5136-SD-VME-R

User's Guide

Version 1.03

Industrial Communication Technologies

50 Northland Road, Waterloo, Ontario N2V 1N3

(519) 725-5136 fax (519) 725-1515

Printed in Canada

Publication Name : SDVMER.DOC
Publication Revision: 1.03
Date Printed: July 23, 1998

--This Document Applies To --
5136-SD-VME-R
Interface Card

1. Introduction

1.1 Purpose of this Document

This document is a user's guide for the SST 5136-SD-VME-R remote I/O card for Allen-Bradley 1771 remote I/O. This card makes it possible for an application running on a VME host computer to communicate with Allen-Bradley PLCs and monitor 1771 remote I/O.

1.2 Card Overview

The 5136-SD-VME-R interface card provides an intelligent front end between VMEbus master cards and Allen-Bradley PLCs.

The card is a co-processor card with circuitry to provide a standard bus interface to the VMEbus backplane. The card's form factor is double-height, single-width with electrical connection to the VMEbus backplane via the P1 connector. The card acts as a slave on the VMEbus.

The card contains a Z180 processor which is loaded with software by the host computer to enable it to perform the communication tasks on the network.

The card contains no software in ROM; the appropriate interface software is downloaded to the card from the host computer.

Interaction between the task resident on the card and the application software on a VME host is made through shared memory. The lower 32 Kbytes of memory on the card are reserved for the software module which is downloaded when the card is initialized; the upper 32 Kbytes are used for data and tables.

The card occupies a 64 Kbyte block of Standard Access space for the shared memory and a 6-byte "control block" in Short I/O space. (By definition, short addressing applies to boards which decode A01-A15. This mode of addressing is normally used for I/O boards. A16-A23 are not decoded; therefore the I/O memory is usually, but not always, mapped into the 64 Kbyte block $FF0000 to $FFFFFF.) The address of the control block is set using DIP switches on the card.

There are three registers in the control block which affect the operation of the card: the control/status register, the interrupt status/ID register and the memory address register. All control block registers are 1 byte wide.

The control/status register is a read-write register which allows the host to control and monitor the card. The interrupt status/ID register is used for interrupt initialization and processing. The memory address register is used to assign the address of the 64 Kbyte block of RAM associated with the card in the VMEbus standard access memory map.

The card supports the VMEbus Priority Interrupt Bus. The interrupt vector is software selectable using the "Interrupt status/ID" register in the control block. The interface card generates interrupt requests using an I(1) through I(7) (Stationary), ROAK interrupter. D08(O) (Stationary) status/ID transfer capability is used.

The card responds to Data D08(O) Transfer Bus (DTB) cycles in the A16 (Short) addressing mode and D16 or D08(EO) DTB cycles in the A24 (Standard) addressing mode. D08(EO) bus cycles are single-byte cycles. When 16-bit values are read from the card (i.e. buffer pointers) the values are organized in low-byte, high-byte order.

The address modifier (AM) codes can be one of standard or short supervisory data access ($3D, $2D) or standard or short non-privileged data access ($39, $29).

For diagnostic purposes the card is considered to be a "non-intelligent" card; diagnostic routines (i.e. memory test) are performed on the card by a master processor rather than by the card itself.

1.3 Manual Organization

This document is divided into several sections: part 2 describes how to install the card, part 3 describes the card software and the steps in writing an application. The appendices give some technical information on the card and explain how to obtain technical support.

1.4 Conventions

In this manual, a leading $ or 0x indicates a hexadecimal number.

VMEbus signals are shown in bold. An asterisk indicates an active-low signal, for example **SYSFAIL*.**

1.5 Reference Documents

Refer to the appropriate Allen-Bradley documents for information on Allen-Bradley hardware and cabling.

Refer to “IEEE Standard for a Versatile Backplane Bus: VMEbus”, ANSI/IEEE Std. 1014-1987 for explanations of VMEbus terminology.

1.6 Licensing

To use the remote I/O module, you must have a license from Allen-Bradley. Contact SST for details.

In addition, the card must have the remote I/O option enabled in the EEPROM on the card. The “-R” in the part number indicates a card with the remote I/O option enabled. Refer to section 2.10 for more information about card options.

2. Installation

The 5136-SD-VME-R interface card contains components that are sensitive to electrostatic discharge. Do not remove the card from its protective bag without using the following precautions:

- Before handling the card, ground yourself by touching a grounded object, such as the case of your computer.
- Never touch the backplane connectors or pins. Handle the card by its mounting bracket.
- Always store the card in its protective bag.

This chapter describes the procedures for:

- setting the switches and jumpers on the interface card
- installing the card in your computer
- downloading the firmware to the card
- getting the card to communicate on a network and verifying that it is working

It also contains information about short I/O register usage on the card.

2.1 Card Installation

Before you install the card in the VMEbus chassis, you must decide on the answers to the following questions:

- which 1 Kbyte block in the Short address space will be used for card control?
- will interrupts be used?
- should the card respond to supervisory mode commands and data accesses only?

In order to answer these questions, you must know the memory usage of the system and the capabilities of the application software.

The answers to these questions determine the settings of the switches and jumpers on the card. These must be set before you install the card in a VMEbus computer.

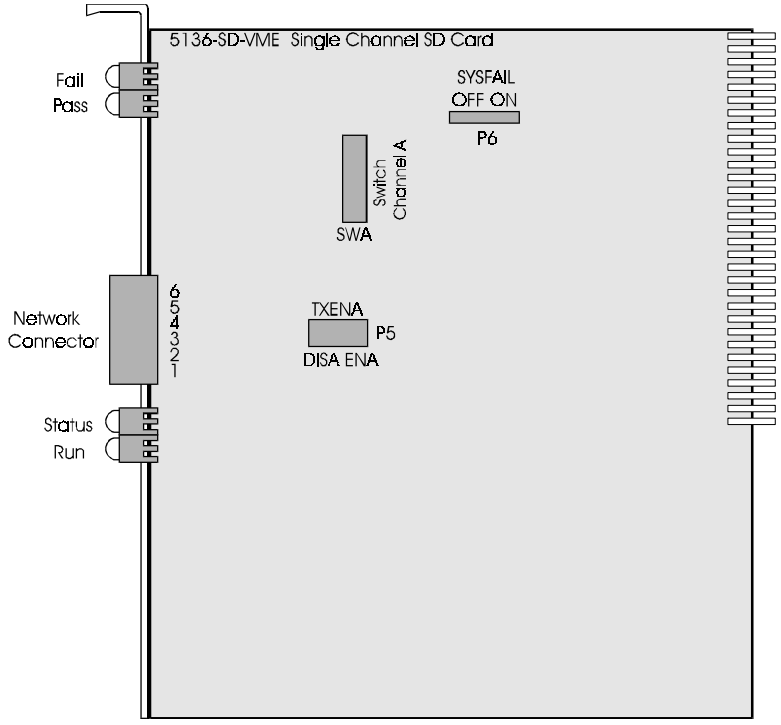
The table below shows the use for the various switch positions on the 10 position switch on the card and shows where to look for information on how to set the switch.

Positions	Used to set	Reference section
1-6	Short I/O address	2.2.1
7	Allowed address modifiers	2.2.2
8-10	Interrupt	2.2.3

In addition there is a jumper which controls transmission from the card. Refer to section 2.2.4.

The SYSFAIL* Jumper controls how the card handles the **SYSFAIL*** line. Refer to section 2.2.5.

The following diagrams show the position of the switches, jumpers and LEDs on each of the cards.


5136-SD-VME Single Channel SD Card
Fail
Pass
SYSFAIL
OFF ON
P6
Switch
Channel A
SWA
Network
Connector
6
5
4
3
2
1
TXENA
P5
DISA ENA
Status
Run

5136-SD-VME-R

2.2 Setting the Switches and Jumpers

2.2.1 Setting the Short I/O Base Address

The 1 Kbyte block of short address space occupied by the card is located on a 1Kbyte boundary at an address selected by positions 1 through 6 of the switch . These switch positions correspond to address bits A15 through A10, respectively, with a switch in the ON position matching the corresponding address signal as a 0.

Switch Position	Address bit
1	A15
2	A14
3	A13
4	A12
5	A11
6	A10

The following table shows possible addresses and the corresponding switch settings.

Address	1	2	3	4	5	6
FC00	OFF	OFF	OFF	OFF	OFF	OFF
F800	OFF	OFF	OFF	OFF	OFF	ON
F400	OFF	OFF	OFF	OFF	ON	OFF
F000	OFF	OFF	OFF	OFF	ON	ON
EC00	OFF	OFF	OFF	ON	OFF	OFF
E800	OFF	OFF	OFF	ON	OFF	ON
E400	OFF	OFF	OFF	ON	ON	OFF
E000	OFF	OFF	OFF	ON	ON	ON
DC00	OFF	OFF	ON	OFF	OFF	OFF
D800	OFF	OFF	ON	OFF	OFF	ON
D400	OFF	OFF	ON	OFF	ON	OFF
D000	OFF	OFF	ON	OFF	ON	ON
CC00	OFF	OFF	ON	ON	OFF	OFF
C800	OFF	OFF	ON	ON	OFF	ON

Address	1	2	3	4	5	6
C400	OFF	OFF	ON	ON	ON	OFF
C000	OFF	OFF	ON	ON	ON	ON
BC00	OFF	ON	OFF	OFF	OFF	OFF
B800	OFF	ON	OFF	OFF	OFF	ON
B400	OFF	ON	OFF	OFF	ON	OFF
B000	OFF	ON	OFF	OFF	ON	ON
AC00	OFF	ON	OFF	ON	OFF	OFF
A800	OFF	ON	OFF	ON	OFF	ON
A400	OFF	ON	OFF	ON	ON	OFF
A000	OFF	ON	OFF	ON	ON	ON
9C00	OFF	ON	ON	OFF	OFF	OFF
9800	OFF	ON	ON	OFF	OFF	ON
9400	OFF	ON	ON	OFF	ON	OFF
9000	OFF	ON	ON	OFF	ON	ON
8C00	OFF	ON	ON	ON	OFF	OFF
8800	OFF	ON	ON	ON	OFF	ON
8400	OFF	ON	ON	ON	ON	OFF
8000	OFF	ON	ON	ON	ON	ON
7C00	ON	OFF	OFF	OFF	OFF	OFF
7800	ON	OFF	OFF	OFF	OFF	ON
7400	ON	OFF	OFF	OFF	ON	OFF
7000	ON	OFF	OFF	OFF	ON	ON
6C00	ON	OFF	OFF	ON	OFF	OFF
6800	ON	OFF	OFF	ON	OFF	ON
6400	ON	OFF	OFF	ON	ON	OFF
6000	ON	OFF	OFF	ON	ON	ON
5C00	ON	OFF	ON	OFF	OFF	OFF
5800	ON	OFF	ON	OFF	OFF	ON
5400	ON	OFF	ON	OFF	ON	OFF
5000	ON	OFF	ON	OFF	ON	ON
4C00	ON	OFF	ON	ON	OFF	OFF
4800	ON	OFF	ON	ON	OFF	ON
4400	ON	OFF	ON	ON	ON	OFF
4000	ON	OFF	ON	ON	ON	ON
3C00	ON	ON	OFF	OFF	OFF	OFF

Address	1	2	3	4	5	6
3800	ON	ON	OFF	OFF	OFF	ON
3400	ON	ON	OFF	OFF	ON	OFF
3000	ON	ON	OFF	OFF	ON	ON
2C00	ON	ON	OFF	ON	OFF	OFF
2800	ON	ON	OFF	ON	OFF	ON
2400	ON	ON	OFF	ON	ON	OFF
2000	ON	ON	OFF	ON	ON	ON
1C00	ON	ON	ON	OFF	OFF	OFF
1800	ON	ON	ON	OFF	OFF	ON
1400	ON	ON	ON	OFF	ON	OFF
1000	ON	ON	ON	OFF	ON	ON
0C00	ON	ON	ON	ON	OFF	OFF
0800	ON	ON	ON	ON	OFF	ON
0400	ON	ON	ON	ON	ON	OFF
0000	ON	ON	ON	ON	ON	ON

2.2.2 Address Modifier Codes

The 5136-SD-VME-R provides 8-bit access to objects in its short address space, and 8- and 16-bit access to objects in its standard address space. Whether a particular bus cycle accesses short, standard, extended, or long (extended and long are not used on the 5136-SD-VME-R) address spaces, and the type of access that is made, are selected by the VME master through the use of the address modifier codes. These codes are decoded by the card and used to determine the object to be accessed.

In addition to selecting from among the four spaces available on the VMEbus, address modifier codes also select whether the master is making a supervisory or non-privileged access, and (for all but short address space accesses) whether the access is to program or data space, and whether it is to be a single-object or block access.

The 5136-SD-VME-R can respond to address modifier codes $3D, $39, $2D, and $29. This means that supervisory and non-privileged data accesses may be made to standard address space ($3D and $39), and that supervisory and non-privileged accesses may be made to short address space ($2D and $29). An access which can be positively determined to address the card, but with an address modifier code that is not supported, causes a VMEbus error.

The selection of whether only supervisory accesses or both supervisory and non-privileged accesses are permitted is made with position 7 of the card's DIP switch. The use of this switch is detailed in the following table. If a non-privileged access is made to the card when this switch is in the supervisory only position, a VMEbus error occurs.

Switch position 7	Permitted accesses
ON	Both Supervisory and Non-privileged
OFF	Supervisory only

2.2.3 Setting the Interrupt

If so enabled, the software module on the local processor can assert a VME interrupt on the switch-selected level. The level is set using positions 8, 9 and 10 of the corresponding switch block.

Switch position			Interrupt
8	9	10	
ON	ON	ON	none
OFF	ON	ON	1
ON	OFF	ON	2
OFF	OFF	ON	3
ON	ON	OFF	4
OFF	ON	OFF	5
ON	OFF	OFF	6
OFF	OFF	OFF	7

2.2.4 Transmit Enable Jumper

The jumper labelled TXENA is used to enable or disable transmission from the card. Transmission is disabled when the jumper is placed over the pins labelled DISA of the block. Transmission is enabled when the jumper is placed over the pins labelled ENA.

The card must transmit so the jumper should be in the ENA position.

2.2.5 SYSFAIL* Jumper

This jumper is labelled P6 on the 5136-SD-VME-R. When it is in the ON (right) position, assertion of the SYSFAIL* control bit in the control register causes the VMEbus **SYSFAIL*** signal to be asserted, the PASS LED to turn off, and the FAIL LED to turn on. When the jumper is in the OFF position, the LEDs reflect the state of the **SYSFAIL*** control signal but the card does not drive the VMEbus signal.

2.3 Short I/O Registers

The card occupies 1 Kbyte of the short I/O space but only three 1-byte registers are used in that 1K.

The following table gives the map of the registers contained in the short address space to which the card responds. Since the short address space of the card is essentially a D08(O) Slave, the objects in this space are not at contiguous addresses. Any attempt to access addresses other than those specified in the table (e.g. offsets 0, 2 or 4), or to access objects larger than 8 bits from the card's short address space, results in a VMEbus error.

Offset from short I/O base	Selected register
1	Control and status register
3	Interrupt vector register
5	Memory address register

2.3.1 Control and Status Register

The control and status register permits the host computer to:

- enable/disable the local processor (Z180)
- assert/deassert an interrupt to the local processor
- sense and clear an interrupt from the local processor
- control an indicator LED
- assert/deassert the VME SYSFAIL signal from the card
- enable/disable access to the card's shared RAM
- enable/disable write protection to the local processor's code space

The control and status register is located at offset 1 from the selected register base address in short address space. Its bits are all high true, and their assignments and reset states are detailed in the following table.

Control/Status Register (byte at offset 1)

BIT	FUNCTION	R/W	RESET
0	local processor reset control	R/W	1
1	interrupt to local processor	R/W	0
2	clear interrupt from local processor (reads as 0)	W	0
3	LED control	R/W	0
4	SYSFAIL control	R/W	0
5	shared RAM enable	R/W	0
6	low 32KB write inhibit	R/W	0
7	interrupt from local processor	R	0

Setting the **local processor reset control** bit to '1' holds the processor on the card in reset and prevents it from running. Since this is the case following a system reset, the processor does not run until the host computer releases it, after the code has been loaded. When this bit is a '1' and the host writes a '0' to it, a reset pulse is issued to the Z180 on the card, causing it to begin execution at its own offset 0000.

The **interrupt to local processor** bit is used by application software to get the attention of the module running on the Z180. This bit cannot be cleared by the local processor, so the host must clear it when it detects that the local processor has handled the interrupt. This interrupt is edge-sensitive. No current modules for the card use interrupts to the local processor.

A status ID is placed on the bus during the interrupt acknowledge cycle, as specified in the Interrupt ID register. The local processor generates an interrupt by setting the interrupt from local processor bit, which causes the selected VME interrupt signal to be asserted. This signal is released when the interrupt acknowledge cycle takes place, but the interrupt from local processor bit that caused that interrupt remains set until cleared explicitly by an interrupt service routine. The routine must do this by writing a '1' to the **clear interrupt from local processor** bit, which clears and re-enables the hardware for further interrupts.

The **LED control** bit turns on (1) and off (0) the RUN LED for the card. (The STATUS LED is controlled by the local processor directly and is used by the software as an indicator of its current state.)

Setting the SYSFAIL control bit to '1' asserts the **SYSFAIL*** signal on the VMEbus if the SYSFAIL jumper is in the "ON" position. This bit also controls the state of the PASS and FAIL LEDs on the card.

Following reset, the **shared RAM enable** bit assures that the card will not drive the VMEbus for any standard address cycles. This bit must not be set to '1', enabling access to the shared RAM, until the Memory Address register has been set to the required base in standard address space.

Since all of the Z180's memory is shared with the VME host, it may be desirable to prevent the host from writing to the lower 32 Kbytes of Z180 memory. SST software uses the lower 32 Kbytes for Z180 code and unintentional writes to this area by the host could cause the Z180 software module to crash. The **low 32KB write inhibit** bit, when '1', prevents host-initiated write cycles from altering shared RAM in the range $0000 to $7FFF (offset from the selected standard access base). Such cycles do not cause VMEbus errors and appear to complete normally. Nevertheless, as with all shared RAM cycles, they should be minimized as they rob cycles from the local processor.

Whenever the card software generates an interrupt, it sets the **interrupt from local processor** bit.

2.3.2 Interrupt Status/ID Register

When an interrupt request is made from the card on one of the VME interrupt lines, an interrupt handler acquires the bus and executes an interrupt acknowledge cycle. The card recognizes the acknowledgment and places an 8-bit interrupt ID on the odd half of the data bus. The value for this ID is taken from the Interrupt ID register, located at offset 3 from the selected register base address in short address space.

This register is zeroed by system reset and may be read or written at any time. If interrupts are used from the card, this register should be initialized before the local processor is allowed to run.

2.3.3 Memory Address Register

Following system reset, the shared RAM enable bit in the Control/Status register is reset, preventing the card from driving the bus during any standard address cycles. This is done in order that the location in standard address space occupied by the shared RAM may be selected. Standard address space accesses use 24 bits of address (16MB) while the card uses only 16 bits of address (64 Kbytes) for generating addresses into the shared RAM. For standard address bus cycles, the upper 8 bits from the VMEbus are compared to the bits in the 8-bit Memory address register, located at offset 5 from the selected register base address in short address space. If these match exactly, and the shared RAM enable bit is set, then a shared RAM access is performed.

For example, if it is required that the shared RAM occupy the 64 Kbyte block between $0D0000 and $0DFFFF in standard address space, then the Memory address register should be set to $0D.

The memory address register is zeroed by system reset, and may be read or written at any time. It should be initialized with the upper 8 bits of the 24-bit address at which the shared RAM area is to begin.

2.4 Standard Address Space

The card contains 64 Kbytes of static RAM that is used as the code and data memory for the local processor (Z180) and the tables and registers for all network activity in which the card is involved. This RAM is accessible at any time to a master on the VMEbus, regardless of whether or not the local processor is running. As necessary, the local processor pauses while the VMEbus master accesses the memory.

This memory is located at offsets $0000 through $FFFF from the 24-bit base address, whose upper 8 bits are determined from the Memory Address register contents. A VME master may access the memory either 8 or 16 bits at a time. If a 16-bit object is accessed, it must be aligned on a 16-bit boundary. That is, an access to a 16-bit object, 8 bits of which are in the odd byte of one address, and whose other 8 bits are in the even byte of the next address, is not possible. If this is attempted by host software, the hardware on the master will probably take care of the details, performing two single-byte accesses invisibly to the software.

Byte addresses on the VMEbus access locations that correspond exactly to addresses on the local processor. Also, 16-bit accesses on the VMEbus access the expected bytes from the local processor's space. For example, if $1234 is written by a VMEbus master to offset 0 in the shared RAM, the local processor sees $12 at address 0, and $34 at address 1. Since the processor on the card puts low bytes at low addresses and VMEbus masters put low bytes at high addresses, care must be taken when accessing 16-bit objects in the shared RAM to ensure that the byte ordering is what the software (and local processor) expect.

2.5 Diagnostic LEDs

The PASS and FAIL LEDs provide information on the overall operation of the card. They follow the state of the SYSFAIL line when the SYSFAIL jumper is in the ON position. See section 2.2.5.

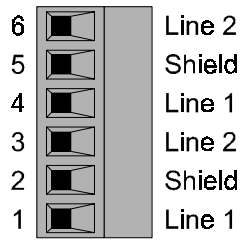
In addition, there are two diagnostic LEDs on the card.

The red RUN LED is controlled by application software using the card's control register. This allows a host processor to perform diagnostics on the card and manipulate the state of the LED to give an indication of the results.

The green STATUS LED is under control of the microprocessor on the card. It is on whenever racks have been enabled in the card.

2.6 Connecting to the Communication Network

The card has been designed to provide direct connection to Allen-Bradley programmable controller communication networks. Allen-Bradley specifies Belden 9463 twinaxial cable ("Blue Hose") for their network installations. The Blue Hose is wired into the card using the Phoenix Combicon connector supplied with the card. See page 57 for the part number of the connector.



For wiring purposes pin 1 of the Phoenix connector is identified on the faceplate of the card and is the bottom pin. Pins 1, 2 and 3 are connected internally to pins 4, 5 and 6, respectively.

Connect Pin 1 to the Line 1 of the remote I/O network. This is usually, but not always, the blue wire. Connect Pin 2 to the shield. Connect Pin 3 to Line 2. This is usually, but not always, the clear wire.

Termination

The nodes at the two physical ends of the network should have terminating resistors. All other nodes should not. Every network should have exactly two terminators.

The card does not have an onboard terminator. If you require a terminator, it consists of a resistor between the blue and clear wires. Allen-Bradley recommends a 150 ohm resistor at 57.6 and 115.2 kbaud and an 82 ohm resistor at 230.4 kbaud.

2.7 Loading a Program on the Card

An application must take the following steps to initialize the interface card.

1. Make sure the processor is reset by writing to the control/status register. Select the card address by writing to the address register.
2. Enable card memory by setting the enable bit in the control register.
3. If interrupts are to be used, initialize the interrupt status/ID register.
4. Perform memory test etc. as required by the application.
5. Set or clear the RUN LED according to the result of the memory test.
6. Load the software module SDDEM.BIN onto the card at the standard memory address. That is, read the file SDDEM.BIN and write it the memory on the card, starting at the base address of the card in standard memory space. Verify that the module has been correctly loaded.
7. Write $A5 into offset $8000 from the start of memory.
8. Set the run bit in the control register.
9. Look at the byte at offset $8000. Wait up to 7 seconds for it to change. If it doesn't change, the module did not run on the card. If it becomes 00, everything has run correctly. If it becomes 01, there is a null-terminated string starting at offset $8001 which describes the error. If it has any other value, the result is invalid and the processor did not run correctly.
10. Run your application.

The distribution disk contains a sample loader for a Xycom XVME computer.

2.8 VME Programming Notes (READ THIS!!!!!)

The following information should always be kept in mind when reading the module descriptions.

The tables on the card are all memory mapped. Within the 64K of card memory, the lower 32K is reserved for the card software; the upper 32K is used by the tables by which the host application communicates with the card.

The card processor organizes data in low-byte/high-byte format. Your application must take this into consideration. For example, messages are placed into buffers at locations given by the card software. When your application reads the card to determine the location of the next free buffer and the bytes read are sequentially $00 and $0A, the next free buffer is at offset $0A00 from the start of the data area (rather than $000A as you might expect in a VMEbus environment.)

The tables on the card contain some elements that are one byte wide. Some compilers by default align structure elements on word boundaries; you must tell your compiler to use byte alignment for structure elements that refer to data on the card.

2.9 Software Modules

The following modules are shipped with the 5136-SD-VME-R.

Module	Purpose
sddopt.bin	module to display card options
sddem.bin	module to emulate remote I/O.

2.10 Card Options

The 5136-SD-VME-R contains an EEPROM that is programmed before the card is shipped to select which software modules the card can run. Possible options include:

DH	Data Highway
DHP	Data Highway Plus
MON	Data Highway Plus monitoring (for DH+ Network Analyzer)
NET	network monitor (for DH+ Network Analyzer)
KTEMU	KT emulation
EXEC	the exec module
RIO	for emulating remote I/O. For use on the 5136-SD-VME-R card. Requires a licence from Allen-Bradley. Contact SST for details.
RMAS	for scanning remote I/O. For use on the 5136-SD-VME-R card

The 5136-SD-VME-R is shipped with just the RIO option enabled.

To determine what options are enabled on your card, load the card module SDDOPT.BIN.

The module simply reads the options enabled and displays them at offset $8001, then sets $8000 to 01 (same as an error).

If you try to load a module for which the option is not enabled, the module puts the message "*** Fatal Error ; Option Not Enabled on this Card ***" at location $8001, $8000 gets set to 01 (same as an error) and the module does not run.

2.11 Troubleshooting Installation

This section describes what to do if the card cannot communicate on a network. It also provides more detailed information on some common sources of problems.

- Check cabling for correct wiring to the card.
- Check for shorted wires, leads on terminating resistors shorted to cables, strands from the shield shorting to the other wires.
- Check baud rate
- Check network termination. Only the two nodes at the physical ends of the network should have terminating resistors.

3. The Remote I/O Card Module

3.1 Overview

The remote I/O network is a master-slave network with the programmable controller acting as the master. During normal operation, the programmable controller sends messages to the remote I/O racks, to which the remote racks send replies. Only the master initiates communication. There can be only one master on the network.

This constant exchange of information allows the programmable controller to maintain an up-to-date image of the status of inputs and to manipulate outputs accordingly. During a program scan, the programmable controller uses the logic defined in the ladder program to update an "output image" table. During an I/O scan, the programmable controller scanner updates the physical outputs from this table and reads the state of inputs into an "input image" table. In some programmable controllers, the logic and I/O scans are synchronized; in most cases they are asynchronous.

The DEM software monitors this data exchange and maintains its own copy of input and output image tables. These tables are accessible to application programs through the interface card's shared memory interface.

The DEM software can also send replies to the PLC as if it were one or more racks of remote I/O. These emulated racks are referred to as virtual racks. The programmable controller acts as if there is a physical rack of I/O on the network. An application can use these virtual racks to simulate I/O or by the logic in the programmable controller (i.e., a rung of ladder logic could use the status of virtual inputs). With appropriate ladder logic in the programmable controller, an application can also use a virtual rack to indirectly monitor the status of local I/O or the state of storage bits defined by the ladder logic.

The DEM software is capable of monitoring and emulating any combination of real and virtual racks up to a total of 32 I/O racks. This is subject to the limits imposed by the type of programmable controller used. For example, the PLC-5/20 scans a maximum of three remote racks.

The DEM module supports partial racks as well as full racks. (See section 3.4.3) It can also generate host computer interrupts on the reception of messages for any or all racks.

DEM is capable of monitoring block transfers on real racks and of responding to block transfer requests from the programmable controller. This includes both block transfer reads and writes.

Code fragments in this manual and sample programs on the distribution disk are written in C.

Hexadecimal numbers are identified by a preceding 0x, as in C.

3.2 Programming Overview

An application program interfaces with the DEM software by reading and writing to memory locations and tables that are defined by the DEM software.

All addresses in this manual are offsets from the start of the data tables on the card. See section 2.8.

Section 3.12 on page 52 summarizes the layout of tables in memory.

Section 3.11 describes the steps involved in configuring and running the DEM software.

The following sections describe the various tables and flags in detail.

3.3 Variables and Flags

Several flags are used to control DEM and to provide information to the host application.

3.3.1 BAUD RATE Flags

Communication over the Allen-Bradley remote I/O network can occur at a baud rate of 57.6K baud, 115.2K baud or 230.4K baud. You must set the DEM baud rate to match the baud rate for the network to which you are attached or you will not be able to receive or transmit any data on the network.

There are two flags in DEM for setting the baud rate: the BAUD_230K flag at offset 0x800 from the base address of card memory and the BAUD_RATE flag at offset 0x801 from the base address. These flags are used in combination to set the baud rate at which DEM communicates.

Baud	BAUD_230K	BAUD_RATE
57.6K	0	0
115.2K	0	0xff
230.4K	0xff	0xff

The DEM software defaults to a baud rate of 115.2K baud when you load the DEM module.

Changes to the baud rate do not take effect until you set the CONFIG flag to reconfigure DEM.

3.3.2 CONFIG Flag

Offset 0x803 contains the CONFIG flag. An application program must set this flag to 0xff to signal DEM to perform a reconfiguration. You must do this if you change BAUD_RATE or ENA_DBL_BUF. When the DEM software recognizes the configuration request, it sets the baud rate and clears all data tables. The DEM software then clears the CONFIG flag to inform the application program that the reconfiguration is complete and communication has started.

3.3.3 PLC_TYPE Byte

Offset 0x804 is a byte value written by the DEM module to tell you the type of programmable controller on the network. DEM uses the messages from the programmable controller to determine what type of programmable controller is on the line. If the card is not receiving messages from a programmable controller, this value is not valid. A value of 0xff indicates that a PLC-2 is connected. A value of 0 indicates that a PLC-3 or a PLC-5 is connected (the messages from the PLC-3 and PLC-5 are identical so DEM cannot distinguish between them).

3.3.4 CHK Bytes

The host can use the three CHK bytes starting at offset 0x80b to determine that the DEM module is present and running. They contain 0xc3, 0x04 and 0x00 respectively. DEM writes these bytes as the last step in its startup initialization after it is loaded.

3.3.5 BT_BASE

This word location, at offset 0x820, contains the beginning address of the area which DEM is to use to store data for any real block transfers it sees on the remote I/O network. If you are using block transfers on virtual racks, you must write an appropriate starting value in this location; otherwise you can ignore it and let DEM initialize it. It is used in conjunction with the BT_CONFIG flag. Refer to section 3.7.3 for more information on using block transfers on virtual racks.

3.3.6 BT_CONFIG

Offset 0x822 is a byte value used by the host application to install block transfers on virtual racks. If you are using virtual block transfers, before setting the CONFIG flag, reset the BT_CONFIG byte to 0. Then set the CONFIG flag and wait until DEM clears CONFIG. At this point, install any block transfers and update the value in BT_BASE. Finally, set BT_CONFIG, which tells DEM to finish the configuration and to go online.

Refer to section 3.7.3 for more information on using block transfers on virtual racks.

3.3.7 COMM Flag

An application program can monitor the status of communication on the remote I/O network using the COMM flag, at offset 0x824. The DEM software sets this flag to 0xff during normal communications and resets the flag to 0 if it does not receive a valid message from the programmable controller within a period of 160 milliseconds.

3.3.8 ALL_VIRT_GOOD

This flag, at offset 0x825, is non-zero if no virtual racks are in error. If any virtual rack is in error, this flag is zero.

3.4 Enabling and Monitoring Racks

3.4.1 Rack Status

The RACK_STATUS table occupies offsets 0x1c00 to 0x1c1f. The RACK_STATUS table shows the status of all real or virtual racks on the network. There is one byte in the table for each rack. Possible states for the status bytes are:

Value	Meaning
0	no rack present.
1	programmable controller in run mode, rack is OK.
2	programmable controller in test or program mode.
3	rack error, programmable controller sees no response from this rack

The following formula gives the correspondence between rack numbers and the location in the table:

RACK_STATUS = 0x1c00 + RACK - PLCTYPE

where RACK is the rack number and PLCTYPE is defined as 1 for the PLC-2 and 0 for the PLC-3 and PLC-5.

Example:

To determine the status of rack 5 in a PLC-2 system, read the byte found at the address calculated from:

RACK_STATUS = 0x1c00 + RACK - PLCTYPE

RACK_STATUS = 0x1c00 + 5 - 1

RACK_STATUS = 0x1c04

Example:

In a PLC-3 system, the address containing the status of rack 12 (octal) is:

RACK_STATUS = 0x1c00 + RACK - PLCTYPE

RACK_STATUS = 0x1c00 + 0x0a - 0

RACK_STATUS = 0x1c0a

Note that rack addresses are in octal. 12 octal is the same as 10 decimal or 0x0A hexadecimal.

If your application uses virtual racks, it must enable them in DEM. You must also tell the programmable controller that you are using these racks or no communication can take place.

WARNING:

It is essential that none of the racks enabled in the RACK_ENABLE table duplicate any real racks on the I/O network. If a virtual rack duplicates a real rack, both real racks and virtual racks attempt to reply to the programmable controller. This causes numerous communication errors and rack faults on the remote I/O network. This could manifest itself as improper operation of real I/O devices and could be hazardous!

Also note that there are limits to the number of racks a specific PLC can communicate with. If there are already a number of real racks in the system, this reduces the number of possible virtual racks.

3.4.2 RACK_ENABLE Table

When an application uses virtual racks, the DEM software must know which racks it is to emulate. This is the purpose of the RACK_ENABLE table. Each byte in the table corresponds to a rack number. When you write 0xff to a corresponding rack location in the RACK_ENABLE table, the DEM software replies as a full rack when the programmable controller sends a message to the given rack. The RACK_ENABLE table occupies memory between 0x2c00 and 0x2c1f on the card. Note that placing a value of 0xff in the RACK_ENABLE table enables a full rack, starting at the first quarter. See section 3.4.3 for information on partial racks.

The following formula gives the address of a rack in the RACK_ENABLE table:

```
ACT_RACK_ADDRESS = 0x2c00 + RACK - PLCTYPE
```

In this formula, RACK is the rack number to be emulated and PLCTYPE is defined as 1 for the PLC-2 and 0 for the PLC-3 and PLC-5.

Example:

To activate rack 2 as a full rack in a PLC-2 system, write 0xff to the address given by:

ACT_RACK_ADDRESS = 0x2c00 + RACK - PLCTYPE

= 0x2c00 + 2 - 1

= 0x2c01

Example:

To activate rack 3 as a full rack in a PLC-5 system, write 0xff to the address given by:

ACT_RACK_ADDRESS = 0x2c00 + RACK - PLCTYPE

= 0x2c00 + 3 - 0

= 0x2c03

3.4.3 Partial Racks

The DEM module can monitor partial racks or emulate partial virtual racks. This is accomplished by breaking the bytes in the various tables (RACK_ENABLE, RACK_STATUS...) into four parts (two bits per quarter rack). The following table summarizes the correlation between bit positions and rack quarters:

7	6	5	4	3	2	1	0
Fourth quarter		Third quarter		Second quarter		First quarter	

There are two bits for each quarter rack. All tables except BTR_LEN, BTR_LOC, BTW_LEN and BTW_LOC are divided this way.

Each quarter rack corresponds to two I/O groups. The following table shows the correlation between quarter racks and I/O groups:

Quarter	I/O Groups
First	0,1
Second	2,3
Third	4,5
Fourth	6,7

To enable a particular quarter rack, place 01 in the corresponding bit position for the starting quarter to be enabled in the RACK_ENABLE table. The status for that rack/quarter may then be read from the corresponding bit positions in the RACK_STATUS table.

The DEM module must know the length of the partial rack that it is to emulate. This is accomplished through the use of another table called the RACK_END table. This table is located at offsets 0xc00-0xc1f. There are two bits available to define the end quarter for a particular rack/quarter. The following table shows the correlation between the two-bit value and the end quarter defined:

Bit Value	Rack End
00	End of first quarter
01	End of second quarter
10	End of third quarter
11	End of fourth quarter

When you write 0xff into the RACK_ENABLE table, the DEM module automatically sets the RACK_END value for the first quarter to 11 binary, enabling a full rack. This allows full racks to be enabled easily and maintains compatibility with existing "full rack" applications. The host must fill in the RACK_END values before writing any value other than 0xff to the RACK_ENABLE table.

The DEM module fills in the RACK_END table for all real racks. This allows an application to know the size of existing racks, which indicates where it can place partial virtual racks.

Example 1:

Enable a virtual three-quarter rack starting at the second quarter in rack 3 of a PLC-3 or PLC-5 system that already has a real quarter rack starting at the first quarter, as in the following table:

Fourth quarter	Third quarter	Second quarter	First quarter
Virtual rack			Real rack

The DEM module automatically detects the real quarter rack that is already attached and updates the tables appropriately. The host should check the RACK_END table to ensure that the real rack is a quarter rack. (Bits 0 and 1 should be 00.)

Since the required virtual rack starts at the second quarter, bits 2 and 3 in the various tables are used.

To enable the virtual three-quarter rack, follow these steps:

1. Put the end quarter value into the RACK_END table at 0x0c03. The virtual rack ends at the end of the fourth quarter. The two bit end quarter value is then 11. Write this in the RACK_END table bits 2 and 3 without changing the other bits. The following line of C code demonstrates this.

```
RACK_END[3] = (RACK_END[3] & ~(0x3 << 2) ) | (0x03 << 2);
```

2. To enable the rack, place 01 in bits 2 and 3 of the RACK_ENABLE table. Even though these are the only bits used, and the other bits are all 0, it is generally better to affect only the bits used. The following line of C code demonstrates this:

```
RACK_ENABLE[3] = (RACK_ENABLE[3] & ~(0x3 << 2) ) | (0x1 << 2);
```

The status for the real rack and the virtual rack is found in the RACK_STATUS table at address 0x1c03. Bits 0 and 1 contain the status for the real rack (since it starts at the first quarter) and bits 2 and 3 contain the status for the virtual rack (since it starts at the second quarter). The following lines of C code demonstrate how to read the rack status.

```
real_rack_status    = (RACK_STATUS[3] >> 0) & 0x3;
virtual_rack_status = (RACK_STATUS[3] >> 2) & 0x3;
```

Example 2:

To enable a real half rack as the second and third quarter of rack 1, a virtual quarter rack starting at the first quarter, and a virtual quarter starting at the fourth quarter as in the following table:

Fourth quarter	Third quarter	Second quarter	First quarter
Virtual rack	Real rack		Virtual rack

It is assumed that the real half rack which is configured as rack 1 starting at module group 2 is already connected to the remote I/O network.

Since the required virtual racks start at the first and fourth quarter, bits 0,1 and 6,7 in the various tables are used. To enable the two virtual quarter racks the following steps should be taken:

1. Put the end quarter values into the RACK_END table at 0x0c01. The virtual racks end at the end of the first quarter and the fourth quarter respectively. The two-bit end quarter value for the first quarter rack is then 00, and the two-bit end quarter value for the quarter rack is 11. Write this in the RACK_END table without changing the other bits. The following lines of C code demonstrate this.

```
RACK_END[1] = (RACK_END[1] & ~(0x3 << 0) ) | (0x0 << 0);

RACK_END[1] = (RACK_END[1] & ~(0x3 << 6) ) | (0x3 << 6);
```

2. To enable the racks, place 01 in bits 0,1 and 6,7 of the RACK_ENABLE table. Even though these are the only bits used, and the other bits are 0, it is generally better to affect only the bits used. The following lines of C code demonstrate this:

```
RACK_ENABLE[1] = (RACK_ENABLE[1] & ~(0x3 << 0) ) | (0x1 << 0);

RACK_ENABLE[1] = (RACK_ENABLE[1] & ~(0x3 << 6) ) | (0x1 << 6);
```

The status for the real rack and the virtual racks is found in the RACK_STATUS table at 0x1c01. Bits 0 and 1 contain the status for the first virtual rack (since it starts at the first quarter), bits 2 and 3 contain the status for the real rack (since it starts at the second quarter) and bits 6 and 7 contain the status for the second virtual rack (since it starts at the fourth quarter). The following lines of C code demonstrate how to read the rack status.

```
virtual_rack_status_1 = (RACK_STATUS[1] >> 0) & 0x3;

real_rack_status  = (RACK_STATUS[1] >> 2) & 0x3;

virtual_rack_status_2 = (RACK_STATUS[1] >> 6) & 0x3;
```

3.4.4 Rack Options

The RACK_OPTION table, 0x2f00-0x2f1f, sets various options for each rack. There are two possible option bits for each rack or partial rack.

The lower bit emulates the ‘last state’ switch on an I/O chassis. If this bit is 1, DEM clears the outputs to 0 when communication with the PLC is lost. If the bit is 1, outputs remain in their last state. The default is 0.

The higher bit determines what happens with the outputs when the PLC is in program mode. If the bit is 0, DEM clears outputs to 0 when the PLC is in program mode. If the bit is 1, outputs remain in their last state. The default is 0.

3.5 Discrete Outputs

The OUTPUT table contains the values of all outputs. This table is located from offset 0x0000 through 0x01ff. Every I/O group (16 points) requires two bytes in the table. The low-order byte contains the data for output points 00_8 to 07_8 of the I/O group (slot 0), the high-order byte contains data for output points 10_8 to 17_8 of the I/O group (slot 1). The following formula gives the address of the output data for a particular I/O group and slot:

OUT_ADDRESS = 0x000 + (RACK - PLCTYPE) * 16 + IOGROUP * 2 + SLOT

Example:

In a PLC-5 system determine the value of output O:12/14.

This address corresponds to the output point in rack 1 group 2, slot 1, bit 4. Using the formula:

OUT_ADDRESS = 0x000 + (RACK - PLCTYPE) * 16 + IOGROUP * 2 + SLOT

OUT_ADDRESS = 0x000 + (1 - 0) * 16 + 2 * 2 + 1

OUT_ADDRESS = 0x000 + 16 + 4 + 1

OUT_ADDRESS = 0x015

I/O group 012 slot 1 corresponds to address 0x015 and output point 14 is located in bit 4 of 0x15 (bit 4 of slot 1).

3.6 Discrete Inputs

The INPUT table, from offset 0x400 through 0x5ff, contains the values of discrete inputs. This table reflects the values of inputs in real racks. Application programs can also fill in this table to update the inputs of virtual racks. To monitor the value of an input point, the application reads from the table at the address that corresponds to that point. To change the value of virtual inputs, the application sets or clears the bits that correspond to the desired input points. Every I/O group (16 points) requires two bytes in the table. The low-order byte contains the data for output points 00_8 to 07_8 of the I/O group (slot 0), the high-order byte contains data for output points 10_8 to 17_8 of the I/O group (slot 1). The following formula gives the location of the input data for a particular I/O group:

IN_ADDRESS = 0x400 + (RACK - PLCTYPE) * 16 + IOGROUP * 2 + SLOT

Example:

In a PLC-2 system determine the value of input 111/07.

This address corresponds to the input point in rack 1, group 1, slot 0, bit 7. Using the formula:

IN_ADDRESS = 0x400 + (RACK - PLCTYPE) * 16 + IOGROUP * 2 + SLOT

IN_ADDRESS = 0x400 + (1 - 1) * 16 + 1 * 2 + 0

IN_ADDRESS = 0x400 + 0 + 2 + 0

IN_ADDRESS = 0x402

I/O group 110 corresponds to address 0x402. Input point 07 is located in bit 7 of 0x402 (bit 7 of slot 0).

Writing inputs to racks that are not virtual racks has no effect on the programmable controller's input data tables. On the next scan, DEM overwrites the inputs on the card by data from messages from the real rack to the programmable controller.

3.7 Block Transfers

DEM monitors block transfers between real racks and the programmable controller and lets you exchange data with the programmable controller using block transfers on virtual racks. DEM maintains tables that tell where block transfers are located and what the current length is, as well as a table of pointers to the actual block transfer data for each block transfer. DEM has a large pool of free memory from which it allocates space for the data from any block transfer it sees on the line. DEM can monitor up to 160 block transfers. (If you use double buffering, the number is reduced to 64. See section 3.9.6.)

3.7.1 Block Transfer Reads

The DEM software can monitor block transfers on the remote I/O network. The DEM software automatically stores the block transfer data to a buffer on the card. An application gains access to this buffer by reading a pointer to the buffer and the length of the data from tables that are organized by rack, group, and slot. Up to 160 block transfers can be buffered by the DEM module.

To access block Transfer Read information, you use the BTR_LOC and BTR_LEN tables. The BTR_LOC table contains pointers into the block transfer buffer. The BTR_LEN table contains the length of the data (number of words) in the buffer. These tables are located at 0x1000 to 0x13ff and 0x1800 to 0x19ff respectively.

The following formula gives the location in the BTR_LOC table for a particular block transfer module:

BTR_LOC = 0x1000 + (RACK - PLCTYPE) * 32 + (IOGROUP * 4) + (SLOT * 2)

and the following formula gives the location in the length table:

BTR_LEN = 0x1800 + (RACK - PLCTYPE) * 16 + (IOGROUP * 2) + SLOT

Note:

The DEM software stores the pointers in Least Significant Byte-Most Significant Byte (LSB-MSB) format.

Example:

For an analog input module located in rack 3, I/O group 2, slot 1 of a PLC-2 system, the pointer to the block transfer read data is stored at:

BTR_LOC = 0x1000 + (RACK - PLCTYPE) * 32 + (IOGROUP * 4) + (SLOT * 2)

BTR_LOC = 0x1000 + (3 - 1) * 32 + (2 * 4) + (1 * 2)

BTR_LOC = 0x1000 + 64 + 8 + 2

BTR_LOC = 0x1000 + 74

BTR_LOC = 0x1000 + 0x4a

BTR_LOC = 0x104a

The current length of the block transfer read is stored at:

BTR_LEN = 0x1800 + (RACK - PLCTYPE) * 16 + (IOGROUP * 2) + SLOT

BTR_LEN = 0x1800 + (3 - 1) * 16 + (2 * 2) + 1

BTR_LEN = 0x1800 + 32 + 4 + 1

BTR_LEN = 0x1800 + 37

BTR_LEN = 0x1800 + 0x25

BTR_LEN = 0x1825

If 0x104a and 0x104b contain 0x80 and 0x30 respectively and address 0x1825 contains 0x06, the block transfer information is found in the 12 bytes (6 words) beginning at address 0x3080.

The DEM module also maintains a table called the BTR_UPDATE beginning at location 0x1400. The correlation between rack and I/O group is the same as in the BTR_LEN table. Each time a BTR updates, the DEM module writes 0xff into the corresponding location in this table. The host polls this location to determine if a given BTR has been updated. The host then clears the location. The DEM module sets it to 0xff when the BTR is next updated.

3.7.2 Block Transfer Writes

The mechanism for monitoring block transfer writes is similar to the one used for block transfer reads. Only the location of the BTW_LOC and BTW_LEN tables is different. These tables start at address 0x2000 for BTW_LOC and address 0x2800 for BTW_LEN. Substitution of 0x2000 for 0x1000 in the BTR_LOC formula and 0x2800 for 0x1800 in the BTR_LEN formula gives the BTW_LOC formula and the BTW_LEN formula:

BTW_LOC = 0x2000 + (RACK - PLCTYPE) * 32 + (IOGROUP * 4) + (SLOT * 2)

BTW_LEN = 0x2800 + (RACK - PLCTYPE) * 16 + (IOGROUP * 2) + SLOT

The DEM module maintains a table called the BTW_UPDATE at location 0x2400. The correlation between rack and I/O group is the same as in the BTW_LEN table. Each time a BTW updates, the DEM module writes a 0xff into the corresponding location in this table. The host polls this location to determine that the given BTW has been updated. The host then clears the given location. The DEM module sets it to 0xff when the BTW is next updated.

3.7.3 Block Transfers on Virtual Racks

The interface card can respond automatically to block transfer requests from the programmable controller on virtual racks. These are referred to as active block transfers. All such block transfers must first be installed on the card.

There's a specific procedure for installing the block transfer data on the card but first we'll describe what data has to be written.

For each emulated block transfer in your programmable controller program, you need to know the length (in words), the location (rack, I/O group and slot) and the block transfer type (read or write).

For each active block transfer, you have to write into the card data tables the following information:

- **the length in words**. The address where the length is written is calculated as described in sections 3.7.1 and 3.7.2. Separate tables are used for block transfer reads and writes.

For example, if you have a block transfer read at rack 2, I/O group 3, slot 0 with length 7 words, you would write a value of 7 to the location

0x1800 + 2 * 0x10 + 2 * 3 + 0 = 0x1826

A second example is a block transfer write at rack 2, I/O group 2, high slot, length 22 words. Write 0x16 (22 decimal) to location

0x2800 + 2 * 0x10 + 2 * 2 +1 = 0x2825

- **the address where the block transfer data is stored**. DEM uses a pool of free memory from addresses 0x3000 to 0x7FFF to store block transfer data for both read and write block transfers. Tables based on the rack, I/O group and slot indicate where the actual data is to be found. You must allocate 2 bytes in the free memory pool for each word of data in your block transfers. To conserve memory, use just as much memory as you need. If you do not know the length of the block transfer, reserve the maximum possible space - 64 words or 128 bytes. The calculations of the location where you write the pointer to the actual data are as described in section 3.7.1 and 3.7.2 except that in this case you are creating the addresses rather than having them automatically created by the card software.

Example:

Let's say you have two block transfers on rack 2, the first a block transfer read at I/O group 3, low slot, length 7 words and the second a block transfer write at I/O group 2, high slot, of length 22 words. To install the block transfer read, you'll need 14 (0x0E) bytes to store the data. Since this is the first block transfer you are installing, memory locations 0x3000 to 0x300D are used. The pointer to this data (0x3000) is stored in the table BTR_LOC at

0x1000 + 2 * 0x20 + 3 * 0x4 + 0 = 0x104C

in low byte, high byte order. In other words, you'd write 0x00 to 0x104C and 0x30 to 0x104D.

The next available location in the free memory pool is 0x300E and that's where you store the data for the block transfer write. It's 22 words long so you need 44 (0x2C) bytes, starting at 0x300E and ending at 0x300E + 0x2C - 1 = 0x3039. The pointer is found in the BTW_LOC table at location

0x2000 + 2 * 0x20 + 4 * 2 + 2 = 0x204A

so you write 0x0E at 0x204A and 0x30 at 0x204B

(Of course, you'd also have to write the lengths of these block transfers in the BTR_LEN and the BTW_LEN tables respectively.)

If you have other block transfers to install, continue until they have all been installed. For each block transfer, write the length and the pointer to the data in the appropriate table and location.

The final step in the installation procedure is to write the next location to be used in the free memory area of memory (again in low-byte, high-byte format) to location 0x820. In the example, the next location is 0x303A, so write 0x3A to 0x820 and 0x30 to 0x821.

DEM uses this value to begin allocating memory for any block transfers on real racks. You must write to this location to ensure that real block transfer data does not overwrite the data for your active block transfers on the virtual racks.

3.7.4 Installing Block Transfers on Virtual Racks

The procedure for the installation is:

- write 0x00 to BT_CONFIG (offset 0x822)
- write 0xff to CONFIG (offset 0x803)
- wait until DEM clears CONFIG. DEM is now waiting while you write any necessary data to the tables on the card.

Write the :

- baud byte
- active rack table
- for each emulated block transfer:
    - block transfer length data
    - block transfer data location
- the base address for future allocation of block transfer data, BT_BASE, at offset 0x820.

When you're done, write 0xff to BT_CONFIG to tell DEM you've finished writing data. DEM then completes the configuration.

DEM now responds to each active block transfer as the programmable controller requests it. All you have to do is write the data to the card memory area for each block transfer read and read the card memory for each block transfer write on the virtual racks. (Of course, you must program the block transfer in the programmable controller and make sure the block transfer is enabled for the block transfer to occur.)

3.8 Diagnostic Counters

DEM maintains various diagnostic counters that show the state of communication and also its internal operation. These counters occupy offsets 0x8c0 to 0x8d0.

Counter	Offset	Description
good_plc_packs	8c0	count of good packets from the PLC
good_rck_packs	8c1	count of good packets from other racks on the network
bad_packs	8c2	count of packets with bad CRCs
lerror	8c3	status of last bad packet
buf_ovrflow	8c4	receive buffer overflow
pkt_too_shrt	8c5	received packet too short
rx_ovrun	8c6	receive processing overrun
bad_inp_rkmg	8c7	rack/module group mismatch
rck_too_hi	8c8	rack number is too high (0-37 octal)
len_err_ctr	8c9	rack length error
bt_rx_err_ctr	8ca	block transfer receive error counter
bt_tx_err_ctr	8cb	block transfer transmit error counter
btw_len_err_ctr	8cc	bad btw length error counter
bt_time_out	8cd	block transfer timeout counter
bt_srch_err	8ce	bt search value was out of range
bad_int	8cf	internal error
int_err	8d0	internal error

You can clear these counters by writing a 0 to them at any time.

3.9 Double Buffering

Double buffering allows an application to insure that the values of any groups of data (such as inputs for a rack, a block transfer etc.) are kept together and all changes are read from or sent to the programmable controller at the same time. For example, if the card is updating the values from a BTW at the same time the host is reading the values from the card, the host could read values that come partially from the last scan and partially from the present scan. With double buffering, two tables are used for each type of quantity (discrete inputs, discrete outputs, BTWs and BTRs). Double buffering can be enabled for each of these types of quantities independently. All values that are supplied by the DEM module (discrete outputs, inputs from non-virtual racks, BTWs, and BTRs), use the buffers alternately and DEM sets a flag telling the host which table to use. All values that are supplied by the host (virtual discrete inputs only), should be placed in alternate buffers and the host must set a flag telling the DEM module which buffer to use. The following subsections describe double buffering.

Double buffering block transfers reduces the number of block transfers that the DEM module can buffer. Without double buffering, the total size of the block transfer data area is 20K, enough for 160 64-word block transfers. If you enable double buffering, this reduces the block transfer data area to 8K, enough for 64 64-word block transfers.

You can also achieve data synchronization by using host interrupts, in which case double buffering is not necessary.

3.9.1 Enabling Double Buffering

To enable double buffering, use the ENA_DBL_BUF byte located at offset 0x828. To enable the various types of double buffering, set various bits within the ENA_DBL_BUF byte. The following table summarizes the correlation between the bit positions and the types of double buffering enabled:

Type of Double Buffering	bit #
Discrete Outputs	0
Passive Discrete Inputs	1
Active Discrete Inputs	2
Block Transfer Writes	3
Block Transfer Reads	4

NOTE:

If you change ENA_DBL_BUF, you must send the reconfig command to the DEM module (set CONFIG to 0xff).

3.9.2 Discrete I/O Double Buffering

Two additional tables are used to allow double buffering for discrete I/O. These tables are the following:

Table	Location
OUT_DBL_TOG	0x1e00
INP_DBL_TOG	0x2e00

These tables are broken up in the same way as the RACK_ENABLE table. In a PLC-3 or PLC-5 system, rack 4 corresponds to 0x1e04 for outputs or 0x2e04 for inputs. Each byte is again broken up into quarters for partial racks, with two bits for each quarter rack.

There is a second input image table and a second output image table. The second output image table is located at offsets 0x0200 to 0x03ff and the second input image table is located at offsets 0x0600 to 0x07ff.

If the value of the two bits in the DBL_TOG table for a given rack/quarter is 00, use the lower or primary I/O table. If these two bits contain 01, use the higher or alternate buffer.

3.9.3 Discrete Output Double Buffering

If you enable discrete output double buffering, each time the DEM module receives an update for a particular rack/quarter from the programmable controller, the DEM module looks at the OUT_DBL_TOG table to see which image table was used last time (primary or alternate). The DEM module then updates the other table. In other words, if the primary table was used last, the alternate table is updated and vice versa. The DEM module then sets the OUT_DBL_TOG table to indicate which output table was just updated.

An application then checks to see which output table to use by checking the appropriate bits for the given start quarter in the appropriate byte for the given rack in the OUT_DBL_TOG table.

NOTE 1:

After checking the OUT_DBL_TOG table to determine which output image table to use for the given rack/quarter, the host must get the values from the output image table within 4ms (2 ms at 230 kbaud). If more time elapses, there is no guarantee that DEM won't update the data in the buffer at the same time as the host accesses it, which defeats the purpose of double buffering.

NOTE 2:

Partial racks are updated and therefore double buffered independently. This means that if, for example, a particular rack number has a real quarter rack, a virtual half rack and a virtual quarter rack connected, the PLC updates them independently. You must check the OUT_BUF_TOG table before you access corresponding locations in the output image table.

3.9.4 Passive Discrete Input Double Buffering

Passive discrete inputs are inputs that do not come from virtual racks running on the DEM module being used. Usually they are inputs from real racks but they could also be from other DEM modules running on other cards. Passive discrete input double buffering works similarly to output double buffering. If you enable passive discrete input double buffering, each time the DEM module receives an update from a particular rack/quarter to the programmable controller, the DEM module looks at the INP_DBL_TOG table to see which image table was used last time (primary or alternate). The DEM module then updates the other table. In other words, if the primary table was used last, the alternate table is updated and vice versa. The DEM module then sets the INP_DBL_TOG table to indicate which input table was just updated.

An application then checks to see which input table to use by checking the appropriate bits for the given start quarter in the appropriate byte for the given rack in the INP_DBL_TOG table.

NOTE 1:

After checking the INP_DBL_TOG table to determine which input image table to use for the given rack/quarter, the host must get the values from the input image table within 4ms (2 ms at 230 kbaud). If more time elapses, there is no guarantee that the DEM does not update data in the buffer as the host accesses it, which defeats the purpose of double buffering.

NOTE 2:

Partial racks are updated and therefore double buffered independently. This means that if, for example, a particular rack number has a real quarter rack, a virtual half rack and another real quarter rack connected, the inputs from the real quarter racks are updated independently. You must check the INP_BUF_TOG table before you access corresponding locations in the input image table.

3.9.5 Active Discrete Input Double Buffering

If you have enabled active discrete input double buffering, each time the DEM module is about to send discrete input data to the programmable controller, it checks the INP_DBL_TOG table for the particular rack/quarter that is to be updated. If the two bits for the given rack quarter contain 00_2, use the primary input image table; if the two bits contain 01_2, use the alternate input image table.

An application checks the INP_DBL_TOG table to see which input table was used last, then fills in the other table with the desired values, then sets the appropriate byte/bits of the INP_DBL_TOG table so that the DEM module uses the buffer which was just updated.

NOTE 1:

You must fill in all the input data for the virtual rack, in the alternate table. For example, if you enable a three-quarter virtual rack, you must fill in 6 words (12 bytes) of data each time the input image buffer is toggled.

NOTE 2:

After changing the INP_DBL_TOG table, the host application must wait at least 500µs before making any changes to the input image table that was not just updated.

3.9.6 Block Transfer Double Buffering

Block transfer double buffering is very simple. Each time the DEM module receives a new BTR or BTW, it puts the data in an appropriate buffer, and changes the pointer in BTR_LOC or BTW_LOC to point to the new buffer. The host application must read the pointer each time it accesses data for any BTR or BTW.

NOTE:

After creating a pointer to the BT data by reading the BTR_LOC or the BTW_LOC table, the host must get the values for the given BT from the BT buffer within 4ms (2 ms at 230 kbaud). If more time elapses, there is no guarantee that the DEM will not update data in the buffer at the same time that the host accesses it, which defeats the purpose of double buffering.

3.10 DEM Host Interrupts

The primary purpose of interrupts with the DEM module is to synchronize the host computer operation to the I/O scan of the programmable controller.

This section describes how to use interrupts with the DEM module. The following table lists all locations associated with interrupts.

Register Name	Location
INT_EN	0x830
VALID_DATA	0x831
INT_RACK_NUM	0x832
INT_BTR_NUM	0x833
INT_BTW_NUM	0x834
RACK_INT_ENABLE	0x2d00-0x2d1f

To globally enable or disable interrupts, use INT_EN (offset 0x830). Setting bit 0 to 1 enables interrupts; clearing bit 0 to 0 disables interrupts.

The interrupt service routine can use VALID_DATA (offset 0x831) to indicate that data in the tables is stable for at least 1.5ms. If the data is stable VALID_DATA contains 0xff; otherwise, it contains 0. VALID_DATA simply means that the next rack update has not begun yet and the I/O table will not change within 1.5ms. There is no guarantee that the data on the card is not being updated by DEM if the card data tables are accessed when VALID_DATA is not set.

INT_RACK_NUM (offset 0x832) indicates to the ISR which rack/quarter has just been updated. Note that the rack number is shifted left 2 bits and the quarter is placed in the least significant two bits. For example, an update to the first quarter of rack 3 causes 0x0c to be placed in INT_RACK_NUM. An update to the third quarter of rack 1 causes 0x6 to be placed in INT_RACK_NUM. Also note that this value is updated regardless of whether interrupts are enabled. The host can poll this location to determine the current location of the I/O scan.

INT_BTR_NUM (offset 0x833) indicates to the host that a BTR has occurred to the rack in INT_RACK_NUM since the previous I/O scan. If this byte is non-zero, bits 0-3 contain the I/O group and slot which sent the block transfer read. The host then checks the BTR data for that location. The following table summarizes INT_BTR_NUM:

7	6	5	4	3	2	1	0
BTR update	Always 0			I/O group of updated BTR			Slot

INT_BTW_NUM (offset 0x834) indicates to the host that a BTW has occurred to the rack in INT_RACK_NUM since the previous I/O scan. If this byte is non-zero, bits 0-3 contain the module group and slot which received the block transfer write. The host then checks the BTW data for that location. The following table summarizes INT_BTW_NUM:

7	6	5	4	3	2	1	0
0	BTW update	Always 0		I/O group of updated BTW			Slot

The RACK_INT_ENABLE (offset 0x2d00 - 0x2d1f) table is divided in the same way as the RACK_ENABLE table. To enable interrupts for a specific rack/quarter, place 01 in the two bits that correspond to start quarter, in the byte that corresponds to the rack, as with the other tables described. If interrupts are enabled for a rack that the programmable controller is not updating, no interrupt is generated for that rack. The following line of C code enables interrupts for a partial rack that starts at the third quarter of rack one. Note that the two bits to be used are cleared first, and only the bits to be used are changed. This is not necessary but it is good programming practice.

```
RACK_INT_ENABLE[1] = (RACK_INT_ENABLE[1] & ~(0x3 << 6) ) | (0x1 << 6);
```

DEM Host Interrupt Timing

The host must respond to the DEM interrupt within 2ms. The ISR length must not exceed 1.5ms. If these conditions are not met, two undesirable consequences may arise:

1. The data in the I/O tables and the various interrupt variables (INT_RACK_NUM...) could change, in which case they would not reflect the information that was intended at the time the interrupt was generated.
2. The interrupt for a given rack/quarter could be missed altogether.

3.11 Running DEM

Following is the procedure an application should take before going online.

1. Read the CHK bytes to determine whether it's OK to begin to access DEM.
2. Set BT_CONFIG to 0.
   Set CONFIG to 0xff
   Wait for DEM to clear CONFIG to 0.
3. Set BT_BASE to 0x3000
4. Set BAUD_RATE and BAUD_230K to match the network baud rate.
5. Enable virtual racks and set rack options. Write to the:
   RACK_ENABLE table
   RACK_END table
   RACK_OPTION table
6. Install block transfers on virtual racks. For each block transfer, write to the :
   BTR or BTW location table
   BTR or BTW length table
   and update BT_BASE
7. Set any initial data values such as discrete inputs or block transfer read data on virtual racks
8. Set BT_CONFIG to 0xff
9. DEM is now online and ready to run.

The sample program DEMCNF.C and the accompanying configuration files can be used to configure and run DEM.

3.11.1 Reconfiguring DEM

If an application program needs to change the baud rate or clear the data tables, repeat these steps. Changes to the BAUD_RATE flag do not take effect until you set the CONFIG flag. Setting the CONFIG flag also clears the data tables on the card; in that case information about the I/O configuration must be reinstalled.

3.12 Summary of Memory Locations

The table below provides a summary of memory locations for interfacing with the DEM software module.

Name	Offset	Type	Function
OUTPUT	0x000-0x1ff	R	primary output image table
DBL_OUT	0x200-0x3ff	R	alternate output image table for use with double buffering
INPUT	0x400-0x5ff	R/W	primary input image table
DBL_INPUT	0x600-0x7ff	R/W	alternate input image table for use with double buffering
BAUD_230	0x800	R/W	sets the baud rate
BAUD_RATE	0x801	R/W	sets the baud rate
CONFIG	0x803	R/W	sets the baud rate and clears all data tables before starting communications
PLC_TYPE	0x804	R	DEM sets this byte to 0xff if the PLC is a PLC-2; 0 otherwise
CHK	0x80b 0x80c 0x80d	R	host can use these bytes to check that DEM is present. The three bytes contain 0xc3 0x04 0x00 when the DEM module is loaded and running
BT_BASE	0x820	R/W	start of block transfer area for real racks. Needed only if you are using block transfers on virtual racks
BT_CONFIG	0x822	R/W	flag used when host is installing block transfers
COMM	0x824	R	communication flag, DEM sets it to 0xff when it's receiving from the PLC, otherwise it's 0
ALL_VIRT_GOOD	0x825	R	non-zero if all virtual racks have good status
ENA_DBL_BUF	0x828	R/W	set by host to enable double buffering. Various bits enable different types of double buffering

Name	Offset	Type	Function
INT_EN	0x830	R/W	bit 0 is global DEM interrupt enable
VALID_DATA	0x831	R	DEM sets it to 0xff when an interrupt is generated and clears it at least 1.5 ms before the next I/O update begins. Used to indicate data tables are stable
INT_RACK_NUM	0x832	R	indicates the rack/quarter which was most recently updated
INT_BTR_NUM	0x833	R	indicates the BTR module group which was updated for the given rack. Used only with interrupts.
INT_BTW_NUM	0x834	R	indicates the BTW module group which was updated for the given rack. Used only with interrupts.
Diagnostic counters	0x8c0-0x8d1	R/W	for monitoring communication
RACK_END	0xc00-0xc1f	R/W	rack end table, used with partial racks
BTR_LOC	0x1000-0x13ff	R	pointer to block transfer read data in buffer
BTR_EVENT	0x1400-0x15ff	R/W	DEM sets the appropriate location in this table when a BTR is updated
BTR_LEN	0x1800-0x19ff	R	length of block transfer read data in buffer
RACK_STATUS	0x1c00-0x1c1f	R	rack status table
OUT_DBUF_TOG	0x1e00-0x1e1f	R	used only with double buffering. Indicates which of the two output image tables should be used for a given rack/quarter
BTW_LOC	0x2000-0x23ff	R	pointer to block transfer write data in buffer
BTW_EVENT	0x2400-0x25ff	R/W	DEM sets the appropriate location in this table when a BTW is updated
BTW_LEN	0x2800-0x29ff	R	length of block transfer write data in buffer
RACK_ENABLE	0x2c00-0x2c1f	R/W	enables virtual racks

Name	Offset	Type	Function
RACK_INT_ENA	0x2d00-0x2d1f	R/W	enables interrupts for any rack/quarter
INP_DBUF_TOG	0x2e00-0x2e1f	R/W	Used only with double buffering. For passive discrete inputs, indicates which of the two input image tables should be used for a given rack/quarter. For active inputs, host sets to indicate to DEM which table to use for a given rack/quarter.
RACK_OPTION	0x2f00-0x2f1f	R/W	enables various options for any rack/quarter
block transfer data	0x3000-end of memory	R/W	block transfer data

3.13 Sample Programs

The distribution diskette contains C source code for sample programs that demonstrate the use of the DEM module. All the programs were developed using Borland C++ but should be easily portable to other C compilers.

DEM.H

This file contains structures and definitions for accessing data on the card.

DEM.C

DEM.C contains functions that:

- set the baud rate
- reconfigure the card
- check the communication status
- add or delete virtual racks
- display rack status
- display inputs or outputs
- set inputs for virtual racks
- use block transfers

These routines may be used in your application.

DEMCNF.C

This program, along with the sample files *.CFG, shows how to configure DEM.

Technical Data

Card Type	5136-SD-VME-R
Function	VMEbus card for Allen-Bradley remote I/O network
Description	IEEE 1014, 6U height, P1 compatible
	Memory SD16, SD08(EO), SADO24
	Registers SD08(O),SADO16
	Standard Addressing: 64 Kbytes on any 64 Kbyte boundary
	Short addressing: 6 bytes on any 1 Kbyte boundary
	Interrupt capability: switch selected level 1-7, software set 8-bit vector, release on acknowledge (ROAK)
Current Consumed	1 A at 5V
	from pins 32 of rows A, B, and C of the P1 connector
Environmental	operating temperature 0-50 degrees Celsius
	storage temperature 0 -70 degrees Celsius
	Operating Humidity 5 to 95 % non-condensing
	storage humidity 0 to 95%
Card connector	Phoenix MSTB1.5/6ST-5.08
Cable	Belden 9463, twinaxial, 20 AWG

Acknowledgments

PLC is a registered trademark of Allen-Bradley.

All other trade names referenced are trademarks or registered trademarks of their respective companies.

Technical Support

Before you call for help ...

Please ensure that you have the following information readily available before calling for technical support.

- Card type and serial number
- Computer make and model and hardware configuration (other cards installed)
- Operating system type and version
- Details of the problem you are experiencing; application module type and version, target network, circumstances that caused the problem.

Getting Help

Technical support is available during regular business hours (eastern standard time) or by fax, mail or e-mail.

Technical Support

SST

50 Northland Road

Waterloo, Ontario N2V 1N3

Voice: (519) 725-5136

Fax: (519) 725-1515

email: techsupport@sstech.on.ca

website: www.sstech.on.ca

Software Updates

The current distribution software for the 5136-SD-VME-R is available from our website at www.sstech.on.ca

Warranty

SST warrants all new products to be free of defects in material and workmanship when applied in the manner for which they were intended and according to SST's published information on proper installation. The Warranty period is one year from the date of shipment for all cards except the following which carry a 10 year warranty from date of purchase: 5136-SD, 5136-SD-104, 5136-SD-VME, 5136-SD-VME/2, 5136-DN, 5136-DN-PCM, 5136-DN-VME, 5136-DNP, 5136-PFB, 5136-PFB-104, 5136-PFB-PCI, 5136-PFB-VME and 5136-CN.

SST will repair or replace, at its option, all products returned to factory freight prepaid, which prove upon examination to be within the Warranty definitions and time period.

The Warranty does not cover costs of installation, removal or damage to user's property or any contingent expenses or consequential damages. Maximum liability of SST is the cost of the product(s).

Product Returns

If it should be necessary to return or exchange items, please contact SST for a Return Authorization number.

SST

50 Northland Road

Waterloo, Ontario N2V 1N3

Voice: (519) 725-5136

Fax: (519) 725-1515

—S—

—T—

—V—

—W—