ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Abaco Systems / Radstone PMV68 ACM1-100E* at our website: *Click HERE*

PMV 68 ACM-1
Manual

Revision B

Publication No. HH681ACM1

Radstone Technology PLC
Water Lane
Towcester
Northants
NN12 6JN

Telephone: 01327 359444
Facsimile: 01327 358112

© Radstone Technology PLC 1995. All rights reserved.
See overleaf for printing history.

This publication is issued to provide outline information only which (unless agreed by the Company in writing) may not be used, applied or reproduced for any purpose or form part of any order or contract or be regarded as a representation relating to products or services concerned. The Company reserves the right to alter without notice the specification, design, price or conditions of supply of any product or service.

March 1995

Trademarks

Radstone, the Radstone symbol and PMV 68 are trademarks of Radstone Technology PLC.

Printing History

The document revision state refers **only** to the **manual**, **not** the board revision.

Revision	Issue A	Issue 1	Rev -	Rev A	Rev B		
Date	1989	5/90	11/91	10/92	3/95		

Related Documents

VMEbus Specification (ANSI/IEEE Std. 1014-1987)
IEEE Std 1101.2-1992 Mechanical Core Specifications for Conduction-Cooled Eurocards.
Intel M80C186 CHMOS High Integration 16-bit Microprocessor Technical Summary (data sheet)
EXAR XR-88C681/40 CMOS Dual Channel UART Data Sheet

Table of Contents

Chapter 1
Introduction

Scope 1-1
Conventions 1-1
PMV 68 ACM-1 1-1
PMV 68 Family 1-3
Mechanical Overview 1-4
PMV 68 ACM-1 Functional Overview 1-5

Chapter 2
Specifications

Common Specifications 2-1
General 2-1
Electrical 2-1
Safety Requirements 2-2
Mechanical 2-2
Front Panel 2-4
PMV 68MI Specification 2-5
PMV 68MA Specification 2-6
PMV 68RI Specifications 2-7
PMV 68R Specification 2-7
PMV 68E Specification 2-8
Reliability 2-9

Chapter 3
VMEbus Connector P2 Assignments

Connector P2 Assignment for a Standard Backplane 3-1
Channel G modem control in RS422 mode 3-2
Connector P2 Assignment for a Dedicated Backplane 3-3
Connector P2 User-Defined Pin Functions 3-5
ACM-1 Board Address Select Links 3-5
Channel A 3-5
Channel B 3-6
Channel C 3-6
Channel D 3-7
Channel E 3-7
Channel F 3-8
Channel G (RS423 Mode) 3-8
Channel G (RS 422 Mode) 3-9
Channel H (When Channel G is in RS423 Mode) 3-9
Channel H (Alternative pinout when Channel G is in RS422 Mode) 3-10

Chapter 4 - Board Configuration

Board Configuration for PWB No. 419731xx-xxx 4-2
Fixed Configuration 4-3
Dual Port RAM Address Pin Options 4-3
Address Links Option 4-4
Dual Ported RAM Access Time Options 4-6
Serial Channels - Transmit Data RS422/RS423 Mode Option 4-7
Serial Channels - Receive Data RS422/RS423 Mode Option 4-8
RS422 Terminator Resistors Option 4-9
Signal Ground to Local Ground Reference Option 4-10
Board Address Selection by Backplane Link Options 4-10
P2 Row B Link Options 4-11
Default Settings 4-14
Board Configuration for PWB No. 419732xx-xxx 4-17
Fixed Configuration 4-18
Dual Port RAM Address Pin Options 4-18
Address Links Option 4-19
Dual Ported RAM Access Time Options 4-21
Serial Channels - Transmit Data RS422/RS423 Mode Option 4-22
Serial Channels - Receive Data RS422/RS423 Mode Option 4-23
Serial Channel G Modem RS422/RS423 Mode Option 4-24
RS422 Terminator Resistors Option 4-25
Signal Ground to Local Ground Reference Option 4-26
Board Address Selection by Backplane Link Options 4-26
P2 Row B Link Options 4-27
Default Settings 4-30

Chapter 5
On-board Memory

On-board ROM 5-1
On-board Global (Dual Ported) RAM 5-1
On-board Local RAM 5-1

Chapter6
On-board Peripherals

Local Control Register 6-1
Interrupt Level Control 6-1
PMV-BITE Self-test Fail Indicator 6-2
MPE Fail Indicators 6-2
Timer-1 Control 6-2
VMEbus Interrupt IRQn* Control 6-2
Watchdog Reset Timer Enable Control 6-3
System Fail Indicator 6-3
Local Status Registers 6-3

Status Register SR0 6-3
Status Register SR1 6-4
Watch-dog Reset Timer 6-4
VME Interrupt Generator 6-5
Programmable Interrupt Controller 6-5
Programmable Interval Timers 6-6
Programmable Wait State Generator 6-6
88C681 Dual Universal Asynchronous Receiver/Transmitter 6-7
Addressing the On-board Peripherals 6-7

Chapter 7
Operational Firmware

Introduction 7-1
Communicating with the Firmware 7-1
Tx Enable and Rx Enable Bytes 7-2
Mode Word 7-2
Tx and Rx Status Words 7-2
Speed Longword 7-3
Size/Parity Byte 7-3
Stop Byte 7-4
VME Interrupt Control Word 7-4
Tx Null Byte 7-4
XON Character 7-5
XOFF Character 7-5
Rx Lower full 7-5
Rx Upper Full 7-5
Enable XOFF 7-5
Enable RTS/CTS 7-5
Set DTR 7-5
DSR/DCD Status 7-5
Tx Buffer Address Longword 7-6
Current Tx Pointer Longword 7-6
Tx Buffer Length Longword 7-6
Rx Buffer Address Longword 7-6
Current Rx Pointer Longword 7-6
Rx Buffer Length Longword 7-6
Performing a Transfer 7-7
Initialization 7-7
Executing the Transfer 7-7
Transfer Modes 7-7
Status Reporting 7-9

Chapter 8
BITE Self-Test

Overview 8-1
MPE Test 8-1
MPE Operations 8-1
MPE Failures 8-1
BITE Test Levels 8-2
BITE Operations 8-2
Triggering BITE with a Software Command 8-3
BITE Configuration Data 8-5
BITE Fail Data 8-6
ACM Buffer Memory Tests 8-6
ACM On-Card BITE Tests 8-8
ACM Transfer Tests 8-8

Appendix A
Example Programs

Example 1 - ACM-1 Transfer Test A-1
Example 2 - Test Program A-6
Example 3 - IFX Driver Routine A-16
ACM-1 Machine Code Routines for IFX Driver A-28

Chapter 1
Introduction

Scope

This manual describes the PMV 68 ACM-1 (Asynchronous Communications Multiplexer), which is designed for use with the PMV 68 family of processors. The ACM-1, in common with all other PMV 68 modules, is available in five environmental build standards.

Conventions

This manual adopts the following conventions. As with the 80C186, bits are numbered from 0 to n where 0 is the least significant bit and n is the most significant. Also 'longword', 'word' and 'byte' mean a 32-bit word, a 16-bit word and an 8-bit word respectively.
Signal names follow the ANSI/IEEE Std 1014-1987.
Signal names ending with *, e.g. TXDA*, are **active low**, all others are active high
GND and V_{CC} mean zero volt and five volt supply rails.
The $ prefix indicates a hexadecimal value.

PMV 68 ACM-1

The PMV 68 ACM-1 board provides eight independent full duplex serial communications channels, each capable of asynchronous operation up to 38.4 kbit/s. An on-board 80C186 16-bit processor provides control over the individual channel communications and processing power for BITE, the program resident in the on-board EPROM.

The main features of the ACM-1 are:

- Supports A32/D16 or A24/D16 VMEbus interface
- 80C186 16 bit processor
- 8 Full Duplex Asynchronous Serial Channels
- Up to 9600 bit/second asynchronous data rate on all 8 channels simultaneously
- Any standard asynchronous data rate from 50 to 38.4 kbit/second
- Link selectable RS423/RS422 for each serial channel
- Three 16-bit timers
- Interrupter to any software selectable VMEbus interrupt level with 8-bit programmable status/ID
- On-board 64 kbytes of dual ported RAM accessible from the VMEbus
- On-board local RAM
- On-board 128 kbytes of EPROM
- BITE self test
- Board address selectable by backplane links (P2)

Figure 1 - PMV 68 ACM-1

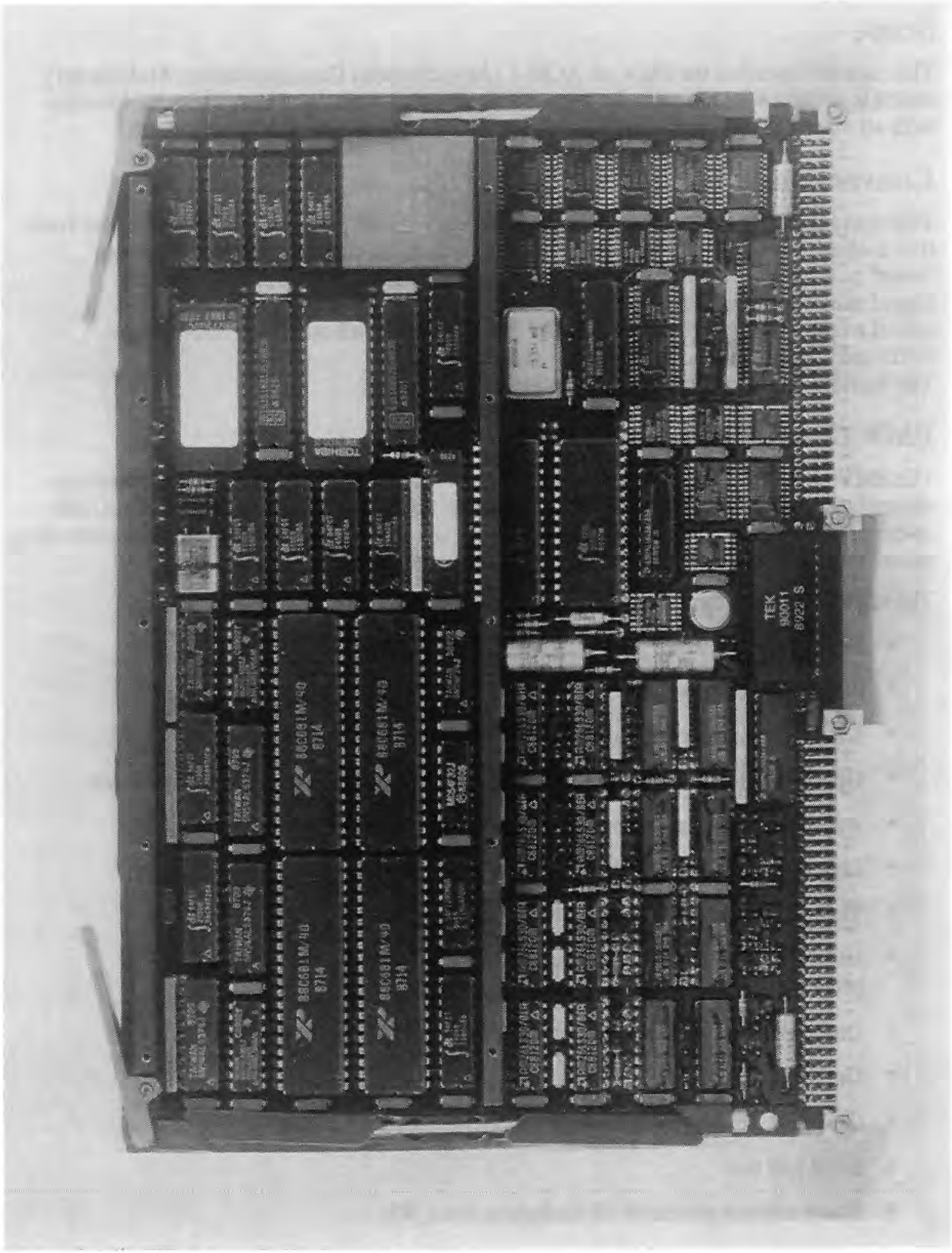
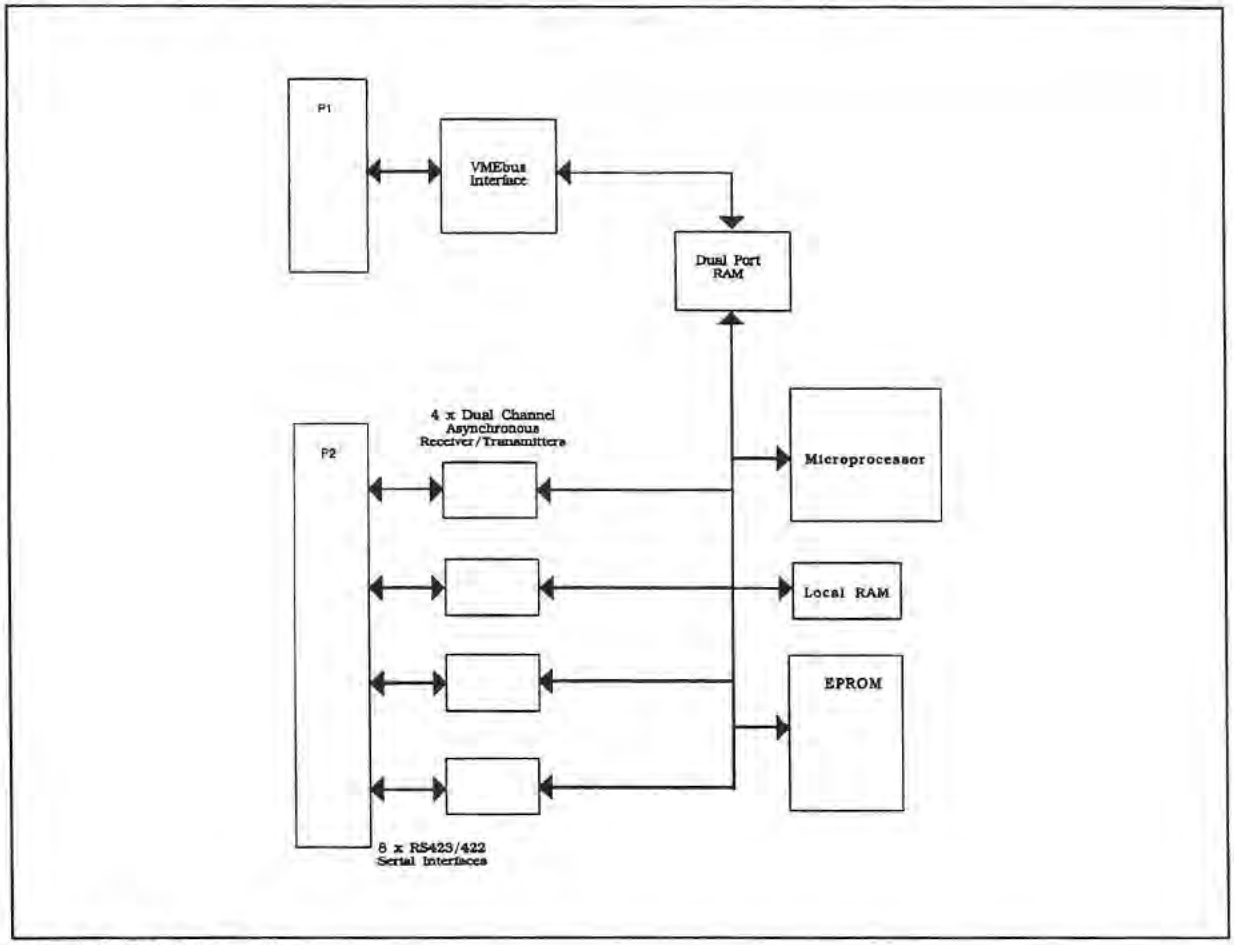


Figure 2 - PMV 68 ACM-1 Block Diagram



PMV 68 Family

The ACM-1 is a member of the PMV 68 VMEbus family, which is a range of high performance processor, memory and I/O boards. PMV 68 satisfies the demands of today's military system designers for reliable, high performance, modular processing systems. It is complemented by ATR boxes of various sizes, plus the necessary software and hardware development facilities. For up-to-date details of this ever growing range, please contact your supplier. The major features of PMV 68 include:

- Totally VMEbus compatible family of modules
- Multiprocessor dual bus architecture
- Communications interfaces
- 32-bit architecture
- ATR box compatible (per IEEE Std 1101.2-1992)
- Commercial VMEbus rack compatible
- EPROM stored Built In Test Equipment (BITE) programs
- Monitor (PLUM-4) supplied as standard
- Five 100% software compatible build standards

Mechanical Overview

All PMV 68 modules come in five electrically compatible build standards, each carefully tailored to a particular set of requirements and environments. All five standards fully support the power and versatility of the VMEbus, so no matter how large or diversified your project, MIL-STD-VME assures absolute compatibility at all stages of development. Due to the common circuit design, this manual describes a 'generic' standard, the exception being the individual mechanical specifications of each standard. A short description of each standard is given below.

PMV 68MI

At the top of the range, the MI-style complies with the most stringent military design rules and represents the highest level of reliability and environmental resistance available. Designed primarily for use in sealed ATR chassis, MI-style boards feature MIL-STD-454 qualified components, an integral thermal management layer and incorporate a central stiffening bar for additional strength. Cooling is achieved through conduction of heat from the thermal management layer to the cold wall of the rack to which the boards are secured by screw driven wedgelocks.
MI-style boards are designed to withstand environments involving extremes of temperature, shock and vibration. Careful attention to design details ensures that the MI-style remains compatible with standard Commercial VMEbus racks.

PMV 68MA

Produced to the same strict military guide lines and specifications as the MI-style boards, using screened military components, MA-style boards are designed for mounting in conventional, rugged or custom VMEbus racking systems. The boards are mechanically stiffened and cooling is by forced-air.

PMV 68RI

Mechanically identical to the MI-style, the RI-style is populated with wide temperature range, industrial grade devices, offering a 100% compatible, economical alternative to MI-style, for use in Radstone ATR boxes and other conduction cooled environments.

PMV 68R

R-style boards are offered for those applications that have extended temperature, shock and vibration requirements, but can be served by conventional racking systems with forced-air cooling. These rugged boards consist of a double-Eurocard size printed wiring board fitted with wide temperature range, industrial grade components and incorporate a centre stiffening bar.

PMV 68E

E-style provides the ideal cost effective method of complete system development. As software compatibility throughout the build standards is absolute, a system intended for final implementation in a severe tactical environment can be developed and debugged at low cost, switching over to target style only in the final stages of system integration.
Each E-style assembly consists of a double-Eurocard size printed wiring board with high quality commercial (plastic encapsulated) ICs. The boards incorporate a centre stiffening bar and survive the same levels of shock and vibration as the MA and R-styles. E-style boards are designed for forced-air cooling.

PMV 68 ACM-1 Functional Overview

With standard VMEbus backplanes, the ACM-1 allows basic '3-wire' control on all 8 channels plus full modem control on 4 channels using RS423 mode or 3 channels when optionally linked for 1 channel in RS422 mode. This uses a 64-way (rows A and C) connector, and the ACM-1 can therefore be slotted anywhere in a standard VMEbus backplane. Optionally, row B of the P2 connector can be re-assigned via links on the ACM-1 to incorporate the remaining modem control lines. This requires a dedicated backplane position.

The VMEbus connector P2 assignments for a standard backplane and a modified backplane are shown in Chapter 3, which also includes a description of the user-defined pin functions. The ACM-1 links for reassigning the P2 connector are detailed in Chapter 4.

The ACM-1 recognises the following address modifier (AM0 to AM5) codes which pass additional binary information to the ACM-1 board during data transfers.

AM Code	Address Modifier						Function
	5	4	3	2	1	0	
3E	H	H	H	H	H	L	Standard Supervisory Program
3D	H	H	H	H	L	H	Standard Supervisory Data
3A	H	H	H	L	H	L	Standard Non-privileged Program
39	H	H	H	L	L	H	Standard Non-privileged Data
0E	L	L	H	H	H	L	Extended Supervisory Program
0D	L	L	H	H	L	H	Extended Supervisory Data
0A	L	L	H	L	H	L	Extended Non-privileged Program
09	L	L	H	L	L	H	Extended Non-privileged Data

Blank page

Chapter 2
Specifications

Common Specifications

This section details the specifications common to all versions of the PMV 68 ACM-1.

General

VMEbus Compliance	ANSI/IEEE Std 1014 - 1987 & IEEE Std 1101.2-1992 A24, D16 A32, D16
Processor	80C186 16-bit
Memory Devices	SRAM, EPROM.
On-board Peripherals	82C59A (equivalent) Programmable Interrupt Controller (on-board the 80C186) 88C681 Dual Universal Asynchronous Receiver/Transmitter (DUART)
Front Panel	Single-width VME containing BITE status indicators (see Figure 5 for front panel layout)

Electrical

Two 96-way DIN41612 compatible connectors that mate with rack mounted backplane connectors form the electrical connection to the board.

Current Consumption

Typical	5V @ 1.41A	+12V @ 0.07A
Maximum	5V @ 2.04A	-12V @ 0.08A

Voltage Supply Requirements

+5V ± 0.25V DC total excursion, including all transients.
V ripple (5V) = 50mV RMS (max) contained within the total excursion.

+12V ± 0.6V DC total excursion, including all transients
V ripple (+12V) = 120mV RMS (max) contained within the total excursion

Safety Requirements

Do not exceed the maximum rated input voltages or apply reversed bias to the assembly. If such conditions occur, toxic fumes may be produced due to the destruction of components.

When exposing boards to a UV light source to erase EPROMS, slight bleaching of black anodised items may occur. This is not detrimental in any way but can be prevented by using an erasure shield.

The ACM-1 uses power from both the P1 and P2 VMEbus connectors. Use of only the P1 connector will cause excessive current to be drawn through the +5V connector pins.



Mechanical

Weight

The approximate weights of the different styles of ACM-1 are as follows:

MI/RI styles	=	700g (25 oz)
MA/R styles	=	620g (22 oz)
E style	=	500g (18 oz)

Dimensions

Figures 3 and 4 show basic VME board dimensions in millimetres with (inches) for guidance only. All build standards of the ACM-1 board are IEEE Std 1014-1987 VMEbus Specification compatible. This allows all styles of assembly to be fitted in any commercial VMEbus development chassis. MI and RI-style cards comply with IEEE Std 1101.2-1992.

Figure 3 - VME Dimensions for MA, R and E Styles

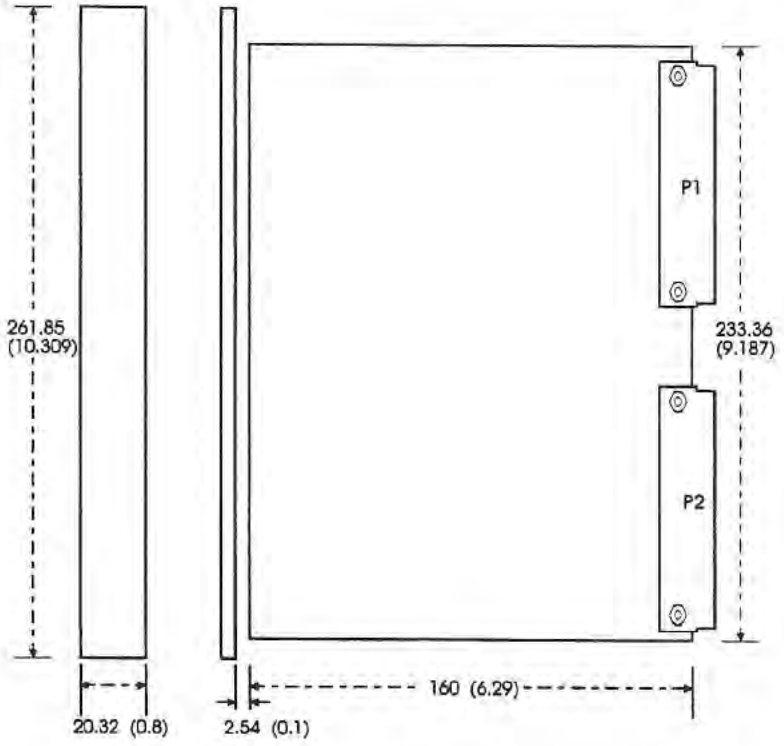
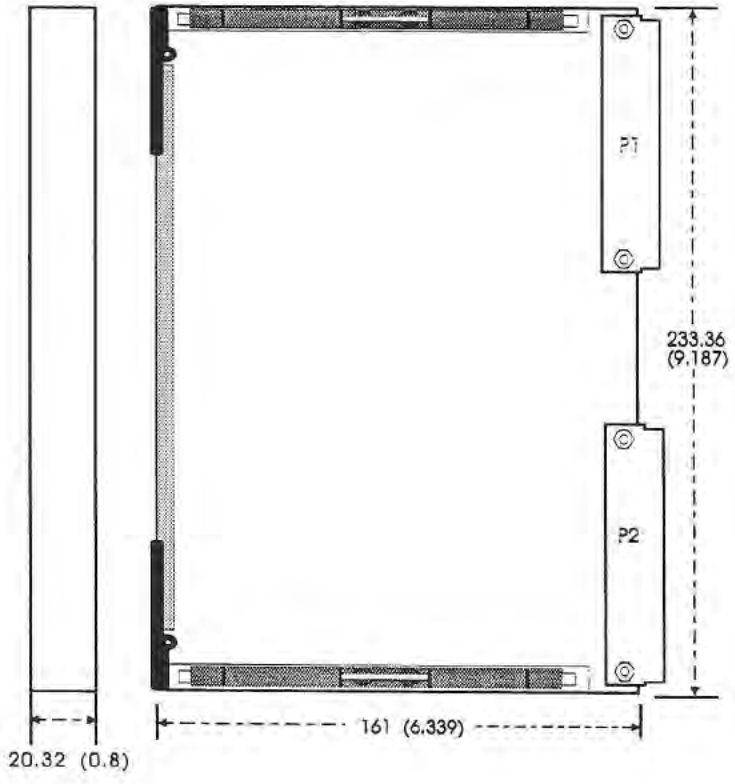


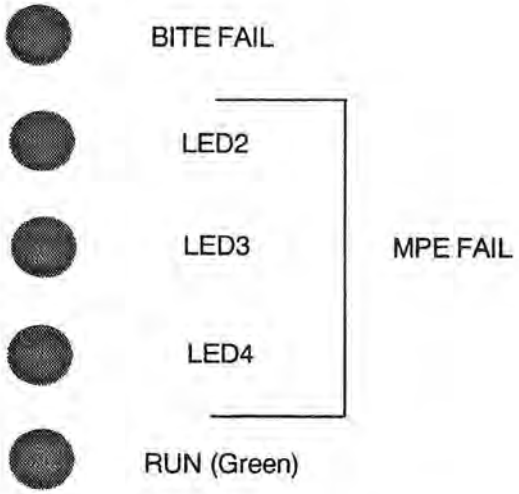
Figure 4 - VME Dimensions For MI and RI Styles



Front Panel

On the front edge of the board is the front panel, which contains five LEDs as shown in the following diagram. The function of the LEDs is described in Chapter 6.

Figure 5 - ACM-1 Front Panel LED Positions



PMV 68MI Specification

Mechanical

Each MI-style assembly has an HML bonded to a multilayer printed wiring board. ICs are mounted astride rungs of the HML, which conducts heat to the edges of the assembly. The unit may be secured in a rack by two screw driven wedgelocks. These clamp the HML to the rack 'cold wall' at the thermal interface on the two short edges of the assembly.

Environmental

In common with other PMV 68MI modules the MI-style ACM-1 can withstand the following conditions:

MIL-STD-810E Test	Method	Procedure	Comments
Low Pressure (altitude)	500.3	I, II & III	Storage and operational tests to 50,000 ft. Also rapid decompression to maximum altitude.
High Temperature	501.3	I & II	Max. Storage 105°C #Max. Operating 85°C
Low Temperature	502.3	I & II	Min. Storage -62°C #Min. Operating -55°C
Temperature Shock	503.3	I	Rapid change over range of operational and storage temperatures
Humidity	507.3	III	Up to 95%RH with varying temperature, 10 cycles 240hrs minimum
Salt Fog	509.3	I	Continuous exposure to mist with 5% salt solution for 48hrs
Acceleration	513.4	I & II	Structural and operational tests. Acceleration levels up to 13.5g
Vibration	514.4	I	Category 5 equipment. Vibration spectrum Fig 514.4-8. 15 to 2,000 Hz to 0.1g ^{2}/Hz (RMS level = 12g approx.)
Shock	516.4	I	Test pulse giving the spectrum of Fig. 516.4-1 approximating to a perfect sawtooth of Fig. 516.4-4 at the level of 40g 11ms.
		VI	Bench handling

Measured at the thermal interface to the module

Environmental Stress Screening

The MI-style ACM-1 is subjected to a Reliability Burn-in, in accordance with Radstone Specification VV68102899 ESS Requirements for PMV 68 Military CCAs.

PMV 68MA Specification

MA-style boards are built to the same design and using the same military qualified components as M style ones, but lack the heat management layer required for conduction cooling. Forced air convection cools the boards.

MA-style ACM-1 can withstand the following conditions (highlights show differences from the MI-style):

Environment	Specification	Method	Procedure	Comments
Low Pressure (altitude)	MIL-STD-810E	500.3	I, II & III	Storage and operational tests to 50,000 ft. Also rapid decompression to maximum altitude
High Temperature	MIL-STD-810E	501.3	I & II	Maximum Storage 105°C **Maximum Operating 85°C with air flow of 600 ft/min Maximum Operating 70°C with air flow of 300 ft/min**
Low Temperature	MIL-STD-810E	502.3	I & II	Minimum Storage -62°C Minimum Operating -55°C
Temperature Shock	MIL-STD-810E	503.3	I	Rapid change over range of operational and storage temperatures
Humidity	MIL-STD-810E	507.3	III	Up to 95%RH with varying temperature, 10 cycles 240hrs minimum
Salt Fog	MIL-STD-810E	509.3	I	Continuous exposure to mist with 5% salt solution for 48hrs
Vibration	**MIL-STD-810E**	**514.4**	**I**	**Fig 514.4-17 swept sine 5g 5-500Hz**
	MIL-STD-202F	**201A**	**I**	**Constant amplitude swept sine 0.06 inch double amplitude 10-55Hz**
Shock	**MIL-STD-202F**	**213**	**J**	**30g 11msec half sine**
	MIL-STD-810E	**516.4**	**V**	**Bench handling**

Acceleration figures are not applicable for MA-style ACM-1.

Environmental Stress Screening

The MA-style ACM-1 is subjected to a Reliability Burn-in, in accordance with Radstone Specification VV68102899 ESS Requirements for PMV 68 Military CCAs.

PMV 68RI Specifications

RI-style boards are mechanically identical to their MI-style equivalents and as such comply with the requirements of IEEE Std 1101.2-1992.

RI-style boards can survive the same environment as the MI-style boards with the following operating and storage temperatures:

Maximum storage temperature	100°C
Maximum operating temperature	55°C
Minimum storage temperature	-50°C
Minimum operating temperature	-40°C

PMV 68R Specification

R-style boards have the same mechanical build standard as MA-style boards. They can withstand the same environment when conformally coated (an option that must be specified separately), apart from the storage and operational temperatures. Wide temperature range, industrial grade components replace full military qualified components.

The differences from the MA specification are as follows:

High Temperature

Maximum Storage Temperature	100°C
Maximum Operating Temperature	85°C with an airflow of 600 ft/min.
	70°C with an airflow of 300 ft/min.

Low temperature

Minimum Storage Temperature	-50°C
Minimum Operating Temperature	-40°C

PMV 68E Specification

The E-style boards use commercial grade, plastic packaged components and can withstand the following conditions:

Altitude

-1000 to +50,000 feet

High Temperature

Maximum Storage Temperature	100°C
Maximum Operating Temperature	70°C with an airflow of 600ft/min
	55°C with an airflow of 300ft/min

Low temperature

Minimum Storage Temperature	-50°C
Minimum Operating Temperature	0°C

Temperature shock

±10°C/min

Humidity

95% non-condensing

Vibration

Swept sine 5g 5 to 500 Hz

Mechanical Shock

30g for 11 milliseconds (half-sine)

Reliability

Using MIL-HBK-217F as a data base and the parts count method, the following estimates have been made of the Mean Time Between Failures (MTBF) for the ACM-1.

Environment	Ground Benign	Ground Fixed	Ground Mobile	Naval Sheltered	Airborne Inhabited Cargo
Temperature	30°C	40°C	45°C	40°C	55°C
MI/MA-styles					
Failure Rate (Failures/million hours)	1.9475222	4.51118	8.5628	7.78026	8.91062
MTBF (Hours)	513,473	221,671	116,784	128,530	112,226
RI/R-style					
Failure Rate (Failures/million hours)	4.200642	11.69721	22.9619	20.43062	22.29852
MTBF (Hours)	238,059	85,490	43,550	48,946	44,846
E-style					
Failure Rate (Failures/million hours)	6.600922	16.81033	37.7467	31.67974	32.8462
MTBF (Hours)	151,494	59,487	26,492	31,566	30,445

Blank page

Chapter 3
VMEbus Connector P2 Assignments

This chapter details the VMEbus connector P2 assignments for both standard backplane and dedicated backplane as used on the ACM-1 board. It also gives a description of the user-defined signals. See the VMEbus Specification ANSI/IEEE Std 1014-1987 for a detailed description of VMEbus.

Connector P2 Assignment for a Standard Backplane

This provides 8 channels of data and 4 channels of RS423 modem control.

Note: The asterisk (*) indicates signals that are **active low**.

Pin	Row A Signal	Row B Signal	Row C Signal
1	TXDA	+5V	TXDB
2	TXDA*	GND	TXDB*
3	RXDA*	RESERVED	RXDB*
4	RXDA	A24	RXDB
5	SIGGNDA	A25	SIGGNDB
6	TXDC	A26	TXDD
7	TXDC*	A27	TXDD*
8	RXDC*	A28	RXDD*
9	RXDC	A29	RXDD
10	SIGGNDC	A30	SIGGNDD
11	TXDE	A31	TXDF
12	TXDE*	GND	TXDF*
13	RXDE*	+5V	RXDF*
14	RXDE	D16	RXDF
15	SIGGNDE	D17	SIGGNDF
16	RTSE	D18	RTSF
17	CTSE	D19	CTSF
18	DTRE	D20	DTRF
19	DCDE	D21	DCDF
20	DSRE	D22	DSRF
21	TXDG	D23	TXDH
22	TXDG*	GND	TXDH*
23	RXDG*	D24	RXDH*
24	RXDG	D25	RXDH
25	SIGGNDG	D26	SIGGNDH
26	RTSG	D27	RTSH
27	CTSG	D28	CTSH
28	DTRG	D29	DTRH
29	DCDG	D30	DCDH
30	DSRG	D31	DSRH
31	LOC<1>	GND	LOC<0>
32	LOC<2>	+5V	SPARE

Channel G modem control in RS422 mode

Pin	Row A Signal	Row B Signal	Row C Signal
1	TXDA	+5V	TXDB
2	TXDA*	GND	TXDB*
3	RXDA*	RESERVED	RXDB*
4	RXDA	A24	RXDB
5	SIGGNDA	A25	SIGGNDB
6	TXDC	A26	TXDD
7	TXDC*	A27	TXDD*
8	RXDC*	A28	RXDD*
9	RXDC	A29	RXDD
10	SIGGNDC	A30	SIGGNDD
11	TXDE	A31	TXDF
12	TXDE*	GND	TXDF*
13	RXDE*	+5V	RXDF*
14	RXDE	D16	RXDF
15	SIGGNDE	D17	SIGGNDF
16	RTSE	D18	RTSF
17	CTSE	D19	CTSF
18	DTRE	D20	DTRF
19	DCDE	D21	DCDF
20	DSRE	D22	DSRF
21	TXDG	D23	TXDH
22	TXDG*	GND	TXDH*
23	RXDG*	D24	RXDH*
24	RXDG	D25	RXDH
25	SIGGNDG	D26	SIGGNDH
26	RTSG	D27	RTSG*
27	CTSG	D28	CTSG*
28	DTRG	D29	DTRG*
29	DCDG	D30	DCDG*
30	DSRG	D31	DSRG*
31	LOC<1>	GND	LOC<0>
32	LOC<2>	+5V	SPARE

Connector P2 Assignment for a Dedicated Backplane

All 8 modem control signals in RS423 mode.

Pin	Row A Signal	Row B Signal	Row C Signal
1	TXDA	+5V	TXDB
2	TXDA*	GND	TXDB*
3	RXDA*	RESERVED	RXDB*
4	RXDA	RTSA	RXDB
5	SIGGNDA	CTSA	SIGGNDB
6	TXDC	DTRA	TXDD
7	TXDC*	DCDA	TXDD*
8	RXDC*	DSRA	RXDD*
9	RXDC	RTSB	RXDD
10	SIGGNDC	CTSB	SIGGNDD
11	TXDE	DTRB	TXDF
12	TXDE*	GND	TXDF*
13	RXDE*	+5V	RXDF*
14	RXDE	DCDB	RXDF
15	SIGGNDE	DSRB	SIGGNDF
16	RTSE	RTSC	RTSF
17	CTSE	CTSC	CTSF
18	DTRE	DTRC	DTRF
19	DCDE	DCDC	DCDF
20	DSRE	DSRC	DSRF
21	TXDG	RTSD	TXDH
22	TXDG*	GND	TXDH*
23	RXDG*	CTSD	RXDH*
24	RXDG	DTRD	RXDH
25	SIGGNDG	DCDD	SIGGNDH
26	RTSG	DSRD	RTSH
27	CTSG	SPARE	CTSH
28	DTRG	SPARE	DTRH
29	DCDG	SPARE	DCDH
30	DSRG	SPARE	DSRH
31	LOC<1>	GND	LOC<0>
32	LOC<2>	+5V	SPARE

7 channels of modem control signal in RS423 mode and Channel 'G' in RS422 mode.

Pin	Row A Signal	Row B Signal	Row C Signal
1	TXDA	+5V	TXDB
2	TXDA*	GND	TXDB*
3	RXDA*	RESERVED	RXDB*
4	RXDA	RTSA	RXDB
5	SIGGNDA	CTSA	SIGGNDB
6	TXDC	DTRA	TXDD
7	TXDC	DCDA	TXDD*
8	RXDC*	DSRA	RXDD*
9	RXDC	RTSB	RXDD
10	SIGGNDC	CTSB	SIGGNDD
11	TXDE	DTRB	TXDF
12	TXDE*	GND	TXDF*
13	RXDE*	+5V	RXDF*
14	RXDE	DCDB	RXDF
15	SIGGNDE	DSRB	SIGGNDF
16	RTSE	RTSC	RTSF
17	CTSE	CTSC	CTSF
18	DTRE	DTRC	DTRF
19	DCDE	DCDC	DCDF
20	DSRE	DSRC	DSRF
21	TXDG	RTSD	TXDH
22	TXDG*	GND	TXDH*
23	RXDG*	CTSD	RXDH*
24	RXDG	DTRD	RXDH
25	SIGGNDG	DCDD	SIGGNDH
26	RTSG	DSRD	RTSG*
27	CTSG	RTSH	CTSG*
28	DTRG	CTSH	DTRG*
29	DCDG	DTRH	DCDG*
30	DSRG	DCDH	DSRG*
31	LOC<1>	GND	LOC<0>
32	LOC<2>	+5V	DSRH

Connector P2 User-Defined Pin Functions

ACM-1 Board Address Select Links

LOC<0>, LOC<1> and LOC<2> (pins C31, A31 and A32 respectively) are the backplane address select pins.

Channel A

1) **TXDA and TXDA* - Pins A1 and A2.** In RS422 mode, TXDA and TXDA* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDA is used for the single ended transmission.

2) **RXDA* and RXDA - Pins A3 and A4.** In RS422 mode, RXDA* and RXDA are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDA is used for the single ended reception.

3) **SIGGNDA - Pin A5.** SIGGNDA is the common reference ground for channel A.

4) **DTRA - Pin B6.** The modem control signal DTRA is Data Terminal Ready for channel A and is a single ended RS423 driver output.

5) **RTSA - Pin B4.** The modem control signal RTSA is Request To Send for channel A and is a single ended RS423 driver output.

6) **DSRA - Pin B8.** The modem control signal DSRA is Data Set Ready for channel A and is a single ended RS423 receiver input.

7) **DCDA - Pin B7.** The modem control signal DCDA is Data Carrier Detect for channel A and is a single ended RS423 receiver input.

8) **CTSA - Pin B5.** The modem control signal CTSA is Clear To Send for channel A and is a single ended RS423 receiver input.

Channel B

1) **TXDB and TXDB* - Pins C1 and C2.** In RS422 mode, TXDB and TXDB* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDB is used for the single ended transmission.

2) **RXDB* and RXDB - Pins C3 and C4.** In RS422 mode, RXDB* and RXDB are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDB is used for the single ended reception.

3) **SIGGNDB - Pin C5.** SIGGNDB is the common reference ground for channel B

4) **DTRB - Pin B11.** The modem control signal DTRB is Data Terminal Ready for channel B and is a single ended RS423 driver output.

5) **RTSB - Pin B9.** The modem control signal RTSB is Request To Send for channel B and is a single ended RS423 driver output.

6) **DSRB - Pin B15.** The modem control signal DSRB is Data Set Ready for channel B and is a single ended RS423 receiver input.

7) **DCDB - Pin B14.** The modem control signal DCDB is Data Carrier Detect for channel B and is a single ended RS423 receiver input.

8) **CTSB - Pin B10.** The modem control signal CTSB is Clear To Send for channel B and is a single ended RS423 receiver input.

Channel C

1) **TXDC and TXDC* - Pins A6 and A7.** In RS422 mode, TXDC and TXDC* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDC is used for the single ended transmission.

2) **RXDC* and RXDC - Pins A8 and A9.** In RS422 mode, RXDC* and RXDC are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDC is used for the single ended reception.

3) **SIGGNDC - Pin A10.** SIGGNDC is the common reference ground for channel C

4) **DTRC - Pin B18.** The modem control signal DTRC is Data Terminal Ready for channel C and is a single ended RS423 driver output.

5) **RTSC - Pin B16.** The modem control signal RTSC is Request To Send for channel C and is a single ended RS423 driver output.

6) **DSRC - Pin B20.** The modem control signal DSRC is Data Set Ready for channel C and is a single ended RS423 receiver input.

7) **DCDC - Pin B19.** The modem control signal DCDC is Data Carrier Detect for channel C and is a single ended RS423 receiver input.

8) **CTSC - Pin B17.** The modem control signal CTSC is Clear To Send for channel C and is a single ended RS423 receiver input.

Channel D

1) **TXDD and TXDD* - Pins C6 and C7.** In RS422 mode, TXDD and TXDD* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDD is used for the single ended transmission.

2) **RXDD* and RXDD - Pins C8 and C9.** In RS422 mode, RXDD* and RXDD are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDD is used for the single ended reception.

3) **SIGGNDD - Pin C10.** SIGGNDD is the common reference ground for channel D.

4) **DTRD - Pin B24.** The modem control signal DTRD is Data Terminal Ready for channel D and is a single ended RS423 driver output.

5) **RTSD - Pin B21.** The modem control signal RTSD is Request To Send for channel D and is a single ended RS423 driver output.

6) **DSRD - Pin B26.** The modem control signal DSRD is Data Set Ready for channel D and is a single ended RS423 receiver input.

7) **DCDD - Pin B25.** The modem control signal DCDD is Data Carrier Detect for channel D and is a single ended RS423 receiver input.

8) **CTSD - Pin B23.** The modem control signal CTSD is Clear To Send for channel D and is a single ended RS423 receiver input.

Channel E

1) **TXDE and TXDE* - Pins A11 and A12.** In RS422 mode, TXDE and TXDE* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDE is used for the single ended transmission.

2) **RXDE* and RXDE - Pins A13 and A14.** In RS422 mode, RXDE* and RXDE are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDE is used for the single ended reception.

3) **SIGGNDE - Pin A15.** SIGGNDE is the common reference ground for channel E

4) **DTRE - Pin A18.** The modem control signal DTRE is Data Terminal Ready for channel E and is a single ended RS423 driver output.

5) **RTSE - Pin A16.** The modem control signal RTSE is Request To Send for channel E and is a single ended RS423 driver output.

6) **DSRE - Pin A20.** The modem control signal DSRE is Data Set Ready for channel E and is a single ended RS423 receiver input.

7) **DCDE - Pin A19.** The modem control signal DCDE is Data Carrier Detect for channel E and is a single ended RS423 receiver input.

8) **CTSE - Pin A17.** The modem control signal CTSE is Clear To Send for channel E and is a single ended RS423 receiver input.

Channel F

1) **TXDF and TXDF* - Pins C11 and C12.** In RS422 mode, TXDF and TXDF* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDF is used for the single ended transmission.

2) **RXDF* and RXDF - Pins C13 and C14.** In RS422 mode, RXDF* and RXDF are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDF is used for the single ended reception.

3) **SIGGNDF - Pin C15.** SIGGNDF is the common reference ground for channel F

4) **DTRF - Pin C18.** The modem control signal DTRF is Data Terminal Ready for channel F and is a single ended RS423 driver output.

5) **RTSF - Pin C16.** The modem control signal RTSF is Request To Send for channel F and is a single ended RS423 driver output.

6) **DSRF - Pin C20.** The modem control signal DSRF is Data Set Ready for channel F and is a single ended RS423 receiver input.

7) **DCDF - Pin C19.** The modem control signal DCDF is Data Carrier Detect for channel F and is a single ended RS423 receiver input.

8) **CTSF - Pin C17.** The modem control signal CTSF is Clear To Send for channel F and is a single ended RS423 receiver input.

Channel G (RS423 Mode)

1) **TXDG and TXDG* - Pins A21 and A22.** In RS422 mode, TXDG and TXDG* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDG is used for the single ended transmission.

2) **RXDG* and RXDG - Pins A23 and A24.** In RS422 mode, RXDG* and RXDG are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDG is used for the single ended reception.

3) **SIGGNDG - Pin A25.** SIGGNDG is the common reference ground for channel G

4) **DTRG - Pin A28.** The modem control signal DTRG is Data Terminal Ready for channel G and is a single ended RS423 driver output in this mode.

5) **RTSG - Pin A26.** The modem control signal RTSG is Request To Send for channel G and is a single ended RS423 driver output in this mode.

6) **DSRG - Pin A30.** The modem control signal DSRG is Data Set Ready for channel G and is a single ended RS423 receiver input in this mode.

7) **DCDG - Pin A29.** The modem control signal DCDG is Data Carrier Detect for channel G and is a single ended RS423 receiver input in this mode.

8) **CTSG - Pin A27.** The modem control signal CTSG is Clear To Send for channel G and is a single ended RS423 receiver input in this mode.

Channel G (RS 422 Mode)

1) **TXDG and TXDG* - Pins A21 and A22.** In RS422 mode, TXDG and TXDG* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDG is used for the single ended transmission.

2) **RXDG* and RXDG - Pins A23 and A24.** In RS422 mode, RXDG* and RXDG are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDG is used for the single ended reception.

3) **SIGGNDG - Pin A25.** SIGGNDG is the common reference ground for channel G

4) **DTRG and DTRG* - Pins A28 and C28.** In modem control signal DTRG is Data Terminal Ready for Channel G and is a differential RS422 driver output in this mode.

5) **RTSG and RTSG* - Pin A26 and C26.** The modem control signal RTSG is Request To Send for channel G and is a differential RS422 driver output in this mode.

6) **DSRG and DSRG* - Pin A30 and C30.** The modem control signal DSRG is Data Set Ready for channel G and is a differential RS422 receiver input in this mode.

7) **DCDG and DCDG* - Pin A29 and C29.** The modem control signal DCDG is Data Carrier Detect for channel G and is a differential RS422 receiver input in this mode.

8) **CTSG and CTSG*-Pin A27 and C27.** The modem control signal CTSG is Clear To Send for channel G and is a differential RS422 receiver input in this mode.

Channel H (When Channel G is in RS423 Mode)

1) **TXDH and TXDH* - Pins C21 and C22.** In RS422 mode, TXDH and TXDH* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDH is used for the single ended transmission.

2) **RXDH* and RXDH - Pins C23 and C24.** In RS422 mode, RXDH* and RXDH are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDH is used for the single ended reception.

3) **SIGGNDH - Pin C25.** SIGGNDH is the common reference ground for channel H.

4) **DTRH - Pin C28.** The modem control signal DTRH is Data Terminal Ready for channel H and is a single ended RS423 driver output.

5) **RTSH - Pin C26.** The modem control signal RTSH is Request To Send for channel H and is a single ended RS423 driver output.

6) **DSRH - Pin C30.** The modem control signal DSRH is Data Set Ready for channel H and is a single ended RS423 receiver input.

7) **DCDH - Pin C29.** The modem control signal DCDH is Data Carrier Detect for channel H and is a single ended RS423 receiver input.

8) **CTSH - Pin C27.** The modem control signal CTSH is Clear To Send for channel H and is a single ended RS423 receiver input.

Channel H (Alternative pinout when Channel G is in RS422 Mode)

1) **TXDH and TXDH* - Pins C21 and C22.** In RS422 mode, TXDH and TXDH* are the differential transmit data driver outputs for synchronous or asynchronous transmission. In RS423 mode, only TXDH is used for the single ended transmission.

2) **RXDH* and RXDH - Pins C23 and C24.** In RS422 mode, RXDH* and RXDH are the differential received data inputs for synchronous or asynchronous reception. In RS423 mode, only RXDH is used for the single ended reception.

3) **SIGGNDH - Pin C25.** SIGGNDH is the common reference ground for channel H.

4) **DTRH - Pin B29.** The modem control signal DTRH is Data Terminal Ready for channel H and is a single ended RS423 driver output.

5) **RTSH - Pin B27.** The modem control signal RTSH is Request To Send for channel H and is a single ended RS423 driver output.

6) **DSRH - Pin C32.** The modem control signal DSRH is Data Set Ready for channel H and is a single ended RS423 receiver input.

7) **DCDH - Pin B30.** The modem control signal DCDH is Data Carrier Detect for channel H and is a single ended RS423 receiver input.

8) **CTSH - Pin B28.** The modem control signal CTSH is Clear To Send for channel H and is a single ended RS423 receiver input.

Chapter 4 - Board Configuration

This chapter provides the information required to configure versions of ACM-1 with PWB No.**419731xx-xxx** and PWB No.**419732xx-xxx.**

Note: The ACM-1 configuration is dependent upon the version of the printed wiring board used.
Versions of the PWB with No. **419732xx-xxx,** provide additional functionality of one channel which is fully RS422/RS423 link selectable on data and modem control lines, whilst retaining a backwards compatibility with earlier versions bearing PWB No.**419731xx-xxx.**

Refer to page 4-2 for Board Configuration for PWB No.419731xx-xxx.

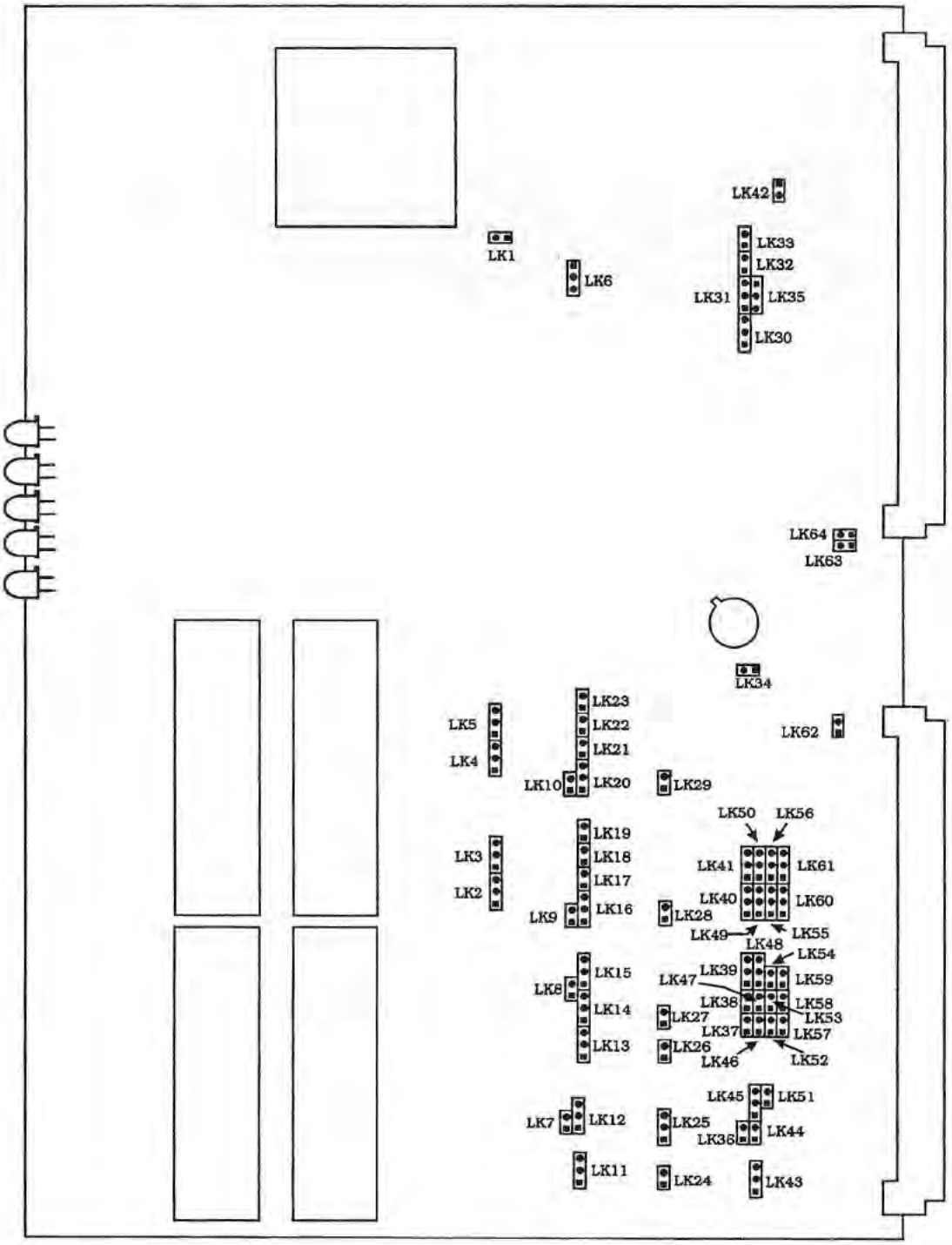
Board Configuration for PWB No.419732xx-xxx begins on page 4-17.

Caution! Conformally coated assemblies - To ensure that CCAs meet required salt/fog specification all exposed pins must be conformally coated after final configuration. Unmated exposed connector pins must be shrouded to prevent salt deposits providing electrical circuits between pins.

Board Configuration for PWB No. 419731xx-xxx

This section details the options for all the functions of the ACM-1 board (printed wiring board number 419731xx-xxx) that can be configured using wire wrap links. The default configuration of this board is shown in the table at the end of this section. Figure 6 shows the location of all links. The square pins show the position of pin 1. The default configuration is set by push-on links, which must be replaced with wire-wraps to meet MI-style specifications.

Figure 6 - PMV 68 ACM-1 Board Layout (PWB No. 419731xx-xxx)



Fixed Configuration

The ACM-1 board must be configured to include LK1 pin 1 to 2 for clock connection. LK42 must be open for the board to operate.

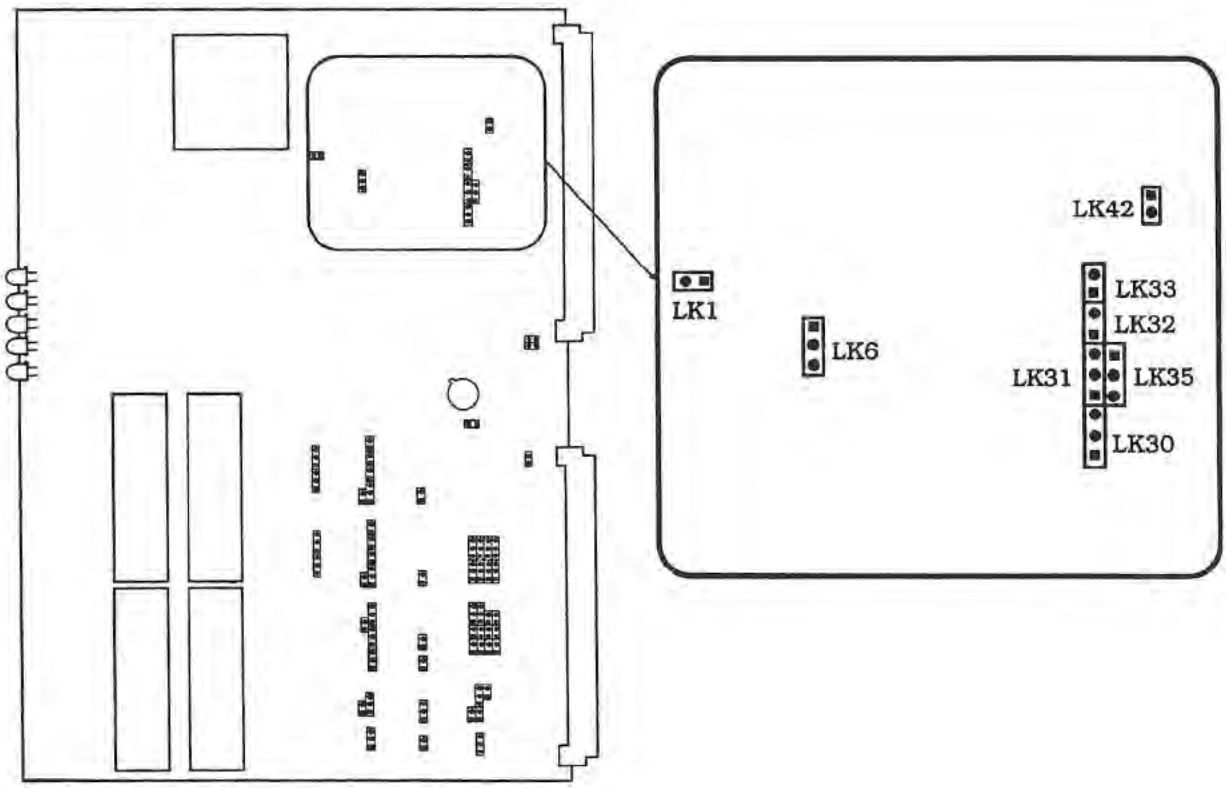
Dual Port RAM Address Pin Options

The dual ported RAM on the ACM-1 board is divided into two byte-wide blocks of memory on two standard JEDEC sites. To allow for future expansion, an address pin link is provided to enable various sizes of SRAM chips to be allocated to the sites.

Memory Capacity (Total in bytes)	LK6		LK35	LK31	LK30
	1-2	2-3	1-2	1-2	1-2
64k	No	Yes	No	No	No
128k	Yes	No	No	No	Yes
256k	Yes	No	No	Yes	Yes
512k	Yes	No	Yes	Yes	Yes

The default configuration is for 64 kbytes total memory capacity.

Note: The links on pins 2-3 of LK35, 31 and 30 are determined by the offset address of the board and covered in the address link tables on pages 4-4 and 4-5.



Address Links Option

Besides the backplane selectable addressing (covered in the Board Address Selection by Backplane Link Options section), which defines the location of the ACM-1 board dual-ported RAM to within one 2 Mbyte block in the total 4096 Mbytes space, there is also a set of links provided that further subdivide the addressing space to a 64/128/256 kbyte block in the default configuration, depending upon the size of memory chips used. This has the dual purpose of enabling more than eight ACM-1 boards to be connected in an A32 or A24 system and limiting the addressable area of the ACM-1 board in an A24 system.

Address Links for 64 kbytes Total Memory Capacity

Offset Address Space	LK33	LK32	LK35	LK31	LK30
	1-2	1-2	2-3	2-3	2-3
0 - 64k **(default)**	Yes	Yes	Yes	Yes	Yes
64k - 128k	Yes	Yes	Yes	Yes	No
128k - 192k	Yes	Yes	Yes	No	Yes
192k - 256k	Yes	Yes	Yes	No	No
256k - 320k	Yes	Yes	No	Yes	Yes
320k - 384k	Yes	Yes	No	Yes	No
384k - 448k	Yes	Yes	No	No	Yes
448k - 512k	Yes	Yes	No	No	No
512k - 576k	Yes	No	Yes	Yes	Yes
576k - 640k	Yes	No	Yes	Yes	No
640k - 704k	Yes	No	Yes	No	Yes
704k - 768k	Yes	No	Yes	No	No
768k - 832k	Yes	No	No	Yes	Yes
832k - 896k	Yes	No	No	Yes	No
896k - 960k	Yes	No	No	No	Yes
960k - 1024k	Yes	No	No	No	No
1024k - 1088k	No	Yes	Yes	Yes	Yes
1088k - 1152k	No	Yes	Yes	Yes	No
1152k - 1216k	No	Yes	Yes	No	Yes
1216k - 1280k	No	Yes	Yes	No	No
1280k - 1344k	No	Yes	No	Yes	Yes
1344k - 1408k	No	Yes	No	Yes	No
1408k - 1472k	No	Yes	No	No	Yes
1472k - 1536k	No	Yes	No	No	No
1536k - 1600k	No	No	Yes	Yes	Yes
1600k - 1664k	No	No	Yes	Yes	No
1664k - 1728k	No	No	Yes	No	Yes
1728k - 1792k	No	No	Yes	No	No
1792k - 1856k	No	No	No	Yes	Yes
1856k - 1920k	No	No	No	Yes	No
1920k - 1984k	No	No	No	No	Yes
1984k - 2048k	No	No	No	No	No

Address Links for 128 kbytes Total Memory Capacity

Offset Address Space	LK33	LK32	LK30	LK31
	1-2	1-2	2-3	2-3
128k - 256k	Yes	Yes	Yes	Yes
256k - 384k	Yes	Yes	Yes	No
384k - 512k	Yes	Yes	No	Yes
512k - 640k	Yes	No	Yes	Yes
640k - 768k	Yes	No	Yes	No
768k - 896k	Yes	No	No	Yes
896k - 1024k	Yes	No	No	No
1024k - 1152k	No	Yes	Yes	Yes
1152k - 1280k	No	Yes	Yes	No
1280k - 1408k	No	Yes	No	Yes
1408k - 1536k	No	Yes	No	No
1536k - 1664k	No	No	Yes	Yes
1664k - 1792k	No	No	Yes	No
1792k - 1920k	No	No	No	Yes
1920k - 2048k	No	No	No	No

Address Links for 256 kbytes Total Memory Capacity

Offset Address Space	LK33	LK32	LK30
	1-2	1-2	2-3
0 - 256k	Yes	Yes	Yes
256k - 512k	Yes	Yes	No
512k - 768k	Yes	No	Yes
768k - 1024k	Yes	No	No
1024k - 1280k	No	Yes	Yes
1280k - 1536k	No	Yes	No
1536k - 1792k	No	No	Yes
1792k - 2048k	No	No	No

Address Links for 512 kbytes Total Memory Capacity

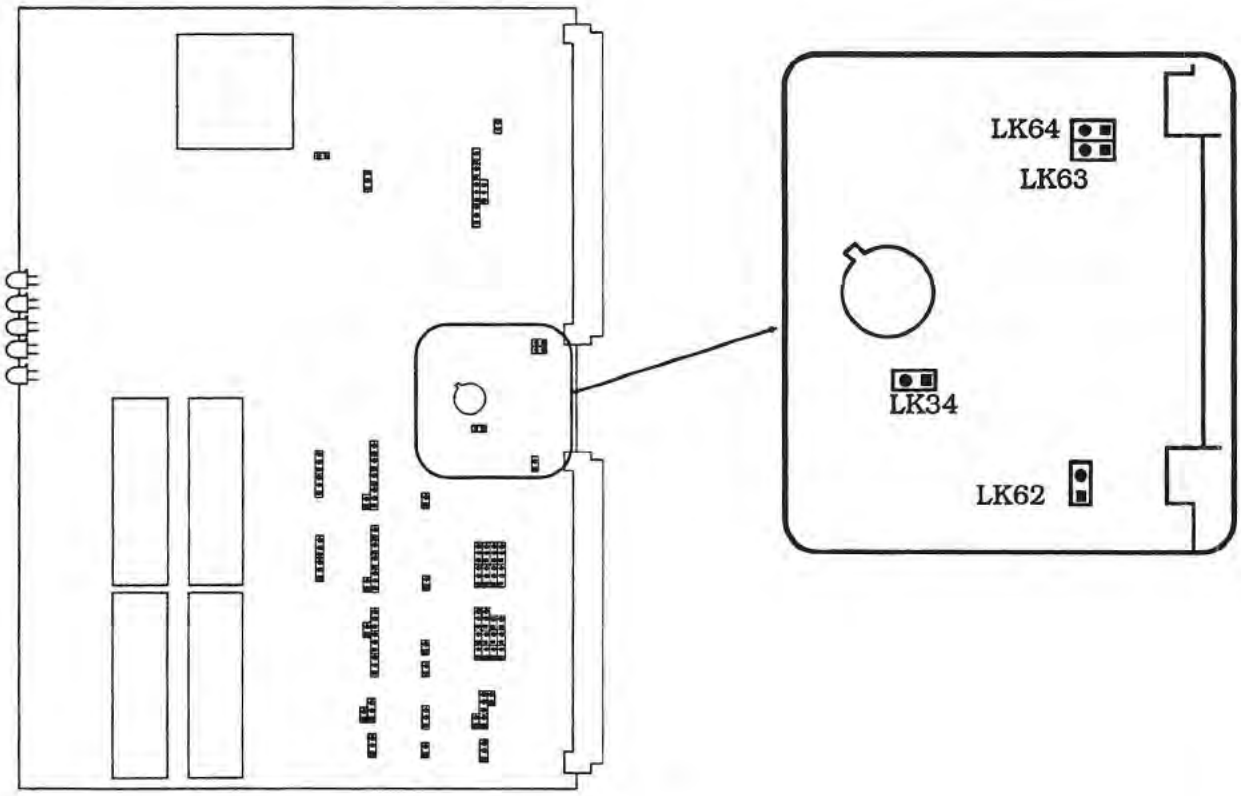
Offset Address Space	LK33	LK32
	1-2	1-2
0 - 512k	Yes	Yes
512k - 1024k	Yes	No
1024k - 1536k	No	Yes
1536k - 2048k	No	No

Dual Ported RAM Access Time Options

The ACM-1 board caters for dual ported RAM modules with different speeds. Depending upon the access time of the module, the links fitted are as follows:

RAM Access Time	LK64	LK63	LK34	LK62
Up to 50 ns	Yes	No	No	No
50 to 70 ns	No	Yes	No	No
70 to 100 ns	No	No	Yes	No
100 to 120 ns	No	No	No	Yes

The default RAM access time is up to 50 ns



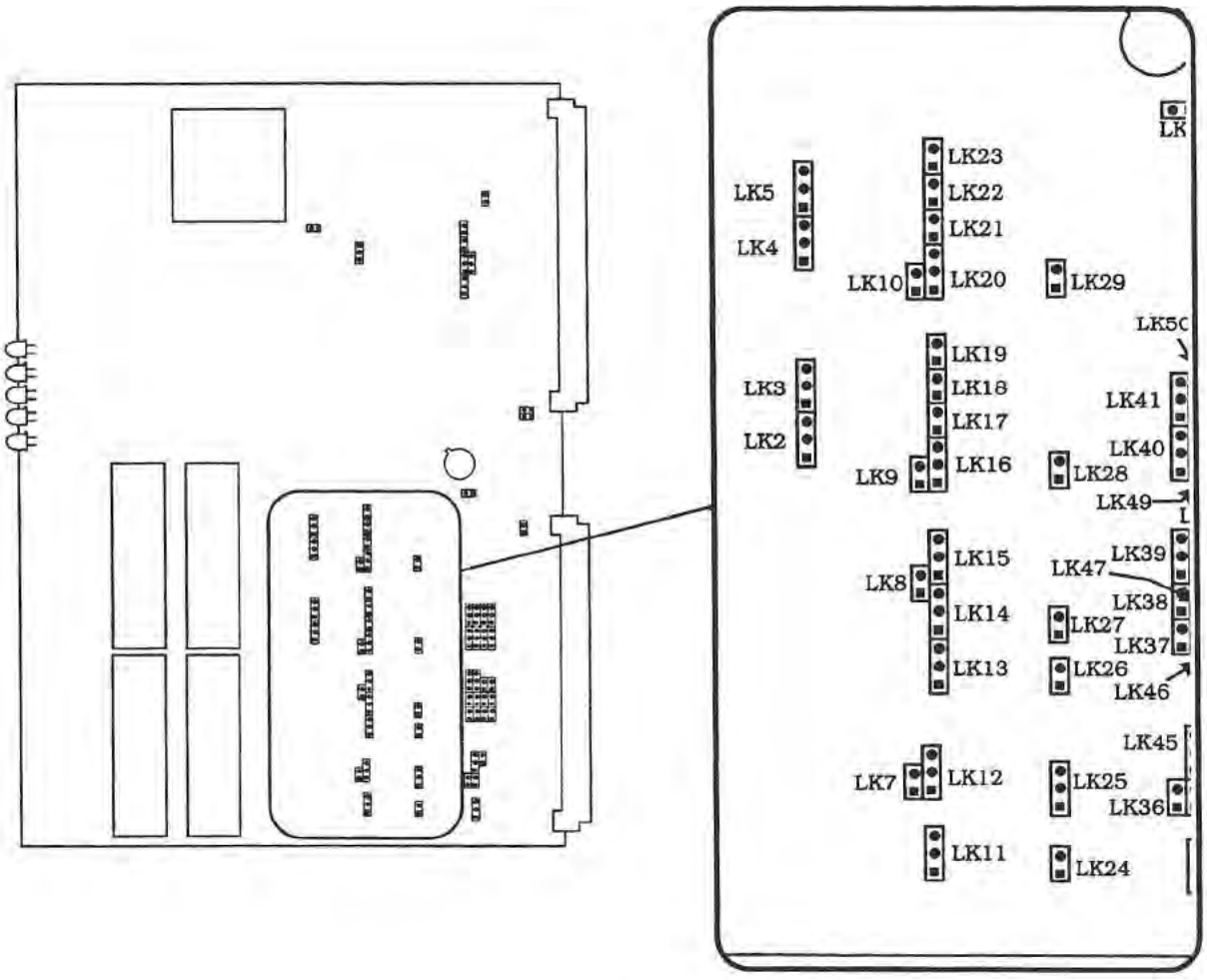
Serial Channels - Transmit Data RS422/RS423 Mode Option

All serial channels' Transmit Data have optional modes of single ended RS423 or differential RS422 operation. To select either RS423 or RS422 for pairs of channels, a group of links are set up as shown below.

Note: These link selections do not affect the selection of any of the other pairs of channels.

Transmit Data	RS423	RS422
Channels A and B (TXDA and TXDB)	Link LK5 pins 1 to 2 Link LK4 pins 1 to 2	Link LK5 pins 2 to 3 Link LK4 pins 2 to 3
Channels C and D (TXDC and TXDD)	Link LK3 pins 1 to 2 Link LK2 pins 1 to 2	Link LK3 pins 2 to 3 Link LK2 pins 2 to 3
Channels E and F (TXDE and TXDF)	Link LK15 pins 1 to 2 Link LK14 pins 1 to 2	Link LK15 pins 2 to 3 Link LK14 pins 2 to 3
Channels G and H (TXDG and TXDH)	Link LK12 pins 1 to 2 Link LK11 pins 1 to 2	Link LK12 pins 2 to 3 Link LK11 pins 2 to 3

The default configuration is for RS423 operation.



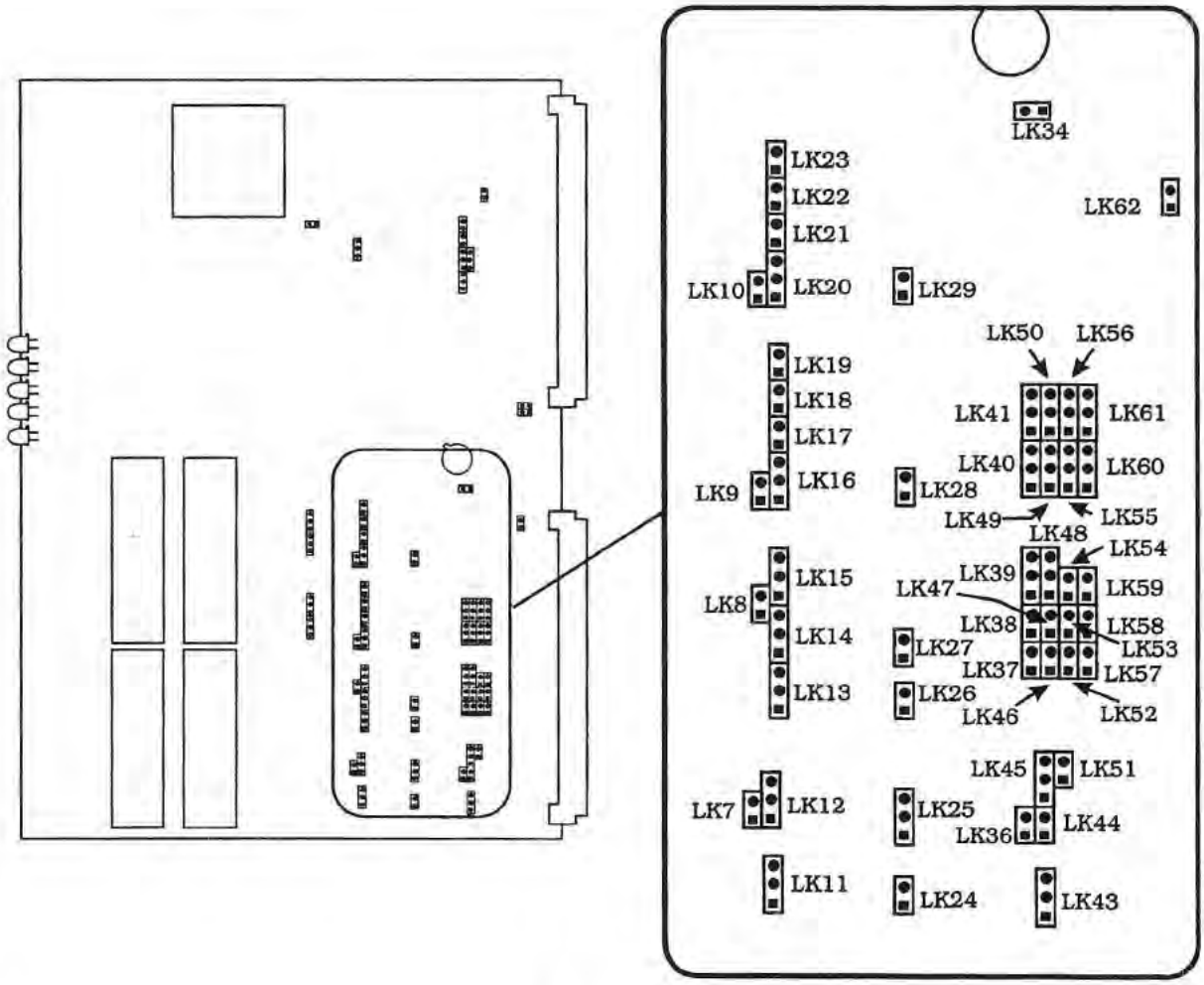
Serial Channels - Receive Data RS422/RS423 Mode Option

All serial channels' Receive Data have optional modes of single ended RS423 or differential RS422 operation. To select either RS423 or RS422 for each channel, a group of links is set up as shown below.

Note: Any channel link selection does not affect the selection of any of the other channels.

Receive Data Channel	RS423	RS422
A	Link LK39 pins 2 to 3	Link LK39 pins 1 to 2
B	Link LK20 pins 2 to 3	Link LK20 pins 1 to 2
C	Link LK48 pins 2 to 3	Link LK48 pins 1 to 2
D	Link LK16 pins 2 to 3	Link LK16 pins 1 to 2
E	Link LK45 pins 2 to 3	Link LK45 pins 1 to 2
F	Link LK13 pins 2 to 3	Link LK13 pins 1 to 2
G	Link LK43 pins 2 to 3	Link LK43 pins 1 to 2
H	Link LK25 pins 2 to 3	Link LK25 pins 1 to 2

The default configuration is for RS423 operation.

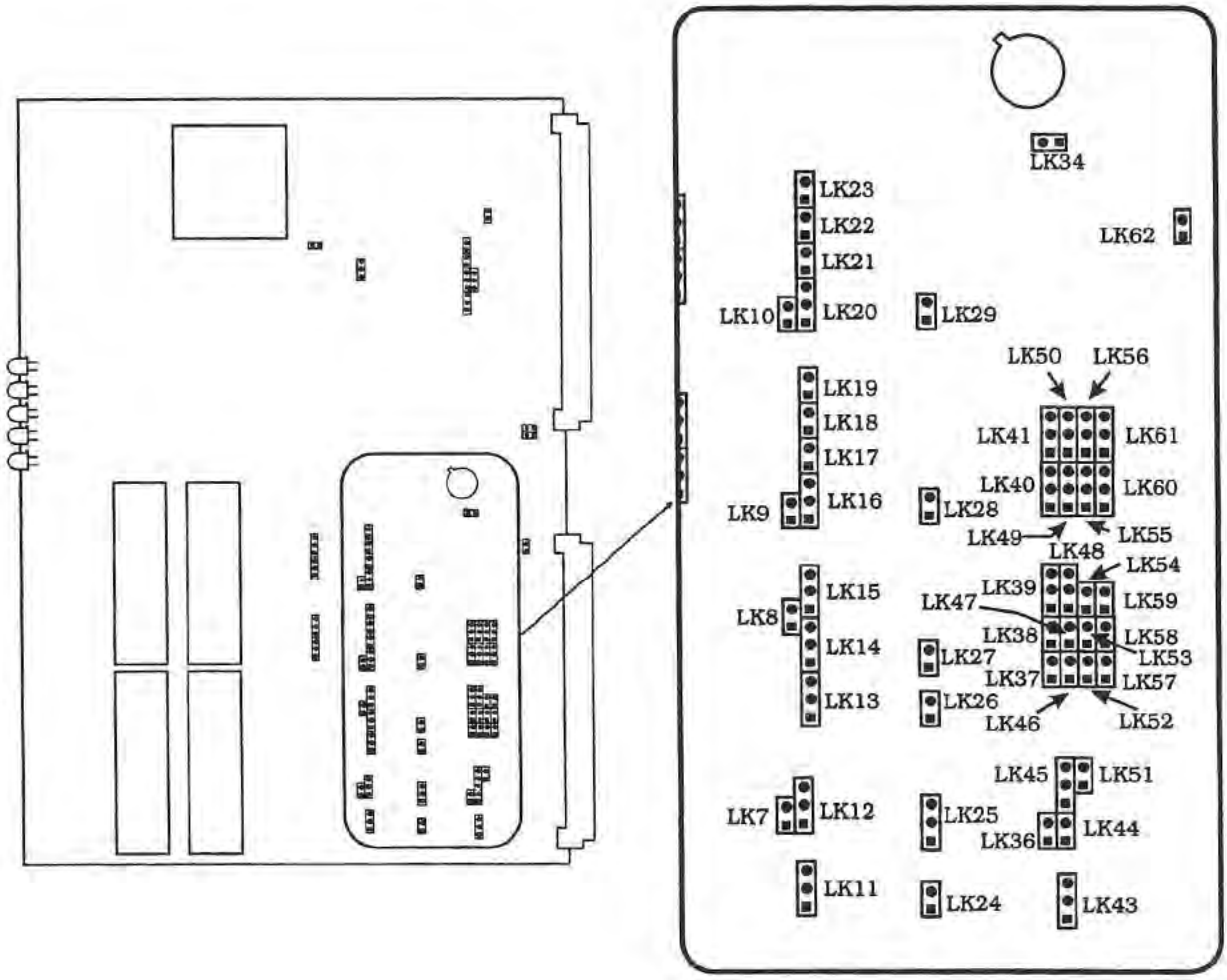


RS422 Terminator Resistors Option

An optional 150Ω resistor is included in each receiver input circuit to terminate the line. Each terminator can be selected by inserting a corresponding link, as follows:

Line Receiver	Link	Terminated	Not Terminated
Received Data (RXDA)	LK38	IN	OUT
Received Data (RXDB)	LK10	IN	OUT
Received Data (RXDC)	LK59	IN	OUT
Received Data (RXDD)	LK9	IN	OUT
Received Data (RXDE)	LK37	IN	OUT
Received Data (RXDF)	LK8	IN	OUT
Received Data (RXDG)	LK44	IN	OUT
Received Data (RXDH)	LK7	IN	OUT

The default configuration is for no terminator.



Signal Ground to Local Ground Reference Option

An option is provided on the ACM-1 board to connect the channel signal ground (SIGGNDA to SIGGNDH) to the local ground. This is necessary for isolated systems, where no reference exists between the systems, to prevent excessive common mode voltages destroying the RS423/RS422 receivers.

Link	Signal	Referenced to Local Ground Link	Isolated from Local Ground Link
LK29	SIGGNDA	IN	OUT
LK22	SIGGNDB	IN	OUT
LK28	SIGGNDC	IN	OUT
LK19	SIGGNDD	IN	OUT
LK46	SIGGNDE	IN	OUT
LK52	SIGGNDF	IN	OUT
LK36	SIGGNDG	IN	OUT
LK24	SIGGNDH	IN	OUT

The default configuration is for SIGGND to be referenced to local ground.

Board Address Selection by Backplane Link Options

The base address of the ACM-1 board is defined by three edge connector pins (a31, a32 and c31) located on the P2 connector. This enables the backplane slots, by using backplane links, to define the address space of the ACM-1 board to one of eight 2 Mbyte boundaries. Further subdivision is possible on board by use of links as defined in the "Address Links Option" section. The backplane links define the board address (LOC<0>, LOC<1> and LOC<2>) as follows:

LOC<0> (P2 pin c31)	LOC<1> (P2 pin a31)	LOC<2> (P2 pin a32)	Board Base Address (A32)	Board Base Address (A24)
GND	GND	GND	$FD000000	$xx000000
OPEN	GND	GND	$FD200000	$xx200000
GND	OPEN	GND	$FD400000	$xx400000
OPEN	OPEN	GND	$FD600000	$xx600000
GND	GND	OPEN	$FD800000	$xx800000
OPEN	GND	OPEN	$FDA00000	$xxA00000
GND	OPEN	OPEN	$FDC00000	$xxC00000
OPEN	OPEN	OPEN	$FDE00000	$xxE00000

Here "OPEN" means no connection and "GND" means connected to 0 Volts.

P2 Row B Link Options

Row B of the P2 connector on the ACM-1 is multifunctional. The link options give the following functions:

1) In an A32 system (Option 1 below), the upper 8 address bits are linked onto the ACM-1. This implies that certain modem control lines for channels A and B are linked out as detailed below.

2) In an A24 system using a standard backplane (Option 2 below), certain serial channels and modem control lines need to be linked out so that no serial channels can drive the bussed P2 Row B address and data lines. In this case no links are fitted. This isolates both the VMEbus high order address and data lines from the P2 connector, and the serial line and modem control lines from the card to the P2 connector.

3) In an A24 system with a dedicated backplane (Option 3 below), all serial channels and modem control lines are available and link options connect the required signals to the P2 connector.

The linking for Row B of the P2 connector is therefore divided into three options with linking for each option as described below. The default is for an A24 system with a standard backplane (Option 2). The VMEbus connector P2 assignments for the options are listed in Chapter 3.

Option 1 - Links Required for an A32 System

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B11	LK60	No	Yes	Address bit A31
B10	LK55	No	Yes	Address bit A30
B9	LK40	No	Yes	Address bit A29
B8	LK49	No	Yes	Address bit A28
B7	LK56	No	Yes	Address bit A27
B6	LK61	No	Yes	Address bit A26
B5	LK50	No	Yes	Address bit A25
B4	LK41	No	Yes	Address bit A24

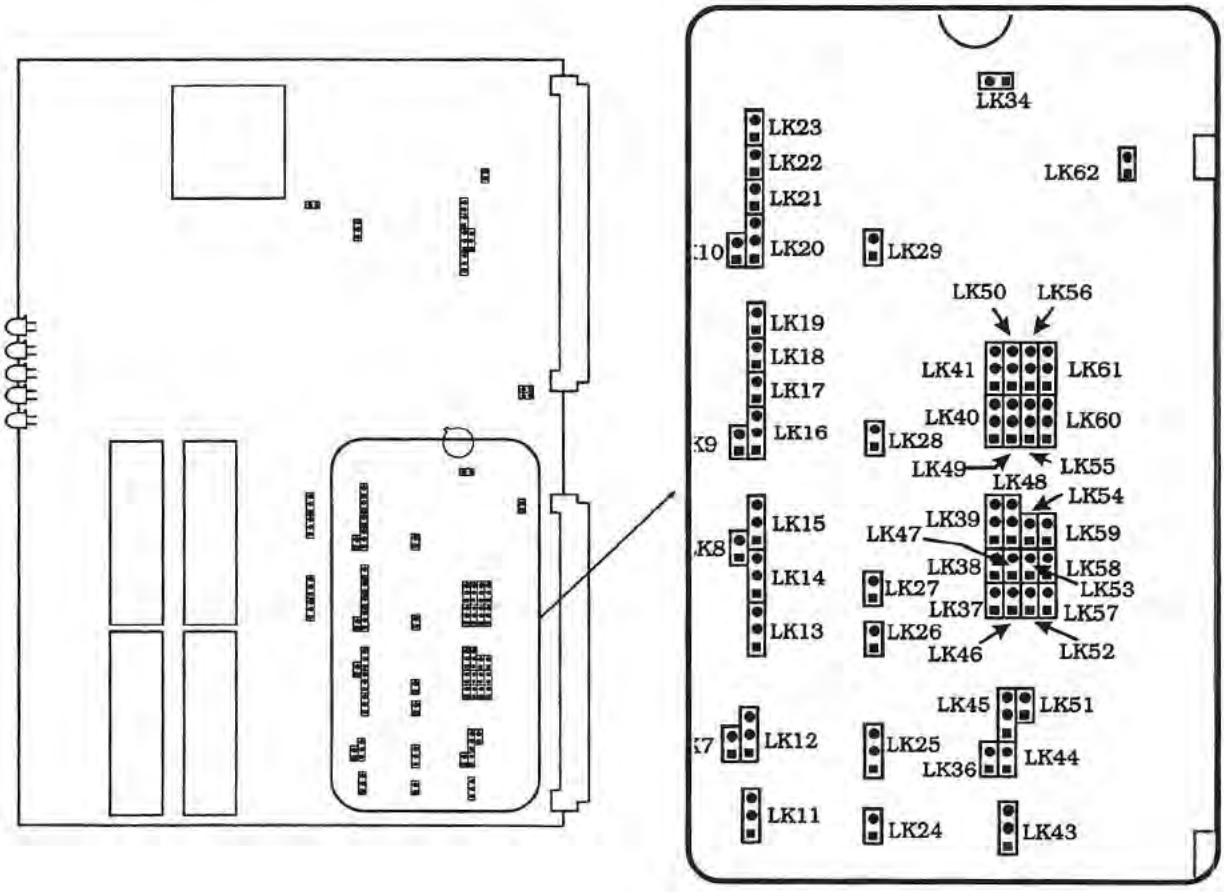
Option 2 - Links Required for an A24 System and Standard Backplane

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B11	LK60	No	No	None
B10	LK55	No	No	None
B9	LK40	No	No	None
B8	LK49	No	No	None
B7	LK56	No	No	None
B6	LK61	No	No	None
B5	LK50	No	No	None
B4	LK41	No	No	None

On both Option 1 and Option 2, the following 2-pin links MUST BE LEFT OPEN otherwise serious contention with VMEbus signals will cause malfunction and possible damage.

P2 Pin Number	Link Number	Function
B18	LK27	Data Bit 20
B24	LK51	Data Bit 25
B16	LK57	Data Bit 18
B21	LK58	Data Bit 23
B23	LK26	Data Bit 24
B25	LK17	Data Bit 26
B26	LK18	Data Bit 27
B14	LK21	Data Bit 16
B15	LK23	Data Bit 17

These two options allow basic "3 wire" control on all 8 channels, and full modem control on 4 channels without having to resort to a modified backplane. This uses a 64-way (rows A and C) connector. These options can be slotted anywhere in a standard VMEbus backplane.

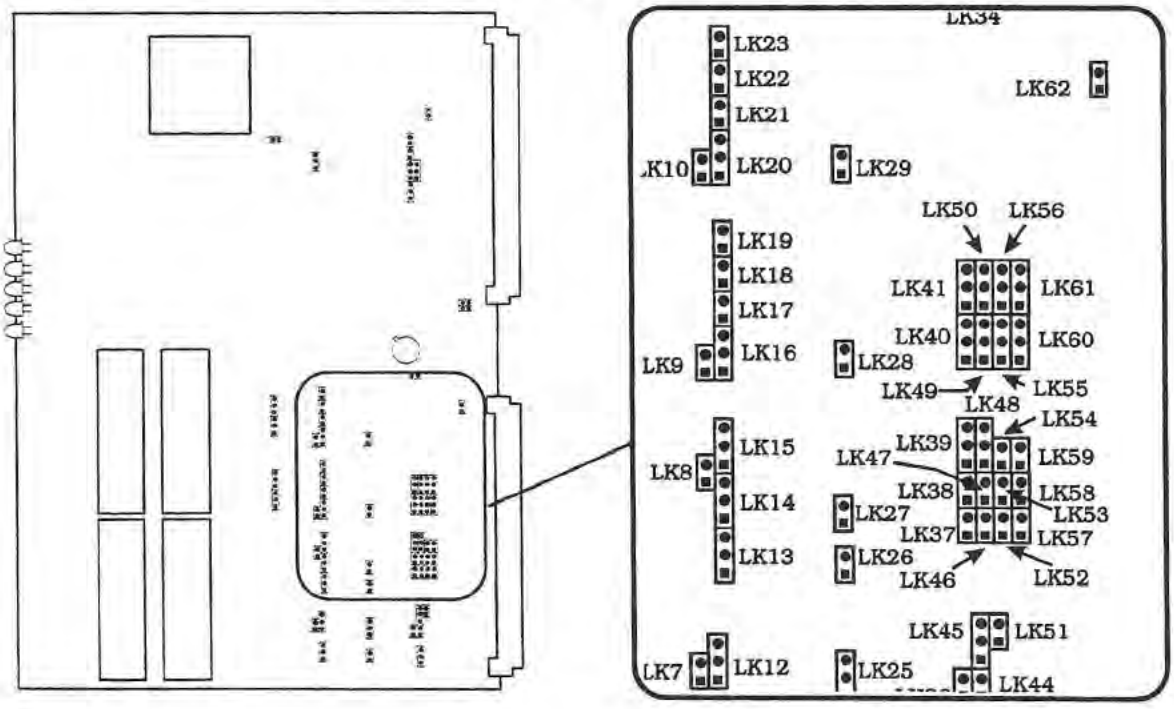


Option 3 - Links Required for an A24 System and Dedicated Backplane

(A backplane with rows A, B and C uncommitted and hard-wired to suit)

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B4	LK41	Yes	No	Request to Send Channel A
B5	LK50	Yes	No	Clear to Send Channel A
B6	LK61	Yes	No	Data Terminal Ready Channel A
B7	LK56	Yes	No	Data Carrier Detect Channel A
B8	LK49	Yes	No	Data Set Ready Channel A
B9	LK40	Yes	No	Request to Send Channel B
B10	LK55	Yes	No	Clear to Send Channel B
B11	LK60	Yes	No	Data Terminal Ready Channel B
B14	LK21	Yes	No	Data Carrier Detect Channel B
B15	LK23	Yes	No	Data Set Ready Channel B
B16	LK57	Yes	No	Request to Send Channel C
B17	LK54	Yes	No	Clear to Send Channel C
B18	LK27	Yes	No	Data Terminal Ready Channel C
B19	LK53	Yes	No	Data Carrier Detect Channel C
B20	LK47	Yes	No	Data Set Ready Channel C
B21	LK58	Yes	No	Request to Send Channel D
B23	LK26	Yes	No	Clear to Send Channel D
B24	LK51	Yes	No	Data Terminal Ready Channel D
B25	LK17	Yes	No	Data Carrier Detect Channel D
B26	LK18	Yes	No	Data Set Ready Channel D

This option allows full use of all 8 channels but requires a dedicated backplane position. The VMEbus connector P2 assignment is detailed in Chapter 3.



Default Settings

Link	Number of Pins	Function	Default Setting	Default Condition
LK1	2	Clock disable/enable	1 - 2	Clock enabled
LK2	3	Mode C/D	1 - 2	RS423
LK3	3	Mode C/D	1 - 2	RS423
LK4	3	Mode A/B	1 - 2	RS423
LK5	3	Mode A/B	1 - 2	RS423
LK6	3	DP RAM Address	2 - 3	RAM size and location 0 - 64 kbytes
LK7	2	RXD H Terminator	Open	Not terminated
LK8	2	RXD F Terminator	Open	Not terminated
LK9	2	RXD D Terminator	Open	Not terminated
LK10	2	RXD B Terminator	Open	Not terminated
LK11	3	Mode G/H	1 - 2	RS423
LK12	3	Mode G/H	1 - 2	RS423
LK13	3	RXD F	2 - 3	RS423
LK14	3	Mode E/F	1 - 2	RS423
LK15	3	Mode E/F	1 - 2	RS423
LK16	3	RXD D	2 - 3	RS423
LK17	2	DB26/DCD D	Open	24-bit address
LK18	2	DB27/DSR D	Open	24-bit address standard backplane
LK19	2	SIGGNDD	1 - 2	Local ground
LK20	3	RXD B	2 - 3	RS423
LK21	2	DB16/DCD B	Open	24-bit address standard backplane
LK22	2	SIGGNDB	1 - 2	Local ground
LK23	2	DB17/DSR B	Open	24-bit address standard backplane
LK24	2	SIGGNDH	1 - 2	Local ground
LK25	3	RXD H	2 - 3	RS423
LK26	2	DB24/CTS D	Open	24-bit address standard backplane
LK27	2	DB20/DTR C	Open	24-bit address standard backplane
LK28	2	SIGGNDC	1 - 2	Local ground
LK29	2	SIGGNDA	1 - 2	Local ground
LK30	3	DP RAM Address	2 - 3	RAM size and location 0 - 64 kbytes
LK31	3	DP RAM address	2 - 3	RAM size and location 0 - 64 kbytes
LK32	2	DP RAM address	1 - 2	RAM size and location 0 - 64 kbytes

Default Settings (*continued*)

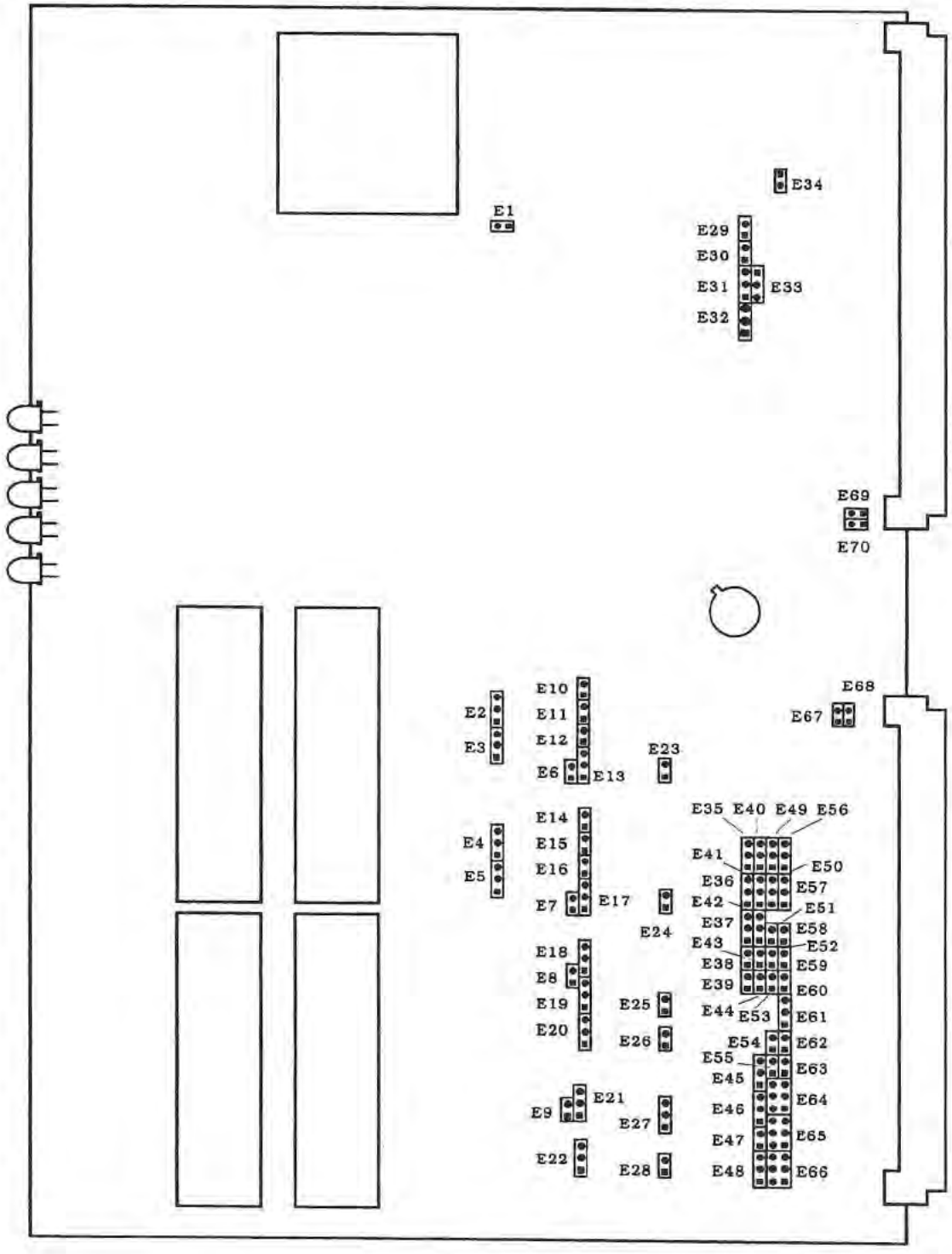
Link	Number of Pins	Function	Default Setting	Default Condition
LK33	2	DP RAM address	1 - 2	RAM size and location 0 - 64 kbytes
LK34	2	DP RAM speed	Open	50 - 70 ns
LK35	3	DP RAM address	2 - 3	RAM size and location 0 - 64 kbytes
LK36	3	SIGGNDG	1 - 2	Local ground
LK37	2	RXD E Terminator	Open	Not terminated
LK38	2	RXD A Terminator	Open	Not terminated
LK39	3	RXD A	2 - 3	RS423
LK40	3	AD29/RTS B	Open	24-bit address standard backplane
LK41	3	AD24/RTS A	Open	24-bit address standard backplane
LK42	2	Reset	Open	Running
LK43	3	RXD G	2 - 3	RS423
LK44	2	RXD G Terminator	Open	Not terminated
LK45	3	RXD E	2 - 3	RS423
LK46	2	SIGGNDE	1 - 2	Local ground
LK47	2	DSR C	Open	24-bit address standard backplane
LK48	3	RXD C	2 - 3	RS423
LK49	3	AD28/DSR A	Open	24-bit address standard backplane
LK50	3	AD25/CTS A	Open	24-bit address standard backplane
LK51	2	DB25/DTR D	Open	24-bit address standard backplane
LK52	2	SIGGNDF	1 - 2	Local ground
LK53	2	DCD C	Open	24-bit address standard backplane
LK54	2	CTS C	Open	24-bit address standard backplane
LK55	3	AD30/CTS B	Open	24-bit address standard backplane
LK56	3	AD27/DCD A	Open	24-bit address standard backplane
LK57	2	DB18/RTS C	Open	24-bit address standard backplane
LK58	2	DB23/RTS D	Open	24-bit address standard backplane
LK59	2	RXD C Terminator	Open	Not terminated

Default Settings (*continued*)				
Link	**Number of Pins**	**Function**	**Default Setting**	**Default Condition**
LK60	3	AD31/DTR B	Open	24-bit address standard backplane
LK61	3	AD26/DTR A	Open	24-bit address standard backplane
LK62	2	DP RAM speed	Open	50 - 70 ns
LK63	2	DP RAM speed	Open	50 - 70 ns
LK64	2	DP RAM speed	1 - 2	50 - 70 ns

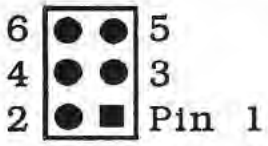
Board Configuration for PWB No. 419732xx-xxx

This section details the options for all the functions of the ACM-1 board (printed wiring board number 419732xx-xxx) that can be configured using wire wrap links. The default configuration of this board is shown in the table at the end of this section. The following diagram shows the location of all links. The square pins show the position of pin 1. The default configuration is set by push-on links, which must be replaced with wire-wraps to meet MI-style specifications.

Figure 7 - PMV 68 ACM-1 Board Layout (PWB No. 419732xx-xxx)



Pin Position of Links E64, E65 and E66



Fixed Configuration

The ACM-1 board must be configured to include link E1 pin 1 to 2 for clock connection. Link E34 must be open for the board to operate.

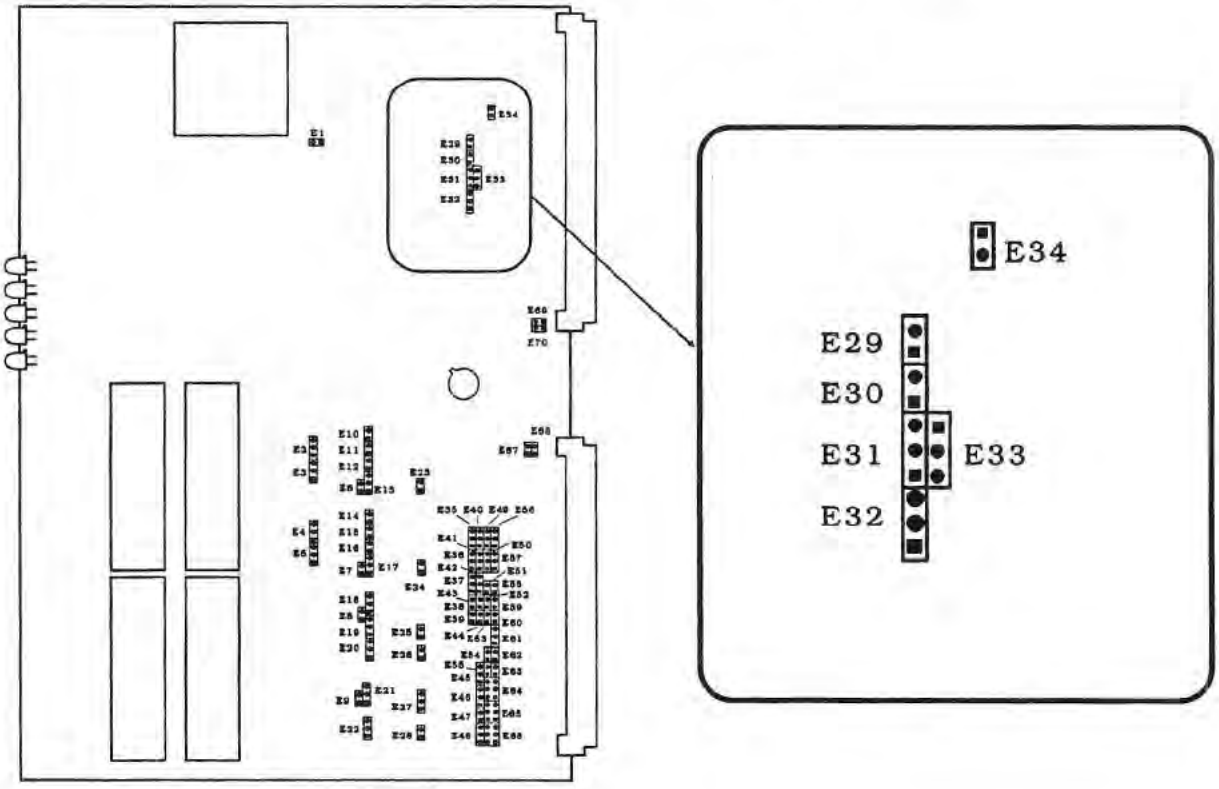
Dual Port RAM Address Pin Options

The dual ported RAM on the ACM-1 board is divided into two byte-wide blocks of memory on two standard JEDEC sites. To allow for future expansion, an address pin link is provided to enable various sizes of SRAM chips to be allocated to the sites.

Memory Capacity (Total in bytes)	E33	E31	E32
	1-2	1-2	1-2
64k	No	No	No
128k	No	No	Yes
256k	No	Yes	Yes
512k	Yes	Yes	Yes

The default configuration is for 64 kbytes total memory capacity.

Note: The links on pins 2-3 of E33, 31 and 32 are determined by the offset address of the board and covered in the address link tables on pages 4-19/20.



Address Links Option

Besides the backplane selectable addressing (covered in the Board Address Selection by Backplane Link Options section), which defines the location of the ACM-1 board dual-ported RAM to within one 2 Mbyte block in the total 4096 Mbytes space, there is also a set of links provided that further subdivide the addressing space to a 64/128/256 kbyte block in the default configuration, depending upon the size of memory chips used. This has the dual purpose of enabling more than eight ACM-1 boards to be connected in an A32 or A24 system and limiting the addressable area of the ACM-1 board in an A24 system.

Address Links for 64 kbytes Total Memory Capacity

Offset Address Space	E29	E30	E33	E31	E32
	1-2	1-2	2-3	2-3	2-3
0 - 64k **(default)**	Yes	Yes	Yes	Yes	Yes
64k - 128k	Yes	Yes	Yes	Yes	No
128k - 192k	Yes	Yes	Yes	No	Yes
192k - 256k	Yes	Yes	Yes	No	No
256k - 320k	Yes	Yes	No	Yes	Yes
320k - 384k	Yes	Yes	No	Yes	No
384k - 448k	Yes	Yes	No	No	Yes
448k - 512k	Yes	Yes	No	No	No
512k - 576k	Yes	No	Yes	Yes	Yes
576k - 640k	Yes	No	Yes	Yes	No
640k - 704k	Yes	No	Yes	No	Yes
704k - 768k	Yes	No	Yes	No	No
768k - 832k	Yes	No	No	Yes	Yes
832k - 896k	Yes	No	No	Yes	No
896k - 960k	Yes	No	No	No	Yes
960k - 1024k	Yes	No	No	No	No
1024k - 1088k	No	Yes	Yes	Yes	Yes
1088k - 1152k	No	Yes	Yes	Yes	No
1152k - 1216k	No	Yes	Yes	No	Yes
1216k - 1280k	No	Yes	Yes	No	No
1280k - 1344k	No	Yes	No	Yes	Yes
1344k - 1408k	No	Yes	No	Yes	No
1408k - 1472k	No	Yes	No	No	Yes
1472k - 1536k	No	Yes	No	No	No
1536k - 1600k	No	No	Yes	Yes	Yes
1600k - 1664k	No	No	Yes	Yes	No
1664k - 1728k	No	No	Yes	No	Yes
1728k - 1792k	No	No	Yes	No	No
1792k - 1856k	No	No	No	Yes	Yes
1856k - 1920k	No	No	No	Yes	No
1920k - 1984k	No	No	No	No	Yes
1984k - 2048k	No	No	No	No	No

Address Links for 128 kbytes Total Memory Capacity

Offset Address Space	E29	E30	E32	E31
	1-2	1-2	2-3	2-3
128k - 256k	Yes	Yes	Yes	Yes
256k - 384k	Yes	Yes	Yes	No
384k - 512k	Yes	Yes	No	Yes
512k - 640k	Yes	No	Yes	Yes
640k - 768k	Yes	No	Yes	No
768k - 896k	Yes	No	No	Yes
896k - 1024k	Yes	No	No	No
1024k - 1152k	No	Yes	Yes	Yes
1152k - 1280k	No	Yes	Yes	No
1280k - 1408k	No	Yes	No	Yes
1408k - 1536k	No	Yes	No	No
1536k - 1664k	No	No	Yes	Yes
1664k - 1792k	No	No	Yes	No
1792k - 1920k	No	No	No	Yes
1920k - 2048k	No	No	No	No

Address Links for 256 kbytes Total Memory Capacity

Offset Address Space	E29	E30	E32
	1-2	1-2	2-3
0 - 256k	Yes	Yes	Yes
256k - 512k	Yes	Yes	No
512k - 768k	Yes	No	Yes
768k - 1024k	Yes	No	No
1024k - 1280k	No	Yes	Yes
1280k - 1536k	No	Yes	No
1536k - 1792k	No	No	Yes
1792k - 2048k	No	No	No

Address Links for 512 kbytes Total Memory Capacity

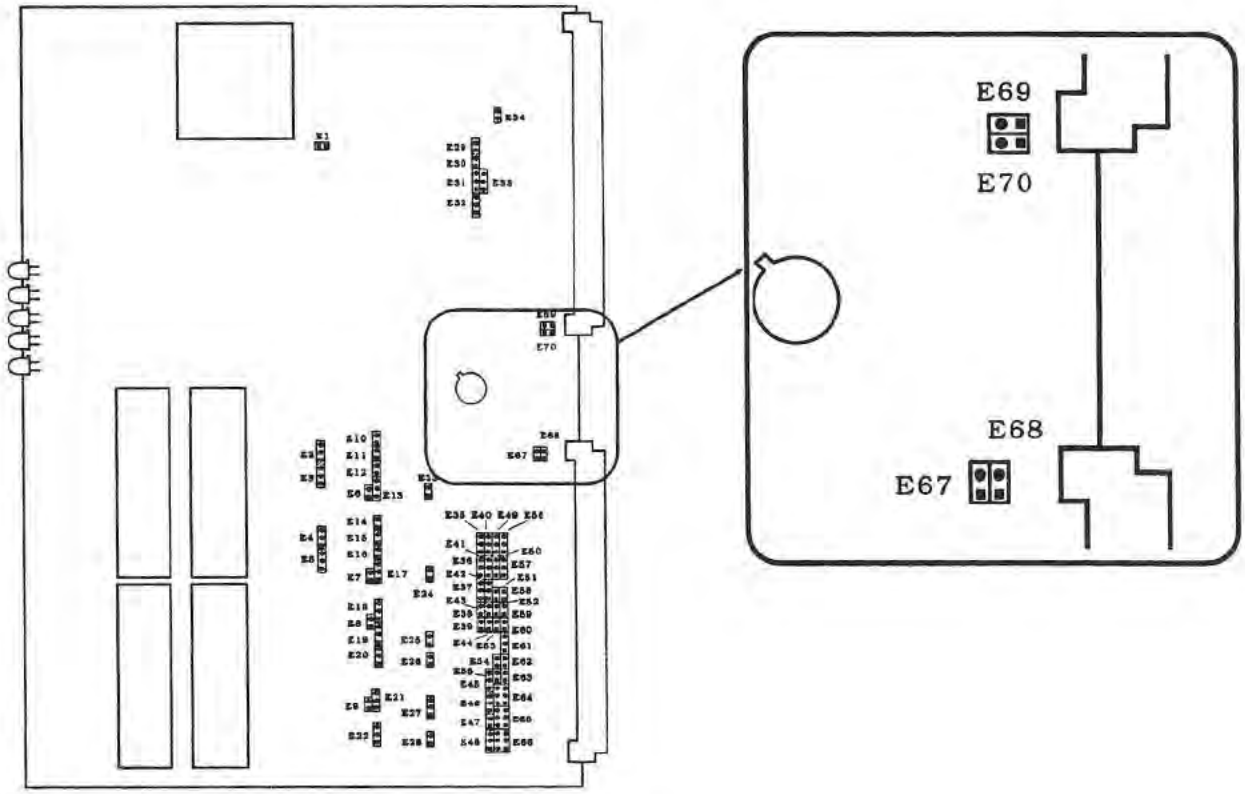
Offset Address Space	E29	E30
	1-2	1-2
0 - 512k	Yes	Yes
512k - 1024k	Yes	No
1024k - 1536k	No	Yes
1536k - 2048k	No	No

Dual Ported RAM Access Time Options

The ACM-1 board caters for dual ported RAM modules with different speeds. Depending upon the access time of the module, the links fitted are as follows:

RAM Access Time	E69	E70	E67	E68
Up to 50 ns	Yes	No	No	No
50 to 70 ns	No	Yes	No	No
70 to 100 ns	No	No	Yes	No
100 to 120 ns	No	No	No	Yes

The default RAM access time is up to 50 ns



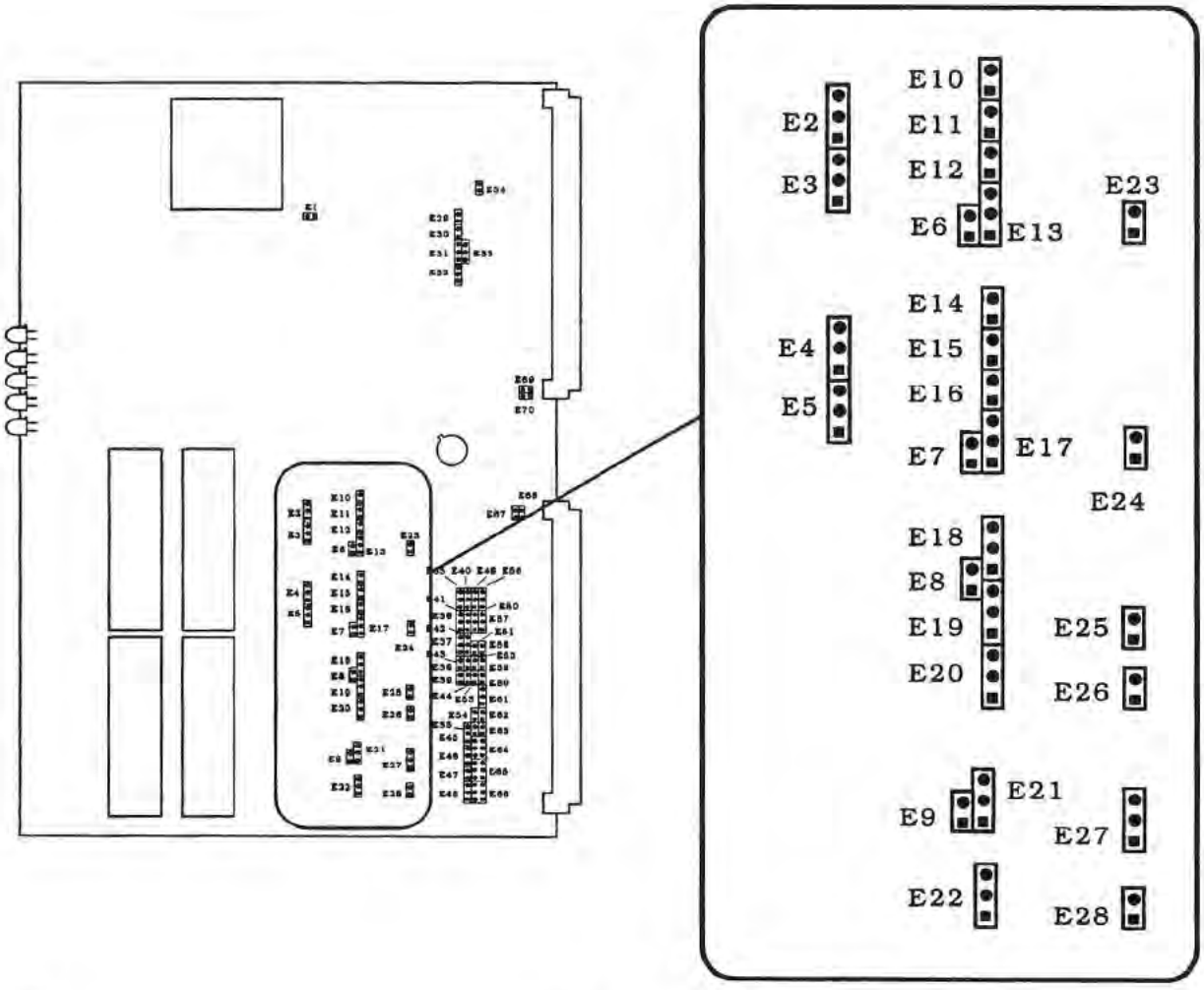
Serial Channels - Transmit Data RS422/RS423 Mode Option

All serial channels' Transmit Data have optional modes of single ended RS423 or differential RS422 operation. To select either RS423 or RS422 for pairs of channels, a group of links are set up as shown below.

Note: These link selections do not affect the selection of any of the other pairs of channels.

Transmit Data	RS423	RS422
Channels A and B (TXDA and TXDB)	Link E2 pins 1 to 2 Link E3 pins 1 to 2	Link E2 pins 2 to 3 Link E3 pins 2 to 3
Channels C and D (TXDC and TXDD)	Link E4 pins 1 to 2 Link E5 pins 1 to 2	Link E4 pins 2 to 3 Link E5 pins 2 to 3
Channels E and F (TXDE and TXDF)	Link E18 pins 1 to 2 Link E19 pins 1 to 2	Link E18 pins 2 to 3 Link E19 pins 2 to 3
Channels G and H (TXDG and TXDH)	Link E21 pins 1 to 2 Link E22 pins 1 to 2	Link E21 pins 2 to 3 Link E22 pins 2 to 3

The default configuration is for RS423 operation.



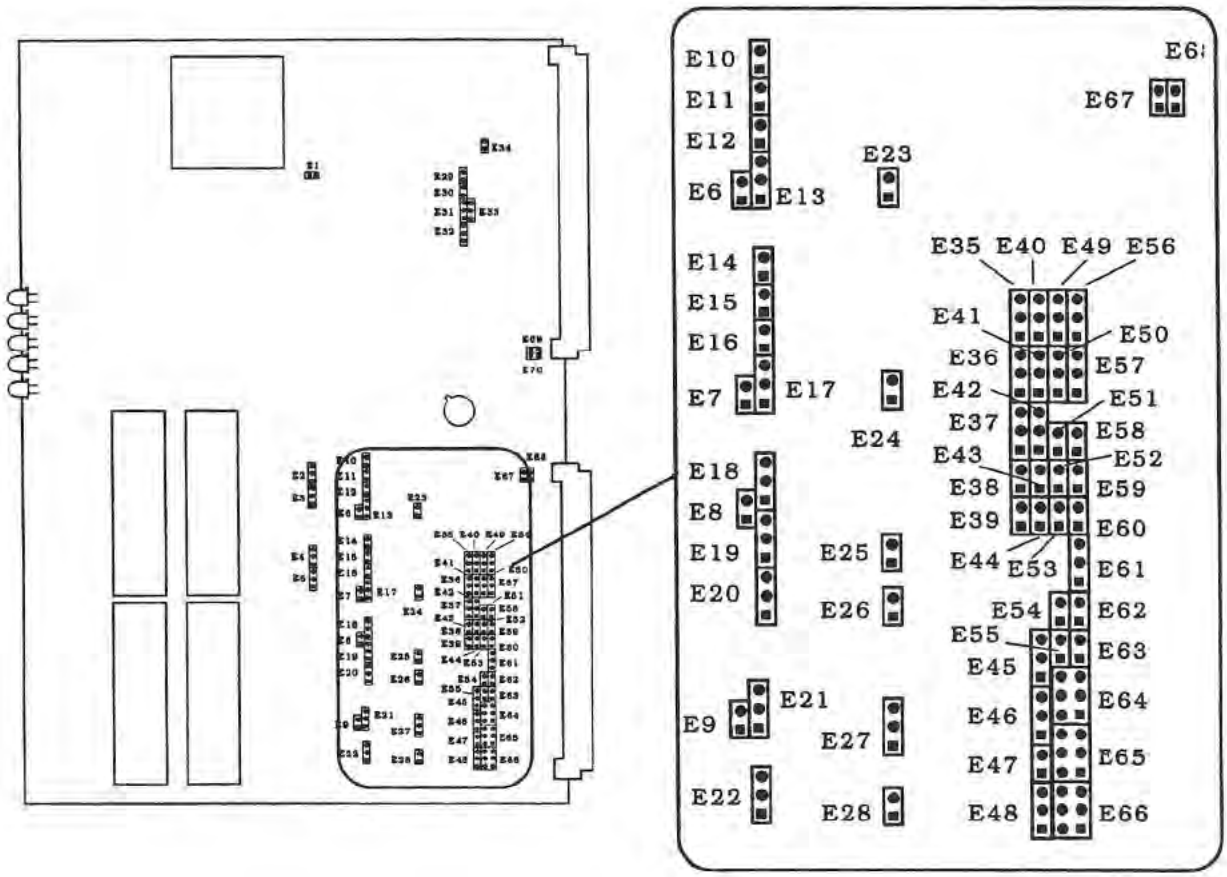
Serial Channels - Receive Data RS422/RS423 Mode Option

All serial channels' Receive Data have optional modes of single ended RS423 or differential RS422 operation. To select either RS423 or RS422 for each channel, a group of links is set up as shown below.

Note: Any channel link selection does not affect the selection of any of the other channels.

Receive Data Channel	RS423	RS422
A	Link E37 pins 2 to 3	Link E37 pins 1 to 2
B	Link E13 pins 2 to 3	Link E13 pins 1 to 2
C	Link E42 pins 2 to 3	Link E42 pins 1 to 2
D	Link E17 pins 2 to 3	Link E17 pins 1 to 2
E	Link E61 pins 2 to 3	Link E61 pins 1 to 2
F	Link E20 pins 2 to 3	Link E20 pins 1 to 2
G	Link E48 pins 2 to 3	Link E48 pins 1 to 2
H	Link E27 pins 2 to 3	Link E27 pins 1 to 2

The default configuration is for RS423 operation.

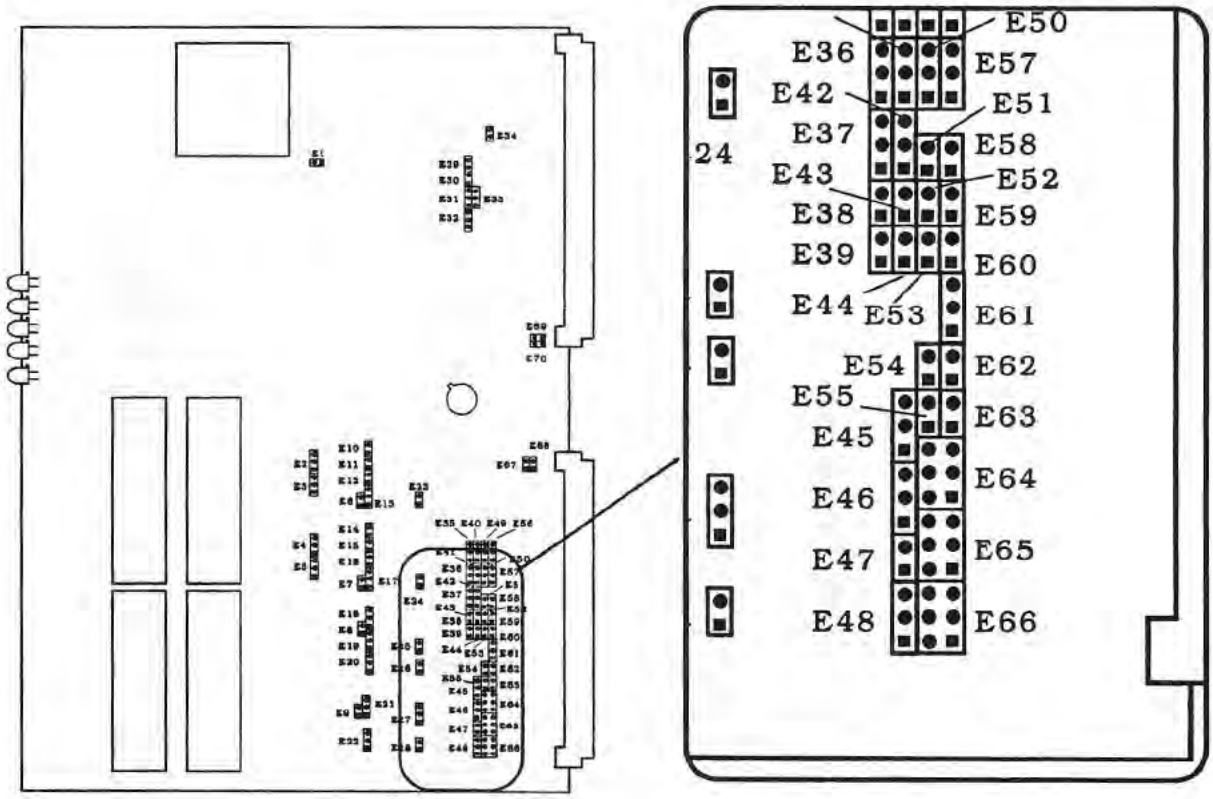


Serial Channel G Modem RS422/RS423 Mode Option

The modem control signals on channel 'G' have optional modes of single ended RS423 or differential RS422 operation. Selection of the mode of operation is determined by the mode option selected for channels G and H transmit data (E21 and E22). Additionally the following links should be set up depending upon the mode.

Modem Line	RS423	RS422
RTSG	Link E63 pins 1 to 2 not fitted	Link E63 pins 1 to 2
DTRG	Link E55 pins 1 to 2 not fitted	Link E55 pins 1 to 2
RTSH	Link E45 pins 1 to 2	Link E45 pins 2 to 3
DTRH	Link E46 pins 1 to 2	Link E46 pins 2 to 3
DCDH	Link E64 pins 2 to 3 Link E64 pins 5 to 6	Link E64 pins 1 to 2 Link E64 pin 3 to 5
CTSH	Link E65 pins 2 to 3 Link E65 pins 5 to 6	Link E65 pins 1 to 2 Link E65 pins 3 to 5
DSRH	Link E66 pins 2 to 3 Link E66 pins 5 to 6	Link E66 pins 1 to 2 Link E66 pins 3 to 5

The default configuration is for RS423 operation.

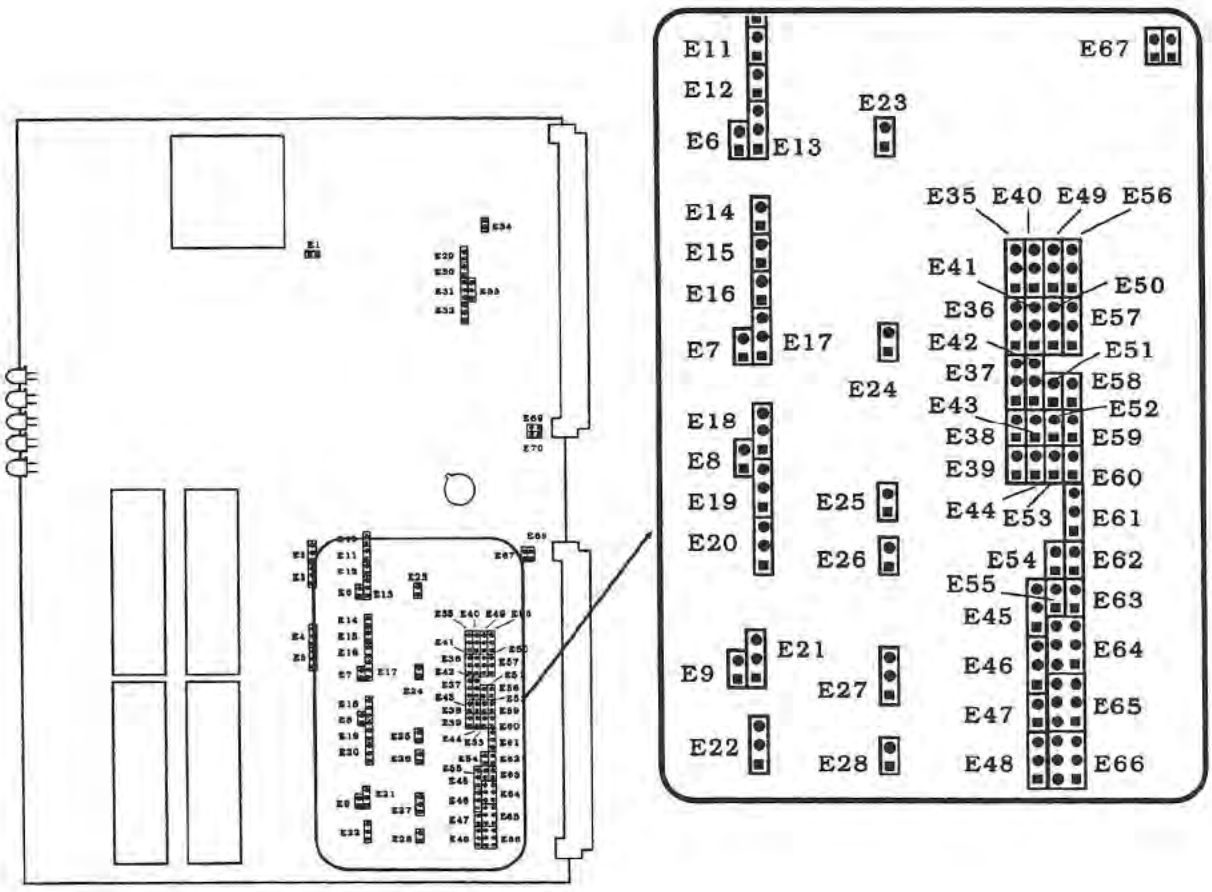


RS422 Terminator Resistors Option

An optional 150Ω resistor is included in each receiver input circuit to terminate the line. Each terminator can be selected by inserting a corresponding link, as follows:

Line Receiver	Link	Terminated Link	Not Terminated Link
Received Data (RXDA)	E38	IN	OUT
Received Data (RXDB)	E6	IN	OUT
Received Data (RXDC)	E58	IN	OUT
Received Data (RXDD)	E7	IN	OUT
Received Data (RXDE)	E53	IN	OUT
Received Data (RXDF)	E8	IN	OUT
Received Data (RXDG)	E47	IN	OUT
Received Data (RXDH)	E9	IN	OUT
(DCDH)	E64	4-5 IN	4-5 OUT
(CTSH)	E65	4-5 IN	4-5 OUT
(DSRH)	E66	4-5 IN	4-5 OUT

The default configuration is for no terminator.



Signal Ground to Local Ground Reference Option

An option is provided on the ACM-1 board to connect the channel signal ground (SIGGNDA to SIGGNDH) to the local ground. This is necessary for isolated systems, where no reference exists between the systems, to prevent excessive common mode voltages destroying the RS423/RS422 receivers.

Link	Signal	Referenced to Local Ground Link	Isolated from Local Ground Link
E23	SIGGNDA	IN	OUT
E11	SIGGNDB	IN	OUT
E24	SIGGNDC	IN	OUT
E14	SIGGNDD	IN	OUT
E39	SIGGNDE	IN	OUT
E44	SIGGNDF	IN	OUT
E62	SIGGNDG	IN	OUT
E28	SIGGNDH	IN	OUT

The default configuration is for SIGGND to be referenced to local ground.

Board Address Selection by Backplane Link Options

The base address of the ACM-1 board is defined by three edge connector pins (a31, a32 and c31) located on the P2 connector. This enables the backplane slots, by using backplane links, to define the address space of the ACM-1 board to one of eight 2 Mbyte boundaries. Further subdivision is possible on board by use of links as defined in the "Address Links Option" section. The backplane links define the board address (LOC<0>, LOC<1> and LOC<2>) as follows:

LOC<0> (P2 pin c31)	LOC<1> (P2 pin a31)	LOC<2> (P2 pin a32)	Board Base Address (A32)	Board Base Address (A24)
GND	GND	GND	$FD000000	$xx000000
OPEN	GND	GND	$FD200000	$xx200000
GND	OPEN	GND	$FD400000	$xx400000
OPEN	OPEN	GND	$FD600000	$xx600000
GND	GND	OPEN	$FD800000	$xx800000
OPEN	GND	OPEN	$FDA00000	$xxA00000
GND	OPEN	OPEN	$FDC00000	$xxC00000
OPEN	OPEN	OPEN	$FDE00000	$xxE00000

Here "OPEN" means no connection and "GND" means connected to 0 Volts.

P2 Row B Link Options

Row B of the P2 connector on the ACM-1 is multifunctional. The link options give the following functions:

1) In an A32 system (Option 1 below), the upper 8 address bits are linked onto the ACM-1. This implies that certain modem control lines for channels A and B are linked out as detailed below.

2) In an A24 system using a standard backplane (Option 2 below), certain serial channels and modem control lines need to be linked out so that no serial channels can drive the bussed P2 Row B address and data lines. In this case no links are fitted. This isolates both the VMEbus high order address and data lines from the P2 connector, and the serial line and modem control lines from the card to the P2 connector.

3) In an A24 system with a dedicated backplane (Option 3 below), all serial channels and modem control lines are available and link options connect the required signals to the P2 connector.

The linking for Row B of the P2 connector is therefore divided into three options with linking for each option as described below. The default is for an A24 system with a standard backplane (Option 2). The VMEbus connector P2 assignments for the options are listed in Chapter 3.

Option 1 - Links Required for an A32 System

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B11	E57	No	Yes	Address bit A31
B10	E50	No	Yes	Address bit A30
B9	E36	No	Yes	Address bit A29
B8	E41	No	Yes	Address bit A28
B7	E49	No	Yes	Address bit A27
B6	E56	No	Yes	Address bit A26
B5	E40	No	Yes	Address bit A25
B4	E35	No	Yes	Address bit A24

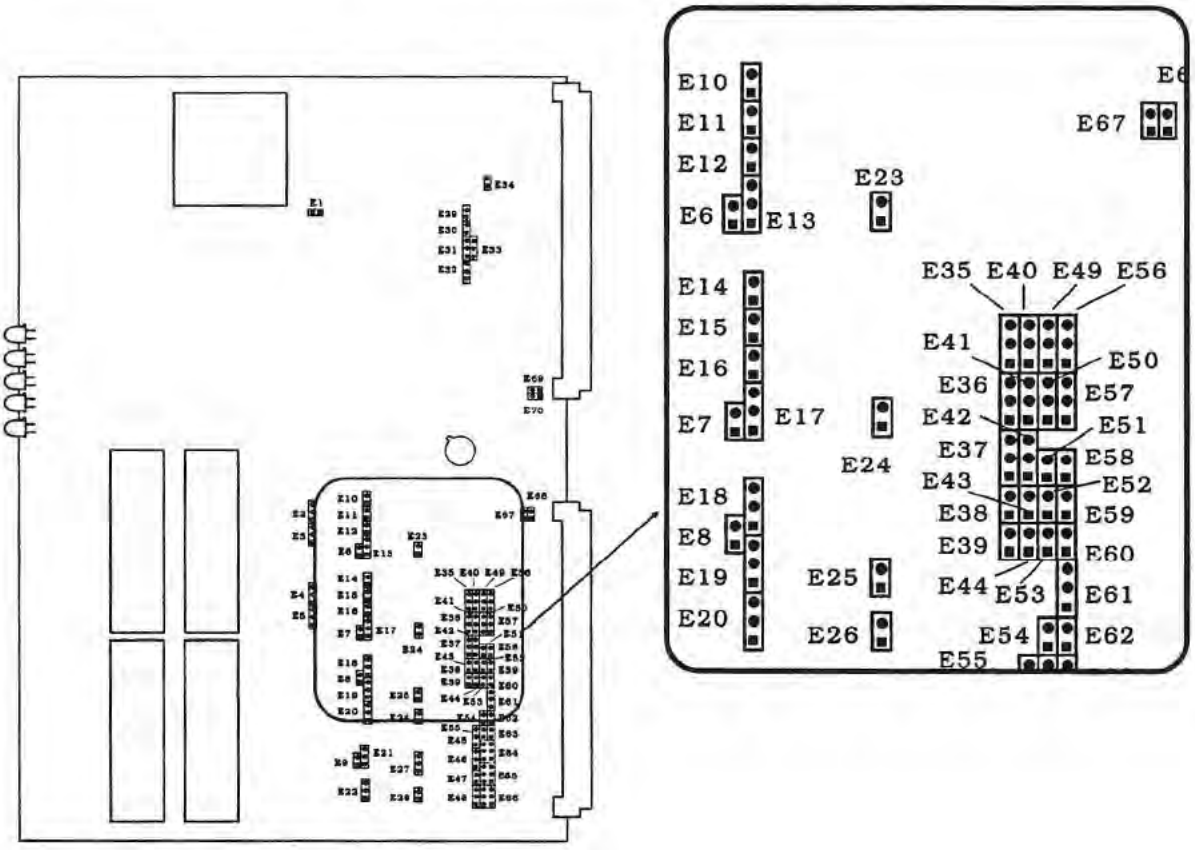
Option 2 - Links Required for an A24 System and Standard Backplane

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B11	E57	No	No	None
B10	E50	No	No	None
B9	E36	No	No	None
B8	E41	No	No	None
B7	E49	No	No	None
B6	E56	No	No	None
B5	E40	No	No	None
B4	E35	No	No	None

On both Option 1 and Option 2, the following 2-pin links MUST BE LEFT OPEN otherwise serious contention with VMEbus signals will cause malfunction and possible damage.

P2 Pin Number	Link Number	Function
B18	E25	Data Bit 20
B24	E54	Data Bit 25
B16	E60	Data Bit 18
B21	E59	Data Bit 23
B23	E26	Data Bit 24
B25	E16	Data Bit 26
B26	E15	Data Bit 27
B14	E12	Data Bit 16
B15	E10	Data Bit 17

These two options allow basic "3 wire" control on all 8 channels, and full modem control on 4 channels without having to resort to a modified backplane. This uses rows A and C of a 64-way connector. These options can be slotted anywhere in a standard VMEbus backplane.

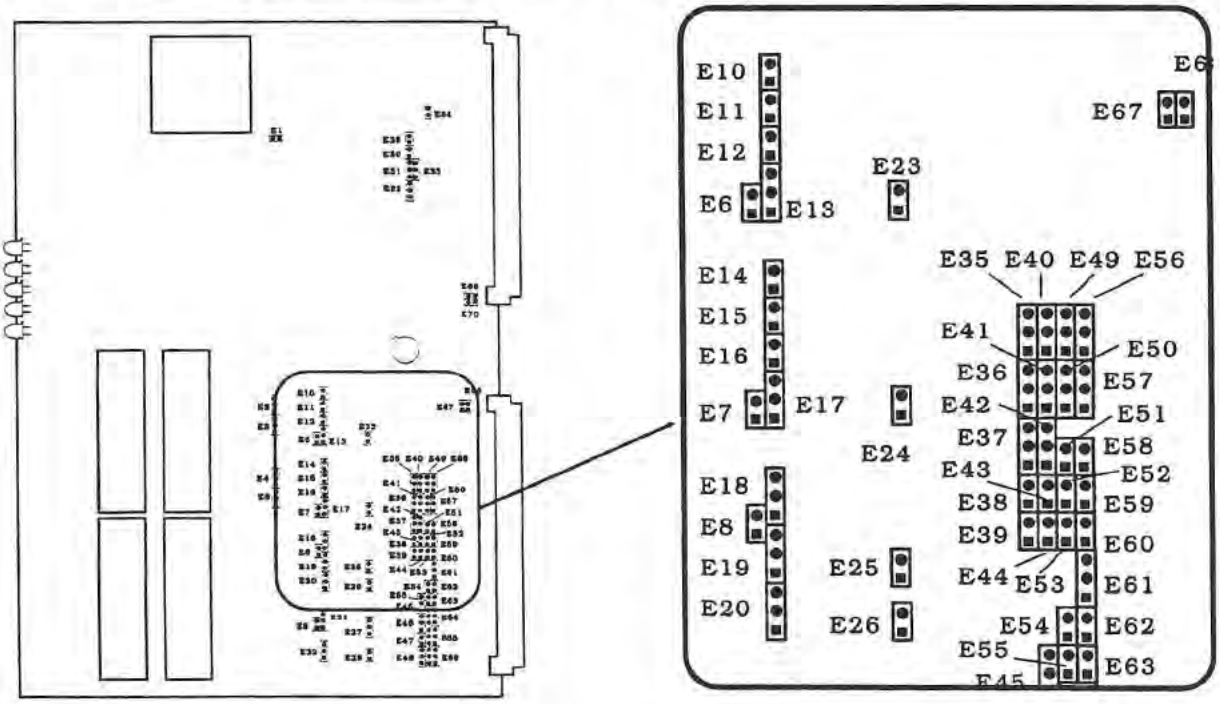


Option 3 - Links Required for an A24 System and Dedicated Backplane

(A backplane with rows A, B and C uncommitted and hard-wired to suit)

P2 Pin Number	Link Number	Pins 1-2	Pins 2-3	Function
B4	E35	Yes	No	Request to Send Channel A
B5	E40	Yes	No	Clear to Send Channel A
B6	E56	Yes	No	Data Terminal Ready Channel A
B7	E49	Yes	No	Data Carrier Detect Channel A
B8	E41	Yes	No	Data Set Ready Channel A
B9	E36	Yes	No	Request to Send Channel B
B10	E50	Yes	No	Clear to Send Channel B
B11	E57	Yes	No	Data Terminal Ready Channel B
B14	E12	Yes	No	Data Carrier Detect Channel B
B15	E10	Yes	No	Data Set Ready Channel B
B16	E60	Yes	No	Request to Send Channel C
B17	E51	Yes	No	Clear to Send Channel C
B18	E25	Yes	No	Data Terminal Ready Channel C
B19	E52	Yes	No	Data Carrier Detect Channel C
B20	E43	Yes	No	Data Set Ready Channel C
B21	E59	Yes	No	Request to Send Channel D
B23	E26	Yes	No	Clear to Send Channel D
B24	E54	Yes	No	Data Terminal Ready Channel D
B25	E16	Yes	No	Data Carrier Detect Channel D
B26	E15	Yes	No	Data Set Ready Channel D

This option allows full use of all 8 channels but requires a dedicated backplane position. The VMEbus connector P2 assignment is detailed in Chapter 3.



Default Settings

Link	Number of Pins	Function	Default Setting	Default Condition
E1	2	Clock disable/enable	1 - 2	Clock enabled
E2	3	Mode A/B	1 - 2	RS423
E3	3	Mode A/B	1 - 2	RS423
E4	3	Mode C/D	1 - 2	RS423
E5	3	Mode C/D	1 - 2	RS423
E6	2	RXD B Terminator	Open	Not terminated
E7	2	RXD D Terminator	Open	Not terminated
E8	2	RXD F Terminator	Open	Not terminated
E9	2	RXD H Terminator	Open	Not terminated
E10	2	DB17/DSR B	Open	24-bit address standard backplane
E11	2	SIGGNDB	1 - 2	Local ground
E12	2	DB16/DCD B	Open	24-bit address standard backplane
E13	3	RXD B	2 - 3	RS423
E14	2	SIGGNDD	1 - 2	Local ground
E15	2	DB27/DSR D	Open	24-bit address standard backplane
E16	2	DB26/DCD D	Open	24-bit address
E17	3	RXD D	2 - 3	RS423
E18	3	Mode E/F	1 - 2	RS423
E19	3	Mode E/F	1 - 2	RS423
E20	3	RXD F	2 - 3	RS423
E21	3	Mode G/H	1 - 2	RS423
E22	3	Mode G/H	1 - 2	RS423
E23	2	SIGGNDA	1 - 2	Local ground
E24	2	SIGGNDC	1 - 2	Local ground
E25	2	DB20/DTR C	Open	24-bit address standard backplane
E26	2	DB24/CTS D	Open	24-bit address standard backplane
E27	3	RXD H	2 - 3	RS423
E28	2	SIGGNDH	1 - 2	Local ground
E29	2	DP RAM address	1 - 2	RAM size and location 0 - 64 kbytes
E30	2	DP RAM address	1 - 2	RAM size and location 0 - 64 kbytes
E31	3	DP RAM address	2 - 3	RAM size and location 0 - 64 kbytes
E32	3	DP RAM Address	2 - 3	RAM size and location 0 - 64 kbytes

Default Settings (*continued*)

Link	Number of Pins	Function	Default Setting	Default Condition
E33	3	DP RAM address	2 - 3	RAM size and location 0 - 64 kbytes
E34	2	Reset	Open	Running
E35	3	AD24/RTS A	Open	24-bit address standard backplane
E36	3	AD29/RTS B	Open	24-bit address standard backplane
E37	3	RXD A	2 - 3	RS423
E38	2	RXD A Terminator	Open	Not terminated
E39	2	SIGGNDE	1 - 2	Local ground
E40	3	AD25/CTS A	Open	24-bit address standard backplane
E41	3	AD28/DSR A	Open	24-bit address standard backplane
E42	3	RXD C	2 - 3	RS423
E43	2	DSR C	Open	24-bit address standard backplane
E44	2	SIGGNDF	1 - 2	Local ground
E45	3	RTSH PIN	1-2	RS423
E46	3	DTRH PIN	1-2	RS423
E47	2	RXD G Terminator	Open	Not terminated
E48	3	RXD G	2 - 3	RS423
E49	3	AD27/DCD A	Open	24-bit address standard backplane
E50	3	AD30/CTS B	Open	24-bit address standard backplane
E51	2	CTS C	Open	24-bit address standard backplane
E52	2	DCD C	Open	24-bit address standard backplane
E67	2	DP RAM speed	Open	50 - 70 ns
E53	2	RXD E Terminator	Open	Not terminated
E54	2	DB25/DTR D	Open	24-bit address standard backplane
E55	2	DTRG	Open	RS423
E56	3	AD26/DTR A	Open	24-bit address standard backplane
E57	3	AD31/DTR B	Open	24-bit address standard backplane
E58	2	RXD C Terminator	Open	Not terminated
E59	2	DB23/RTS D	Open	24-bit address standard backplane

Default Settings (*continued*)				
Link	**Number of Pins**	**Function**	**Default Setting**	**Default Condition**
E60	2	DB18/RTS C	Open	24-bit address standard backplane
E61	3	RXDE	2-3	RS423
E62	3	SIGGNDG	1 - 2	Local ground
E63	2	RTSG	Open	RS423
E64	6	DCDH	2-3 5-6	RS423
E65	6	CTSH	2-3 5-6	RS423
E66	6	DSRH	2-3 5-6	RS423
E67	2	DP RAM speed	Open	50 - 70 ns
E68	2	DP RAM speed	Open	50 - 70 ns
E69	2	DP RAM speed	1 - 2	50 - 70 ns
E70	2	DP RAM speed	Open	50 - 70 ns

Chapter 5
On-board Memory

The ACM-1 board has the following on-board memory:

a) Two 8-bit ROM JEDEC format sites

b) Two 8-bit RAM JEDEC format sites

c) 16 kbytes Local RAM for processor use only

On-board ROM

The on-board CMOS ROM stores the default interrupt vector table, board initialization data, the board PMV-BITE self test program, and the application program. It has two byte-wide memory devices, giving a total ROM capacity of 128 kbytes, expandable to 512 kbytes arranged word-wide (16-bit). The devices may be any JEDEC format UVPROM or EPROM. The number of 80C186 wait states required to access the ROM can be selected by programming the on-board wait state generator within the 80C186 chip, which allows between 0 and 3 wait states. This selection is independent of the wait state selection for on-board RAM or peripherals.

On-board Global (Dual Ported) RAM

The on-board CMOS Global RAM forms a buffer area for data to be transmitted or received by the serial channels. It has two sites of JEDEC format byte-wide devices, giving a total RAM capacity of 64 kbytes, expandable to 256 kbytes arranged word-wide (16-bit). The number of 80C186 wait states required to access the RAM can be selected by programming the on-board wait state generator within the 80C186 chip, which allows between 0 and 3 wait states. This selection is independent of the wait state selection for on-board ROM or peripherals.

On-board Local RAM

The on-board CMOS Local RAM stores non-default interrupt vector tables, data used by the board PMV-BITE self test program and the operating program. It has two 28-pin byte-wide devices, giving a capacity of 16 kbytes arranged as 16-bit words.

Blank page

Chapter6
On-board Peripherals

This section describes the on-board I/O and interrupt peripherals on the ACM-1 board. The number of 80C186 wait states required to access the peripherals can be selected by programming the on-board wait state generator within the 80C186 chip, which allows between 0 and 7 wait states. This selection is independent of the wait state selection for on-board ROM or RAM.

Local Control Register

The local control register is a 16-bit output port that controls the operation of the ACM-1 board. Each bit of the register has a specific function as described in detail below. The following table shows the allocation of the control register bits:

Bit	Function
0	System Fail Indicator
1	Watch-dog Reset Timer Enable Control
2	VMEbus Interrupt IRQn* Control
3	Not used
4	Not used
5	Not used
6	Not used
7	Timer 1 Control
8	MPE Fail Indicator Bit 0
9	MPE Fail Indicator Bit 1
10	MPE Fail Indicator Bit 2
11	PMV-BITE Self-test Fail Indicator
12	Not used
13	Interrupt Level Control Bit 0
14	Interrupt Level Control Bit 1
15	Interrupt Level Control Bit 2

Interrupt Level Control

Bits 13 to 15 of the control register form bits 0 to 2 respectively of the Interrupt Level Control. The three bits form a binary code that represents the level of interrupt request required (IRQ1* to IRQ7*). Resetting the ACM-1 board automatically resets these bits to 0. The following table shows the interrupt level for a given bit pattern:

Interrupt Request Level	Bit 15	Bit 14	Bit 13
IRQ7*	1	1	1
IRQ6*	1	1	0
IRQ5*	1	0	1
IRQ4*	1	0	0
IRQ3*	0	1	1
IRQ2*	0	1	0
IRQ1*	0	0	1

PMV-BITE Self-test Fail Indicator

Bit 11 of the control register shows whether the ACM-1 board has PMV-BITE self-test failures (see Chapter 8 - BITE Self-Test). It drives the BITE FAIL indicator on the front panel. When this bit is '1', the LED is off, showing that the ACM-1 board has passed PMV-BITE self-test. When this bit is '0', the LED is on, showing that the ACM-1 board has not passed its PMV-BITE self-test. Resetting the ACM-1 board automatically resets this bit to '0'. User application software should always set this bit to '1'.

MPE Fail Indicators

Bits 8 to 10 of the control register form bits 0 to 2 respectively of the MPE fail code (see Chapter 8 - BITE Self-test). They drive LED2, LED3 and LED4 on the front panel. When each bit is '1', the corresponding LED is off. When each bit is '0', the corresponding LED is on. Resetting the ACM-1 board automatically resets these bits to '0'. User application software should always set these bits to '1'.

Timer-1 Control

Bit 7 of the control register controls Timer-1 within the 80C186 chip. The timer may be programmed to accept this input as a "gate" to time specific events, or as a counter to count the number of rising edges of this signal. Resetting the ACM-1 board automatically resets this bit to '0', which disables timing or counting.

VMEbus Interrupt IRQn* Control

Bit 2 of the control register raises an interrupt request on any of the possible 7 levels previously coded in bits 13 to 15 of this register (see above). When this bit is '1', the interrupt request is raised. After this bit has been set to '1', it must be set to '0' again before any further interrupts can be raised. Resetting the ACM-1 board automatically resets this bit to '0'.

Watchdog Reset Timer Enable Control

Bit 1 of the control register enables and disables the watchdog reset timer. When this bit is '1', the watchdog reset timer is enabled. When this bit is '0', the watchdog reset timer is disabled. Resetting the ACM-1 board automatically resets this bit to '0'.

System Fail Indicator

Bit 0 of the control register drives the VMEbus SYSFAIL* line low to indicate that a system fault has occurred. When this bit is '1', the SYSFAIL* line is not driven by the ACM-1 board. When this bit is '0', the SYSFAIL* line is driven low by the ACM-1 board. Resetting the ACM-1 board automatically resets this bit to '0'. User application software should normally set this bit to '1'.

Local Status Registers

The local status registers are one 8-bit and one 16-bit port (SR0 and SR1 respectively) that monitor the operation of certain functions of the ACM-1. Each bit of the registers has a different function as described below. The allocation of the status bits for each register follows:

Status Register SR0

Bit	Function
0	Data Set Ready Indicator Channel A
1	Data Set Ready Indicator Channel B
2	Data Set Ready Indicator Channel C
3	Data Set Ready Indicator Channel D
4	Data Set Ready Indicator Channel E
5	Data Set Ready Indicator Channel F
6	Data Set Ready Indicator Channel G
7	Data Set Ready Indicator Channel H

The Data Set Ready (DSR) modem control signals of channels A to H drive bits 0 to 7 of status register SR0 respectively. When the corresponding bit is '0', that channel is available and ready for communications. When the bit is '1', then the channel is unavailable.

Status Register SR1

Bit	Function
0	Transmit Buffer Empty Channel A
1	Transmit Buffer Empty Channel B
2	Transmit Buffer Empty Channel C
3	Transmit Buffer Empty Channel D
4	Transmit Buffer Empty Channel E
5	Transmit Buffer Empty Channel F
6	Transmit Buffer Empty Channel G
7	Transmit Buffer Empty Channel H
8	Receive Buffer Empty Channel A
9	Receive Buffer Empty Channel B
10	Receive Buffer Empty Channel C
11	Receive Buffer Empty Channel D
12	Receive Buffer Empty Channel E
13	Receive Buffer Empty Channel F
14	Receive Buffer Empty Channel G
15	Receive Buffer Empty Channel H

The TXRDY DUART output signals of channels A to H drive bits 0 to 7 of status register SR1 respectively. When the corresponding bit is '0', that channel is ready to accept another character for transmission. When the bit is '1', then that channel's transmit holding register is full.

The RXRDY/FFULL DUART output signals of channels A to H drive bits 8 to 15 of the status register SR1 respectively. When the corresponding bit is '0', that channel has at least one character received. When the bit is '1', then that channel's receive FIFO is empty.

Watch-dog Reset Timer

The ACM-1 board has a watchdog timer, which is provided to reset the ACM-1 board in the event of a software crash. To operate it, the 80C186 initialises Timer 0 within the 80C186 with a timeout count, and then enables the watchdog timer reset output using the on-board local control register bit 1 (watchdog reset timer enable control).

As all timers have an effective one-eighth CPU crystal frequency input clock, the timeout count used equals the timeout period required divided by 400 nanoseconds. When the timeout expires, the board is reset. Additionally, if required, Timer 2 can be used as a prescaler to extend the watchdog time.

VME Interrupt Generator

The VME interrupt generator allows the ACM-1 board to generate VMEbus interrupts to VME interrupt handlers in any VMEbus slot in the system.

There are seven levels of VMEbus interrupt request, numbered 1 to 7. The ACM-1 can generate an interrupt request on any one of these levels at any one time. When a VME interrupt is acknowledged, the VME interrupt handler fetches a status/ID code from the interrupt generator.

The ACM-1 board can generate any 8-bit status/ID code. The 8-bit Vector Register and the 16-bit Local Control Register control the VME interrupt generator on the ACM-1. First the 80C186 writes the 8-bit status/ID code that is to be used for the interrupt to the vector register. The level of the interrupt is then defined by writing to bits 13 to 15 of the control register, and the interrupt is raised by setting bit 2 of the control register to a logic '1'. Bit 2 must subsequently be set to logic '0' before any further interrupts can be raised

Programmable Interrupt Controller

The 80C186 has, on chip, the equivalent of an 82C59A Programmable Interrupt Controller (PIC). This controller can operate in Master mode or in a special slave mode, in which the internal interrupt controller acts as a slave to an external master. There are 5 interrupt pins dedicated to external sources, one of these being the non-maskable interrupt (NMI). The other four pins may function as:

1) Four interrupt input lines with internally generated interrupt vectors

2) An interrupt line and an interrupt acknowledge line (called the cascade mode) along with two other lines with internally generated interrupt vectors

3) Two interrupt lines and two dedicated interrupt acknowledge output lines.

If the interrupt lines are configured in cascade mode, the 80C186 interrupt controller does not generate internal interrupt vectors.

The ACM-1 board uses the following interrupt configuration:

Interrupt Level	Source of Interrupt
INT0	Serial Controllers
INT1	Test mode

Programmable Interval Timers

Three 16-bit programmable interval timers reside within the 80C186. These timers all share a common clock input which, on the ACM-1 board, has a frequency of 1/8th of the 80C186 crystal frequency (i.e. 2.5 MHz.). This gives a basic timing interval of 400 ns. The timers can operate in six different modes. In all modes, the current count can be latched and read by the 80C186 at any time. Use of the timers is as follows:

1) **Timer 0.** Used as Watch-dog Reset Timer

2) **Timer 1.** Uncommitted, may be used as a software timer with a basic timing interval of 400 ns. Terminal count may be programmed to flag an internal interrupt.

3) **Timer 2.** Uncommitted, may be used as a software timer with a basic timing interval of 400 nanoseconds. Terminal count may be programmed to flag an internal interrupt. Additionally this timer may be set up as a pre-scaler for either of the other two timers or as a DMA request source.

Programmable Wait State Generator

The 80C186 can generate a READY signal internally for each of the memory or peripheral "Chip Select" lines. The number of WAIT states to be inserted for each peripheral or memory is programmed to provide 0-3 wait states for all accesses to the area for which the chip select is active. In addition, the 80C186 may be programmed either to ignore external READY for each chip select range individually or to factor external READY with the integrated ready generator. READY control has 3 bits for each chip select line or group of lines generated by the 80C186, as follows:

READY Bits			Number of WAIT States Generated
R2	R1	R0	
0	0	0	0 wait states, external RDY also used
0	0	1	1 wait state inserted, external RDY also used
0	1	0	2 wait states inserted, external RDY also used
0	1	1	3 wait states inserted, external RDY also used
1	0	0	0 wait states, external RDY ignored
1	0	1	1 wait state inserted, external RDY ignored
1	1	0	2 wait states inserted, external RDY ignored
1	1	1	3 wait states inserted, external RDY ignored

The number of wait states assigned to the various memory and peripheral areas within the ACM-1 are as follows:

Peripheral	Number of Waits	READY Bits		
		R2	R1	R0
PROM	0	0	0	0
Local RAM	0	0	0	0
Global RAM	EXT	1	0	0
Serial Controllers	3	0	1	1
Command/Status	0	0	0	0

88C681 Dual Universal Asynchronous Receiver/Transmitter

The 88C681 Dual Universal Asynchronous Receiver/Transmitter (DUART) is a dual channel communications peripheral that functions as a serial-to-parallel or parallel-to-serial converter/controller. The DUART contains various internal features including:

1) An on-chip baud rate generator.

2) Automatic echo and local loopback for diagnostics.

3) Programmable baud rates for each receiver and transmitter selectable from:

   a. 23 fixed rates: 50 to 38.4kbaud using standard crystal

   b. One user-defined rate derived from the programmable timer/counter

   c. External 1X or 16X clock.

4) Programmable data format:

   a. 5 to 8 data bits plus parity

   b. Odd, even, no parity or force parity

   c. 1, 1.5 or 2 stop bits programmable in ^{1}/16-bit increments.

5) Parity, framing and overrun error detection.

6) Facilities for modem control.

Addressing the On-board Peripherals

The on-board allocation of addressing space for memory mapped peripherals is subdivided as shown in the following table:

Peripheral	Memory Base Address	Data Width	Read or Write
PROM (128Kbytes)	$E0000	Word	Read
Local RAM	$00000	Word	Read/Write
Dual Ported RAM	$40000	Word	Read/Write

The on-board allocation for port addressed peripherals is subdivided as follows:

Peripheral	Port Base Address	Data Width	Read or Write
Vector Register	$0000	Byte	Write
Control Register	$0080	Word	Write
Status Register0 (SR0)	$0080	Byte	Read
Status Register1 (SR1)	$0100	Word	Read
DUART Chip A	$0180	Byte	Read/Write
DUART Chip B	$01A0	Byte	Read/Write
DUART Chip C	$01C0	Byte	Read/Write
DUART Chip D	$01E0	Byte	Read/Write
Clear DUARTs	$0200	Byte	Write

The DUARTs have a group of control and status registers associated with them that are accessed by the on-board 80C186. The relative offsets of these registers from the base addresses are given below for each device:

Peripheral -88C681(DUART)	Port Address Offset from Base	Read or Write
Mode Register A (MR1A, MR2A)	$00	Read/Write
Status Register A (SRA)	$02	Read
Clock Select Register A (CSRA)	$02	Write
Interrupt Status Register, Masked (ISR)	$04	Read
Command Register A (CRA)	$04	Write
Rx Holding Register A (RHRA)	$06	Read
Tx Holding Register A (THRA)	$06	Write
Input Port Change Register (IPCR)	$08	Read
Auxiliary Control Register (ACR)	$08	Write
Interrupt Status Register, Unmasked (ISR)	$0A	Read
Interrupt Mask Register (IMR)	$0A	Write
Counter/Timer Upper Byte (CTU)	$0C	Read
Counter/Timer Upper Register (CTUR)	$0C	Write
Counter/Timer Lower Byte (CTL)	$0E	Read
Counter/Timer Lower Register (CTLR)	$0E	Write
Mode Register B (MR1B, MR2B)	$10	Read/Write
Status Register B (SRB)	$12	Read
Clock Select Register B (CSRB)	$12	Write
Reserved	$14	Read
Command Register B (CRB)	$14	Write
Rx Holding Register B (RHRB)	$16	Read
Tx Holding Register B (THRB)	$16	Write
Interrupt Vector Register (IVR)	$18	Read/Write
Input Port	$1A	Read
Output Port Configuration Register (OPCR)	$1A	Write
Start Counter Command	$1C	Read
Set Output Port Bits Command	$1C	Write
Stop Counter Command	$1E	Read
Reset Output Port Bits Command	$1E	Write

Chapter 7
Operational Firmware

Introduction

The ACM-1 operational firmware responds to commands issued by any processor in the system that has VMEbus master capabilities. The command interface is implemented by direct access from the host to data structures in the on-board Global (dual-ported) RAM that control operation of the firmware.

Communicating with the Firmware

The operational firmware uses an area of dual-port RAM to communicate with the host processor card. The dual-port RAM contains a control block for each channel and one BITE control block. The control blocks are arranged at the beginning of RAM as follows:

BITE control block	RAM_start
Channel A control block	RAM_start + $40
Channel B control block	RAM_start + $80
Channel C control block	RAM_start + $C0
Channel D control block	RAM_start + $100
Channel E control block	RAM_start + $140
Channel F control block	RAM_start + $180
Channel G control block	RAM_start + $1C0
Channel H control block	RAM_start + $200

Each control block contains initialization information, operational details and status information for the channel. The following diagram shows the format of the block:

Bit 31				Bit 0	Byte Offset
Tx enable	Rx enable	Mode			0
Tx status		Rx status			4
Speed					8
Size/parity	Stop	VME interrupt control			C
TX null		XON char	XOFF char		10
Rx lower full		Rx upper full			14
	En XOFF	En RTS	DTR Set		18
DSR Status					1C
Tx Buffer Address					20
Current Tx pointer					24
Tx Buffer Length					28
					2C
Rx Buffer Address					30
Current Rx Pointer					34
Rx Buffer Length					38
					3C

Note: All unused bits are reserved and must be cleared to '0'. Reserved values are not allowed. A bit is set to '1', and reset or cleared to '0'.

Tx Enable and Rx Enable Bytes

The Tx Enable and Rx Enable bytes request the ACM-1 to perform an activity. Requests are made by setting individual bits. The on-board firmware resets these when the desired operation is achieved. Depending on which operation is required, other configuration and/or operational data must be set up in the control block first. The following table shows the bit allocation for the enable bytes:

Bit	Meaning When Set
0	Enable data transfers
6	Close a channel
7	Initialize a channel

All other bits are reserved. When the activity is completed or under way, the bit is cleared in the enable byte.

Mode Word

The "Mode" word specifies the overall operating mode of the channel and applies to both the transmitter and receiver. This word must be set up before channel initialization.

The following table shows the fields and values:

Value	Operating Mode
$0	Single character transfer
$1	Single character, continuous operation
$2	Buffered transfer
$3	Buffered transfer, continuous operation
$7	Flow control

All other values are reserved.

Tx and Rx Status Words

The Tx Status and Rx Status words are set by the ACM-1 to indicate the result of the corresponding transmit or receive operation initiated by the host processor.

Speed Longword

The Speed longword specifies the baud rate of the transmitter and receiver. This longword must be set up before initializing the channel. The values are as follows:

Value	Baud Rate
$0	50 Baud
$2	75 Baud
$4	110 Baud
$6	134.5 Baud
$8	150 Baud
$A	200 Baud
$C	300 Baud
$E	600 Baud
$10	1050 Baud
$12	1200 Baud
$14	1800 Baud
$16	2000 Baud
$18	2400 Baud
$1A	4800 Baud
$1C	7200 Baud
$1E	9600 Baud
$20	9200 Baud
$22	38400 Baud

All other values are reserved.

Size/Parity Byte

The Size/Parity byte defines the bit size of each character and what type of parity, if any, is required. This byte must be set up before initializing the channel. The fields and values are as follows:

Bits	Value	Size/Parity
0 - 3	$0	No parity
	$1	Odd parity
	$2	No parity
	$3	Even parity
4 - 7	$5	5 bit characters
	$6	6 bit characters
	$7	7 bit characters
	$8	8 bit characters

All other values are reserved.

Stop Byte

The Stop byte defines how many stop bits are required for each character. This byte must be set up before initializing the channel. The values are as follows:

Value	Number of Stop Bits
$1	1 stop bit
$2	1.5 stop bits
$3	2 stop bits

All other values are reserved.

VME Interrupt Control Word

The VME Interrupt Control word determines which VME interrupt levels, if any, are activated, what vector value is returned, and under what conditions an interrupt is produced. The bit fields are as follows:

Bits	Meaning
0 to 7	Base vector value
8	Set to interrupt on transmit complete
9	Set to interrupt on receive complete
10	Set to interrupt on receive event
11	Reserved
12 to 14	VME interrupt level
15	Set to modify base vector value

Transmit Complete and Receive Complete interrupts result from a successful transfer. Receive event interrupts are generated when an event occurs during reception and the transfer is not halted. Bits 8 to 11 control the type of events that are required to produce interrupts. Bits 12 to 14 specify which of the VME interrupt lines (1 to 7) is to be driven when an interrupt occurs. This can be different for each channel.

When another VME card acknowledges an interrupt from the ACM-1, a vector must be returned. This value is entered in bits 0 to 7. If bit 15 is set, the vector value is modified depending on which interrupt has occurred. A transmit complete sets bit 0, a receive complete sets bit 1 and a receive event sets bit 2. Bit 3 is reserved. If the modify mode is being used, the least significant four bits of the supplied vector must be zero.

This word can be set up after initialization, but must be set before a transfer starts. If it is set to zero, no interrupt is generated.

Tx Null Byte

The Tx Null byte specifies the byte character that is used in "single character, continuous" mode to show that there is no character to transmit. This byte can be set up after initialization, but should be set to the required value before enabling continuous transfer.

XON Character

The XON Char byte specifies the character to be used for XON in flow control mode with the Enable XOFF byte set. A value must be written to this byte by the user before the XON/XOFF protocol is enabled. Typically, control-Q would be used ($11).

XOFF Character

The XOFF Char byte specifies the character to be used for XOFF in flow control mode with the Enable XOFF byte set. A value must be written to this byte by the user before the XON/XOFF protocol is used. Typically, control-S would be used ($13).

Rx Lower full

This flag is set to $0001 in flow control mode whenever the first half of the receive buffer becomes full. It is the responsibility of the user to read the characters from the lower half of the buffer and clear this flag to $0000 so that the buffer space can be reused.

Rx Upper Full

This flag is set to $0001 in flow control mode whenever the top half of the receive buffer becomes full. It is the responsibility of the user to read the characters from the top half of the buffer and clear this flag to $0000 so that the buffer space can be reused.

Enable XOFF

Setting this byte to $01 in flow control mode ensures that an XOFF character is sent by the channel whenever the receiver is at least seven-eighths full and another character is received. The receiver causes an XON character to be sent whenever at least half its buffer space is free. When the receiver receives an XON or XOFF character it enables or disables the transmitter.

Enable RTS/CTS

This field should be sent to a $01 value before initialising a channel if the RTS/CTS hardware flow control mechanism is required. When enabled this will cause the RTS line to go inactive high whenever the receiver is full. In addition, the transmitter will only start sending the next character if the CTS signal is active low. By connecting these signals between the transmitter and the receiver overflows can be avoided.

Set DTR

This single byte field should be set to $01 to take the DTR control line active for this channel. The line can be taken inactive by writing a $00. This field is initialised to $00 by the ACMS firmware after a reset.

DSR/DCD Status

This single byte field contains the current status of the DSR and the DCD control lines for this channel. A $01 will be returned if DSR is active and $02 will be returned if the DCD control line is active. These bits may be set independently. A value of $00 indicates that neither signal is currently active.

Tx Buffer Address Longword

The Tx Buffer Address longword contains the address of the transmit buffer for buffered modes or a single transmit character in single transfer mode. In single character mode, the data must be placed in the least significant byte of the longword (address $23 from the beginning of the control block). This longword can be set up after initialization, but must be set before a transfer starts. The buffer address cannot be set to point below RAM_start + $400 or beyond the top address of the RAM. It is the responsibility of the host processor to ensure that channel buffers do not conflict.

Current Tx Pointer Longword

The ACM-1 sets the Current Tx Pointer longword to show the position that the current transmit operation has reached in the transmit buffer.

Tx Buffer Length Longword

The Tx Buffer Length longword contains the length of the transmit buffer for buffered modes. This longword can be set up after initialization, but must be set before a transfer starts. Permissible lengths are between 1 and 64511 bytes.

Rx Buffer Address Longword

The Rx Buffer Address longword contains the address of the receive buffer for buffered modes or a single receive character in single transfer mode. In single character mode, the ACM-1 places data in the least significant byte of the longword (address $33 from the beginning of the control block). This longword can be set up after initialization, but must be set before a transfer starts. The buffer address cannot be set to point below RAM_start + $400 or beyond the top address of the RAM. It is the responsibility of the host processor to ensure that channel buffers do not conflict.

Current Rx Pointer Longword

The ACM-1 sets the Current Rx Pointer longword to show the position the current receive operation has reached in the receive buffer.

Rx Buffer Length Longword

The Rx Buffer Length longword contains the length of the receive buffer for buffered modes. This longword can be set up after initialization, but must be set before a transfer starts. The maximum permissible length is 64511 bytes. The minimum length in normally 1 byte. However, in flow control mode, the minimum length is 16 bytes.

Performing a Transfer

Performing a transfer can be split into two operations: initializing the channel and doing the transfer.

Initialization

A channel can only be initialized if both the transmitter and receiver ports are closed. When initializing the receiver or transmitter, the channel is set up according to the channel control block; when initializing the other port, the fact that its "partner" port has already set up the channel is noted and the initialization stage is missed out. Hence the effect is just to show successful initialization of the port. This also means that if one port is closed and the control block changed, when the port is opened again the configuration change has no effect on the channel. To successfully re-initialize the channel, both ports must be closed.

If required, a channel transmitter or receiver port can be closed by issuing a close request (set bit 6) on the relevant enable byte in the channel control block. The corresponding status word can then be polled until a channel closed code ($8000) is returned.

The configuration data can now be set up in the channel control block and an initialization request issued by setting bit 7 of the Tx/Rx enable byte. The Tx/Rx status should be polled until an Initialization Status code is returned (see the Status Reporting section). If an error is reported, the Tx/Rx port is not initialized, so a second attempt can be made immediately.

Note: Both the transmitter and receiver ports may be opened together by setting the open bit in both enable bytes at the same time, but the appropriate status word must be checked for completion of the operation before attempting to use the port.

Executing the Transfer

Once the channel transmitter or receiver has been initialized, as many transfers as required can be performed without re-initialization.

First set up the required mode and relevant buffer pointers and lengths together with the interrupt vector control word, then issue a request by setting bit 0 of the appropriate enable byte. The status can then be polled, indicating transfer in progress.

The status values returned depend on the mode of operation set up. Completion of the transfer can be indicated by an interrupt or by polling the status word for a completion code ($20xx).

Transfer Modes

Single Character Transfer Mode

In this mode, a single character byte is transferred to or from the least significant byte of the Receive or Transmit Buffer Address pointer. A completion code ($20xx) is then placed in the appropriate status word location and an interrupt generated if required. The channel can then be enabled for a new data transfer.

Single Character Continuous Mode

When receiving data in this mode, each character received is placed in the dual-port RAM in the least significant byte of the Receive Buffer Address. An interrupt is then generated if required and the RX status word is updated to indicate busy ($30xx).

If composite characters are likely to be encountered (i.e. Cursor Control keys), then the single character continuous mode of operation (mode = 1) must be used to ensure that the channel is always available for receipt of characters.

When transmitting data, the character byte in the least significant byte of the Transmit Buffer Address is compared to the Tx Null byte in the command block. If different, the character is transmitted and replaced in the Transmit Buffer Address by the Tx Null character. An interrupt can also be generated if required. The Tx Status word indicates busy at all times. The channel must be closed before it can be used in any other mode.

Buffered Mode

In this mode, the address of the buffer and the number of bytes to transfer are set up before enabling the channel transmit or receive. When the transfer finishes, a completion code ($20xx) is placed in the appropriate status word location and an interrupt generated if required. The channel can then be enabled for a new data transfer.

The host processor can interrogate the current pointer in the command block at any time to determine how far the transfer has progressed.

Buffered Continuous Mode

In this mode, the receiver is set up and data received in the same manner as in buffered mode, but when the end of the buffer is reached an interrupt may be generated if required. The receiver then continues from the start of the buffer. The transmitter transmits 'Tx Buffer Length' bytes then indicates completion in the same manner as buffered mode.

A busy status is maintained in the receiver status word throughout. The channel must be closed before it can be used in any other mode.

Flow Control Mode

In the mode, the transfer is set up and executed in the same manner as in buffered continuous mode, but when the receive buffer reaches its half way of full mark a buffer full flag is set and optionally an interrupt generated. These flags are the Rx lower full and Rx upper full entries in the channel control block which are set to $0001. The user must read the data from the receive buffer and then clear these flags to $0000 in order to ensure the buffer does not overflow. Interrupts will be generated whenever these flags are set if the VME interrupt control word has the interrupt on receive event bit set.

The transmitter operates in the same manner as in the buffered and buffered continuous modes.

In addition, if the user sets the En XOFF bytes to $01 then the XON/XOFF flow control protocol is enabled. This causes the bytes defined in the XON char and XOFF char control block entries to be used to pause transmission locally or remotely. The user should set the En XOFF byte at channel installation time if the XON/XOFF protocol is required in addition to the improved receiver buffering provided by the flow control mode.

Transmission is paused locally if the receiver receives an XOFF character and resumed when it receives an XON character. While paused the transmitter status word contains $3040.

A request to pause remote transmission by sending the XOFF character occurs each time a new character is received when the receive buffer is at least seven-eighths full. XON is sent when the receive buffer becomes half empty.

Status Reporting

Each channel control block contains two status words, one for the receiver and one for the transmitter. The status codes are split into groups identifiable by the most significant nibble (half byte).

Event Codes

Certain status codes can include an 'event code' in their lower byte. This indicates that an error or event has occurred during the transfer. This code is represented as 'xx' in the status word and has the following bit allocations:

Bits	Meaning
0, 1 & 3	Reserved
2	Receive buffer overflow
4	Overrun error in received data
5	Parity error in received data
6	Rx framing error or Tx paused by XOFF
7	Break received

Initialization Status Codes

The following codes are returned when initialization has completed. Any code other than $1000 means that initialization has failed and the channel has not been opened.

Code	Meaning
$1000	Initialization completed successfully
$1001	Illegal size or parity selected
$1002	Illegal stop bits selected
$1003	Illegal baud rate selected

Transfer Complete Status Codes

The following codes are returned when a transfer has completed. The channel does not require re-initialization after returning these values

Code	Meaning
$2000	Transfer completed successfully
$20xx	Transfer error/event occurred

Transfer In Progress Status Codes

The following codes are present while a transfer is in progress.

Code	Meaning
$3000	Transfer in progress
$30xx	Transfer event occurred with the last character

Channel Closed Status Code

The following code is returned when a channel has been closed. To continue operation, the channel must be reinitialized.

Code	Meaning
$8000	Channel closed successfully

Channel Control Status Codes

If the request is not consistent with the current state of the channel or cannot be granted, then one of the following codes is returned. (e.g. attempting a transfer on a closed channel).

Code	Meaning
$F000	Channel has not been initialized or illegal mode word
$F001	Channel has already been initialized
$F002	Channel is already closed
$F003	Illegal Buffer Address Pointer
$F004	Illegal Buffer Length

Chapter 8
BITE Self-Test

Overview

ACM-1 BITE software comprises a Minimum Processing Environment (MPE) test and on-card test software. The test software has two levels of operation to support, field self-test and diagnostic and production testing (namely "primary" and "secondary"). BITE runs automatically when powered up, on reset, or when requested to do so by a software command.

MPE Test

MPE Operations

The MPE forms the basis of an expanding kernel, i.e. elements of a system are first themselves tested and, if functional, used to test further elements. The MPE test is entered when the ACM-1 is powered up or reset, and carries out the following actions:

1) A checksum test on the ACM-1 EPROM. This adds all the bytes of the EPROM, with the exception of the last two, into a word checksum. This is then compared with the known value stored in the last two bytes.

2) A test of the ACM-1 local RAM. Using a non-destructive method preserves the RAM contents.

3) An 80C186 processor test. This exercises most of the 80186 instructions and addressing modes.

4) A test of the ACM-1 dual-port RAM. Using a non-destructive method preserves the RAM contents.

5) Finally the BITE reporting area in dual-port RAM is cleared and the control shell is accessed.

MPE Failures

If any of the above tests fail, then the MPE has failed and the ACM-1 halts.
As each section of the MPE test starts execution, the ACM-1 front panel LEDs display a code. Thus in the event of a failure, you can see where the failure occurred in the MPE test. The MPE also places an error code word into the dual-port RAM at address $2 from the beginning of RAM (see Triggering BITE With A Software Command). The LED codes and RAM values are as follows:

MPE Section	LED 2	LED 3	LED 4	Error Code
Processor failed to start	on	on	on	
EPROM checksum failed	on	off	on	$0010
Local RAM test failed	on	off	off	$0020
80C186 processor test failed	off	on	on	$0030 to $0082
Dual-port RAM test failed	off	on	off	$0090 to $0091
MPE passed	off	off	off	$0000

If the MPE tests pass, the processor executes the on-card test software followed by the operational firmware.

BITE Test Levels

The primary BITE check is an ACM-1 BITE facility that runs on power-up, after reset or after a software request. The primary BITE performs tests that can run in a system as configured.

The secondary BITE check is a superset of the primary BITE facility and includes additional tests to provide a higher degree of confidence in the ACM-1. The extra tests require external loopback connections that may not be practical with the ACM-1 *in situ*. Secondary BITE can only run after a software request.

BITE Operations

The ACM-1 BITE consists of the following tests:

1) 80C186 PIT Test. Each of the timers is tested as follows:

   a. Timer mode "halt on terminal count" is tested with different word counts. Speed is checked using a software timing loop.

   b. Timer mode "continue on terminal count" is tested. Speed is checked using a software timing loop.

   c. Timer mode "start/stop counter" is tested.

   d. Timer 2 is tested as a prescaler to Timers 0 and 1.

2) 88C681 DUART Test. Each of the eight serial channels channels is tested as follows:

   a. Test loopback at 2400, 9600 and 38.4k baud using the PIT to check timing.

   b. Test loopback with 5, 6, 7 and 8 bit data sizes.

   c. Check all DUART status register bits in TXRDY and FFULL modes using loopback.

   d. Test the break character using loopback.

   e. Test block status mode using loopback.

   f. Test the on-chip counter.

   Having performed these steps using internal loopback, if in secondary mode, repeat the steps using external loopback.

3) 80C186 Prorgammable Interrupt Controller (PIC) Test. Interrupt control of the DUARTs and timers is tested as follows:

   a. Set up the interrupt controller, and generate and verify an interrupt from each DUART.

   b. Set up the interrupt controller, and generate and verify a timer interrupt.

Triggering BITE with a Software Command

Besides performing BITE on power-up and after a reset, the operation can be triggered by a software command.

An area of memory starting at the beginning of dual-port RAM is reserved for initiating BITE and reporting BITE errors. This area is known as the BITE control block and has the following format:

Bit 31	Bit 0	Byte Offset
BITE Enable	MPE Error	0
BITE Failword		4
Error Number		8
		C
BITE Ignore Mask	BITE Test Level	10

BITE Enable Word

The BITE Enable word triggers certain BITE functions. This is done by setting the required bit. When the bit is cleared again by the resident firmware, the BITE operation is complete. The bit allocation is as follows:

Bit	Function
0	Perform cold start BITE
1	Perform warm start BITE using provided parameters
2	Switch off BITE fail LED
3	Switch on BITE fail LED

When running "cold start" BITE, the on-board software re-starts from the beginning, executing MPE test and primary BITE on all modules. When running "warm start" BITE, a more detailed way of controlling BITE is provided.

BITE Ignore Mask Word

The BITE Ignore Mask word can be used to disable specific test modules. The bit allocation is as follows:

Ignore Mask Bit	Meaning
0	Set to ignore the 80C186 PIT test
1	Not used
2	Not used
3	Not used
4	Set to ignore the 88C681 DUART channel A test
5	Set to ignore the 88C681 DUART channel B test
6	Set to ignore the 88C681 DUART channel C test
7	Set to ignore the 88C681 DUART channel D test
8	Set to ignore the 88C681 DUART channel E test
9	Set to ignore the 88C681 DUART channel F test
10	Set to ignore the 88C681 DUART channel G test
11	Set to ignore the 88C681 DUART channel H test
12	Set to ignore the 80C186 PIC test
13	Not used
14	Not used
15	Not used

BITE Test Level Word

The BITE Test Level word determines whether to run in primary or secondary BITE mode as follows:

Value	Meaning
$1	Primary BITE
$2	Secondary BITE

BITE Status

The remainder of the BITE control block is used for reporting information. The MPE Error word contains the result of the MPE test. If the MPE passed, this is zero; if the MPE failed, then it contain one of the values listed in the MPE section. The BITE Failword has a bit set in it for each test module that has failed. The bit allocation is the same as for the BITE Ignore Mask word. The Error Number longword contains the fail point from the last test module that failed.

BITE Configuration Data

The Asynchronous Communications Multiplexer (ACM) test requires 7 longwords of configuration data, 4 of which are the fixed usage test masks and slave processor timeout.

Configuration Data Word	Meaning	
0	ACM test ignore mask. Bits are set to ignore tests as follows:	
	Bit	**Test**
	0	ACM buffer memory test
	1	ACM reset BITE test
	2	ACM software trigger BITE test
	3	ACM transfer test
1	ACM test serious failure mask. The bit allocation is as above	
2	ACM test fatal failure mask. The bit allocation is as above	
3	ACM test slave processor time-out count	
4	ACM buffer memory address	
5	ACM on-card BITE timeout value	
6	ACM on-card BITE test module mask.	
7	ACM channel count	

BITE Fail Data

The ACM test package is split up into 4 test modules. The following sections describe each test module along with its fail data. The 'mask_code' indicates the test module that failed and the 'fail_code' indicates the fail point within the test module. The remainder of the fail data is fail point specific.

The following table shows the 'mask_code' values for the test modules:

Mask_Code	Test Module
0001	ACM buffer memory test
0002	ACM reset BITE test
0004	ACM software trigger BITE test
0008	ACM transfer test

ACM Buffer Memory Tests

The ACM buffer memory is tested destructively using two different algorithms. A fast algorithm is used in primary mode and a slow algorithm is used in secondary and tertiary modes.

The fast algorithm first fills the memory with longword values of $5AA55AA5 and checks for unexpected exceptions (0210); the codes in parentheses refer to the relevant fail_code (see following tables for ACM fail points). The algorithm then reads, checks and inverts the memory. Failures occur if the read value was incorrect (0230) or reading/writing caused an unexpected exception (0220). The algorithm repeats the previous operation of reading, checking and inverting (0240 and 0250) so that the memory should now contain values of $5AA55AA5 again. The algorithm then carries out a final read and check of the memory (0260 and 0270).

The slow algorithm writes a pattern of longwords (addr, addr+1, addr+2 etc) into memory, checking for any exceptions (0300). The algorithm then reads the memory and checks values (0320). If an exception occurs when reading, a failure is reported (0310). The algorithm repeats the sequence a total of 32 times. Each time, it rotates the pattern in a particular location left by one bit.

On any failure in the ACM buffer memory test module, the following information is stored in the fail_data words:

Fail_Data Word	Meaning
0	Address of memory location that has failed
1	Value actually read from memory

The following table shows the fail points for the buffer memory test:

Mask_Code	Fail_Code	Meaning
0001	0050	Initial power-up BITE has not completed
0001	0210	Fast algorithm, initial write caused exception
0001	0220	Fast algorithm, first check/invert caused exception
0001	0230	Fast algorithm, first check/invert comparison failed
0001	0240	Fast algorithm, second check/invert caused exception
0001	0250	Fast algorithm, second check/invert comparison failed
0001	0260	Fast algorithm, final check caused exception
0001	0270	Fast algorithm, final comparison failed
0001	0300	Slow algorithm, write caused exception
0001	0310	Slow algorithm, read caused exception
0001	0320	Slow algorithm, read/write value comparison failed

ACM On-Card BITE Tests

The on-card BITE tests are carried out as follows. Writing to the ACM's BITE control block triggers the on-card BITE. The value written determines the mode in which the on-card BITE runs. In cold start mode, the BITE runs as from power-up. In warm start mode, the test mode level and test ignore mask are extracted from the BITE control block. The test ignore mask originates from configuration data word 6.

In both cases, the completion of power-up BITE is first verified (0050). Software then retriggers the BITE. If BITE does not complete in time, a failure occurs (0100). If BITE returns a non-zero fail word, a failure is also indicated (0200).

On any failure in the ACM on-card BITE test module, the following information is stored in the fail_data words:

Fail_Data Word	Meaning
0	On-card BITE MPE Error Word
1	On-card BITE Fail Word
2	On-card BITE Error Number Word
3	On-card BITE MPE Vector Number

The following table shows the fail points for the on-card BITE tests:

Mask_Code	Fail_Code	Meaning
0002	0050	Initial power-up BITE has not completed
0002	0100	Reset on-card BITE timed out, 'BITE enable flag' was not cleared
0002	0200	Reset on-card BITE returned a non-zero fail word
0004	0050	Initial power-up BITE has not completed
0004	0100	Software triggered on-card BITE timed out, 'BITE enable flag' was not cleared
0004	0200	Software triggered on-card BITE returned a non-zero fail word

ACM Transfer Tests

The transfer test takes the following form. After verifying the completion of any power-up BITE (0004), the test resets all the ACM channels and checks for either Channel Deleted or Channel Non-Existent status. If this did not occur, the test reports a failure (0004 to 000C). The test then sets up all the ACM channels to carry out transfers of length 4031 bytes, at 9600 baud. If the transmitting channel fails to initialize (0020) or the receiving channel fails to initialize (0010), a failure results.
The test enables system interrupts and starts the receiver, followed by the transmitter. If the receiver (0110) or the transmitter (0120) does not return a Transfer In Progress status, the test reports an error.

The test now waits for the transfers to complete, by checking for the reception of an interrupt from channel 7. This is done by polling the interrupt flag table. If the interrupt does not occur within a certain time, then an error results (0200). The test also checks the interrupts from the remaining channels (0210).
The test checks the transmitter status (0310) and the receiver status (0300) for completion values. It also checks the buffer progress pointers (0320 and 0330). Finally it checks the received data (0340).

On any failure in the ACM transfer test module, the following information is stored in the fail_data words:

Fail_Data Word	Meaning
0	Channel under test (0 to 7)
1	Transmitter status for channel under test
2	Receiver status for channel under test
3	For fail_code (0340), failed location in receive buffer
4	For fail_code (0340), transmit buffer data
5	For fail_code (0340), receive buffer data

The following table shows the fail points for the transfer tests:

Mask_Code	Fail_Code	Meaning
0008	0004	Initial power-up BITE has not completed
0008	0008	Receiver channel did not reset
0008	000C	Transmitter channel did not reset
0008	0010	Receiver channel did not initialize
0008	0020	Transmitter channel did not initialize
0008	0110	Receiver channel did not start transfer
0008	0120	Transmitter channel did not start transfer
0008	0200	Interrupt from channel 7 was not received
0008	0210	Interrupt flag table was not correct
0008	0300	Receiver task did not complete successfully
0008	0310	Transmitter task did not complete successfully
0008	0320	Receiver buffer pointer was not correct
0008	0330	Transmitter buffer pointer was not correct
0008	0340	Received data was incorrect

Blank Page

Appendix A
Example Programs

Example 1 - ACM-1 Transfer Test

This example will set up each channel on the ACM-1 to transfer a number of characters from a transmit buffer into a receive buffer (with the use of an external loopback connector which connects the transmit data for each channel into its own receive data channel). On completion of the transfer, the ACM-1 raises a VMEbus interrupt for each channel, the data is then compared between transmit and receive buffers for errors.

```
include(../shell/comon.h)
include(acm.h)

extern  struct mode_byte_type test_mode_byte;
extern  struct test_list_entr *test_list_ptr;
extern  struct fail_data_buff *fail_data_ptr;

extern  int mfptr;



/*
        External functions
*/

extern int sys_cntrl();
extern int set_level();

define(BSIZE, 0xfbf)

acmxfer(modno)
int modno;
{
        register CHAN_BLK *chan_ptr;
        register BITE_BLK *bite_ptr;
        register BYTE   *tptr, *rptr;
        register LWORD  c, i, j;
        LWORD   buffers, fpnt;
        BYTE    tdata, rdata;

        if (test_mode_byte.level == 1) return(PASS);

        /*
         * set frame ptr
         */

        mfptr = fail_data_ptr->fail_flag.nfail;
        mfptr = (mfptr > 15) ? 15 : mfptr;
        fail_data_ptr->fail_fram[mfptr].mask_code = modno;

        /*
         * fetch configuration data and setup buffer addresses
         */
```

```
i = test_list_ptr->conf_data->test_data[ACM_DATA];
bite_ptr = (BITE_BLK *)i;
chan_ptr = (CHAN_BLK *)(i + sizeof(BITE_BLK));
buffers = i + 0x400;

/*
 * Wait for possible completion of power-up BITE
 */

i = test_list_ptr->conf_data->test_data[ACM_DATA + 1];
while ((--i > 0) && bite_ptr->bite_enable);
fpnt = 0x4;
if (i == 0) goto failure;


/*
 * Attempt to close all channels, either wait for a CLOSEOK or a
 * channel not open.
 */

for (i = 0; i < 8; i++)
{
        chan_ptr[i].rxenable = CLOSE;
        c = 0x10000;
        fpnt = 0x8;
        while ((chan_ptr[i].rxstatus < CLOSEOK) && --c);
        if (c == 0) goto failure;

        chan_ptr[i].txenable = CLOSE;
        c = 0x10000;
        fpnt = 0xc;
        while ((chan_ptr[i].txstatus < CLOSEOK) && --c);
        if (c == 0) goto failure;
}


/*
 * Fill transmit buffers and clear receive buffers
 */

tptr =  (BYTE *) buffers;
for (i = 0; i < 8; i++)
{
        for (j = 0; j < BSIZE; j++)
        {
                *tptr = (BYTE) -j + i;
                *(tptr + BSIZE) = 0;
                tptr++;
        }
        tptr += BSIZE;
}


/*
 * Set channel parameters. Note channel A interrupt control is setup
 * seperately. This is to prevent an interrupt level of 0 and to try
 * out the vector modification function.
 */
```

```
j = buffers;
for (i = 0; i < 8 ; i++)
{
        chan_ptr[i].mode = BUFF;
        chan_ptr[i].speed = S9600;
        chan_ptr[i].parsize = BITS8 | NOPAR;
        chan_ptr[i].stop = STOP1;
        chan_ptr[i].intcontrol = (0x0680 + i*0x1009) & 0x76FF;
        chan_ptr[i].txnull = TXNULL;
        chan_ptr[i].txlen = BSIZE;
        chan_ptr[i].rxlen = BSIZE;
        chan_ptr[i].txpoint = 0;
        chan_ptr[i].rxpoint = 0;
        chan_ptr[i].txaddr = (BYTE *) j;
        j += BSIZE;
        chan_ptr[i].rxaddr = (BYTE *) j;
        j += BSIZE;
}
chan_ptr[0].intcontrol = 0x9680;

/*
 * Initialize all channels and wait for complete
 */

for (i = 0; i < 8; i++)
{
        chan_ptr[i].rxenable = INIT;
        c = 0x10000;
        fpnt = 0x10;
        while ((chan_ptr[i].rxstatus != INITOK) && --c);
        if (c == 0) goto failure;

        chan_ptr[i].txenable = INIT;
        c = 0x10000;
        fpnt = 0x20;
        while ((chan_ptr[i].txstatus != INITOK) && --c);
        if (c == 0) goto failure;
}


/*
 * Enable VME interrupt levels 4-7, enable processor to acknowledge
 * on level 7 (channel 3). Clear interrupt flag table and new interrupt
 * indicator.
 */

sys_cntrl(SYS_VINT_ENB, 0xFE);
set_level( 7 );
sys_cntrl(SYS_VINT_CLR);


/*
 * Enable each receive channel.
 */
```

```
for (i = 0; i < 8; i++)
{
        chan_ptr[i].rxenable = ENABLE;
        c = 0x10000;
        fpnt = 0x110;
        while ((chan_ptr[i].rxstatus != TRIP) && --c);
        if (c == 0) goto failure;
}


/*
 * Enable each transmit channel.
 */

for (i = 0; i < 8; i++)
{
        chan_ptr[i].txenable = ENABLE;
        c = 0x10000;
        fpnt = 0x120;
        while ((chan_ptr[i].txstatus != TRIP) && --c);
        if (c == 0) goto failure;
}

/*
 * Wait for interrupt to be received ie. last channel complete
 */

fpnt = 0x200;
c = 0x400000;
set_level( 0 );
while (!sys_cntrl(SYS_VVEC_ASK, 184) && --c)
         sys_cntrl(SYS_VINT_ENB, 0xFE);
if (c == 0) goto failure;

/*
 * Check interrupt flag table, only receive complete interrupts should
 * have been generated. The interrupt from channel 1 is tested seperate
 * -ly since it is used to test the vector modification function.
 */

fpnt = 0x210;
if (sys_cntrl(SYS_VVEC_ASK, 128) != 0x4) goto failure;
for (i = 1; i < 8; i++)
{
        if (sys_cntrl(SYS_VVEC_ASK, 128+i*8) != (1 < i)) goto failure;
}


/*
 * Check completion statuses and rx/tx pointers are correct and verify
 * received data.
 */

for (i = 0; i < 8; i++)
{
        rptr = chan_ptr[i].rxaddr;
        tptr = chan_ptr[i].txaddr;
```

```
                fpnt = 0x300;
                if (chan_ptr[i].rxstatus != TROK)
                        goto failure;

                fpnt = 0x310;
                if (chan_ptr[i].txstatus != TROK)
                        goto failure;

                fpnt = 0x320;
                if (chan_ptr[i].rxpoint != (rptr + BSIZE))
                        goto failure;

                fpnt = 0x330;
                if (chan_ptr[i].txpoint != (tptr + BSIZE))
                        goto failure;

                fpnt = 0x340;
                for (j = 0; j < BSIZE; j++)
                {
                        if ((tdata = *tptr) != (rdata = *rptr))
                                goto failure;
                        tptr++;
                        rptr++;
                }
        }

        sys_cntrl(SYS_VINT_DIS, 0xFE);
        set_level( 7 );
        return(PASS);

failure:
        fail_data_ptr->fail_fram[mfptr].fail_code = fpnt;
        fail_data_ptr->fail_fram[mfptr].fail_data[0] = i;
        fail_data_ptr->fail_fram[mfptr].fail_data[1] = chan_ptr[i].txstatus;
        fail_data_ptr->fail_fram[mfptr].fail_data[2] = chan_ptr[i].rxstatus;
        fail_data_ptr->fail_fram[mfptr].fail_data[3] = j;
        fail_data_ptr->fail_fram[mfptr].fail_data[4] = tdata;
        fail_data_ptr->fail_fram[mfptr].fail_data[5] = rdata;
        fail_data_ptr->fail_flag.nfail++;
        sys_cntrl(SYS_VINT_DIS, 0xFE);
        set_level( 7 );
        return(FAIL);
}
```

Example 2 - Test Program

This program will exercise all 8 channels of the ACM-1 at a variety of speeds and data sizes. By using an external loopback connector Data is transmitted from the transmitter and received on the same channel's receiver.

```
#include "comms.h"

BYTE            bitcon[NCHAN];
BYTE            parcon[NCHAN];
BYTE            parsize[NCHAN];
BYTE            stop[NCHAN];
WORD            mode[NCHAN];
WORD            intcontrol[NCHAN];
LWORD           speed[NCHAN];
LWORD           trans_cnt[NCHAN];

LWORD           passcount;

BYTE            *data_ptr, *chk_ptr;

BITE_BLK        *bite_ptr = 0;
CHAN_BLK        *chan_ptr;

extern void     putchar();
extern BYTE     tm_rawchar();

main()
{
        extern void display();
        extern void setparams();

        register BYTE   tdata, rdata;
        register WORD   rstatus,tstatus;
        register LWORD  i, j;


        /*
         * Initialise system via cold BITE
         */

        bite_ptr = (BITE_BLK *) BITE_COMM;
        bite_ptr->bite_enable = 0x0001;
        while (bite_ptr->bite_enable != 0)
                ;
        if ((rstatus = bite_ptr->mpe_error) != 0)
        {
                /*
                 * MPE failure
                 */

                xprintf("\r\n MPE failed - status = %4x\r\n", rstatus);
                asm("trap &14");
        }
```

```
chan_ptr = (CHAN_BLK *) CHN_COMM;       /* Locate command blocks */

/*
 * Set up transmit and clear receive data buffers
 */

data_ptr =  (BYTE *)BUFFERS;
for (i = 0; i < NCHAN; i++)
{
        for (j = 0; j < BUFFSEP; j++)
        {
                *data_ptr = (BYTE) 0x15;
                *(data_ptr + BUFFSEP) = 0;
                data_ptr++;
        }
        data_ptr += BUFFSEP;
}

/*
 * Initialize channel parameters
 */

j = BUFFERS;
for (i = 0; i < NCHAN; i++)
{
       mode[i] = BUFF;
       speed[i] = S50;
       bitcon[i] = BITS5;
       parcon[i] = NOPAR;
       parsize[i] = bitcon[i] | parcon[i];
       stop[i] = STOP1;
       intcontrol[i] = NOINT;
       setparams(i);
       chan_ptr[i].txaddr = j;
       j += BUFFSEP;
       chan_ptr[i].rxaddr = j;
       j += BUFFSEP;
}

/*
 * Initialize all channels and wait for complete. Clear log
 */

for (i = 0; i < NCHAN; i++)
{
       chan_ptr[i].rxenable = INIT;
       chan_ptr[i].txenable = INIT;
       while (chan_ptr[i].rxenable && chan_ptr[i].txenable != 0)
               ;

       trans_cnt[i] = 0;
}

/*
 * Transfer on each channel. When a channel is complete
 * close it  then re-open and re-enable.
 */
```

```
passcount = 0;

do
{
        for (i = 0; i < NCHAN; i++)
        {
                if (tm_rawchar(1) == 3)
                        goto endit;

                if (chan_ptr[i].rxenable == 0 && chan_ptr[i].txenable == 0)
                {
                        rstatus = chan_ptr[i].rxstatus;
                        tstatus = chan_ptr[i].txstatus;
                        if (rstatus == INITOK)
                        {
                                /*
                                 * Rx initialized so clear receive
                                 * buffer and enable it
                                 */

                                chk_ptr = (BYTE *)(chan_ptr[i].rxaddr);
                                for (j=0; j<chan_ptr[i].rxlen; j++)
                                {

                                        *chk_ptr = 0;
                                        chk_ptr++;
                                }
                                chan_ptr[i].rxenable = ENABLE;
                                while(chan_ptr[i].rxstatus != TRIP);
                                if (tstatus == INITOK)
                                {
                                        chan_ptr[i].txenable = ENABLE;
                                }
                                else
                                {
                                        display(i,2);
                                        goto endit;
                                }
                        }
                        else if (rstatus == TROK)
                        {
                                /*
                                 * Rx finished so close channel
                                 */

                                chan_ptr[i].rxenable = CLOSE;
                                if (tstatus == TROK)
                                {
                                        chan_ptr[i].txenable = CLOSE;
                                }
                                else
                                {
                                        display(i,2);
                                        goto endit;
                                }
```

```
                                        /*
                                         * Check the data
                                         */

                                        data_ptr = (BYTE *)(chan_ptr[i].txaddr);
                                        chk_ptr = (BYTE *)(chan_ptr[i].rxaddr);
                                        for (j=0; j < chan_ptr[i].rxlen; j++)
                                        {
                                                if ((tdata = *data_ptr) !=
                                                          (rdata = *chk_ptr))
                                                {
                                                        /*
                                                         * Data error
                                                         */

                                                        display(i,3);
                                                        goto endit;
                                                }
                                                data_ptr++;
                                                chk_ptr++;
                                        }
                                        display(i,0);
                                        setparams(i);
                                        trans_cnt[i]++;
                                        if (tm_rawchar(1) == 3)
                                                goto endit;
                                }
                                else if (rstatus == CLOSEOK)
                                {
                                        /*
                                         * Rx closed so re-open it
                                         */

                                        chan_ptr[i].rxenable = INIT;
                                        if (tstatus == CLOSEOK)
                                        {
                                                chan_ptr[i].txenable = INIT;
                                        }
                                        else
                                        {
                                                display(i,2);
                                                goto endit;
                                        }
                                }
                                else if (rstatus != TRIP)
                                {
                                        /*
                                         * Error in receiver
                                         */

                                        chan_ptr[i].rxstatus = rstatus;
                                        display(i,1);
                                        goto endit;
                                }
                        }
                }
        } while (1);
```

```
        /*
         * This is the end point of the test
         */

endit:
        xprintf("\r\n Channel A - %ld iterations.", trans_cnt[0]);
        xprintf("\r\n Channel B - %ld iterations.", trans_cnt[1]);
        xprintf("\r\n Channel C - %ld iterations.", trans_cnt[2]);
        xprintf("\r\n Channel D - %ld iterations.", trans_cnt[3]);
        xprintf("\r\n Channel E - %ld iterations.", trans_cnt[4]);
        xprintf("\r\n Channel F - %ld iterations.", trans_cnt[5]);
        xprintf("\r\n Channel G - %ld iterations.", trans_cnt[6]);
        xprintf("\r\n Channel H - %ld iterations.", trans_cnt[7]);

        asm("trap &14");


}



void display(i,j)

int     j;                      /*Display option*/
long    i;                      /*Channel identifier*/

/* This routine displays the channel and option
        i = channel     0-7
        j = display option       0 - Pass
                                 1 - Receiver error (R)
                                 2 - Transmitter error (T)
                                 3 - Data error (f)
*/
{

        if (j == 0)
        {
                /*
                 * Pass
                 */

                if (i == (NCHAN - 1))
                {
                        /*
                         * Last channel passed
                         * Increment pass count and display
                         */

                        xprintf("\r\n PASS      %ld", ++passcount);
                }
        }
        else
        {
                /*
                 * Display channel
                 */
```

```
                switch(i) {
                        case 0: putchar('A');
                                break;
                        case 1: putchar('B');
                                break;
                        case 2: putchar('C');
                                break;
                        case 3: putchar('D');
                                break;
                        case 4: putchar('E');
                                break;
                        case 5: putchar('F');
                                break;
                        case 6: putchar('G');
                                break;
                        case 7: putchar('H');
                                break;
                        default: putchar('****');
                                break;
                        }

                switch(j) {
                        case 0: break;
                        case 1: putchar('R');
                                break;
                        case 2: putchar('T');
                                break;
                        case 3: putchar('f');
                                break;
                        default: putchar('!!!!');
                                break;
                        }
        }
}

void setparams(i)

long    i;                                      /*Channel number */
/*
 * This routine updates the parameters for the specified channel
 */
{
        chan_ptr[i].mode = mode[i];
        chan_ptr[i].speed = speed[i];
        chan_ptr[i].parsize = parsize[i];
        chan_ptr[i].stop = stop[i];
        chan_ptr[i].intcontrol = intcontrol[i];
        chan_ptr[i].txnull = TXNULL;
        chan_ptr[i].txlen = LENGTH * (i + 1);
        chan_ptr[i].rxlen = LENGTH * (i + 1);
        chan_ptr[i].txpoint = chan_ptr[i].rxpoint = 0;

        speed[i] += 2;
        if (speed[i] > S38400)
        {
```

```
                /*
                * Top speed exceeded. Reset to 50 baud
                */

               speed[i] = S50;
               stop[i]++;
               if (stop[i] > STOP2)
               {
                       /*
                        * Back to 1 stop bit
                        */

                        stop[i] = STOP1;
                        if (bitcon[i] == 0x80)
                        {
                                /*
                                 * Back to 5 data bits
                                 */

                                bitcon[i] = BITS5;
                                parcon[i]++;
                                if (parcon[i] > EVENPAR)
                                {
                                        /*
                                         * Back to no parity
                                         */

                                        parcon[i] = NOPAR;

                                }
                        }
                        else
                        {
                                bitcon[i] += 0x10;      /* Next bit size */
                        }
                        parsize[i] = bitcon[i] | parcon[i];
               }
        }
}
```

```
      MODULE:  COMMS.H (DEFINITIONS MODULE)

typedef unsigned char  BYTE;
typedef unsigned short WORD;
typedef unsigned long  LWORD;

typedef struct  bite_block
{
        WORD    bite_enable;            /* Initiate BITE        */
        WORD    mpe_error;              /* MPE Error Code       */
        LWORD   bite_failword;          /* BITE fail word       */
        LWORD   error_number;           /* BITE fail number     */
        LWORD   vector_number;          /* Not used             */
        WORD    ignore_mask;            /* BITE ignore mask     */
        WORD    test_level;             /* BITE test level      */
        WORD    return_data[22];        /* Specific fail data   */
} BITE_BLK;


typedef struct  chan_block
{
        BYTE    txenable;
        BYTE    rxenable;
        WORD    mode;
        WORD    txstatus;
        WORD    rxstatus;
        LWORD   speed;
        BYTE    parsize
        BYTE    stop;
        WORD    intcontrol;
        BYTE    txnull;
        BYTE    resv1[3];
        LWORD   resv2[3];
        LWORD   txaddr;
        LWORD   txpoint;
        LWORD   txlen;
        LWORD   resv3;
        LWORD   rxaddr;
        LWORD   rxpoint;
        LWORD   rxlen;
        LWORD   resv4;
} CHAN_BLK;


/* BITE controls */

#define BITE_COMM       0x01c00000      /* Base address of BITE block */
#define COLD_BITE       0x0001
#define WARM_BITE       0x0002
#define BITE_ON         0x0004
#define BITE_OFF        0x0008
#define PRIMARY_BITE    0x0001
#define SECONDARY_BITE  0x0002
#define BITE_DONE       0x0000
#define BITE_PASS       0x00000000
#define TEST_ALL        0x00000000
```

```
/* Command control */

#define CHN_COMM        0x01c00040      /* Start address */

#define NCHAN           8

#define ENABLE          0x01
#define INIT            0x80
#define CLOSE           0x40

#define CHAR            0x0000
#define CCHAR           0x0001
#define BUFF            0x0002
#define CBUFF           0x0003

#define LENGTH          0x07                    /* Basic buffer length */
#define BUFFERS         BITE_COMM + 0x1000      /* Start of buffers */
#define BUFFSEP         0x800                   /* Buffer separations */

#define S50             0x00000000
#define S75             0x00000002
#define S110            0x00000004
#define S134            0x00000006
#define S150            0x00000008
#define S200            0x0000000A
#define S300            0x0000000C
#define S600            0x0000000E
#define S1050           0x00000010
#define S1200           0x00000012
#define S1800           0x00000014
#define S2000           0x00000016
#define S2400           0x00000018
#define S4800           0x0000001A
#define S7200           0x0000001C
#define S9600           0x0000001E
#define S19200          0x00000020
#define S38400          0x00000022

#define STOP1           0x01
#define STOP1_5         0x02
#define STOP2           0x03
#define BITS5           0x50
#define BITS6           0x60
#define BITS7           0x70
#define BITS8           0x80

#define NOPAR           0x00
#define ODDPAR          0x01
#define EVENPAR         0x03

#define TXNULL          0xA5

#define NOINT           0x0000
#define INTVECT1        0x1355
```

```
/* Channel statuses */

#define INITOK          0x1000
#define PZERR           0x1001
#define STPERR          0x1002
#define BDERR           0x1003

#define TROK            0x2000

#define TRIP            0x3000

#define CLOSEOK         0x8000

#define NOTINIT         0xF000
#define ALINIT          0xF001
#define ALCLOSED        0xF002
#define ILADDR          0xF003
#define ILLEN           0xF004
```

Example 3 - IFX Driver Routine

```
#include <compiler.h>
#include <vrtxvisi.h>
#include <ifxvisi.h>
#include <radstone\ifxparams.h>
#include <radstone\acm-1.h>



#define DEBUG( x )          if( tx_debug ) printf( "%s", x )
#define DEBUGV( a, y )      if( tx_debug ) printf( "%x", a )
#define SET_IVECTORS( t, r, e ) *((DWORD *)((vec + 1) << 2)) = (DWORD)t ; \
                                *((DWORD *)((vec + 2) << 2)) = (DWORD)r ; \
                                *((DWORD *)((vec + 4) << 2)) = (DWORD)e
#define TIME_OUT            10      /* 0.1 second (approx) for control actions */
#define TXFR_TIMEOUT        1000    /* 10 second buffered mode txfr timeout */
#define INSTL_TIMEOUT       1000    /* 10 second timeout for channel to free */
#define ACM_TO_VME( a )             (a + acm1_base_addr)
#define CHAN_BASE( c )              (acm1_base_addr + ((c+1)*ACM_BLK_SIZE))
#define CHAN_FREE           0x0001  /* Used to indicate channel may be installed */
#define TX_TXFR_DONE        0x0002  /* Set to indicate channel free to transmit */
#define RX_TXFR_DONE        0x0004  /* Set to indicate channel free to receive */


extern  void    aise0(), aise1(), aise2(), aise3() ;
extern  void    aise4(), aise5(), aise6(), aise7() ;
extern  void    aisx0(), aisx1(), aisx2(), aisx3() ;
extern  void    aisx4(), aisx5(), aisx6(), aisx7() ;
extern  void    aisr0(), aisr1(), aisr2(), aisr3() ;
extern  void    aisr4(), aisr5(), aisr6(), aisr7() ;


typedef struct {
        BYTE    chan ;  /* Needed when driver called with just the dcb */
        DWORD   flags ; /* Used to signal channel free + txfr states */
} ACM1_INFO ;

IFXSCB          *acm1_IFXSCB_addr[ NUM_CHANS ] ;  /* Used by ISRs in acm1b.s */
IFXDDCB         acm1_ddcb[ NUM_CHANS ] ;          /* linked to dcb on install */
ACM1_INFO       acm1_info[ NUM_CHANS ] ;          /* used for global data */
DWORD           acm1_base_addr ;                  /* Set by an ifx_install() */
int             tx_debug = 0 ;


/* Prototypes for functions defined in this file */
void            bcopy ( char *src, char *dest, int len ) ;
int             acm_hw_init( IFXPARAM *options ) ;
int             acm_hw_rx_enable( BYTE chan ) ;
int             acm_hw_tx_open( BYTE chan ) ;
int             acm_hw_open( BYTE chan ) ;
void            acm_hw_tx_enable( BYTE chan ) ;
int             acm_hw_close( BYTE chan ) ;
void            acm1eint( int chan ) ;
```

```
int             ACM1Initialise( IFXPARAM *params ) ;
int             ACM1TransmitDriver( int chan, int ch ) ;
void            rx_over( DWORD ch ) ;
int             ACM1DeviceDriver( int opcode, IFXDCB *dcb_ptr, IFXPARAM *p1 ) ;
void            tx_over( DWORD ch ) ;
void            rx_over( DWORD ch ) ;

void
bcopy (src, dest, len)
char    *src, *dest ;
int     len ;
{
        while( len-- )
                *dest++ = *src++ ;
}


int
acm_hw_init( options )
IFXPARAM        *options ;
{
        /*
        Copies data to the correct ACM-1 control block
        and installs the rx, tx and event interrupt vectors. This
        routine does not issue the initialise channel command to
        the ACM-1.
        */
        acm1    *m, *p ;
        BYTE    vec ;
        WORD    chan = options->acm1.install.channel ;
        int     err ;


        err = RET_OK ;

        m = (acm1 *)CHAN_BASE( chan ) ;
        p = options->acm1.install.setup ;

        if (chan >= NUM_CHANS) {
                printf( "init: ACM-1 illegal channel used\r\n");
                err = IFXEDVNTFOUND ;           return( err ) ;
        }

        bcopy( (char *)p, (char *)m, sizeof(acm1) ) ;

        vec = (p-interrupt_ctrl & BASE_VECTOR_MASK) ;

        switch( chan ) {
        case 0 :        SET_IVECTORS( aisx0, aisr0, aise0 ) ;
                        break ;
        case 1 :        SET_IVECTORS( aisx1, aisr1, aise1 ) ;
                        break ;
        case 2 :        SET_IVECTORS( aisx2, aisr2, aise2 ) ;
                        break ;
        case 3 :        SET_IVECTORS( aisx3, aisr3, aise3 ) ;
                        break ;
        case 4 :        SET_IVECTORS( aisx4, aisr4, aise4 ) ;
                        break ;
```

```
        case 5 :        SET_IVECTORS( aisx5, aisr5, aise5 ) ;
                        break ;
        case 6 :        SET_IVECTORS( aisx6, aisr6, aise6 ) ;
                        break ;
        case 7 :        SET_IVECTORS( aisx7, aisr7, aise7 ) ;
                        break ;
        default :       break ;
        }

        return( err ) ;
}

int
acm_hw_rx_enable( chan )
BYTE chan ;
{
        /*
        Re-enables a channel for reception of a character
        or a block or characters.
        */
        acm1    *m ;
        int     count ;
        WORD    status ;
        int     err = RET_OK ;


        m = (acm1 *)CHAN_BASE( chan ) ;

        if (chan >= NUM_CHANS) {
                printf( "rx_enable: ACM-1 bad channel\r\n");
                return ;
        }

        /* Enable channel */
        m->rx_ctrl = ENABLE_TXFRS ;

        count = 0 ;
        do { status = m->rx_status ;
                sc_delay( 1 ) ;
                if( count++ >= TIME_OUT ) {
                        printf( "open: ACM-1 bad rx enable (status=");
                        printf( "%x",  status ) ;
                        printf( ")\r\n");
                        err = IFXEIOERR ;
                        break ;
                }
        } while ((status & 0xFF00) != TXFR_ACTIVE) ;

        return( err ) ;
}
```

```
int
acm_hw_tx_open( chan )
BYTE chan ;
{
        /*
        Sends the initialise command to an ACM-1 transmit channel
        */
        acm1    *m ;
        DWORD   status ;
        DWORD   count ;
        int     err = RET_OK ;


        m = (acm1 *)CHAN_BASE( chan ) ;

        if (chan >= NUM_CHANS) {
                printf( "tx open: ACM-1 illegal channel used\r\n");
                return ;
        }

        count = 0 ;

        m->tx_ctrl = INIT_CHANNEL ;
        do { status = m->tx_status ;
                sc_delay( 1 ) ;
                if( count++ >= TIME_OUT ) {
                        printf( "tx open: ACM-1 bad open (status=");
                        printf( "%x",  status ) ;
                        printf( ")\r\n");
                        err = IFXEIOERR ;
                        break ;
                }
        } while ((status != GOOD_INIT) && (status != INIT_ALREADY_DONE)) ;

        return( err ) ;
}



int
acm_hw_open( chan )
BYTE chan ;
{
        /*
        Sends the initialise command to the receive channel and
        transmit channel. If this channel is in a single character
        mode it will then enable the receiver for reception of a
        character. In buffered modes, this is done at the time of
        the ifx_read() call.
        */
        acm1    *m ;
        DWORD   status ;
        DWORD   count ;
        int     err = RET_OK ;
```

```
        m = (acm1 *)CHAN_BASE( chan ) ;

        if (chan >= NUM_CHANS) {
                printf( "open: ACM-1 illegal channel used\r\n");
                return( IFXEIOERR ) ;
        }

        /*
        Open tx channel first to ensure good line level for
        rx channel when loopbacks are present
        */
        err = acm_hw_tx_open( chan ) ;

        if (err == RET_OK) {

                /* Init channel */
                m->rx_ctrl = INIT_CHANNEL ;
                count = 0 ;
                do { status = m->rx_status ;
                        sc_delay( 1 ) ;
                        if( count++ >= TIME_OUT ) {
                                printf( "open: ACM-1 bad rx open (status=");
                                printf( "%x",  status ) ;
                                printf( ")\r\n");
                                err = IFXEIOERR ;
                                break ;
                        }
                } while ((status != GOOD_INIT)&&(status != INIT_ALREADY_DONE)) ;
        }

        /*
        Only ready for reception if in a single character mode, for buffered
        modes wait for a ifx_read() call.
        */
        if ((m->mode == CHAR_TXFR) || (m->mode == CHAR_TXFR_CONT))
                if (err == RET_OK) err = acm_hw_rx_enable( chan ) ;
        return( err ) ;
}



void
acm_hw_tx_disable( chan )
BYTE chan ;
{
        /*
        Disables a channel from sending any characters.
        */
        acm1    *m ;
        int     count ;
        WORD    status ;

        m = (acm1 *)CHAN_BASE( chan ) ;
```

```
        if (chan >= NUM_CHANS) {
                printf( "tx_disable: ACM-1 bad channel\r\n");
                return ;
        }

        count = 0 ;
        m->tx_ctrl = CLOSE_CHANNEL ;
        do {    status = m->tx_status ;
                sc_delay( 1 ) ;
                if( count++ >= TIME_OUT ) {
                        printf( "hw_close: ACM-1 bad tx close (status=");
                        printf( "%x",  status ) ;
                        printf( ")\r\n");
                        break ;
                }
        } while ((status != GOOD_CLOSE) && (status != ALREADY_CLOSED)) ;

}



void acm_hw_tx_enable( chan )
BYTE chan ;
{
        /*
        Enables a single character of a block of characters to
        be sent by a channel.
        */
        acm1   *m ;
        int     count ;
        WORD    status ;
        m = (acm1 *)CHAN_BASE( chan ) ;

        if (chan >= NUM_CHANS) {
                printf( "tx_enable: ACM-1 bad channel\r\n");
                return ;
        }

        /* Enable channel */
        m->tx_ctrl = ENABLE_TXFRS ;

        count = 0 ;
        do { status = m->tx_status ;
                sc_delay( 1 ) ;
                if( count++ >= TIME_OUT ) {
                        printf( "open: ACM-1 bad tx enable (status=");
                        printf( "%x",  status ) ;
                        printf( ")\r\n");
                        break ;
                 }
        } while ((status & 0xFF00) != TXFR_ACTIVE) ;
}
```

```
int
acm_hw_close( chan )
BYTE chan ;
{
        acm1    *m ;
        WORD     status ;
        DWORD    count ;
        int      err = RET_OK ;


        m = (acm1 *)CHAN_BASE( chan ) ;

        if (chan >= NUM_CHANS) {
                printf( "hw_close: ACM-1 bad channel\r\n");
                return( IFXEIOERR ) ;
        }

        /* Close channel */
        m->rx_ctrl = CLOSE_CHANNEL ;
        count = 0 ;

        do {
                status = m->rx_status ;
                sc_delay( 1 ) ;
                if( count++ >= TIME_OUT ) {
                        printf( "hw_close: ACM-1 bad rx close (status=");
                        printf( "%x",  status ) ;
                        printf( ")\r\n");
                        err = IFXEIOERR ;
                        break ;
                }
        } while ((status != GOOD_CLOSE) && (status != ALREADY_CLOSED)) ;

        acm_hw_tx_disable( chan ) ;

        return( err ) ;
}



void
acm1eint( chan )
WORD    chan ;
{
        BYTE            ch ;
        DWORD           status ;
        acm1            *m ;


        m = (acm1 *)CHAN_BASE( chan ) ;

        printf( "event: ACM1 event occurred \r\n" ) ;
```

```
        status = m->rx_status ;
        if( (status & 0xFF00) != TXFR_ACTIVE )
                printf( "receive: ACM1 transfer no longer active \r\n" ) ;

        if( (status & 0x00FF) != 0x0000 ) {
                printf( "receive: ACM1 receive event occurred " ) ;
                switch (status & 0x00FF) {
                case OVER_RUN :
                        printf( "OVER_RUN" ) ;
                        break ;
                case PARITY_ERROR :
                        printf( "PARITY_ERROR" ) ;
                        break ;
                case FRAMING_ERROR :
                        printf( "FRAMING_ERROR" ) ;
                        break ;
                case BREAK_RXED :
                        printf( "BREAK_RXED" ) ;
                        break ;
                default :
                        printf( "%x",  status ) ;
                        break ;
                }
                printf( "\r\n" ) ;

        }

}



int
ACM1Initialise( params )
IFXPARAM        *params ;
{
        /*
        This will set up the control block for the channel and
        then send the initialise command to the receiver and
        transmitter. In addition, when in a single character mode
        it will ready the receiver for reception of a character. For
        buffered modes this will be done after an ifx_read() call.
        */
        WORD    chan = params->acm1.install.channel ;
        int     err ;


        DEBUG( "ACM1Initialise: channel =" ) ;
        DEBUGV( chan, 4 ) ;
        DEBUG( "/r/n" ) ;

        err = acm_hw_close( chan ) ;
        if (err == RET_OK) err = acm_hw_init( params ) ;
        if (err == RET_OK) err = acm_hw_open( chan ) ;

        return( err ) ;
}
```

```
int
ACM1TransmitDriver( chan, ch )
int     chan, ch ;
{
        acm1            *m ;


        m = (acm1 *)CHAN_BASE( chan ) ;
        m->tx_buffer = (DWORD)ch ;
        m->tx_ctrl = ENABLE_TXFRS ;
}


void
rx_over( ch )
DWORD   ch ;
{
        /*
        Called by receive interrupt service routine to
        signal the end of a buffered mode transfer.
        */
        int     err ;
        acm1    *cblk ;


        sc_fpost( acm1_info[ ch ].flags, RX_TXFR_DONE, &err ) ;

        /*
        Will have multiple postings during buffered continous mode
        which are ignored.
        */
        cblk = (acm1 *)CHAN_BASE( ch ) ;
        if ((cblk->mode != BUFF_TXFR_CONT) && (err != RET_OK))
                printf( "ACM-1: spurious rx interrupt\r\n" ) ;
}


void
tx_over( ch )
DWORD   ch ;
{
        /*
        Called by transmit interrupt service routine to
        signal the end of a buffered mode transfer.
        */
        int     err ;
        acm1    *cblk ;


        sc_fpost( acm1_info[ ch ].flags, TX_TXFR_DONE, &err ) ;
```

```
        /*
        Will have multiple postings during buffered continous mode
        which are ignored.
        */
        cblk = (acm1 *)CHAN_BASE( ch ) ;
        if ((cblk->mode != BUFF_TXFR_CONT) && (err != RET_OK))
                printf( "ACM-1: spurious tx interrupt\r\n" ) ;
}



int
ACM1DeviceDriver( opcode, dcb_ptr, p1 )
int     opcode ;
IFXDCB  *dcb_ptr ;
IFXPARAM        *p1 ;
{
        int             err = RET_OK ;
        int             chan, mode ;
        ACM1_INFO       *info ;
        int             i ;
        acm1            *ctrl_blk ;


        switch( opcode ) {

        case IFXFDRIVER :
                /* Save base address of the ACM-1 card */
                acm1_base_addr = (DWORD)(p1-acm1.driver.base) ;

                /*
                Clear any old values so they should cause problems if not
                correctly initailised before use.

                Create flag group to indicate if the channel is in use
                and if a rx ot tx transfer is in progress.
                */
                for (i=0; i<NUM_CHANS; i++) {
                        acm1_IFXSCB_addr[ i ]  = 0 ;
                        acm1_ddcb[ i ].ddt     = 0 ;
                        acm1_info[ i ].chan    = NUM_CHANS + 1 ;
                        acm1_info[ i ].flags = sc_fcreate( &err ) ;
                        if (err == RET_OK)
                                /*
                                Mark channel as free for installation and
                                transfers
                                */
                                sc_fpost( acm1_info[i].flags, CHAN_FREE, &err) ;
                }
                break ;

        case IFXFRMDRIVER :
                acm1_base_addr = 0 ;
                for (i=0; i<NUM_CHANS; i++) {
                        sc_fdelete( acm1_info[ i ].flags, 1, &err ) ;
```

```
        }
        break ;

case IFXFINSTALL :
        chan = p1->acm1.install.channel ;

        /*
        Wait INSTL_TIMEOUT ticks for channel to become free
        then hold on to it.
        */
        sc_fpend( acm1_info[ chan ].flags, INSTL_TIMEOUT,
                        CHAN_FREE, 1, &err ) ;
        if (err != RET_OK)
                break ;
        sc_fclear( acm1_info[ chan ].flags, CHAN_FREE, &err ) ;

        acm1_IFXSCB_addr[ chan ]  = (IFXSCB *)dcb_ptr->dt ;
        dcb_ptr->my_driver = &acm1_ddcb[ chan ] ;
        acm1_ddcb[ chan ].ddt = (char *)&acm1_info[ chan ] ;
        acm1_info[ chan ].chan = chan ;
        if (err == RET_OK)
                err = ACM1Initialise( p1 ) ;
        break ;

case IFXFIOCTL:
        /*
        The circular buffer manager cannot be installed
        with either of the buffered modes.
        */
        info = (ACM1_INFO *)dcb_ptr->my_driver-ddt ;
        chan = info->chan ;
        ctrl_blk = (acm1 *)CHAN_BASE( chan ) ;
        if ((ctrl_blk->mode == BUFF_TXFR) ||
            (ctrl_blk->mode == BUFF_TXFR_CONT))
                err = IFXEIOERR ;
                break ;
        }
        if (p1->ioc.op == IFXOTXRDY)
                err = ACM1TransmitDriver( chan, p1->ioc.prm.sio.ch ) ;
        break ;

case IFXFREMOVE :
        info = (ACM1_INFO *)dcb_ptr->my_driver-ddt ;
        chan = info->chan ;
        err = acm_hw_close( chan ) ;
        if (err == RET_OK)
                sc_fpost( acm1_info[ chan ].flags, CHAN_FREE, &err ) ;
        break ;

case IFXFREAD :
        /*
        Read and writes must be handled by the circular
        buffer manager when using either of the single
        character modes.
        */
        info = (ACM1_INFO *)dcb_ptr->my_driver->ddt ;
        chan = info->chan ;
        ctrl_blk = (acm1 *)CHAN_BASE( chan ) ;
```

```
            if ((ctrl_blk->mode == CHAR_TXFR) ||
                (ctrl_blk->mode == CHAR_TXFR_CONT)) {
                    err = IFXEIOERR ;
                    break ;
            }
            ctrl_blk->rx_buffer   = (DWORD)(p1->rdwr.buff) ;
            ctrl_blk->rx_curr_ptr = (DWORD)(p1->rdwr.buff) ;
            ctrl_blk->rx_buff_len = p1->rdwr.nbytes ;
            acm_hw_rx_enable( chan ) ;      /* start transfer */
            /*
            Wait TXFR_TIMEOUT vrtx ticks for rx interrupt service routine
            to set the TXFR_DONE flag on this channel
            */
            sc_fpend( acm1_info[ chan ].flags, TXFR_TIMEOUT,
                            RX_TXFR_DONE, 1, &err ) ;
            if (err == RET_OK)
                    sc_fclear( acm1_info[ chan ].flags, RX_TXFR_DONE, &err);

            *(p1->rdwr.act) = ctrl_blk->rx_curr_ptr ;
            break ;

        case IFXFWRITE :
            /*
            Read and writes must be handled by the circular
            buffer manager when using either of the single
            character modes.
            */
            info = (ACM1_INFO *)dcb_ptr->my_driver->ddt ;
            chan = info->chan ;
            ctrl_blk = (acm1 *)CHAN_BASE( chan ) ;
            if ((ctrl_blk->mode == CHAR_TXFR) ||
                (ctrl_blk->mode == CHAR_TXFR_CONT)) {
                    err = IFXEIOERR ;
                    break ;
            }
            ctrl_blk->tx_buffer   = (DWORD)(p1->rdwr.buff) ;
            ctrl_blk->tx_curr_ptr = (DWORD)(p1->rdwr.buff) ;
            ctrl_blk->tx_buff_len = p1->rdwr.nbytes ;
            acm_hw_tx_enable( chan ) ;      /* start transfer */
            /*
            Wait TXFR_TIMEOUT vrtx ticks for tx interrupt service routine
            to set the TXFR_DONE flag on this channel
            */
            sc_fpend( acm1_info[ chan ].flags, TXFR_TIMEOUT,
                            TX_TXFR_DONE, 1, &err ) ;
            if (err == RET_OK)
                    sc_fclear( acm1_info[ chan ].flags, TX_TXFR_DONE, &err);
            *(p1->rdwr.act) = ctrl_blk->tx_curr_ptr ;
            break ;

        default :
            err = IFXENOTIMP ;
            break ;

        }

        return( err ) ;
}
```

ACM-1 Machine Code Routines for IFX Driver

```
        include ../../../include/ifxvisi.inc
        include ../../../include/vrtxvisi.inc
        include ../../../include/radstone/config.inc
        include ../../../include/radstone/acm-1.inc

        SECT    13
        XDEF    .aisx0
        XDEF    .aisx1
        XDEF    .aisx2
        XDEF    .aisx3
        XDEF    .aisx4
        XDEF    .aisx5
        XDEF    .aisx6
        XDEF    .aisx7

        XDEF    .aisr0
        XDEF    .aisr1
        XDEF    .aisr2
        XDEF    .aisr3
        XDEF    .aisr4
        XDEF    .aisr5
        XDEF    .aisr6
        XDEF    .aisr7

        XDEF    .aise0
        XDEF    .aise1
        XDEF    .aise2
        XDEF    .aise3
        XDEF    .aise4
        XDEF    .aise5
        XDEF    .aise6
        XDEF    .aise7

        XREF    .acm1_IFXSCB_addr
        XREF    .tx_over
        XREF    .acm1eint
        XREF    .rx_over

.aisx0:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$00,d0
        move.l  #ACM_RAM_BASE_A32+$040,a0
        bra.s   .aisxx
.aisx1:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$04,d0
        move.l  #ACM_RAM_BASE_A32+$080,a0
        bra.s   .aisxx
.aisx2:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$08,d0
        move.l  #ACM_RAM_BASE_A32+$0C0,a0
        bra.s   .aisxx
```

```
.aisx3:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$0C,d0
        move.l  #ACM_RAM_BASE_A32+$100,a0
        bra.s   .aisxx
.aisx4:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$10,d0
        move.l  #ACM_RAM_BASE_A32+$140,a0
        bra.s   .aisxx
.aisx5:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$14,d0
        move.l  #ACM_RAM_BASE_A32+$180,a0
        bra.s   .aisxx
.aisx6:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$18,d
        move.l  #ACM_RAM_BASE_A32+$1C0,a0
        bra.s   .aisxx
.aisx7:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$1C,d0
        move.l  #ACM_RAM_BASE_A32+$200,a0

.aisxx:
*       Must find current mode in order to respond correctly

        move.w  MODE(a0),d1
        cmp.w   #BUFF_TXFR,d1
        beq     txbuff
        cmp.w   #BUFF_TXFR_CONT,d1
        beq     txbuff

        move.l  a0,a2
        move.l  #.acm1_IFXSCB_addr,a1           ;/* each entry is 8 bytes */
        move.l  (a1,d0),a0                      ;/* A0 should point to SCB */
        move.l  a0,a1
        add.l   #IFXSCBtransmit_ready,a1
        move.l  (a1),a1
        jsr     (a1)
        tst.l   d0
        bne.s   txret

*       Only re-enable transmitter if in single character mode

        move.w  MODE(a2),d0
        cmp.w   #CHAR_TXFR,d0
        bne     txret1

        move.l  d1,TX_BUFF_ADDR(a2)             ;/* transmit next char */
        move.b  #ENABLE_TXFRS,TX_EN(a2)         ;/* re-enable transmitter */
        bra     txret
```

```
txret1: cmp.w   #CHAR_TXFR_CONT,d0
        bne     txbuff
        move.l  d1,TX_BUFF_ADDR(a2)             ;/* transmit next char */
        bra     txret

*       Only sc_post() to tx semaphore if in a buffered mode

txbuff: asr.l   #2,d0
        move.l  d0,-(sp)
        jsr     .tx_over
        add.l   #4,sp

txret:  movem.l (sp)+,A0-A7/D0-D7
        move.l  d0,-(sp)
        move.l  #UIFEXIT,d0
        trap    #VRTX_TRAP

.aise0:
        move.l  d0,-(sp)
        move.l  #0,d0
        bra.s   .aisee
.aise1:
        move.l  d0,-(sp)
        move.l  #1,d0
        bra.s   .aisee
.aise2:
        move.l  d0,-(sp)
        move.l  #2,d0
        bra.s   .aisee
.aise3:
        move.l  d0,-(sp)
        move.l  #3,d0
        bra.s   .aisee
.aise4:
        move.l  d0,-(sp)
        move.l  #4,d0
        bra.s   .aisee
.aise5:
        move.l  d0,-(sp)
        move.l  #5,d0
        bra.s   .aisee
.aise6:
        move.l  d0,-(sp)
        move.l  #6,d0
        bra.s   .aisee
.aise7:
        move.l  d0,-(sp)
        move.l  #7,d0

.aisee: movem.l A0-A7/D0-D7,-(sp)

        move.l  d0,-(sp)
        add.l   #4,sp
        jsr     .acmleint

        movem.l (sp)+,A0-A7/D0-D7
        add.l   #4,sp
        rte
```

```
.aisr0:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$00,d0
        move.l  #ACM_RAM_BASE_A32+$040,a0
        bra.s   .aisrx
.aisr1:
        movem.l A0-A7/D0-D7,-(sp
        move.l  #$04,d0
        move.l  #ACM_RAM_BASE_A32+$080,a0
        bra.s   .aisrx
.aisr2:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$08,d0
        move.l  #ACM_RAM_BASE_A32+$0C0,a0
        bra.s   .aisrx
.aisr3:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$0C,d0
        move.l  #ACM_RAM_BASE_A32+$100,a0
        bra.s   .aisrx
.aisr4:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$10,d0
        move.l  #ACM_RAM_BASE_A32+$140,a0
        bra.s   .aisrx
.aisr5:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$14,d0
        move.l  #ACM_RAM_BASE_A32+$180,a0
        bra.s   .aisrx
.aisr6:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$18,d0
        move.l  #ACM_RAM_BASE_A32+$1C0,a0
        bra.s   .aisrx
.aisr7:
        movem.l A0-A7/D0-D7,-(sp)
        move.l  #$1C,d0
        move.l  #ACM_RAM_BASE_A32+$200,a0

.aisrx:

*       Must find current mode in order to respond correctly

        move.w  MODE(a0),d1
        cmp.w   #BUFF_TXFR,d1
        beq     rxbuff
        cmp.w   #BUFF_TXFR_CONT,d1
        beq     rxbuff

        move.l  a0,a2
        move.l  #$7F,d1
        and.l   RX_BUFF_ADDR(a0),d1             ;/* received char in D1 */
        move.l  #.acm1_IFXSCB_addr,a1           ;/* each entry is 8 bytes */
        move.l  (a1,d0),a0                      ;/* A0 should point to SCB */
        move.l  (a0),a1
        jsr     (a1)
```

```
*       Only re-enable receiver if in single character mode

        move.w  MODE(a2),d0
        cmp.w   #CHAR_TXFR,d0
        bne     rxret1

        move.b  #ENABLE_TXFRS,RX_EN(a2)        ;/* re-enable transmitter */
        bra     rxret

rxret1: cmp.w   #CHAR_TXFR_CONT,d0
        beq     rxret
*       Only sc_post() to rx semaphore if in a buffered mode

rxbuff: asr.l   #2,d0
        move.l  d0,-(sp)
        jsr     .rx_over
        add.l   #4,sp

rxret:
        movem.l (sp)+,A0-A7/D0-D7
        move.l  d0,-(sp)
        move.l  #UIFEXIT,d0
        trap    #VRTX_TRAP
```

Copies of circuit diagrams for the PMV 68 ACM-1 may be obtained subject to a Confidentiality Agreement being made. For details please contact:

The Contracts Manager
Radstone Technology PLC
Water Lane
Towcester
Northants NN12 6JN

or

The General Manager
Radstone Technology Corporation
50 Craig Road
Montvale
New Jersey 07645-1709

Radstone Technology, world-wide:

UK

Radstone Technology PLC
Water Lane
Towcester
Northants NN12 6JN
Telephone: +44 (0)1327 359444
Fax: +44 (0)1327 358112
E-mail: sales@radstone.co.uk
support@radstone.co.uk

FRANCE

Radstone Technology
Miniparc
6, avenue des Andes
91952 COURTABOEUF Cedex
Téléphone: +33 (1) 01.64.46.04.03
Fax: +33 (1) 01.69.28.03.40

AUSTRALIA

Radstone Technology Pty Ltd.
PO Box 59/
1a, First Avenue
Eastwood, NSW 2122
Telephone: +61 2 9874 9143
Fax: +61 2 9874 9150

TAIWAN

Radstone Technology
Floor 4 Number 568
Chung Cheng Road
Hsintien
Taipei
Telephone: +886 2 218 5819 or 5829
Fax: +886 2 218 6248

USA

North East Region

Radstone Technology Corporation
50, Craig Road
Montvale
New Jersey 07645 - 1709
Telephone: (201) 391 2700
Fax: (201) 391 2899
E-mail: radstone@radstone.com

South East Region

10530, Warwick Avenue
Suite C2
Fairfax
Virginia, 22030
Telephone: (703) 383 3150
Fax: (703) 383 3156

North Central Region

421 N. Northwest Highway
Suite 201
Barrington
Illinois, 60010
Telephone: (847) 304 0202
Fax: (847) 304 0618

North West Region

3333 Bowers Avenue
Suite 130
Santa Clara
California, 95054
Telephone: (408) 727 4795 or 4796
Fax: (408) 727 4797

South West Region

3350 Shelby Street
Suite 200
Ontario
California, 91764
Telephone: (909) 944 2560
Fax: (909) 944 2568

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP