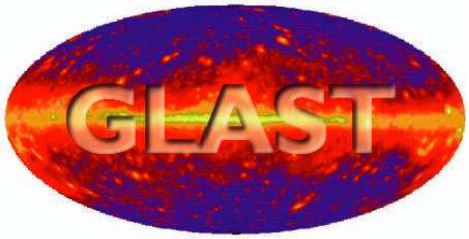


LAT Flight Software

MEM Package User Manual

Type:	User Manual
Version:	V5-2-0
Author:	Don May
Created:	21 February 2005
Updated:	2 May 2005
Printed:	2 May 2005

A description of the operation and usage of the LAT flight software Memory Dump and Memory Load functions found within the MEM package.

Contents

Tables

0 Introduction

The MEM package contains functions that handle memory dump and memory load operations.

0.0 Memory Dump and Load

The MEM package contains functions to dump and load various types of memory within the SIU and EPU CPUs. These functions handle the LMEMDUMP and LMEMLOAD telecommands and return dump data in a series of telemetry packets.

The following types of memory can be dumped and loaded by the MEM package:

- RAM
- PCI device headers
- RAD750 CPU internal registers
- Power PCI bridge chip registers
- Locations within the PCI address space, including SIB EEPROM, LCB registers, and 1553 interface shared RAM and registers.

The following types of read-only memory and status information can be dumped by the MEM package, but not loaded:

- RAD750 SUROM
- Memory pool statistics
- Values of symbols within the VxWorks system symbol table

0.1 Low-Level Facilities

In addition to handling memory accesses at the application level, the MEM package provides low-level dump, load, and memory map functions. These functions are used not only by the MEM telecommand handlers but also by the CPU boot code.

Using these low-level functions, the CPU boot code handles a subset of the LMEMDUMP and LMEMLOAD telecommands. All dump data is returned by the boot code in its Boot Housekeeping telemetry packets.

1 Memory Dump

The MEM package provides handlers for the LMEMDUMP telecommands, which are summarized in Table 1, below. These commands respond by sending data in a series of telemetry packets, which are summarized in Table 2.

Table 1 – Memory Dump Telecommands (APID 1604)

Function Code	Mnemonic	Description
0	LMEMDUMPMEM	Memory Data Dump – Dump the contents of a region within the CPU address space.
1	LMEMDUMPCANCEL	Memory Dump Cancel – Cancel a memory dump in progress.
2	LMEMDUMPPCI	PCI Device Header Dump – Dump the contents of a PCI device header.
3	LMEMDUMPREG	Processor Register Dump – Dump the contents of the CPU's internal registers.
7	LMEMDUMPPOOL	Dump Pool Statistics – Dump statistics about a VxWorks memory pool.
8	LMEMDUMPSYMVAL	Memory Symbol Lookup – Dump the value of a symbol within the VxWorks system symbol table.
9	LMEMDUMPSYMREL	Memory Dump Symbol Relative – Dump a region of memory relative to the value of a symbol within the VxWorks system symbol table.

Table 2 – Memory Dump Telemetry

APID	Mnemonic	Description
785	LMEMPOOLDATA	Memory Pool Statistics Dump – Statistics for a memory pool.
786	LMEMSYMVAL	Symbol Value Dump – Value of a symbol from the VxWorks system symbol table.
788	LMEMSIUDATA	SIU Memory Dump Data – Data from the SIU CPU.
789	LMEMEPU0DATA	EPU 0 Memory Dump Data – Data from the EPU 0 CPU.
790	LMEMEPU1DATA	EPU 1 Memory Dump Data – Data from the EPU 1 CPU.
791	LMEMEPU2DATA	EPU 2 Memory Dump Data – Data from the EPU 2 CPU.

1.0 Memory Data Dump (LMEMDUMPMEM)

The Memory Data Dump (LMEMDUMPMEM) telecommand is used to dump a region of a CPU's address space. Any type of memory that is mapped directly to the CPU address space can be dumped with this command. This includes RAM, RAD750 SUROM, SIB EEPROM, Power PCI bridge chip registers, 1553 interface registers and shared RAM, and LCB registers.

Table 3 – Memory Data Dump (LMEMDUMPMEM) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.
2	16	Pad	Padding to align the next field to a 32-bit boundary.
4	32	Address	Starting address of the memory region to dump.
8	32	Size	Number of 32-bit words to dump.

In response to this telecommand, each CPU uses a different telemetry packet to return the dumped data, as follows:

- SIU – LMEMSIUDATA
- EPU 0 – LMEMEPU0DATA
- EPU 1 – LMEMEPU1DATA
- EPU 2 – LMEMEPU2DATA

Within the SCP environment, the **MEM_sendDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- start: the 'Address' value.
- count: the 'Size' value.

The MEM_sendDump() function always sets the 'Transaction ID' value to 0x135.

1.1 Memory Dump Cancel (LMEMDUMPCANCEL)

The Memory Dump Cancel (LMEMDUMPCANCEL) telecommand is used to cancel a dump that is in progress. If a memory dump is in progress, this telecommand must be used to stop the dump before the CPU will accept another LMEMDUMPMEM, LMEMDUMPPCI, LMEMDUMPREG, or LMEMDUMPSYMREL telecommand.

Table 4 – Memory Dump Cancel (LMEMDUMPCANCEL) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is unused by the CPU.

Within the SCP environment, the **MEM_sendCancel()** function can be used to send this telecommand. This function requires the following parameter:

- unit: the 'LAT unit' value.

The MEM_sendCancel() function always sets the 'Transaction ID' value to 0x246.

1.2 PCI Device Header Dump (LMEMDUMPPCI)

The PCI Device Header Dump (LMEMDUMPPCI) telecommand is used to dump an entire 256-byte configuration header from a device on the PCI bus.

Table 5 – PCI Device Header Dump (LMEMDUMPPCI) Telecommand Format

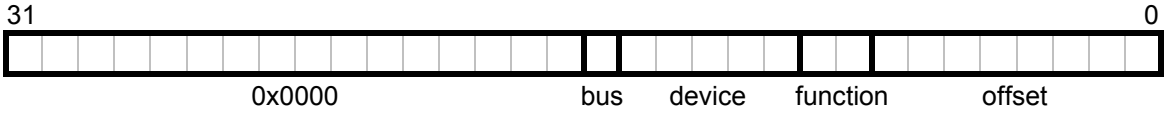
Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.
2	1	Bus	PCI 'Bus' on which the device is located,
	5	Device	PCI 'Device' ID of the device.
	2	Function	PCI 'Function' ID of the configuration header within the device.
3	8	Pad	Padding to align the end of the telecommand to a 16-bit boundary.

In response to this telecommand, each CPU uses a different telemetry packet to return the dumped data, as follows:

- SIU – LMEMSIUDATA
- EPU 0 – LMEMEPU0DATA
- EPU 1 – LMEMEPU1DATA
- EPU 2 – LMEMEPU2DATA

When one of these telemetry packets contains PCI device header data, the address value specified in the telemetry packet contains the bus, device, and function values for the configuration header, along with the offset within the header of the first word of packet data. The format of this address is shown in Figure 1.

Figure 1 – PCI Configuration Header Address



Within the SCP environment, the **MEM_sendPciDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- tran: the 'Transaction ID' value.
- bus: the 'Bus' value.
- device: the 'Device' value.
- function: the 'Function' value.

1.3 Processor Register Dump (LMEMDUMPREG)

The Processor Register Dump (LMEMDUMPREG) telecommand is used to dump the contents of the RAD750 CPU's internal registers. When the CPU receives this telecommand, it saves a snapshot of its internal register values in a 'register buffer', then sends the data within this 'register buffer' as the dump data.

Table 6 – Processor Register Dump (LMEMDUMPREG) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.

In response to this telecommand, each CPU uses a different telemetry packet to return the dumped data, as follows:

- SIU – LMEMSIUDATA
- EPU 0 – LMEMEPU0DATA
- EPU 1 – LMEMEPU1DATA
- EPU 2 – LMEMEPU2DATA

When one of these telemetry packets contains processor register data, the address value specified in the telemetry packet contains the offset within the 'register buffer' of the first word of packet data. The offsets of the CPU's internal registers within the 'register buffer' are shown in Table 7.

Within the SCP environment, the **MEM_sendRegDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- tran: the 'Transaction ID' value.

Table 7 – CPU 'Register Buffer' Offsets

Offset	Register	Offset	Register	Offset	Register	Offset	Register
0x000	GPR 0	0x0b0	IBAT 2U	0x160	FPR 1L	0x210	FPR 23L
0x004	GPR 1	0x0b4	IBAT 2L	0x164	FPR 2U	0x214	FPR 24U
0x008	GPR 2	0x0b8	IBAT 3U	0x168	FPR 2L	0x218	FPR 24L
0x00c	GPR 3	0x0bc	IBAT 3L	0x16c	FPR 3U	0x21c	FPR 25U
0x010	GPR 4	0x0c0	DBAT 0U	0x170	FPR 3L	0x220	FPR 25L
0x014	GPR 5	0x0c4	DBAT 0L	0x174	FPR 4U	0x224	FPR 26U
0x018	GPR 6	0x0c8	DBAT 1U	0x178	FPR 4L	0x228	FPR 26L
0x01c	GPR 7	0x0cc	DBAT 1L	0x17c	FPR 5U	0x22c	FPR 27U
0x020	GPR 8	0x0d0	DBAT 2U	0x180	FPR 5L	0x230	FPR 27L
0x024	GPR 9	0x0d4	DBAT 2L	0x184	FPR 6U	0x234	FPR 28U
0x028	GPR 10	0x0d8	DBAT 3U	0x188	FPR 6L	0x238	FPR 28L
0x02c	GPR 11	0x0dc	DBAT 3L	0x18c	FPR 7U	0x23c	FPR 29U
0x030	GPR 12	0x0e0	SDR1	0x190	FPR 7L	0x240	FPR 29L
0x034	GPR 13	0x0e4	PVR	0x194	FPR 8U	0x244	FPR 30U
0x038	GPR 14	0x0e8	DAR	0x198	FPR 8L	0x248	FPR 30L
0x03c	GPR 15	0x0ec	SPRG 0	0x19c	FPR 9U	0x24c	FPR 31U
0x040	GPR 16	0x0f0	SPRG 1	0x1a0	FPR 9L	0x250	FPR 31L
0x044	GPR 17	0x0f4	SPRG 2	0x1a4	FPR 10U	0x254	HID 0
0x048	GPR 18	0x0f8	SPRG 3	0x1a8	FPR 10L	0x258	HID 1
0x04c	GPR 19	0x0fc	DSISR	0x1ac	FPR 11U	0x25c	MMCR 0
0x050	GPR 20	0x100	SRR 0	0x1b0	FPR 11L	0x260	MMCR 1
0x054	GPR 21	0x104	SRR 1	0x1b4	FPR 12U	0x264	PMC 1
0x058	GPR 22	0x108	DEC	0x1b8	FPR 12L	0x268	PMC 2
0x05c	GPR 23	0x10c	DABR	0x1bc	FPR 13U	0x26c	PMC 3
0x060	GPR 24	0x110	EAR	0x1c0	FPR 13L	0x270	PMC 4
0x064	GPR 25	0x114	SR 0	0x1c4	FPR 14U	0x274	SIA
0x068	GPR 26	0x118	SR 1	0x1c8	FPR 14L	0x278	IABR
0x06c	GPR 27	0x11c	SR 2	0x1cc	FPR 15U	0x27c	L2CR
0x070	GPR 28	0x120	SR 3	0x1d0	FPR 15L	0x280	ICTC
0x074	GPR 29	0x124	SR 4	0x1d4	FPR 16U	0x284	THRM 1
0x078	GPR 30	0x128	SR 5	0x1d8	FPR 16L	0x288	THRM 2
0x07c	GPR 31	0x12c	SR 6	0x1dc	FPR 17U	0x28c	THRM 3
0x080	CR	0x130	SR 7	0x1e0	FPR 17L	0x290 - 0x2fc	<unused>
0x084	FPSCR	0x134	SR 8	0x1e4	FPR 18U		
0x088	XER	0x138	SR 9	0x1e8	FPR 18L		
0x08c	LR	0x13c	SR 10	0x1ec	FPR 19U		
0x090	CTR	0x140	SR 11	0x1f0	FPR 19L		
0x094	TBL	0x144	SR 12	0x1f4	FPR 20U		
0x098	TBU	0x148	SR 13	0x1f8	FPR 20L		
0x09c	MSR	0x14c	SR 14	0x1fc	FPR 21U		
0x0a0	IBAT 0U	0x150	SR 15	0x200	FPR 21L		
0x0a4	IBAT 0L	0x154	FPR 0U	0x204	FPR 22U		
0x0a8	IBAT 1U	0x158	FPR 0L	0x208	FPR 22L		
0x0ac	IBAT 1L	0x15c	FPR 1U	0x20c	FPR 23U		

1.4 Dump Pool Statistics (LMEMDUMPPOOL)

The Dump Pool Statistics (LMEMDUMPPOOL) telecommand is used to retrieve information about the VxWorks memory pools.

Table 8 – Dump Pool Statistics (LMEMDUMPPOOL) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.
2	16	Pool ID	ID of the memory pool to query. 0 = system memory pool 1 = application memory pool

In response to this telecommand, the CPU sends the LMEMPOOLDATA telemetry packet.

Within the SCP environment, the **MEM_sendPoolDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the ‘LAT unit’ value.
- tran: the ‘Transaction ID’ value.
- pool: the “Pool ID’ value.

1.5 Memory Symbol Lookup (LMEMDUMPSYMVAL)

The Memory Symbol Lookup (LMEMDUMPSYMVAL) telecommand is used to lookup the value of a symbol within the VxWorks system symbol table.

Table 9 – Memory Symbol Lookup (LMEMDUMPSYMVAL) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.
2	8	Name Size	Length, in bytes, of the name of the symbol.
3	8	Pad	Padding to align the next field to a 32-bit boundary.
4	384 (48x8)	Name String	Symbol name character string. The Name Size field specifies the number of valid characters in this field (up to 48 characters).

In response to this telecommand, the CPU sends the LMEMSYMVAL telemetry packet.

Within the SCP environment, the **MEM_sendSymValDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the ‘LAT unit’ value.
- name: the ‘Name String’ value.

The MEM_sendSymValDump() function calculates the length of the ‘name’ parameter and sets the ‘Name Size’ value appropriately. If the ‘name’ parameter is longer than 48 characters, it is truncated to 48 characters. This function always sets the ‘Transaction ID’ value to 0x123.

1.6 Memory Dump Symbol Relative (LMEMDUMPSYMREL)

The Memory Dump Symbol Relative (LMEMDUMPSYMREL) telecommand is used to dump a region of a CPU's address space, relative to the value of a symbol in the VxWorks system symbol table. This could be used, for example, to dump the contents of a named structure when the structure's address is not known. (Alternatively, the LMEMDUMPSYMVAL telecommand could be used to get the address of the structure, and then the LMEMDUMPMEM telecommand could be used to dump the contents of the structure.)

Table 10 – Memory Dump Symbol Relative (LMEMDUMPSYMREL) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is echoed in the telemetry packets containing the dumped data.
2	16	Pad	Padding to align the next field to a 32-bit boundary.
4	32	Offset	Starting offset of the memory region to dump, relative to the value of the symbol.
8	32	Size	Number of 32-bit words to dump.
12	8	Name Size	Length, in bytes, of the name of the symbol.
13	24	Pad	Padding to align the next field to a 32-bit boundary.
16	288 (36x8)	Name String	Symbol name character string. The Name Size field specifies the number of valid characters in this field (up to 36 characters).

In response to this telecommand, each CPU uses a different telemetry packet to return the dumped data, as follows:

- SIU – LMEMSIUDATA
- EPU 0 – LMEMEPU0DATA
- EPU 1 – LMEMEPU1DATA
- EPU 2 – LMEMEPU2DATA

Within the SCP environment, the **MEM_sendSymRelDump()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- tran: the 'Transaction ID' value.
- offset: the 'Offset' value.
- count: the 'Size' value.
- name: the 'Name String' value.

The MEM_sendSymRelDump() function calculates the length of the 'name' parameter and sets the 'Name Size' value appropriately. If the 'name' parameter is longer than 36 characters, it is truncated to 36 characters.

1.7 SIU Memory Dump Data (LMEMSIUDATA)

The SIU Memory Dump Data (LMEMSIUDATA) telemetry packet contains data dumped from the SIU CPU. This telemetry packet is sent in response to the LMEMDUMPMEM, LMEMDUMPPCI, LMEMDUMPREG, and LMEMDUMPSYMREL telecommands.

Table 11 – SIU Memory Dump Data (LMEMSIUDATA) Telemetry Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that sent this telemetry packet (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	Transaction ID value from the telecommand that requested the dump. This can be used to match dump requests to the resulting dumped data.
2	32	Address	Address or offset of the first word of data in this telemetry packet. • For PCI device configuration header data, this address has the format shown in Figure 1. • For CPU internal register data, this address is the offset into the 'Register Buffer' shown in Table 7. • For Symbol-Relative memory dumps, this address is the absolute address of the first word of data – not the relative offset from the symbol value.
6	16	Word Count	Number of valid 32-bit words of data in this telemetry packet.
8	16	Function Code	Function code from the telecommand that requested the dump. This can be used to determine the type of data found within this telemetry packet.
10	2880 (90x32)	Data	Dumped data (up to 90, 32-bit words).

1.8 EPU x Memory Dump Data (LMEMEPUxDATA)

The EPU x Memory Dump Data (LMEMEPUxDATA) telemetry packets contain data dumped from the EPU CPUs. These telemetry packets are sent in response to the LMEMDUMPMEM, LMEMDUMPPCI, LMEMDUMPREG, and LMEMDUMPSYMREL telecommands. They have the same format as the LMEMSIUDATA telemetry packet, as shown in Table 11. The only difference is that they use different APID values, as shown in Table 2.

1.9 Memory Pool Statistics Dump (LMEMPOOLDATA)

The Memory Pool Statistics Dump (LMEMPOOLDATA) telemetry packet contains information about a VxWorks memory pool. This telemetry packet is sent in response to the LMEMDUMPPOOL telecommand.

Table 12 – Memory Pool Statistics Dump (LMEMPOOLDATA) Telemetry Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that sent this telemetry packet (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	Transaction ID value from the telecommand that requested the pool statistics. This can be used to match requests to the resulting pool statistics.
2	16	Pool ID	ID of the memory pool from which the statistics were gathered. 0 = system memory pool 1 = application memory pool
4	16	Pad	Padding to align the next field to the proper boundary.
6	32	Free Bytes	Number of bytes available within the pool.
10	32	Free Blocks	Number of blocks available within the pool.
14	32	Max Block Bytes	Size, in bytes, of the largest available block.
18	32	Allocated Bytes	Number of bytes allocated within the pool.
22	32	Allocated Blocks	Number of blocks allocated within the pool.

1.10 Symbol Value Dump (LMEMSYMVAL)

The Symbol Value Dump (LMEMSYMVAL) telemetry packet contains the value of a symbol within the VxWorks system symbol table. This telemetry packet is sent in response to the LMEMDUMPSYMVAL telecommand.

Table 13 – Symbol Value Dump (LMEMSYMVAL) Telemetry Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that sent this telemetry packet (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	Transaction ID value from the telecommand that requested the symbol value. This can be used to match requests to the resulting symbol value.
2	32	Symbol Value	Value of the symbol.
6	8	Name Size	Length, in bytes, of the name of the symbol.
7	24	Pad	Padding to align the next field to the proper boundary.
10	384 (48x8)	Name String	Symbol name character string. The Name Size field specifies the number of valid characters in this field (up to 48 characters).

2 Memory Load

The MEM package provides functions that handle the LMEMLOAD telecommands, which are summarized in Table 14, below.

Table 14 – Memory Load Telecommands (APID 1604)

Function Code	Mnemonic	Description
4	LMEMLOADMEM	Memory Write – Load the contents of a region within the CPU address space.
5	LMEMLOADPCI	PCI Device Header Write – Load the contents of a single 16-bit value within a PCI device header.
6	LMEMLOADREG	Processor Register Write – Load the contents of one or more of the CPU's internal registers.

2.0 Memory Write (LMEMLOADMEM)

The Memory Write (LMEMLOADMEM) telecommand is used to write to a region of a CPU's address space. Any type of write-enabled memory that is mapped directly to the CPU address space can be written with this command. This includes RAM, SIB EEPROM, Power PCI bridge chip registers, 1553 interface registers and shared RAM, and LCB registers.

Table 15 – Memory Write (LMEMLOADMEM) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is unused by the CPU.
2	16	Size	Number of 32-bit words to load.
4	32	Address	Starting address of the memory region to load.
8	352 (11x32)	Data	Data to load (up to 11, 32-bit words).

Within the SCP environment, the **MEM_sendLoad()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- address: the 'Address' value.
- value: the first (and only) 32-bit 'Data' value.

The MEM_sendLoad() function always sets the 'Transaction ID' value to 0xabc, the 'Size' value to 1, and the unused 'Data' values to 0. Since the 'Size' value is always set to 1, the MEM_sendLoad() function can't be used to load more (or less) than 4 bytes of data.

2.1 PCI Device Header Write (LMEMLOADPCI)

The PCI Device Header Write (LMEMLOADPCI) telecommand is used to write a single 16-bit value to a location within the configuration header of a device on the PCI bus.

Table 16 – PCI Device Header Write (LMEMLOADPCI) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is unused by the CPU.
2	1	Bus	PCI 'Bus' on which the device is located,
	5	Device	PCI 'Device' ID of the device.
	2	Function	PCI 'Function' ID of the configuration header within the device.
3	8	Offset	Byte offset of the location to load within the configuration header.
4	16	Data	16-bit value to load.

Within the SCP environment, the **MEM_sendPciLoad()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- bus: the 'Bus' value.
- device: the 'Device' value.
- function: the 'Function' value.
- offset: the 'Offset' value.
- value: the 'Data' value.

The MEM_sendPciLoad() function always sets the 'Transaction ID' value to 0x8ac.

2.2 Processor Register Write (LMEMLOADREG)

The Processor Register Write (LMEMLOADREG) telecommand is used to write to the CPU's internal registers. To identify these registers, they are mapped to locations within an imaginary 'Register Buffer', as described in Table 7. For this telecommand, the 'address' of a particular register is its offset within this 'Register Buffer'. When this telecommand is received, the CPU translates the offset value to determine which internal registers to load.

Table 17 – Processor Register Write (LMEMLOADREG) Telecommand Format

Byte Offset	Length (Bits)	Field Name	Description
0	4	LAT Unit	ID of the CPU that should execute the command (SIU, EPU 0, etc.), as defined by ITC.
	12	Transaction ID	User-defined value that is unused by the CPU.
2	16	Size	Number of 32-bit words to load.
4	32	Offset	Starting offset to load within the 'Register Buffer'.
8	352 (11x32)	Data	Data to load (up to 11, 32-bit words).

Within the SCP environment, the **MEM_sendRegLoad()** function can be used to send this telecommand. This function requires the following parameters:

- unit: the 'LAT unit' value.
- offset: the 'Offset' value.
- value: the first (and only) 32-bit 'Data' value.

The MEM_sendRegLoad() function always sets the 'Transaction ID' value to 0xdef, the 'Size' value to 1, and the unused 'Data' values to 0. Since the 'Size' value is always set to 1, the MEM_sendRegLoad() function can't be used to load more (or less) than 4 bytes of data into the CPU's internal registers.

3 Control and Status

At the application level, the MEM package provides functions that control the operation of the MEM command handlers and report their status. These functions act as the interface to the MEM package.

3.0 MEM_initialize()

The MEM_initialize() function allocates resources used by the MEM package and sets configuration values to their default state. It accepts two parameters:

- a pointer to the task description block of the task to which the MEM command handlers should be attached
- the ID of the task to which the MEM command handlers should be attached.

These parameters are redundant, in that they provide two methods for specifying the MEM command handler task. If the first parameter is non-NULL, it will be used to attach the MEM command handlers to the corresponding task (and the second parameter will be ignored). If the first parameter is NULL, however, then MEM_initialize() will call an ITC function to translate the second parameter into a task description block pointer, which it will then use to attach the MEM command handlers. Finally, if the first parameter is NULL and the second parameter is not a valid task ID (e.g. -1), then MEM_initialize() will not attach the MEM command handlers to any task. In the latter case, it is assumed that some other application-level code will perform this attachment.

To configure the MEM package, MEM_initialize() performs the following operations:

- establishes memory map entries for the following regions of the address space:
    - Power PCI bridge chip registers
    - RAD750 SUROM (read-only)
    - RAM
    - SIB registers
    - SIB shared RAM
    - SIB EEPROM
- configures the memory load facility
- configures the memory dump facility
- attaches handlers for the MEM dump and load telecommands (if a valid task is identified by the MEM_initialize() parameters)

MEM_initialize() returns a MSG status code.

3.1 MEM_shutdown()

The MEM_shutdown() function frees resources allocated by the MEM package. It requires no parameters, and it returns a MSG status code.

3.2 MEM_getInfo()

The MEM_getInfo() function returns information about the state of the MEM package. It accepts a single parameter, which is a pointer to a MEM_Info buffer into which it should store the requested information. The types of information stored in a MEM_Info buffer include:

- general MEM status
    - status of the most recent MEM telecommand
    - function code of the most recent MEM telecommand
- MEM load status
    - status of the most recent load telecommand
    - indication that a load operation is in progress
    - starting address of most recent load telecommand
    - number of bytes requested by most recent load telecommand
    - current offset of load operation, if one is in progress
- MEM dump status
    - status of the most recent dump telecommand
    - indication that a dump operation is in progress
    - starting address of most recent dump telecommand
    - number of bytes requested by most recent dump telecommand
    - current address of dump operation, if one is in progress
    - function code of the most recent dump telecommand
    - transaction ID of the most recent dump telecommand

MEM_getInfo() returns a MSG status code.

4 Low-Level Facilities

The MEM package implements the memory load and store functionality within low-level facilities. These facilities are used internally by the MEM telecommand handlers and externally by the Primary Boot Code.

4.0 Memory Dump Facility

The memory dump facility provides the core functionality for dumping regions of memory. To use this facility, a caller must establish a dump context to hold the state information needed to perform a dump operation. A dump context is an opaque handle for a structure that the dump facility maintains during dump operations. Once a dump operation has completed, a dump context can be used again for another dump operation.

Table 18 is a summary of the functions provided by the memory dump facility. The MEM_dump.h header file defines the interface to these functions.

Table 18 – Memory Dump Facility Functions

Function	Description
MEM_dumpInit()	Initializes the memory dump facility. This function must be called once, before calling any other dump facility functions.
MEM_dumpGetContext()	Allocates a context for dump operations.
MEM_dumpReleaseContext()	Frees a dump context.
MEM_dumpGetInfo()	Returns information about the state of a dump operation. This information is a subset of the information stored within a dump context.
MEM_dumpStart()	Starts a dump operation. In addition to providing a dump context handle for this function, the caller must also provide a pointer to a MEM_Descriptor that specifies the type of memory to be dumped (memory, PCI device configuration header, or CPU internal register), the address or offset at which the dump operation should start, and the amount of data to dump.
MEM_dumpGetData()	Reads a ‘chunk’ of data for a dump operation. Typically, a dump operation requires multiple calls to this function, with each call performing a portion of the operation.
MEM_CancelDump()	Cancels a dump operation, which allows a dump context to be used for another dump operation.

4.1 Memory Load Facility

The memory load facility provides the core functionality for writing regions of memory. To use this facility, a caller must establish a load context to hold the state information needed to perform a load operation. A load context is an opaque handle for a structure that the load facility maintains during load operations. Once a load operation has completed, a load context can be used again for another load operation.

Table 19 is a summary of the functions provided by the memory load facility. The MEM_load.h header file defines the interface to these functions.

Table 19 – Memory Load Facility Functions

Function	Description
MEM_loadInit()	Initializes the memory load facility. This function must be called once, before calling any other load facility functions.
MEM_loadGetContext()	Allocates a context for load operations.
MEM_loadReleaseContext()	Frees a load context.
MEM_loadGetInfo()	Returns information about the state of a load operation. This information is a subset of the information stored within a load context.
MEM_loadPktMemory()	Starts a memory load operation. In addition to providing a load context handle for this function, the caller must also provide a pointer to an LMEMLOADMEM telecommand packet that specifies the address at which the load operation should start, the amount of data to load, and the data values to load.
MEM_loadPktPci()	Starts a PCI device header load operation. In addition to providing a load context handle for this function, the caller must also provide a pointer to an LMEMLOADPCI telecommand packet that specifies the bus, device, function, and offset at which the load operation should start and the 16-bit data value to load.
MEM_loadPktReg()	Starts a processor register load operation. In addition to providing a load context handle for this function, the caller must also provide a pointer to an LMEMLOADREG telecommand packet that specifies the register ‘offset’ at which the load operation should start, the amount of data to load, and the data values to load.
MEM_loadWrite()	Writes a ‘chunk’ of data for a load operation. Typically, a load operation requires only a single call to this function, since the amount of data loaded tends to be small.

4.2 Memory Map Facility

The memory map facility provides the core functionality for accessing memory. It contains the lowest-level functions that read and write various types of memory, the PCI device configuration headers, and the CPU’s internal registers. It also provides a mechanism for creating memory map entries that define regions of the CPU’s address space. These definitions include information

about a region, such as its starting address, length, access restrictions, and callback functions to use when accessing it.

Table 20 is a summary of the functions provided by the memory map facility. The MEM_map.h header file defines the interface to these functions.

Table 20 – Memory Map Facility Functions

Function	Description
MEM_mapInit()	Initializes the memory map facility. This function must be called once, before calling any other map facility functions.
MEM_mapAddEntry()	Adds a memory map entry that defines a region of the CPU address space. Parameters to this function include the starting address of the region, the length of the region, the region's access restrictions, callback functions for reading and writing memory within the region, and a context parameter to pass to the callback functions.
MEM_mapAddPpciEntries()	Convenience function that adds memory map entries for the Power PCI bridge chip registers.
MEM_mapRWGeneric ()	Callback function for reading and writing memory that has no alignment or access restrictions, such as RAM.
MEM_mapRead32Swap()	Callback function for reading 32-bit words that require byte swapping, such as the Power PCI bridge chip registers.
MEM_mapWrite32Swap()	Callback function for writing 32-bit words that require byte swapping, such as the Power PCI bridge chip registers.
MEM_mapReadBuffer()	Callback function for reading data from a buffer. This differs from MEM_mapRWGeneric() in that addresses passed to this function are interpreted as offsets from the start of a buffer (instead of absolute addresses). This callback function gets the starting address of the buffer from the context parameter specified during the call to MEM_mapAddEntry().
MEM_mapWriteBuffer()	Callback function for writing data to a buffer. This differs from MEM_mapRWGeneric() in that addresses passed to this function are interpreted as offsets from the start of a buffer (instead of absolute addresses). This callback function gets the starting address of the buffer from the context parameter specified during the call to MEM_mapAddEntry().

4.3 Usage by Primary Boot Code

The Primary Boot Code (PBC) allocates a single dump context and a single load context to handle the LMEMDUMP and LMEMLOAD telecommands. It also establishes memory map entries for the following regions of the CPU address space:

- RAM
- RAD750 SUROM (read-only)
- Power PCI bridge chip registers
- SIB registers
- SIB shared RAM

- SIB EEPROM

The PBC uses MEM_dumpStart() to start a memory dump operation whenever it receives an LMEMDUMPMEM, LMEMDUMPPCI, or LMEMDUMPREG telecommand. Then each time it builds a Boot Housekeeping telemetry packet, PBC calls MEM_dumpGetData() to get the next 'chunk' of the dump data for inclusion in the telemetry packet. If PBC receives an LMEMDUMPCANCEL telecommand, it calls MEM_CancelDump() to stop the dump operation.

For LMEMLOADMEM, LMEMLOADPCI, and LMEMLOADREG telecommands, the PBC uses the MEM_loadPktMemory(), MEM_loadPktPci(), or MEM_loadPktReg() function to start a memory load operation. Each time through its command loop, PBC then calls MEM_loadWrite() to write the next 'chunk' of load data to the target locations.

4.4 Usage by MEM Application Code

The MEM_initialize() function allocates a single dump context and a single load context to handle LMEMDUMP and LMEMLOAD telecommands. This function also uses the memory map facility to establish memory map entries for the regions of address space described in section 3.0.

When an LMEMDUMPMEM, LMEMDUMPPCI, LMEMDUMPREG, or LMEMDUMPSYMREL telecommand is received, the command handlers use MEM_dumpStart() to start a dump operation using the context allocated by MEM_initialize(). At a rate of four times per second, the MEM package then uses the MEM_dumpGetData() function to get the next 'chunk' of dump data and send it in a telemetry packet. If an LMEMDUMPCANCEL telecommand is received, its handler calls MEM_CancelDump() to stop the dump operation.

When an LMEMLOADMEM, LMEMLOADPCI, or LMEMLOADREG telecommand is received, the command handlers use the MEM_loadPktMemory(), MEM_loadPktPci(), or MEM_loadPktReg() function to start a memory load operation with the context allocated by MEM_initialize(). The command handlers then spin in a loop calling MEM_loadWrite() until all the data is loaded. (An error is reported if the data has not been loaded within 32 passes through the spin-loop.)

The LMEMDUMPPOOL and LMEMDUMPSYMVAL telecommand handlers do not use the low-level MEM facilities. Instead, they use VxWorks functions to retrieve the requested information.

5 MEM Modules

The functions within the MEM package are organized into several VxWorks modules, as shown in the following table.

Table 21 – MEM Package Modules

Module	Description
mem_base	Low-level facilities.
mem	Application-level control and status functions, as well as the telecommand handlers.
mem_scp	Functions for use in the SCP environment, which send telecommands and display telecommands and telemetry packets.

6 Error Reporting

The MEM package functions support the MSG status reporting system. The following message codes are defined.

Table 22 – MEM Package MSG Codes – Success and Information

MSG Code	Description	Parameters
SUCCESS	Function succeeded.	
DMPCANC	Information message indicating that a dump was cancelled.	
DMPEND	Information message indicating that a dump completed.	
DMPSTART	Information message indicating that a dump was started.	• Length of the dump, in words • Starting address/offset of the dump

Table 23 – MEM Package MSG Codes – Errors

ALLOCFPA	Unable to allocate from a packet from an FPA.	String describing the FPA
ALLOCMEM	Unable to allocate memory.	• Size of allocation request, in bytes • Description string of the allocated region
ALLOCMTS	Unable to allocate a mutex.	
BADACC	The starting address of a memory dump request was not aligned to a 32-bit boundary, a memory dump request was for a region of memory that did not allow read access, or a memory load request was for a region of memory that did not allow writing.	
BADATACH	Unable to attach command handlers using ITC_attachApid().	
BADFUNC	A telecommand was received with an unrecognized function code.	Function code from telecommand
BADMAP	Internal software error.	

BADPRAM	An invalid parameter was passed to a low-level facility function.	Name string of the invalid parameter
BADSTATE	The MEM application code is not in the correct state for the requested operation.	• Current state • Expected state
CSDSFUNC	Unable to extract the function code from a telecommand packet.	MSG status code from CCSDS_pktHdrGetFuncCode()
CSDSHDR	Unable to create a CCSDS packet header.	MSG status code from CCSDS_pktHdrCreate()
CSDSSUM	Unable to insert a checksum into a CCSDS packet.	MSG status code from CCSDS_pktChksumInsert()
EBADBIND	Unable to bind a queue item with ITC_bind().	MSG status code from ITC_bind()
EBADDCTX	An invalid dump context handle was passed to one of the low-level dump facility functions.	
EBADDUMP	The MEM_dumpGetData() function was unable to read the data to be dumped.	
EBADLCTX	An invalid load context handle was passed to one of the low-level load facility functions.	
EBADLOAD	The MEM_loadWrite() function was unable to write the data to be loaded.	
EBADMAP	The callback function for an allowed type of access (read or write) was NULL when MEM_mapAddEntry() was called.	
EBADPOOL	Invalid Pool ID value.	The invalid Pool ID value
EBADSEND	Unable to send a packet with ITC_send().	MSG status code from ITC_send()
ECNTBAD	The number of words for a memory load request is too large.	• Maximum number of words that can be loaded • Requested number of words to load
ECNTLONG	The number of words for a memory load request is larger than the amount of data in the request packet.	• Number of words of data in the request packet • Requested number of words to load
EDPALLOC	Unable to allocate a dump context.	
EDPSTAT	Unable to start a memory dump operation because a dump operation in progress.	
ELDALLOC	Unable to allocate a load context.	
ELDSTAT	Unable to start a memory load operation because a load operation is in progress.	

ELDTMO	An LMEMLOAD telecommand handler was unable to write the data to be loaded after trying many times.	Number of attempts to write the data
ELOADTO	The MEM_loadWrite() function was unable to write the data to be loaded after trying 1000 times.	
EPKTSHRT	A telecommand packet was too short.	• Minimum required length of the packet, in bytes • Actual length of the packet, in bytes
ESTATS	Unable to get statistics about a memory pool.	Pool ID value
ESYMLKUP	Symbol not found.	• Symbol name string • Status value from the VxWorks symFindByName() function
ETBLFULL	The memory map table is full, i.e. no more memory map entries can be added with MEM_mapAddEntry().	
FREEMEM	Unable to free memory.	
FREEMTX	Unable to free a mutex.	
GENERROR	Generic error within an SCP function.	String describing the error
INUSEMEM	Unable to free memory because it is in use.	String describing the memory
LOCKMTX	Unable to lock a mutex.	
NOTRDY	The MEM package application functions have not been initialized by MEM_initialize().	
NULLPTR	A MEM function pointer parameter was NULL.	
OUTOFRNG	The address range from a memory dump or load request did not fall within one of the valid memory maps. This error will also occur if a zero-length dump is requested.	
UNKNOWN	Internal software error.	
UNLCKMTX	Unable to unlock a mutex.	