11-W driver |
| XALINK.MAR | — | Sample DR11-W to DR11-W link program |
| XAMESSAGE.MAR | — | DR11-W test program |
| XATEST.COM | — | Used to set up XALINK.MAR |
| XATEST.FOR | — | Companion program for XAMESSAGE |
| XIDRIVER.MAR | — | Example driver for parallel port on DMF32 |

11/730 Dual RL02 Tailoring Files:

| | | | |
|---|---|---|---|
| VMSTAILOR.COM | BLISSREQ.TLR | DECNET.TLR | DEVELOP.TLR |
| EXAMPLES.TLR | FILETOOLS.TLR | HELP.TLR | LIBRARY.TLR |
| MANAGER.TLR | MISCTOOLS.TLR | QUEUES.TLR | REQUIRED.TLR |
| TEXTTOOLS.TLR | UETP.TLR | VMSTLRHLP.HLB | |

User Environment Test Package files:

| | | |
|---|---|---|
| TCNTRL.CLD | — | Defines UETP DCL commands |
| UETP.COM | — | Main command procedure |
| UETCLIG00.COM | — | For cluster-integration phase |
| UETCLIG00.DAT | — | For cluster-integration phase |
| UETCLIG00.EXE | — | For cluster-integration phase |
| UETCOMS00.EXE | — | DMC and DMR device test |
| UETDISK00.EXE | — | Disk device test |
| UETDMPF00.EXE | — | DMP and DMF32 device test |
| UETDNET00.COM | — | For DECnet phase |
| UETDNET00.DAT | — | For DECnet phase |
| UETDR1W00.EXE | — | DR11-W device test |
| UETDR7800.EXE | — | DR780 and DR750 device test |
| UETFORT01.DAT | — | Used by load test |
| UETFORT01.EXE | — | Used by load test |
| UETFORT02.EXE | — | Used by load test |
| UETFORT03.EXE | — | Used by load test |
| UETINIT00.EXE | — | Initializes UETP environment |
| UETINIT01.EXE | — | Initializes UETP environment |
| UETLOAD00.DAT | — | Used by load test |
| UETLOAD02.COM | — | User script for load test |
| UETLOAD03.COM | — | User script for load test |
| UETLOAD04.COM | — | User script for load test |
| UETLOAD05.COM | — | User script for load test |
| UETLOAD06.COM | — | User script for load test |
| UETLOAD07.COM | — | User script for load test |
| UETLOAD08.COM | — | User script for load test |
| UETLOAD09.COM | — | User script for load test |
| UETLOAD10.COM | — | User script for load test |
| UETLOAD11.COM | — | User script for load test |
| UETLPAK00.EXE | — | LPA11-K device test |
| UETMA7800.EXE | — | MA780 device test |
| UETMEMY01.EXE | — | Artificial load for load test |
| UETNETS00.EXE | — | Used by DECnet phase |
| UETPHAS00.EXE | — | Test controller |
| UETRSXFOR.EXE | — | Artificial load for load test |
| UETSUPDEV.DAT | — | Supported device data file |
| UETTAPE00.EXE | — | Magnetic tape device test |
| UETTTYS00.EXE | — | Terminal and line printer device test |
| UETUNAS00.EXE | — | DEUNA device test |

Unsupported files for linking against system images:

| | | |
|---|---|---|
| RMS.STB | — | RMS symbol table |
| CLUSTRLOA.STB | — | Symbol table for loadable VAXcluster routines |
| MP.STB | — | Symbol table for MP.EXE |
| SCSDEF.STB | — | Symbol table for loadable SCS routines |
| RMSDEF.STB | — | Global definitions for VAX RMS structures |
| IMGDEF.STB | — | Global definitions for image activator structures |
| DCLDEF.STB | — | Global definitions for DCL structures |
| NETDEF.STB | — | Symbol table for network definition |

BLISS Require Files:

| | | |
|---|---|---|
| LIB.REQ | — | Structure definitions of executive internals for use by BLISS programs |
| STARLET.REQ | — | User interface structures for use by BLISS programs |
| TPAMAC.REQ | — | Structure definitions for BLISS programs using TPARSE |

Superceded or obsolete:

| | | |
|---|---|---|
| VMSUPDATE.COM | — | For updating VMS or adding layered products |

SUMMARY

This MicroNote details the contents of the MicroVMS system, as well as the portion of VAX/VMS not supplied with MicroVMS. The reader can use this information to determine whether any unnecessary files can be omitted from a turnkey system. Additional information can be found in MicroNote # 37 ("In Search of NanoVMS") which describes a working minimal VMS subset. Further information about the structure of VMS can be found in the full VMS document set (Order # QL001-GZ-V4.0 and update QL001-WZ-V4.1), "VAX/VMS Internals and Data Structures" (Digital Press, 1984) and the VAX/VMS source listings on microfiche (source license required — see sales representative).

NOTE

DIGITAL does not recommend the deletion of any component files of the MicroVMS or VAX/VMS operating systems except where explicitly stated in the respective document sets (of which this is NOT a part). A subset operating system cannot be warranted or supported by DIGITAL in any way. This MicroNote is to be used for informational purposes only, and represents the research, conclusions and opinions of the author, not those of DIGITAL or OEM Technical Support.

| Title: In Search of "NanoVMS | Date: 19-Jul-85 |
|---|---|
| Originator: Edward P. Luwish | Page 1 of 8 |


## ABSTRACT

This MicroNote describes the results of ongoing research into the size and composition of a minimal VMS system.


## A WORD ABOUT NOTATION

Where command lines, prompts and messages are discussed, text printed by the computer is indicated by normal type, text entered by the user is indicated by **boldface** type.
Unless explicitly stated otherwise, all user entries are to be terminated by the "return" character.

## WHY "NanoVMS"

MicroVMS as currently packaged and supported by Digital Equipment Corporation is not always an ideal solution for customers who would like to use it as a realtime application bed (rather than a multi-user timesharing system). For this reason, research has been done for over a year on minimal VMS systems. Also, the number of floppy diskettes required to bring up the system has been found excessive by some users. Currently three floppies for standalone backup are followed by thirteen for the base system. This incurs user inconvenience and a greater likelihood of failure in the process. The TK50 cartridge tape is not yet a universal solution to this problem.

## HOW WAS "NanoVMS" BUILT?

The problem was approached by building a VMS system, component by component, on the second winchester of a two-disk MicroVAX. This system could be tested simply by shutting the system down and booting the second disk. If unsuccessful, the full system disk would be rebooted, some files added to the minimal system, and it would be tried again, guided by the error messages produced in the previous attempt.

## HOW WAS "NanoVMS BUILT? (continued)

The first cut at a solution was derived from the "Reboot Consistency Check" offered by sys$system:SHUTDOWN.COM. Notably absent from the list of files was the disk driver! A number of other missing files were disclosed, and success was eventually achieved through repeated attempts. Often the proper functioning of an image depended on the presence of another (such as a library file). The dependencies are shown in the attached directory listing.

## HOW DOES ONE GENERATE A "NanoVMS" SYSTEM?

### 1. Make some choices

There are a number of choices which affect how you generate a "NanoVMS" system. You can create a backup set that can be loaded onto a MicroVAX processor using standalone backup [Note - floppies only. TK50 bootable MicroVMS distributions cannot be created without access to a source kit]. The other choice is to create "NanoVMS" on a winchester and physically install it in to another MicroVAX. This choice affects the third and fourth steps in the process, "Creating the distribution" and "Loading the distribution". Another up-front choice that affects your work is the list of target CPU's you intend to be able to run "NanoVMS" on - currently this includes MicroVAX I, MicroVAX II, VAXstation I and VAXstation II. The work on the latter two has not yet been done - special graphics font files and server files need to be added to the list. Read "Copy the files" below for details.

### 2. Create the directories

If you have a single-winchester system, create [sys1]. If you have a two-disk system, create [sys0] on the non-system disk instead. In either case, create the subdirectories [.sysexe], [.syslib], [.sysmgr] and [.sysmsg] in the [sys0] or [sys1] directories you just created.

If you have chosen the distribution option of physically installing a bootable winchester, and your non-system disk's data is expendable, then you will want to initialize it. Study the section "Creating the Distribution", below, with respect to initialization options. After initializing the disk, create the previously mentioned directories.

### 3. Copy the files

Copy all of the files listed on pages 7 and 8 from the corresponding directories of your MicroVMS V4.1M system. Note that a MicroVAX I CPU requires the file SYSLOAUV1.EXE and a MicroVAX II CPU requires SYSLOAUV2.EXE. Make sure to include the correct one (or both) as required by the target CPU. Also be sure to use the /CONTIGUOUS option when copying SYSBOOT.EXE.

4. Create the distribution

You will first want to decide whether your distribution will include all of the files (including your own application files) of the final system, or merely a basic "NanoVMS" skeleton to which additional files are added from separate disk volumes. If the former, at least list and count all the additional files you need. Note that VMS utilities occasionally need runtime library files, and your application files may need language-related runtime libraries not part of the basic "NanoVMS" system. Be sure to include some extra files in your count. Remember that there are nine files that are part of any VMS volume, and that each directory and subdirectory you create is a file as well.

The nine files that are part of every VMS volume

| | | |
|---|---|---|
| INDEXF.SYS | BITMAP.SYS | BADBLK.SYS |
| 000000.DIR | CORIMG.SYS | VOLSET.SYS |
| CONTIN.SYS | BACKUP.SYS | BADLOG.SYS |

If you chose to create a backup set, simply issue the appropriate mount and backup commands, and insert the floppies into the RX50 drive (n = floppy unit number, x = "NanoVMS" disk unit number, r = root number):

```
$ MOUNT/FOREIGN DUAn:
```

If you wish to merge your own software into the backup set:

```
$ BACKUP/INITIALIZE/LOG/VERIFY -m
$_ DUAx:[SYSr...]*.*;*,[yourdirectory...]*.*;* -m
$_ DUAn:MICROVMS./SAVE_SET
```

If you wish to separate your own software from the backup set:

```
$ BACKUP/INITIALIZE/LOG/VERIFY -m
$_ DUAx:[SYSr...]*.*;* -m
$_ DUAn:MICROVMS./SAVE_SET
```

The backup set created will then install exactly as described in the Installation chapter of the MicroVMS User's Manual.

With standalone backup, you lose flexibility in initializing your system disk - it uses all the default values, which may be unsuitable or wasteful in a bounded system. It is therefore recommended to use the "walking winchester" as a way to transport bounded systems. You would not be considering "NanoVMS" unless you have a legitimate need to save disk space, and standalone backup will often waste space. The index file on an RD-volume is greater than 1000 blocks in allocated size, in order to accomodate a large number of files on the disk. If you can put an upper bound on the number of files you expect to have, you can realize considerable savings in index file size. The two parameters which most affect disk usage, Cluster Factor and Maximum File Count, are explained in the next two paragraphs.

4. Create the distribution (continued)

The cluster factor is the number of disk blocks allocated every time a new file is created, or if additional blocks are needed when editing, etc. If you issue the DCL command

**$ DIRECTORY/SIZE=ALLOCATION**

you will notice that all the sizes are divisible by 3 (the default cluster size for "large" disks). If you have many small files, this can be wasteful. Unless you add the /CLUSTER_SIZE=n option to the INITIALIZE command, this number will be 3 for disks larger than 50,000 blocks, or 1 for smaller disks. Small cluster sizes will adversely affect disk performance since files may be stored as many small pieces scattered widely over the disk surface. On the other hand large cluster sizes will waste disk space, since only one or two of the three (or more) allocated blocks may have data in them.

The maximum number of files contained on a disk is determined at initialization time by the number of empty file headers allocated contiguously in the index file. The DCL command

**$ INITIALIZE/MAXIMUM_FILES=x DUAn: label**

initializes a disk with an index file capable of storing x file headers, no more. The largest value of x is derived from the formula

$$\text{maximum number of files} = \frac{\text{volume size in blocks}}{\text{cluster factor} + 1}$$

The default (when the /MAXIMUM_FILES switch is omitted, or when

**$ BACKUP/INITIALIZE**

is issued) is equal to one-half the number derived by the above formula. If the default is much larger than the actual largest number of files you anticipate storing, then you can gain many free blocks by specifying /MAXIMUM_FILES=x, where x is an upper bound on the number of files you expect on the disk. In fact, you get exactly (default_maxfiles)-x free blocks, since each file header occupies a full block. The potential disadvantage is that the disk will have to be reinitialized (i.e. erased) if you need to store x+1 or more files.

The following table can be compared with your overall needs, so that the appropriate initialization command line can be given:

| | RD51 | RD52 | RD53 | |
|---|---|---|---|---|
| Volume Size | 19530 | 60480 | 138649 | blocks |
| Cluster Factor | 1 | 3 | 3 | blocks |
| Maximum Files | 4880 | 7560 | 17331 | files |

5. Load the distribution

If the distribution is the "walking winchester", then loading it consists of installing the disk in the MicroVAX enclosure, removing the existing one if necessary. Instructions for the installation and removal of hard disk drives can be found in the system's Owner's Manual.

If the distribution is a backup set stored on a number of RX50 floppy diskettes, follow the instructions in the Installation chapter of the MicroVMS User's Guide.

6. Boot "NanoVMS"

This procedure departs from the normal process, since "NanoVMS" does not have a paging file or a system-parameter file, nor can it execute the full system startup command procedure since many of the commands in it try to invoke image (.EXE) files and libraries that are not on the disk. A number of error messages (shown below) will be displayed on the console terminal. These are normal for "NanoVMS". To boot the system, you must use the conversational boot by typing the commands

```
>>> B/1 DUAn

2..1..0..

%SYSBOOT-E-Unable to locate file
SYSBOOT> SET STARTUP_P1 "MIN"
SYSBOOT> SET VAXCLUSTER 0
SYSBOOT> CONTINUE
```

Note

If your system is a MicroVAX I, you will be prompted for the time and date before the "MicroVMS Version 4.1M" text appears.

```
MicroVMS Version 4.1M 13-MAY-1985 22:29

%SYSINIT-E- lookup failure on paging file, status = 00000000
%DCL-W-ACTIMAGE, error activating image SETP0
-CLI-E-IMGNAME, imagefile DUAn:[SYS0.SYSCOMMON.][SYSEXE]SETP0.EXE;
-RMS-E-DNF, directory not found
-SYSTEM-W-NOSUCHFILE, no such file
%RMS-E-FNF, file not found
%SET-I-NOMSG, Message number 007781B3
SYSTEM job terminated at 8-AUG-1985 09:39:05.01
```

6. Boot "NanoVMS" (continued)

At this point, press carriage return and you will be prompted with "Username:". Type any non-null alphabetic string, followed by a carriage return. You will then be prompted twice with "Password:". Respond both times with a carriage return. The familiar "$" DCL prompt will then appear. The default directory is SYS$SYSTEM.

7. Further steps...

The only DCL commands available at this point are COPY, BACKUP, RUN, MOUNT and DISMOUNT, and a very small subset of the SET commands. If you have access to a second winchester with a full MicroVMS system on it, you can mount it and copy additional files to your "NanoVMS" disk. You can also use BACKUP/SELECT to copy selected files from the MicroVMS distribution floppies. Remember to try the commands first before walking away, since sometimes additional files (primarily in sys$library and sys$message) are needed.

A paging file is often needed, particularly for applications which set up large arrays or data buffers, especially when the system has limited physical memory. DECnet requires a 1000 block paging file just to initialize itself. The file can be created by the following commands:

```
$ RUN SYSGEN
SYSGEN> CREATE PAGEFILE.SYS /SIZE=1000 /NOCONTIGUOUS
SYSGEN> EXIT
```

To make it less painful to reboot, and to allow you to save any system tuning work, create a parameter file with the following commands:

```
$ RUN SYSGEN
SYSGEN> SET STARTUP_P1 "MIN"
SYSGEN> SET VAXCLUSTER 0
SYSGEN> WRITE CURRENT
SYSGEN> EXIT
```

The next time you reboot, you need only type ">>> **B DUAn**".

In order to have a secure system, or to have incoming DECnet access, or to support login command files or to run turnkey applications, you will probably want a user authorization file. To support this, copy the following files from the MicroVMS distribution or from a MicroVMS system disk: sys$system:AUTHORIZE.EXE, sys$library:MTHRTL.EXE and sys$library:PLIRTL.EXE and INSTALL sys$library:SECURESHR.EXE with the /prot/shar/open options. When running AUTHORIZE for the first time, you will create the authorization file SYSUAF.DAT. You will have to reboot before you can log in successfully.

DIRECTORIES OF FILES NEEDED FOR "NanoVMS"

Directory DUA1:[SYS0.SYSEXE]

```
BACKUP.EXE;1        189 ! Required for adding additional files
                        ! to basic "NanoVMS" from backup sets
COPY.EXE;1           58 ! Required for adding additional files
                        ! to basic "NanoVMS" from VMS volumes
DCL.EXE;1           132 ! Required by STARTUP.COM and subsequent
                        ! user command execution
DISMOUNT.EXE;1        8 ! Required to remove auxiliary volumes
DUDRIVER.EXE;1       27 ! Disk controller protocol driver
F11BXQP.EXE;1       107 ! Files-11 server
FPEMUL.EXE;1         20 ! Floating-point emulator
INSTALL.EXE;1        46 ! VMOUNT.EXE must be INSTALLed to be run
JOBCTL.EXE;1        102 ! Required to create LOGINOUT as detached process
LOGINOUT.EXE;2      103 ! Permits logins after STARTUP.COM exits
PUDRIVER.EXE;1       13 ! Disk controller port driver
RMS.EXE;1           211 ! RMS-32 server - required to find and open files
RUNDET.EXE;1         14 ! Required to run JOBCTL as detached process
SCSLOA.EXE;1          8 ! Required when using MSCP system disks
SET.EXE;1           167 ! Required to enable interactive logins
STARTUP.COM;1        17 ! Sets up LOGINOUT and system logical symbols
SYS.EXE;1           344 ! The operating system image (except for
                        ! device and file support, instruction emulation)
SYSBOOT.EXE;1        87 ! The primary bootstrap
SYSGEN.EXE;1        115 ! Required to alter and save system parameters
SYSINIT.EXE;1        87 ! The first image to be run as a process
SYSLOAUV2.EXE;1      15 ! MicroVAX II processor-specific initialization code
TTDRIVER.EXE;1       45 ! The terminal driver
VAXEMUL.EXE;1        23 ! Emulates non-floating-point VAX instructions
VMOUNT.EXE;1         16 ! Required to mount auxiliary volumes

Total of 24 files, 1954 blocks.
```

Directory DUA1:[SYS0.SYSLIB]

```
DCLTABLES.EXE;3  248 ! Required by sys$system:DCL.EXE
DISMNTSHR.EXE;1   11 ! Required by sys$system:DISMOUNT.EXE
ENCRYPSHR.EXE;1   18 ! Required by sys$system:BACKUP.EXE
LBRSHR.EXE;1      76 ! Required by sys$system:INSTALL.EXE
LIBRTL.EXE;1     128 ! Required by sys$system:JOBCTL.EXE
LIBRTL2.EXE;1     39 ! Required by sys$system:JOBCTL.EXE
MOUNTSHR.EXE;1   120 ! Required by sys$system:VMOUNT.EXE
SCRSHR.EXE;1      21 ! Required by sys$system:SYSGEN.EXE
SECURESHR.EXE;1   58 ! Required by sys$system:BACKUP.EXE

Total of 9 files, 719 blocks.
```

DIRECTORIES OF FILES NEEDED FOR "NanoVMS" (continued)

```
Directory DUA1:[SYS0.SYSMGR]

ACCOUNTNG.DAT;1    5 ! This file is created by LOGINOUT.EXE

Total of 1 file, 5 blocks.

Directory DUA1:[SYS0.SYSMSG]

SYSMGTMSG.EXE;1   49 ! Required for intelligible error messages
SYSMSG.EXE;1     268 ! Required for intelligible error messages

Total of 2 files, 317 blocks.

Grand total of 4 directories, 36 files, 2995 blocks.

Not shown:
     Index file (see text)
     Other files produced by volume initialization
     Directory files (average 5 blocks each)
     Page file (see text)
     Sysgen parameter files (15 blocks each)
```

NOTE

This MicroNote describes a minimal VAX/VMS system which is not in any way warranted or supported by Digital Equipment Corporation - it reports ongoing research by the author, and represents solely his conclusions and opinions, not those of Digital Equipment Corporation. The minimal VAX/VMS system described here can only be used on MicroVAX computer systems which are licensed to run MicroVMS. It is composed of files which are normal components of MicroVMS V4.1M. Earlier or later versions of MicroVMS may not successfully execute in the described subset environment.

| uNOTE # 038 |
|---|

| Title: DECnet Down-line Loading | Date: 26-Jul-85 |
|---|---|
| Originator: Scott D. Blessley | Page 1 of 9 |

This article overviews the downline load process used by DECnet for remote loading of PDP-11 and VAX processors. Downline loading is a versatile and complicated process. This article is intended as an introduction/aid to the process, not a tutorial. An extensive reference list appears at the end of the article for readers interested in pursuing the subject further.

# 1 OVERVIEW OF THE DOWNLINE LOAD PROCESS

DECnet downline load is the set of hardware and software features which allow complete systems (RSX-11S and VAXELN) to be loaded into remote, potentially unattended processors. In addition, RSX-11S offers the capability to: dynamically load tasks over comm lines, checkpoint tasks out of memory over the line, and upline dump a failed operating system. RSX-11S functionality is a subset of those features found in RSX-11M. VAXELN also features a symbolic debugger that allows debug of the target from the host processor.

# 2 DEFINITIONS

## 2.1 HOST

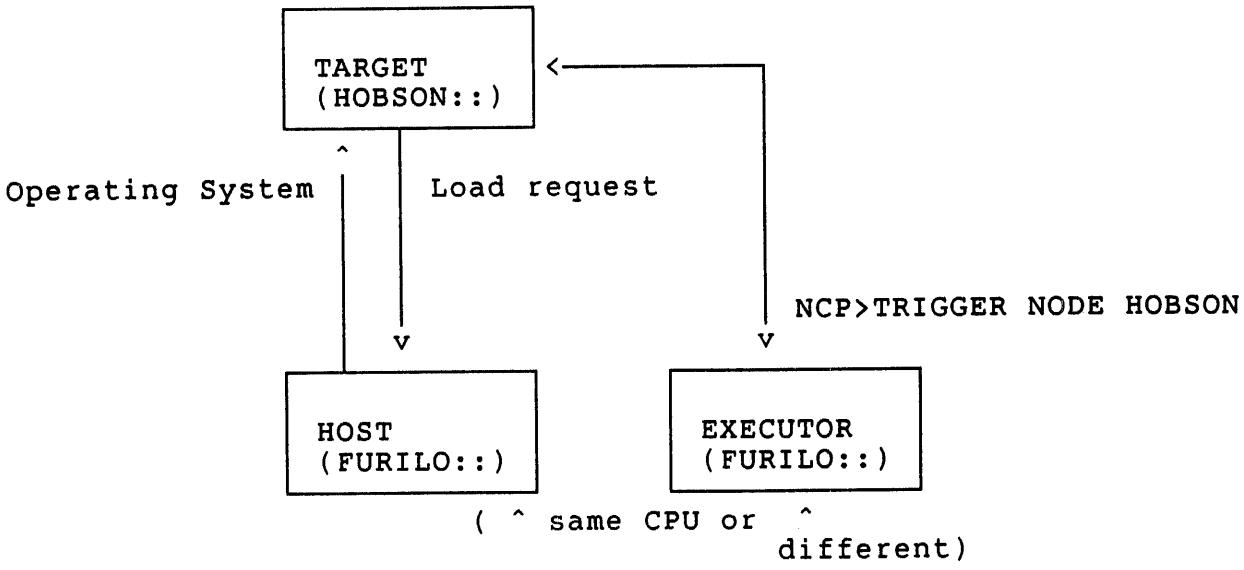
The host [node] (for this discussion) refers to both the machine which originates the load, as well as the machine which loads the software. In actuality, these need not be the same machine.

## 2.2 TARGET

The target [node] is the machine being loaded with new software. It must be powered up, but may or may not have to be awaiting primary bootstrap, depending on its communications bootstrap device and boot options.

## 2.3 EXECUTOR

The executor [node] is the node which initiates the commands to downline load a target. It need not be the same as the host.




Relationship between host, target, and executor nodes

## 3 PREPARING FOR DOWNLINE LOAD

There are three overall steps:

- o Configuring the communications hardware - both on the host and target systems
- o Configuring the DECnet software and the operating system image that will be loaded.
- o Verification and testing.

The most frequent problem is the failure to correctly configure the host and/or target hardware. There are two principal places for error:

- setting DIP switches inappropriately resulting in incorrect line speeds, wrong power-up boot options (BSEL switches), etc.
- Skew between hardware configuration and software (for example mis-specifying the vector/CSR or the Ethernet address for the target device). Be sure to have the latest version of the hardware manuals.

Specific points to be careful about:

1. Vector/CSR

2. B[oot]SEL[ect] switches - these specify under what conditions the comm interface will force a bootstrap.

3. Service password - this is a protection feature to lessen the likelihood of an inadvertent or malicious downline load.

4. Ethernet address (Ethernet circuits only) - the unique address that the target Ethernet device will respond to.

5. Service circuit - this identifies at the end of which wire to expect the target.

6. Enable SERVICE for the circuit being downline loaded. This enables the DECnet software to start the downline load process on the host.

7. Make sure event logging is enabled, or you'll miss error messages.

8. It may seem obvious, but make sure that the target comm device you're using, and the processor bootstrap are *capable* of doing what you ask. For details, see uNote #015, "Q-bus Hardware Bootstraps"

## 4 RSX-11S OVERVIEW

Configure the RSX-11S image on a VAX/VMS, RSX-11M or RSX-11M+ system, using the normal tools (SYSGEN, MAC, TKB, VMR). Prepare and build the host loader table. (Refer to the References section for documentation on RSX downline load procedures)

## 5 VAXELN OVERVIEW

The program/VAXELN image development sequence is:

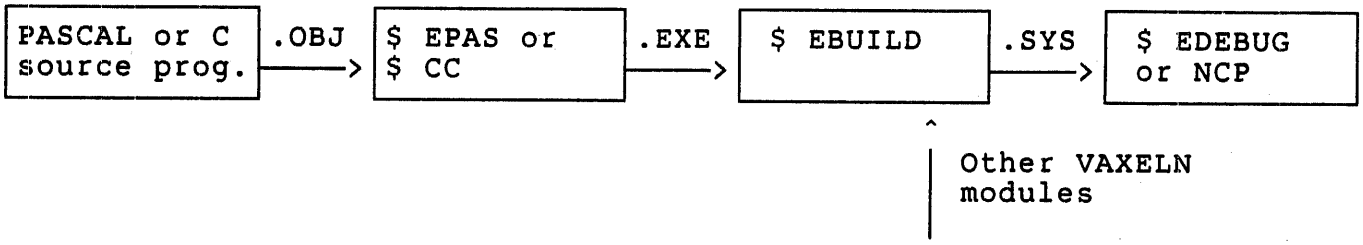
1. Develop the source modules

2. LINK the source modules with the RTLSHARE.OLB and RTL.OLB libraries

3. EBUILD the LINKed image to produce a bootable VAXELN system. To be able to downline load the VAXELN image, you must select Boot Method DOWNLINE in the VAXELN System Characteristics portion of EBUILD.

4. Downline load the target over Ethernet (this is not the only method of loading a VAXELN target). The target can be loaded through EDEBUG commands, or via NCP LOAD/TRIGGER commands.

5. Debug the application and repeat steps 1-4.




VAXELN Program development sequence

## 6 OBSERVING & DEBUGGING

There are a variety of error indications, and diagnostic aids available. Some are at the hardware level (comm device diagnostic LEDs), some from the host software (upline dump for RSX-11S, EDEBUG for VAXELN). Others are provided by the DECnet components that handle downline load, through the EVL (EVent logger) capability. Some diagnostic facilities:

1. Device status LEDs give a numeric indication as to potential hardware, or hardware configuration problems with the interface or its connections.

2. Event logging messages:

   - Event code 0.3 - Auto service: indication that a portion of a DLL is taking place.

   - Event code 0.7 - Aborted service request: indication that a portion of a DLL load request is failing. This is not always bad. For example, it is an acceptable "error" when multiple hosts (on an Ethernet) offer service and the target accepts only one. The remaining systems report event 0.7. EVL messages come with varying amounts of diagnostic information. A 0.3 or 0.7 event includes the type of load being requested, from where it is requested, to whom it is requested, etc.

3. Failure message from NCP on LOAD/TRIGGER commands - may give some insight as to the source of the problem

4. Line/circuit counters - indicate how much data has been transmitted and received , and what errors have occurred.

Remember: Enable event logging: NCP SET LOGGING <facility> KNOWN EVENTS STATE ON. Otherwise, EVL will not display this diagnostic information!

## 7 EXAMPLES

One the next three pages are examples showing the successful load of a terminal server and a VAXELN target. The appearance of the output will be approximately the same for your targets, only with different node names and files.

```
$ reply/enable=network

%%%%%%%%%%  OPCOM  26-JUL-1985 13:09:11.96  %%%%%%%%%%
Operator _FURILO$RTA1: has been enabled, username BLESSLEY
%%%%%%%%%%  OPCOM  26-JUL-1985 13:09:11.99  %%%%%%%%%%
Operator status for operator _FURILO$RTA1:
NETWORK

$ run sys$system:ncp
NCP>show node hobson characteristics      !Hobson is a DECserver 100

Node Volatile Characteristics as of 26-JUL-1985 13:10:49

Remote node =   13.102 (HOBSON)

Service circuit           = UNA-0
Hardware address          = 08-00-2B-02-0F-5F
Load file                 = SYS$SYSROOT:[DECSERVER]PS0801ENG.SYS
Dump file                 = SYS$SYSROOT:[DECSERVER]PSDMP0F5F.SYS

NCP>trigger node hobson
NCP>^Z

%%%%%%%%%%  OPCOM  26-JUL-1985 13:10:13.16  %%%%%%%%%%
Message from user DECNET on FURILO
DECnet event 0.3, automatic line service
From node 7.272 (FURILO), 26-JUL-1985 13:10:13.04
Circuit UNA-0, Load, Requested, Node = 13.102 (HOBSON)
File = MOM$LOAD:PS0801ENG, Operating system, Ethernet address
= 08-00-2B-02-0F-5F

%%%%%%%%%%  OPCOM  26-JUL-1985 13:10:18.08  %%%%%%%%%%
Message from user DECNET on FURILO
DECnet event 0.3, automatic line service
From node 7.272 (FURILO), 26-JUL-1985 13:10:16.40
Circuit UNA-0, Load, Successful, Node = 13.102 (HOBSON)
File = MOM$LOAD:PS0801ENG, Operating system,
Ethernet address = 08-00-2B-02-0F-5F
```

DECserver-100 Downline Load Example

Below is an example of down-line load of a MicroVAX II system using EDEBUG. The first part of the illustration shows the DECnet volatile database entries for the target, as seen from the system that ultimately boots the target, and from another that fails to boot the target.

```
[This command is issued on system "BELKER"]

BELKER> NCP SHOW NODE TUBBS CHARACTERISTICS

Node Volatile Characteristics as of  7-AUG-1985 10:38:59

Remote node =   7.453 (TUBBS)

Service circuit              = UNA-0
Hardware address             = 08-00-2B-02-1C-63
Load file                    = DISK$USER:[SAMPLE.ELN]CHER.SYS;

[This command is issued from system "FURILO"]

FURILO> NCP SHOW NODE TUBBS CHARACTERISTICS

Node Volatile Characteristics as of  7-AUG-1985 10:40:52

Remote node =   7.453 (TUBBS)

Load file                    = DISK$USER:[SAMPLE.ELN]CHER.SYS;
```

---

Command to downline load the VAXELN image from EDEBUG:

```
[This command and related output is from system "BELKER"]

$ EDEBUG/LOAD=PROG1 TUBBS

Edebug V2.0-00
Loading "TUBBS"
Connecting to "TUBBS"

.
.
.
(screen mode debugger)
```

Event log messages resulting from the EDEBUG command:

```
DECnet event 0.7, aborted service request
From node 7.272 (FURILO),  6-AUG-1985 16:13:14.46
Circuit UNA-0, Line open error, Unrecognized component, Node
Ethernet address = 08-00-2B-02-1C-63
          ^ FURILO:: didn't have this address defined in its
            volatile database so it couldn't service the request
```

```
DECnet event 0.3, automatic line service
From node 7.190 (BELKER),  6-AUG-1985 16:12:51.34
Circuit UNA-0, Load, Requested, Node = 7.453 (TUBBS)
File = DISK$USER:[SAMPLE.ELN]CHER.SYS;, Operating system
Ethernet address = 08-00-2B-02-1C-63
          ^ BELKER:: did have the address, corresponding to an
            entry for "TUBBS::" Here, the target system is
            requesting an operating system load. BELKER's DECnet
            software is acknowledging the request.

DECnet event 0.3, automatic line service
From node 7.190 (BELKER),  6-AUG-1985 16:12:57.62
Circuit UNA-0, Load, Successful, Node = 7.453 (TUBBS)
File = DISK$USER:[SAMPLE.ELN]CHER.SYS;, Operating system
Ethernet address = 08-00-2B-02-1C-63
            BELKER responds with packet(s) containing the
            requested O.S. code for the target
```

## 8 SUMMARY

This Micronote has presented an overview of the downline loading process. Many "components" are involved in the process - at least two CPUs and at least two communication interfaces, target operating system and images plus host, target, and executor DECnet software. The procedure is akin to bringing up a complex, customized operating system up on a remote processor, without a local load device. Once configured, it offers the benefits of not requiring any mass storage devices (resulting in higher MTBF and lower cost), plus the ability to bootstrap a processor which my be physically inaccessible. There is a great amount of information on DECnet downline load. There is no single compendium, although each of the supported operating systems, plus VAXELN have a chapter on the subject.

## 9 REFERENCES

All of the following references are to Digital publications.

DECnet-RSX Volume IV - System Manager's Guide AA-H224C-TC

Guide to Networking VAX/VMS - AA-Y512A-TE (chapter 4)

QMA DMV11 Synchronous Controller Technical Manual EK-DMVQM-TM

DEQNA User's Guide EK-DEQNA-UG

DECnet Digital Network Architecture (Phase IV) General Description AA-N149A- TC

DECnet Digital Network Architecture (Phase IV) Maintenance Operations Functional Specification AA-X436A-TC

Networks and Communications Buyer's Guide ED-26347

"VAXELN Installation and Programming" AA-Z454A-TE

"MicroVax I Owner's Manual" EK-KD32-OM

"MicroVAX I Owner's Manual" (Release Notes) EK-KD32A-OM-CN1

Micronote #015, "Q-Bus Hardware Bootstraps"

Software Product Descriptions:

DECnet-11S #10.74

DECnet-11M #10.75

DECnet-11M+ #10.66

DECnet-VAX #25.03

| uNOTE # 039 |
|---|

| Title: Differences between KDJ11-A and KDJ11-B | Date: 8-Aug-85 |
|---|---|
| Originator: Peter Kent | Page 1 of 5 |

## Purpose

The purpose of this MicroNote is to identify and discuss the differences between the KDJ11-A and KDJ11-B CPU modules.

The table that follows lists the differences between the CPU modules. Differences that require explanation follow the table and are marked *.

| FEATURE | KDJ11-A | KDJ11-B |
|---|---|---|
| Cache * | Single tag | Dual tag |
| PMI support * | No | Yes |
| On-board bootstrap * and diagnostic ROM | No | Yes |
| Boot/diagnostics * control status reg. | No | Yes |
| Boot page * control register | No | Yes |
| Boot configuration * and display register | No | Yes |
| Instruction implementation differences * | | |
| DCJ-11 speed/FPJ11 differences * | | |
| Backplane compatibility * | | |
| Maintenance register differences * | | |

cont'd

The differences between the CPU modules. cont'd:

| FEATURE | KDJ11-A | KDJ11-B |
|---|---|---|
| Module I.D. | M8192 | M8190 |
| Size | Dual | Quad |
| Power: 5 volt (12 Volt) | 4.5 | 5.5 (0.2) |
| AC loading | 3.4 | 2.3 |
| Console serial line | No | One |

Cache

For a full discussion of cache memory as used on the KJD11-A and KDJ11-B refer to MicroNote #9 and the KDJ11-A and KDJ11-B User's Guides. Both CPU modules have a similar cache organization using a nine bit tag. This nine bit field contains information that is compared to the address label, which is part of the physical address. When the physical address is generated, the address label is compared to the tag field. If there is a match it can be considered a hit provided that there is entry validation and no parity errors. The KDJ11-B has an additional tag store called the DMA tag. The DMA tag is an identical copy of the cache tag store and is used to monitor the main memory DMA updates while the cache tag store monitors the DCJ11 requirements. The presence of the second tag store - DMA tag - allows the J-11 microprocessor to continue processing after it has relinquished the system bus to a DMA device. When the DMA tag detects a hit (main memory location written to by the DMA device), the microprocessor stops and relinquishes the internal bus to the cache controller to allow it to monitor further DMA activity on the bus. The KDJ11-A, however, has only one tag store and stops processing as soon as it relinquishes the system bus to a DMA device.

PMI support

The PMI or Private Memory Interconnect is described in MicroNote #30. The PMI consists of 14 unique signals which use the CD interconnect side of the backplane of certain Q-bus backplanes. PMI is used only with the KDJ11-B and MSV11-J. PMI DATI and DATO bus transactions between the KDJ11-B and MSV11-J are more than twice as fast as those between non-PMI CPU and memory configurations. The KDJ11-A does not offer a PMI capability.

The Unibus adaptor used with PDP11/84 systems enables the Unibus map if a particular PMI signal - PMAPE (Unibus map enable) - is asserted and disables the Unibus map when PMAPE is negated. The memory modules associated with PMI (MSV11-J) do not use this signal. PMAPE is asserted if Memory Management Register 3, bit 05 is set, and negates this signal

if MMR3 is clear. If there are devices which require this signal to work, the KDJ11-A will not and cannot (without warranty violating modifications) be made to issue this signal.

## KDJ11-B boot/diagnostic ROM

The are basically 3 parts to the ROM code. The first part is the diagnostic tests which are run at power up or at the initiation of the operator. These tests perform checks on the CPU module, the memory module(s), and the Unibus adaptor for Unibus systems. The second part of the ROM is the boot code. The following devices can be booted from the KDJ11-B: RA80/81/60, RD51/52, RX50, RC25, RL01/02, RX01/02, TU58, RK05, TK25/50, TS05, TU81, DEQNA, DECnet DUV11, DECnet DLV11-E, DECnet and DLV11-F. The boot code can also start programs stored in the EEPROM or programs stored in M9312 type boot ROMs located on the KTJ11 Unibus adaptor module. The third part of the the code allows the storing and modification of parameters for the CPU, the Unibus adaptor, and the system. This portion of the boot code also provides support for the EEPROM itself. The user can also create (using a machine code editor) his own custom boot code and save this code in the EEPROM.

## Boot and diagnostic controller status register

The boot and diagnostic controller status register (BCSR) is both word and byte addressable. The BCSR allows the boot and diagnostic ROM programs to test battery backup and reboot status, to set parameters for the PMG (processor mastership grant) counter and the line clock, to enable the console halt on break feature and to enter or exit from stand alone mode.

## Boot page control register

The page control register is a read/write register which is both byte and word addressable. The PCR high byte provides the most significant ROM address bits when the 16 bit ROM sockets are accessed by bus address 1777300-17773776. The PCR low byte provides the most significant ROM (or EEPROM) address bits when the 16 bit or 8 bit ROM sockets are accessed by addresses 17765000-17765776.

## Configuration and display register

The configuration and display register reflects the status of the eight switches edge mounted at the top of the module. It also allows boot and diagnostic programs to update the 8 bit LED display located at the top of the KDJ11-B module.

## Instruction set differences

Instructions which are required to do a read-modify-write are implemented differently on the KDJ11-A and KDJ11-B. There are only 3 instructions which are defined by the PDP11 architecture to be uninterruptible during its read- modify-write. They are the TSTSET,

WRTLCK, and ASRB instructions. The KDJ11-A will implement these read-modify-write instructions differently than a KDJ11-B. The KDJ11-A processor uses the AIO signal outputs of the J11 to determine whether it performs either a 1) DATIO cycle or 2) a DATI cycle followed by a DATO cycle. The KDJ11-A will ONLY do a DATIO Q-bus cycle when it executes a TSTSET, WRTLCK or ASRB instruction. Past implementations of the PDP-11 architecture have also implemented other instructions doing a read-modify-write cycle as being uniterruptable. The BIS (Bit Set) instruction will be used as an example. This instruction requires the CPU to READ a word from memory, possibly MODIFY that word, then WRITE the word back to memory. A KDJ11-B uses a Q-bus DATIO cycle to implement this instruction. Therefore, the instruction is not interruptable between doing the READ and the WRITE. When it executes other instructions which want to do a read-modify-write operation like the BIS instruction, it will use two separate Q-bus cycles. This implementation allows for an interrupt or DMA request to be granted between the DATI and the DATO (case 2 above). There are third party vendors whose equipment assume that a BIS instruction will use a DATIO bus cycle. Those devices will work fine in a system with a KDJ11-B, but will work intermittently in a system with a KDJ11-A because what they assume to be uninterruptible is now interruptible and affects their on-board firmware.

Speed and the FPA

| | 15 MHz | 18 MHz | FPA compatible | FPA on board | system |
|---|---|---|---|---|---|
| KDJ11-AA | Yes | No | No | No | Q18/Q22 |
| KDJ11-AB | Yes | No | Yes | No | Q18/Q22 |
| KDJ11-AC | Yes | No | Yes | Yes | Q18/Q22 |
| KDJ11-BB | Yes | No | Yes | No | 11/73 |
| KDJ11-BC | Yes | No | No | No | 11/73 |
| KDJ11-BF | No | Yes | Yes | Yes | 11/83 |

Notes on the above table

The 15 MHz or 18 MHz refers to the crystal frequency at which the DCJ-11 will run. FPA means the FPJ11 floating point accelerator chip. Refer to MicroNote #25 for more information on upgrading with the FPJ11. The KDJ11-A is a CPU module that is not sold as part of a package system. The reference to Q18/Q22 refers to the fact that the KDJ11-A can be used in any 18 or 22 bit Q-bus backplane. The notation 11/73 means that the indicated KDJ11-B CPUs are used with the MicroPDP-11/73 systems and the indicated KDJ11-B CPU are used with the MicroPDP-11/83 system.

## Backplanes

The KDJ11-A CPU may be used in any Q18/Q22 slot. The KDJ11-B, being a quad module must be accomodated in a backplane which has Q-bus in AB slots and the CD interconnect in the CD slots. The KDJ11-B cannot be used in a backplane (or that part of the backplane) where there is Q-bus in both AB and CD slots.

## Maintenance register differences

The maintenance register contains the following information in both the KJD11-A and KDJ11-B: POK (power ok), power up mode selected, HALT status, Module ID, FPA available, and Boot address. The module ID number is a 4 bit code that differs between the KDJ11-A and KDJ11-B. The ID number for the KDJ11-A is 0001 and 0002 for the KDJ11-B.

| uNOTE # 040 |
|---|

| Title: FPJ11 Theory of Operation | Date: 17-SEP-85 |
|---|---|
| Originator: Bill Jackson | Page 1 of 4 |

The goal of this MicroNote is to introduce the FPJ11, a floating point coprocessor to the DCJ11, and to explain the interprocessor communication between the FPJ11 and DCJ11. Figure 1 shows a typical DCJ11 based system which includes the FPJ11. For a discussion of FPJ11 support on the KDJ11-A processor, see MicroNote #025.

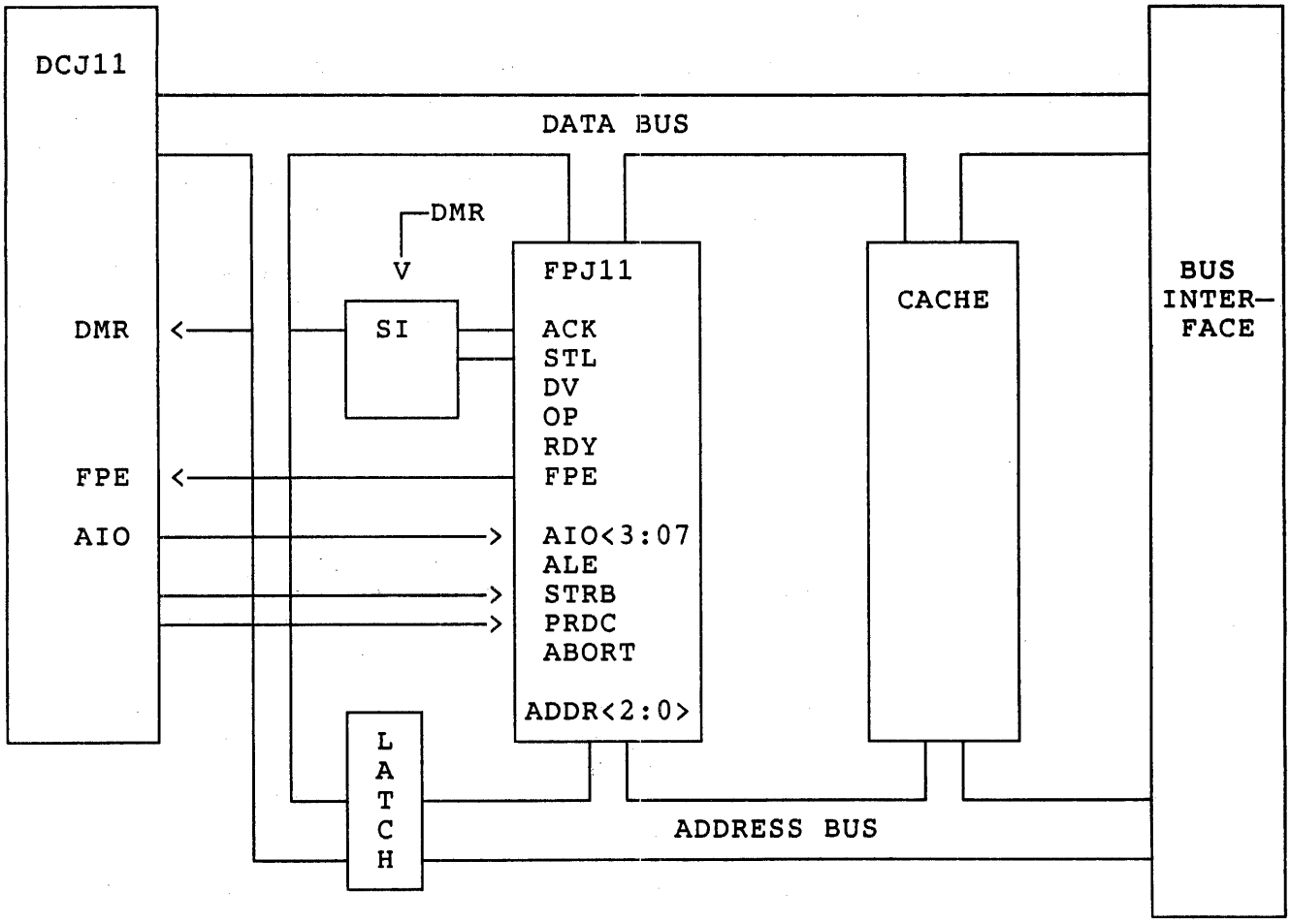



Figure 1 - Typical DCJ11/FPJ11 Application

The FPJ11 is a single chip floating point accelerator for the DCJ11 microprocessor. Its coprocessor interface, along with optimizations within the chip allow for 5 to 8 times acceleration over the DCJ11 microcoded floating point performance. The FPJ11 provides

- o Complete FP11 floating point instructions (46)
- o Two FP11 floating point data types (F and D)
- o Six 64-bit floating point accumulators
- o FEC floating exception code register

The FPJ11 interfaces as a true Co-processor. The Bus Interface Unit or BIU) inputs all instruction stream data and decodes instructions in parallel with the DCJ11. Support microcode in the DCJ11 initiates all I/O cycles required by the FPJ11. This Co-processor interface allows overlap of floating point instruction executing in the FPJ11 and integer instructions executing in the DCJ11. This allows for reduced execution time by interleaving floating point and integer instructions.

The interface to the FPJ11 involves several DCJ11 signals which indicate the state of the DCJ11 processor, and synchronize the two processors. Table 1 lists the signals and their use in the FPJ11 interface, Figure 1 shows their use.

TABLE 1
DCJ11/FPJ11 signals

| DCJ11 signal | Use |
|---|---|
| AIO<3:0> | Indicate to FPJ11 current DCJ11 cycle type |
| PRDC | signal instruction decode to FPJ11 |
| STRB | signal beginning of bus cycle to FPJ11 |
| DMR | used by FPJ11 to stall the DCJ11 |
| FPE | indicate to DCJ11 a floating point exception |
| ALE | used to latch cache hit data (trailing edge) |
| DV | used to latch non-cache data (trailing edge) |
| DAL<2:0> | indicate GPREAD and GPWRITE information |
| FPA-OP | signal to SI that writes come from FPA |
| FPA-STL | used to stall DCJ11 during multiple FPA instructions |
| FPA-RDY | indicates FPA data will be ready in 125ns |
| FPA-ACK | enable FPA output drivers |

The DCJ11 supports several types of bus operations in communicating with the external system. Since the FPJ11 relies on the DCJ11 to initiate

all I/O cycles, the FPJ11 will monitor DCJ11 I/O cycles for activity. When specific I/O cycles occur, the FPJ11 will 'wake up' and start processing. A subset of the DCJ11 bus cycles are used by the FPJ11 in communicating with the DCJ11 and the system interface (SI). These bus cycles are listed in Table 2. The bus read and write cycles are used to read/write data to/from memory (or cache). The GP transactions are used for interprocessor communications between the DCJ11 and FPJ11.

Table 2
DCJ11 bus operations

| Cycle | Description |
|---|---|
| IREAD | latched to search for FP instruction |
| DREAD | used for data fetches for FP instructions |
| WRITE | used to write FP data back to memory |
| GPREAD | used to read FP data to DCJ11 internal registers |
| GPWRITE | used to write FP data to DCJ11 internal registers |

The DCJ11 divides all bus reads into 3 categories: Instruction stream Demand READ (IDREAD), Instruction stream Request READ (IRREAD), and Data stream READ (DREAD). Instruction stream reads are used by the DCJ11 to fetch instruction stream data such as PDP-11 instructions, immediate data and absolute data. All other DCJ11 reads are classified as Data stream reads (for more information on DCJ11 bus cycles and data classification see the DCJ11 data sheet EK-26921-00)

Request reads are reads that the DCJ11 will attempt when doing internal cycles such as register transfers. This is an attempt at filling the DCJ11 4 stage pipeline. Demand reads are reads that must be completed in order to finish an instruction. The DCJ11 will always try to keep the pipeline full by doing request reads, but will do demand reads as necessary.

The most typical FPJ11 operation is the common FP11 instruction that reads some data from memory, operates on it, then writes the result back to memory. In this operation, the FPJ11 monitors the DCJ11 bus cycles for either type of instruction stream read. When the FPJ11 sees any type of Instruction stream read, it latches the data from the data/address lines (DAL) and holds it in its instruction register. The FPJ11 keeps doing this until it sees the DCJ11 indicate that it is now doing a instruction decode by the assertion of the DCJ11 signal PRDC. The FPJ11 then does a parallel decode of the instruction and checks if it is a floating point instruction. If the instruction is not a floating point instruction, the FPJ11 'goes to sleep' and continues to latch all I-stream read data. If the instruction is a floating point instruction, the DCJ11 will initiate all bus cycles while the FPJ11 will remove data from the bus. The FPJ11 will then do the floating point

operation. If the operation is a store type operation the DCJ11 will initiate the bus write operation signaling to the FPJ11 to write data to memory.

When either the source or destination of the floating point instruction is a DCJ11 general purpose register, a GP cycle will be used to access the DCJ11 register. Load type operations would use the GPWRITE bus cycle to write the contents of the DCJ11 register to the FPJ11. GPREAD operations are used to read data from the FPJ11 and deposit it into the DCJ11 general purpose registers.

Because of the overlapping instruction capability, the DCJ11/FPJ11 combination can start to fetch operands for a next floating point operation while one is executing. Because there is a single datapath internal to the FPJ11, this prefetched data must not get to the execution unit until the current operation is finished. The FPJ11 will output FPA-STL to the DCJ11 if the current operation will not be completed before all of the data is ready. This mechanism guarantees that the datapath will only be used by one floating point operation at a time. When the FPJ11 is done with the current operation the DCJ11 is continued, and the floating point operation proceeds.

| uNOTE # 041 |
|---|

| Title: Device Ordering Chart for Q-bus Systems | Date: 16-Sep-85 |
|---|---|
| Originator: Jack Howes and Peter Kent | Page 1 of 6 |

## Primary Device Ordering Determination

DMA devices as well as interrupt devices on the Q-bus are position dependant. That means that the order in which devices are placed in the bus relative to the CPU determines in what order their DMA (or interrupt) requests will be serviced.

The primary factor in determining the device sequencing order is the length of time that each device can wait to become bus master without error. These errors normally occur when a controller's data buffer fills to capacity before the device connected to it has finished its transfer. Generally, the cause of this is a higher priority device (or devices) transferring data over the Q-bus, and the controller that gets the error is blocked from becoming bus master. Characteristics that influence whether a device will fail in this way are: the on board buffer size that a controller/device has, the intelligence of the controller/device, and the transfer speed of the device connected to the controller.

## Methodology

There has been an in-house test instrument developed which can detect the failure of Q-bus devices when they cannot get bus service. This measurement is the length of time a device can wait before getting a data late or device timeout error. The test instrument can be programmed to hold the Q-bus, per acquisition, for any time between 1 microsecond and 3.9 milliseconds. It runs in conjunction with the device under test. During the test process the length of time the test instrument holds the Q-bus is varied until the device under test fails. This measurement is called "latency tolerance". Holding the Q-bus for 3.9 milliseconds is equivalent to the test instrument transferring 4095 words (non-block mode DMA) per assertion of BSACK.

## Secondary Device Ordering Determination

The large buffer sizes and intelligence built into some of the newer controllers make them less susceptible to data late or device time out errors. These devices do not get an error waiting for the Q-bus up to 3.9 milliseconds and consequently are ordered by other criteria:

1. The type of Q-bus transfer they do (block mode, burst mode or single cycle). See MicroNote #12 for a description of Q-bus DMA transaction types.

2. If interrupted by a lower priority DMA device will they pass the Q-bus (DMG) grant signal, when they are doing a blockmode transfer.

3. The effect the ordering has on device and system performance.

4. The power consumption and/or cabling requirements of the device.

## Examples:

- A magtape advertised as a "streaming" tape drive may not stream if it is assigned a lower priority than a device that utilizes a significant amount of bus time. In this instance the tape drive will be ordered ahead of a device that it would normally follow according to its latency tolerance.

- A device that consumes a substantial amount of power may have to be configured in an expander box with another power supply for practical reasons, even though this device would normally precede other devices.

The following pages contain the recommendations for the order of devices on the Q-bus. Also contained in these pages are the measurements taken and the reasons for suggesting the guidelines. This ordering table is only a guideline for Q-bus system configurations, and it should be noted that a system will work satisfactorily in many other configurations as well. Additionally, customers may alter the configurations to better meet their specific application needs.

The measurement process to determine the device sequence chart has been an evolving one and as such not all the devices on the chart contain the same data. The measurements on this chart were only true for the system that they were measured on. These measurements will vary from system to system dependent upon the memory type, system architecture (mapped or unmapped DMA) and the speed of each CPU's arbitrator logic. The variations in measurements that can occur between CPU's should not theoretically effect the bus placement of each device.

Table 1 lists the order in which devices should be placed in the Q-bus. Please refer to the detailed information about each device that follows the table.

Table 1

1. TSV05 9 track magtape controller
2. DMV11 Microprocessor controlled DECnet communications interface
3. TQK25 Controller for 8 inch magtape drive
4. DHV11 Microprocessor controlled communications multiplexor
5. DEQNA Ethernet controller
6. TQK50 Controller for single spindle cartridge magtape
7. RLV12 Controller for 14 inch RL series disk drives
8. RQDX3 Controller for 5 1/4 inch RD/RX drives
9. KDA50 Controller for 14 inch RA series disk drives
10. RQC25 Controller for 8 inch RC25 series disk drives
11. RQDX2 Controller for 5 1/4 inch RD/RX drives
12. DRV11-WA General purpose 22 bit DMA interface

Table 2 lists the transfer time and time between requests for DATI and DATO cycles and latency tolerance. All times are microseconds.

Table 2

| Device | No. words transferred | Transfer time DATI | Transfer time DATO | Time between requests DATI | Time between requests DATO | Latency tolerance |
|---|---|---|---|---|---|---|
| TSV05 | 1 word | 2.8 | 2.8 | 8.7 | | 12 |
| DMV11 | 1 word | 3.1 | 3.1 | 280 @ 56 Kbits | | 175 |
| TQK25 | 4 word block mode | 5.1 | | | | 280 |
| | 4 W single & burst mode | | 7.25 | | | |
| DHV11 | 1 word | 2.15 | 2.15 | | | 1200 |
| DEQNA | 16 word block mode | 11.78 | 13.87 | 5.1 | 5.1 | = or > 3900 |
| TQK50 | 4 word burst | 9.49 | 10.7 | 24.35 | 20.63 | = or > 3900 |
| RLV12 | 4 word burst | 7.09 | 7.8 | 5.92 | 5.7 | = or > 3900 |
| RQDX3 | 16 word block mode | 13.5 | 12.7 | 4.48 | 4.48 | = or > 3900 |
| KDA50 | 8 word block mode | 9.0 | 8.9 | 6.68 | 6.68 | = or > 3900 |
| RQC25 | 2 X 8 word block mode | 14.84 | 16.52 | 15.41 | 14.41 | = or > 3900 |
| RQDX2 | 16 word block mode | 15.23 | 15.48 | 1.7 | 1.7 | = or > 3900 |
| DRV11-WA | 4 word burst | 6.6 | 7.0 | 0.17 | 0.17 | = or > 3900 * |

* for DEC/X11 only

## Additional device information

### TSV05

This device does not do 16 word block transfer so it doesn't monitor the Q-bus BDMR (DMA request line) signal.

### DMV11

This device does not do 16 word block transfers, so it doesn't monitor the Q-bus BDMR signal.

Latency Tolerance was determined while running DECNET on a uVAX-I. The following error occurred when the DMV11 was held off the bus for more than 175us: "COPYEOPENIN RMS-F-SYS, QIO Request Failed, SYSTEM-F-LINK exit".

### TQK25

This device does not do 16 word block transfers, so it doesn't monitor the Q-bus BDMR signal.

### DHV11

This device does not do 16 word block transfers, so it doesn't monitor the Q-bus BDMR signal. It performs DMA on output to the controller, however is interrupt driven when accepting data from the attached asynchronous lines.

### DEQNA

This device monitors Q-bus BDMR signal and passes grant when interrupted by a lower priority device. This device is placed relatively close to the CPU because it cannot quickly recover from bus latency conditions. Re-transfers over the ethernet are costly in system resources.

### TQK50

This device does not do 16 word block transfers, so it doesn't monitor the Q-bus BDMR signal.

While this tape drive has a high latency tolerance, it should be placed in front of the other devices that utilize a significant amount of bus time. Doing this enhances its ability to stream data.

## RLV12

This device does not do 16 word block transfers so it doesn't monitor the Q-bus "BDMR" signal. It is configured in this position because of its ability to avoid bus latency conditions even though it doesn't do block mode transfers.

## RQDX3

This device monitors BDMR and passes grant when interrupted by a lower priority device. This device may have to be placed as the last device in the CPU box because of cabling requirements.

## KDA50

Instead of monitoring BDMR, the KDA50 does an eight word block transfer and releases the bus between transfers to allow other devices access. Although this device works more efficiently before the RQDX2 and RC25, it may have to be configured in an expansion box due to its high power consumption.

## RQC25

This device does not monitor BDMR. It performs two consecutive eight word block transfers, during which it will not pass the grant to a lower priority device.

## RQDX2

This device monitors BDMR however it does not pass grant to a lower priority device when interrupted. It holds the Q-bus from a lower priority device for 288us average. In a dual expander box system this device may have to be configured in the first expansion box due to cabling requirements.

## DRV11-WA

This device does not do 16 word block transfers so it doesn't monitor the Q-bus BDMR signal. The DRV11-WA may "monopolize" the bus if traffic to/from the device is sufficiently high. The placement of this device is very dependent upon the customers application. For DEC/X11 and diagnostic testing it should be configured as the last device on the Q-bus.

# APPENDIX A

## TABLE OF CONTENTS
## ORIGINAL MICRONOTES

052 PARITY MEMORY IN LSI-11/23 SYSTEMS
053 PDP-11 FAMILY DIFFERENCES
054 MXV11 CONFIGURATION
055 LSI-11 VS. LSI-11/23 BUS TIMING
056 DLV11-J CABLING
057 LOCATION OF W13 ON THE BDV11
058 CONFIGURING MEMORY FOR LSI-11 SYSTEMS WITH MORE THAN 64K BYTES
060 MAXIMUM CONFIGURATION OF DLV11-J MODULES
061 PROGRAMMING THE MRV11-C
062A BOOTSTRAPS FOR TU58, RL01, RK05, RX02, RX01
063A RL01 TYPE-IN BOOTSTRAP
064A DLV11-J I/O PAGE ADDRESS PROBLEM REPORT
065A BOOTSTRAP FOR RX02
066 11/123 FLOATING POINT COMPATIBILITY
067B DLV11-J RECEIVER CHIP PROBLEM
068 MICROCOMPUTER MODULE ENVIRONMENTAL CONSIDERATIONS
069 18-BIT DMA WITH CHIPKITS
070 LSI-11 VS. LSI-11/23 TRANSACTION DIFFERENCES
071 EXPANDING BA11-MA AND BA11-NC BASED SYSTEMS
072 PERIPHERAL COMPATIBILITY WITH 11/23 SYSTEMS
073 TU58 CABLING
074 MXV11-AA, -AC CABLING
075 MXV11-A2 BOOTSTRAP ERROR HALTS
076 DLV11-F CURRENT LOOP PROBLEM
077 SUMMARY OF BOOTSTRAP SOURCES
078 LSI-11/23 PROCESSOR DIFFERENCES
079 THE LSI-11/23 AND THE LSI-11/2 BUSES ARE THE SAME
080A LSI-11/23 I/O PAGE ADDRESSING
081 USE OF RECOMMENDED DISKETTES
082 HANDLERS FOR SERIAL LINE PRINTERS
083 ALTERNATE CLOCK FREQUENCIES FOR THE MXV11
084 IMPROVED DLV11-F
085 WAKE-UP CIRCUIT IMPLEMENTATIONS
086 INTERFACING TO THE TU58 W/O BREAK
087 TYPE-IN BOOTSTRAP FOR TU58
088 RT-11 V3B FB MONITOR AND TU58'S
089 STANDALONE PROBLEM LOADER
090 USING THE BA11-VA IN SMALL SYSTEMS
091 USING THE MXV11-A2 BOOTSTRAP ON THE MRV11-C
092 TWO POTENTIAL PROBLEMS WITH THE RXV21
093 USER WRITTEN SYSTEM TASKS UNDER RT-11 V4
094A RL02 SUPPORT BY DIGITAL SOFTWARE
095 VT103 APPLICATIONS FOR UNUSUAL BAUD RATES
096 TU58 TIPS
097 TU58 TAPE FORMAT AND ADDRESSING MODES
098 11/23 & 11/03 RL01 BASED PACKAGED SYSTEM EXPANSION
099 RL02 BOOTSTRAP FAILURES USING BDV11
100 UPGRADES FOR PB11
101 TU58 SYSTEM POWER PROBLEM
102 MMU CONFIGURATION JUMPERS
103 CREATING A DIAGNOSTICS DECTAPE II UNDER XXDP+
104 11/23 ECO STATUS
105 MXV11 BOOTSTRAP PROBLEMS
106 MXV11 FUNCTIONALITY

# APPENDIX B
## Subject Index - By MicroNote Number