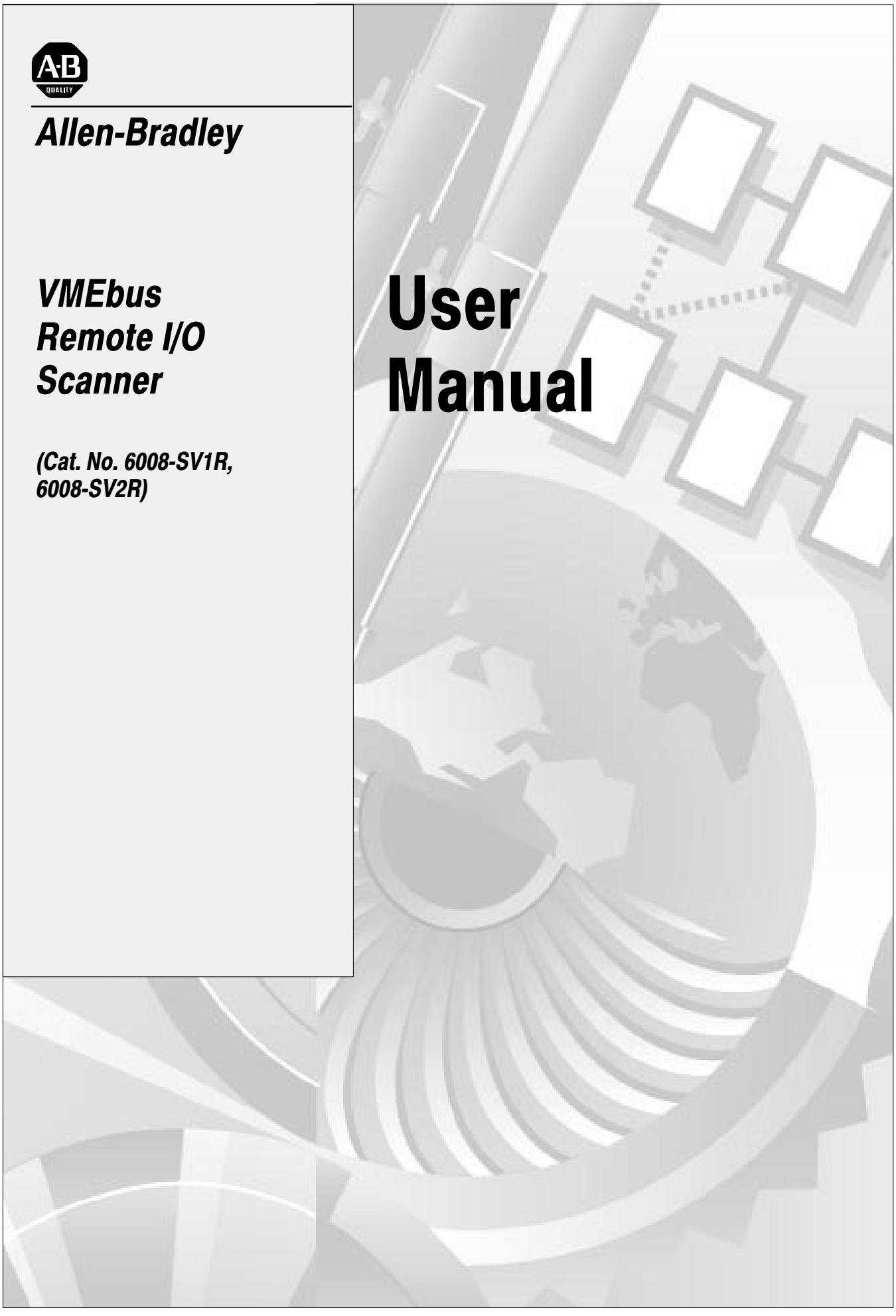


Allen-Bradley

VMEbus Remote I/O Scanner

(Cat. No. 6008-SV1R, 6008-SV2R)

User Manual

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION: This notation identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss.

Attention statements help you to:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: This notation identifies information that is critical for successful application and understanding of the product.

Table of Contents

Using This Manual

Introduction

This manual describes how to install and use the VMEbus remote I/O scanners (catalog numbers 6008-SV1R and 6008-SV2R).

Audience

You should have experience in system development and integration and in writing software for VMEbus master processors. You should also have a working knowledge of the C programming language, including the concepts of structures and pointers. Knowledge of Allen-Bradley 1771 I/O products is helpful but not essential.

Required Hardware

You need a VMEbus-compatible VME master processor to set up and control the VMEbus remote I/O scanner. You install the scanner in a standard 6U, full-height VME rack.

The 1771 I/O modules that the scanner monitors and controls depend on your application. You also need an adapter in the 1771 chassis to allow communication between the scanner and the I/O modules. You can use any A-B adapter module or a PLC-5 processor that operates in adapter mode.

Terms

This table defines common terms:

This term:	Refers to the:
scanner	both remote scanners (catalog numbers 6008-SV1R and 6008-SV2R)
VME master processor	main CPU of your VME system The VME master processor runs the application program that accesses the scanner. A VME system can have more than one VME master processor, each assigned different duties and both accessing the same scanner(s).
VME chassis	frame that VME cards are mounted in Both the scanner and the VME master processor are mounted in the chassis along with other VME hardware.
VMEbus	circuit board or backplane mounted in the chassis that the scanner, the VME master processor, and other VME cards plug into
I/O chassis	Allen-Bradley 1771 series I/O chassis This is the frame that houses the I/O modules, power supply, and adapter or PLC processor.
input image table	area of global memory in the scanner that contains the data from the input terminals of input modules When an input switch is closed its corresponding input bit in the image table is set to 1.
output image table	area of global memory in the scanner that contains output data for terminals of output modules When a bit is set to 1, the corresponding output turns on.
block transfer	transfer of data between an intelligent I/O module and a scanner A block transfer sends as many as 64 words of data at a time.
general data area	designated area of global VME memory, existing within the scanner, that is used to pass information between the scanner and a VME master processor. Scanner commands are processed in this data area.
global RAM	an area of global VME memory in the scanner that can be accessed by both the scanner and the VME master processor(s). This area of memory is the key means for communication between the scanner and the VME master
semaphore bit	bit that indicates whether part of the global RAM (the general data area) is being used. Typically this bit is used to prevent multiple masters or the scanner from writing to the general data area simultaneously.
scan list	list that the scanner maintains internally to determine the I/O racks that it is to scan, and the order in which it is to scan them. You create the scan list using the AUTOCONFIGURE command or the SCAN LIST command.

Chapter 1

Scanner Overview

Using This Chapter

This chapter provides an overview of the scanner. This chapter describes how the scanner relates to the VMEbus and to the remote I/O link.

Introduction

The VMEbus scanners (catalog number 6008-SV1R and 6008-SV2R) monitor and control remote Allen-Bradley I/O modules without using a PLC processor. Use your VME master processor(s) to manage as many as 32 racks of remote A-B I/O (16 per scanner channel).

The scanner communicates with I/O adapters that reside in the left slot of a remote chassis and with other products that have node adapters built into them. The scanner transfers the information necessary to control discrete and block-transfer data to and from the VMEbus.

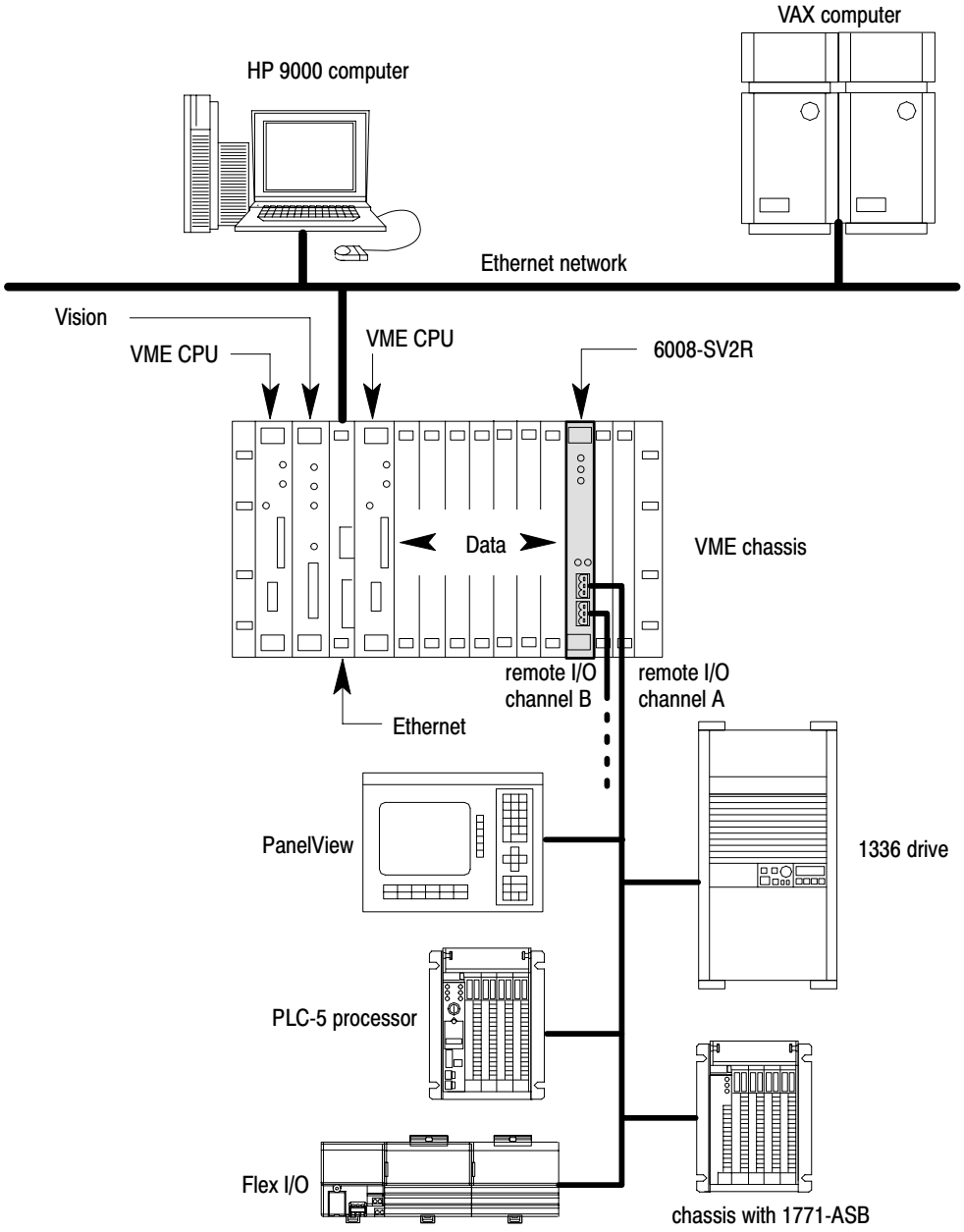
The VMEbus scanner physically resides in the VME chassis. The scanner occupies one 6U (full-height) VME slot. The scanner uses the P1 connector to interface to the VMEbus. You can use more than one scanner in your VME system to create large and flexible I/O subsystems.

To the VMEbus, the scanner is a memory-mapped slave that responds to 8-bit or 16-bit accesses in either A16 or A24 address space. The scanner can act as a VME interrupter on any of the seven VMEbus interrupt lines.

The SV1R and SV2R scanners replace the Allen-Bradley 6008-SV scanner. The SV1R has one remote I/O channel; the SV2R has two remote I/O channels. The SV1R and SV2R scanners have an extra embedded communication microprocessor, which gives them more flexibility and faster performance than the 6008-SV scanner. The new scanners are backward-compatible with the 6008-SV scanner and offer:

- continuous block-transfer operations
- each remote I/O channel supports as many as 16 racks of remote I/O
- configurable scan rate at 57.6, 115.2, or 230.4 kbps
- VME interrupt signals change in the scanner input table

Figure 1.1
System connection overview



Note: The 6008-SV1R scanner is interchangeable with the 6008-SV2R scanner, except that the SV1R scanner supports only one channel of remote I/O.

Each scanner channel supports as many as 32 physical adapters (16 logical racks). Figure 1.2 shows the front panels of the scanners.

Figure 1.2
Scanner front panel

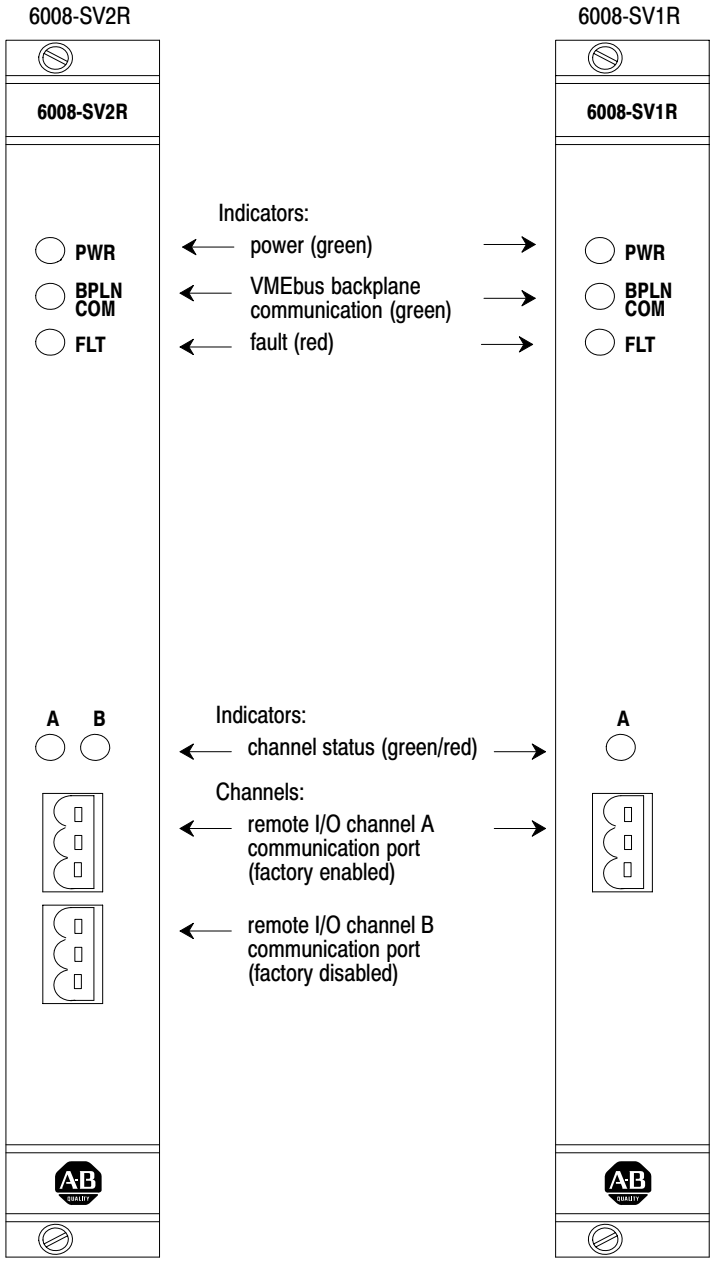


Table 1.A
Significance of scanner indicators

When this indicator:	is:	it means:
PWR (power) green LED	illuminated	power is applied to the module
BPLN COM (backplane communication) green LED	illuminated for approximately a half second	a VMEbus access is made to the scanner board
FLT (fault) red LED	illuminated	the scanner board is reset, performing a self test, or a fault has been detected

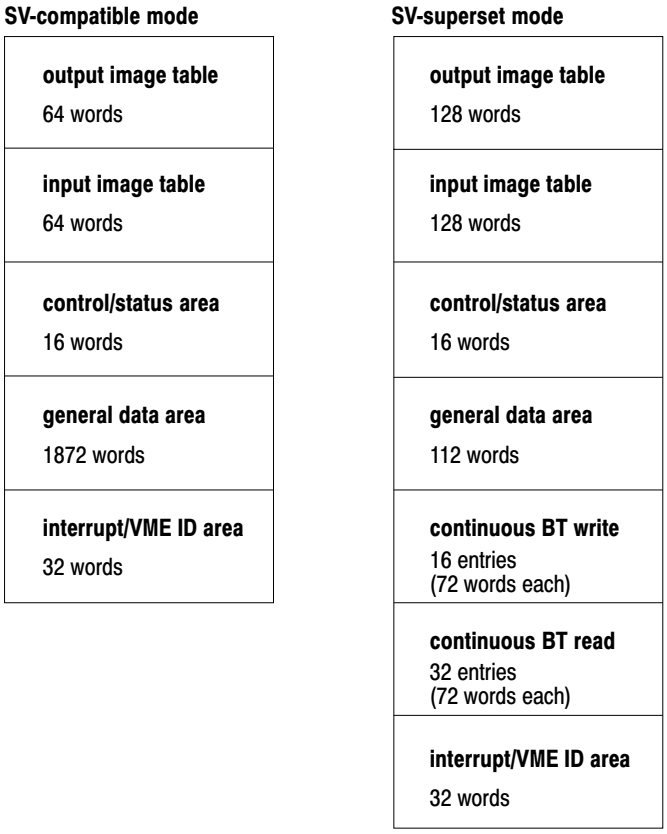
Table 1.B
Significance of channel status indicators

When the status indicator is:	the scanner:
off	is off line
green	is on line, in Run mode, and scanning the racks in the scan list
blinking green	is on line, in Run mode, and scanning only some of the racks in the scan list
red	has an unrecoverable fault
blinking red	has a recoverable fault

VMEbus Relationship

The scanner complies with the VMEbus specification (revision C.1) and responds to VME masters on the VMEbus. The scanner appears as an area of global VME RAM to other master processors on the VMEbus. This area contains the I/O image area, control/status area, general data area, and interrupt ID area.

memory map for one channel



For more details on these memory areas, see chapter 5 or 6 for SV-compatible mode or SV-superset mode, respectively.

There is no direct communication between a VME master processor and the discrete I/O, rather the VME master processor communicates with the I/O image table in the scanner (shown above). The VME master processor reads the status of inputs from the input image table and controls the outputs by writing data to the output image table.

These VMEbus transfers are asynchronous to the scanner's I/O update. This means there is no way to know exactly when the data being put in the output image table will be sent to the appropriate I/O rack. Data is sent to an adapter only when that adapter is being scanned. Best-case timing is if the data is placed in the output image table just before the specified adapter is scanned; worst-case timing is if the data is placed in the output image table just after the specified adapter was scanned. In the worst-case scenario, the data does not reach the specified I/O rack until the next time that adapter is scanned.

How the Scanner Responds to VME Signals

The scanner can generate interrupts on any of seven request levels (IRQ1-IRQ7). When a VMEbus master acknowledges the interrupt, the scanner replies with a vector (status/id) using the odd 8 bits of the data bus.

Important: The VME master processor might "crash" if there is no software routine written to process an interrupt from the scanner. Or, you can use the SETUP command to configure the scanner so that it never generates interrupts, in case no interrupt software routine has been written. If the scanner does not generate interrupts, the application program must "poll" the scanner to see when commands have been processed.

The scanner responds to Data Transfer Bus (DTB) cycles initiated by masters that transfer data 16 bits at a time or 8 bits transferred in an even and odd format (D16, D08EO). The scanner works in the 16-bit (short) addressing mode or the 24-bit (standard) addressing mode.

The scanner responds to common VME signals as follows:

This VME signal:	means:
SYSFAIL	When the scanner recognizes a SYSFAIL signal, it can either ignore the signal or shut itself down, depending on how the scanner is configured. When the scanner shuts down, the I/O serviced by the adapter either resets to a default condition or holds all of its current values, as determined by switches on the I/O chassis backplane. When the scanner is faulted or shut down, it asserts SYSFAIL on the VMEbus.
ACFAIL	When the scanner recognizes an ACFAIL signal, it shuts itself down because this means that power will soon be gone. When the scanner shuts down, the I/O serviced by the adapter either resets to a default condition or holds all of its current values as determined by switches on the I/O chassis backplane.
SYSRESET	If SYSRESET is asserted on the VMEbus, the scanner resets itself and goes through its initialization tests. The scanner does not clear (reset to 0) the input and output image tables. After a SYSRESET signal, you have to wake up the scanner, the same as a power-up situation.

For more information, see the VMEbus specification (revision C.1) published by VITA (VMEbus International Trade Association), 10229 N. Scottsdale Rd., Suite B, Scottsdale, AZ, 85253, (602) 951-8866. Contact a VITA representative for a copy.

VMEbus Address Modifier Codes

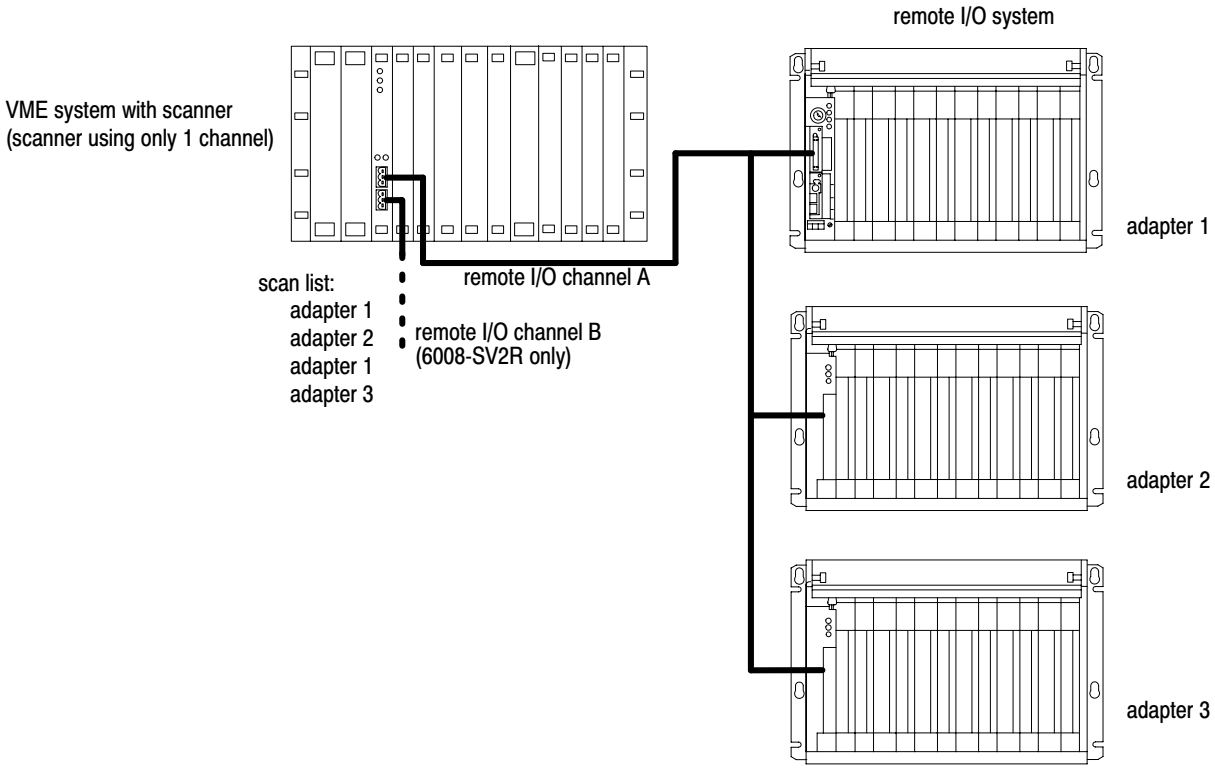
The scanner can respond to the following VMEbus address modifier codes, depending on how you configure the scanner's address space and response to VME accesses.

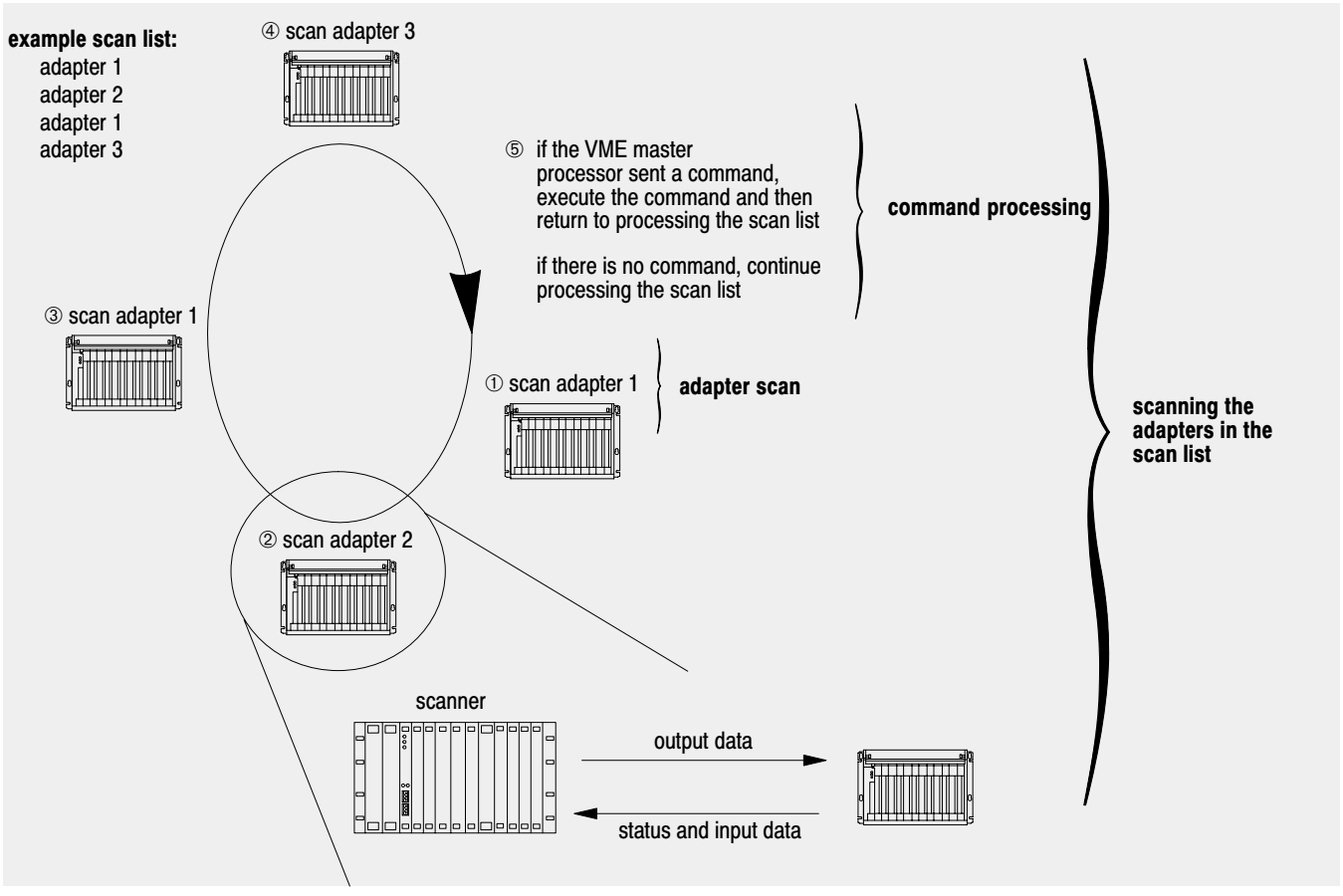
This code (hex):	means:
3D	standard (A24) supervisory access
39	standard (A24) non-privileged access
2D	short (A16) supervisory access
29	short (A16) non-privileged access

How the Scanner Scans

The scanner runs asynchronously to other VME master processors. Once in Run mode, the scanner continuously scans all the adapters in its scan list. The scan list identifies which adapters to scan and in what order to scan them. An adapter can appear several times in the scan list. For more information about using the scan list, see chapter 4.

When the scanner scans an adapter, it brings in digital input data and places the data in the scanner's input image table. At the same time, the scanner sends digital output data to the adapter.


remote I/O system
VME system with scanner
(scanner using only 1 channel)
adapter 1
scan list:
adapter 1
adapter 2
adapter 1
adapter 3
remote I/O channel A
remote I/O channel B
(6008-SV2R only)
adapter 2
adapter 3


example scan list:
adapter 1
adapter 2
adapter 1
adapter 3
④ scan adapter 3
⑤ if the VME master processor sent a command, execute the command and then return to processing the scan list
if there is no command, continue processing the scan list
command processing
③ scan adapter 1
① scan adapter 1
adapter scan
scanning the adapters in the scan list
② scan adapter 2
scanner
output data
status and input data

Getting the Scanner's Attention

For a VME master processor to get the scanner's attention, it must write a value (any value) to any byte in the scanner's identification area. This area is located in the last 64 bytes of the global RAM for each scanner channel.

The scanner gets the attention of a VME master processor by generating a VMEbus interrupt to which the VME master processor must respond. This interrupt is sent when the scanner finishes commands that a VME master processor initiated.

Comparison to 6008-SV Scanner

The 6008-SV1R and 6008-SV2R scanners replace and are backward compatible with the Allen-Bradley 6008-SV VMEbus remote I/O scanner. The 6008-SV1R and 6008-SV2R scanners offers these improvements:

- each remote I/O channel controls as many as 32 adapters
- configurable communication rate of 230.4, 115.2, or 57.6 kbps lets you select I/O scan time
- embedded communication microprocessor increases scanner performance
- VME interrupt signals change in the scanner input table
- configurable VME operating mode lets you select the scanner features you need for your application

Important: The SCAN LIST command is the only difference between the 6008-SV1R and 6008-SV2R scanners and the 6008-SV scanner. If you use that command, you must modify the command to specify the rack size.

The 6008-SV1R and 6008-SV2R also offer improved block transfer operations. In addition to single block transfer operations, the 6008-SV1R and 6008-SV2R support continuous block transfer operations. A single block transfer is a single read or write transfer to a specific intelligent I/O module. If your application needs to continuously poll a module to receive up-to-date data, use a continuous block transfer request. The continuous block transfer requests uses less programming overhead than programming a single block transfer request each time you need the data.

Operating Modes

Before you begin using the scanner, you have several choices to make concerning how the scanner operates. You need to specify how the scanner operates in the VME system and how you want to program the scanner.

Selecting VME Operating Mode

The scanner offers two VME operating modes. The mode you select determines the command set available to the scanner and the memory structure the scanner uses. You set a switch on the scanner to specify the operating mode you want.

If you want:	**select this VME operating mode:**
the scanner to operate exactly as the 6008-SV scanner This mode is compatible with the 6008-SV so you can run previously-developed applications with minor modifications. Select this mode if you are replacing a 6008-SV with a 6008-SV2R and do not want to modify your application. In any application that uses the SCAN LIST command with the 6008-SV scanner, you must modify the command to specify the rack size. For more information see chapter 5.	SV-compatible
the scanner to use the new commands and additional memory, as compared to the SV-compatible mode This mode provides additional features, as compared to the 6008-SV. For more information, see chapter 6.	SV-superset

Important: An application developed for one operating mode will not work in another operating mode.

Selecting a Programming Mode

Select the appropriate programming mode for programming the scanner.

Table 1.C
Programming modes

If you want these conditions:	select this programming mode:
• the scanner doesn't send output information to the adapters • all module outputs are reset (off); outputs are disabled, so they remain reset • discrete input information is updated • the scanner doesn't send block-transfer requests to the adapters, but the scanner will queue the requests from the VME master processor	Program
• the scanner sends output information to the adapters • all module outputs are held reset (off) outputs are disabled, so they remain reset • discrete input information is updated • the scanner sends block-transfer requests to the adapters, but actual outputs are disabled (reset)	Test
• the scanner sends output information to the adapters • input information is updated • the scanner sends block-transfer requests to the adapters • all outputs are allowed to energize	Run

When your application program first starts the scanner with the SETUP command, the scanner is in the Program mode. Your program must issue a SET MODE command to change the scanner to Run mode.

VME Master Processor Watchdog Timer

The VME master processor must issue a valid command to the scanner at least once in a user-specified time period (the default is 500 msec). If the scanner fails to see a valid command from a VME master processor in this time period (as counted by the watchdog timer), the scanner resets itself and repeats its startup initialization sequence. This causes the I/O racks on the link to fault within 100 msec and the I/O all turn off or remain in their last state, depending on the switch setting on the I/O chassis.

You can disable the watchdog timer or change its timeout period with the SETUP command.

To keep the watchdog from shutting down the scanner, periodically issue a LINK STATUS command. This command provides the application program with important diagnostic information about the status of the I/O link and, at the same time, causes the least amount of overhead for the scanner to complete the command.

To debug your application program you can select debug mode and disable the watchdog timer using the SETUP command.

ATTENTION: Unwanted machine action can result from disabling the VME master processor watchdog. When the VME master processor watchdog is disabled, the scanner has no way of knowing that communication has been lost with your VME master processor and will continue to send data from the output image table to the output modules.

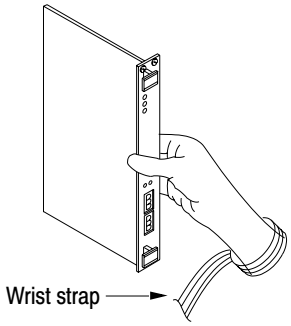
Chapter 2

Installing the Scanner

Using This Chapter

This chapter explains how to install the scanner and connect it to a remote I/O link. For information about programming and using the scanner, use the flow chart preceding each chapter to determine where to find the information you need.

Handling the Scanner



The scanner is shipped in a static-shielded bag to guard against electrostatic damage. Electrostatic discharge can damage integrated circuits or semiconductors in the scanner. Avoid electrostatic damage by observing these precautions.

- Remain in contact with an approved ground point while handling the scanner (by wearing a properly grounded wrist strap).
- Do not touch the backplane connector or connector pins.
- When not in use, keep the scanner in its static-shielded bag.

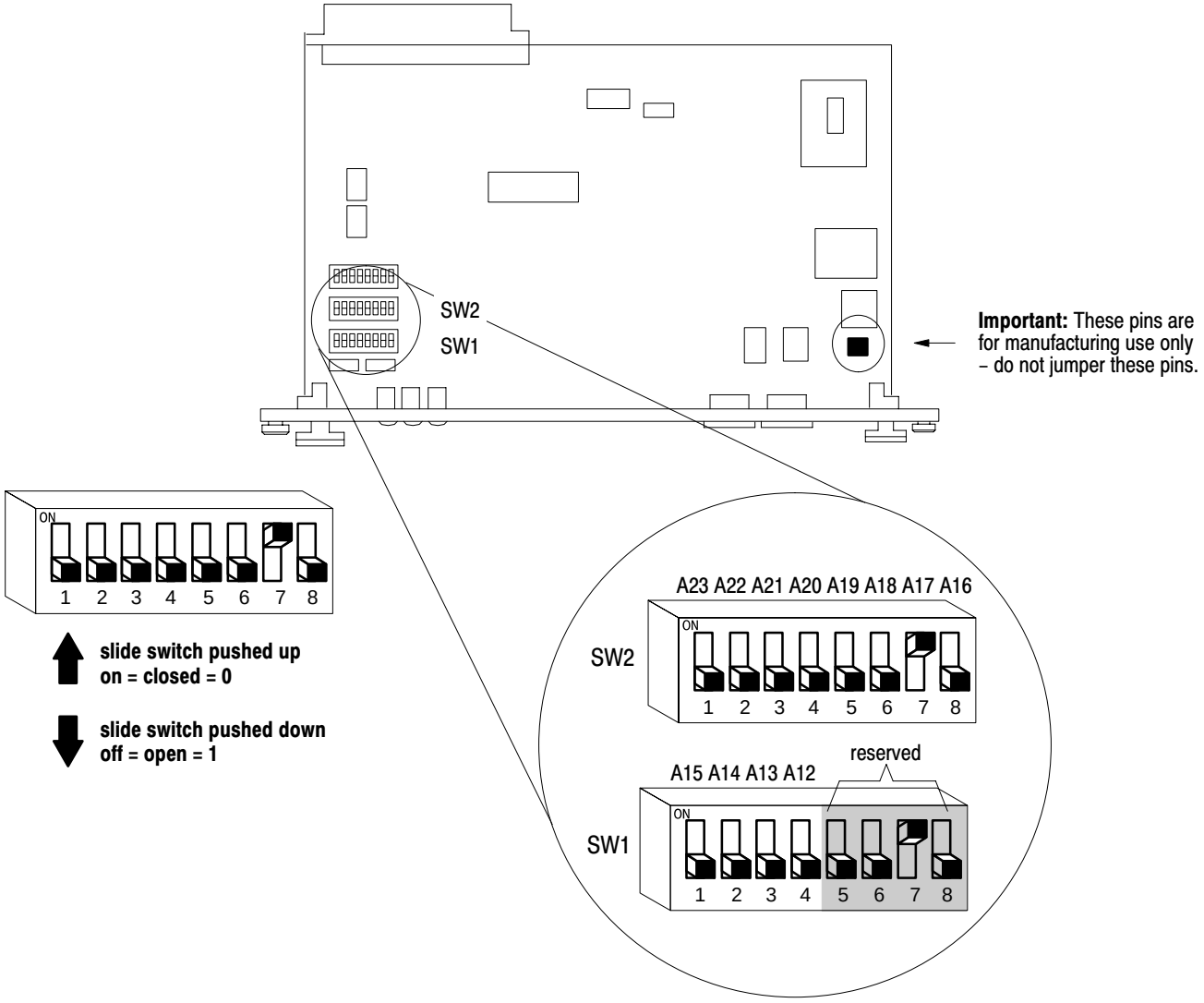
Setting Switches

The scanner has several on-board switches you set to configure:

- address space
- VME operating mode
- VME address space
- scanner responses to VME accesses

Determine the VMEbus A24/A16 Address Space

Use this diagram and the example to set SW1 and SW2 for the correct VMEbus address space.



In this VME operating mode:	these bits:		are valid for this address space:
SV-compatible	A23 through A12	1 channel active	A24
	A23 through A13	2 channels active	
	A15 through A12	1 channel active	A16
	A15 through A13	2 channels active	
SV-superset	A23 through A13	1 channel active	A24
	A23 through A14	2 channels active	

If there are switches not accounted for in a particular address space, such as the switches for A16-A23 for SV-compatible, A16 address space, the switch position does not affect scanner operation.

Determine the Operating Mode, Address Space, Scanner Response, and Rack Configuration

Use this diagram and Table 2.A on page 2-4 to set SW3.

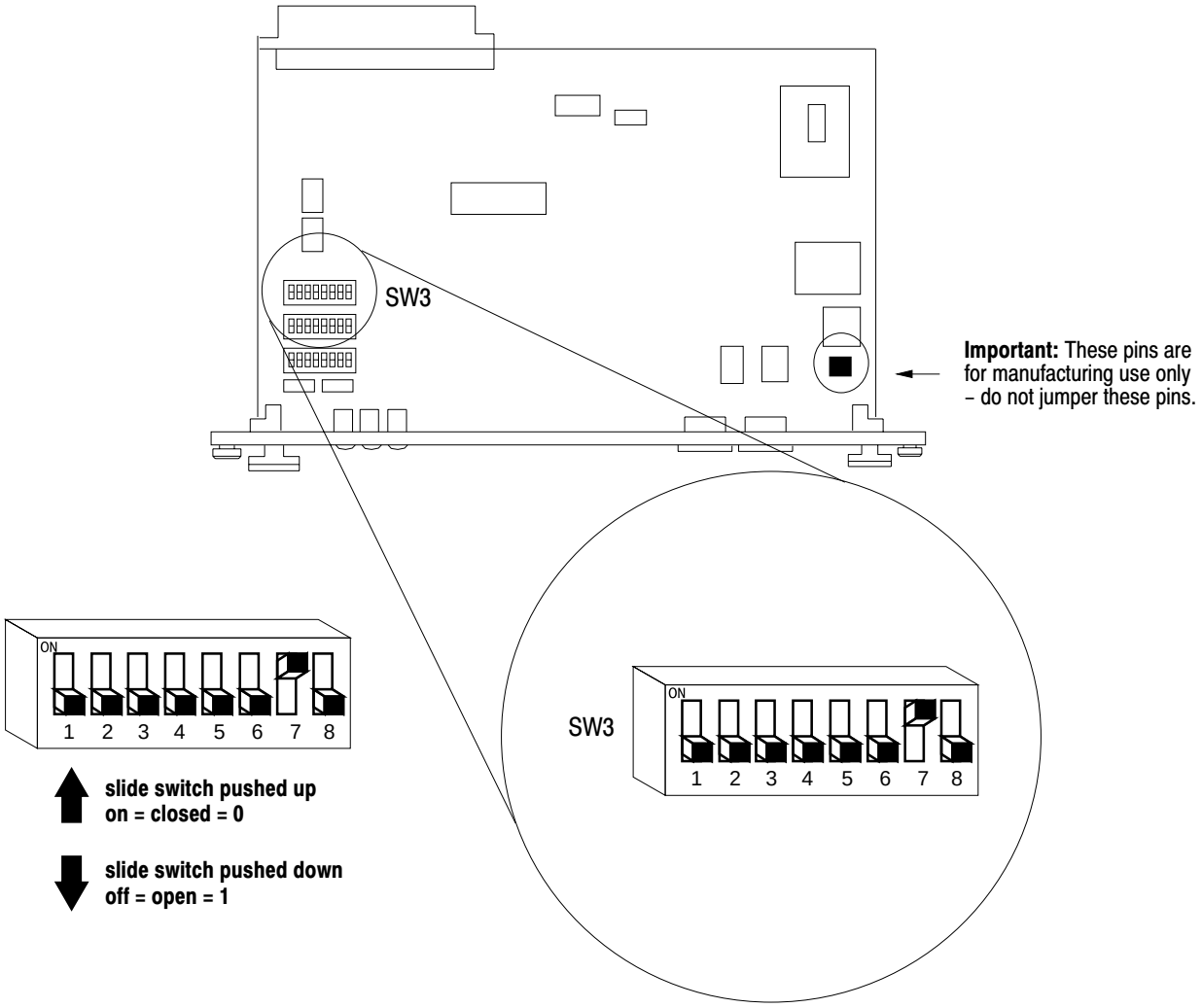
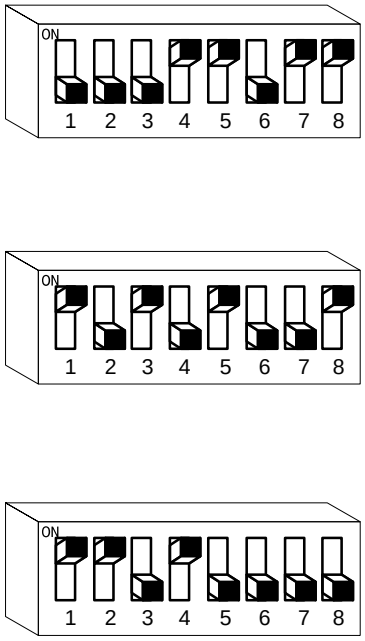


Table 2.A
Switch settings for SW3

This switch:	configures:	with these options:
switch 1, 2, 3	not used	set to off
switch 4, 5	VME operating mode	switch 4 / switch 5 on / on — SV compatible on / off — SV superset off / off — reserved off / on — reserved
switch 6	VME address space	on — A16 off — A24 Select A24 if you select SV-superset as the VME operating mode.
switch 7	how the scanner responds to VME accesses	on — responds to non-privileged and supervisory VME accesses (2D, 3D, 29, and 39 address modifiers) off — responds to supervisory VME accesses (29 and 39 address modifiers)
switch 8	which channels are active	on — only channel A is active off — both channel A and B are active this switch is ignored if you are configuring the 6008-SV1R

For example:



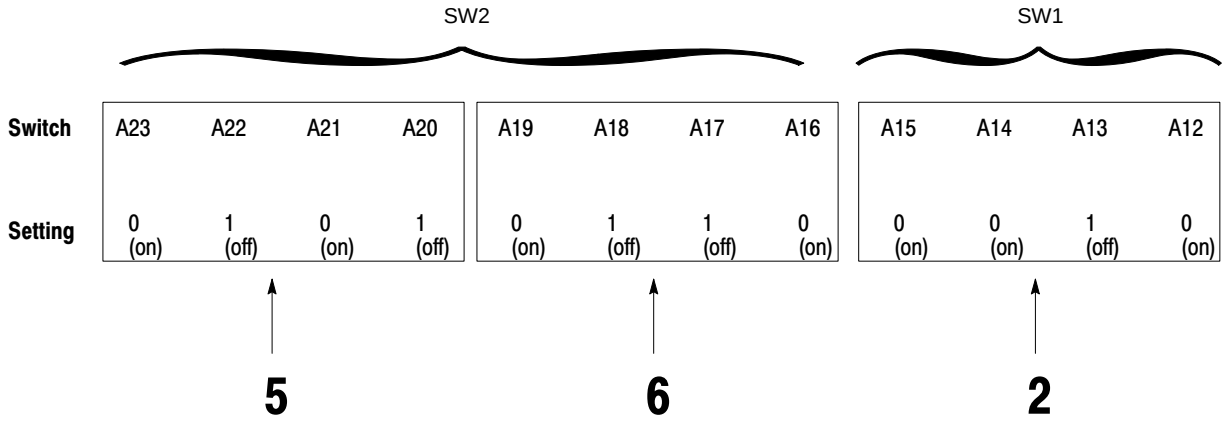
These switch settings specify:
- SV-compatible mode
- A24 address space
- response to both non-privileged and supervisory access
- only channel A is active

These switch settings specify VME address 562,000.

Specifying VME addresses

You specify the first digit (for A16) or first 3 digits (for A24) of the address space; the remaining digits are always 0. The switches are set from left to right. For example, to set the address space at 562,000 (hex) in A24 with one channel active, set the bits as:

VME address space of 562,000 (hex)
A24 address mode with 1 channel active

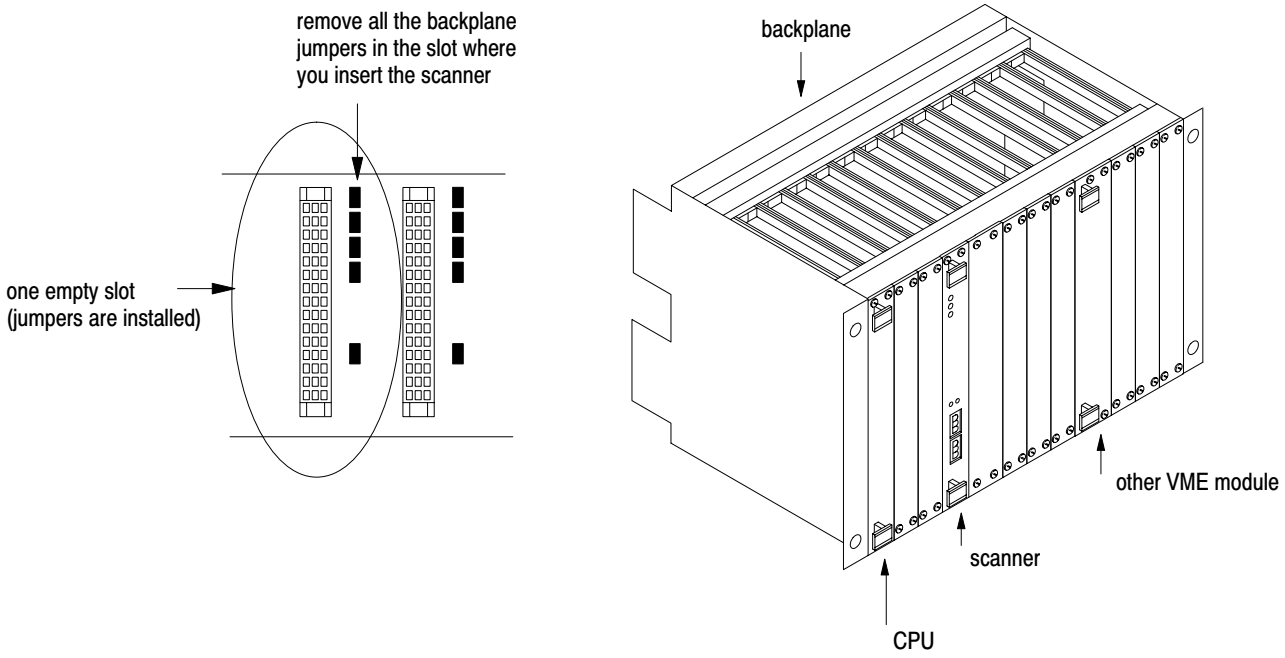


The last three digits in 562,000 (hex) address are already determined by the scanner, so there are no switches to set.

Removing VME Backplane Jumpers

The VMEbus has several daisy-chained control signals. Almost all VMEbus backplanes contain jumpers for these control signals to allow systems to operate with empty slots. There are five jumpers per VME slot, one for each of the four bus-grant arbitration levels and one for the interrupt-acknowledge daisy chain.

Depending on the backplane manufacturer, the jumpers can be on the rear pins of the J1 connector or alongside it on the front of the backplane. The scanner uses 1 slot of the VME backplane. Remove these jumpers from the slot where you plan to insert the scanner.



Grounding the VME Chassis

Allen-Bradley makes specific recommendations for properly grounding its racks so that their operation is as safe and error-free as possible. VME systems, on the other hand, may have no formal specifications for grounding the VME chassis frame. Allen-Bradley recommends that you ground the VME chassis frame and that you connect the logic ground (common) of the VME power supply to the chassis frame's earth ground.

The specific procedure for grounding a VME chassis varies depending on the style of the chassis. Read the Programmable Controller Wiring and Grounding Guidelines, publication 1770-4.1, for information on how Allen-Bradley racks are grounded, and try to ground your VME chassis frame in a similar way.

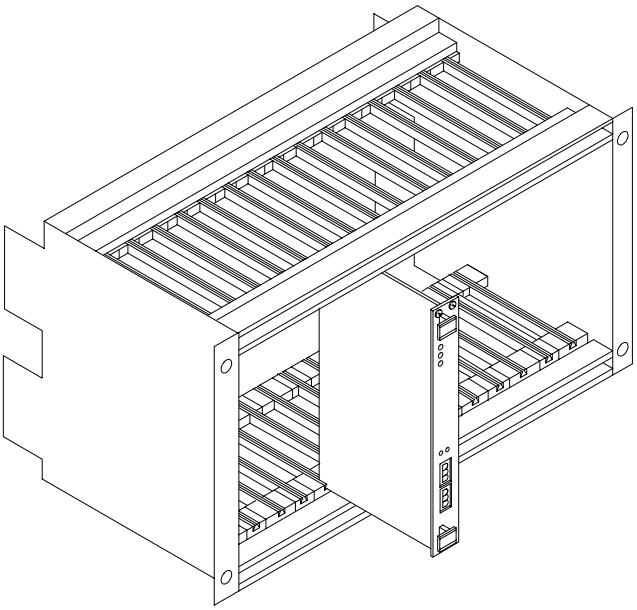
Inserting the Scanner

Insert the scanner in one slot in a 6U (full-height) VMEbus chassis.

ATTENTION: Make sure that your VME system is powered off. The scanner is not designed to be inserted or removed from a live system.

ATTENTION: Avoid touching the circuit board and connectors. You might damage the board, or electrostatic discharge might damage the board.

Use the VME chassis card guides to slide the scanner into the chassis. Use firm pressure on the top and bottom handles of the scanner to make its P1 connector fit firmly into the connector on the backplane. Tighten the screws on the top and bottom of the front panel to prevent the scanner from loosening.



Determining Power-Supply Requirements

The scanner operates on 5V dc @ 2.3A (typical), 2.5A (maximum).

Connecting to the Remote I/O Link

Each scanner channel supports as many as 32 physical adapters. Use 1770-CD (or Belden 9463) cable. Connect a remote I/O network using a daisy-chain or trunkline/dropline configuration.

Table 2.B
Total number of devices the scanner supports

In this mode:	the maximum number of logical racks per channel is:	and the maximum number of physical adapters per channel is:
SV-compatible	8	16
SV-superset	16	32

Important: The maximum cable length for remote I/O depends on the transmission rate. Configure all devices on a remote I/O link to communicate at the same transmission rate.

Table 2.C
Choose the correct cable length based on the link's communication rate

A remote I/O link using this communication rate:	cannot exceed this cable length:
57.6 kbps	3,048 m (10,000 ft)
115.2 kbps	1,524 m (5,000 ft)
230.4 kbps	762 m (2,500 ft)

For proper operation, terminate **both** ends of a remote I/O link by using external resistors. See Table 2.D for information on whether to use a 150Ω or 82Ω terminator.

Table 2.D
Terminating the remote I/O link

If your remote I/O link:	use this resistor rating:	the maximum number of *physical* devices that you can connect on the link is:	the maximum number of racks that you can scan on the link is:
operates at 230.4 K bit/s	82Ω	32	16
operates at 57.6 or 115.2 K bit/s, and **no** devices listed below are linked			
Scanners: 1771-SN; 1772-SD, -SD2; 1775-SR, -S4A, -S4B; 6008-SQH1, -SQH2			
Adapters: 1771-AS; 1771-ASB (series A only); 1771-DCM			
Miscellaneous: 1771-AF			
connects to any device listed below:	150Ω	16	16
Scanners: 1771-SN; 1772-SD, -SD2; 1775-SR, -S4A, -S4B; 6008-SQH1, -SQH2			
Adapters: 1771-AS; 1771-ASB (series A only); 1771-DCM			
Miscellaneous: 1771-AF			
operates at 57.6 or 115.2 K bit/s, and you do not require over 16 physical devices			

You can connect a remote I/O link in one of two ways:

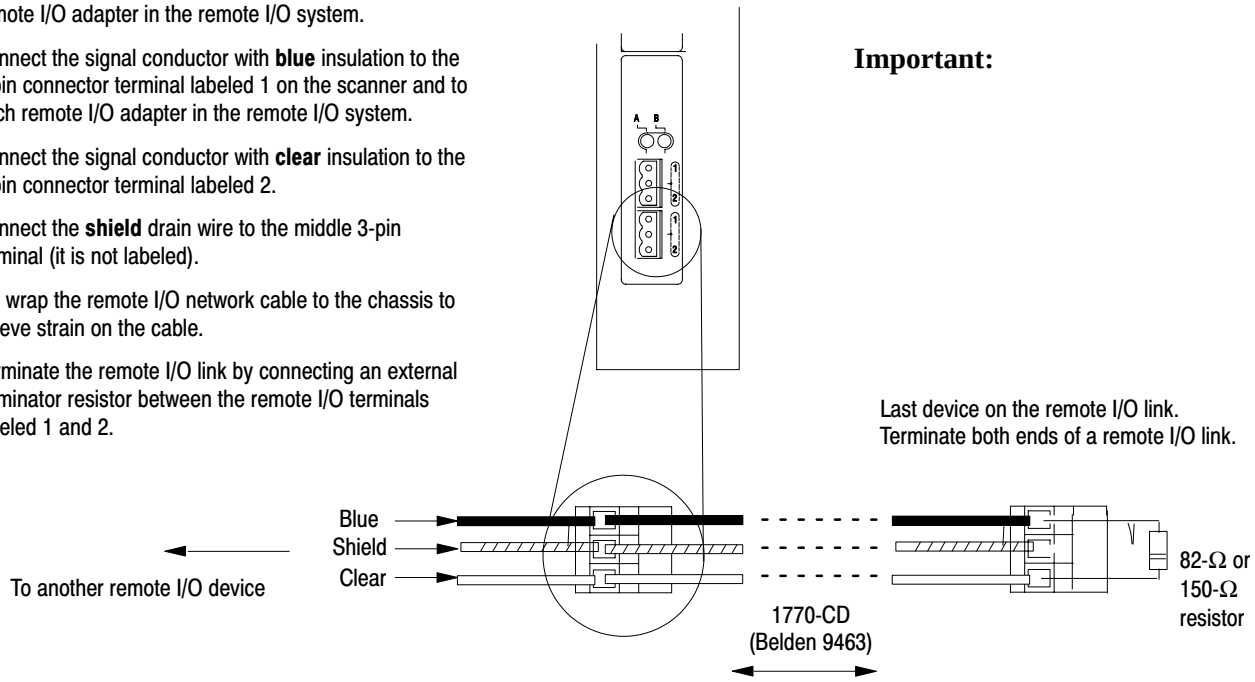
- trunkline/dropline—from the drop line to the connector screw terminals on the remote I/O connectors of the scanner
- daisy chain—to the connector screw terminals on the remote I/O connectors of the scanner and then to the remote I/O connector screw terminals of the next remote I/O device

Important The cable connections for the 6008-SV1R and 6008-SV2R scanner are opposite from those for the earlier 6008-SV scanner. Make sure you follow the instructions in Figure 2.1 below.

Figure 2.1
Make remote I/O connections

To connect the remote I/O cable, do the following:

1. Run the cable (1770-CD) from the processor to each remote I/O adapter in the remote I/O system.
2. Connect the signal conductor with **blue** insulation to the 3-pin connector terminal labeled 1 on the scanner and to each remote I/O adapter in the remote I/O system.
3. Connect the signal conductor with **clear** insulation to the 3-pin connector terminal labeled 2.
4. Connect the **shield** drain wire to the middle 3-pin terminal (it is not labeled).
5. Tie wrap the remote I/O network cable to the chassis to relieve strain on the cable.
6. Terminate the remote I/O link by connecting an external terminator resistor between the remote I/O terminals labeled 1 and 2.



Chapter 3

Addressing I/O

Using This Chapter

This chapter provides an overview of I/O addressing. This chapter also explains the basics of how the scanner processes discrete I/O and block-transfer data.

I/O Addressing Concept

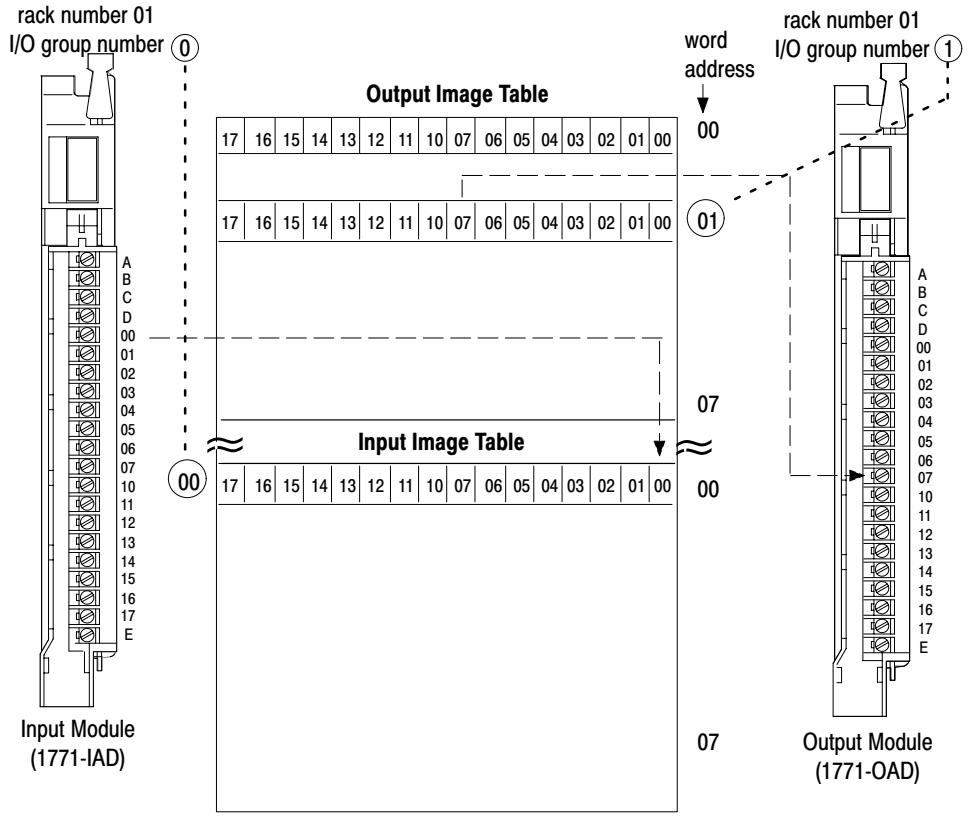
Each terminal on an input or output module that can be wired to a field device occupies a bit within the scanner's input image table or output image table.

I/O addressing maps the physical location of an I/O module terminal to a bit location in the processor memory. I/O addressing is just a way to segment memory:

Classification:	Term:	Relation to memory:
A specific terminal on an I/O module	terminal or point	The density of an I/O module, i.e., 8-point, 16-point, 32-point, directly relates to the amount of memory (bits) the module occupies in memory. For example, a 16-point input module occupies 16 bits in the input image table.
I/O terminals that when combined occupy 1 word in the input image table **and** 1 word in the output image table	I/O group	16 input bits = 1 **word** in the input image table 16 output bits = 1 **word** in the output image table
Combinations of bits or I/O groups	I/O rack	128 input bits and 128 output bits **or** 8 input words and 8 output words **or** 8 I/O groups

Figure 3.1 shows the relationship between an I/O terminal and its location in scanner memory.

Figure 3.1
I/O addressing as it relates to an I/O terminal



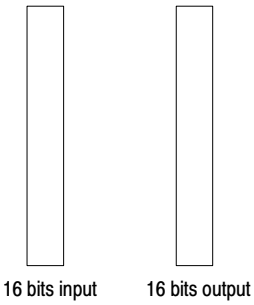
Important: The scanner addresses the image table with hexadecimal values. The addresses depend on the VME operating mode.

Choosing an Addressing Mode

For each chassis in your I/O system, you must define how many I/O chassis slots make up an I/O group (one word each in the input image table and output image table); this choice is the chassis' addressing mode. Choose from among these available modes:

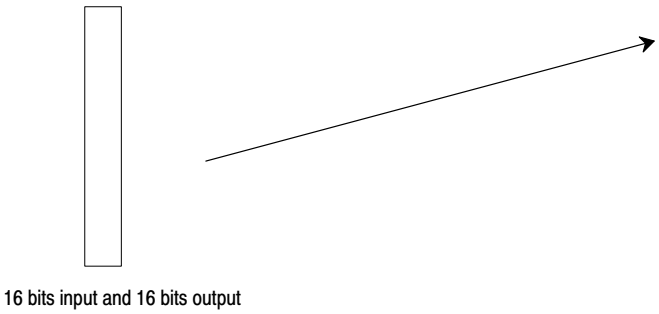
2-slot addressing

2 I/O chassis slots = 1 I/O group = 1 input image word and 1 output image word = 16 input bits and 16 output bits.



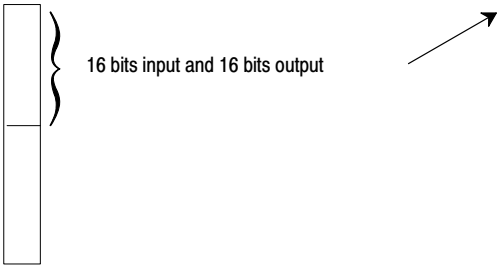
1-slot addressing

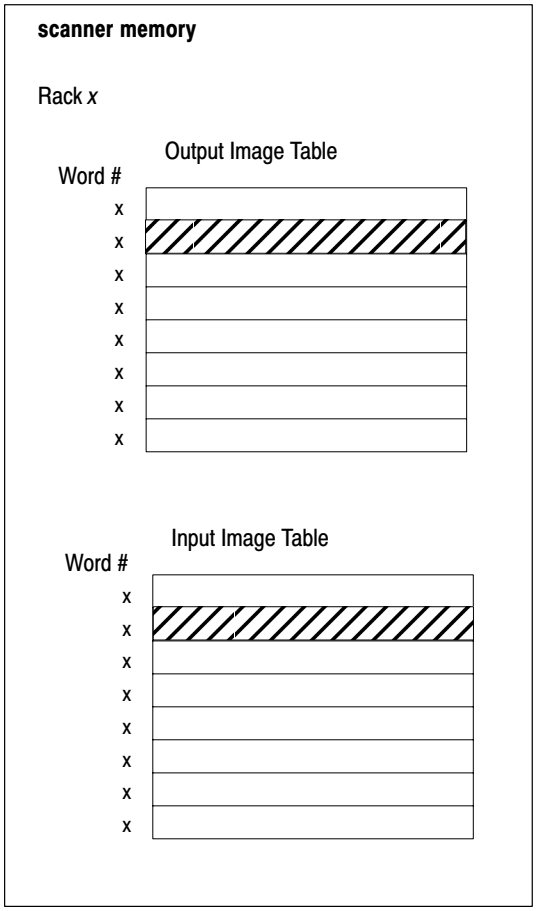
1 I/O chassis slot = 1 I/O group = 1 input image word and 1 output image word = 16 input bits and 16 output bits.



1/2-slot addressing

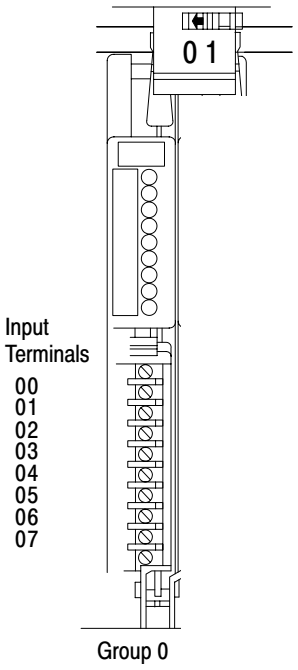
1/2 of an I/O chassis slot = 1 I/O group = 1 input image word and 1 output image word = 16 input bits and 16 output bits.



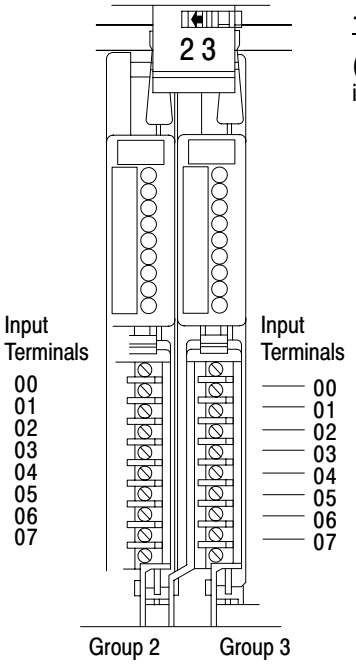


When you place your I/O modules in the I/O chassis slots, the module's density determines how quickly I/O groups form.

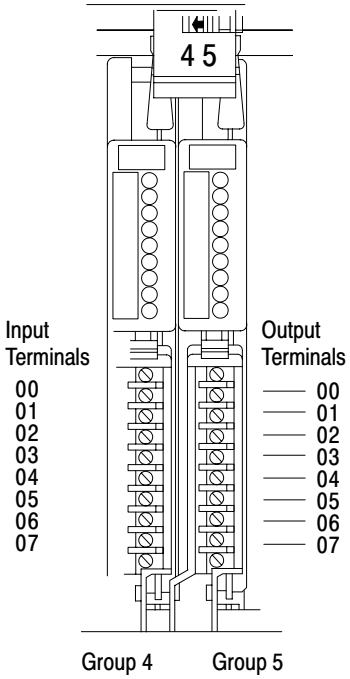
8- and 16-point examples



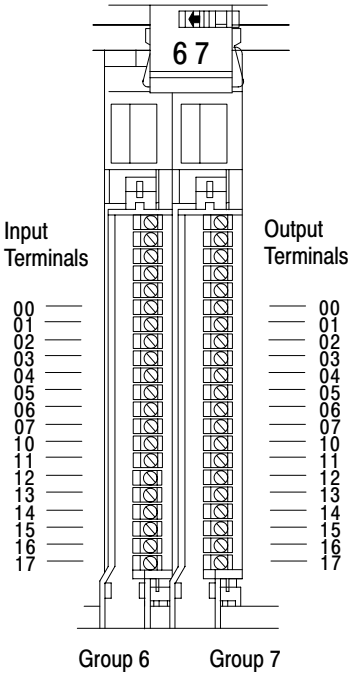
An 8-point I/O module occupies 8 bits in a word. See ①



Two 8-point input modules occupy 8 bits of each group. See ②



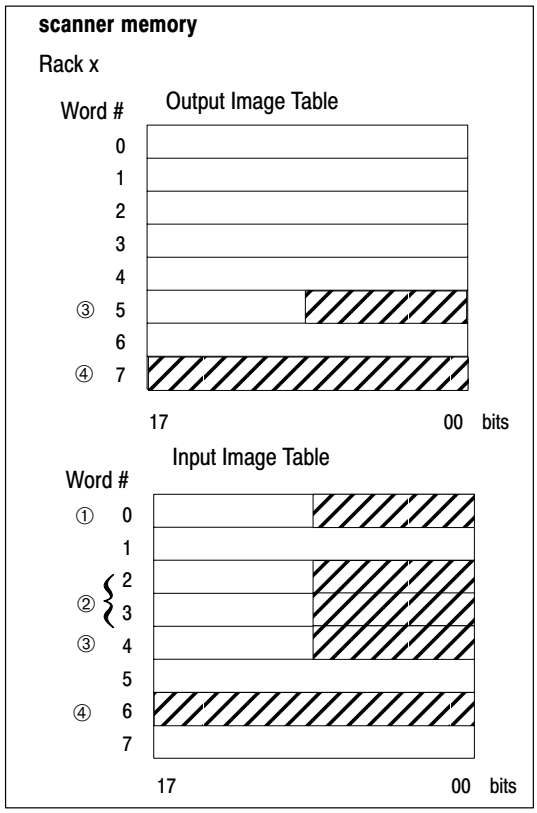
An 8-point input module in group 4 occupies the first 8 bits of input word 4. The 8-point output module occupies the first 8 output bits in output word 5. See ③



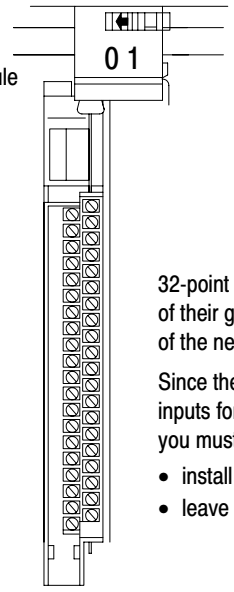
16-point I/O modules occupy 16 bits, an entire word, in the image table. See ④

1-slot addressing

(1 I/O chassis slot = 1 I/O group = 1 input image word and 1 output image word = 16 input bits and 16 output bits.)



32-point examples



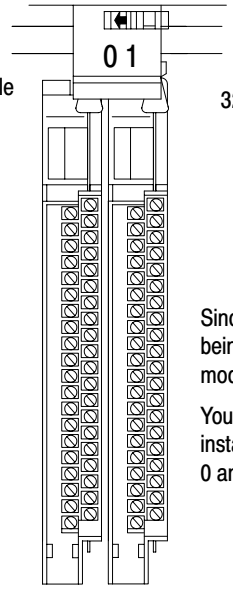
32-point input module

Group 0

32-point I/O modules use the entire word of their group and borrow the entire word of the next group. See ①.

Since the module is in group 0 and the inputs for group 0 and group 1 are used, you must:

- install an output module in group 1, or
- leave the slot empty



32-point input module

32-point output module

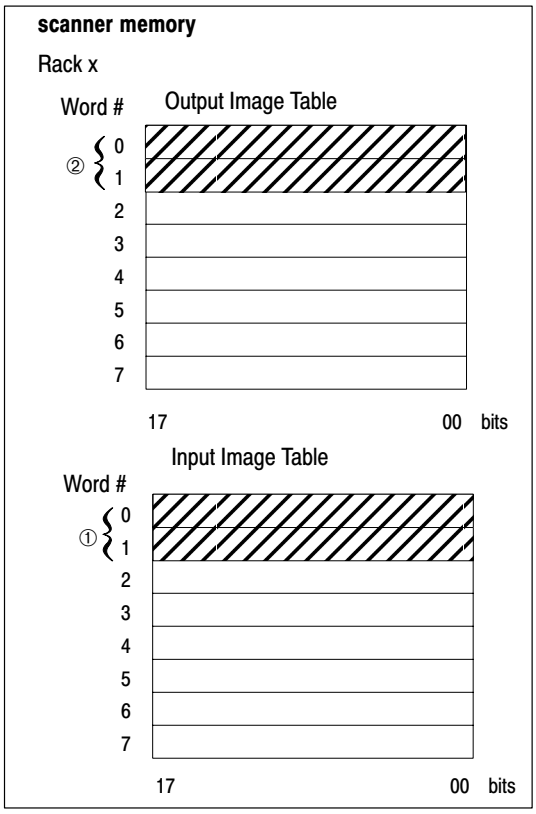
Group 0 Group 1

Since the input image table for group 1 is unavailable because it is being used by the input module of group 0, installing a 32-point output module makes use of output image table of group 0 and 1. See ②.

You can also install 8- or 16-point output modules. But you cannot install another input module since all the input image space for groups 0 and 1 is used by the input module of group 0.

1-slot addressing

(1 I/O chassis slot = 1 I/O group = 1 input image word and 1 output image word = 16 input bits and 16 output bits.)



When planning your system design, consider the densities of the I/O modules you are using and choose an addressing mode that most efficiently uses processor memory.

Choose the addressing mode for each I/O chassis by setting the chassis backplane switch assembly.

Addressing Summary

Addressing Mode:	Guidelines:
2-slot	• Two I/O module slots = 1 group • Each physical 2-slot I/O group corresponds to one word (16 bits) in the input image table and one word (16 bits) in the output image table • When you use 16-point I/O modules, you must install as a pair an input module and an output module in an I/O group; if you use an input module in slot 0, you must use an output module in slot 1 (or it must be empty). This configuration gives you the maximum use of I/O. • You cannot use a block-transfer module and a 16-point module in the same I/O group because block-transfer modules use 8 bits in both the input and output table. Therefore, 8 bits of the 16-point module would conflict with the block-transfer module. • You cannot use 32-point I/O modules.
1-slot	• One I/O module slot = 1 group • Each physical slot in the chassis corresponds to one word (16 bits) in the input image table and one word (16 bits) in the output image table • When you use 32-point I/O modules, you must install as a pair an input module and an output module in an even/odd pair of adjacent I/O group; if you use an input module in slot 0, you must use an output module in slot 1 (or it must be empty). This configuration gives you the maximum use of I/O. • Use any mix of 8- and 16-point I/O modules, block-transfer or intelligent modules in a single I/O chassis. Using 8-point modules results in fewer total I/O.
1/2-slot	• One half of an I/O module slot = 1 group • Each physical slot in the chassis corresponds to two words (32 bits) in the input image table and two words (32 bits) in the output image table • Use any mix of 8-, 16-, and 32-point I/O or block-transfer and intelligent modules. Using 8-point and 16-point I/O modules results in fewer total I/O.

Addressing Block-Transfer Modules

Block-transfer modules occupy 8 bits in the I/O image table. Since all block-transfer modules are bidirectional, they cannot be used to complement either input or output modules.

To address:	use the:
single-slot modules	assigned I/O rack and group number of the slot in which the module resides and 0 for the module number When using 1/2-slot addressing, use the assigned rack number and the **lowest** group number and 0 for the module number.
double-slot modules	assigned rack number and the **lowest** group number and 0 for the module number

Assigning Racks

The number of racks in a chassis depends on the chassis size and the addressing mode.

If using this chassis size:	2-slot addressing, rack type is:	1-slot addressing, rack type is:	1/2-slot addressing, rack type is:
4-slot	1/4 rack	1/2 rack	1 rack
8-slot	1/2 rack	1 rack	2 racks
12-slot	3/4 rack	1-1/2 racks	3 racks
16-slot	1 rack	2 racks	4 racks

Design Tip

One I/O rack number is eight I/O groups, regardless of the addressing mode that you select.

When assigning remote I/O rack numbers, use the following guidelines:

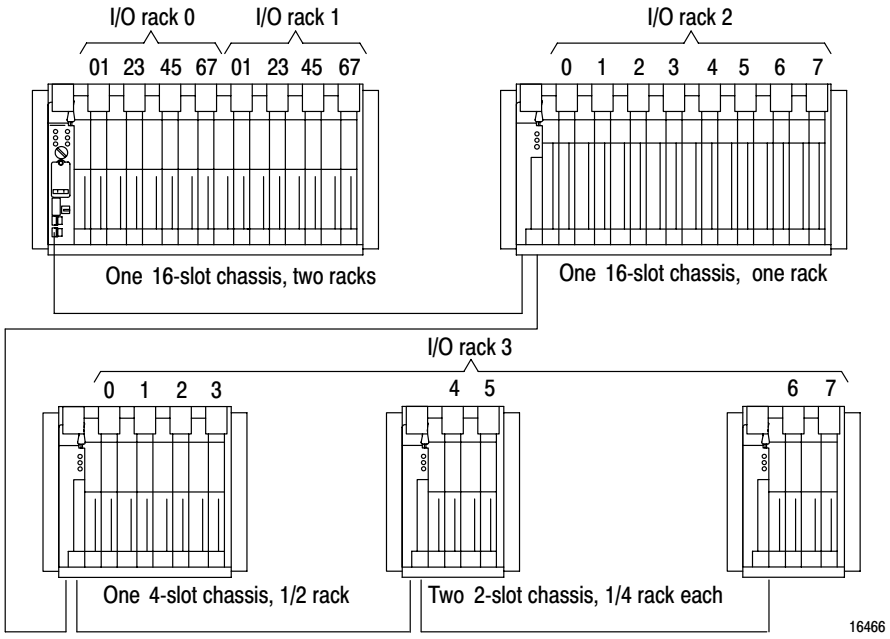
- Each scanner channel supports as many as 16 physical I/O adapters.
- The number of racks the scanner supports depends on the VME operating mode.

This operating mode:	supports as many as:	with a total maximum number of adapters:
SV-compatible	8 full racks	16
SV-superset	16 full racks	32
SV-adapter	1 rack	0

For example, you can configure 8 racks as:

8 full racks of 128 I/0 each (8 adapters)
or
16 half racks of 64 I/O each (16 adapters)
or
6 full racks of 128 I/O each and 8 quarter racks of 32 I/O each (14 adapters)

- You can assign a remote I/O rack to a fraction of a chassis, a single I/O chassis, or multiple I/O chassis:



Chapter 4

Communicating with Remote I/O

Using This Chapter

This chapter provides an overview of remote I/O communication.

If you want to read about:	go to page:
selecting devices that you can connect	4-1
introduction to remote I/O	4-2
designing a remote I/O link	4-3
specifying a scan list	4-5
processing discrete data	4-6
processing block data	4-8

Selecting Devices that You Can Connect

These are the devices you can use on a remote I/O link with the scanner.

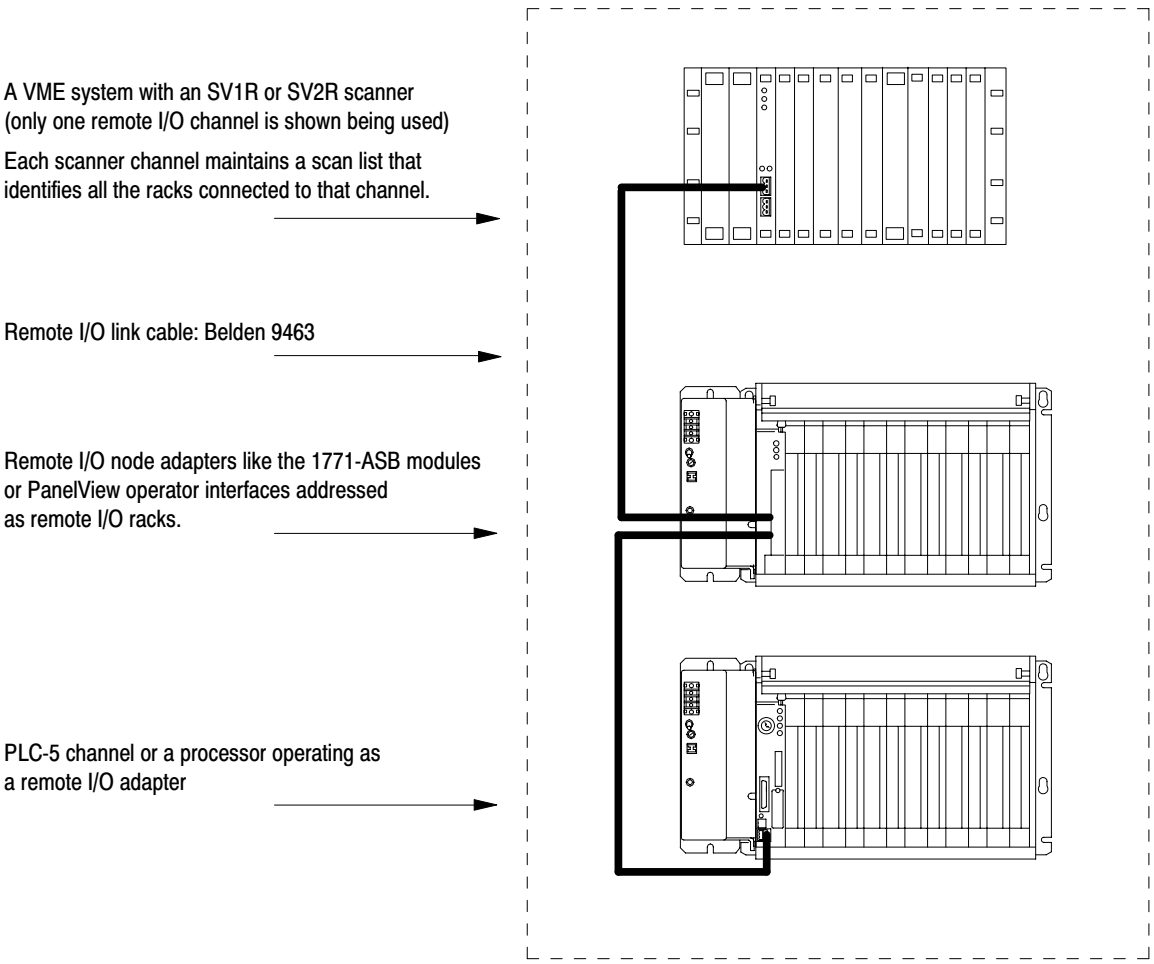
Category:	Product:	Catalog Number:
PLC processors (in adapter mode)	classic PLC-5 processors	1785-LT 1785-LT2 1785-LT3
	enhanced PLC-5 processors	1785-L11B 1785-L20B 1785-L30B 1785-L40B 1785-L60B 1785-L80B
	ethernet PLC-5 processors	1785-L20E 1785-L40E 1785-L80E
	local PLC-5 processors	1785-L40L 1785-L60L
	VME PLC-5 processors	1785-V30B 1785-V40B 1785-V40L
	Direct Communication Module for SLC Processors	1747-DCM
to remote I/O	SLC 500 Remote I/O Adapter Module	1747-ASB
	1791 Block I/O	1791 series
	FLEX I/O	1794 series
	Remote I/O Adapter Module	1771-ASB
	1-slot I/O Chassis with Integral Power Supply and Adapter	1771-AM1
	2-slot I/O Chassis with Integral Power Supply and Adapter	1771-AM2
	Direct Communication Module	1771-DCM
operator interfaces	DL40 Dataliner	2706 series
	RediPANEL	2705 series
	PanelView Terminal	2711 series
drives	Remote I/O Adapter for 1336 AC Industrial Drives	1336-RIO
	Remote I/O Adapter for 1395 AC Industrial Drives	1395-NA
third party devices	any devices incorporating A-B node adapter chip sets under the ENABLED Technology program	

Introduction to Remote I/O

A remote I/O system lets you control Allen-Bradley I/O that is not within a VME master processor's chassis. The SV1R has one remote I/O channel; the SV2R has two remote I/O channels. Each channel transfers discrete and block-transfer data with remote I/O devices.

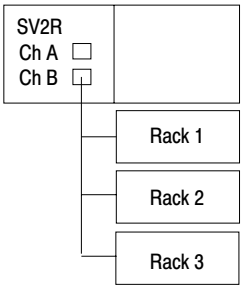
These components make up a remote I/O system:

Figure 4.1
Components of a remote I/O system



The scanner keeps a list of all of the devices connected to each channel. Figure 4.2 shows an example scan list for one channel.

Figure 4.2
Example scan list



Ch B Scan List

Rack Address	Starting Group	Rack Size	Range
1	0	Full	010-017
2	0	1/2	020-023
3	0	Full	030-037

In this example, channel B continually scans the three racks in its scan list.

The steps for setting up a remote I/O system are:

Step:		See:
1.	configure the remote I/O adapter devices	the device's user manual
2.	layout and connect the remote I/O link cable	• page 4-3 for design • your adapter's installation information
3.	specify a scan list	page 4-5

Designing a Remote I/O Link

Designing a remote I/O link requires applying:

- remote I/O link design guidelines
- cable design guidelines

Design Tip

Link Design Guidelines

Keep these rules in mind as you design remote I/O links:

- All devices connected to a remote I/O link must communicate using the same communication rate. The rate you choose depends on the VME operating mode:

This VME operating mode:	supports these communication rate:
SV compatible	57.6 kbps
	115.2 kbps
SV superset	57.6 kbps
	115.2 kbps
	230.4 kbps

- Two channels on the same scanner cannot scan the same partial or full rack address. Assign unique partial and full racks to each channel.
- You can split rack addresses between scanner channels; however, issues arise when performing block-data transfer. If you split remote rack addresses between scanner channels, channel A has priority over channel B.
- A scan list can have a maximum of 16 rack numbers or a maximum of 64 physical devices connected to it using 82-Ω termination resistors.

Design Tip

Cable Design Guidelines

Specify 1770-CD (Belden 9463) cable. Connect a remote I/O network using a daisy chain or trunk line/drop line configuration.

Important: The maximum cable length for remote I/O depends on the transmission rate. Configure all devices on a remote I/O link to communicate at the same transmission rate.

For trunk line/drop line configurations, use the 1770-SC station connectors and follow these guidelines:

- the length of the trunk line cable depends on the communication rate
- the length of the drop cable is 30.4 m (100 cable-ft)

For more information about designing trunk line/drop line configurations, see the Data Highway/Data Highway Plus/Data Highway II/Data Highway-485 Cable Installation Manual, publication 1770-6.2.2.

For daisy chain configurations, determine the total cable length you need.

A remote I/O link using this communication rate:	cannot exceed this cable length:
57.6 kbps	3,048 m (10,000 ft)
115.2 kbps	1,524 m (5,000 ft)
230.4 kbps	762 m (2,500 ft)

For proper operation, terminate both ends of a remote I/O link by using external resistors. Use either a 150Ω or 82Ω terminator.

Table 4.A
Selecting the external resistor

If your remote I/O link:	use this resistor rating:	the maximum number of *physical* devices that you can connect on the link is:	the maximum number of racks that you can scan on the link is:
operates at 230.4 K bit/s	82Ω	32	16
operates at 57.6 or 115.2 K bit/s, and **no** devices listed below are linked Scanners: 1771-SN; 1772-SD, -SD2; 1775-SR, -S4A, -S4B; 6008-SQH1, -SQH2 Adapters: 1771-AS; 1771-ASB (series A only); 1771-DCM Miscellaneous: 1771-AF			
connects to any device listed below: Scanners: 1771-SN; 1772-SD, -SD2; 1775-SR, -S4A, -S4B; 6008-SQH1, -SQH2 Adapters: 1771-AS; 1771-ASB (series A only); 1771-DCM Miscellaneous: 1771-AF	150Ω	16	16
operates at 57.6 or 115.2 K bit/s, and you do not require over 16 physical devices			

Specifying a Scan List

The scan list is a map of the I/O devices the scanner channel scans. To create the scan list, use either of these commands:

If you want:	use this command:
the scanner to poll all available adapter addresses and assemble a list with one entry for each adapter in SV-compatible mode, there will be a maximum of 16 entries in the scan list; in SV-superset mode , there will be a maximum of 32 entries in the scan list	AUTOCONFIGURE
to create the scan list manually add adapters multiple times in the scan list there can be a maximum of 64 entries in the scan list, as long as there are only 16 distinct physical adapters	SCAN LIST

Design Tip

If you need multiple updates to an I/O device during an I/O scan, enter a logical address in the scan list more than one time. Do not assign the same partial or full rack address to more than one channel in scanner mode. Each channel must scan unique partial and/or full rack addresses.

The automatic configuration always displays the actual hardware configuration, except for racks that have their global-rack inhibit bit set. In this case, the global-rack bit overrides the automatic configuration.

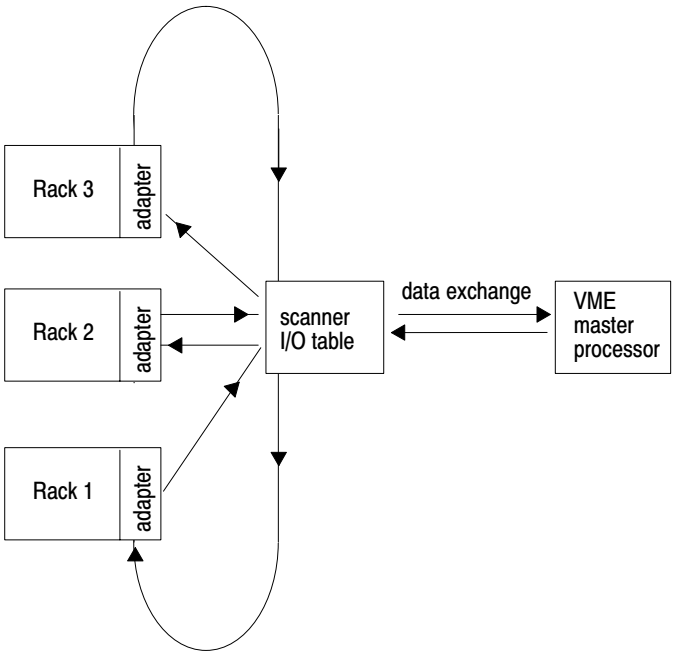
Processing Discrete I/O

Discrete I/O devices include such external devices as:

- switches
- relay contacts
- indicator lights
- control relays
- motor starters

A scanner channel exchanges discrete data (digital and analog) with remote I/O node adapters like 1771-ASB modules via the scanner I/O image table (Figure 4.3).

Figure 4.3
Remote I/O scan

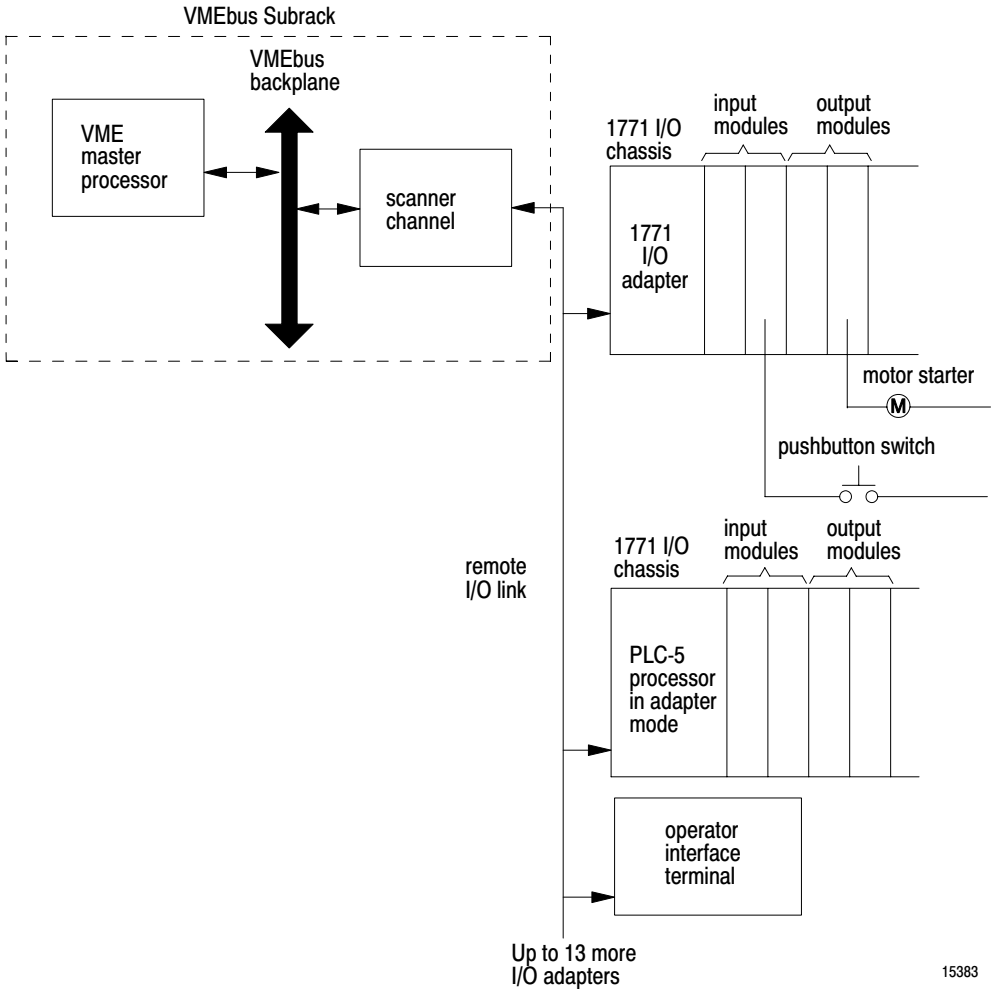


The remote I/O scan is the time it takes for the scanner to communicate with all of the entries in its scan-list once.

Important: The remote I/O scan for one channel is independent of and asynchronous to the remote I/O scan for the other channel.

The status of the discrete devices is represented by a single bit in an input or output group (word). The scanner is related to these devices as shown in Figure 4.4.

Figure 4.4
The Relationship between the scanner and I/O



An external device, such as the switch shown above, causes an input of a discrete input module to turn on. This sets a bit to a “1” condition in the I/O module. The next time the I/O adapter scans the module, the new information is reported to the adapter. When the scanner scans the adapter, the corresponding bit in the I/O image table is set to a condition that reflects the status of the external device, the switch.

A VME master processor then reads the input image information from the global RAM via the VMEbus.

To generate an output, a VME master processor sets a specific bit in the I/O image table in the scanner's global RAM, corresponding to the desired output device, for example a control relay.

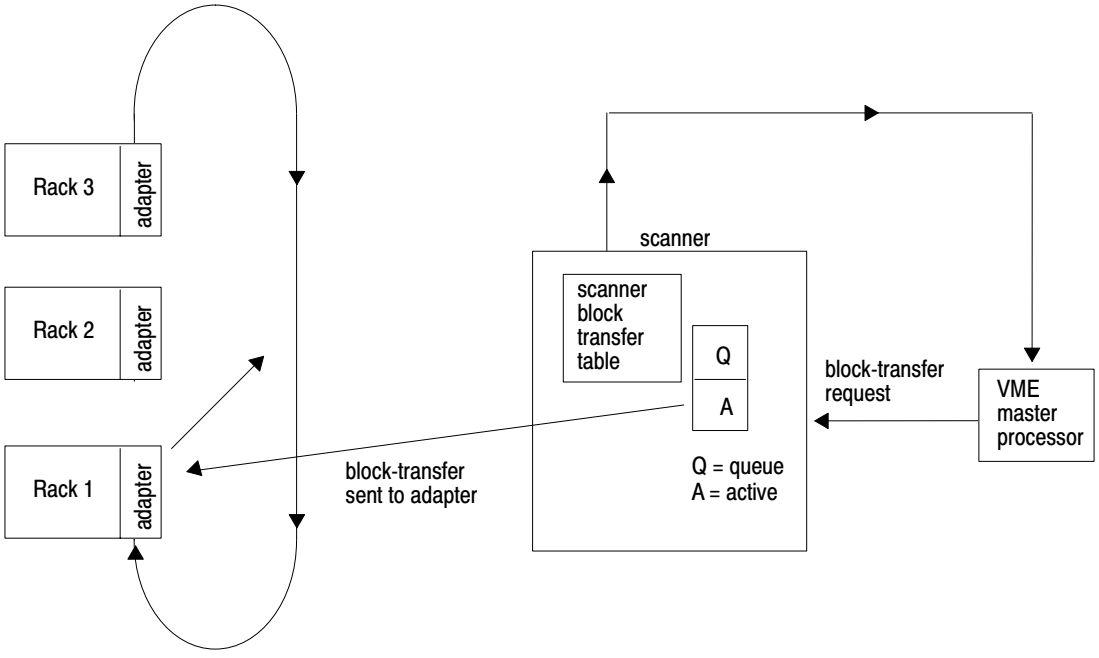
The scanner updates the adapter during the next scan cycle. The adapter sets the appropriate bit in the output module and the discrete output device energizes the control relay.

Processing Block Data

In addition to discrete data, the scanner can also exchange block data with remote I/O. Block transfers are the communication of data blocks (files) between the scanner and intelligent I/O modules. These are any I/O modules that respond to read or write block transfers. These modules handle data such as analog input, analog output, positioning, and communication data.

Block-transfer instructs the scanner to transfer as many as 64 words of data to/from a selected I/O module. The operating mode of the scanner determines how many block-transfer requests the scanner can queue as many as 42 requests when operating in SV-compatible mode; only 1 request when operating in SV-superset mode. Figure 4.5 shows how the scanner handles a block-transfer.

Figure 4.5
Block-transferring data to remote I/O



The scanner processes block-transfers differently depending on the VME operating mode. In general though, block-transfer allows the VME master processor to read or write up to 64 16-bit words from an adapter.

In SV-compatible mode
The scanner processes single block-transfer operations. You must program a read or write command for each data transfer in your application. The scanner can queue as many as 42 block-transfer requests from the VME master processor.

In SV-superset mode
The scanner processes single block-transfer operations and continuous block-transfer operations. The single block-transfer operations work the same as in SV-compatible mode. The only difference is that the scanner can only queue one request from the VME master processor. Use continuous block-transfer requests for applications that need to continuously poll an adapter. The continuous block-transfer requests uses less overhead than programming a single block-transfer request each time you need the data.

For specifics on block-transfers based on VME operating mode, see chapter 5 or 6 for SV-compatible or SV-superset mode, respectively.

Understanding the Block-Transfer Sequence

The VME master processor and scanner alternately access the scanner's global RAM while transferring a block transfer. See Figure 4.6 and Figure 4.7.

Figure 4.6
Block-transfer sequence – sending a request

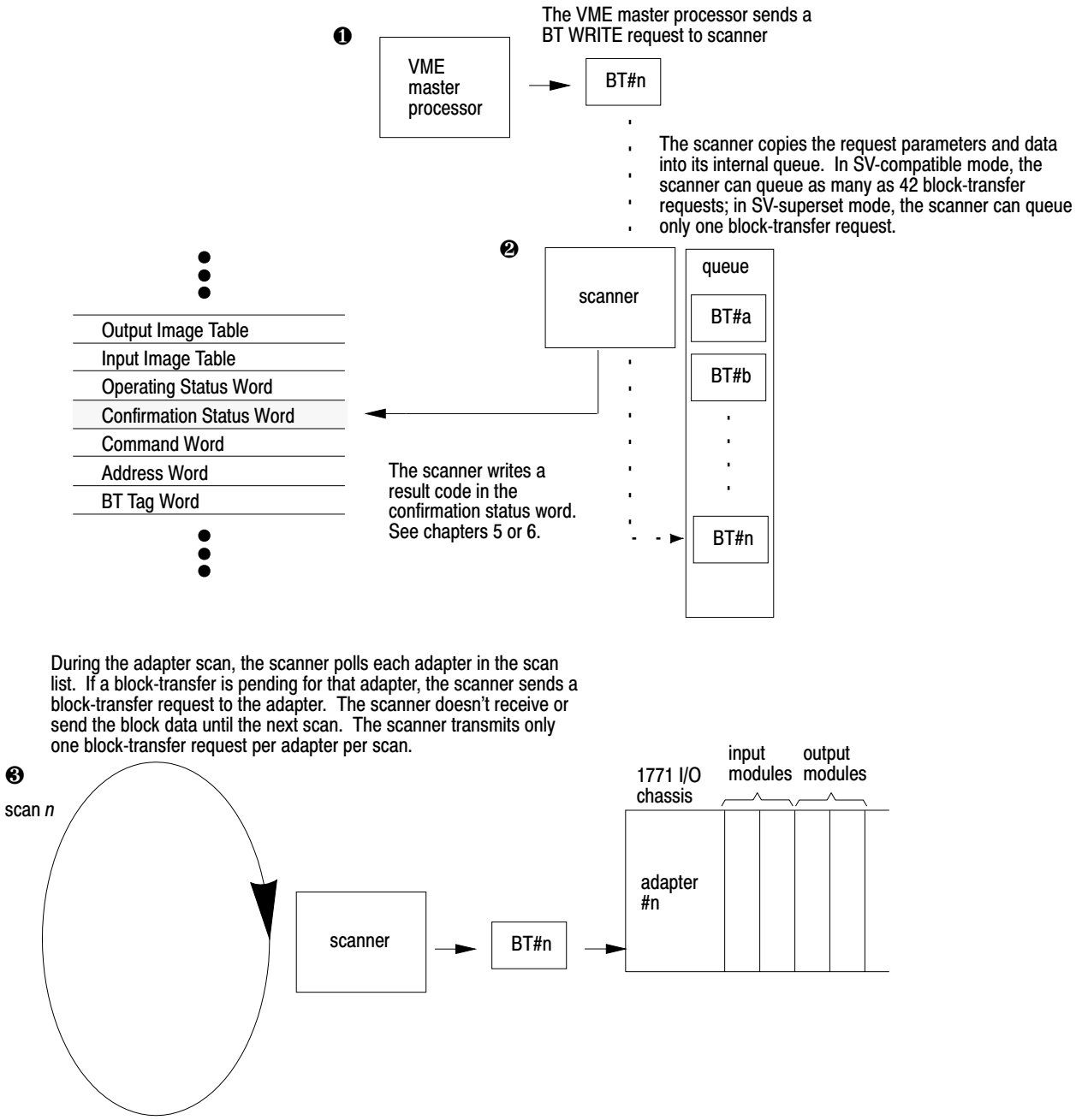
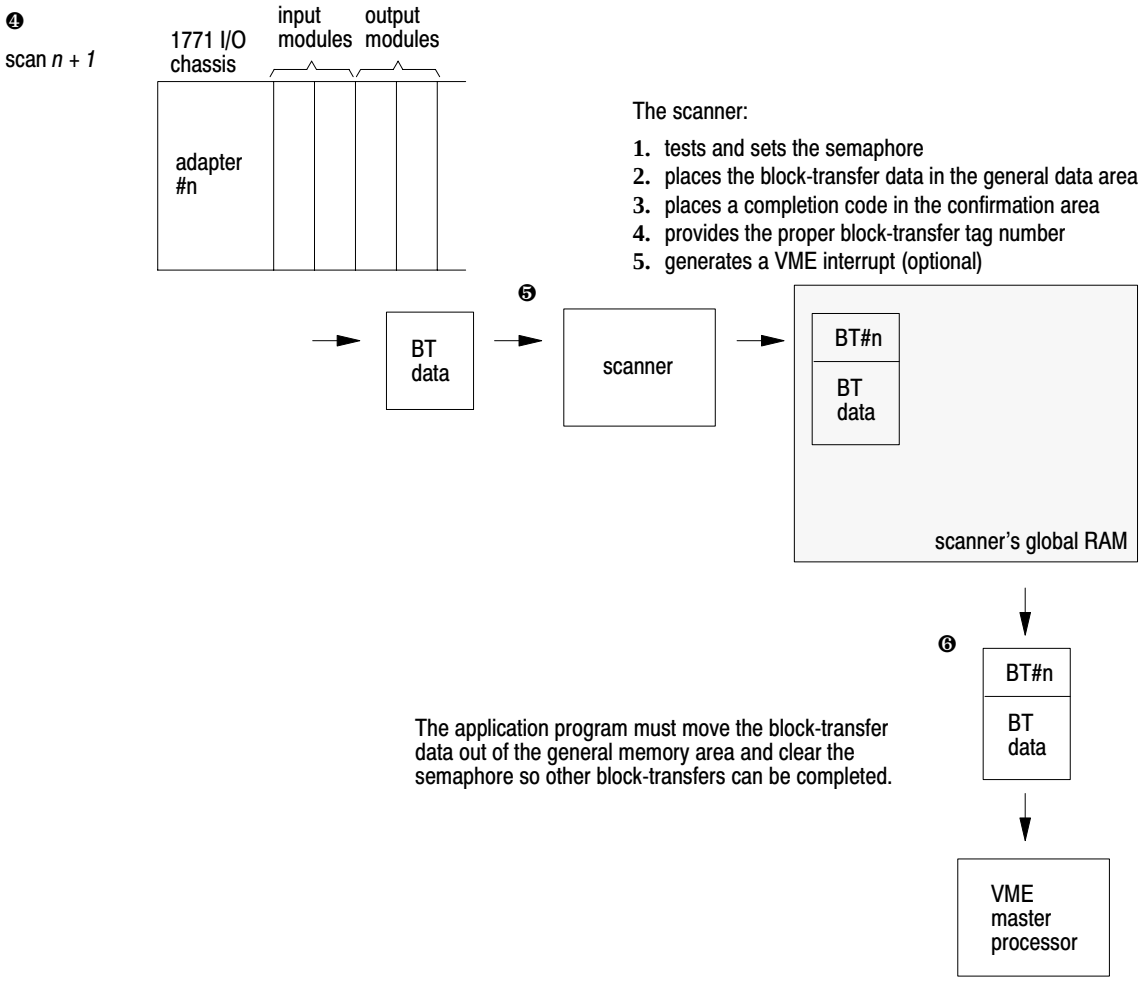


Figure 4.7
Block-transfer sequence – receiving data from a completed transfer

On the previous adapter scan, the scanner sent a block transfer request. During this scan, the adapter returns the response data to the scanner.



Sending Multiple Block-Transfers to the Same Adapter

The scanner can send as many block-transfer requests as it needs to an adapter (which corresponds to 1 physical rack), but it can only send one block-transfer request to a rack per scan. To achieve the most efficient block-transfers, arrange your intelligent I/O modules in multiple racks. Another method to achieve faster block-transfers is to use the SCAN LIST command so that a rack with several intelligent I/O modules is scanned more often through multiple listings in the scan list.

The scanner keeps track of each block-transfer request by its unique tag number. You assign the tag number when you configure the block-transfer command. When a block-transfer command completes, the scanner returns the tag number along with the data so your application can determine which block-transfer completed.

In SV-compatible mode
Each scanner channel can queue as many as 42 block-transfer requests. Each request has a block-transfer tag number that you assign in your application program. When a specific block-transfer is complete, the scanner sends the block-transfer tag number back to the VME master processor, along with the data. This way, your application program will know which block-transfer completed, since block-transfers do not necessarily get completed in the same order they are requested.

In SV-superset mode
The scanner still processes a block-transfer request the same way as in SV-compatible mode, but each scanner channel can queue only 1 block-transfer request. If you need multiple block-transfers to the same adapter, use the continuous block-transfer operation. It achieves the same result as sending several single block-transfer requests and it also reduces programming overhead.

Before initializing a continuous block-transfer requeues to an I/O modules, try a single block-transfer to the I/O module to verify the block-transfer parameters. After verifying the single block-transfer, use those parameters in a continuous block-transfer request.

Maintaining Block Integrity

To maintain block integrity within the global RAM, use bit 7 of the semaphore byte. The semaphore byte is in the control/status area of global RAM for each channel. See chapters 5–6, depending on the operating mode you select for the scanner.

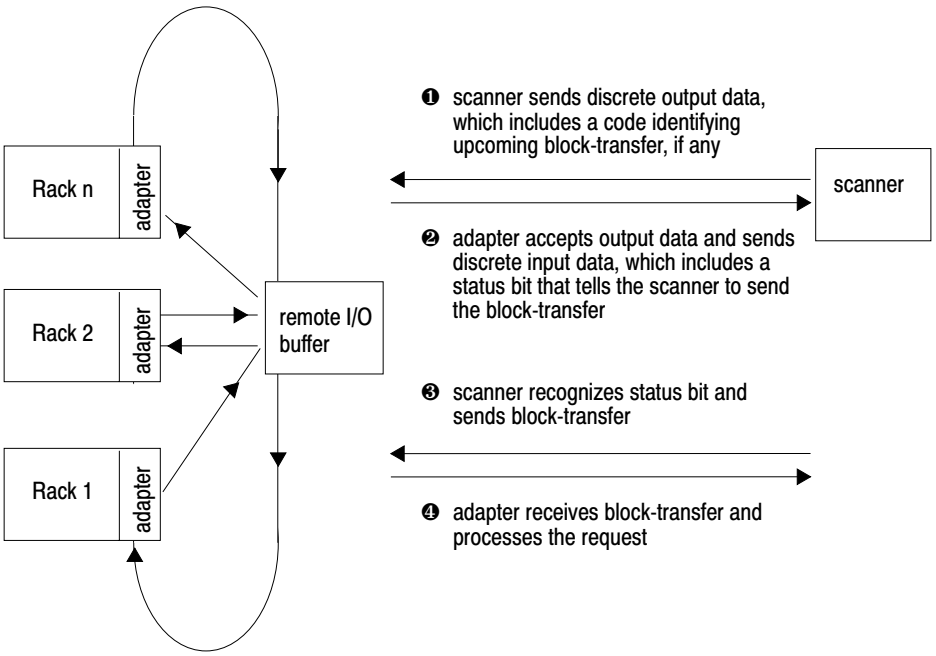
At power up, when the scanner is turned on, the semaphore bit is 0 (reset). When the scanner completes a block-transfer, it tests and sets the semaphore. If the semaphore was reset, the scanner loads status (either confirmation or error) into the confirmation status word and optionally interrupts the VME master processor to notify it of pending status. If the semaphore was set, the scanner holds the block-transfer information until the semaphore is reset by a VME master processor.

After receiving the status, the VME master processor must copy the data from the global RAM and reset the semaphore.

Processing One Block Transfer Request

The order in which the scanner sends block-transfer requests to adapters depends on the scanner's scan list. The scan list tells the scanner when to poll an adapter; an adapter can be in a scan list multiple times.

Figure 4.8
Adapter scan sequence



Important: Do not accidentally write data in the output image table in global RAM of the scanner. Any data in an output byte for an intelligent I/O module overrides the scanner block-transfer command and ruins any block-transfer request. Such an error is detected as a bit being set in the operating status word identifying an unsolicited block-transfer.

When the scanner is done with the block-transfer, it tests and sets the semaphore, places the necessary data in the general data area, places a completion code in the confirmation status byte, provides the block-transfer tag number, and generates a VMEbus interrupt. If the scanner sees that the semaphore is already set, it holds the block transfer data until the semaphore is reset.

Your application should copy the block-transfer data from the general data area and clear the semaphore.

Block-Transfer Timeout

If the block-transfer request does not complete within 4 seconds of the VME master processor sending the request to the scanner, the scanner:

1. tests and sets the semaphore
2. dequeues the failed request
3. places an error code in the confirmation status word
4. copies the correct tag value
5. optionally interrupts the VME master processor to indicate that the block-transfer requests timed out.

The VME master processor then retrieves the result code and clears the semaphore.

The most common failure of a block-transfer is a block timeout, error code 23 (hex) in the confirmation status word. This error occurs if the block-transfer does not complete within 4 seconds of the initial request. This 4-second value is not variable.

Another frequent reason a block-transfer times out is because an incorrect intelligent module address is given to the scanner. Another common mistake is to put an incorrect value for the length of data words. The value for the length of data words must equal the number of words the I/O module expects to transfer. If these two values are not the same, the transfer will never happen.

The best way to avoid a block-transfer timeout is to give the scanner a transfer length of zero. This lets the I/O module decide how many words to send. When the transfer completes, the I/O modules replaces the 0 in the length of data words with the actual number of how many words were transferred.

Chapter 5

Operating in SV-Compatible Mode

Using This Chapter

The SV-compatible mode supports these features:

- each channel supports as many as 8 logical racks (16 physical adapters)
- the global data area is 1872 words long
- using the SETUP command, you can enable or disable the VME-master-processor watchdog timer; change the timeout interval of the watchdog timer; select the command complete interrupt level and status ID; select whether to monitor SYSFAIL conditions; and select a communication rate of 57.6 or 115.2 kbps

Use this mode if you are replacing a 6008-SV scanner and want the 6008-SV1R or 6008SV2R scanner to operate exactly as the scanner you are replacing. Your programs for the 6008-SV scanner work for the 6008-SV1R or 6008-SV2R scanner. However, you need to change any reference in your application to the VME ID for the 6008-SV scanner to the VME ID for the 6008-SV1R or 6008-SV2R scanner.

Important: The SCAN LIST command uses two bits to identify rack size. These bits were not required for the 6008-SV. If you are migrating your application from a 6008-SV to a 6008-SV1R or 6008-SV2R, make sure to set these additional bits. See page 5-13.

This chapter describes the SV-compatible mode.

Addressing Global RAM

Both the VME master processor and the scanner can read and write to the scanner's VME global RAM. The global RAM structure depends on the VME operating mode of the scanner. Figure 5.1 shows the general structure for the SV-compatible operating mode. Page 5-3 shows the specifics of the structure for channel A.

Figure 5.1
General structure of global RAM for SV-compatible mode

Channel A

byte offset (hex)	
000	**output image table** 64 words
080	**input image table** 64 words
100	**control/status area** 16 words
120	**general data area** 1872 words
FC0	**interrupt/VME ID area** 32 words

Channel B
(6008-SV2R scanner only)

byte offset (hex)	
1000	**output image table** 64 words
1080	**input image table** 64 words
1100	**control/status area** 16 words
1120	**general data area** 1872 words
1FC0	**interrupt/VME ID area** 32 words

The physical address is the base address plus the byte offset.

Important: Add 1000 (hex) to channel A addresses to get the corresponding addresses for channel B.

global RAM structure for SV-compatible mode
Channel A

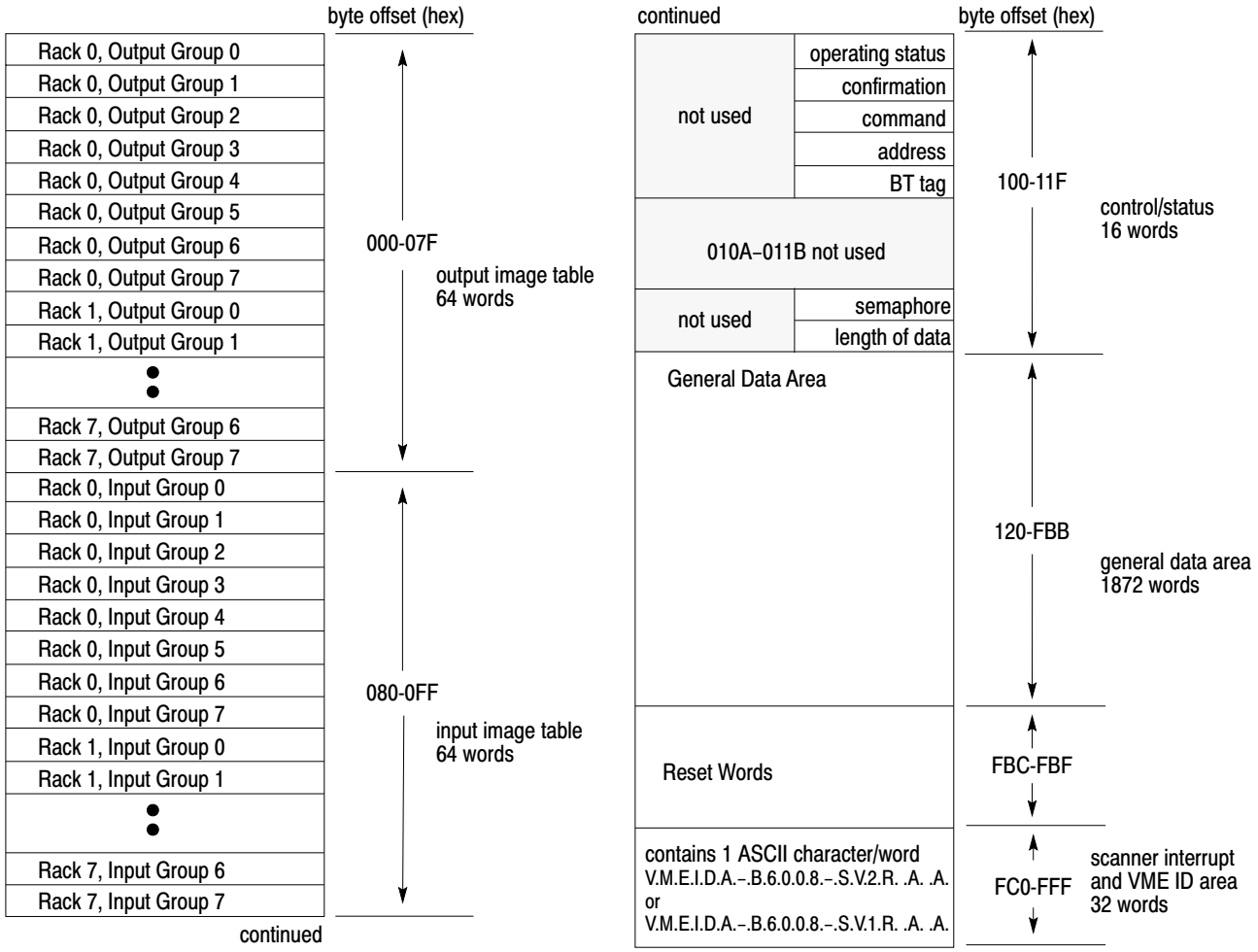


Table 5.A describes the components of global RAM.

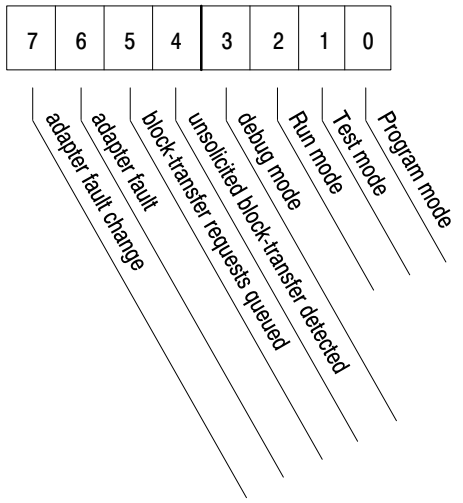
Table 5.A
Descriptions of the global memory areas

This area:	stores the:	
input and output image tables	input and output data for as many as 8 logical I/O racks with no more than 16 adapters.	
control/status area	operating status byte	reflects the current status of the scanner (Figure 5.2) lets the VME master processor poll the scanner's status without interrupting ongoing operations
	confirmation status byte	contains the result of the executed command A result of 00 (hex) indicates the command completed successfully. See chapter 9 for a list of error codes.
	command byte	identifies the command the VME master processor wants the scanner to execute next
	address byte	contains I/O rack, group, and slot parameters for block transfer commands For more information, see BT READ command on page 5-25 or BT WRITE command on page 5-27.
	block-transfer tag byte	provided by the VME master processor to uniquely identify each block-transfer request
	semaphore byte	provides for the integrity of command requests, command responses, and completed block transfers If bit 7 is set, a VME master processor is using the control/status and general data area or the scanner just completed a block-transfer and set the semaphore so no other VME master processor will overwrite the data. If the semaphore is set because of a block-transfer, the appropriate VME master will know to come and retrieve the data and reset the semaphore. If the semaphore bit is clear, the general data area is available for access by any VME master processor. **Important:** A VME master processor doesn't use the semaphore when it accesses the I/O image table. The VME master processor can access the I/O image table anytime. For command requests, the scanner returns status, either confirmation or error, when the command is complete. When the VME master processor receives the confirmation status, it must retrieve the data from the general data area and clear the semaphore. **Important:** Only 1 command can be issued at a time. A status confirmation or error must be received before a new command is issued.
	length of data byte	specifies the amount of data associated with a command or response Only the lower byte is used. Interpret the length of data based on the context of the requested command or received confirmation (i.e., number of words for block-transfers and bytes for scanner management requests).
general data area	contains input parameters and result data for scanner commands	
scanner interrupt and VME ID	interrupt from the VME master processor to the scanner The scanner interrupt area and VME ID area can be read anytime without causing an interrupt to the scanner. It contains 32 words in which the odd (lower byte) only is used; the information is an ASCII character string: V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.2.R. .x. .y. **or** V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.1.R. .x. .y. where .x. .y. are the series and revision levels of the scanner. For example: V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.2.R. .A. .A. for series A, revision A. Writing to any byte in this area interrupts the scanner. The scanner then looks at the command byte to determine which command to execute. All even bytes have the value 0xFF; all odd bytes without ASCII characters have the value 0x00.	

Operating Status Byte

The operating status byte holds scanner status information for the VME master processor. The scanner updates this byte every time it completes either a block-transfer or a command. The master processor can clear bits 4 and 7; the remaining bits are read only. The structure is as follows:

Figure 5.2
Structure of the operating status byte



These bits:	Considerations:	These bits:	Considerations:
bits 0–2	These bits specify the programming mode of the scanner. These bits are mutually exclusive – the scanner can be in only one of these modes.	bits 5	If bit 5 is set, the scanner has at least one block-transfer request in its internal queue. When the queue is empty, the scanner clears the bit.
bit 3	If the debug mode bit is set by the scanner, the scanner cannot be shutdown by the internal watchdog. **ATTENTION:** Unwanted machine motion can result from disabling the VME master processor watchdog. Only use the debug mode when you are debugging the application program for the VME master processor.	bit 6	If bit 6 is set, at least one adapter has faulted, lost power, or has been dropped from the I/O link. The VME master processor can issue the LINK STATUS command for fault details. When all the adapters in the scan list are cleared of faults or have been brought back on line, the scanner will clear this bit.
bit 4	The scanner sets bit 4 when it detects an unsolicited block-transfer. An unsolicited block-transfer results if a VME master processor accidentally writes discrete information to an output image table byte that is mapped to an I/O slot requiring a block transfer. When this bit is set, take action to correct this situation because it can seriously degrade scanner response time. The scanner can only set this bit; the VME master processor must clear it.	bit 7	If bit 7 is set, an adapter's operating status has changed. The scanner uses this bit to tell VME master processors of a change in status of one of the adapters on the I/O link. An example is that if an operator temporarily pulled the swing arm from an adapter, the scanner would lose communications with that adapter. Maybe the operator could quickly put it back and the status LED and the adapter fault bit (bit 6) would say everything is working properly, but the adapter fault change bit would be left set, indicating that an adapter was temporarily off line. A VME master processor can issue the LINK STATUS command to make sure the I/O link and all adapters are operating properly upon seeing this bit set. Only the scanner can set this bit. The VME master processor must clear it.

Command Summary

The SV-compatible mode of the scanner supports these commands:

If you want to:	Use this command:	With this command byte (hex):	The scanner must be in this programming mode:	See page:
set the baud rate, watchdog rate, VMEbus interrupt level, how the scanner responds to SYSFAIL, and whether the scanner issues VMEbus interrupts	SETUP	13	program	5-7
establish a default scan list and provide status of the I/O system to the VME master processor	AUTOCONFIGURE	10	program	5-9
establish your own scan list	SCAN LIST	11	program	5-13
establish a fault dependent group structure	FAULT DEPENDENT GROUP	12	program	5-16
change the operating mode of the scanner	SET MODE	20	program test run	5-19
check adapter status and the scan list without affecting scanner operation	LINK STATUS	21	program test run	5-21
transfer a block of data from a specified I/O module to the scanner	BT READ	01	program test run	5-25
transfer a block of data from the scanner to a specified I/O module	BT WRITE	02	program test run	5-27
cause the scanner to reset itself	RESET	none	program test run	5-29

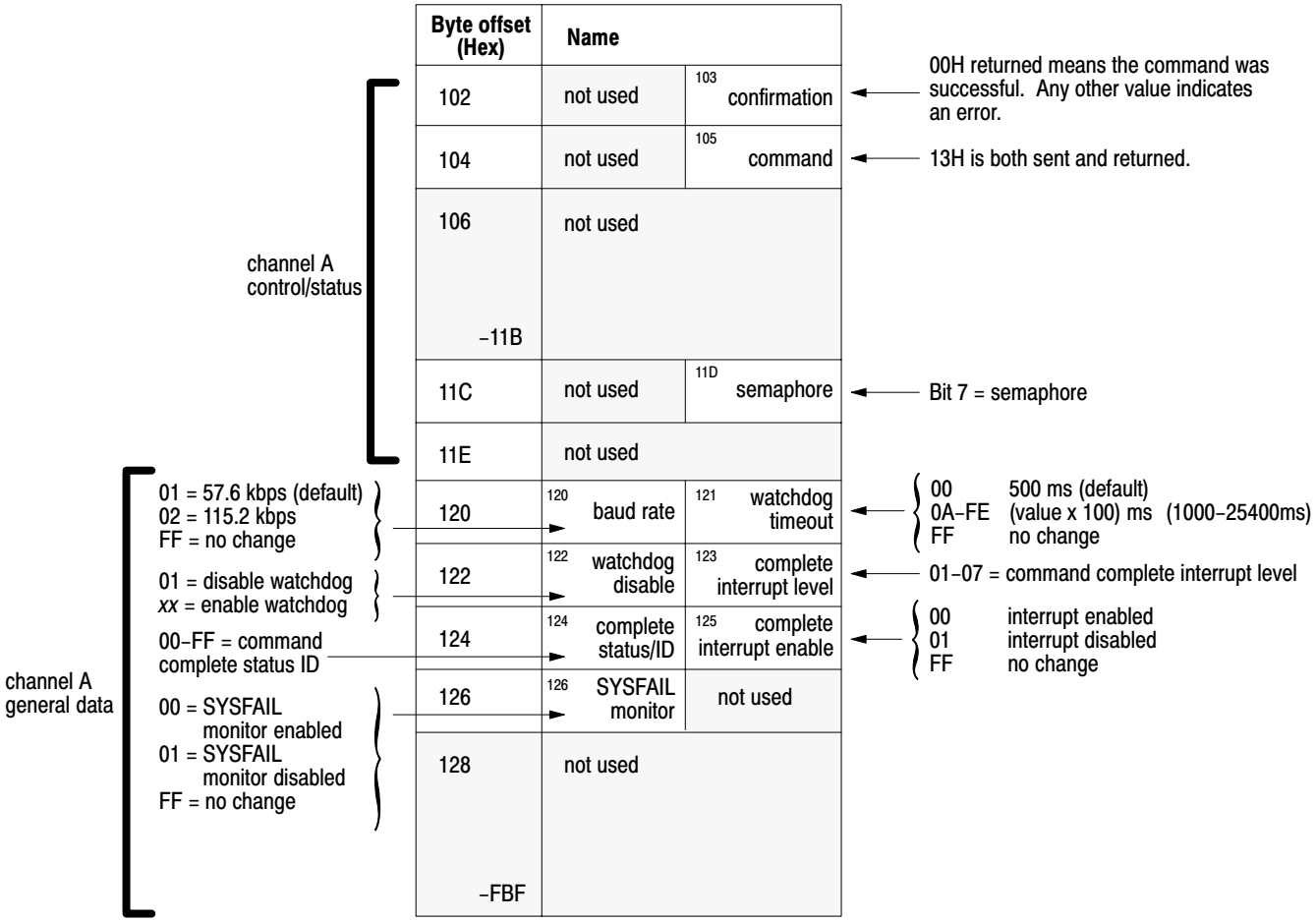
Waking up the scanner

When the scanner is first turned on, it does a self-test and then goes to sleep. A VME master processor must wake the scanner up by interrupting it (writing any value to the scanner's ID area), which moves the scanner from sleep mode to program mode. Chapter 7 explains how to wake up the scanner. Chapter 8 provides additional programming examples.

SETUP

command byte 13

description SETUP configures the scanner. The scanner must be in Program mode to execute this command. This is normally the first command sent to the scanner.



parameters The VME master processor writes 13 (hex) to the command byte. Then the VME master processor writes these parameters to the first 7 bytes of the selected channel's general data area:

Byte offset:	Parameter:	Description:
120	I/O link baud rate	Write one of these values (hex): 01 57.6 kbps (default) 02 115.2 kbps FF no change Values 00, 03–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
121	master processor watchdog timeout	Write one of these values (hex): 00 500 ms (default) 0A–FE (value x 100) ms (1000–25400ms) FF no change Values 01–09 are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
122	master processor watchdog disable	Write one of these values (hex): 01 disable watchdog timer 00, 02–FF enable watchdog timer (default) Disable the watchdog timer when you want to run a master processor application in debug mode without incurring timeouts that disrupt the application. The default is debug disabled.
123	command complete VMEbus interrupt level	Write one of these values (hex): 01 level 1 05 level 5 02 level 2 06 level 6 03 level 3 07 level7 04 level 4 (default) Values outside the range 01–07 are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
124	command complete interrupt status ID	The VME master processor writes an 8-bit value here that the scanner will pass to the VME interrupt handler during the VMEbus interrupt cycle. The default value is 40H.
125	command complete interrupt enable	Write one of these values (hex): 00 generate VMEbus interrupt upon command completion (default) 01 do not generate VMEbus interrupt upon command completion FF no change Values 02–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
126	SYSFAIL mask enable	Write one of these values (hex): 00 SYSFAIL monitor enabled (default) 01 SYSFAIL monitor disabled FF no change Values 02–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.

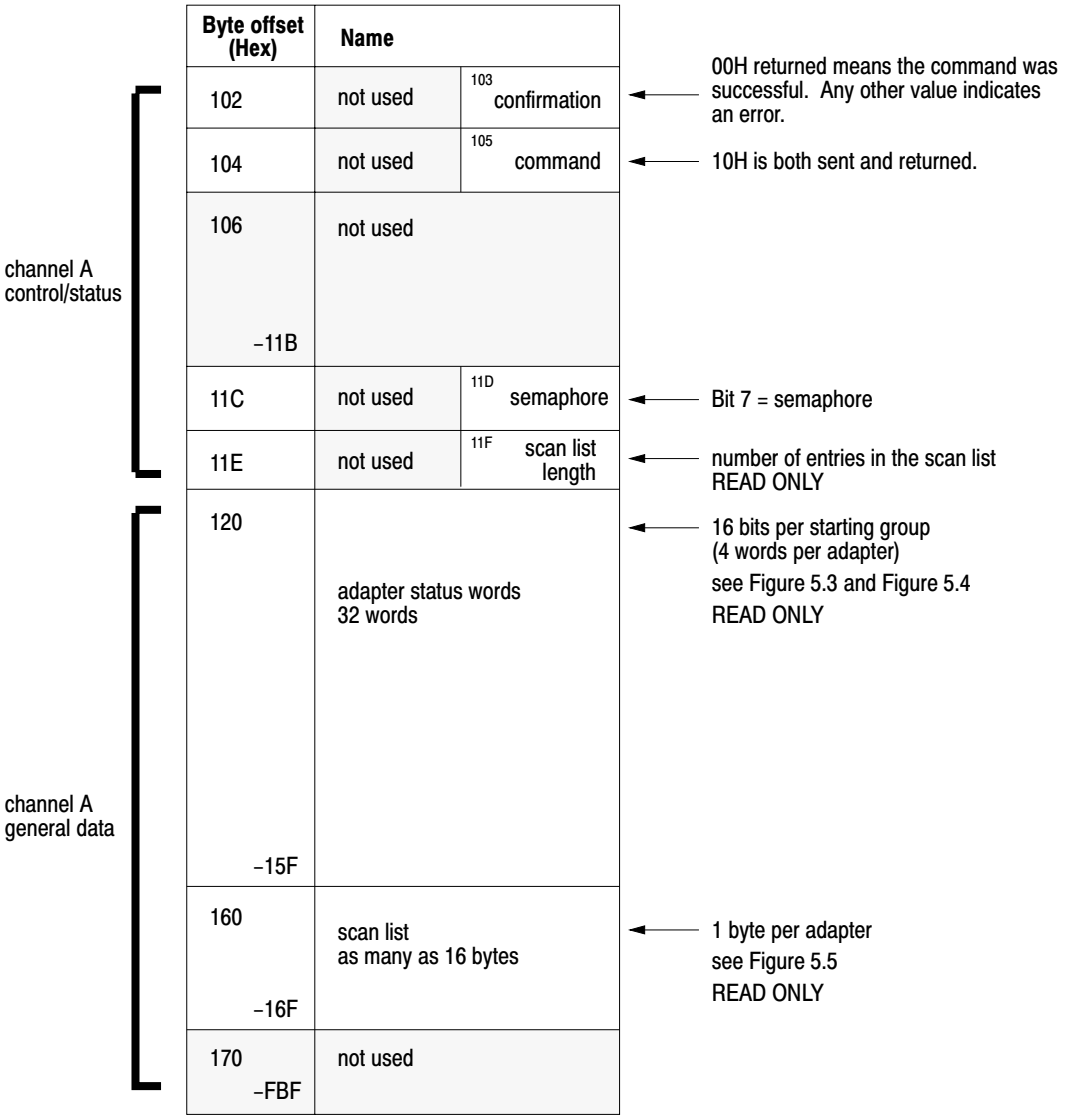
coding sequence Your code for a SETUP command should include these tasks:

1. get the semaphore
2. set up the general data area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

AUTOCONFIGURE

command byte 10

description AUTOCONFIGURE builds the scan list by polling every possible adapter address. Every adapter on the link that responds is placed once in the scan list. The scanner must be in Program mode to issue this command.



parameters The VME master processor writes 10 (hex) to the command byte. There are no input parameters from the VME master processor. The scanner writes these parameters to the selected channel's general data area:

Byte offset:	Parameter:	Description:
11F	scan list length READ ONLY	The scan list length indicates the number of entries in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area.
120	I/O adapter status word block READ ONLY	This word block contains four 16-bit entries (4 words) for each adapter – 16 bits for each starting group per adapter (maximum of 8 groups of entries). See Figure 5.3 and Figure 5.4.
160	scan list READ ONLY	This list is a maximum of 16 bytes long (1 byte per adapter). The list contains one byte-sized entry for each adapter found on the I/O link. See Figure 5.5 on page 5-12.

Figure 5.3
Format of the I/O adapter status word block and scan list

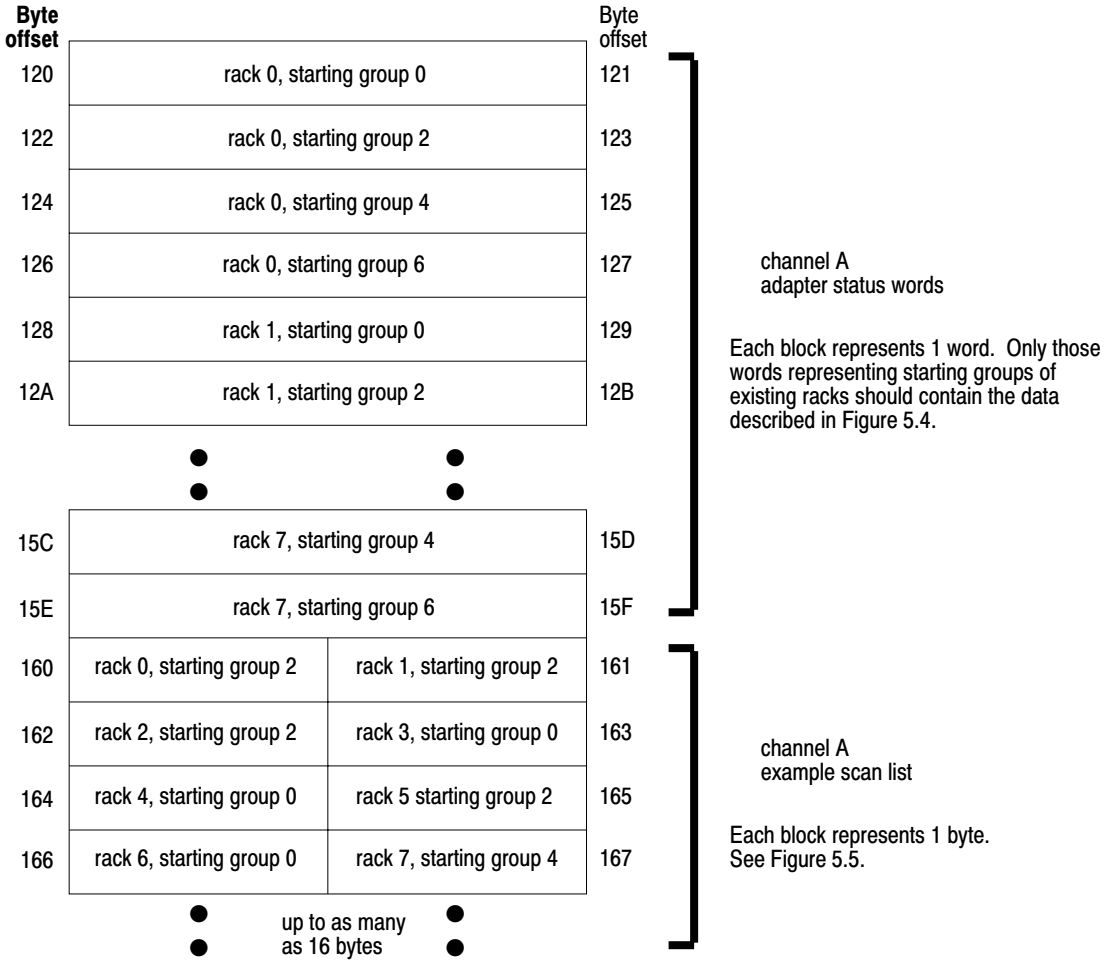
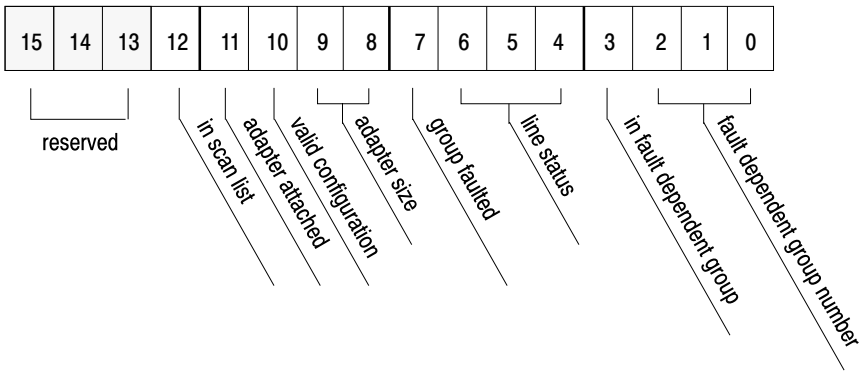
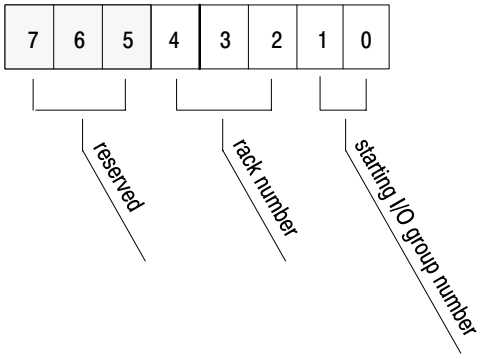


Figure 5.4
Format of one entry in the I/O adapter status word block (as shown in Figure 5.3)



Format item:	Description:
in scan list bit 12	Contains one of these values: 1 adapter is in the current scan list 0 adapter not in current scan list
adapter attached bit 11	Contains one of these values: 1 adapter resides at this address 0 no adapter resides at this address
valid configuration bit 10	Contains one of these values: 1 adapter type, size, and address are valid 0 adapter type, size, and/or address not valid; adapter is improperly configured
adapter size bits 9–8	Contains one of these values (hex): 00 1/4 rack addressing 01 1/2 rack addressing 10 3/4 rack addressing 11 full rack addressing
group faulted bit 7	Contains one of these values: 1 fault exists in the fault dependent group associated with the adapter 0 no fault exists in the fault dependent group associated with the adapter
line status bit 6–4	Contains one of these values (hex): 000 adapter is off line any other value adapter is on line
in fault dependent group bit 3	Contains one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	The value (0–7 binary) identifies the fault dependent group to which this adapter belongs.

Figure 5.5
Format for one entry in the scan list



Format item:	Description:
rack number bits 4–2	The value (0–7 binary) identifies the rack number.
starting I/O group number bits 1–0	One of the following values identifies the starting I/O group: 00 group 0 01 group 2 10 group 4 11 group 6

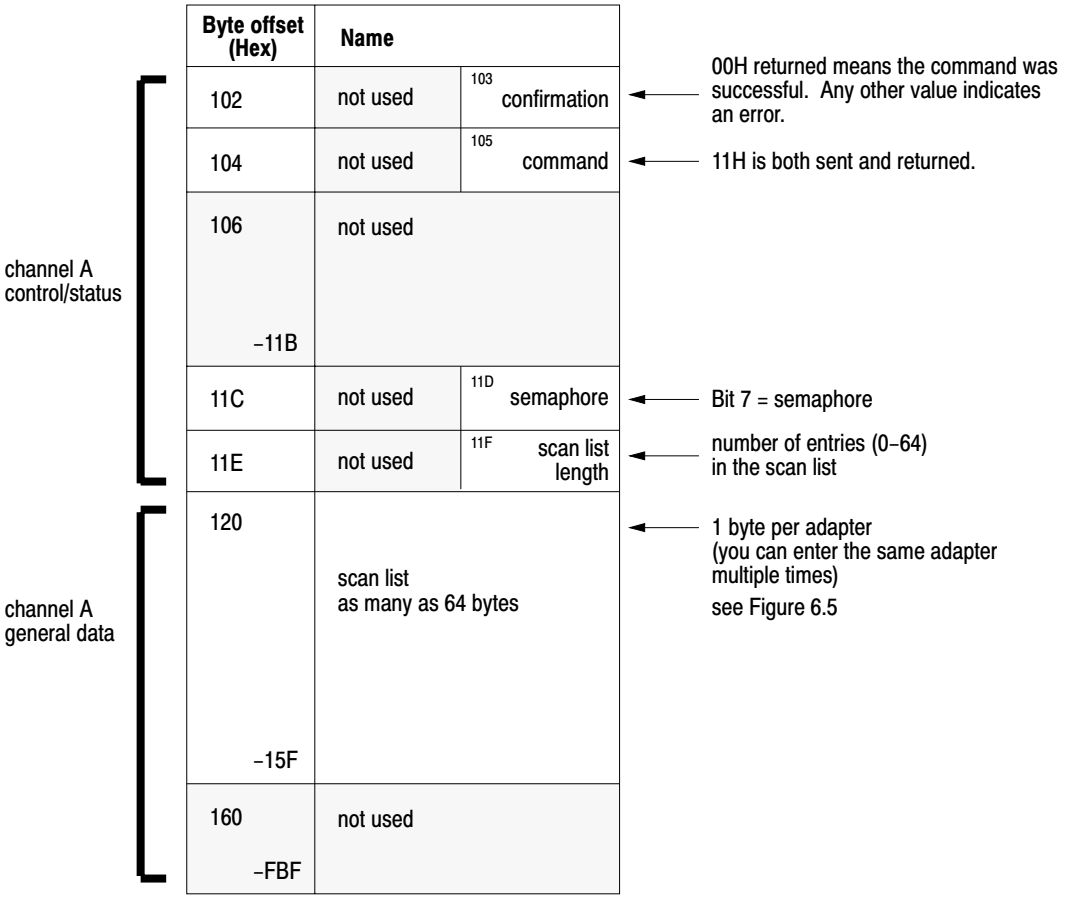
coding sequence Your code for the AUTOCONFIGURE command should include these tasks:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. check the result
6. clear the semaphore

SCAN LIST

command byte 11

description SCAN LIST replaces the current scan list. The scanner must be in Program mode. You can issue a SCAN LIST command without first issuing an AUTOCONFIGURE command.



parameters The VME master processor writes 11 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's general data area:

Byte offset:	Parameter:	Description:
11F	scan list length	The scan list length indicates the number of entries (0–64) in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area. You can enter a scan list length of 0.
120	scan list	This list is a maximum of 64 bytes long (1 byte per adapter), but it can contain only 16 distinct physical adapter addresses. The list contains one byte-sized entry for each adapter you want to place in the list. An adapter can appear in the list multiple times. See Figure 5.6 and Figure 5.7.

Figure 5.6
Example scan list

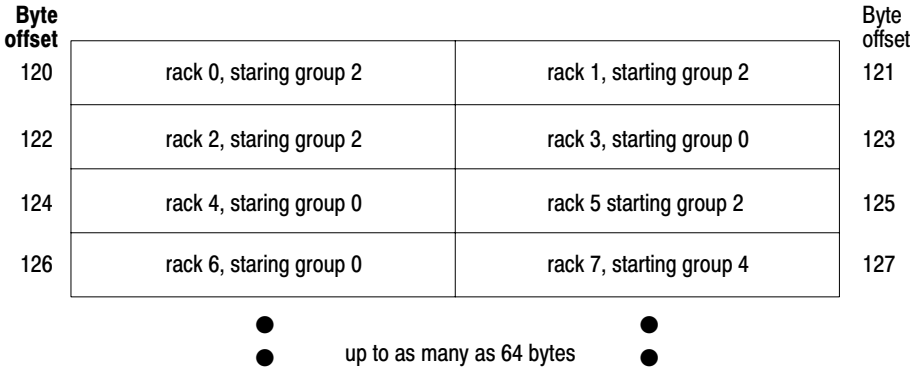
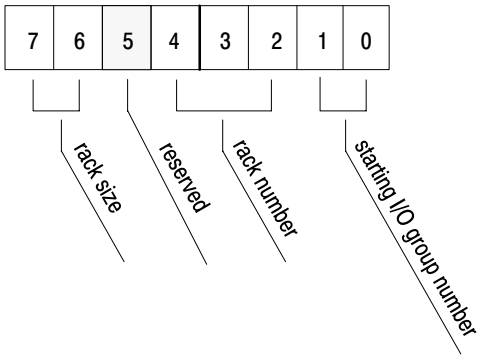


Figure 5.7
Format for one entry in the scan list



Format item:	**Description:**
rack size bits 7–6	Write one of these values: 00 1/4 rack 01 1/2 rack 10 3/4 rack 11 full rack These bits function differently than with the AUTOCONFIGURE command. With this command, you must specify the rack size you want – which could be larger than the actual rack size to allow for future expansion. The AUTOCONFIGURE command determines the actual rack size and writes the appropriate value.
rack number bits 4–2	Write the value (0–7 binary) of the rack number to be scanned.
starting I/O group number bits 1–0	Write one of these values: 00 group 0 01 group 2 10 group 4 11 group 6

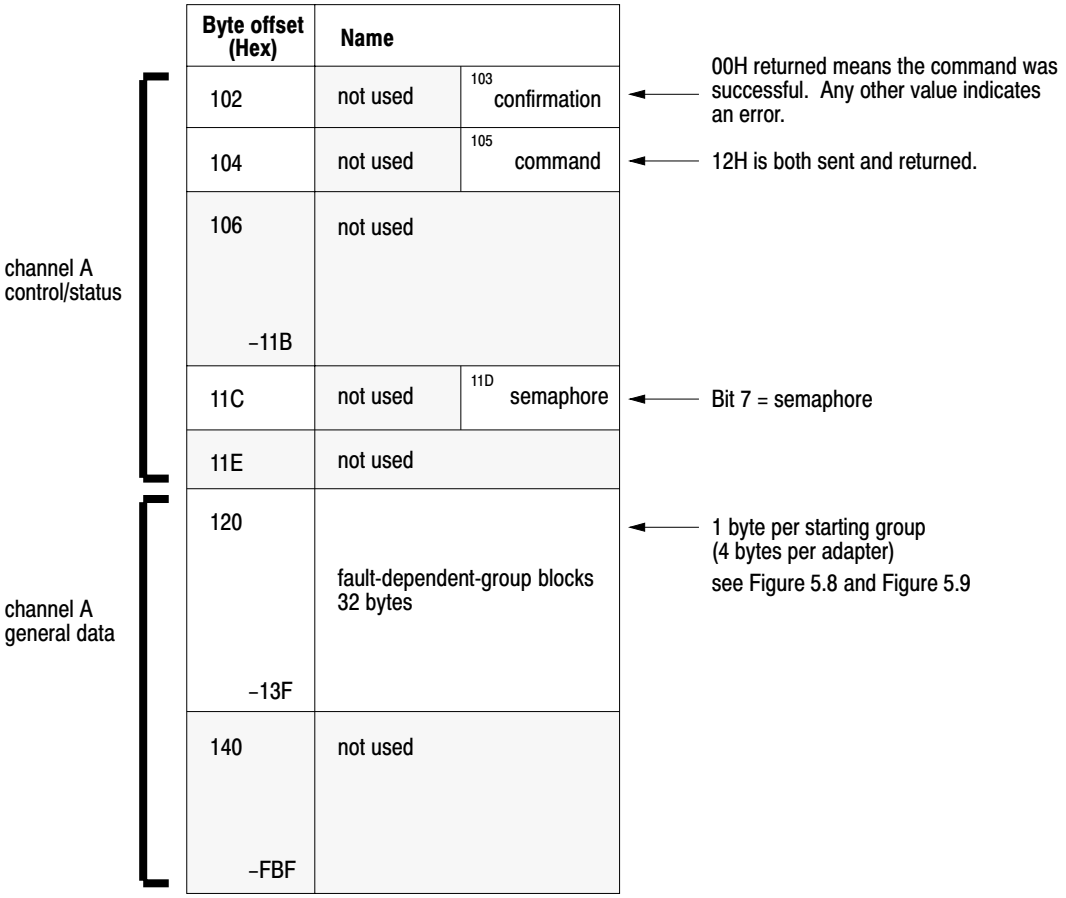
coding sequence Your code for the SCAN LIST command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the scan list to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. clear the semaphore

FAULT DEPENDENT GROUP

command byte 12

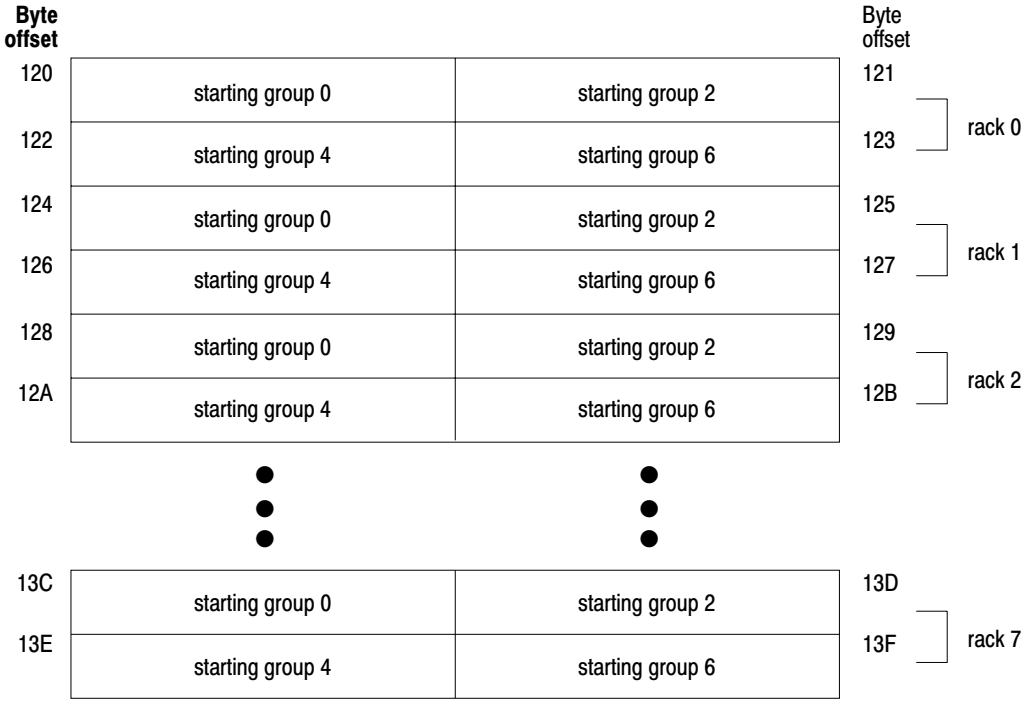
description FAULT DEPENDENT GROUP associates a set of adapters such that if one adapter in the group faults, all the other adapters in the group fault, going to their fail-safe mode of operation. The scanner must be in Program mode to issue this command. You can specify as many as 8 fault groups.



parameters The VME master processor writes 12 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's general data area:

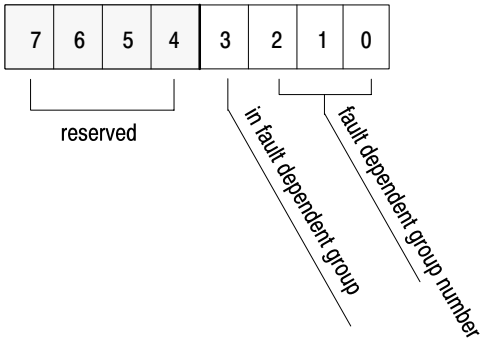
Byte offset:	Parameter:	Description:
120	fault dependent group block	There are 4 configurable bits for each starting group entry. An adapter must be in the current scan list to be assigned to a fault-dependent group. If you specify an adapter that is not in the scan list, the scanner writes an illegal configuration error (16H) to the confirmation status byte. See Figure 5.8 and Figure 5.9.

Figure 5.8
Format of the fault dependent group block



Each block represents 1 byte. There is 1 byte for each starting group for each adapter. Only those bytes representing starting groups of existing racks should contain the data described in Figure 5.9. Zero all entries for which adapters do not exist.

Figure 5.9
Format for one entry in the fault dependent group block



Format item:	Description:
in fault dependent group bit 3	Write one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	Write a value (0–7 binary) to identify the fault dependent group to which this adapter belongs.

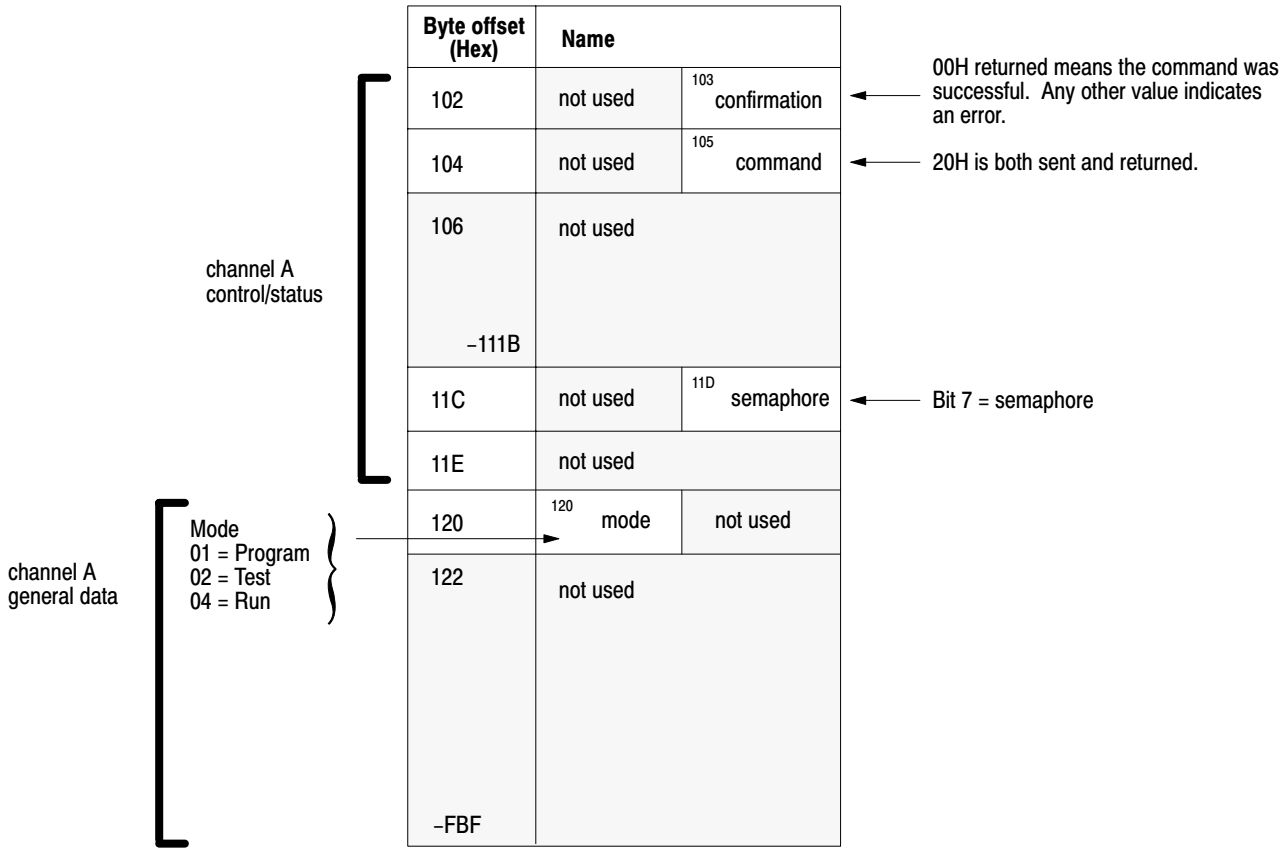
coding sequence Your code for the FAULT DEPENDENT GROUP command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the fault dependent group data to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. clear the semaphore

SET MODE

command byte 20

description SET MODE changes the operating mode of the scanner.



parameters The VME master processor writes 20 (hex) to the command byte. Then the VME master processor writes this parameter to the selected channel's general data area:

Byte offset:	Parameter:	Description:
120	mode	Write one of these values (hex): 01 Program mode 02 Test mode 04 Run mode Values 00, 03, and 05–FF are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte.

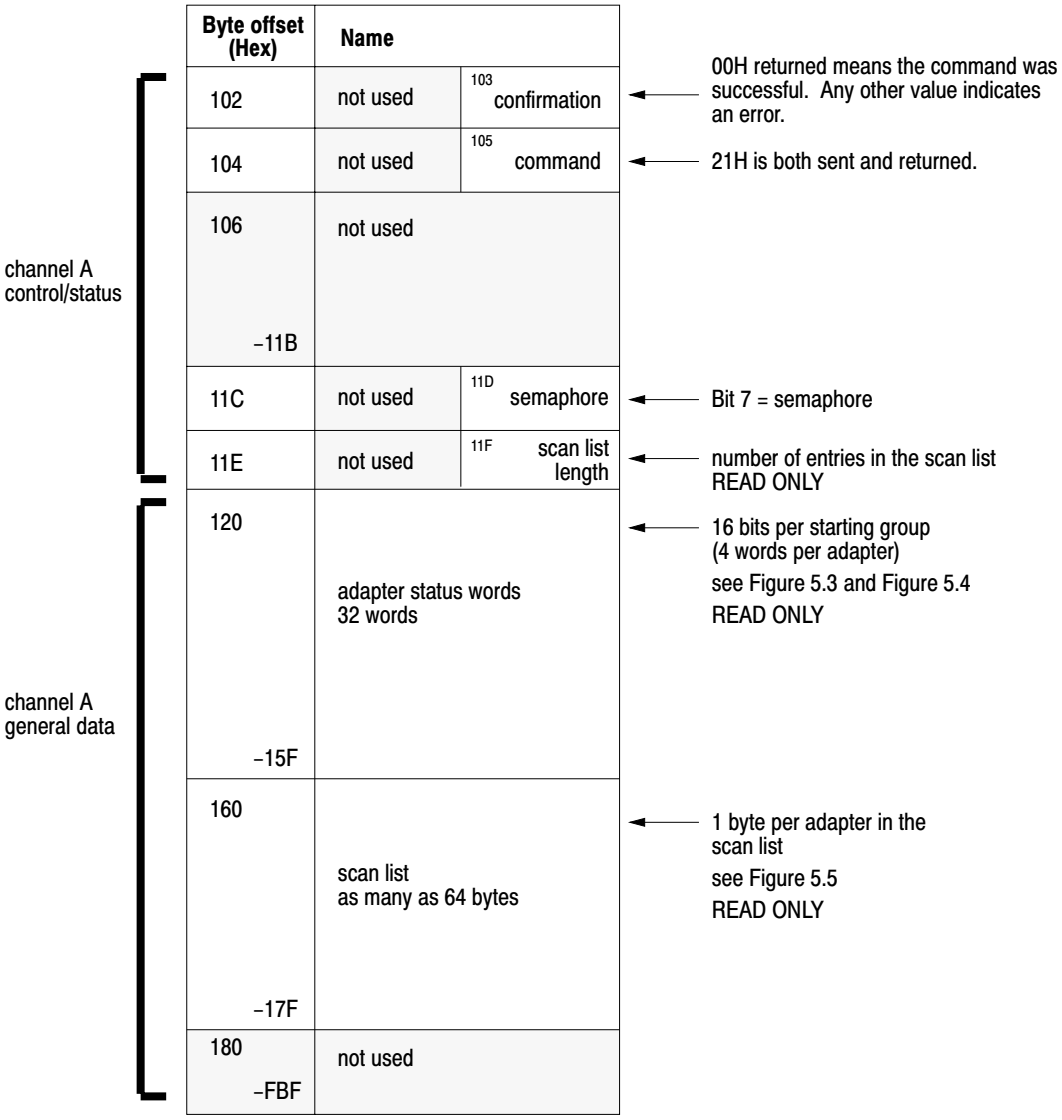
coding sequence Your code for the SET MODE command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the mode value to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. check the confirmation status
7. clear the semaphore

LINK STATUS

command byte 21

description LINK STATUS determines the current status of the adapters on the selected channel's I/O link. LINK STATUS doesn't affect the scanner. The VME master processor can issue LINK STATUS any time.



parameters The VME master processor writes 21 (hex) to the command byte. There are no input parameters from the VME master processor. The scanner writes these parameters to the selected channel's general data area:

Byte Offset	Parameter	Description
11F	scan list length READ ONLY	The scan list length indicates the number of entries in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area. You can enter a scan list length of 0.
120	I/O adapter status word block READ ONLY	This word block contains four 16-bit entries (4 words) for each adapter – 16 bits for each starting group per adapter (maximum of 8 groups of entries). See Figure 5.10 and Figure 5.11.
160	scan list READ ONLY	This list is a maximum of 64 bytes long. The list contains one byte-sized entry for each adapter in the scan list, but it contains only 16 distinct physical adapter addresses. An adapter can appear in the list multiple times. See Figure 5.12 on page 5-24.

Figure 5.10
Format of the I/O adapter status word block and scan list

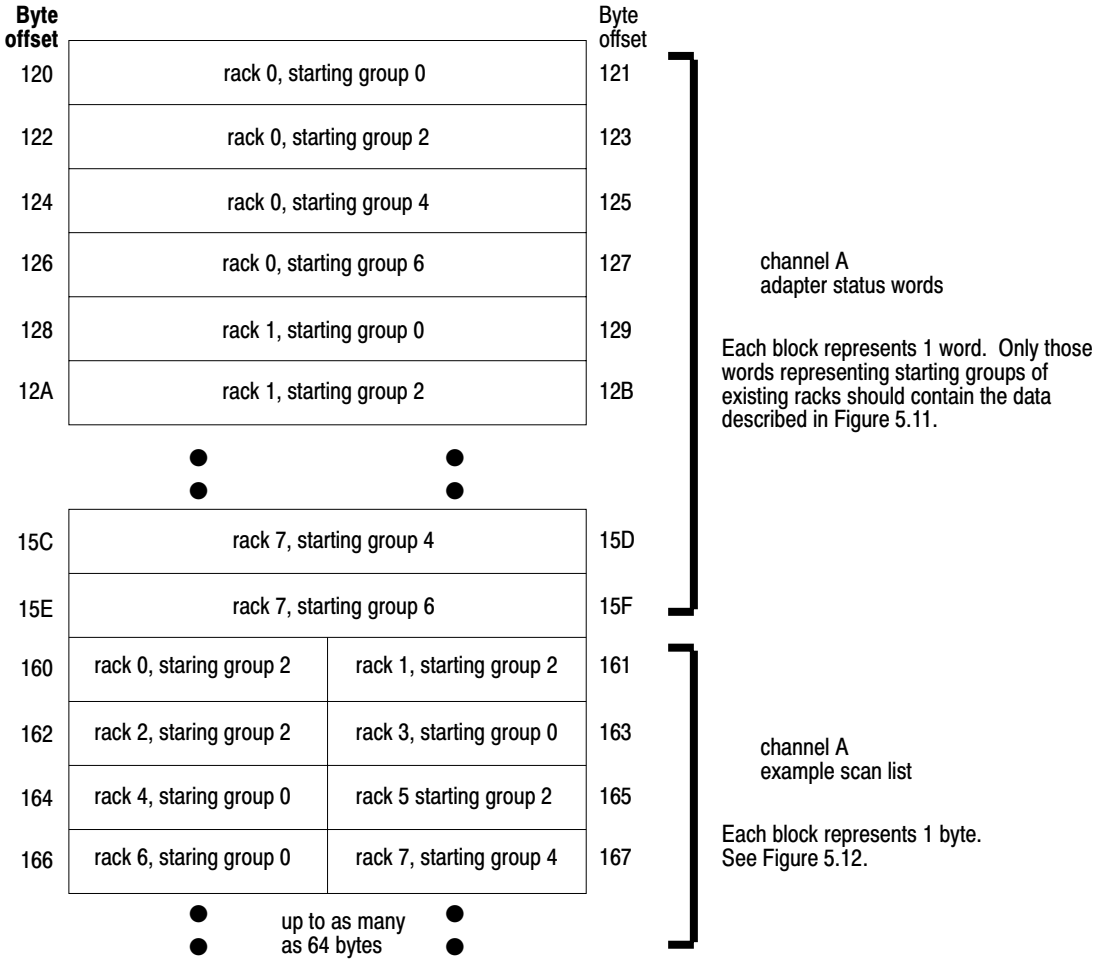
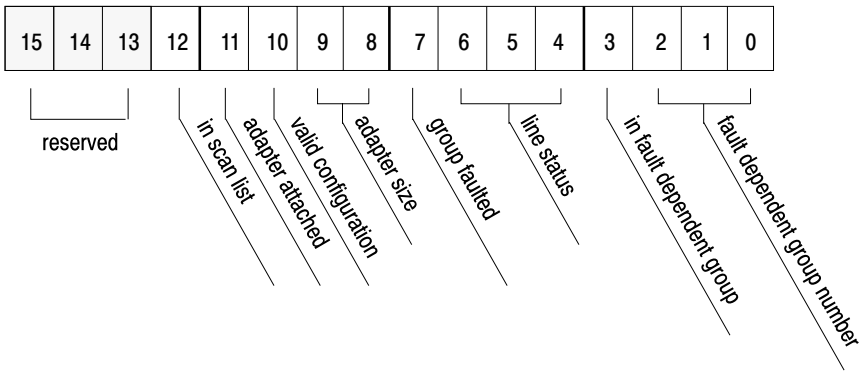
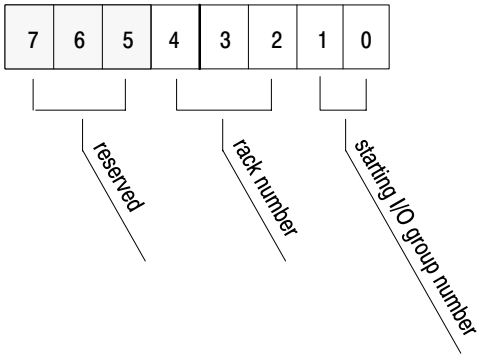


Figure 5.11
Format of one entry in the I/O adapter status word block (as shown in Figure 5.3)



Format item:	Description:
in scan list bit 12	Contains one of these values: 1 adapter is in the current scan list 0 adapter not in current scan list
adapter attached bit 11	Contains one of these values: 1 adapter resides at this address 0 no adapter resides at this address
valid configuration bit 10	Contains one of these values: 1 adapter type, size, and address are valid 0 adapter type, size, and/or address not valid; adapter is improperly configured
adapter size bits 9–8	Contains one of these values (hex): 00 1/4 rack addressing 01 1/2 rack addressing 10 3/4 rack addressing 11 full rack addressing
group faulted bit 7	Contains one of these values: 1 fault exists in the fault dependent group associated with the adapter 0 no fault exists in the fault dependent group associated with the adapter
line status bit 6–4	Contains one of these values (hex): 000 adapter is off line any other value adapter is on line
in fault dependent group bit 3	Contains one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	The value (0–7 binary) identifies the fault dependent group to which this adapter belongs.

Figure 5.12
Format for one entry in the scan list



Format item:	Description:
rack number bits 4–2	The value (0–7 binary) identifies the rack number.
starting I/O group number bits 1–0	One of the following values identifies the starting I/O group: 00 group 0 01 group 2 10 group 4 11 group 6

coding sequence Your code for the LINK STATUS command should include these tasks:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. check the result
6. clear the semaphore

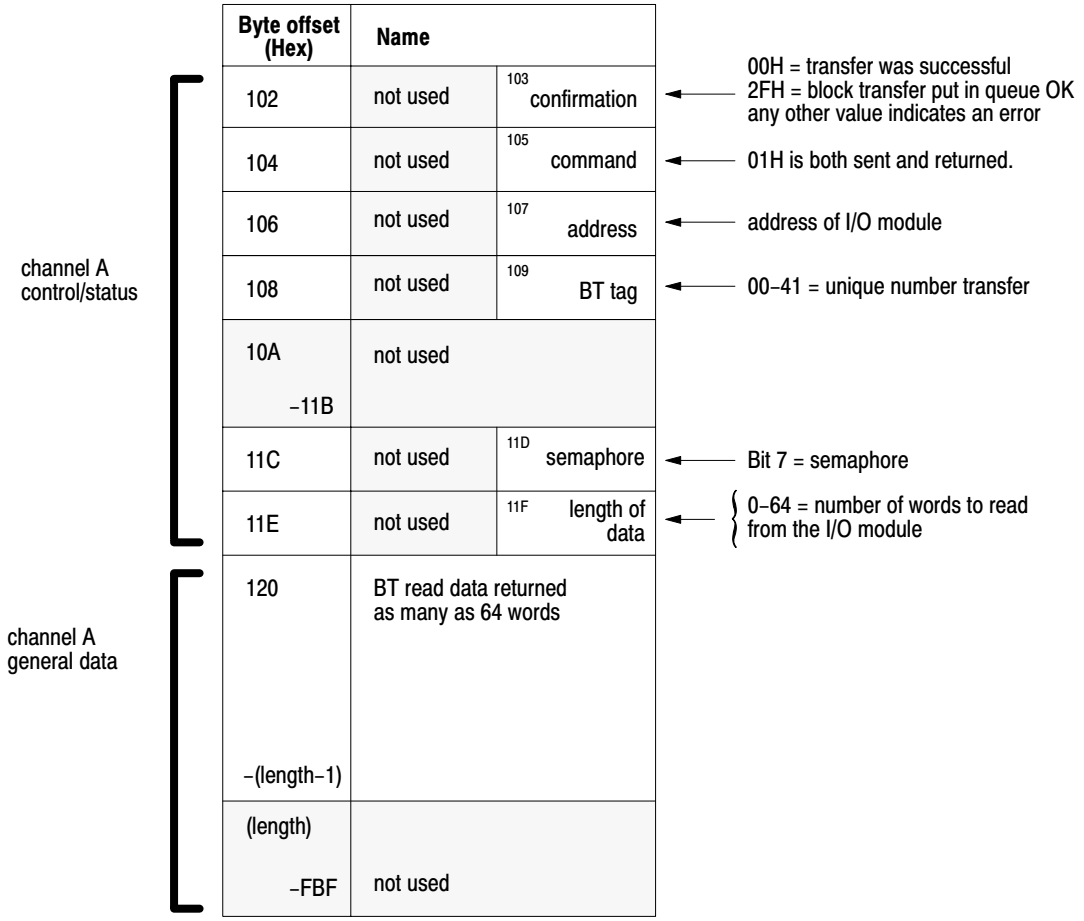
BT READ

command byte 01

description BT READ transfers a block of data from the specified I/O module into the scanner.

Upon issuing a block-transfer request, the scanner writes 0x2F in the confirmation status byte to indicate that the scanner accepted the request and put it into its queue. When the scanner actually completes the block-transfer, the scanner sets the semaphore and then it updates this data in the following order:

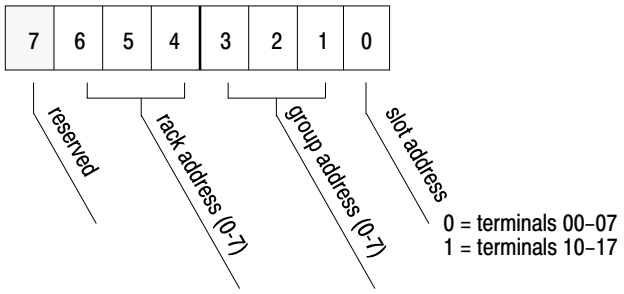
1. address
2. block transfer tag
3. length of data
4. confirmation status
5. block transfer read data
6. command complete interrupt (if enabled)



parameters The VME master processor writes 01 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's control status area:

Byte offset:	Parameter:	Description:
107	module address	The module address is the address of the I/O module from which to read the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 5.13.
109	block-transfer tag number	The block-transfer tag number is an 8-bit integer that uniquely identifies each block-transfer. The scanner writes a value 0–41 to the block-transfer request tag number in the control/status area. Values greater than 41 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16) in the confirmation status byte. This tag is returned with the status when the block transfer is complete so the VME master processor can match the block transfer status with the request.
11F	length of data	The length of data byte is an 8-bit value that specifies the number of 16-bit words (0–64 decimal) to be read from the target I/O module. Use the value 0 to let the I/O module determine how many words the scanner can read. When the block-transfer completes, the 0 is replaced by the actual number of words read. Values greater than 64 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16) in the confirmation status byte.
120	BT data	The BT data is the set of data words (0–64) read from the target I/O module. The BT data block ends at offset (length of data – 1).

Figure 5.13
Structure of the address byte



coding sequence Your code for the BT READ command should include these tasks:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

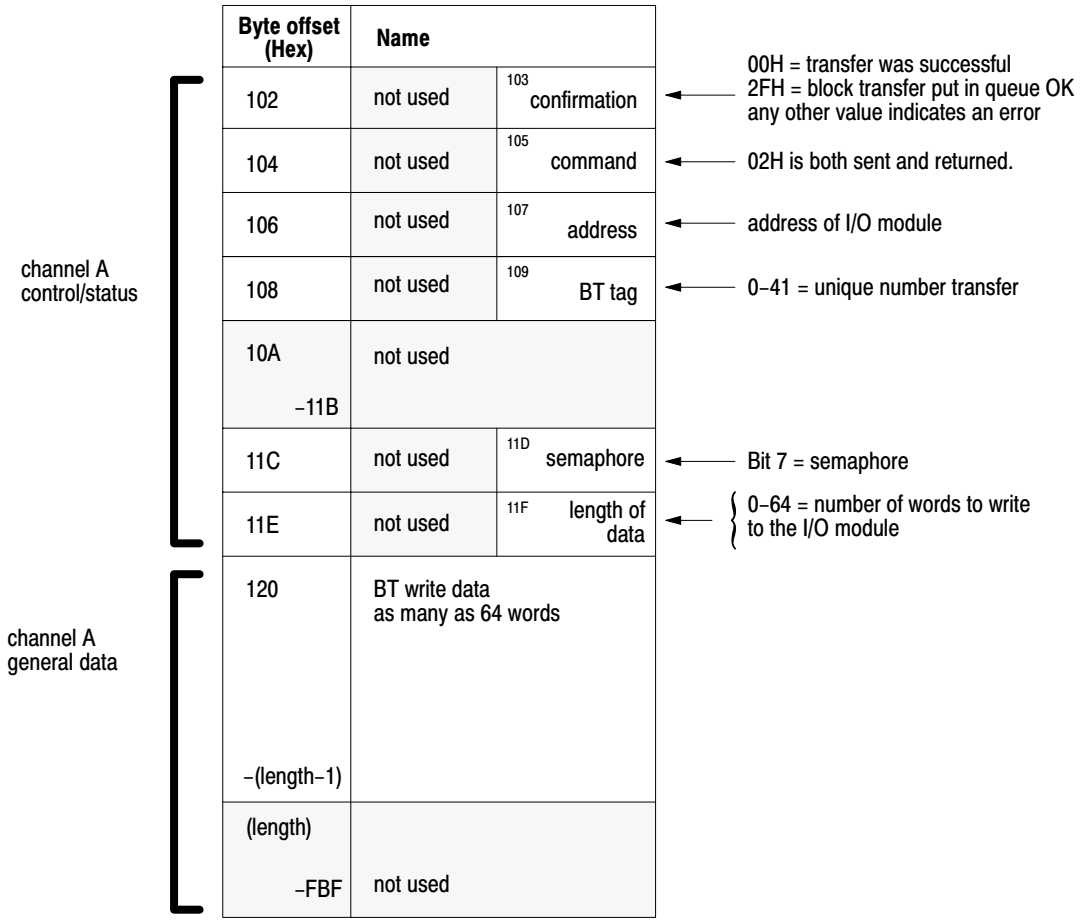
BT WRITE

command byte 02

description BT WRITE transfers a block of data from the scanner to the specified I/O module.

Upon issuing a block-transfer request, the scanner writes 0x2F in the confirmation status byte to indicate that the scanner accepted the request and put it into its queue. When the scanner actually completes the block-transfer, the scanner sets the semaphore and then it updates this data in the following order:

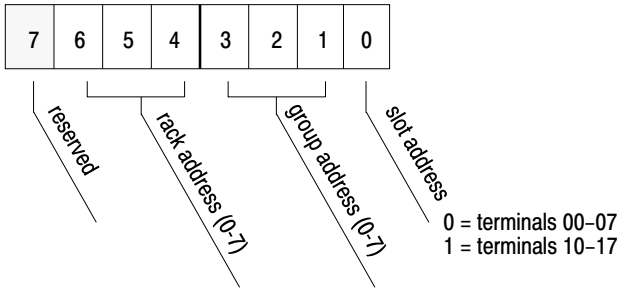
1. address
2. block transfer tag
3. length of data
4. confirmation status
5. block transfer read data
6. command complete interrupt (if enabled)



parameters The VME master processor writes 02 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's control status area:

Byte offset:	Parameter:	Description:
107	module address	The module address is the address of the I/O modules that is to receive the block data. The module address is the address of the I/O module that is to receive the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 5.14.
109	block-transfer tag number	The block-transfer tag number is an 8-bit integer that uniquely identifies each block-transfer. The scanner writes a value 0–41 to the block-transfer request tag number in the control/status area. Values greater than 41 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16) in the confirmation status byte. This tag is returned with the status when the block transfer is complete so the VME master processor can match the block transfer status with the request.
11F	length of data	The length of data byte is an 8-bit value that specifies the number of 16-bit words (0–64 decimal) to be written to the target I/O module. Use the value 0 to let the I/O module determine how many words the scanner can write. When the block-transfer completes, the 0 is replaced by the actual number of words written. Values greater than 64 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16) in the confirmation status byte.
120	BT data	The BT data is the set of data words (0–64) sent to the target I/O module. The BT data block ends at offset (length of data – 1).

Figure 5.14
Structure of the address byte



coding sequence Your code for the BT WRITE command should included these tasks:

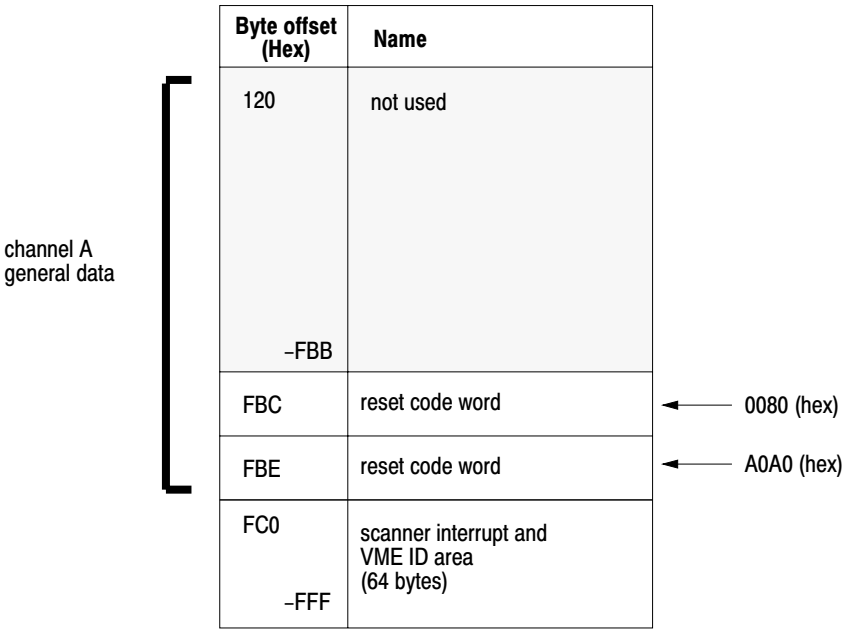
1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

RESET

description RESET causes the scanner to reset itself. The VME master processor can issue RESET any time. When the scanner stops running due to another VME module asserting SYSFAIL, you can use RESET to reset the scanner, which causes the scanner to re-initialize itself.

RESET looks for the appropriate values in the last 2 words of the selected channel's general data area. If the values are there, the scanner resets itself, performs its power-on self-test, and enters the SLEEP state. The application program has to wake up the scanner and then send a SETUP command to configure the scanner (see chapter 8 for an example). This applies to either channel.

During the RESET, the scanner stops scanning the remote I/O and all three LEDs on the scanner light. RESET is different than a power cycle to the scanner in that with RESET, the scanner doesn't assert SYSFAIL during the self tests and the scanner doesn't clear the I/O image tables. The scanner leaves the I/O image tables in their last state. The scanner then enters the SLEEP state.



parameters There are no parameters for the RESET command. Before the scanner executes the RESET command, the VME master processor writes these parameters to the last 2 words of the selected channels' general data area:

Byte offset:	Parameter:	Description:
FBC	second-to-last word in the selected channel's general data area	Write the value 0080 (hex) to this word.
FBE	last word in the selected channel's general data area	Write the value A0A0 (hex) to this word.

coding sequence Your code for the RESET command needs to write the above words to byte offsets FBC and FBE.

Chapter 6

Operating in SV-Superset Mode

Using This Chapter

The SV-superset mode provides these features in addition to those supported by the SV-compatible mode:

- each channel supports as many as 16 logical racks per channel (32 physical adapters); the SV-compatible mode supports as many as 8 logical racks per channel (16 physical adapters)
- the global data area is 224 bytes long
- you can configure the scanner to interrupt the VME master processor when the scanner detects a change in a specified range of input image table; the scanner also provides the offset for the first changed data
- the scanner supports continuous block transfers and each channel has a table for continuous block-transfer read data and a table for continuous block-transfer write data
- use the SETUP command to select a communication rate of 57.6, 115.2, or 230.4 kbps

This chapter describes the SV-superset mode.

Addressing Global RAM

Both the VME master processor and the scanner can read and write to the scanner's VME global RAM. The global RAM structure depends on the VME operating mode of the scanner. Figure 6.1 shows the general structure for the SV-superset operating mode. Page 6-3 shows the specifics of the structure for channel A.

Important: The scanner must be configured for A24 address space if you use the SV-superset mode. For more information about configuring address space, see chapter 2.

Figure 6.1
General structure of global RAM for SV-superset mode

Channel A byte offset (hex)		Channel B (6008-SV2R scanner only) byte offset (hex)	
0000	**output image table** 128 words	2000	**output image table** 128 words
0100	**input image table** 128 words	2100	**input image table** 128 words
0200	**control/status area** 16 words	2200	**control/status area** 16 words
0220	**general data area** 112 words	2220	**general data area** 112 words
0300	**continuous BT write** 16 entries (72 words each)	2300	**continuous BT write** 16 entries (72 words each)
0C00	**continuous BT read** 32 entries (72 words each)	2C00	**continuous BT read** 32 entries (72 words each)
1FC0	**interrupt/VME ID area** 32 words	3FC0	**interrupt/VME ID area** 32 words

The physical address is the base address plus the byte offset.

Important: Add 2000 (hex) to channel A addresses to get the corresponding addresses for channel B.

global RAM structure for SV-superset mode
Channel A

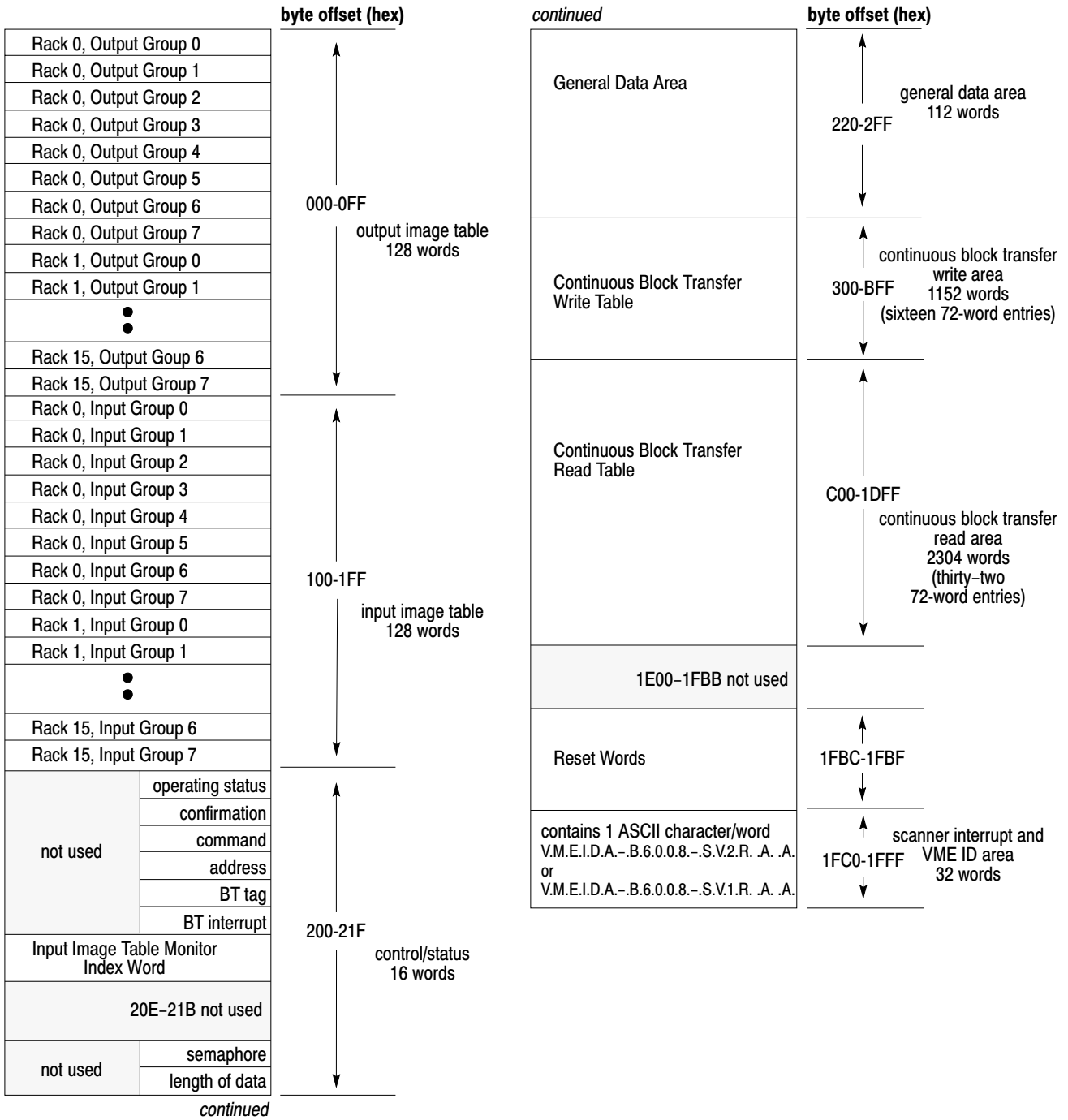


Table 6.A describes the components of global RAM.

Table 6.A
Descriptions of the global memory areas

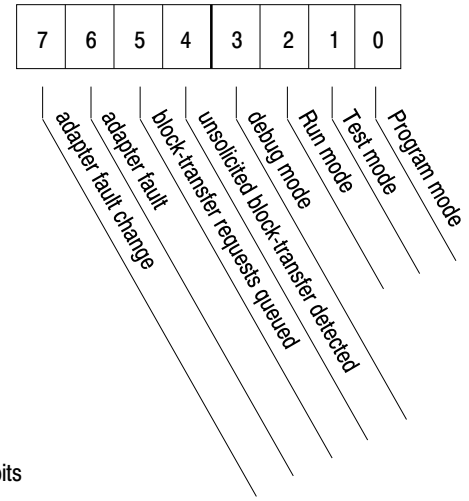
This area:	stores the:	
input and output image table	input and output data for as many as 16 logical I/O racks with no more than 32 adapters.	
control/status area	operating status byte	reflects the current status of the scanner lets the VME master processor poll the scanner's status without interrupting ongoing operations see Figure 6.2 on page 6-5
	confirmation status byte	contains the result of the executed command A result of 00 (hex) indicates the command completed successfully. See chapter 9 for a list of error codes.
	command byte	identifies the command the VME master processor wants the scanner to execute next
	address byte	contains I/O rack, group, and slot parameters for block transfer commands For more information, see BT READ command on page 6-28 BT WRITE command on page 6-32 CONTINUOUS BT READ command on page 6-36 CONTINUOUS BT WRITE command on page 6-40
	block transfer tag byte	provided by the VME master processor to uniquely identify each one-shot block transfer request or continuous block-transfer entry
	block transfer interrupt byte	indicates the VME interrupt level and interrupt acknowledgement vector for when a continuous block transfer periodically completes For more information, see BT WRITE command on page 6-32 or BT READ command on page 6-36. Use the SETUP command to specify the interrupt level and interrupt vector.
	input image table monitor index word	when the input image table monitor is enabled, this word contains the index of the last entry that changed in the input image table when the input image table monitor is disabled, this word is undefined Use the SETUP command to enable or disable the input image table monitor.
	semaphore byte	provides for the integrity of command requests, command responses, and completed one-shot block transfers If bit 7 is set, a VME master processor is using the control/status and general data area or the scanner just completed a block transfer and set the semaphore so no other VME master processor will overwrite the data. If the semaphore is set because of a block transfer, the appropriate VME master will know to come and retrieve the data and reset the semaphore. If the semaphore bit is clear, the general data area is available for access by any VME master processor. **Important:** A VME master processor doesn't use the semaphore when it accesses the I/O image table. The VME master processor can access the I/O image table anytime. For command requests, the scanner returns status, either confirmation or error, when the command is complete. When the VME master processor receives the confirmation status, it must retrieve the data from the general data area and clear the semaphore. **Important:** Only 1 command can be issued at a time. A status confirmation or error must be received before a new command is issued.

This area:	stores the:	
	length of data byte	specifies the amount of data associated with a command or response Only the lower byte is used. Interpret the length of data based on the context of the requested command or received confirmation (i.e., number of words for block transfers and bytes for scanner management requests).
general data area	contains input parameters and result data for scanner commands	
continuous block transfer read and write table	each table contains either write or read data for continuous block transfers The write table has 16 72-word buffers; the read table has 32 72-word buffers. For more details, see the CONTINUOUS BT WRITE command on page 6-40 or the CONTINUOUS BT READ command on page 6-36.	
scanner interrupt and VME ID	interrupt from the VME master processor to the scanner The scanner interrupt area and VME ID area can be read anytime without causing an interrupt to the scanner. It contains 32 words in which the odd (lower byte) only is used; the information is an ASCII character string: V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.2.R. .x. .y. **or** V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.1.R. .x. .y. where .x. .y. are the series and revision levels of the scanner. For example: V.M.E.I.D.A.–.B.6.0.0.8.–.S.V.2.R. .A. .A. for series A, revision A. Writing to any byte in this area interrupts the scanner. The scanner then looks at the command byte to determine which command to execute. All even bytes have the value 0xFF; all odd bytes without ASCII characters have the value 0x00.	

Operating Status Byte

The operating status byte holds scanner status information for the VME master processor. The scanner updates this byte every time it completes either a block transfer or a command. The master processor can clear bits 4 and 7; the remaining bits are read only. The structure is as follows:

Figure 6.2
Structure of the operating status byte



see the following table for descriptions of these bits

These bits:	Considerations:
bits 0–2	These bits specify the programming mode of the scanner. These bits are mutually exclusive – the scanner can be in only one of these modes.
bit 3	If the debug mode bit is set by the scanner, the scanner cannot be shutdown by the internal watchdog. **ATTENTION:** Unwanted machine motion can result from disabling the VME master processor watchdog. Only use the debug mode when you are debugging the application program for the VME master processor.
bit 4	The scanner sets bit 4 when it detects an unsolicited block transfer. An unsolicited block transfer results if a VME master processor accidentally writes discrete information to an output image table byte that is mapped to an I/O slot requiring a block transfer. When this bit is set, take action to correct this situation because it can seriously degrade scanner response time. The scanner can only set this bit; the VME master processor must clear it.
bit 5	If bit 5 is set, the scanner has at least one block transfer request in its internal queue. When the associated block transfer is completed and the queue is empty, the scanner will clear the bit.
bit 6	If bit 6 is set, at least one adapter has faulted, lost power, or has been dropped from the I/O link. The VME master processor can issue the LINK STATUS command for fault details. When all the adapters in the scan list are cleared of faults or have been brought back on line, the scanner will clear this bit.
bit 7	If bit 7 is set, an adapter's operating status has changed. The scanner uses this bit to tell VME master processors of a change in status of one of the adapters on the I/O link. An example is that if an operator temporarily pulled the swing arm from an adapter, the scanner would lose communications with that adapter. Maybe the operator could quickly put it back and the status LED and the adapter fault bit (bit 6) would say everything is working properly, but the adapter fault change bit would be left set, indicating that an adapter was temporarily off line. A VME master processor can issue the LINK STATUS command to make sure the I/O link and all adapters are operating properly upon seeing this bit set. Only the scanner can set this bit. The VME master processor must clear it.

Command Summary

The SV-superset mode of the scanner supports these commands:

If you want to:	Use this command:	With this command byte (hex):	The scanner must be in this programming mode:	See page:
set the baud rate, watchdog rate, VMEbus interrupt level, how the scanner responds to SYSFAIL, command complete interrupt, and the input image table interrupt	SETUP	13	program	6-8
establish a default scan list and provide status of the I/O system to the VME master processor	AUTOCONFIGURE	10	program	6-11
establish your own scan list	SCAN LIST	11	program	6-15
establish a fault dependent group structure	FAULT DEPENDENT GROUP	12	program	6-18
change the operating mode of the scanner	SET MODE	20	program test run	6-21
check adapter status and the scan list without affecting scanner operation	LINK STATUS	21	program test run	6-23
transfer a block of data from a specified I/O module to the scanner	BT READ	01	program test run	6-28
transfer a block of data from the scanner to a specified I/O module	BT WRITE	02	program test run	6-32
continuously transfer a block of data from a specified I/O module to the scanner	CONTINUOUS BT READ	06	program	6-36
continuously transfer a block of data from the scanner to a specified I/O module	CONTINUOUS BT WRITE	07	program	6-40
cause the scanner to reset itself	RESET	none	program test run	6-44

Waking up the scanner

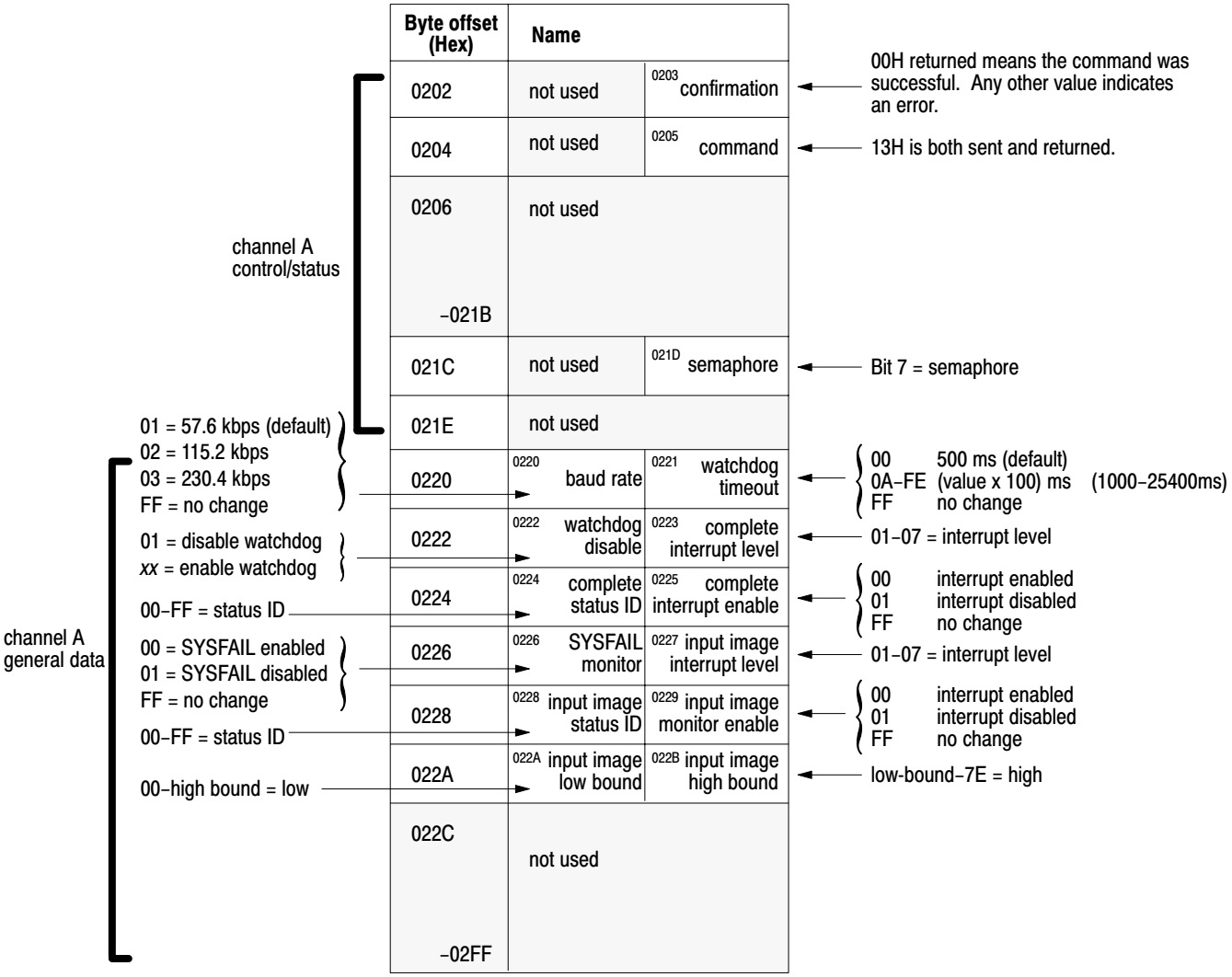
When the scanner is first turned on, it does a self-test and then goes to sleep. A VME master processor must wake the scanner up by interrupting it (writing any value to the scanner's ID area), which moves the scanner from sleep mode to program mode. Chapter 7 explains how to wake up the scanner.

Chapter 8 provides additional programming examples.

SETUP

command byte 13

description SETUP configures the scanner. The scanner must be in Program mode to execute this command. This is normally the first command sent to the scanner.



parameters The VME master processor writes 13 (hex) to the command byte. Then the VME master processor writes these parameters to the first 12 bytes of the selected channel's general data area:

Byte offset:	Parameter:	Description:
0220	I/O link baud rate	Write one of these values (hex): 01 57.6 kbps (default) 02 115.2 kbps 03 230.4 kbps FF no change Values 00, 04–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0221	master processor watchdog timeout	Write one of these values (hex): 00 500 ms (default) 0A–FE (value x 100) ms (1000–25400ms) FF no change Values 01–09 are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0222	master processor watchdog disable	Write one of these values (hex): 01 disable watchdog timer 00, 02–FF enable watchdog timer (default) Disable the watchdog timer when you want to run a master processor application in debug mode without incurring timeouts that disrupt the application. The default is debug disabled.
0223	command complete interrupt level	Write one of these values (hex): 01 level 1 05 level 5 02 level 2 06 level 6 03 level 3 07 level7 04 level 4 (default) Values outside the range 01–07 are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0224	command complete interrupt status ID	The VME master processor writes an 8-bit value here that the scanner will pass to the VME interrupt handler during the VMEbus interrupt cycle. The default value is 40H.
0225	command complete interrupt enable	Write one of these values (hex): 00 generate VMEbus interrupt upon command completion (default) 01 do not generate VMEbus interrupt upon command completion FF no change Values 02–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0226	SYSFAIL monitor enable	Write one of these values (hex): 00 SYSFAIL monitor enabled (default) 01 SYSFAIL monitor disabled FF no change Values 02–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0227	input image table monitor interrupt level	Write one of these values (hex): 01 level 1 05 level 5 02 level 2 06 level 6 03 level 3 07 level7 04 level 4 (default) Values outside the range 01–07 are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
0228	input image table monitor interrupt status ID	The VME master processor writes an 8-bit value here that the scanner will pass to the VME interrupt handler in response to an input image table monitor interrupt. The default value is 42H.
0229	input image table monitor enable	Write one of these values (hex): 00 input image table monitor enabled 01 input image table monitor disabled (default) FF no change Values 02–FE are reserved. If you write a reserved value to this field, the scanner writes error 11H in the confirmation status byte.
022A	input image table low bound	The low bound sets one boundary for a contiguous segment of the input image table that the scanner monitors for a change. The default low bound is rack 0, group 0. Legal values are 00H to the high bound. Values greater than the high bound result in error 11H in the confirmation status byte.
022B	input image table high bound	The high bound sets one boundary for a contiguous segment of the input image table that the scanner monitors for a change. The default high bound is rack 15, group 7. Legal values are from the low bound to 7EH. Values between 80–FF (hex) and values less than the low bound result in error 11H in the confirmation status byte.

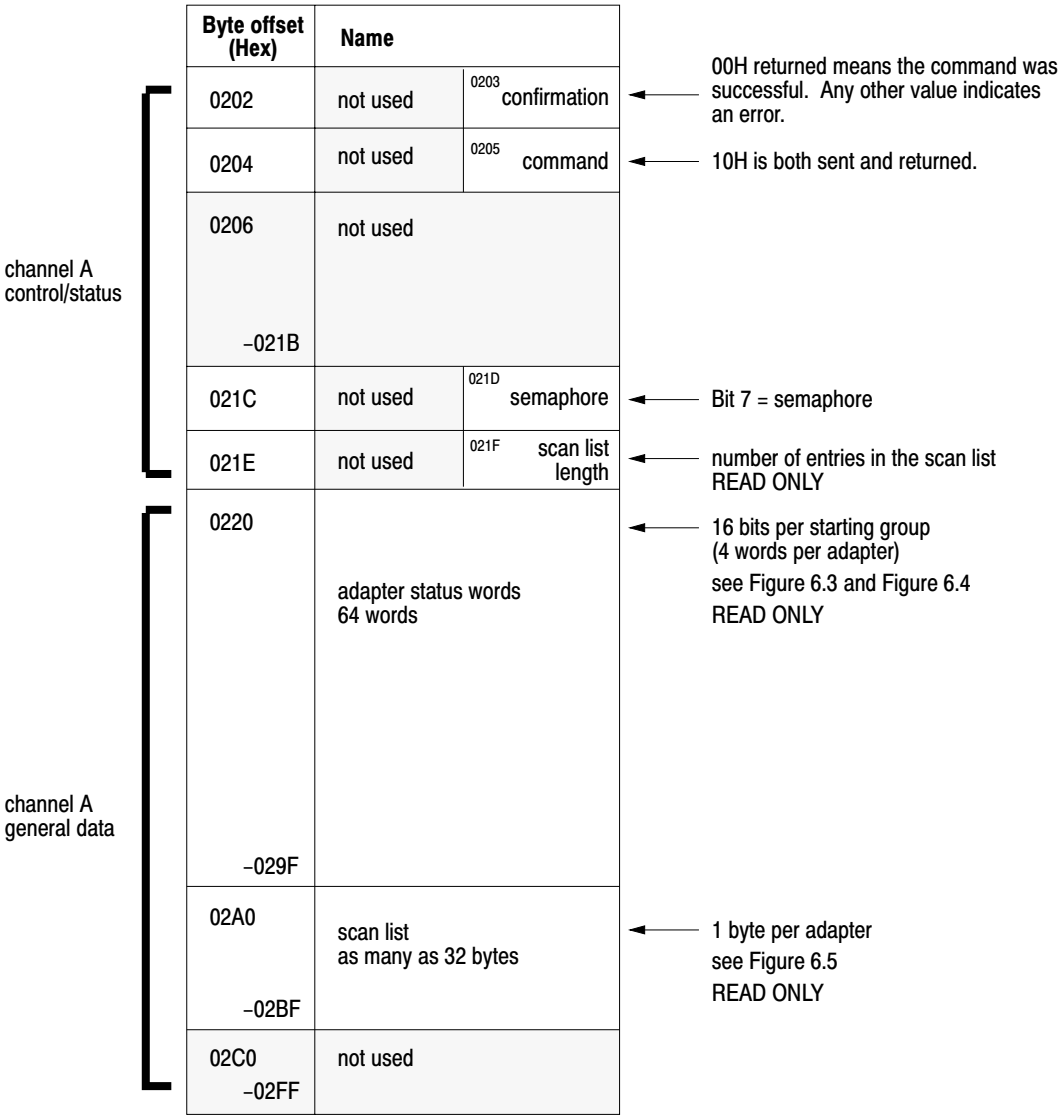
coding sequence Your code for the SETUP command should include these tasks:

1. get the semaphore
2. set up the general data area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

AUTOCONFIGURE

command byte 10

description AUTOCONFIGURE builds the scan list by polling every possible adapter address. Every adapter on the link that responds is placed once in the scan list. The scanner must be in Program mode to issue this command.



parameters The VME master processor writes 10 (hex) to the command byte. There are no input parameters from the VME master processor. The scanner writes these parameters to the selected channel's general data area:

Byte offset:	Parameter:	Description:
021F	scan list length READ ONLY	The scan list length indicates the number of entries in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area. You can enter a scan list length of 0.
0220	I/O adapter status word block READ ONLY	This word block contains four 16-bit entries (4 words) for each adapter – 16 bits for each starting group per adapter (maximum of 16 groups of entries). See Figure 6.3 and Figure 6.4.
02A0	scan list READ ONLY	This list is a maximum of 32 bytes long (1 byte per adapter). The list contains one byte-sized entry for each adapter found on the I/O link. See Figure 6.5 on page 6-14.

Figure 6.3
Format of the I/O adapter status word block and scan list

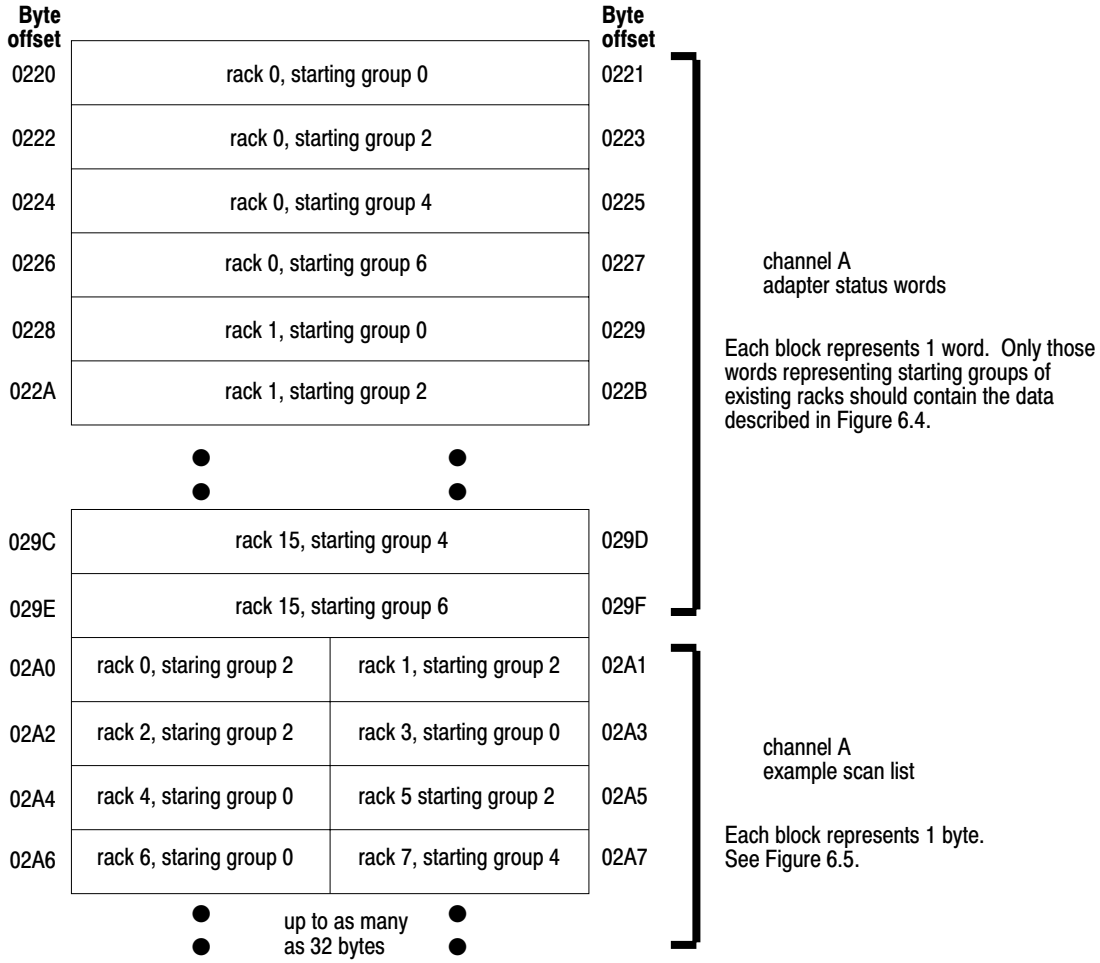
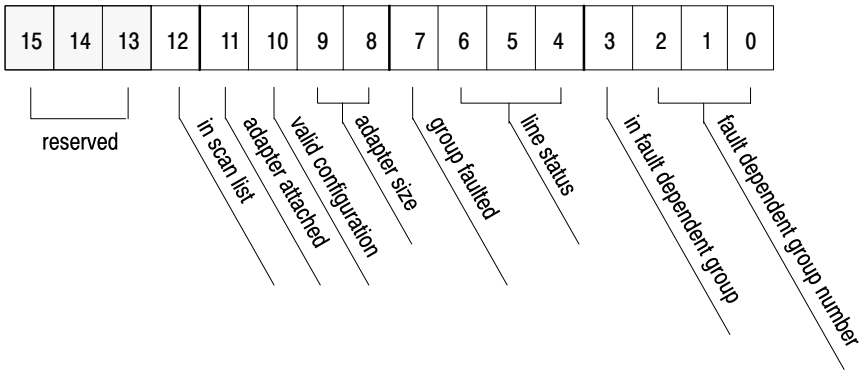
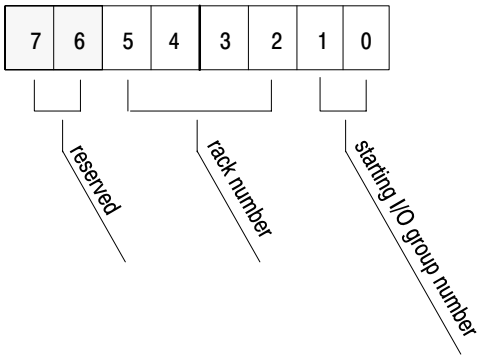


Figure 6.4
Format of one entry in the I/O adapter status word block



Format item:	Description:
in scan list bit 12	Contains one of these values: 1 adapter is in the current scan list 0 adapter not in current scan list
adapter attached bit 11	Contains one of these values: 1 adapter resides at this address 0 no adapter resides at this address
valid configuration bit 10	Contains one of these values: 1 adapter type, size, and address are valid 0 adapter type, size, and/or address not valid; adapter is improperly configured
adapter size bits 9–8	Contains one of these values (hex): 00 1/4 rack addressing 01 1/2 rack addressing 10 3/4 rack addressing 11 full rack addressing
group faulted bit 7	Contains one of these values: 1 fault exists in the fault dependent group associated with the adapter 0 no fault exists in the fault dependent group associated with the adapter
line status bit 6–4	Contains one of these values (hex): 000 adapter is off line any other value adapter is on line
in fault dependent group bit 3	Contains one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	The value (0–7 binary) identifies the fault dependent group to which this adapter belongs.

Figure 6.5
Format for one entry in the scan list



Format item:	Description:
rack number bits 5–2	The value (0–15 binary) identifies the rack number.
starting I/O group number bits 1–0	One of the following values identifies the starting I/O group: 00 group 0 01 group 2 10 group 4 11 group 6

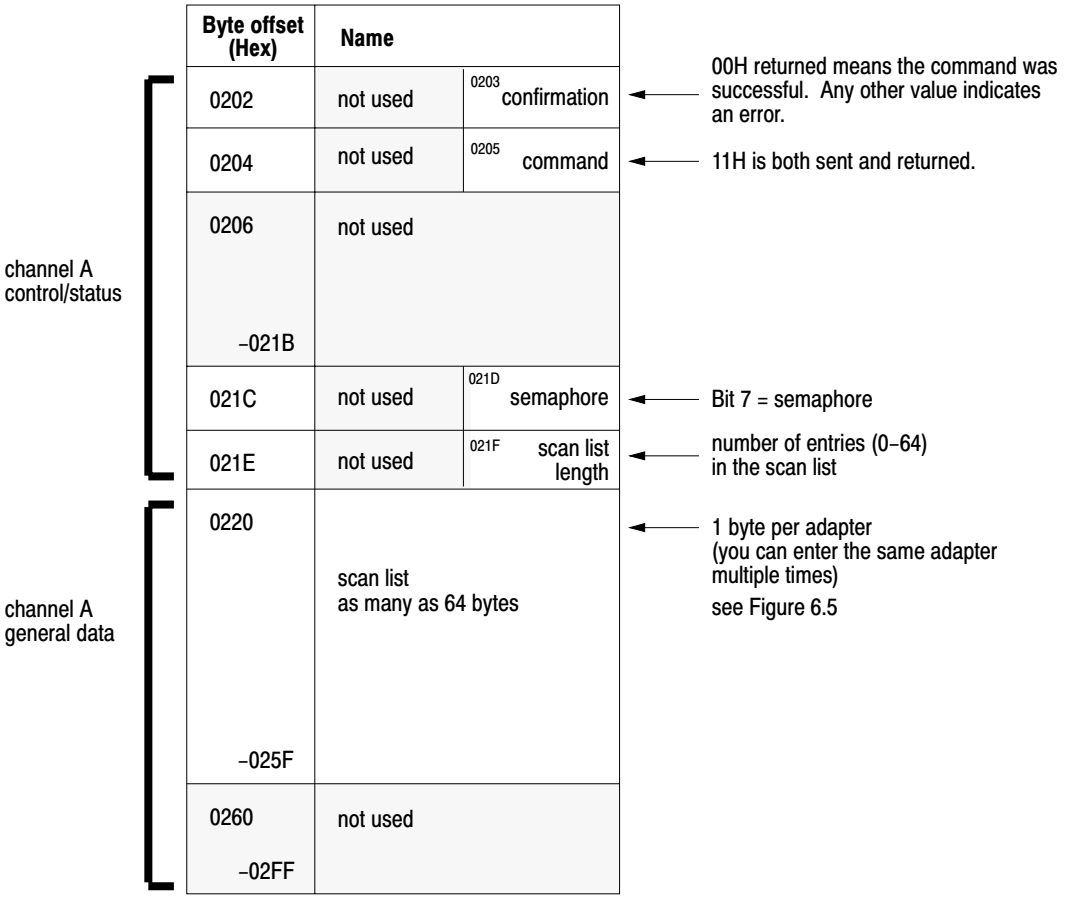
coding sequence Your code for the AUTOCONFIGURE command should include these tasks:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. check the result
6. clear the semaphore

SCAN LIST

command byte 11

description SCAN LIST replaces the current scan list. The scanner must be in Program mode. You can issue a SCAN LIST command without first issuing an AUTOCONFIGURE command.



parameters The VME master processor writes 11 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's general data area:

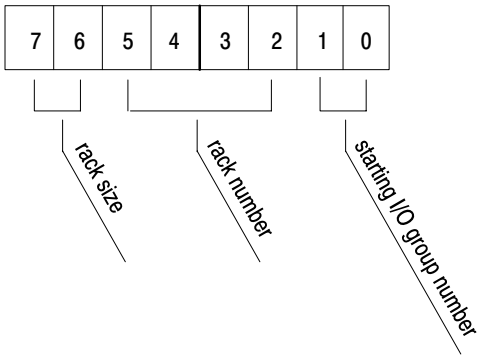
Byte offset:	Parameter:	Description:
021F	scan list length	The scan list length indicates the number of entries (0–64) in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area. You can enter a scan list length of 0.
0220	scan list	This list is a maximum of 64 bytes long (1 byte per entry), but it can contain only 32 distinct physical adapter addresses. The list contains one byte-sized entry for each adapter you want to place in the list. An adapter can appear in the list multiple times. See Figure 6.6 and Figure 6.7.

Figure 6.6
Example scan list

Byte offset			Byte offset
0220	rack 0, staring group 2	rack 1, starting group 2	0221
0222	rack 2, staring group 2	rack 3, starting group 0	0223
0224	rack 4, staring group 0	rack 5 starting group 2	0225
0226	rack 6, staring group 0	rack 7, starting group 4	0227

up to as many as 64 bytes

Figure 6.7
Format for one entry in the scan list



Format item:	Description:
rack size bits 7–6	Write one of these values: 00 1/4 rack 01 1/2 rack 10 3/4 rack 11 full rack These bits function differently than with the AUTOCONFIGURE command. With this command, you must specify the rack size you want – which could be larger than the actual rack size to allow for future expansion. The AUTOCONFIGURE command determines the actual rack size and writes the appropriate value.
rack number bits 5–2	Write the value (0–15 binary) of the rack number to be scanned.
starting I/O group number bits 1–0	Write one of these values: 00 group 0 01 group 2 10 group 4 11 group 6

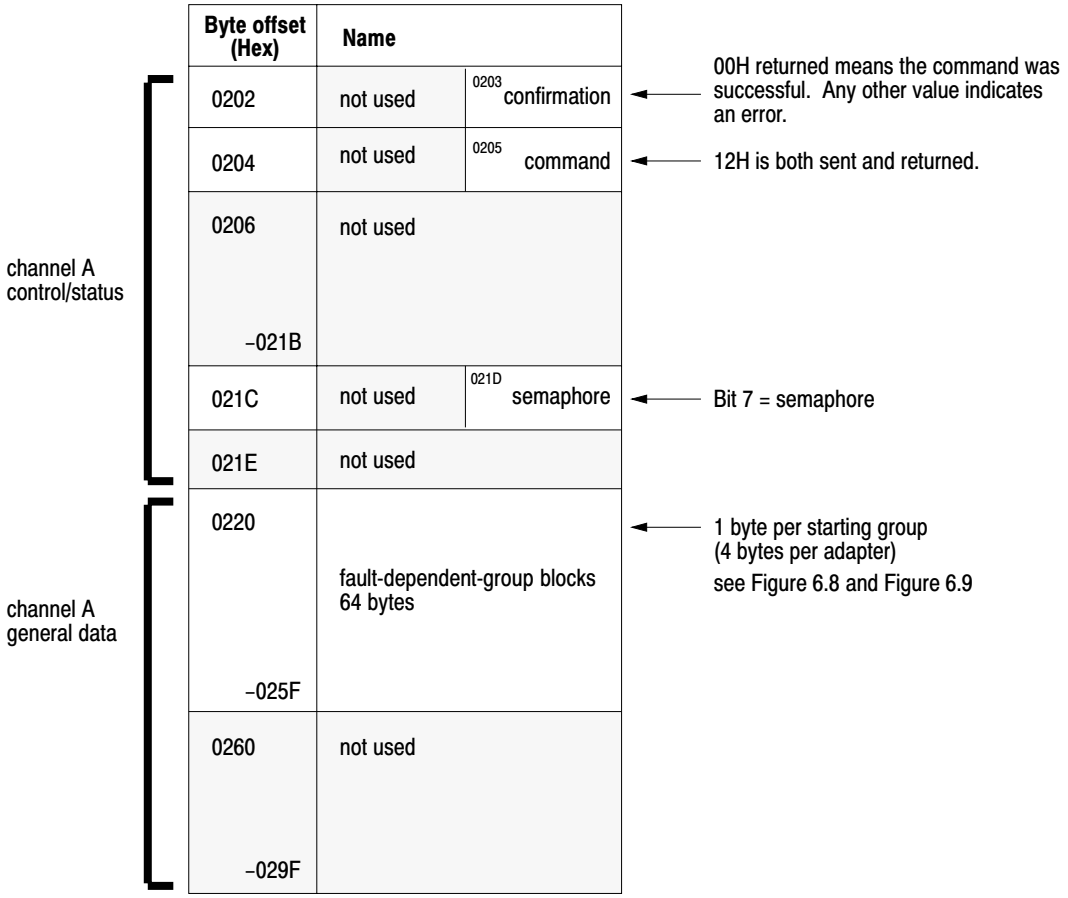
coding sequence Your code for the SCAN LIST command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the scan list to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. clear the semaphore

FAULT DEPENDENT GROUP

command byte 12

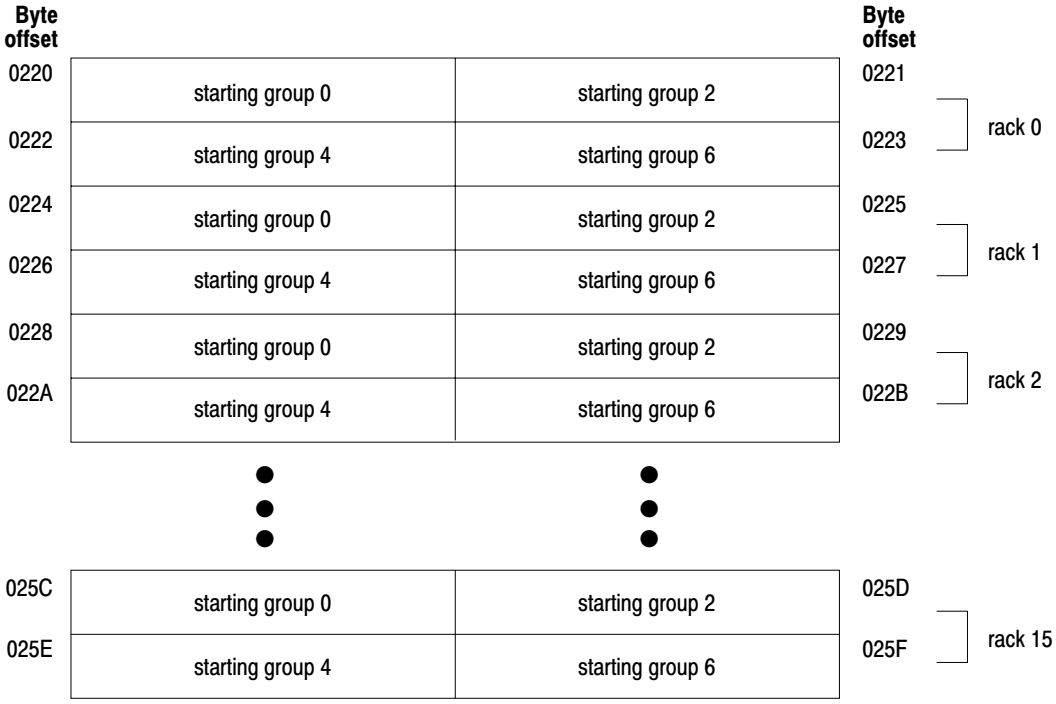
description FAULT DEPENDENT GROUP associates a set of adapters such that if one adapter in the group faults, all the other adapters in the group fault, going to their fail-safe mode of operation. The scanner must be in Program mode to issue this command. You can specify as many as 8 fault groups.



parameters The VME master processor writes 12 (hex) to the command byte. Then the VME master processor writes this parameter to the selected channel's general data area:

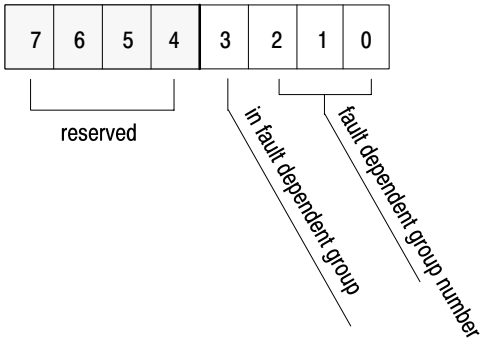
Byte offset:	Parameter:	Description:
0220	fault dependent group block	There are 4 configurable bits for each starting group entry. An adapter must be in the current scan list to be assigned to a fault-dependent group. If you specify an adapter that is not in the scan list, the scanner writes an illegal configuration error (16H) to the confirmation status byte. See Figure 6.8 and Figure 6.9.

Figure 6.8
Format of the fault dependent group block



Each block represents 1 byte. There is 1 byte for each starting group for each adapter. Only those entries representing starting groups of existing racks should contain the data described in Figure 6.9. Zero all entries for which adapters do not exist.

Figure 6.9
Format for one entry in the fault dependent group block



Format item:	Description:
in fault dependent group bit 3	Write one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	The value (0–7 binary) identifies the fault dependent group to which this adapter belongs.

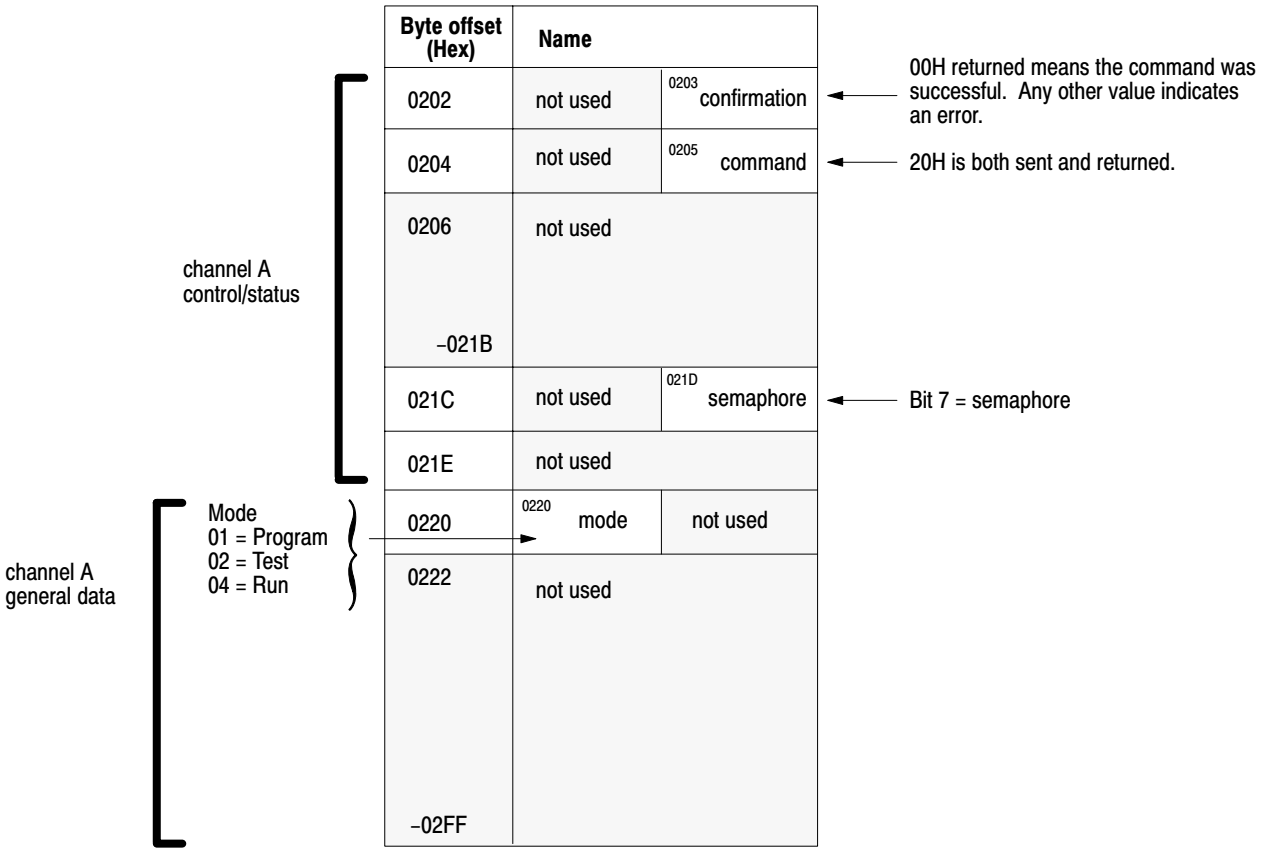
coding sequence Your code for the FAULT DEPENDENT GROUP command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the fault dependent group data to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. clear the semaphore

SET MODE

command byte 20

description SET MODE changes the operating mode of the scanner.



parameters The VME master processor writes 20 (hex) to the command byte. Then the VME master processor writes this parameter to the selected channel's general data area:

Byte offset:	Parameter:	Description:
0220	mode	Write one of these values (hex): 01 Program mode 02 Test mode 04 Run mode Values 00, 03, and 05–FF are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte.

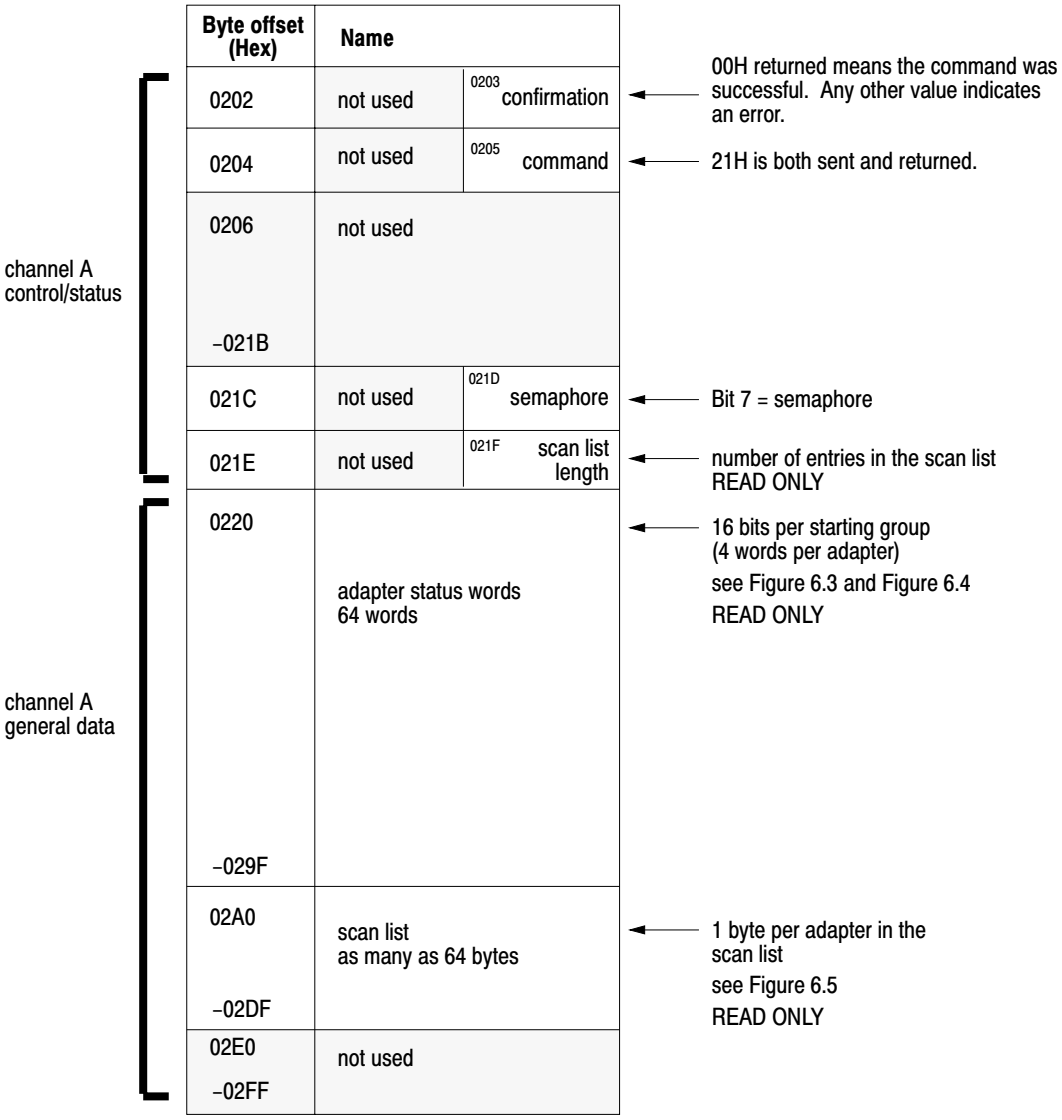
coding sequence Your code for the SET MODE command should include these tasks:

1. get the semaphore
2. set up the control status area
3. copy the mode value to the general data area
4. send the command interrupt
5. wait for the result
   (either poll for confirmation status or wait for an interrupt)
6. clear the semaphore

LINK STATUS

command byte 21

description LINK STATUS determines the current status of the adapters on the selected channel's I/O link. LINK STATUS doesn't affect the scanner. The VME master processor can issue LINK STATUS any time.



parameters The VME master processor writes 21 (hex) to the command byte. There are no input parameters from the VME master processor. The scanner writes these parameters to the selected channel's general data area:

Byte offset:	Parameter:	Description:
021F	scan list length READ ONLY	The scan list length indicates the number of entries in the scan list. The scan list length is an 8-bit quantity that the scanner writes to the length of data byte in the control/status area. You can enter a scan list length of 0.
0220	I/O adapter status word block READ ONLY	This word block contains four 16-bit entries (4 words) for each adapter – 16 bits for each starting group per adapter (maximum of 16 groups of entries). See Figure 6.10 and Figure 6.11.
02A0	scan list READ ONLY	This list is a maximum of 64 bytes long (1 byte per adapter)), but it contains only 32 distinct physical adapter addresses.. The list contains one byte-sized entry for each adapter in the scan list. An adapter can appear in the list multiple times. See Figure 6.12 on page 6-26.

Figure 6.10
Format of the I/O adapter status word block and scan list

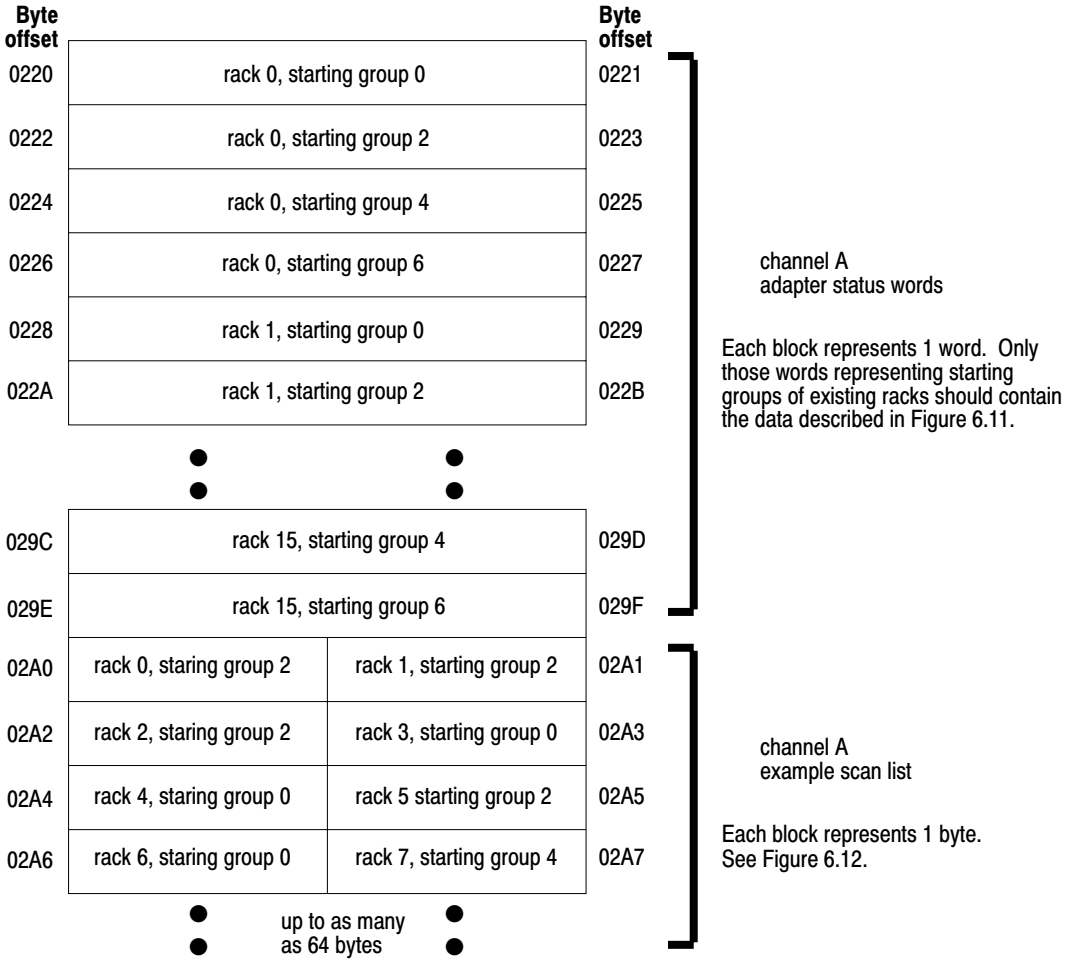
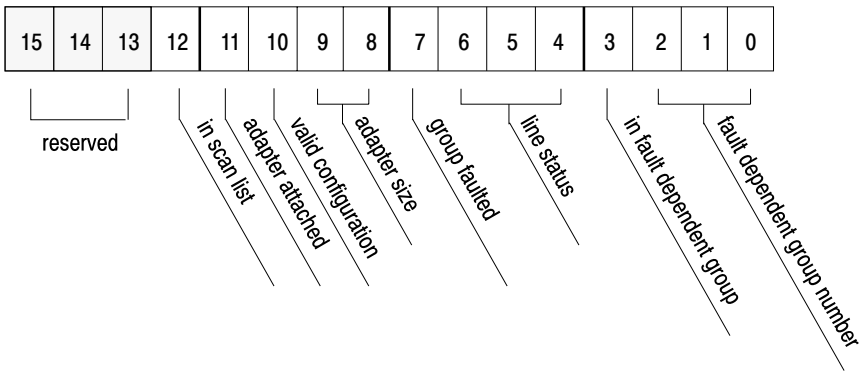
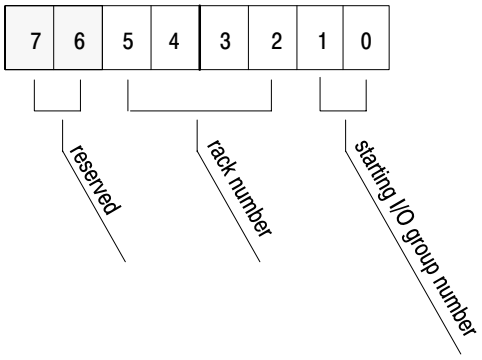


Figure 6.11
Format of one entry in the I/O adapter status word block



Format item:	Description:
in scan list bit 12	Contains one of these values: 1 adapter is in the current scan list 0 adapter not in current scan list
adapter attached bit 11	Contains one of these values: 1 adapter resides at this address 0 no adapter resides at this address
valid configuration bit 10	Contains one of these values: 1 adapter type, size, and address are valid 0 adapter type, size, and/or address not valid; adapter is improperly configured
adapter size bits 9–8	Contains one of these values (hex): 00 1/4 rack addressing 01 1/2 rack addressing 10 3/4 rack addressing 11 full rack addressing
group faulted bit 7	Contains one of these values: 1 fault exists in the fault dependent group associated with the adapter 0 no fault exists in the fault dependent group associated with the adapter
line status bit 6–4	Contains one of these values (hex): 000 adapter is off line any other value adapter is on line
in fault dependent group bit 3	Contains one of these values: 1 adapter is in the fault dependent group identified by the fault dependent group number (bits 2–0) 0 adapter is not in a fault dependent group
fault dependent group number bits 2–0	The value (0–7 binary) identifies the fault dependent group to which this adapter belongs.

Figure 6.12
Format for one entry in the scan list



Format item:	Description:
rack number bits 5–2	The value (0–15 binary) identifies the rack number.
starting I/O group number bits 1–0	One of the following values identifies the starting I/O group: 00 group 0 01 group 2 10 group 4 11 group 6

coding sequence The following diagram shows one method for applying the LINK STATUS command:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. check the result
6. clear the semaphore

BT READ

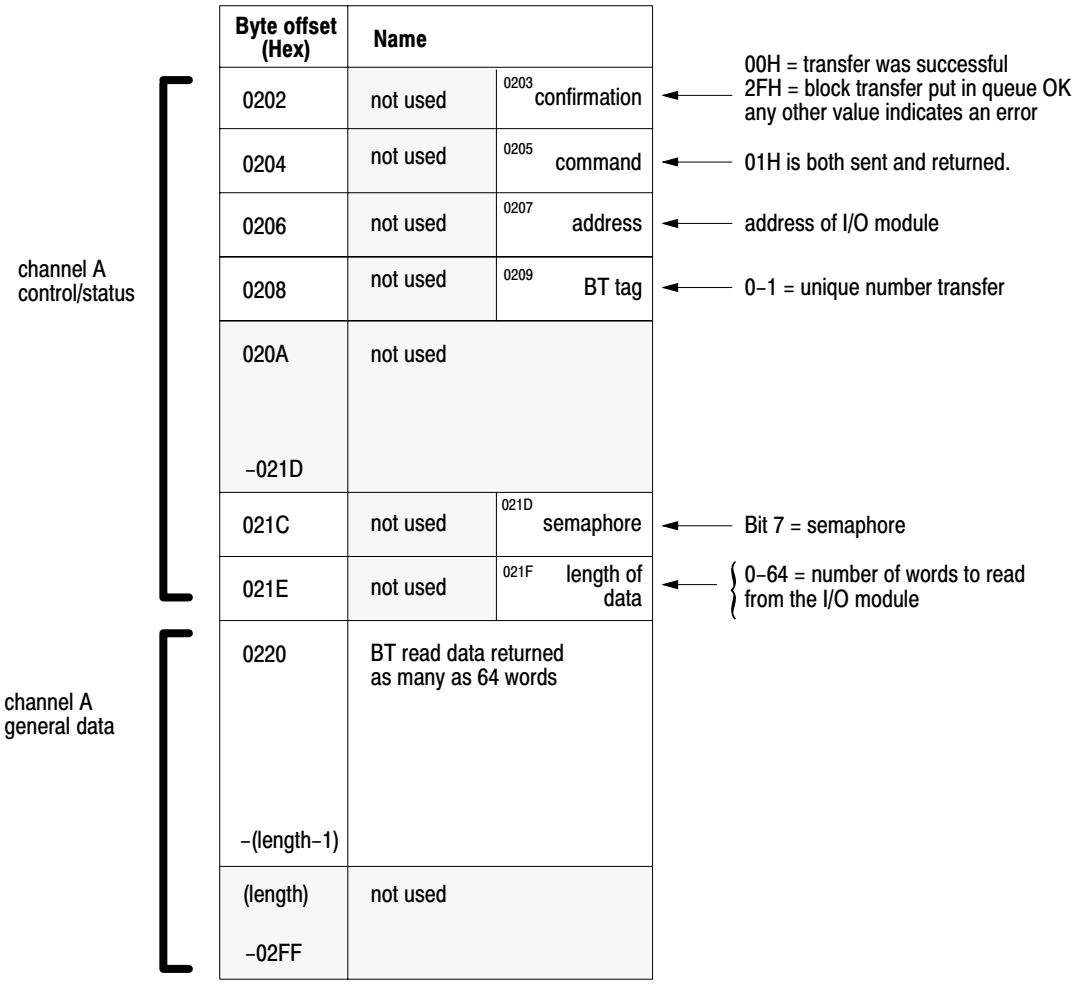
command byte 01

description BT READ transfers a block of data from specified I/O module into the scanner. The scanner queues at most one block-transfer request – write or read.

In SV-superset mode, each scanner channel can queue only one block-transfer request. If you need to send multiple block-transfer requests, use the continuous block-transfer commands. See pages 6-40 and 6-36.

Upon issuing a block-transfer request, the scanner writes 0x2F in the confirmation status byte to indicate that the scanner accepted the request and put it into its queue. When the scanner actually completes the block-transfer, the scanner sets the semaphore and then it updates this data in the following order:

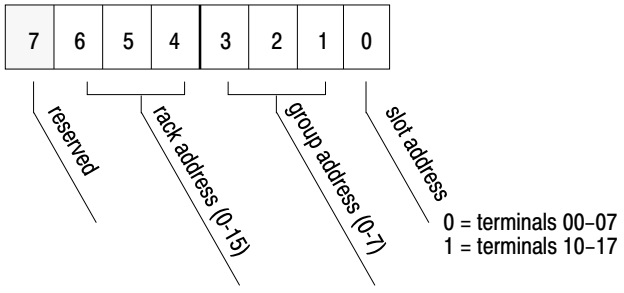
1. address
2. block transfer tag
3. length of data
4. confirmation status
5. block transfer read data
6. command complete interrupt (if enabled)


channel A
control/status
channel A
general data
Byte offset (Hex)
Name
0202
not used
0203 confirmation
0204
not used
0205 command
0206
not used
0207 address
0208
not used
0209 BT tag
020A
–021D
not used
021C
not used
021D semaphore
021E
not used
021F length of data
0220
–(length–1)
BT read data returned as many as 64 words
(length)
–02FF
not used
00H = transfer was successful
2FH = block transfer put in queue OK
any other value indicates an error
01H is both sent and returned.
address of I/O module
0–1 = unique number transfer
Bit 7 = semaphore
0–64 = number of words to read from the I/O module

parameters The VME master processor writes 01 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's control status area:

Byte offset:	Parameter:	Description:
0207	module address	The module address is the address of the I/O module from which to read the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 6.13.
0209	block-transfer tag number	The block-transfer tag number is an 8-bit integer that uniquely identifies each block-transfer. The scanner writes a value of 0 or 1 to the block-transfer request tag number in the control/status area. Values greater than 1 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte. This tag is returned with the status when the block transfer is complete so the VME master processor can match the block transfer status with the request.
021F	length of data byte	The length of data byte is an 8-bit value that specifies the number of 16-bit words (0–64 decimal) to be read from the target I/O module. Use the value 0 to let the I/O module determine how many words the scanner can read. When the block-transfer completes, the 0 is replaced by the actual number of words read. Values greater than 64 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte.
0220	BT data	The BT data is the set of data words (0–64) read from the target I/O module. The BT data block ends at offset (length of data – 1).

Figure 6.13
Format for the address byte



coding sequence Your code for the BT READ command should include these tasks:

1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

BT WRITE

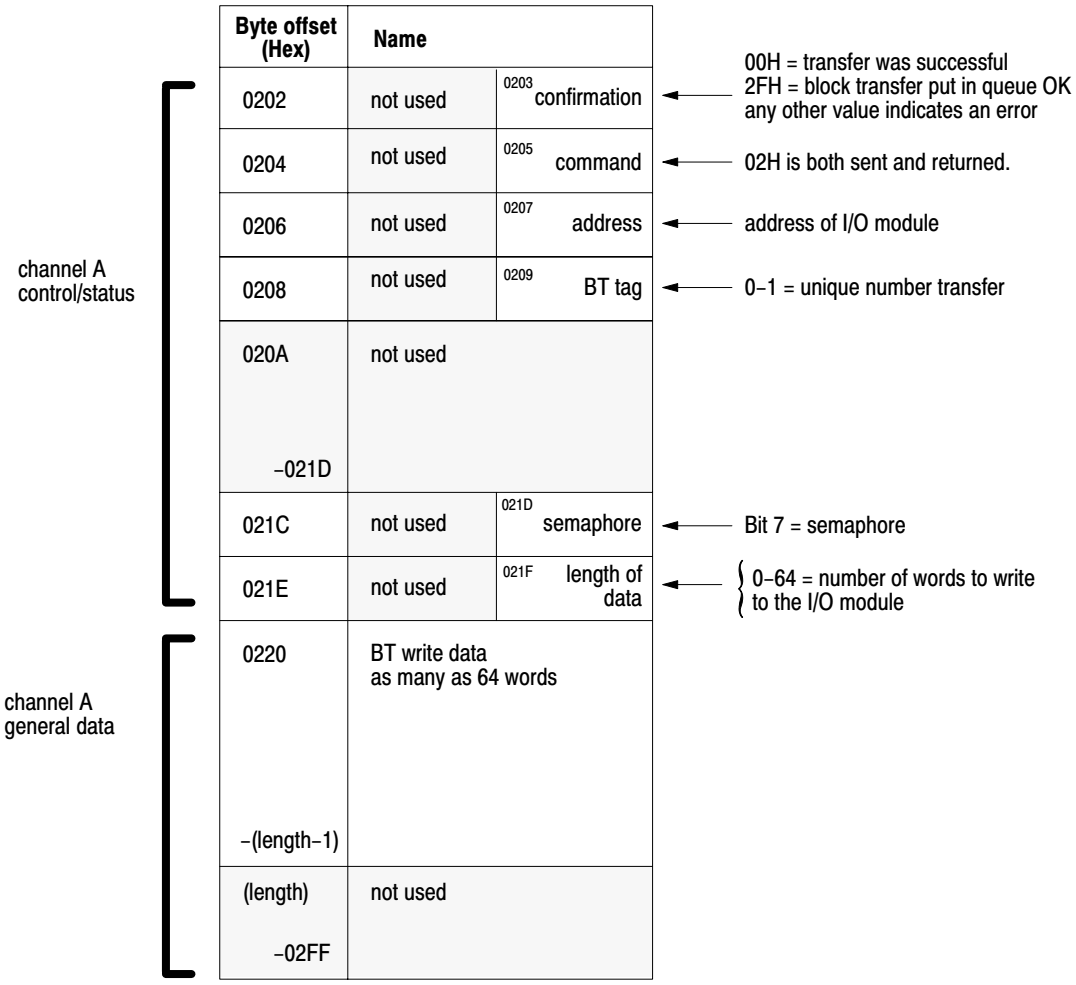
command byte 02

description BT WRITE transfers a block of data from the scanner to the specified I/O module. The scanner queues at most one block-transfer request – write or read.

In SV-superset mode, each scanner channel can queue only one block-transfer request. If you need to send multiple block-transfer requests, use the continuous block-transfer commands. See pages 6-40 and 6-36.

Upon issuing a block-transfer request, the scanner writes 0x2F in the confirmation status byte to indicate that the scanner accepted the request and put it into its queue. When the scanner actually completes the block-transfer, the scanner sets the semaphore and then it updates this data in the following order:

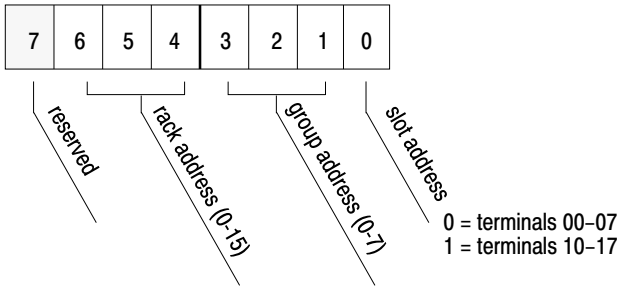
1. address
2. block transfer tag
3. length of data
4. confirmation status
5. block transfer read data
6. command complete interrupt (if enabled)


channel A
control/status
channel A
general data
Byte offset (Hex)
Name
0202
not used
0203 confirmation
0204
not used
0205 command
0206
not used
0207 address
0208
not used
0209 BT tag
020A
–021D
not used
021C
not used
021D semaphore
021E
not used
021F length of data
0220
–(length–1)
BT write data
as many as 64 words
(length)
–02FF
not used
00H = transfer was successful
2FH = block transfer put in queue OK
any other value indicates an error
02H is both sent and returned.
address of I/O module
0–1 = unique number transfer
Bit 7 = semaphore
0–64 = number of words to write
to the I/O module

parameters The VME master processor writes 02 (hex) to the command byte. Then the VME master processor writes these parameters to the selected channel's control status area:

Byte offset:	Parameter:	Description:
0207	module address	The module address is the address of the I/O module that is to receive the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 6.14.
0209	block-transfer tag number	The block-transfer tag number is an 8-bit integer that uniquely identifies each block-transfer. The scanner writes a value of 0 or 1 to the block-transfer request tag number in the control/status area. Values greater than 1 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte. This tag is returned with the status when the block transfer is complete so the VME master processor can match the block transfer status with the request.
021F	length of data block	The length of data byte is an 8-bit value that specifies the number of 16-bit words (0–64 decimal) to be written to the target I/O module. Use the value 0 to let the I/O module determine how many words the scanner can write. When the block-transfer completes, the 0 is replaced by the actual number of words written. Values greater than 64 are reserved. If you write a reserved value to this field, the scanner writes an illegal confirmation error (16H) in the confirmation status byte.
0220	BT data	The BT data is the set of data words (0–64) sent to the target I/O module. The BT data block ends at offset (length of data – 1).

Figure 6.14
Format for the address byte



coding sequence Your code for the BT WRITE command should include these tasks:

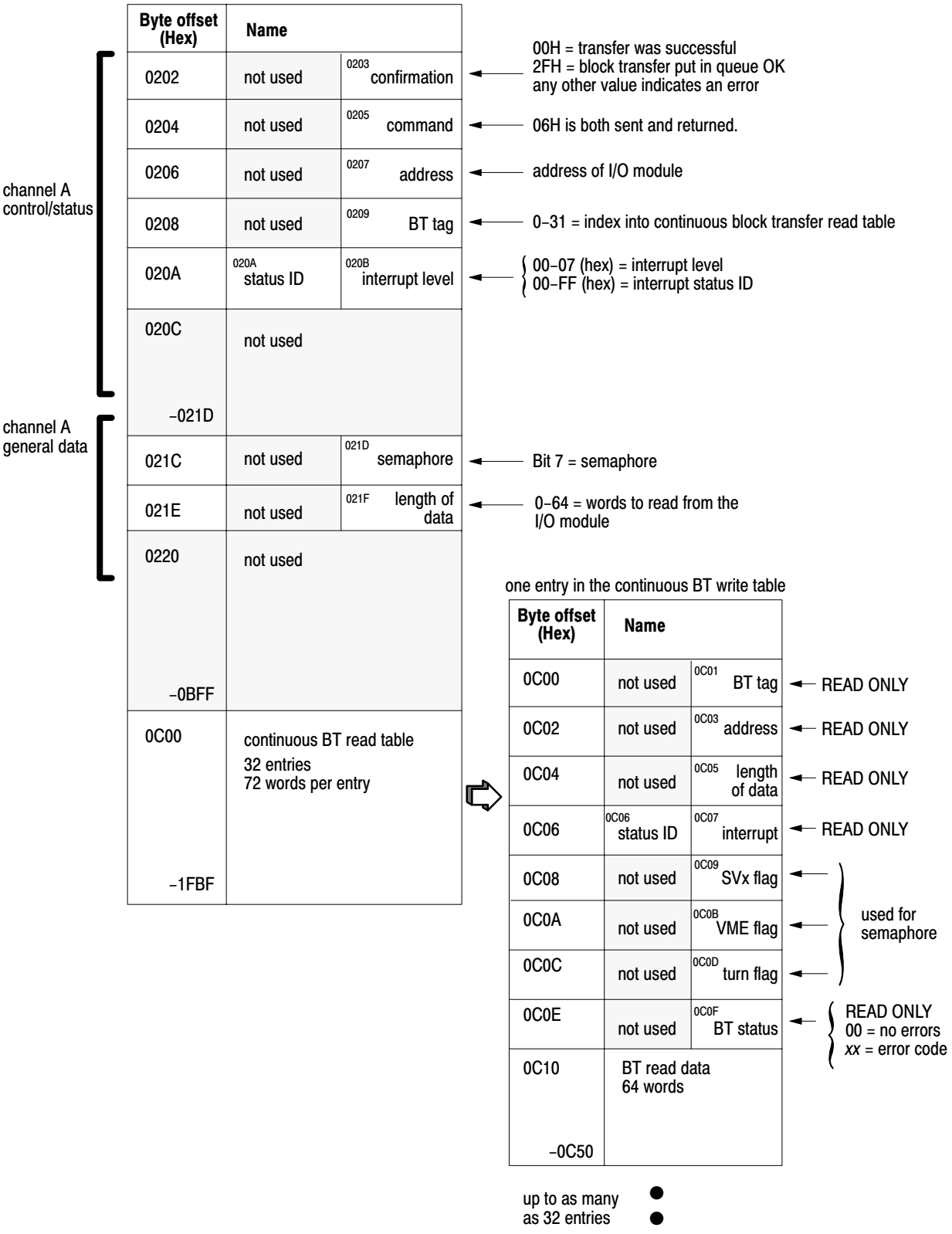
1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

CONTINUOUS BT READ

command byte 06

description CONTINUOUS BT READ transfers a block of data from the specified adapter to the scanner's general data area at regular intervals. When one block-transfer read completes, the scanner queues another one. The cycle continues until you disable the continuous block transfer.

The scanner must be in Program mode to execute this command. If the scanner isn't in Program mode, the scanner returns error code 13H (Program mode required) to the confirmation status byte. Once the command is running, you can halt a continuous block-transfer by switching the scanner to Program mode.


Byte offset (Hex)
Name
channel A control/status
channel A general data
0202
not used
0203 confirmation
00H = transfer was successful
2FH = block transfer put in queue OK
any other value indicates an error
0204
not used
0205 command
06H is both sent and returned.
0206
not used
0207 address
address of I/O module
0208
not used
0209 BT tag
0–31 = index into continuous block transfer read table
020A
020A status ID
020B interrupt level
00–07 (hex) = interrupt level
00–FF (hex) = interrupt status ID
020C
–021D
not used
021C
not used
021D semaphore
Bit 7 = semaphore
021E
not used
021F length of data
0–64 = words to read from the I/O module
0220
–0BFF
not used
0C00
–1FBF
continuous BT read table
32 entries
72 words per entry
one entry in the continuous BT write table
Byte offset (Hex)
Name
0C00
not used
0C01 BT tag
READ ONLY
0C02
not used
0C03 address
READ ONLY
0C04
not used
0C05 length of data
READ ONLY
0C06
0C06 status ID
0C07 interrupt
READ ONLY
0C08
not used
0C09 SVx flag
0C0A
not used
0C0B VME flag
0C0C
not used
0C0D turn flag
used for semaphore
0C0E
not used
0C0F BT status
READ ONLY
00 = no errors
xx = error code
0C10
–0C50
BT read data
64 words
up to as many as 32 entries

parameters The VME master processor writes 06 (hex) to the command byte. Then the VME master processor writes these parameters to the control status area of the selected channel:

Byte offset:	Parameter:	Description:
0207	continuous block-transfer address	The module address is the address of the I/O module from which to read the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 6.15 on page 6-39.
0209	continuous block-transfer tag	The VME master processor writes a number (0–31) that is an index into the continuous block transfer read table. Only the lower byte is used.
020A	continuous block-transfer status ID	Specifies the interrupt status ID used when the block-transfer is completed. See Figure 6.16 on page 6-39.
020B	continuous block-transfer interrupt level	Specifies the VME interrupt level used when the block-transfer is completed. See Figure 6.16 on page 6-39.
021F	continuous block-transfer length of data	The VME master processor writes the number of words of continuous block-transfer data (0–64). Use the value 0 to let the I/O module determine how many words the scanner can read. When the block-transfer completes, the 0 is replaced by the actual number of words read.
0C01	continuous block-transfer tag READ ONLY	configure this parameter at byte offset 0209 (see above)
0C03	continuous block-transfer address READ ONLY	configure this parameter at byte offset 0207 (see above)
0C05	continuous block-transfer length of data READ ONLY	configure this parameter at byte offset 021F (see above)
0C06	continuous block-transfer status ID READ ONLY	configure this parameter at byte offset 020A (see above)
0C07	continuous block-transfer interrupt level READ ONLY	configure this parameter at byte offset 020B (see above)
0C09	continuous block-transfer scanner flag	Used for Peterson semaphore. See sample code in chapter 8.
0C0B	continuous block-transfer VME flag	Used for Peterson semaphore. See sample code in chapter 8.
0C0D	continuous block-transfer turn flag	Used for Peterson semaphore. See sample code in chapter 8.
0C0F	continuous block-transfer status READ ONLY	Indicates the status of the continuous block-transfer. 0100 = block transfer disabled 0200 = block transfer pending 0400 = block transfer sent 0800 = block transfer received
0C10–0C50	continuous block-transfer write/read data (64 words)	Contains as many as 64 words of write or read data

Use the SV2 flag, VME flag, and turn flag to determine who has access to the continuous block-transfer write table. If the scanner has the lock so that it can update the block data, the scanner copies 64 words at 6.4 msecs before releasing the lock. The VME master processor would at most busy wait 7 msecs before it could acquire the lock.

Figure 6.15
Format for the continuous block-transfer address byte

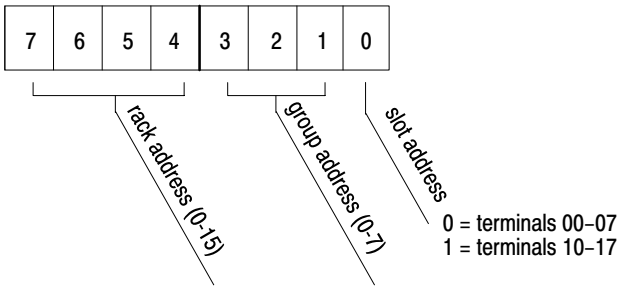
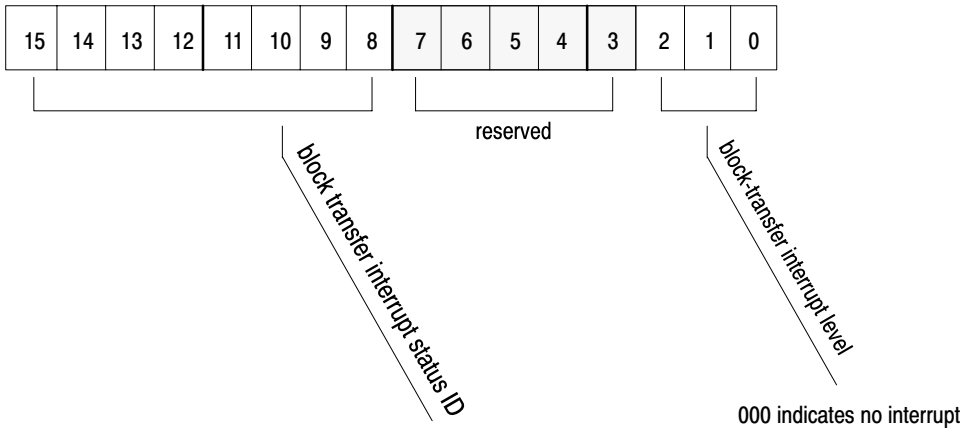


Figure 6.16
Format for the continuous block-transfer interrupt level and status ID



coding sequence

Your code for the CONTINUOUS BT READ command should include these tasks:

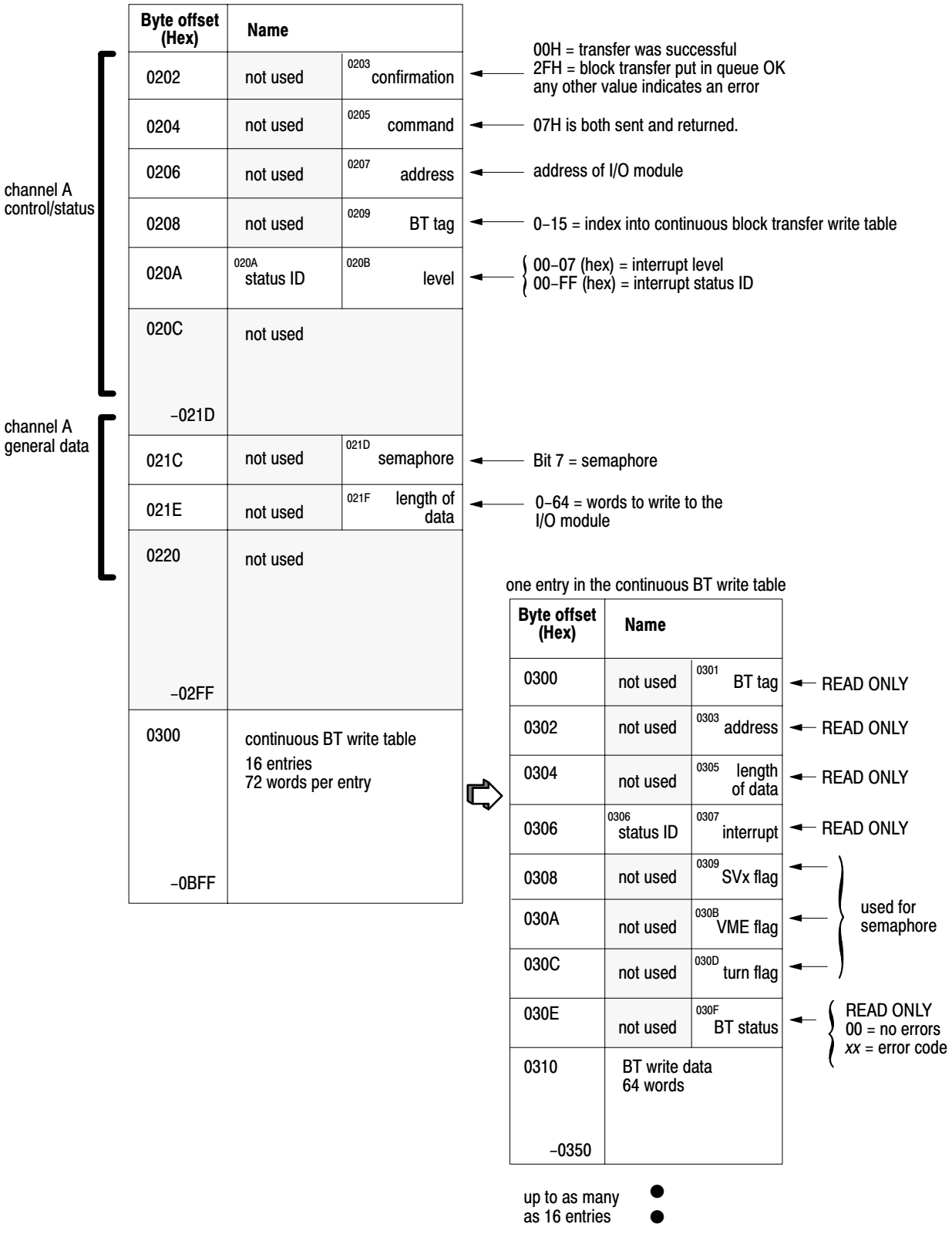
1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

CONTINUOUS BT WRITE

command byte 07

description CONTINUOUS BT WRITE transfers a block of data from the scanner to the specified I/O module at regular intervals. When one block-transfer write completes, the scanner queues another one. The cycle continues until you disable the continuous block transfer.

The scanner must be in Program mode to execute this command. If the scanner isn't in Program mode, the scanner returns error code 13 (Program mode required) to the confirmation status byte. Once this command is running, you can halt a continuous block-transfer by switching the scanner to Program mode.


Byte offset (Hex)
Name
channel A control/status
channel A general data
0202
not used
0203 confirmation
00H = transfer was successful
2FH = block transfer put in queue OK
any other value indicates an error
0204
not used
0205 command
07H is both sent and returned.
0206
not used
0207 address
address of I/O module
0208
not used
0209 BT tag
0–15 = index into continuous block transfer write table
020A
020A status ID
020B level
00–07 (hex) = interrupt level
00–FF (hex) = interrupt status ID
020C
–021D
not used
021C
not used
021D semaphore
Bit 7 = semaphore
021E
not used
021F length of data
0–64 = words to write to the I/O module
0220
–02FF
not used
0300
–0BFF
continuous BT write table
16 entries
72 words per entry
one entry in the continuous BT write table
Byte offset (Hex)
Name
0300
not used
0301 BT tag
READ ONLY
0302
not used
0303 address
READ ONLY
0304
not used
0305 length of data
READ ONLY
0306
0306 status ID
0307 interrupt
READ ONLY
0308
not used
0309 SVx flag
030A
not used
030B VME flag
030C
not used
030D turn flag
used for semaphore
030E
not used
030F BT status
READ ONLY
00 = no errors
xx = error code
0310
–0350
BT write data
64 words
up to as many as 16 entries

parameters The VME master processor writes 07 (hex) to the command byte. Then the VME master processor writes these parameters to the control status area of the selected channel:

Byte offset:	Parameter:	Description:
0207	continuous block-transfer address	The module address is the address of the I/O module that is to receive the block data. It contains the rack (0–15), group (0–7), and slot (0 or 1) numbers of the I/O module. See Figure 6.17 on page 6-43
0209	continuous block-transfer tag	The VME master processor writes a number (0–15) that is an index into the continuous block transfer write table. Only the lower byte is used.
020A	continuous block-transfer status ID	Specifies the VME status ID used when the block-transfer is completed. See Figure 6.18 on page 6-43.
020B	continuous block-transfer interrupt level	Specifies the VME interrupt level used when the block-transfer is completed. See Figure 6.18 on page 6-43.
021F	continuous block-transfer length of data	The VME master processor writes the number of words of continuous block-transfer data (0–64). Use the value 0 to let the I/O module determine how many words the scanner can write. When the block-transfer completes, the 0 is replaced by the actual number of words written.
0301	continuous block-transfer tag READ ONLY	configure this parameter at byte offset 0209 (see above)
0303	continuous block-transfer address READ ONLY	configure this parameter at byte offset 0207 (see above)
0305	length of data READ ONLY	configure this parameter at byte offset 021F (see above)
0306	continuous block-transfer status ID READ ONLY	configure this parameter at byte offset 020A (see above)
0307	continuous block-transfer interrupt level READ ONLY	configure this parameter at byte offset 020B (see above)
0309	continuous block-transfer scanner flag	Used for Peterson semaphore. See sample code in chapter 8.
030B	continuous block-transfer VME flag	Used for Peterson semaphore. See sample code in chapter 8.
030D	continuous block-transfer turn flag	Used for Peterson semaphore. See sample code in chapter 8.
030F	continuous block-transfer status READ ONLY	Indicates the status of the continuous block-transfer. 0100 = block transfer disabled 0200 = block transfer pending 0400 = block transfer sent 0800 = block transfer received
0310–0350	continuous block-transfer write/read data (64 words)	Contains as many as 64 words of write or read data

Use the SV2 flag, VME flag, and turn flag to determine who has access to the continuous block-transfer write table. If the scanner has the lock so that it can update the block data, the scanner copies 64 words at 6.4 msecs before releasing the lock. The VME master processor would at most busy wait 7 msecs before it could acquire the lock.

Figure 6.17
Format for the continuous block-transfer address byte

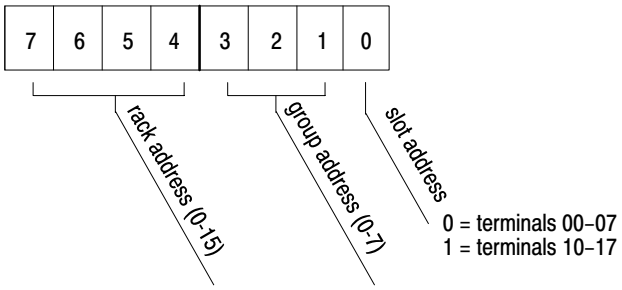
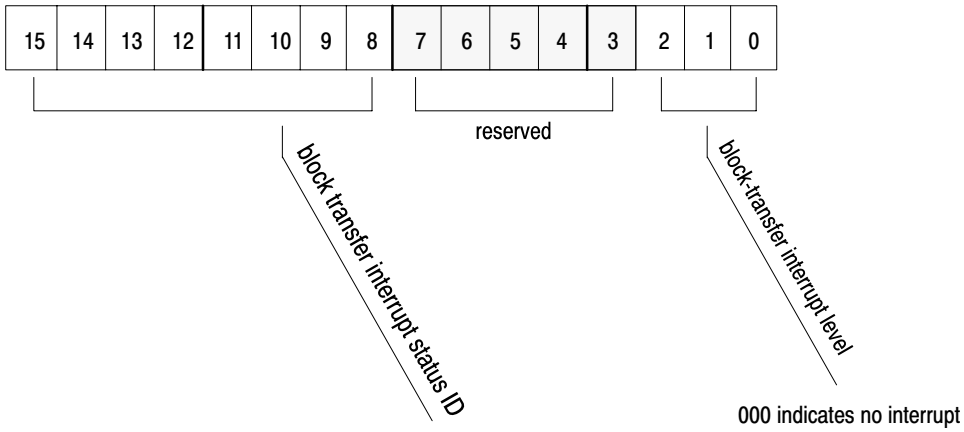


Figure 6.18
Format for the continuous block-transfer interrupt level and status ID



coding sequence

Your code for the CONTINUOUS BT WRITE command should include these tasks:

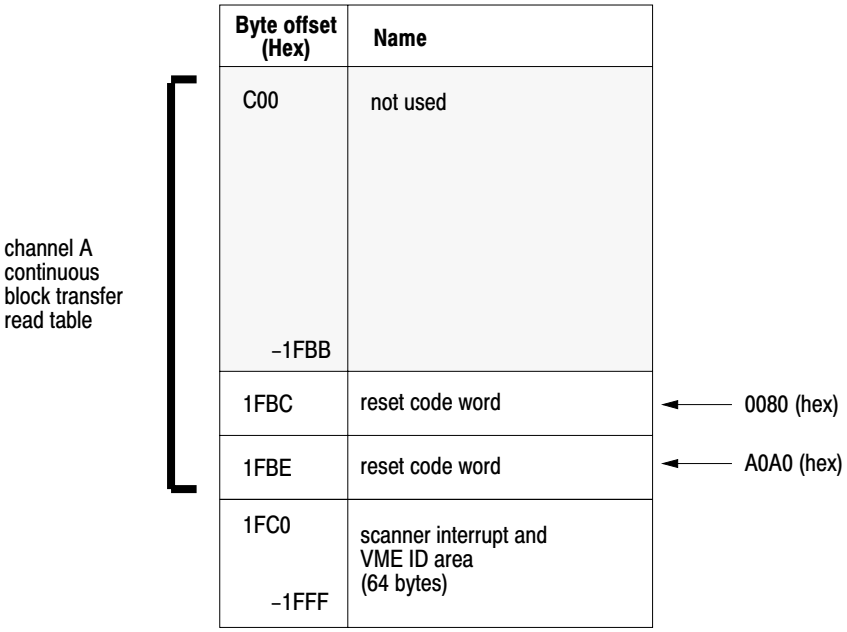
1. get the semaphore
2. set up the control status area
3. send the command interrupt
4. wait for the result
   (either poll for confirmation status or wait for an interrupt)
5. clear the semaphore

RESET

description RESET causes the scanner to reset itself. The VME master processor can issue RESET any time. When the scanner stops running due to another VME module asserting SYSFAIL, you can use RESET to reset the scanner, which causes the scanner to re-initialize itself.

RESET looks for the appropriate values in the 2 words right before the VME status ID area. If the values are there, the scanner resets itself, performs its power-on self-test, and enters the SLEEP state. The application program has to wake up the scanner and then send a SETUP command to configure the scanner (see chapter 8 for an example). This applies to either channel.

During the RESET, the scanner stops scanning the remote I/O and all three LEDs on the scanner light. RESET is different than a power cycle to the scanner in that with RESET, the scanner doesn't assert SYSFAIL during the self tests and the scanner doesn't clear the I/O image tables. The scanner leaves the I/O image tables in their last state. The scanner then enters the SLEEP state.



parameters There are no parameters for the RESET command. Before the scanner executes the RESET command, the VME master processor writes these parameters to the 2 words before the VME ID area:

Byte offset:	Parameter:	Description:
1FBC	second-to-last word in the selected channel's general data area	Write the value 0080 (hex) to this word.
1FBE	last word in the selected channel's general data area	Write the value A0A0 (hex) to this word.

coding sequence Your code for the RESET command needs to write the above words to byte offsets 1FBC and 1FBE.

Chapter 7

Starting the Scanner

Using This Chapter

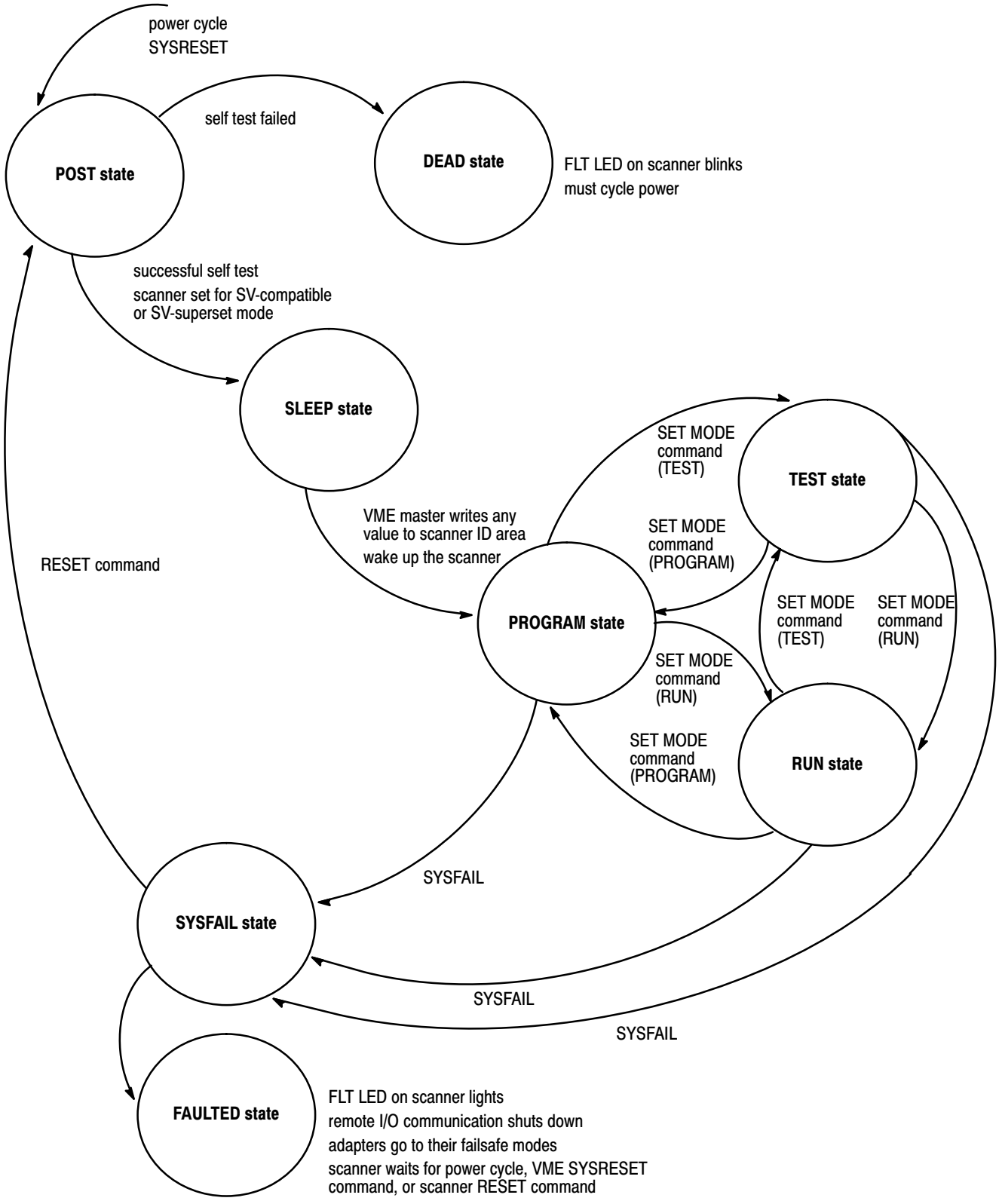
This chapter provides programming examples that illustrate the scanner management commands.

Important: The programming examples in this publication are examples only. They will need modification before they can run correctly in your application. These examples were developed for a Radisys EPC computer; they were written in Microsoft C version 7.00. To use these examples on other VME systems, search for and replace all “EPC” calls with the appropriate functions for your system.

Understanding the Scanner States

The scanner operates in one of several different states, depending on what commands you have sent the scanner, as well as the scanner’s health. The following figure shows the possible scanner states, the inputs that result in the states, and the states that follow a given state.

possible scanner states



The following table describes each scanner state.

In this state:	the scanner enters this state:
POST	at power-up, as a result of a VME SYSRESET, or as a result of the RESET command. In this state the scanner executes its self-test. The scanner ignores all commands or interrupts from VME processors and ignores all packets from the I/O link. If the self-test completes successfully, the scanner initializes global memory and sends status to the operating status word. In SV-compatible and SV-superset modes, the scanner enters the SLEEP state. If the scanner enters POST state from a VME SYSRESET or a RESET command, the self-test preserves the contents of the I/O image tables.
DEAD	when the self-test detects irrecoverable hardware faults. The scanner asserts VME SYSFAIL, enters an infinite loop, and ignores all commands or interrupts from VME processors and ignores all packets from the I/O link. Cycle power to leave the DEAD state.
SLEEP	when the self-test completes successfully and the scanner is in SV-compatible or SV-superset mode. In the SLEEP state, the scanner ignores all packets from the I/O links. The scanner doesn't leave this state until the VME master processor writes any value to the scanner interrupt/VME ID area. Then the scanner enters the PROGRAM state.
PROGRAM	when the scanner leaves the SLEEP state or when the scanner (in TEST or RUN state) receives a SET MODE command. In the PROGRAM state: • the scanner doesn't send output information to the adapters • all module outputs are held reset (off) • discrete input information is updated • the scanner doesn't send block transfer requests to the adapters, but the scanner will queue the requests from the VME master processor
TEST	when the scanner (in PROGRAM or RUN state) receives a SET MODE command. In the TEST state: • the scanner sends output information to the adapters • all module outputs are held reset (off) • discrete input information is updated • the scanner sends block transfer requests to the adapters, but actual outputs are held reset (off)
RUN	when the scanner (in PROGRAM or TEST state) receives a SET MODE command. In the RUN state: • the scanner sends output information to the adapters • input information is updated • the scanner sends block transfer requests to the adapters • all outputs are allowed to energize
SYSFAIL	when the VME SYSFAIL monitor is enabled and SYSFAIL is asserted. In the SYSFAIL state, the scanner places a failure code in the confirmation status word, lights the Fault indicator, and enters a program loop waiting for a RESET command. The scanner ignores all other commands or interrupts from VME processors and ignores all packets from the I/O link. When the scanner receives a RESET command, it enters the POST state.

Powering Up the Scanner

You only power up the scanner after you've installed the scanner or when you are trying to recover from a faulted state. When you power up the scanner, the scanner does a self-test (POST state). If the self-test is successful, the scanner:

- moves into the SLEEP state.
- writes 0xF0 to the command byte
- writes the scanner's name, firmware version, and copyright notice to the VME ID area
- sets input and output image tables to 0
- deasserts SYSFAIL
- write 0x90 to the confirmation status byte

The VME master should monitor the confirmation status word to determine if the power-up is successful.

When the scanner is in the SLEEP state, you have to "wake-up" the scanner, which moves the scanner to the PROGRAM state, so it can accept scanner management commands.

How the Scanner Uses SYSFAIL During Power Up

While the scanner is doing its self tests during power up, the scanner asserts SYSFAIL on the VMEbus. When the scanner successfully completes its self tests, it no longer asserts SYSFAIL. During a power-up sequence, many other VME boards also assert SYSFAIL.

During normal operation in a VME system, the scanner watches the SYSFAIL line and shuts itself down whenever the SYSFAIL signal is asserted. However, during a system power-up sequence, some VME boards might assert SYSFAIL longer than the scanner. In this situation, the scanner doesn't start monitoring for a SYSFAIL until SYSFAIL is no longer asserted by another board during power up. You can also use the SETUP command to instruct the scanner to ignore SYSFAIL signals.

There are some VME boards that assert SYSFAIL while the operating system is booting up, which is after power has been applied and the other boards have finished their self tests. In this case, the scanner will light the FLT LED and go into the FAULTED state. And because the scanner is faulted, it asserts SYSFAIL. You can RESET the scanner with the RESET command.

Waking Up the Scanner

Only a VME master processor can wake up the scanner. The VME master wakes up the scanner by interrupting the scanner (writing any value to the scanner's ID area). The scanner responds to the interrupt by moving to the PROGRAM state.

As soon as the scanner receives the first command (usually a SETUP command), the operating status byte gets set to 0x01 and the scanner starts the VME master processor watchdog timer. The scanner must receive a valid command at least every 500 msec or the scanner will reset itself. You can adjust this time value by using the SETUP command.

The following example function shows one way to wake up the scanner.

```
/*
**
** WAKEUP() -- wake up the 6008-SV2
**
**     When the 6008-SV2 is first powered up, it enters the SLEEP state.  Writing a scratch
**     value to the VME ID Area will assert a local interrupt on the scanner, transitioning
**     it from the SLEEP state to the PROGRAM state.
**
**     parameters:         pointer to channel structure
**     return value:       none
**
*/

void
wakeup (channel far *chan)
{
       csa_data far *csa;

       /* map the channel & establish pointers */
       map_channel(chan);
       csa    =     (csa_data far *)    chan->ControlStatusArea;

       /* wait for SV2 to complete POST */
       while (csa->cnfstat != 0x0090);

       /* write the VME ID AREA of scanner */
       assert_cmd(chan);

       /* unmap the channel */
       unmap_channel(chan);

       return;
}
```

```
/*
**
** MAP_CHANNEL() -- map a 6008SV2 channel
**
**     This function accepts a channel structure as input, and maps the E-page VMEbus window
**     to the specified channel.  The function then initializes the channel structure
**     pointers to the various objects in the channel window (e.g., discrete I/O tables,
**     control/status area, and so forth.
**
**     parameters:         pointer to channel structure
**     return value:       none
*/

void
map_channel (channel far *chan)
{
       char far     *base;

       /* save current bus state */
       EpcSaveState(&chan->stash);

       /* get pointer to base of scanner memory */
       if (EpcSetAmMap(chan->AddressMode, chan->ScannerAddress,
              (void far * far *) &base) != EPC_SUCCESS) {
              fprintf(stderr, "cannot map target scanner memory\n");
              exit(-1);
       }

       /*
       ** establish pointers to scanner objects
       */

       if (chan->ScannerMode == MODE_COMPATIBLE) {

              /* output table at base of global memory */
              chan->OutputTable = &base[SVC_OTABLE_OFFSET];

              /* input table at offset 80H */
              chan->InputTable = (char far *) &base[SVC_ITABLE_OFFSET];

              /* control/status area at offset 100H */
              chan->ControlStatusArea = (char far *) &base[SVC_CSA_OFFSET];

              /* general data area at offset 120H */
              chan->GeneralDataArea = (char far *) &base[SVC_GDA_OFFSET];

              /* continuous block transfer write table */
              chan->ConBTWTable = (char far *) NULL;

              /* continuous block transfer read table */
              chan->ConBTRTable = (char far *) NULL;

              /* reset area at offset FBCH */
              chan->ResetArea = (char far *) &base[SVC_RESET_OFFSET];

              /* vme ID area at offset FC0H */
              chan->VmeIdArea = (char far *) &base[SVC_VIA_OFFSET];
```

```
        } else if (chan->ScannerMode == MODE_SUPERSET) {

                /* output table at base of global memory */
                chan->OutputTable = &base[SVS_OTABLE_OFFSET];

                /* input table at offset 100H */
                chan->InputTable = (char far *) &base[SVS_ITABLE_OFFSET];

                /* control/status area at offset 200H */
                chan->ControlStatusArea = (char far *) &base[SVS_CSA_OFFSET];

                /* general data area at offset 220H */
                chan->GeneralDataArea = (char far *) &base[SVS_GDA_OFFSET];

                /* continuous block transfer write table at offset 300H */
                chan->ConBTWTable = (char far *) &base[SVS_CONBTW_OFFSET];

                /* continuous block transfer read table at offset C00H */
                chan->ConBTRTable = (char far *) &base[SVS_CONBTR_OFFSET];

                /* reset area at offset 1FBCH */
                chan->ResetArea = (char far *) &base[SVS_RESET_OFFSET];

                /* vme ID area at offset 1FC0 */
                chan->VmeIdArea = (char far *) &base[SVS_VIA_OFFSET];

        }

        return;
}

/*
**
** UNMAP_CHANNEL() -- unmap a channel's VME shared memory
**
**     This function accepts a channel structure as input, and restores the EPC's VMEbus
**     window to the state it had when the channel was last mapped.
**
**     parameters:         pointer to channel structure
**     return value:       none
**
*/

void
unmap_channel (channel far *chan)
{
        EpcRestState(&chan->stash);
        return;
}
```

```
/*
**
** ASSERT_CMD() -- touch scanner channel location monitor
**
**     This function does a word-write to the base of a 6008-SV2 channel VME-ID/Scanner
**     interrupt area.  This triggers a local interrupt on the 6008-SV2, alerting it
**     that a command has been placed in the channel's control/status area for processing.
**
**     parameters:         pointer to channel structure
**     return value:       none
**
*/

void
assert_cmd (channel far *chan)
{
       unsigned short far  *vid;

       /* map channel & establish pointers */
       map_channel(chan);
       vid = (unsigned short far *) chan->VmeIdArea;

       /* touch the scanner's channel location monitor */
       *vid = (unsigned short) 0x0000;

       /* unmap the channel */
       unmap_channel(chan);

       return;
}
```

After Waking Up the Scanner

After you wake up the scanner, the scanner write 0x10 to the command status byte. Now the scanner is ready to accept commands. For example, send these commands:

- SETUP — to configure the scanner
- AUTOCONFIGURE or SCAN LIST — to create a scan list
- SET MODE — to put the scanner in run mode

If the VME master processor interrupts the scanner after waking the scanner up but without having a command ready to send to the scanner, the scanner ends up resetting itself. This is because there is no command for the scanner to execute, so the scanner's watchdog timer, which defaults at 500msec, runs out.

The programming examples in chapter 8 show how to use the scanner management commands. Each command description ends with a recommended procedure for using the command to avoid interrupting the scanner without having commands ready for the scanner to execute.

Chapter 8

Programming the Scanner

Using This Chapter

This chapter provides programming examples that illustrate the scanner management commands.

Important: The programming examples in this publication are examples only. They will need modification before they can run correctly in your application. These examples were developed for a Radisys EPC computer; they were written in Microsoft C version 7.00. To use these examples on other VME systems, search for and replace all “EPC” calls with the appropriate functions for your system.

For more details on the structure of the scanner commands, see chapter 5 or 6 for SV-compatible or SV-superset mode, respectively.

Using the Semaphore

Before the VME master processor issues a command request to the scanner, use a read-modify-write cycle to test and set the scanner’s semaphore. If the semaphore bit is set (non-zero), the VME master processor should not access the control/status and general data areas of the scanner (except for the operating status word). If, however, the semaphore is clear (zero), the VME master processor can then access the control/status and general data areas and should set the semaphore bit to signify to other master processors that these areas are in use.

Important: When the semaphore is set, other VME master processors can still overwrite data in the global RAM with their own data if necessary. The semaphore does not physically lock-out other VME master processors. It is only a flag to warn other VME master processors that another processor is accessing global RAM. When the scanner sees the semaphore set, it won’t write to the global data area. The scanner waits until a VME master processor resets the semaphore to zero. The scanner can never reset the semaphore itself. The scanner continues to update the I/O image tables and operating status words regardless of the status of the semaphore bit.

Knowing When a Command Is Complete

There are two ways to determine whether a command is complete. You can poll the confirmation status bye or you can wait for an interrupt.

If you:	You should:
poll for confirmation status	clear the confirmation status byte of the command prior to executing the command. When the command completes, the scanner returns a value to the confirmation status byte. The value 00H means the command completed successfully; any other value indicates an error.
wait for an interrupt	wait for the interrupt you specified in the command-complete interrupt byte of the SETUP command. You can specify interrupt values 01–07.

Programming Examples of Each Scanner Management Command

The following code shows examples of programming the scanner management commands. There is also sample code for various utilities and files needed for the scanner management commands as they are shown. See the following table for a list of the programming examples.

ATTENTION: These examples are meant to help you create your own programs. They need modification to work correctly on your processor system.

SETUP command

```
/*
**
** SETUP() -- set up a 6008-SV2 channel
**
**     This function executes a SETUP command, using the parameters provided by the caller
**     in the setup_data structure.
**
**     parameters:         pointer to channel structure
**                         pointer to setup_data structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
**     NOTE: this routine overwrites the channel structure CmdCmplIrq & CmdCmplVec fields
**     with the interrupt level and vector contained in the setup_param structure.
**
*/

int
setup (channel far *chan, setup_data far *setup_param)
{
       csa_data far              *csa;
       unsigned short far        *vid;
       unsigned short            setup_len;
       unsigned short            res;

       /* update the command complete interrupt settings */
       chan->CmdCmplIrq    =     (unsigned short)    setup_param->cmdcmpl_irq_level;
       chan->CmdCmplVec    =     (unsigned short)    setup_param->cmdcmpl_irq_vector;

       /* map the channel & establish pointers */
       map_channel(chan);
       csa    = (csa_data far *) chan->ControlStatusArea;
       vid    = (unsigned short far *) chan->VmeIdArea;

       /* determine length of setup data */
       setup_len = (unsigned short)
              ((chan->ScannerMode == MODE_COMPATIBLE) ? SVC_SETUP_SIZE :
              SVS_SETUP_SIZE);

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              fprintf(stdout, "setup: lock aquire FAILED\n");
              unmap_channel(chan);
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat = 0x000F;           /* confirmation status word */
       csa->cmd     = CMD_SETUP;        /* command word */
       csa->datalen = setup_len;        /* length of data word */

       /* copy the setup parameters */
       blockcpy(chan->GeneralDataArea, setup_param, (size_t) setup_len);
```

```
        /* send the command interrupt */
        assert_cmd(chan);

        /* wait for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
                unlock_gda(chan);
                unmap_channel(chan);
                fprintf(stdout, ”setup: wait failure, locked cleared\n”);
                return –2;
        }

        /* clear the semaphore bit & unmap the channel */
        res = csa–>cnfstat;
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

AUTOCONFIGURE command

```
/*
**
** AUTOCFG() -- autoconfigure a 6008-SV2 channel
**
**     This function executes a AUTOCONFIGURE command.  The results of the autoconfigure are
**     placed in the link_config structure provided by the caller.
**
**     parameters:         pointer to channel structure
**                         pointer to link_config structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
**     Note that if the confirmation status word is not 0000H, the contents of the
**     link_config structure passed by the caller are NOT defined.
**
*/

int
autocfg (channel far *chan, link_config far *linkcfg)
{
       csa_data far               *csa;
       unsigned short far         *gda;
       unsigned char far          *scanlist;
       unsigned short             ioaswblk_size;
       unsigned short             res;
       unsigned short             i;

       /* map channel & establish pointers */
       map_channel(chan);
       csa    = (csa_data far *)         chan->ControlStatusArea;
       gda    = (unsigned short far *)   chan->GeneralDataArea;

       /* determine I/O Adapter Status Word block size */
       ioaswblk_size = (unsigned short)
              ((chan->ScannerMode == MODE_COMPATIBLE) ? SVC_IOASWBLK_SIZE :
              SVS_IOASWBLK_SIZE);

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "autocfg: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat = 0x000F;
       csa->cmd     = CMD_AUTOCFG;
       csa->datalen = 0;

       /* send the command interrupt */
       assert_cmd(chan);
```

```
        /* poll for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
                /* clear the semaphore */
                unlock_gda(chan);
                unmap_channel(chan);
                fprintf(stdout, "autocfg: wait failure, lock cleared\n");
                return -2;
        }

        /* check autoconfigure result */
        res = csa->cnfstat;
        if (res != 0) {
                /* autoconfigure failed */
                unlock_gda(chan);
                unmap_channel(chan);
                return (int) res;
        }

        /* place scan list size in autocfg structure */
        linkcfg->scanlist_len = csa->datalen;

        /* copy I/O adapter status word block to autocfg structure */
        for (i = 0; i < ioaswblk_size; i++) {
                linkcfg->ioaswblk[i] = gda[i];
        }

        /* copy the scan list to the autocfg structure */
        scanlist = (unsigned char far *) (&gda[ioaswblk_size]);
        blockcpy(linkcfg->scanlist, scanlist, (size_t) csa->datalen);

        /* clear the semaphore & unmap channel */
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

SCANLIST command

```
/*
**
** SCANLIST() -- configure a 6008-SV2 channel scan list
**
**     This function executes a SCANLIST command, using the scanlist provided by the
**     caller in the scanlist_data parameter.
**
**     parameters:         pointer to channel structure
**                         pointer to scanlist_data structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
*/

int
scanlist (channel far *chan, scanlist_data far *scancfg)
{
       csa_data far              *csa;
       unsigned short            res;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "scanlist: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat  = 0x000F;                 /* confirm status word */
       csa->cmd      = CMD_SCANLIST;           /* command word */
       csa->datalen  = scancfg->scanlist_len;  /* length of data word */

       /* copy the scan list to the general data area */
       blockcpy(chan->GeneralDataArea, scancfg->scanlist,
              (size_t) scancfg->scanlist_len);

       /* send the command interrupt */
       assert_cmd(chan);

       /* poll for command result */
       if (wait_for_cmdcmpl(chan) == FALSE) {
              /* clear the semaphore */
              unlock_gda(chan);
              unmap_channel(chan);
              fprintf(stdout, "scanlist: wait failure, lock cleared\n");
              return -2;
       }

       /* clear the semaphore & unmap the channel */
       res = csa->cnfstat;
       unlock_gda(chan);
       unmap_channel(chan);
       return (int) res;
}
```

FAULT DEPENDENT GROUP command

```
/*
**
** FAULTGROUP() -- configure 6008-SV2 channel fault-dependent groups
**
**     This function executes a FAULT GROUP command, using the fault group block provided
**     by the caller in the fdg_data parameter.
**
**     parameters:         pointer to channel structure
**                         pointer to fdg_data structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
*/

int
faultgroup (channel far *chan, fdg_data far *fdgcfg)
{
       csa_data far              *csa;
       unsigned short            fdgblk_size;
       unsigned short            res;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;

       /* determine size of fault-dependent group block */
       fdgblk_size = (unsigned short)
              ((chan->ScannerMode == MODE_COMPATIBLE) ? SVC_FAULTGRP_SIZE :
              SVS_FAULTGRP_SIZE);

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "fdg: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat  = 0x000F;           /* confirmation status word */
       csa->cmd      = CMD_FDGROUP;      /* command word */
       csa->datalen  = fdgblk_size;      /* length of data word */

       /* copy the fault-dependent group data to the general data area */
       blockcpy(chan->GeneralDataArea, fdgcfg, (size_t) fdgblk_size);

       /* send the command interrupt */
       assert_cmd(chan);
```

```
        /* poll for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
                /* clear the semaphore */
                unlock_gda(chan);
                unmap_channel(chan);
                fprintf(stdout, ”fdg: wait failure, lock cleared\n”);
                return –2;
        }

        /* clear the semaphore & unmap the channel */
        res = csa–>cnfstat;
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

SET MODE command

```
/*
**
** SETMODE() -- configure 6008-SV2 channel operating mode
**
**     This function executes a SET MODE command, using the mode passed by the caller in
**     the mode parameter.
**
**     parameters:         pointer to channel structure
**                         mode parameter (1 = PROGRAM, 2 = TEST, 4 = RUN)
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
*/

int
setmode (channel far *chan, int mode)
{
       csa_data far              *csa;
       unsigned short            res;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "set mode: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat  = 0x000F;           /* confirmation status word */
       csa->cmd      = CMD_SETMODE;      /* command word */
       csa->datalen  = 0x0001;           /* length of data word */

       /* put mode parameter in general data area */
       *chan->GeneralDataArea = (unsigned char) mode;

       /* send the command interrupt */
       assert_cmd(chan);

       /* poll for command result */
       if (wait_for_cmdcmpl(chan) == FALSE) {
              /* clear the semaphore */
              unlock_gda(chan);
              unmap_channel(chan);
              fprintf(stdout, "setmode: wait failure, lock cleared\n");
              return -2;
       }

       /* clear the semaphore & unmap the channel */
       res = csa->cnfstat;
       unlock_gda(chan);
       unmap_channel(chan);
       return (int) res;
}
```

LINK STATUS command

```
/*
**
** LINKSTATUS() -- get 6008-SV2 channel link status
**
**     This function executes a LINK STATUS command.  The results of the LINK STATUS are
**     placed in the link_config structure provided by the caller.
**
**     parameters:         pointer to channel structure
**                         pointer to link_config structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
**     Note that if the confirmation status word is not 0000H, the contents of the
**     link_config structure passed by the caller are NOT defined.
**
*/

int
linkstatus (channel far *chan, link_config far *linkcfg)
{
       csa_data far              *csa;
       unsigned short far        *gda;
       unsigned char far         *scanlist;
       unsigned short            ioaswblk_size;
       unsigned short            res;
       unsigned short            i;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;
       gda = (unsigned short far *) chan->GeneralDataArea;

       /* determine I/O Adapter Status Word block size */
       ioaswblk_size = (unsigned short)
              ((chan->ScannerMode == MODE_COMPATIBLE) ? SVC_IOASWBLK_SIZE :
              SVS_IOASWBLK_SIZE);

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "linkstat: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat = 0x000F;           /* confirmation status word */
       csa->cmd     = CMD_LINKSTAT;     /* command word */
       csa->datalen = 0;          /* length of data word */

       /* send the command interrupt */
       assert_cmd(chan);
```

```
        /* poll for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
                /* clear the semaphore */
                unlock_gda(chan);
                unmap_channel(chan);
                fprintf(stdout, ”fdg: wait failure, lock cleared\n”);
                return –2;
        }

        /* check autoconfigure result */
        res = csa–>cnfstat;
        if (res != 0) {
                /* link status failed */
                unlock_gda(chan);
                unmap_channel(chan);
                return (int) res;
        }

        /* place scan list size in autocfg structure */
        linkcfg–>scanlist_len = csa–>datalen;

        /* copy I/O adapter status word block to autocfg structure */
        for (i = 0; i < ioaswblk_size; i++) {
                linkcfg–>ioaswblk[i] = gda[i];
        }

        /* copy the scan list to the autocfg structure */
        scanlist = (unsigned char far *) (&gda[ioaswblk_size]);
        blockcpy(linkcfg–>scanlist, scanlist, (size_t) csa–>datalen);

        /* clear the semaphore & unmap channel */
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

BT WRITE / BT READ

```
/*
**
** QUEUE_ONESHOT_BT() -- queue a one-shot block transfer read or write
**
**     This function is driven by the data contained in the bt_data structured provided by
**     the caller.  The command word (OSBTW/OSBTR), BT address word, tag word, and
**     BT transfer length word are placed in the Control/Status Area.  If the command is a
**     OSBTW, then data is copied from the bt_data struct's data buffer to the channel's
**     General Data Area.  The command is then asserted and its results returned to
**     the caller.
**
**     parameters:         pointer to channel structure
**                         pointer to the bt_data structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
*/

int
queue_oneshot_bt (channel far *chan, bt_data far *block)
{
       csa_data far              *csa;
       unsigned short far        *gda;
       unsigned short            res;
       unsigned short            i;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;
       gda = (unsigned short far *) chan->GeneralDataArea;

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "osbtwrite: lock aquire FAILED\n");
              return -1;
       }


       /* set up the control/status area */
       csa->cnfstat = 0x000F;           /* confirmation status word */
       csa->cmd     = block->cmd;       /* command word */
       csa->addr    = block->addr;      /* address word */
       csa->tag     = block->tag;       /* tag word */
       csa->datalen = block->datalen;   /* length of data word */

       /* if a BT write */
       if (block->cmd == CMD_OSBTWRITE) {
              /* copy the block data */
              for (i = 0; i < block->datalen; i++) {
                     gda[i] = block->data[i];
              }
       }
```

```
        /* send the command interrupt */
        assert_cmd(chan);

        /* poll for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
                /* clear the semaphore */
                unlock_gda(chan);
                unmap_channel(chan);
                fprintf(stdout, ”osbtwrite: wait failure, lock cleared\n”);
                return –2;
        }

        /* clear the semaphore */
        res = csa–>cnfstat;
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

CONTINUOUS BT WRITE / CONTINUOUS BT READ command

```
/*
**
** CONTINUOUS_BT_UPDATE() -- update a continuous block transfer write or read operation.
**
**     The bt_data parameter passed by the caller allows this routine to identify the
**     specific continuous block transfer operation to update.  The command word field
**     identifies the block transfer table and the tag field specifies the table entry.
**
**     The routine copies the table entry's command word, address word, tag word, transfer
**     length word, and interrupt word parameters to the bt_data parameter.
**
**     If the specified operation is a write block transfer, the routine copies data from the
**     bt_data parameter's data buffer to the table entry data buffer.  If the specified
**     operation is a read block transfer, the routine copies data from the table entry's
**     data buffer to the bt_data parameter's data buffer.
**
**     parameters:         pointer to channel structure
**                         pointer to the bt_data structure
**     return values:      0      = success
**                         -1     = could not lock table entry's data buffer
**
*/

int
continuous_bt_update (channel far *chan, bt_data far *block)
{
       csa_data far              *csa;
       unsigned short far        *gda;
       btentry far               *bt;
       btentry far               *entry;
       unsigned short            i;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;
       gda = (unsigned short far *) chan->GeneralDataArea;

       /* make pointer to continuous BT table */
       if (block->cmd == CMD_CONBTREAD) {
              /* read operation */
              bt = (btentry far *) chan->ConBTRTable;
       } else {
              /* write operation */
              bt = (btentry far *) chan->ConBTWTable;
       }

       /* make pointer to continuous BT table entry */
       entry = (btentry far *) &bt[block->tag];

       /* update transfer parameters */
       block->tag          = entry->tag;
       block->addr         = entry->addr;
       block->datalen      = entry->datalen;
       block->irq          = entry->irq;
```

```
        /* try to lock the table entry buffer */
        if (lock_conbt_entry(entry) != 0) {
                /* could not lock buffer */
                unmap_channel(chan);
                return –1;
        }

        /* copy the data in or out */
        if (block–>cmd == CMD_CONBTREAD) {
                /* continuous BT read */
                for (i = 0; i < block–>datalen; i++) {
                        block–>data[i] = entry–>data[i];
                }
        } else {
                /* continuous BT write */
                for (i = 0; i < block–>datalen; i++) {
                        entry–>data[i] = block–>data[i];
                }
        }

        /* unlock the table entry buffer */
        unlock_conbt_entry(entry);
        unmap_channel(chan);
        return 0;
}
```

RESET command

```
/*
**
** RESET() -- soft-reset a 6008-SV2
**
**     This routine forces a 6008-SV2 into the RESET state.  Note that this affects BOTH
**     scanner channels.  Once the 6008-SV2 has been soft RESET, each channel must be
**     awakened by the wakeup() routine and reconfigured.
**
**     parameters:         pointer to channel structure
**     return values:      none
**
*/

void
reset (channel far *chan)
{
       unsigned short far  *wordptr;

       /* map channel & establish pointers */
       map_channel(chan);
       wordptr = (unsigned short far *) chan->VmeIdArea;

       /* write the VME ID AREA of scanner */
       wordptr[0] = 0x0080;
       wordptr[1] = 0xA0A0;

       /* unmap the channel */
       unmap_channel(chan);
       return;
}
```

sample code for VMEbus operations

```
/*
**
** BUS.C -- Sample Code VME bus operations
**
*/

#include "6008sv2.h"
#include "busmgr.h"

/*
**
** INIT_BUS_OPS() -- initialize VMEbus access
**
**     This function checks for the EPC Bus Manager driver, then enables interrupts
**     on IRQ4, the VMEbus interrupt used by the 6008SV2 as a default.
**
*/

void
init_bus_ops (void)
{

       int    i;

       /* check for bus manager */
       if (EpcCkBm() != EPC_SUCCESS) {
              fprintf(stderr, "Bus Manager not loaded\n");
              exit(-1);
       }

       for (i = 1; i < 8; i++)
              EpcEnIntr((unsigned short) i);

       return;
}
```

```
/*
**
** MAP_CHANNEL() -- map a 6008SV2 channel
**
**     This function accepts a channel structure as input, and maps the E-page VMEbus window
**     to the specified channel.  The function then initializes the channel structure
**     pointers to the various objects in the channel window (e.g., discrete I/O tables,
**     control/status area, and so forth.
**
**     parameters:         pointer to channel structure
**     return value:       none
*/

void
map_channel (channel far *chan)
{
       char far     *base;

       /* save current bus state */
       EpcSaveState(&chan->stash);

       /* get pointer to base of scanner memory */
       if (EpcSetAmMap(chan->AddressMode, chan->ScannerAddress,
              (void far * far *) &base) != EPC_SUCCESS) {
              fprintf(stderr, "cannot map target scanner memory\n");
              exit(-1);
       }

       /*
       ** establish pointers to scanner objects
       */

       if (chan->ScannerMode == MODE_COMPATIBLE) {

              /* output table at base of global memory */
              chan->OutputTable = &base[SVC_OTABLE_OFFSET];

              /* input table at offset 80H */
              chan->InputTable = (char far *) &base[SVC_ITABLE_OFFSET];

              /* control/status area at offset 100H */
              chan->ControlStatusArea = (char far *) &base[SVC_CSA_OFFSET];

              /* general data area at offset 120H */
              chan->GeneralDataArea = (char far *) &base[SVC_GDA_OFFSET];

              /* continuous block transfer write table */
              chan->ConBTWTable = (char far *) NULL;

              /* continuous block transfer read table */
              chan->ConBTRTable = (char far *) NULL;


              /* reset area at offset FBCH */
              chan->ResetArea = (char far *) &base[SVC_RESET_OFFSET];

              /* vme ID area at offset FC0H */
              chan->VmeIdArea = (char far *) &base[SVC_VIA_OFFSET];
```

```
        } else if (chan->ScannerMode == MODE_SUPERSET) {

                /* output table at base of global memory */
                chan->OutputTable = &base[SVS_OTABLE_OFFSET];

                /* input table at offset 100H */
                chan->InputTable = (char far *) &base[SVS_ITABLE_OFFSET];

                /* control/status area at offset 200H */
                chan->ControlStatusArea = (char far *) &base[SVS_CSA_OFFSET];

                /* general data area at offset 220H */
                chan->GeneralDataArea = (char far *) &base[SVS_GDA_OFFSET];

                /* continuous block transfer write table at offset 300H */
                chan->ConBTWTable = (char far *) &base[SVS_CONBTW_OFFSET];

                /* continuous block transfer read table at offset C00H */
                chan->ConBTRTable = (char far *) &base[SVS_CONBTR_OFFSET];

                /* reset area at offset 1FBCH */
                chan->ResetArea = (char far *) &base[SVS_RESET_OFFSET];

                /* vme ID area at offset 1FC0 */
                chan->VmeIdArea = (char far *) &base[SVS_VIA_OFFSET];

        }

        return;
}

/*
**
** UNMAP_CHANNEL() -- unmap a channel's VME shared memory
**
**     This function accepts a channel structure as input, and restores the EPC's VMEbus
**     window to the state it had when the channel was last mapped.
**
**     parameters:         pointer to channel structure
**     return value:       none
**
*/

void
unmap_channel (channel far *chan)
{
        EpcRestState(&chan->stash);
        return;
}
```

```
/*
**
** WAIT_FOR_IRQ() -- wait for scanner interrupt
**
**     This function waits for an interrupt from the 6008-SV2.
**
**     parameters:         timeout -- timeout interval in milliseconds
**     return value:       FALSE -- interrupt not received or error
**                         TRUE  -- interrupt received
**
*/

int
wait_for_cmdcmpl (channel far *chan)
{
       short               res;
       unsigned long       status;
       unsigned short      masks[8] = { 0x0000, 0x0002, 0x0004, 0x0008,
              0x0010, 0x0020, 0x0040, 0x0080 };

       /* enable specified VMEbus interrupt */
       EpcEnIntr(chan->CmdCmplIrq);

       /* wake up on the specified VMEbus interrupt */
       res = EpcWaitIntr(masks[chan->CmdCmplIrq], &status, chan->TimeOut);

       /* check result for timeout or error */
       if ((res == 0) || (res == ERR_FAIL)) {
              printf("wait: no interrupt received (timeout)!\n");
              return FALSE;
       }

       /* check result for correct level */
       if ((res & masks[chan->CmdCmplIrq]) != masks[chan->CmdCmplIrq]) {
              printf("wait: incorrect interrupt level received!\n");
              return FALSE;
       }

       /* check status for correct vector */
       if ((status & 0x000000FF) != chan->CmdCmplVec) {
              printf("wait: incorrect interrupt vector received\n");
              return FALSE;
       }

       return TRUE;
}
```

sample code for utility routines

```
/*
**
** UTILS.C -- sample code utility routines
**
*/

#include "6008sv2.h"
#include "bus.h"
#include "utils.h"

/*
**
** ASSERT_CMD() -- touch scanner channel location monitor
**
**     This function does a word-write to the base of a 6008-SV2 channel VME-ID/Scanner
**     interrupt area.  This triggers a local interrupt on the 6008-SV2, alerting it
**     that a command has been placed in the channel's control/status area for processing.
**
**     parameters:         pointer to channel structure
**     return value:       none
**
*/

void
assert_cmd (channel far *chan)
{
       unsigned short far  *vid;

       /* map channel & establish pointers */
       map_channel(chan);
       vid = (unsigned short far *) chan->VmeIdArea;

       /* touch the scanner's channel location monitor */
       *vid = (unsigned short) 0x0000;

       /* unmap the channel */
       unmap_channel(chan);

       return;
}
```

```
/*
**
** READ_CSA() -- read control status area
**
**     This function reads a 6008-SV2 channel's control/status area and returns the data
**     to the caller.
**
**     parameters:         pointer to a csa_data struct
**     return value:       none
**
*/

void
read_csa (channel far *chan, csa_data far *param)
{
       csa_data far  *csaptr;

       /* map channel & establish pointers */
       map_channel(chan);
       csaptr = (csa_data far *) chan->ControlStatusArea;

       /* copy VME csarea to local structure */
       param->opstat       = csaptr->opstat;
       param->cnfstat      = csaptr->cnfstat;
       param->cmd          = csaptr->cmd;
       param->addr         = csaptr->addr;
       param->tag          = csaptr->tag;
       param->irq          = csaptr->irq;
       param->semaphore    = csaptr->semaphore;
       param->datalen      = csaptr->datalen;

       /* unmap the channel */
       unmap_channel(chan);

       return;
}
```

```
/*
**
** LOCK_GDA() -- lock a channel's general data area
**
**     This function locks a channel's general data area semaphore.
**
**     parameters:         pointer to channel struct
**     return value:       none
**
*/

int
lock_gda(channel far *chan)
{
       csa_data far *csaptr;
       int          res;

       /* map channel & establish pointers */
       map_channel(chan);
       csaptr = (csa_data far *) chan->ControlStatusArea;

       /* attempt to lock the semaphore */
       res = sv2_lock((unsigned short far *) &csaptr->semaphore);

       /* unmap channel & return */
       unmap_channel(chan);
       return res;
}

/*
**
** UNLOCK_GDA() -- unlock a channel's General Data Area
**
**     This function unlocks a channel's general data area semaphore.
**
**     parameters:         pointer to channel struct
**     return value:       none
**
*/

void
unlock_gda (channel far *chan)
{
       csa_data far *csa;

       /* map channel & establish pointers */
       map_channel(chan);
       csa = (csa_data far *) chan->ControlStatusArea;

       /* clear the semaphore bit */
       csa->semaphore &= (unsigned short) (~SEMAPHORE);

       /* unmap the channel */
       unmap_channel(chan);
       return;
}
```

```
/*
**
** LOCK_CONBT_ENTRY() -- lock the data buffer of a Continuous Block Transfer Table Entry
**
**     The function busy-waits on the entry's turn & sv2_flag fields, using the standard
**     Peterson's Solution method.
**
**     parameters:         pointer to the Continuous BT Table entry
**     return value:       always success (0)
**
*/

int
lock_conbt_entry (btentry far *entry)
{
       entry->vmp_flag     = 1;
       entry->turn_flag    = 1;
       while ((entry->sv2_flag) && (entry->turn_flag == 1));
       return 0;
}

/*
**
** UNLOCK_CONBT_ENTRY() -- unlock the data buffer of a Continuous Block Transfer Table Entry
**
**     parameters:         pointer to the Continuous BT Table entry
**     return value:       none
**
*/

void
unlock_conbt_entry (btentry far *entry)
{
       entry->vmp_flag = 0;
       return;
}
```

```
/*
**
** PRINT_LINK_CONFIG() -- print contents of link_config data structure
**
**     This routine prints the contents of a link_config structure, which is used by the
**     autocfg() and linkstatus() routines.
**
**     parameters:         pointer to link_config structure
**     return value:       none
**
*/

void
print_link_config (channel far *chan, link_config far *ptr)
{
       unsigned short      ioaswblk_size;
       unsigned short      i;

       /* determine I/O adapter status word block size */
       ioaswblk_size = (unsigned short)
              ((chan->ScannerMode == MODE_COMPATIBLE) ? SVC_IOASWBLK_SIZE :
              SVS_IOASWBLK_SIZE);

       printf("\n\track\tGRP0\tGRP1\tGRP2\tGRP3\n");
       for (i = 0; i < ioaswblk_size; i += 4) {
              printf("\t%d\t%XH\t%XH\t%XH\t%XH\n", (i/4),
                     ptr->ioaswblk[i+0], ptr->ioaswblk[i+1],
                     ptr->ioaswblk[i+2], ptr->ioaswblk[i+3]);
       }

       printf("\n\tscanlist length = %d\n", ptr->scanlist_len);
       for (i = 0; i < ptr->scanlist_len; i++) {
              printf("\tentry[%d] = %XH\n", i, ptr->scanlist[i]);
       }


       return;
}
```

```
/*
**
** PRINT_BT_DATA() -- print contents of bt_data structure
**
**     This routine prints the contents of a bt_data structure, which is used by the
**     queue_oneshot_bt(), oneshot_bt_complete(), configure_continuous_bt(), and
**     continuous_bt_update() routines.
**
**     parameters:         pointer to bt_data structure
**     return value:       none
**
*/

void
print_bt_data (bt_data far *ptr)
{
       unsigned short      i;

       /* print command-type */
       if (ptr->cmd == CMD_OSBTWRITE) {
              fprintf(stdout, "\tcmd     = ONE-SHOT BT WRITE\n");
       } else if (ptr->cmd == CMD_OSBTREAD) {
              fprintf(stdout, "\tcmd     = ONE-SHOT BT READ\n");
       } else if (ptr->cmd == CMD_CONBTWRITE) {
              fprintf(stdout, "\tcmd     = CONTINUOUS BT WRITE\n");
       } else {
              fprintf(stdout, "\tcmd     = CONTINUOUS BT READ\n");
       }

       /* print tag word */
       fprintf(stdout, "\ttag     = %XH\n", ptr->tag);

       /* print address */
       fprintf(stdout, "\taddress = %XH { rack %d, group %d, slot %d }\n",
              ptr->addr, ((ptr->addr & 0x00F0) >> 4),
              ((ptr->addr & 0x000E) >> 1), (ptr->addr & 0x0001));

       /* print data length */
       fprintf(stdout, "\tdatalen = %d words\n", ptr->datalen);

       /* if continuous operation, print irq */
       if ((ptr->cmd == CMD_CONBTWRITE) || (ptr->cmd == CMD_CONBTREAD)) {
       fprintf(stdout, "\tinterrupting on IRQ %d with vector %XH\n",
              (ptr->irq & 0x0007), ((ptr->irq & 0xFF00) >> 8));
       }

       /* print data buffer */
       for (i = 0; i < ptr->datalen; i++) {
              fprintf(stdout, "\tbuffer[%d] = %XH\n", i, ptr->data[i]);
       }

       return;

}
```

```
/*
**
** BLOCKCPY() -- copy far source buffer to far destination buffer in byte mode.
**
**     This routine replaces _fmemcpy(), which unfortunately uses 16-bit or 32-bit transfers.
**     _fmemcpy() is great for general purposes, but unfortunately byte strings get shredded
**     when copied to the E-page bus window.
**
**     parameters:         far pointer to destination buffer
**                         far pointer to source buffer
**                         number of bytes to copy
**     return value:       none
**
*/

void
blockcpy (void far *dst, void far *src, size_t len)
{
       char far     *dstptr = (char far *) dst;
       char far     *srcptr = (char far *) src;
       size_t       i;

       for (i = 0; i < len; i++) {
              dstptr[i] = srcptr[i];
       }

       return;
}
```

6008sv2.h

```
/*
**
** 6008SV2.H -- defines for 6008-SV2 Sample Code
**
*/

/* booleans */
#define FALSE       0
#define TRUE        1

/* 6008-SV2 Operating Modes */
#define MODE_COMPATIBLE    0      /* SV-compatible mode */
#define MODE_SUPERSET      1      /* SV-superset mode */
#define MODE_ADAPTER       2      /* SV-adapter mode */

/* VMEBUS Address Modes */
#define ADRSPC_A16         0      /* A16, non-privileged */
#define ADRSPC_A16S        1      /* A16, supervisory */
#define ADRSPC_A24D        2      /* A24, non-privileged data */
#define ADRSPC_A24SD       3      /* A24, supervisory data */

/* 6008-SV2 Scanner Management Commands */
#define CMD_AUTOCFG        ((unsigned short) 0x0010)  /* autoconfigure */
#define CMD_SCANLIST       ((unsigned short) 0x0011)  /* scan list */
#define CMD_FDGROUP        ((unsigned short) 0x0012)  /* fault group */
#define CMD_SETUP          ((unsigned short) 0x0013)  /* setup */
#define CMD_SETMODE        ((unsigned short) 0x0020)  /* set mode */
#define CMD_LINKSTAT       ((unsigned short) 0x0021)  /* link status */
#define CMD_OSBTREAD       ((unsigned short) 0x0001)  /* one-shot BT read */
#define CMD_OSBTWRITE      ((unsigned short) 0x0002)  /* one-shot BT write */
#define CMD_CONBTREAD      ((unsigned short) 0x0006)  /* continuous BT read */
#define CMD_CONBTWRITE     ((unsigned short) 0x0007)  /* continuous BT write */

/* 6008-SV2 Scanner Operating Modes */
#define MODE_PROGRAM       ((unsigned char) 0x01)     /* program mode */
#define MODE_TEST          ((unsigned char) 0x02)     /* test mode */
#define MODE_RUN           ((unsigned char) 0x04)     /* run mode */

/* 6008-SV2 Scanner Channel Offsets (Compatible-Mode) */
#define SVC_OTABLE_OFFSET         0x0000       /* output table */
#define SVC_ITABLE_OFFSET         0x0080       /* input table */
#define SVC_CSA_OFFSET            0x0100       /* control/status area */
#define SVC_GDA_OFFSET            0x0120       /* general data area */
#define SVC_RESET_OFFSET          0x0FBC       /* soft reset register */
#define SVC_VIA_OFFSET            0x0FC0       /* VME ID area */

/* 6008-SV2 Scanner Channel Offsets (Superset-Mode) */
#define SVS_OTABLE_OFFSET         0x0000       /* output table */
#define SVS_ITABLE_OFFSET         0x0100       /* input table */
#define SVS_CSA_OFFSET            0x0200       /* control/status area */
#define SVS_GDA_OFFSET            0x0220       /* general data area */
#define SVS_CONBTW_OFFSET         0x0300       /* cont. BT write table */
#define SVS_CONBTR_OFFSET         0x0C00       /* cont. BT read table */
#define SVS_RESET_OFFSET          0x1FBC       /* soft reset register */
#define SVS_VIA_OFFSET            0x1FC0       /* VME ID area */
```

```
/* 6008-SV2 Scanner Constants */
#define SVC_SETUP_SIZE     7
#define SVC_IOASWBLK_SIZE  32
#define SVC_FAULTGRP_SIZE  32
#define SVS_SETUP_SIZE     12
#define SVS_IOASWBLK_SIZE  64
#define SVS_FAULTGRP_SIZE  64

/* 6008-SV2 Scanner Channel semaphore */
#define SEMAPHORE   0x0080       /* bit 7 of semaphore word */

/*
** Channel Context structure
*/

typedef struct {

      unsigned long       ScannerAddress;           /* VME address */
      unsigned short      AddressMode;        /* VME address mode */
      int                 ScannerMode;        /* operating mode (SVC,SVS) */
      unsigned short      CmdCmplIrq;         /* cmd-complete IRQ level */
      unsigned short      CmdCmplVec;         /* cmd-complete IRQ vector */
      unsigned long       TimeOut;            /* interrupt timeout, in ms */
      char far            *ScannerBase;       /* local ptr to channel */
      char far            *OutputTable;       /* channel output table */
      char far            *InputTable;        /* channel input table */
      char far            *ControlStatusArea; /* channel CSA */
      char far            *GeneralDataArea;   /* channel general data area */
      char far            *ConBTWTable;       /* continuous BT write table */
      char far            *ConBTRTable;       /* continuous BT read table */
      char far            *ResetArea;         /* reset area */
      char far            *VmeIdArea;         /* VME ID/interrupt area */
      unsigned long       stash;              /* bus manager context */

} channel;

/*
** Control/Status Area structure
*/

typedef struct {

      unsigned short      opstat;      /* Operating Status Word */
      unsigned short      cnfstat;     /* Confirmation Status Word */
      unsigned short      cmd;         /* Command Word */
      unsigned short      addr;        /* Block Transfer Address Word */
      unsigned short      tag;         /* Block Transfer Tag Word */
      unsigned short      irq;         /* Block Transfer Interrupt Word */
      unsigned short      iitm;        /* IITM Index Word */
      unsigned short      res[7];      /* reserved fields */
      unsigned short      semaphore;   /* Semaphore Word */
      unsigned short      datalen;     /* Length of Data Word */

} csa_data;
```

```
/*
** Setup Data Structure
*/

typedef struct {

       unsigned char       rio_baudrate;       /* RIO link baudrate */
       unsigned char       wdt_interval;       /* watchdog timer interval */
       unsigned char       wdt_enable;         /* watchdog timer enable */
       unsigned char       cmdcmpl_irq_level;  /* cmd complete IRQ level */
       unsigned char       cmdcmpl_irq_vector; /* cmd complete IRQ vector */
       unsigned char       cmdcmpl_irq_enable; /* cmd complete IRQ enable */
       unsigned char       vsf_monitor_enable; /* sysfail monitor enable */
       unsigned char       iitm_irq_level;     /* IITM IRQ level */
       unsigned char       iitm_irq_vector;    /* IITM IRQ vector */
       unsigned char       iitm_irq_enable;    /* IITM IRQ enable */
       unsigned char       iitm_lo_bound;      /* IITM index low bound */
       unsigned char       iitm_hi_bound;      /* IITM index high bound */

} setup_data;

/*
** Autoconfigure/Link Status Data Structure
*/

/* maximum size of Scan List */
#define SCANLIST_SIZE            64

/* maximum size of I/O Adapter Status Word Block */
#define IOASWBLK_SIZE            64

typedef struct {

       unsigned short      scanlist_len;              /* Length of Scan List */
       unsigned short      ioaswblk[IOASWBLK_SIZE];   /* IO Adapter Status Block */
       unsigned char scanlist[SCANLIST_SIZE];         /* Scan List Block */

} link_config;

/*
** Scan List Data Structure
*/

typedef struct {

       unsigned short      scanlist_len;          /* Length of Scan List */
       unsigned char scanlist[SCANLIST_SIZE];   /* Scan List Block */

} scanlist_data;

/*
** Fault-Dependent Group Data Structure
*/

/* maximum size of fault-dependent group block */
#define FDG_BLOCK_SIZE           64

typedef struct {

       unsigned char fdg_block[FDG_BLOCK_SIZE];

} fdg_data;
```

```
/*
** Block Transfer Data Structure
*/

/* maximum block transfer length, in words */
#define MAX_BT_DATALEN           64

typedef struct {

      unsigned short      cmd;         /* Block Transfer Command Word */
      unsigned short      cnfstat;     /* Block Transfer Result */
      unsigned short      tag;         /* Block Transfer Tag Word */
      unsigned short      addr;        /* Block Transfer Address Word */
      unsigned short      datalen;     /* Block Transfer Length Word */
      unsigned short      irq;         /* Block Transfer Interrupt Word */

      /* Block Transfer Data Buffer */
      unsigned short      data[MAX_BT_DATALEN];

} bt_data;

/*
** Block Transfer Table Entry
*/

typedef struct {

      unsigned short      tag;         /* Block Transfer Tag Word */
      unsigned short      addr;        /* Block Transfer Address Word */
      unsigned short      datalen;     /* Block Transfer Length Word */
      unsigned short      irq;         /* Block Transfer Interrupt Word */
      unsigned short      sv2_flag;    /* semaphore: sv6008–write */
      unsigned short      vmp_flag;    /* semaphore: VME host write */
      unsigned short      turn_flag;   /* semaphore: turn flag */
      unsigned short      status;      /* status field */

      /* Block Transfer Data Buffer */
      unsigned short      data[MAX_BT_DATALEN];

} btentry;
```

cmds.h

```
/*
**
** CMDS.H -- function prototypes for cmds.c
**
*/

void   wakeup                   (channel far *);
int    setup                    (channel far *, setup_data far *);
int    autocfg                  (channel far *, link_config far *);
int    scanlist                 (channel far *, scanlist_data far *);
int    faultgroup               (channel far *, fdg_data far *);
int    setmode                  (channel far *, int);
int    linkstatus               (channel far *, link_config far *);
void   reset                    (channel far *);
void   read_input_word          (channel far *, int, int, unsigned short far *);
void   write_output_word        (channel far *, int, int, unsigned short);
int    queue_oneshot_bt         (channel far *, bt_data far *);
int    configure_continuous_bt  (channel far *, bt_data far *);
int    oneshot_bt_complete      (channel far *, bt_data far *);
int    continuous_bt_update     (channel far *, bt_data far *);
```

bus.h

```
/*
**
** BUS.H -- prototypes for bus utility functions
**
*/

void init_bus_ops         (void);
void map_channel          (channel far *);
void unmap_channel        (channel far *);
int  wait_for_cmdcmpl     (channel far *);
```

utils.h

```
/*
**
** UTILS.H -- prototypes for utility functions
**
*/

void assert_cmd           (channel far *);
void read_csa             (channel far *, csa_data far *);
int lock_gda              (channel far *);
void unlock_gda           (channel far *);
int lock_conbt_entry      (btentry far *);
void unlock_conbt_entry   (btentry far *);
void print_link_config    (channel far *, link_config far *);
void print_bt_data        (bt_data far *);
void blockcpy             (void far *, void far *, size_t);
int sv2_lock              (unsigned short far *);
```

Programming Block Transfers

Here is a brief review of the protocol to issue a block-transfer command:

1. Test and set the scanner's semaphore.
2. Place the command code and other required data in the scanner's control/status and general data area.
3. Interrupt the scanner to let it know it should process your command.

   The scanner puts the block-transfer request in its queue and sends a confirmation status code. The scanner then interrupts the VME master processor.
4. The application program should acknowledge the interrupt and clear the semaphore.

Upon issuing a block-transfer request, the scanner writes 0x2F in the confirmation status byte to indicate that the scanner accepted the request and put it into its queue. When the scanner actually completes the block-transfer, the scanner sets the semaphore and then it updates this data in the following order:

1. address
2. block transfer tag
3. length of data
4. confirmation status
5. block transfer read data
6. command complete interrupt (if enabled)

The scanner executes block-transfers in run and test modes. In program mode, the scanner only queues block-transfers.

The programming example on page 8-13 shows how to program a single block-transfer command in the scanner. These following examples show how to update block-transfer data and how to test for complete block-transfer operations.

Programming Examples

```
/*
**
** READ_INPUT_WORD() -- read from selected input image table entry
**
**     This routine reads the specified entry in the discrete input table.
**
**     parameters:         pointer to channel structure
**                         target rack number
**                         target IO group word (0-7)
**                         pointer to rack/IO group word value
**     return values:      none
**
*/

void
read_input_word (channel far *chan, int rack, int iogrp,
       unsigned short far *rval)
{
       unsigned short far *wordptr;
       unsigned offset = ((rack * 8) + iogrp);

       /* map channel & establish pointers */
       map_channel(chan);
       wordptr = (unsigned short far *) chan->InputTable;
       wordptr += offset;

       /* read the word */
       *rval = *wordptr;

       /* unmap the channel */
       unmap_channel(chan);
       return;
}
```

```
/*
**
** WRITE_OUTPUT_WORD() -- write to selected output image table entry
**
**     This routine writes the specified entry in the discrete output table with the
**     given value.
**
**     parameters:         pointer to channel structure
**                         target rack number
**                         target IO group word (0-7)
**                         word value to write
**     return values:      none
**
*/

void
write_output_word (channel far *chan, int rack, int iogrp, unsigned short wval)
{
       unsigned short far *wordptr;
       unsigned offset = ((rack * 8) + iogrp);

       /* map the channel & establish pointers */
       map_channel(chan);
       wordptr = (unsigned short far *) chan->OutputTable;
       wordptr += offset;

       /* write the word */
       *wordptr = wval;

       /* unmap the channel */
       unmap_channel(chan);
       return;
}

/*
**
** ONESHOT_BT_COMPLETE() -- process a completed one-shot block transfer
**
**     This function should be called when it is known that a one-shot block transfer has
**     completed.  (This is typically signalled via a VME interrupt.)  The routine will
**     access the channel's Control/Status Area and General Data Area, placing the result
**     data in the bt_data structure passed by the caller.  If the completed operation
**     is a BT read, the routine will place the read data in the bt_data parameter's data
**     buffer field.
**
**     parameters:         pointer to channel structure
**                         pointer to the bt_data structure
**     return values:      -1     = General Data Area semaphore not locked
**                         ELSE   channel confirmation status word
**
*/

int
oneshot_bt_complete (channel far *chan, bt_data far *block)
{
       csa_data far              *csa;
       unsigned short far        *gda;
       unsigned short            i;
```

```
        /* map channel & establish pointers */
        map_channel(chan);
        csa    =      (csa_data far *)          chan->ControlStatusArea;
        gda    =      (unsigned short far *)    chan->GeneralDataArea;

        /* check for semaphore */
        if ((csa->semaphore & SEMAPHORE) == 0x0000) {
               /* no block transfer has completed */
               unmap_channel(chan);
               return -1;
        }

        /* get block transfer parameters */
        block->cnfstat      =      csa->cnfstat;
        block->cmd          =      csa->cmd;
        block->addr         =      csa->addr;
        block->tag          =      csa->tag;
        block->datalen      =      csa->datalen;

        /* if read */
        if (csa->cmd == CMD_OSBTREAD) {
               /* copy block data */
               for (i = 0; i < block->datalen; i++) {
                      block->data[i] = gda[i];
               }
        }

        /* release semaphore */
        unlock_gda(chan);
        unmap_channel(chan);
        return block->cnfstat;
}
```

Continuous Block-Transfers

Continuous block-transfer operations are similar to single block-transfer operations. They send and retrieve data the same way. The continuous block-transfer offers a way to continuously poll an adapter and reduce programming overhead.

While running a continuous block-transfer, the scanner prevents the scanner and a VME processor from simultaneously updating the same block table entry. Your application must resolve possible contention among multiple master VME processors simultaneously accessing the same block table entry.

Important: Your application should read block updates as highly critical sections of code. The application should have the master processor acquire a lock on the entry, copy the read/write data, and release the lock without pausing, sleeping, polling, or handling interrupts. As long as the master processor holds the lock, the scanner cannot update the block.

The programming example on page 8-15 shows how to program a continuous block-transfer command in the scanner. The following example shows how to update the block-transfer data.

Programming Example

```
/*
**
** CONFIGURE_CONTINUOUS_BT() -- configure a continuous block transfer read or write
**
**     This function is driven by the data contained in the bt_data structured provided by
**     the caller.  The command word (CNBTW/CNBTR), BT address word, tag word, interrupt
**     word, and BT transfer length word are placed in the Control/Status Area.  The command
**      is then asserted and its results returned to the caller.
**
**     parameters:         pointer to channel structure
**                         pointer to the bt_data structure
**     return values:      -1     = unable to lock general data area semaphore
**                         -2     = did not get cmd-complete interrupt
**                         ELSE   channel confirmation status word
**
*/

int
configure_continuous_bt (channel far *chan, bt_data far *block)
{
       csa_data far              *csa;
       unsigned short far        *gda;
       unsigned short            res;

       /* map channel & establish pointers */
       map_channel(chan);
       csa    =     (csa_data far *) chan->ControlStatusArea;
       gda    =     (unsigned short far *) chan->GeneralDataArea;

       /* get the semaphore */
       if (lock_gda(chan) != 0) {
              unmap_channel(chan);
              fprintf(stdout, "osbtread: lock aquire FAILED\n");
              return -1;
       }

       /* set up the control/status area */
       csa->cnfstat =     0x000F;            /* confirmation status word */
       csa->cmd     =     block->cmd;        /* command word */
       csa->addr    =     block->addr;       /* address word */
       csa->tag     =     block->tag;        /* tag word */
       csa->irq     =     block->irq;        /* interrupt word */
       csa->datalen =     block->datalen;    /* length of data word */
```

```
        /* send the command interrupt */
        assert_cmd(chan);

        /* poll for command result */
        if (wait_for_cmdcmpl(chan) == FALSE) {
              /* clear the semaphore */
              unlock_gda(chan);
              unmap_channel(chan);
              fprintf(stdout, "continuous_bt: wait failure, lock cleared\n");
              return -2;
        }

        /* clear the semaphore & unmap channel */
        res = csa->cnfstat;
        unlock_gda(chan);
        unmap_channel(chan);
        return (int) res;
}
```

Communicating with PLC-5 Processor in Adapter Mode

A common VME control system using a scanner has the scanner communicating with a PLC-5 processor set for adapter mode. In this system, the scanner communicates with the PLC-5 processor over a remote I/O link and the PLC-5 processor emulates a 1771-ASB module.

The PLC-5 processor in adapter mode has only local I/O. The rack number of the adapter-mode processor determines the addresses you use.

Direct Transfer

You specify configuration files (output source file and input destination file) in the adapter-mode processor If you want the scanner to control outputs of the adapter-mode processor, write ladder logic in the adapter-mode processor to move the data from its input destination file to its output image table. Use XIC and OTE instructions for bit data; use move and copy instructions for word data.

If you want the scanner to read data from a data file in the adapter-mode processor, write logic in the adapter-mode processor to move that data to its output source file for transfer to the scanner's input image table.

The adapter-mode processor transfers 2, 4, 6, 8 words, depending on whether it is configured as a 1/4, 1/2, 3/4, or full rack. For example, if the adapter-mode processor is configured as a full rack, scanner and adapter-mode processor exchange 8 input words and 8 output words each I/O scan.

Important: Of the words that are transferred back and forth between the scanner and PLC-5 processor, the first word in the input and output image table contains rack status and block-transfer status. Don't program any data in these words.

Use the remaining input words and output words to move data between the scanner and the PLC-5 processor. If you need to transfer more than 7 words, use a block-transfer, which can transfer up to 64 words of data.

Block-Transfer

Adapter-mode block-transfers are essentially continuous. As soon as a transfer is completed, another block-transfer is queued immediately in the PLC-5 processor; the processor then waits (with a buffered snap-shot of the data) for the scanner to perform another block-transfer request. The data that is transferred after the request is data from the previous block-transfer. For example, if the scanner performs a block-transfer request from the adapter-more processor every 500 msec, the data is at least 500 msec old.

The scanner controls the actual communication transmission of the block-transfer request. The adapter-mode processor controls the:

- actual number of words of data that is transferred
- location from which the data is transferred

Do not use ladder-logic block-transfer instructions for the adapter-mode processor. You configure the block-transfers when you configure the adapter channel.

Important: Adapter-mode block-transfer reads and writes in the same group/module location must have the same length.

Affects of block-transfers on direct transfers
You can have as many as 15 writes and 15 reads. Each block-transfer to a particular group/module location uses the I/O addresses for that rack/group for status bits. These locations are lost to direct transfer. Therefore, if you configure all available 15 block-transfer read/write pairs, no bits will be available for direct transfer.

Design Tip

Do not program a block-transfer to group 0, module 1 because this area of the direct transfer configuration file is used for communication status exchanges between the scanner and the adapter-mode processor.

Chapter 9

Troubleshooting

Using This Chapter

If you want to read about:	go to page:
indicators	9-1
error codes	9-2
troubleshooting suggestions	9-4

Indicators

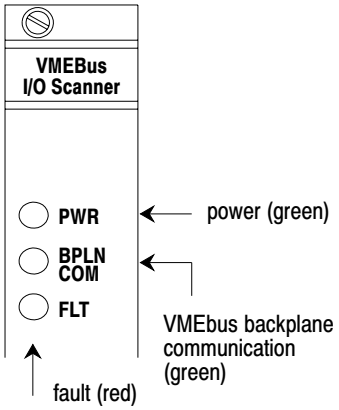
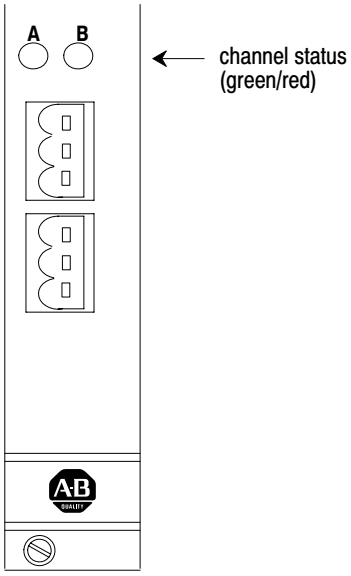


Table 9.A
Significance of scanner indicators

When this indicator:	is:	it means:
PWR (power) green LED	illuminated	power is applied to the module
BPLN COM (backplane communication) green LED	illuminated for approximately a half second	a VMEbus access is made to the scanner board
FLT (fault) red LED	illuminated	the scanner board is reset, performing a self test, or a fault has been detected



Note: channel B is on the 6008-SV2R scanner only

Table 9.B
Significance of channel status indicators

When the status indicator is:	the scanner:
off	is off line
green	is on line, in run mode, and scanning the racks in the scan list
blinking green	is on line, in run mode, and scanning only some of the racks in the scan list
red	has an unrecoverable fault
blinking red	has a recoverable fault

Error Codes

Table 9.C lists the error codes the scanner can write to the confirmation status word in the control/status area for a scanner channel. The error code uses the lower byte of the confirmation status word.

The scanner cannot generate any error codes that are not listed below. If an error code shows up in the confirmation status word that is not listed here, it is likely that another VME master processor accidentally wrote a value into that word.

Table 9.C
Error Codes

Code (hex):	Significance:	Corrective Action:
00	Successful command.	none
01	**Unspecified error.** The error is unknown and indicates serious scanner malfunction.	Consult the factory and/or replace the scanner board.
10	**Unknown command.** The VME master processor put an unknown command in the command field.	Use a valid command code.
11	**Illegal command.** The VME master processor issued a command without fully specifying or incorrectly specifying one or more parameters needed with the command.	Check the parameters that were provided to the scanner module.
12	**Management command already pending.** The VME master processor issued another command while one is in process.	Wait for the scanner to finish the current command.
13	**Program mode required.** The VME master processor issued a command that is valid only in the program mode.	Place the scanner in the program mode and re-issue the command.

Code (hex):	Significance:	Corrective Action:
14	**Block transfer queue full.** The VME master processor issued a block transfer command when the scanner's block transfer queue was full.	Wait for one or more of the block transfers already in the scanner's queue to finish, then re-issue the command.
15	**Invalid length of data.** The VME master processor specified a length that cannot be used for the command issued.	Specify a proper length.
16	**Illegal configuration.** The VME master processor issued a: • SCAN LIST command that deletes an adapter listed in a FAULT DEPENDENT GROUP. • command placing an adapter not in the scan list into a FAULT DEPENDENT GROUP.	**1.** Adjust the fault dependent group and then re-issue the scan list. **2.** Adjust the scan list and the re-issue the fault dependent group command.
23	**Block transfer timeout.** A block transfer specified by the BT tag did not complete within 4 seconds and was aborted.	**1.** Make sure a block transfer module is in the slot specified. **2.** Make sure the chassis holding the module is turned on. **3.** Make sure the adapter is in the scan list. **4.** Make sure the scanner is not in the program mode. **5.** Check the length of data value to make sure it matches the module, or make the value zero. **6.** Check the manual for the module in use.
25–2E 30–43	**I/O link error.** A failure occurred in the transfer of information between the I/O module and the scanner. An application program cannot cause these errors, because they are associated with the communication protocol on the I/O link.	**1.** Check the block transfer hardware configuration. **2.** Eliminate electrical noise that may interfere with remote I/O communication. **3.** Use 57.6 kbps instead of 115.2 kpbs. **4.** Ensure proper grounding of your I/O racks. **5.** Consult the factory and/or replace the block transfer module. **6.** An error occurred within the scanner. Try cycling power before removing the scanner.
2F	**Block transfer accepted.** The block transfer was accepted by the scanner and is now in the scanner's queue.	none

Code (hex):	Significance:	Corrective Action:
90	**Ready confirmation.** Used only when the scanner module is turned on and shows that the scanner has passed its self tests.	none
91	**SYSFAIL**. SYSFAIL has been asserted on the VMEbus.	Remove the card that is asserting SYSFAIL on the VMEbus. Reset the scanner with the RESET command.
92	**ACFAIL.** ACFAIL has been asserted on the VMEbus.	Power to the system is going to be removed shortly.
93	**Spurious interrupt to the scanner CPU.**	Reset the scanner with the RESET command. Eliminate any electrical noise to the VMEbus.

Troubleshooting Suggestions

The following table describes some problems you may encounter and possible solutions to them.

Problem:	Possible Solution:
The scanner won't communicate with the VME master processor (as indicated by the front panel LEDs).	Make sure the address of the global RAM is what you think it is. See chapter 2 for information on configuring the scanner.
	Check the SETUP parameters for error and re-issue the command.
	Check the memory map to ensure a clear 2K memory window.
Scanner is in sync with the VME master processor, but there is no communications with the I/O racks (as indicated by the front panel LEDs).	Check status of LEDs at the adapter: red light ON indicates a rack fault. green light ON indicates normal operation in run mode. green light blinking indicates normal operation in the program mode.
	Check the communication rate of the adapter and the SETUP command.
	Send an AUTOCONFIGURE command. Check the LEDs at the adapter.
	Send a LINK STATUS command determine if the adapter numbers returned correspond to those of the adapters. Use this formula to determine adapter number: Adapter # = (4 x rack) + (starting group/2)
	Check I/O cable for proper construction and connection.
	Check power at the I/O rack.
	Make sure the watchdog is disabled during testing, or is being called frequently enough to guard against timing out.
Discrete I/O updating, but block transfers are not.	Make sure you are addressing the proper module by using this formula: Module Address = (rack x 16) + slot
	Check the application program to make sure that the code for block transfer exactly follows the pseudo code found in the I/O Concepts Manual, publication 6008-6.5.1.
	Make sure the block transfer length is specified in the block transfer data packet (typically use zero for length)
Block transfers are working erratically.	Declare the block transfer data packet as "static." Use a different data packet for each block transfer.

Appendix A

Specifications

Environmental Specifications

Characteristic:		Value:
Temperature	Operating	0° to 60° C
		derated 2° C per 1000 ft (300m) over 6600 ft (2000m)
	Storage	-40° to 85° C
Humidity	Operating	5 to 95% noncondensing
	Storage	5 to 95% noncondensing
Altitude	Operating	0 to 10,000 ft (3000 m)
	Storage	0 to 40,000 ft (12,000 m)
Vibration	Operating	2.5 g peak (max)
	Storage	5.0 g peak (max)
		acceleration over 5-500 Hzsine wave (point-to-point) 1 oct/min sine sweep
Shock	Operating	30 g, 11 ms duration, 1/2 sine shock pulse
	Storage	50 g, 11 ms duration, 1/2 sine shock pulse
Power	Maximum	5V dc at 2.5A
	Typical	5V dc at 2.3A

Performance Specifications

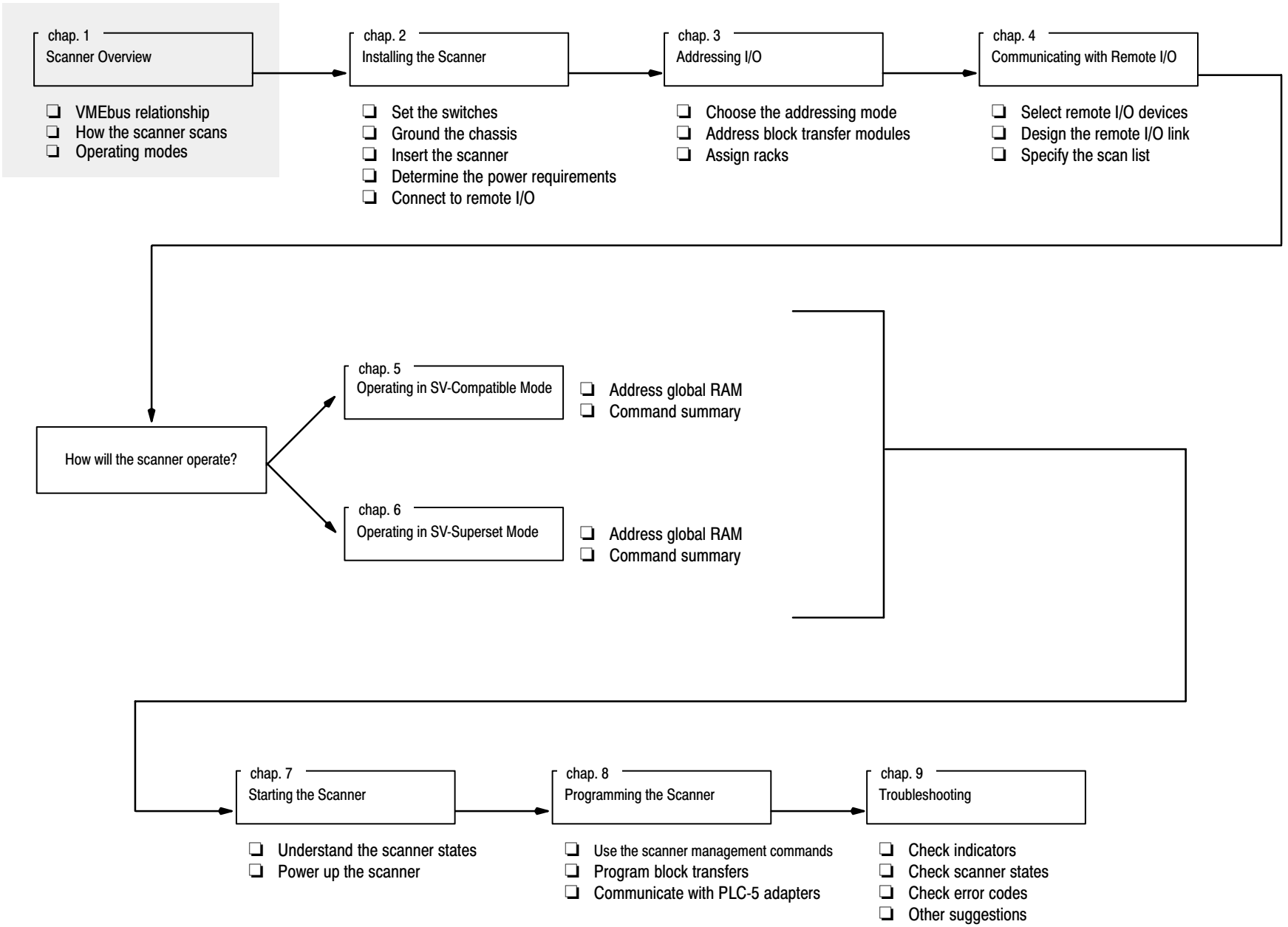
Scanner:		Scan time per logical rack (ms):
SV1R and SV2R	230.4 kbps	3
	115.2 kbps	6
	57.6 kbps	12

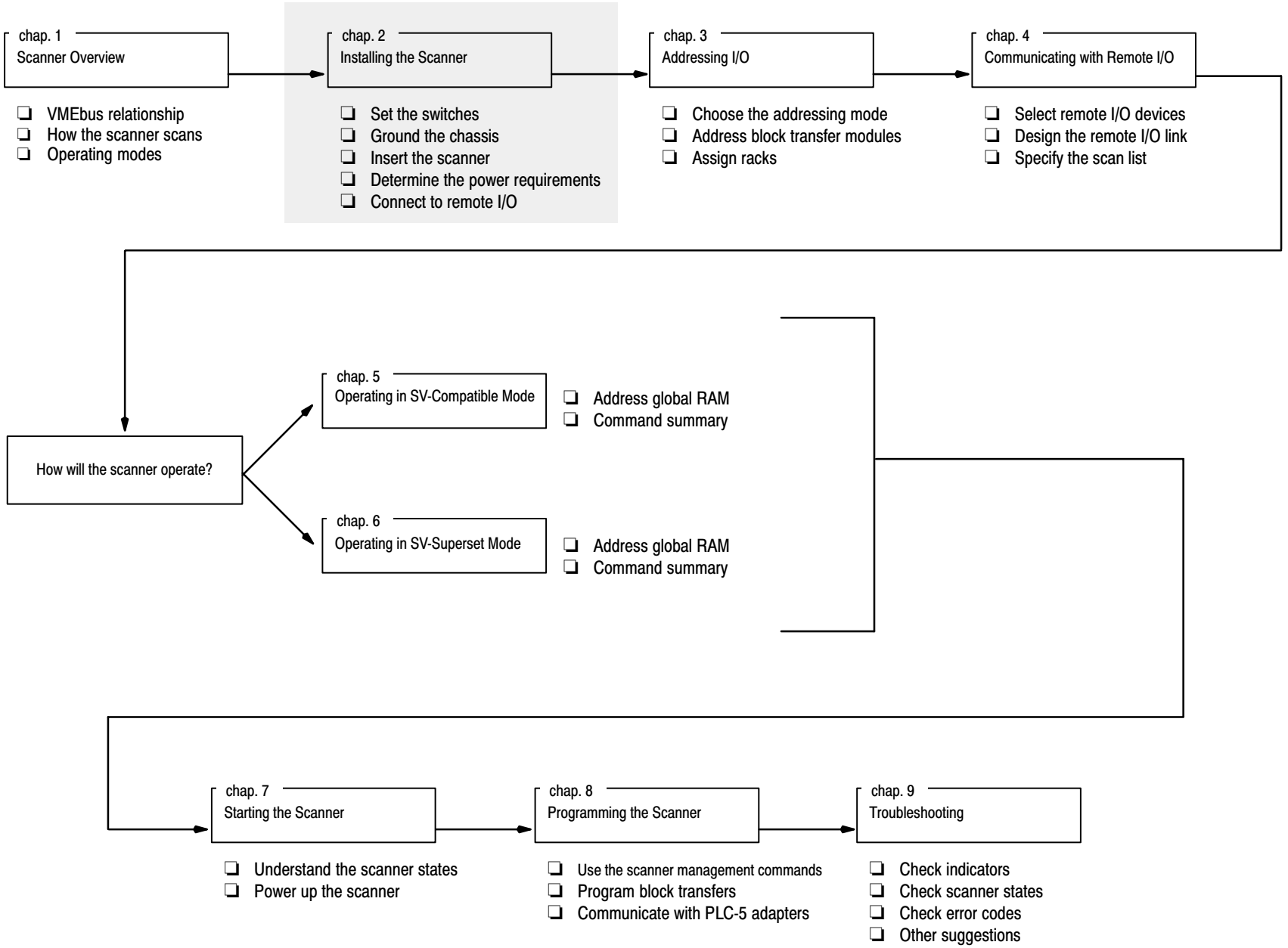
Command:	Execution time (ms):
SETUP	5
AUTOCONFIGURE	unbounded
SCAN LIST	5
FAULT DEPENDENT GROUP	5
SET MODE	5
LINK STATUS	1
BT WRITE	2
BT READ	2
CONTINUOUS BT WRITE	2
CONTINUOUS BT READ	2
RESET	unbounded

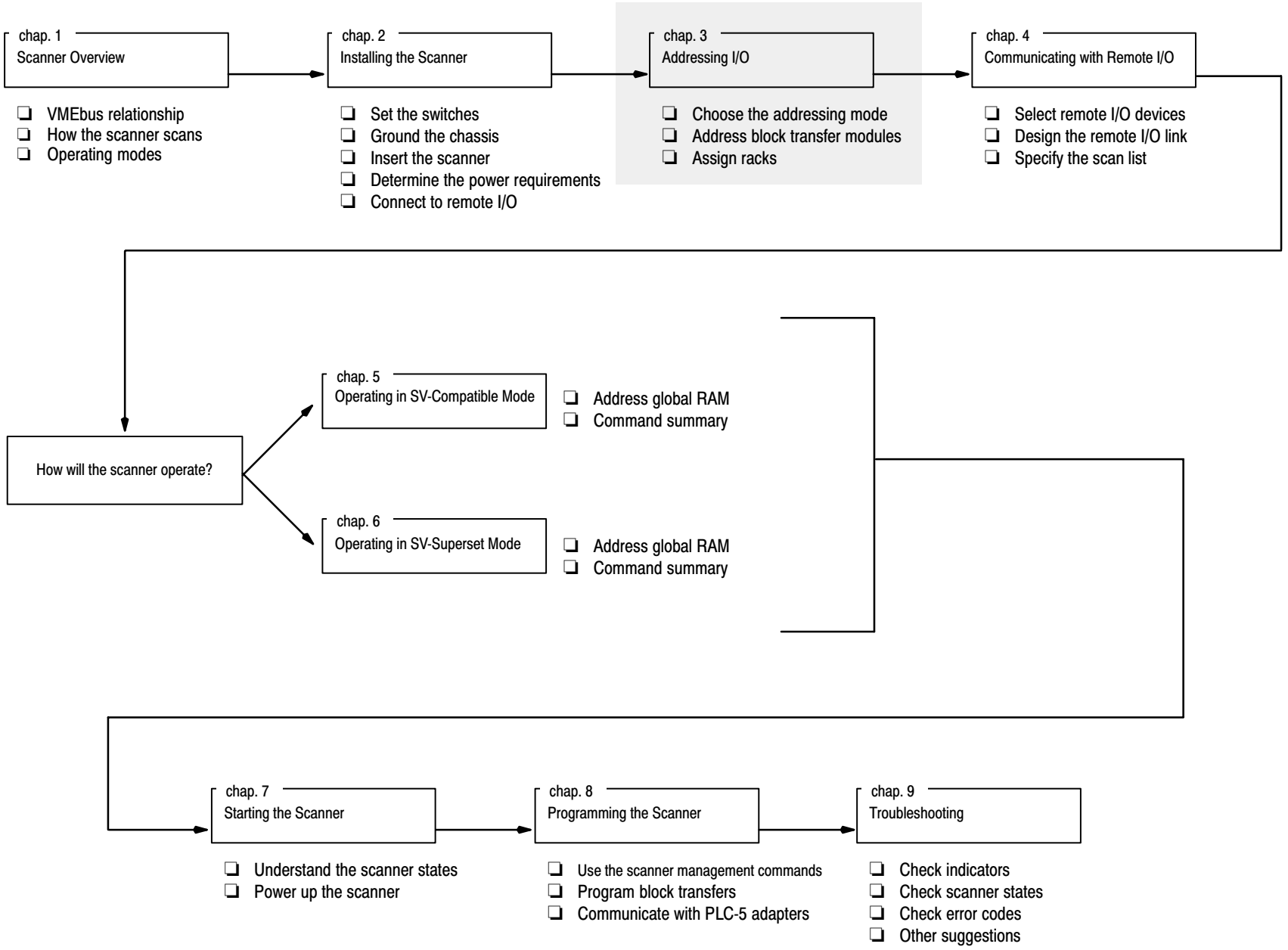
The AUTOCONFIGURE and RESET commands have unbounded execution times because the scanner must poll each legal address on the I/O link for each channel, incurring a timeout for each unoccupied address

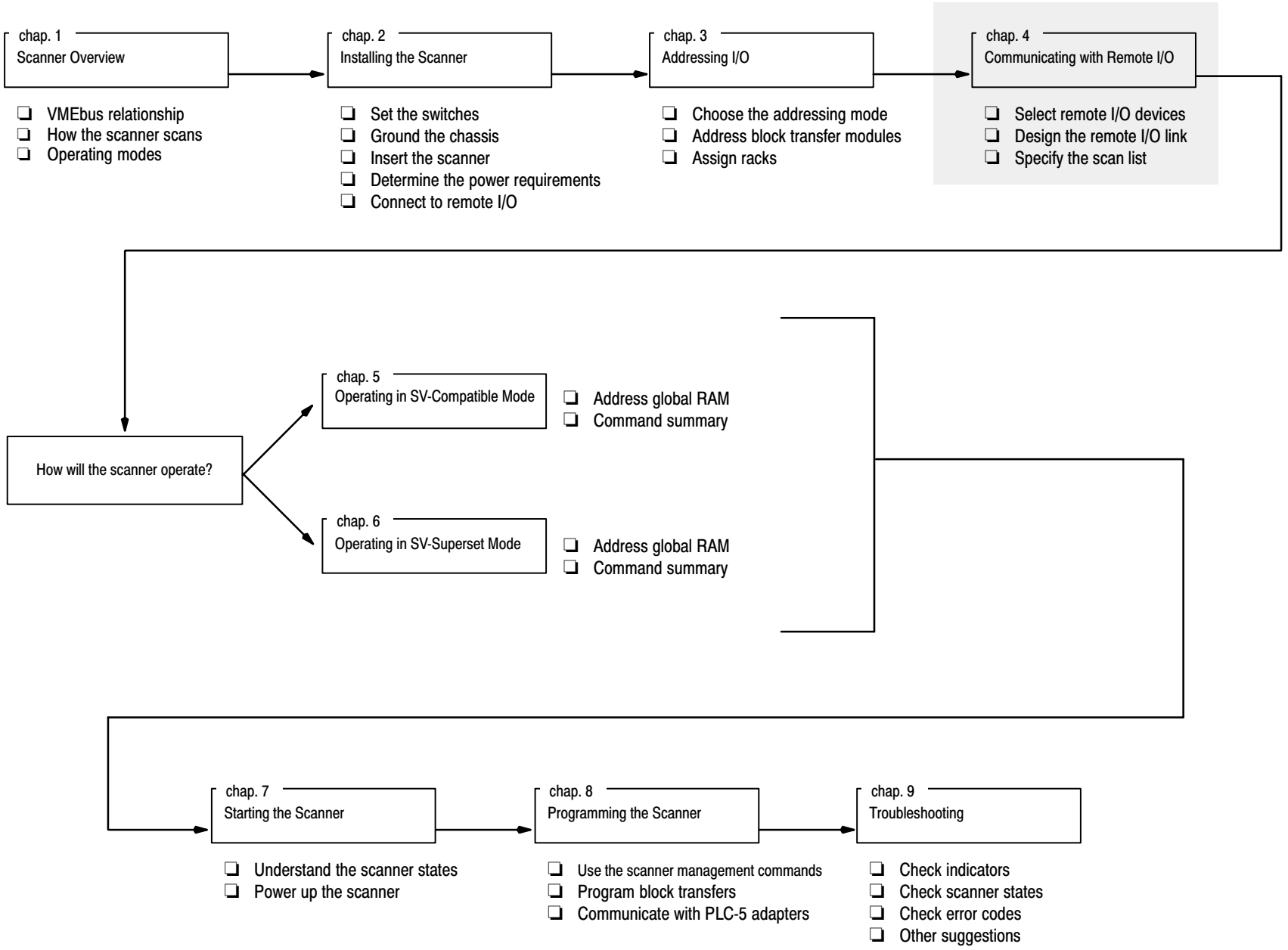
VMEbus Specifications

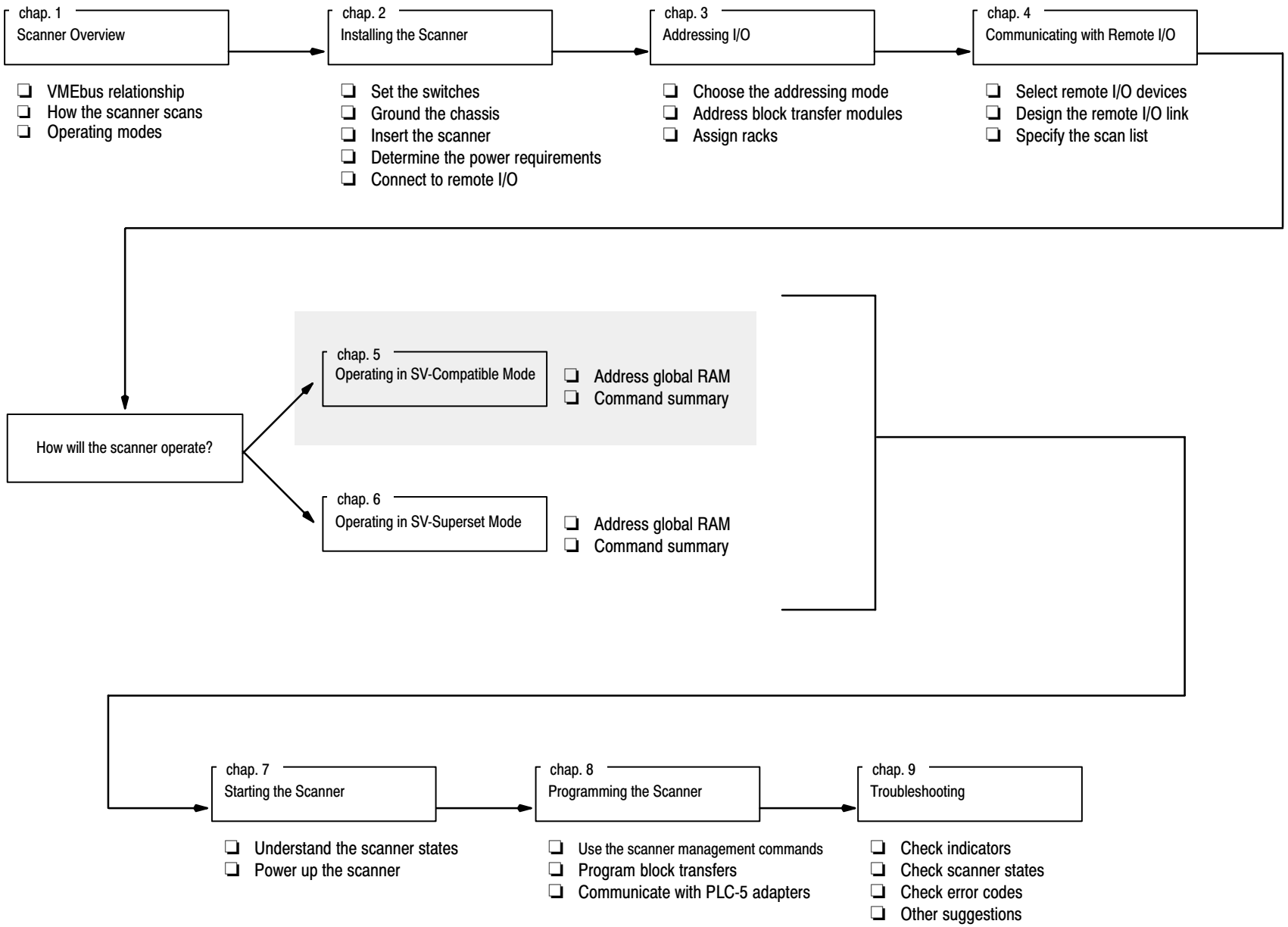
Characteristic (revision C.1):	Value:
Slave address	A16, A24
Slave transfer	D08(EO), D16
Interrupter	I(1–7), D08(O)

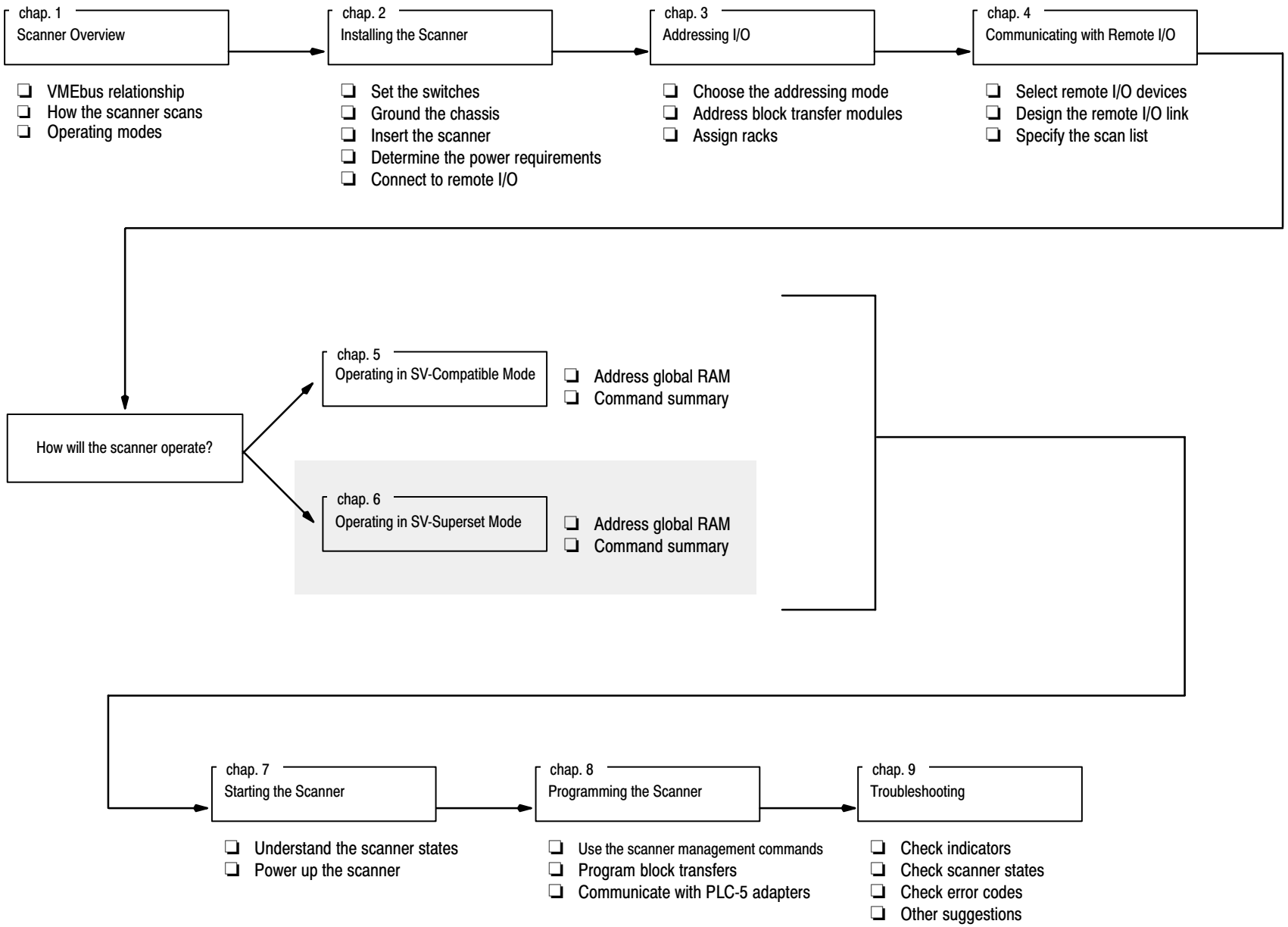

chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions

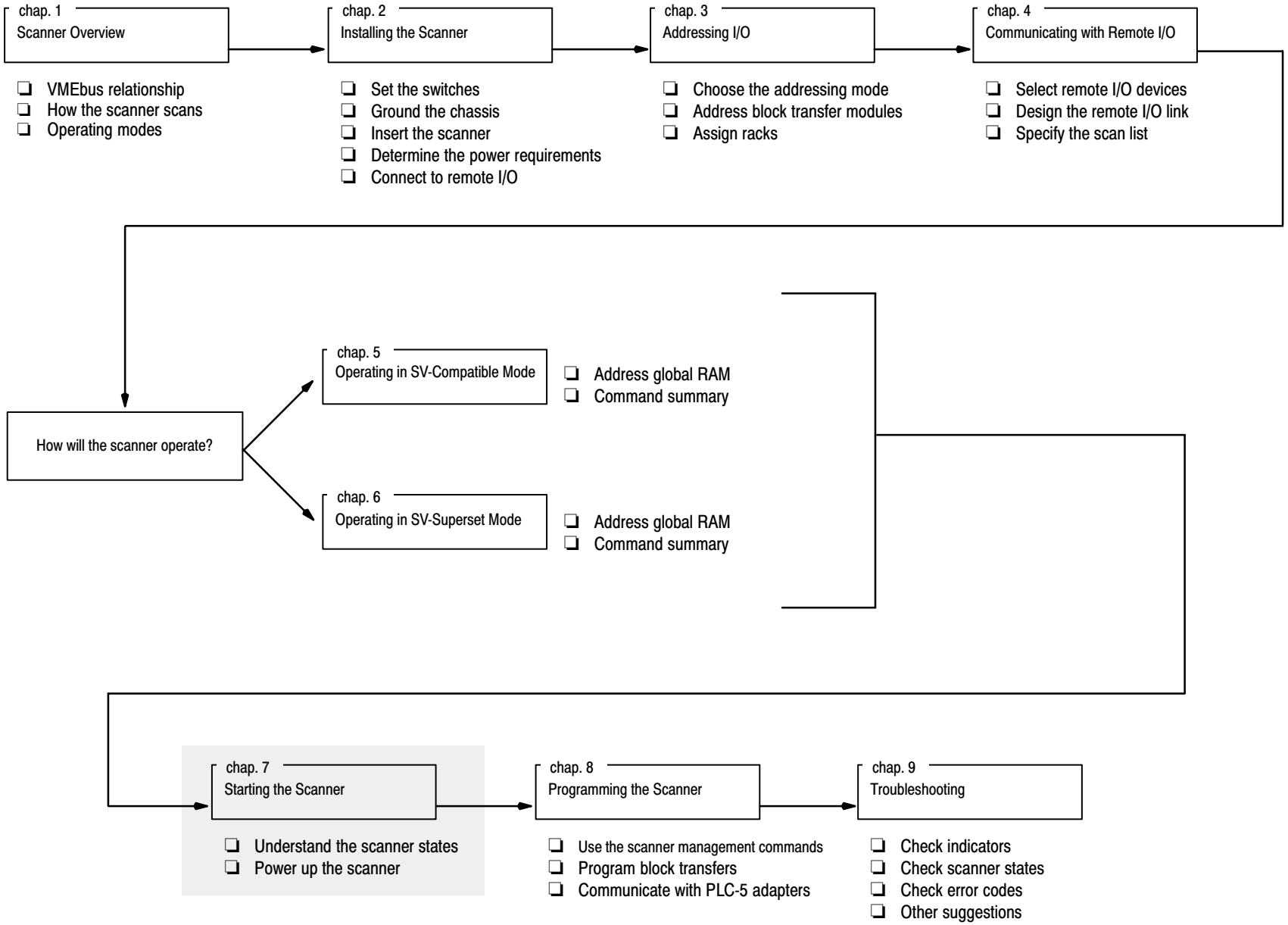

chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions

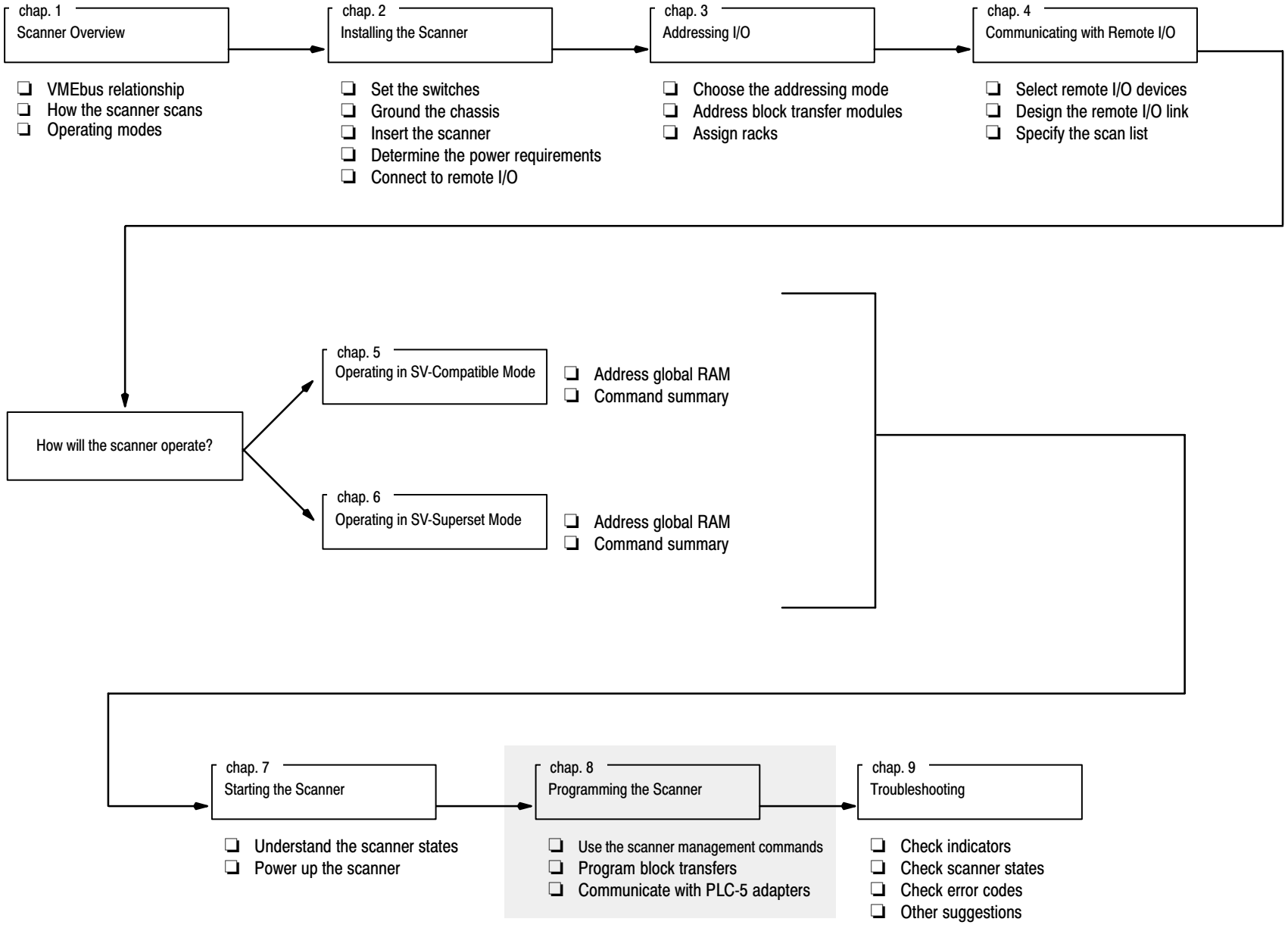

chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions

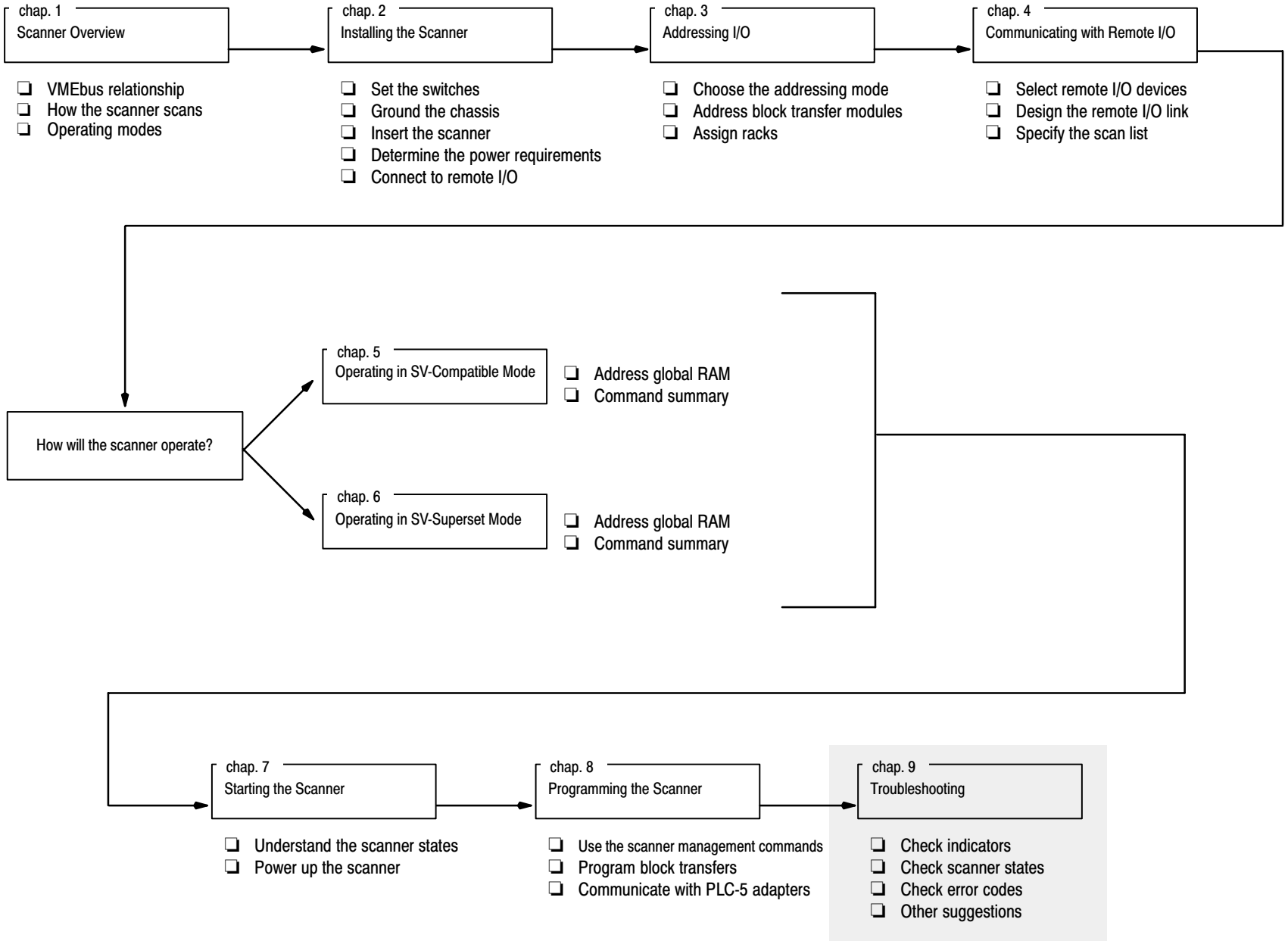

chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions


chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions


chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions


chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions


chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions


chap. 1
Scanner Overview
❏ VMEbus relationship
❏ How the scanner scans
❏ Operating modes
chap. 2
Installing the Scanner
❏ Set the switches
❏ Ground the chassis
❏ Insert the scanner
❏ Determine the power requirements
❏ Connect to remote I/O
chap. 3
Addressing I/O
❏ Choose the addressing mode
❏ Address block transfer modules
❏ Assign racks
chap. 4
Communicating with Remote I/O
❏ Select remote I/O devices
❏ Design the remote I/O link
❏ Specify the scan list
How will the scanner operate?
chap. 5
Operating in SV-Compatible Mode
❏ Address global RAM
❏ Command summary
chap. 6
Operating in SV-Superset Mode
❏ Address global RAM
❏ Command summary
chap. 7
Starting the Scanner
❏ Understand the scanner states
❏ Power up the scanner
chap. 8
Programming the Scanner
❏ Use the scanner management commands
❏ Program block transfers
❏ Communicate with PLC-5 adapters
chap. 9
Troubleshooting
❏ Check indicators
❏ Check scanner states
❏ Check error codes
❏ Other suggestions

H

I

K

L

N

O

P

R

S

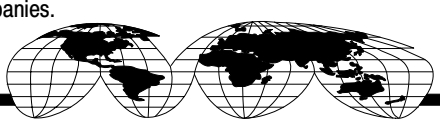
T

U

Rockwell Automation

Allen-Bradley

Allen-Bradley, a Rockwell Automation Business, has been helping its customers improve productivity and quality for 90 years. We design, manufacture, and support a broad range of control and automation products worldwide. They include logic processors, power and motion control devices, man-machine interfaces, sensors, and a variety of software. Rockwell is one of the world's leading technology companies.



Worldwide representation.

Algeria • Argentina • Australia • Austria • Bahrain • Belgium • Brazil • Bulgaria • Canada • Chile • China, PRC • Colombia • Costa Rica • Croatia • Cyprus • Czech Republic Denmark • Ecuador • Egypt • El Salvador • Finland • France • Germany • Greece • Guatemala • Honduras • Hong Kong • Hungary • Iceland • India • Indonesia • Israel • Italy Jamaica • Japan • Jordan • Korea • Kuwait • Lebanon • Malaysia • Mexico • New Zealand • Norway • Oman • Pakistan • Peru • Philippines • Poland • Portugal • Puerto Rico Qatar • Romania • Russia–CIS • Saudi Arabia • Singapore • Slovakia • Slovenia • South Africa, Republic • Spain • Switzerland • Taiwan • Thailand • The Netherlands • Turkey United Arab Emirates • United Kingdom • United States • Uruguay • Venezuela • Yugoslavia

Allen-Bradley Headquarters, 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414 382-2000 Fax: (1) 414 382-4444

Publication 6008-6.5.11 September 1995
Supersedes Publication 6008-6.5.2 August 1990

PN 955119-08