SMA Technical Memo #141: Using Reflective Memory

Charlie Katz

2000 May 11

`$Id: reflmem.tex,v 1.10 2008/06/18 19:45:26 ckatz Exp $`

Contents

1 Introduction

The Submillimeter Array computers make use of fiber-optic reflective memory from VME Microsystems International Corporation, part number VMIVME-5576. These VME boards allow 256K of RAM to be shared between the array's central computer and the antenna computers. Information written to reflective memory on one end appears on the other end within 1.5μs. They also provide the capability for a board to cause an interrupt to be generated on the VME bus of another board. This feature is used to provide notification between computers when reflective memory contents are changed.

In the final configuration of the SMA, with 8 antennas plus the JCMT and CSO, there will be 10 pairs of reflective memory cards. Each will connect the central computer to one of the 10 antennas.

2 Hardware

2.1 Configuration

The reflective memory cards are configured with a set of jumpers. This section describes the required configuration for use in the SMA. A full description of the jumpers and their meaning may be found in section 5 of the reflective memory instruction manual.

Jumper	Installed?	Comment
J1	yes	factory configured; do not change
J2	yes	" "
A1	yes	" "
A2	yes	" "
A3	yes	" "
A4	yes	" "
J3	no	mask field
J5	yes	select 3.2 MBytes/sec with error checking
J6	no	unused
J8	yes	install across the two rightmost† pins
J10	yes	activates A32 addressing

†Looking at component side of board with VMEbus connectors to the right

The reflective memory cards are accessed through the A32 address space of the VMEbus, appearing at addresses from `0` to `0xa00000`. Any card installed in an antenna computer crate should be given address `0`. Cards installed in the central computer crate are addressed according to which antenna is on the other end of the link. These addresses are configured on pins A20–A23 (the middle four pins on J7) as shown in the table. Also, each card must have a node ID set on pins ID0–ID3 (looking

at the component side of the board with the VMEbus connectors to the right, the rightmost four pins in J9).

card location	linked to	address	jumper installed?‡ A23	A22	A21	A20	node ID	jumper installed?‡ ID3	ID2	ID1	ID0
antenna	central	`0x000000`	yes	yes	yes	yes	0	yes	yes	yes	yes
central	ant. 1	`0x100000`	yes	yes	yes	no	1	yes	yes	yes	no
central	ant. 2	`0x200000`	yes	yes	no	yes	2	yes	yes	no	yes
central	ant. 3	`0x300000`	yes	yes	no	no	3	yes	yes	no	no
central	ant. 4	`0x400000`	yes	no	yes	yes	4	yes	no	yes	yes
central	ant. 5	`0x500000`	yes	no	yes	no	5	yes	no	yes	no
central	ant. 6	`0x600000`	yes	no	no	yes	6	yes	no	no	yes
central	ant. 7	`0x700000`	yes	no	no	no	7	yes	no	no	no
central	ant. 8	`0x800000`	no	yes	yes	yes	8	no	yes	yes	yes
central	ant. 9	`0x900000`	no	yes	yes	no	9	no	yes	yes	no
central	ant. 10	`0xa00000`	no	yes	no	yes	10	no	yes	no	yes

‡Jumper numbers increase right to left, looking at component side of board with VMEbus connectors to the right

Jumpers should be installed in all remaining positions (A24-A31 in J4, AP18-AP19 and A18-A19 in J7, and ID4-ID7 in J9).

2.2 Installation

Once the jumpers are properly configured, the cards may be installed simply by plugging them in to the appropriate VME crates. Each antenna computer crate will contain one reflective memory card. The central crate will contain a reflective memory card for each of the antennas.

Finally, the cards in the antennas must be linked to those in the central crate using multi-mode optical fiber. Be sure that the transmit port of one card connects to the receive port of its partner, and vice versa.

3 Software

The VME controllers which will interact with the reflective memory cards are Motorola PowerPC machines running LynxOS 3.1.0a. The software interface to the reflective memory cards consists of a device driver and an application programming interface. The software is located in the directory `$COMMON/reflmem`.

3.1 Installation

Normally the software may be installed simply by checking out the "reflmem" module from the SMA CVS repository. This will create the directory `reflmem`, which should reside in the `$COMMON/` directory. The software system may be built by typing `make all` in the `reflmem` directory.

For reference, this section shows how the files should be installed in the `reflmem` directory and its subdirectories (before compilation). The contents of the CVS directories have been omitted.

```
% cd $COMMON/
% ls -alR reflmem
reflmem/:
total 36
drwxrwxrwx   6 ckatz    ckatz        4096 May  8 11:45 ./
drwxr-xr-x  43 ckatz    ckatz        4096 May  8 11:45 ../
drwxrwxrwx   2 ckatz    ckatz        4096 May  8 11:45 CVS/
-rw-r--r--   1 ckatz    ckatz         436 Dec 30 14:02 Makefile
drwxr-xr-x   3 ckatz    ckatz        4096 May  8 11:45 api/
drwxr-xr-x   3 ckatz    ckatz        4096 May  8 11:45 doc/
drwxr-xr-x   3 ckatz    ckatz        4096 May  8 11:45 driver/
-rw-r--r--   1 ckatz    ckatz        7082 Dec 30 14:02 rm_allocation

reflmem/api:
total 56
drwxrwxrwx   3 ckatz    ckatz        4096 May  8 11:45 ./
drwxr-xr-x   6 ckatz    ckatz        4096 May  8 11:45 ../
drwxr-xr-x   2 ckatz    ckatz        4096 May  8 11:45 CVS/
-rw-r--r--   1 ckatz    ckatz         368 Dec  3 17:54 allocation.mk
-rw-r--r--   1 ckatz    ckatz         224 Dec  3 17:54 api.mk
-rw-r--r--   1 ckatz    ckatz       22407 Dec 30 14:25 rm.c
-rw-r--r--   1 ckatz    ckatz        3509 Dec 30 14:25 rm.h
-rwxr-xr-x   1 ckatz    ckatz        6615 Dec  3 17:54 rm_allocator*

reflmem/doc:
total 164
drwxr-xr-x   3 ckatz    ckatz        4096 May  8 11:45 ./
drwxr-xr-x   6 ckatz    ckatz        4096 May  8 11:45 ../
drwxr-xr-x   2 ckatz    ckatz        4096 May  8 11:45 CVS/
-rw-r--r--   1 ckatz    ckatz       31201 Feb 15 14:40 reflmem.tex

reflmem/driver:
total 88
drwxr-xr-x   3 ckatz    ckatz        4096 May  8 11:45 ./
drwxr-xr-x   6 ckatz    ckatz        4096 May  8 11:45 ../
```

```
drwxr-xr-x   2 ckatz    ckatz        4096 May  8 11:45 CVS/
-rw-r--r--   1 ckatz    ckatz         498 Dec  3 17:54 driver.mk
-rwxr-xr-x   1 ckatz    ckatz         569 Dec  3 17:54 find_driver_service_calls*
-rwxr-xr-x   1 ckatz    ckatz        1942 Dec  6 11:42 install_reflmem*
-rw-r--r--   1 ckatz    ckatz       49800 Dec  3 17:54 reflmem.c
-rw-r--r--   1 ckatz    ckatz        6775 Dec  3 17:54 reflmem.h
```

There must also be symbolic links in `$COMMON/include` and `$COMMON/lib` pointing to the C header and object library files:

```
% cd $COMMON
% ls -l include/ lib/
include:
lr--------   1 root           19 May  4 13:17 rm.h@ -> ../reflmem/api/rm.h

lib:
lr--------   1 root           19 May  4 13:17 rm.o@ -> ../reflmem/api/rm.o
```

3.2 Reflective Memory Allocation Description

Data stored in reflective memory are identified by a set of names, which are listed in the file `$COMMON/reflmem/rm_allocation`. Entries in this file follow a naming convention which includes several "fields":

```
name1_name2_..._type   or   name1_name2_..._Vm_..._type

name1, name2, etc. form a descriptive name
type is a code describing the data type:
   B  = integer, 8 bit                (1 byte)
   S  = integer, 16 bit               (2 bytes)
   L  = integer, 32 bit               (4 bytes)
   F  = IEEE floating point, 32 bit   (4 bytes)
   D  = IEEE floating point, 64 bit   (8 bytes)
   Cn = character array (string) of length n

The code may be preceded by one or more sequences of type "Vm" to
specify a vector.

Examples:
  RM_THIS_VAL_F        : a 4-byte float
  RM_FILE_NAME_C50     : a 50 character string
  RM_DATA_V10_S        : an array of 10 16-bit integers
```

```
RM_FDATA_V128_V128_D : a 128x128 array of 64-bit floating point values
```

Names may not be longer than (RM_NAME_LENGTH − 1) characters, where RM_NAME_LENGTH is defined in the C header file rm.h (the −1 is necessary to leave space for a terminating zero byte). Names are case-sensitive.

To add, change, or delete a data storage location, rm_allocation must be edited, after which the changes must be registered by typing make allocation. Also, the new version should be registered to CVS by typing cvs commit rm_allocation.

If there is difficulty in running make allocation or cvs commit rm_allocation, check the permissions of the directories $COMMON/reflmem, $COMMON/reflmem/CVS, and $COMMON/reflmem/api. See §3.1.

NOTE: After the new allocation table has been registered, all programs using reflective memory must be stopped and the drivers on all computers must be reloaded. This may be done manually (see §3.3), or the computers which use reflective memory may all be rebooted.

3.3 Device Driver

The device driver files are in $COMMON/reflmem/driver. A script entitled install_reflmem is provided to properly install the device driver and create the device node in the filesystem. The script may also be used to uninstall the driver by providing the -u command-line switch. This procedure requires superuser privileges.

When the driver is loaded, the contents of the reflective memory card on the other end of the link are automatically copied into the local card. Thus programs may safely assume that the contents of their local reflective memory cards are consistent with those on the other end of the link, even immediately after a reboot.

3.4 Application Programming Interface

The application programming interface (API) is a set of functions and definitions used by the application programmer to access reflective memory. These functions may be used by including the C header file $COMMONINC/rm.h and linking with the object file $COMMONLIB/rm.o.

In addition to the return codes listed below, all functions may return the value RM_INTERNAL_ERROR. This indicates that an unexpected or inconsistent condition has occurred inside the driver or interface. It should be reported to the reflective memory software maintainer.

`int rm_open(int *alist);`

Prepare for reflective memory use. This function must be called before any other reflective memory activity is performed.

`alist` should point to an `int` array with size `RM_ARRAY_SIZE`. The `rm_open()` function will fill in this array to provide a list of antenna numbers for which reflective memory cards are present in the crate. These numbers may be used to index the data structures returned by other reflective memory functions. The end of the list is indicated by the integer value `RM_ANT_LIST_END`.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_SERVICE`	There is no reflective memory service available
`RM_ALREADY_OPEN`	Reflective memory operations have already been initialized with `rm_open()`
`RM_ALLOC_TABLE_ERR`	There was an error reading the allocation table
`RM_ALLOC_VERS`	Allocation table version mismatch; reload the drivers

`int rm_close(void);`

End reflective memory use. Once this function has been called, no reflective memory activity may be performed unless `rm_open()` is called again. `rm_close()` clears the monitor list and releases any blocked calls to `rm_read_wait()`.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_INIT`	Reflective memory operations have not been initialized with `rm_open()`

`int rm_write(int antenna, char *name, void *buf);`

Write data to reflective memory. No notification is sent to the receiving end.

`antenna` specifies the antenna which should receive the data, and may be one of `RM_ANT_0`, `RM_ANT_1`, `RM_ANT_2`, ..., or `RM_ANT_10`. `RM_ANT_0` is used on an antenna computer to indicate that the data will be received on the central computer. `antenna` may also take the special value `RM_ANT_ALL` to cause data to be written to all antennas for which reflective memory cards are present (see the description of `buf`).

`name` is the name of the location in reflective memory where data should be written. It must be one of the names listed in the file `$COMMON/reflmem/rm_allocation`.

buf is a pointer to a buffer holding the data to be written. If antenna is RM_ANT_ALL, buf should be a pointer to an array of size RM_ARRAY_SIZE containing the data to be written to each antenna. The array is indexed by antenna number. For example, if the central crate contains reflective memory for antennas 3 and 7, data may be written to name on those cards by placing the datum for antenna 3 in buf[RM_ANT_3] and the datum for antenna 7 in buf[RM_ANT_7]. Then a single call to rm_write() with antenna set to RM_ANT_ALL will cause data to be written to both antennas. The available antenna numbers are provided in alist by the rm_open() function.

Returns RM_SUCCESS if successful. On failure, returns a code describing the error:

RM_NO_INIT	Reflective memory operations have not been initialized with rm_open()
RM_ANTENNA_INVALID	There is no reflective memory card for the specified antenna
RM_NAME_INVALID	name is not a valid name from $COMMON/reflmem/rm_allocation
RM_ALLOC_VERS	There is a mismatch between the allocation table for the local reflective memory card and the remote reflective memory card (for antenna). Reinstall the drivers. (If antenna is RM_ANT_ALL, this error will not be returned even if the allocation table for one of the cards does not match. The write to that card will fail silently.)

```
int rm_write_notify(int antenna, char *name, void *buf);
```

Write data to reflective memory. This function is identical to rm_write(), except that any call to rm_read_wait() which is monitoring for changes to name (on both the sending and receiving ends) will be awakened and provided with the new data.

```
int rm_read(int antenna, char *name, void *buf);
```

Read data from reflective memory without waiting for notification.

antenna specifies the antenna from which to read, and may be one of RM_ANT_0, RM_ANT_1, RM_ANT_2, . . . , or RM_ANT_10. RM_ANT_0 is used on an antenna computer to read from the reflective memory for the central computer. antenna may also take the special value RM_ANT_ALL to read data from all reflective memory cards which are present in the crate (see the description of buf).

name is the name of the location in reflective memory from which data should be read. It should be one of the names defined in the file $COMMON/reflmem/rm_allocation.

buf is a pointer to a buffer in which the data will be stored. If antenna is RM_ANT_ALL, buf should be a pointer to an array of size RM_ARRAY_SIZE; data will be written in the array

slots corresponding to the antennas present. The array is indexed by antenna number. For example, if the central crate contains reflective memory cards for antennas 3 and 7, data from `name` for antenna 3 will be placed in `buf[RM_ANT_3]`, and data from `name` for antenna 7 will be placed in `buf[RM_ANT_7]`. The slots which will be filled by the `rm_read()` function are those returned in `alist` by the `rm_open()` function. The values in `alist` may be used directly to index `buf`.

Note: It is the responsibility of the caller to insure that `buf` points to a space large enough to hold the requested data. The size of data stored in a particular location is encoded into the name (see §3.2). It may also be determined with the `rm_get_size()` function.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_INIT`	Reflective memory operations have not been initialized with `rm_open()`
`RM_ANTENNA_INVALID`	There is no reflective memory card for the specified antenna
`RM_NAME_INVALID`	`name` is not a valid name from `$COMMON/reflmem/rm_allocation`
`RM_ALLOC_VERS`	There is a mismatch between the allocation table for the local reflective memory card and the remote reflective memory card (for `antenna`). Reinstall the drivers. (If `antenna` is `RM_ANT_ALL`, this error will not be returned even if the allocation table for one of the cards does not match. The read from that card will fail silently, leaving that element of `buf` unchanged.)

```
int rm_read_wait(int *antennap, char *name, void *buf);
```

Wait for notification that the data at one of the locations in the monitor list has changed, and return the new data. When notification is received, the function will return and the space pointed to by the parameters will contain the new information.

`*antennap` will contain one of `RM_ANT_0`, `RM_ANT_1`, `RM_ANT_2`, ..., or `RM_ANT_10`, indicating the associated antenna for which the data changed. On an antenna computer, `*antennap` is set to `RM_ANT_0`.

`name` will contain one of the names from the file `$COMMON/reflmem/rm_allocation`, indicating which data changed. `name` must point to a buffer of size `RM_NAME_LENGTH` bytes.

`buf` points to a buffer in which the new data will be stored. **NOTE: It is the responsibility of the caller to insure that `buf` points to a space large enough to hold the largest value which may be returned, as specified in the monitor list.** For example, if an 8-byte double and a 50-byte string are being monitored, `buf` must point to a 50-byte space.

The monitor list is manipulated using the functions function `rm_monitor()`, `rm_no_monitor()`, and `rm_clear_monitor()`. The monitor list in place at the time of the call to `rm_read_wait()` is used; changes to the monitor list will become active in subsequent calls to `rm_read_wait()`.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

RM_NO_INIT	Reflective memory operations have not been initialized with rm_open()
RM_MON_LIST_EMPTY	No reflective memory locations are being monitored; the monitor list is empty
RM_REL_PREMATURE	The function returned prematurely due to a signal or a manual release by rm_release_read() or rm_close()

```
int rm_monitor(int antenna, char *name);
```

Add an entry to the monitor list for a particular data location on a particular antenna, so that any changes may be detected through rm_read_wait().

antenna specifies the antenna to monitor, and may be one of RM_ANT_0, RM_ANT_1, RM_ANT_2, ..., or RM_ANT_10. RM_ANT_0 is used on an antenna computer to monitor for changes made by the central computer. antenna may also take the special value RM_ANT_ALL to monitor name on all antennas for which reflective memory is present in the crate.

name specifies the name of the location to be monitored, and should be one of the names defined in the file $COMMON/reflmem/rm_allocation.

Changes to the monitor list become active only in subsequent calls to rm_read_wait().

If name is added with antenna RM_ANT_ALL, any entries already in the monitor list for name with individual antennas are removed from the list since they are superseded by the RM_ANT_ALL specifier.

Returns RM_SUCCESS if successful. On failure, returns a code describing the error:

RM_NO_INIT	Reflective memory operations have not been initialized with rm_open()
RM_ANTENNA_INVALID	There is no reflective memory card for the specified antenna
RM_NAME_INVALID	name is not a valid name from $COMMON/reflmem/rm_allocation
RM_IN_MONITOR_LIST	The specified name/antenna pair is already in the monitor list

---

```
int rm_no_monitor(int antenna, char *name);
```

Remove an antenna/name pair from the monitor list.

antenna specifies the antenna to stop monitoring, and may be one of RM_ANT_0, RM_ANT_1, RM_ANT_2, ..., or RM_ANT_10. RM_ANT_0 is used on an antenna computer to refer to the central computer. If antenna has the special value RM_ANT_ALL, all monitor list entries for the specified name will be removed. The only way to remove a monitor list entry for RM_ANT_ALL is to call rm_no_monitor() with antenna set to RM_ANT_ALL.

`name` specifies the name of the datum for which monitor list entries will be removed, and should be one of the names defined in the file `$COMMON/reflmem/rm_allocation`.

Changes to the monitor list become active only in subsequent calls to `rm_read_wait()`.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_INIT`	Reflective memory operations have not been initialized with `rm_open()`
`RM_ANTENNA_INVALID`	There is no reflective memory card for the specified antenna
`RM_NAME_INVALID`	`name` is not a valid name from `$COMMON/reflmem/rm_allocation`
`RM_NOT_IN_MON_LIST`	The specified name/antenna pair does not appear in the monitor list

---

```
int rm_clear_monitor(void);
```

Remove all entries from the monitor list.

Changes to the monitor list become active only in subsequent calls to `rm_read_wait()`.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_INIT`	Reflective memory operations have not been initialized with `rm_open()`
`RM_MON_LIST_EMPTY`	No reflective memory locations are being monitored; the monitor list is empty

---

```
int rm_release_read(void);
```

Release all blocked `rm_read_wait()` functions belonging to the calling process.

Returns `RM_SUCCESS` if successful. On failure, returns a code describing the error:

`RM_NO_INIT`	Reflective memory operations have not been initialized with `rm_open()`
`RM_NO_BLOCKED_READ`	There were no blocked `rm_read_wait()` functions to release

```
int rm_get_num_alloc(int *np);
```

Report the number of valid allocation names. The value is stored in the location pointed to by `np`.

Returns RM_SUCCESS if successful. On failure, returns a code describing the error:

RM_NO_INIT	Reflective memory operations have not been initialized with rm_open()

```
int rm_get_alloc_names(char *namelist[]);
```

Report the valid allocation names by storing them in namelist. namelist must be declared as type char **, then initialized with malloc() to hold n pointers, where n is the number provided by rm_get_num_alloc(). Then each of the n pointers must be initialized to point to a space of length RM_NAME_LENGTH. See §3.5 for an example.

Returns RM_SUCCESS if successful. On failure, returns a code describing the error:

RM_NO_INIT	Reflective memory operations have not been initialized with rm_open()

```
size_t rm_get_size(char *name);
```

Find the size (in bytes) of the datum stored at location name. Returns the size if successful, zero otherwise.

```
void rm_error_message(int s, char *string);
```

Write a message with format "string: description" to stderr. string is a string provided by the caller, and description is a short string describing the code s, which should be the integer returned by a reflective memory service function.

If the code s is not recognized, nothing is printed.

3.5 Sample Program

This program demonstrates the use of the API. It could be compiled with the command

```
gcc -o rm_sample -I$COMMONINC rm_sample.c $COMMONLIB/rm.o
```

```
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <rm.h>

int main(void) {

  int antlist[RM_ARRAY_SIZE];
  int status, antenna, i;
  char name[RM_NAME_LENGTH];
  char **namelist;
  int num_alloc;

  double dval_array[RM_ARRAY_SIZE],dval;
  short  sval_array[RM_ARRAY_SIZE],sval;

  /* intialize */
  status = rm_open(antlist);
  if(status != RM_SUCCESS) {
    rm_error_message(status, "rm_open()");
    exit(1);
  }

  /* report the list of available antennas */
  printf("available antennas: ");
  i=0;
  while(antlist[i] != RM_ANT_LIST_END) printf("%d ",antlist[i++]);
  printf("\n");

  /* report the list of allocation names */
  status = rm_get_num_alloc(&num_alloc);
  if(status != RM_SUCCESS) {
    rm_error_message(status, "rm_get_num_alloc()");
    exit(1);
  }

  namelist = (char **)malloc(num_alloc*sizeof(char *));
  for(i=0; i<num_alloc; i++)
    namelist[i] = (char *)malloc(RM_NAME_LENGTH);

  status = rm_get_alloc_names(namelist);
  if(status != RM_SUCCESS) {
    rm_error_message(status, "rm_get_alloc_names()");
    exit(1);
  }
  printf("Allocation list (%d entries):\n",num_alloc);
  for(i=0; i<num_alloc; i++)
    printf("  %s\n",namelist[i]);

  /* write the double precision value 1.234 to */
  /* RM_HOUR_ANGLE_HR_D on antenna 1 */
```

```
dval = 1.234;
status = rm_write(RM_ANT_1, "RM_HOUR_ANGLE_HR_D", &dval);
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_write()");
  exit(1);
}

/* write the short integer value 1 to RM_YIG1_LOCKED_S for */
/* all antennas which have cards in this crate; send notification to */
/* the receiving ends */
i=0;
while(antlist[i] != RM_ANT_LIST_END) sval_array[ antlist[i++] ] = 1;

status = rm_write_notify(RM_ANT_ALL, "RM_YIG1_LOCKED_S", sval_array);
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_write_notify()");
  exit(1);
}

/* read the double precision float from RM_HOUR_ANGLE_HR_D */
/* for antenna 7 */
status = rm_read(RM_ANT_1, "RM_HOUR_ANGLE_HR_D", &dval);
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_read()");
  exit(1);
}

/* read the double precision float values at RM_HOUR_ANGLE_HR_D */
/* for all antennas */
status = rm_read(RM_ANT_ALL, "RM_HOUR_ANGLE_HR_D", dval_array);
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_read()");
  exit(1);
}
i=0;
while(antlist[i] != RM_ANT_LIST_END)
  printf("value at \"%s\" for antenna %d is %f\n",
 "RM_HOUR_ANGLE_HR_D", i, dval_array[ antlist[i++] ]);

/* wait for the data to change at RM_YIG1_LOCKED_S on any antenna */
status = rm_monitor(RM_ANT_ALL, "RM_YIG1_LOCKED_S");
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_monitor()");
  exit(1);
}

status = rm_read_wait(&antenna, name, &sval);
if(status != RM_SUCCESS) {
  rm_error_message(status, "rm_read_wait()");
  exit(1);
}
```

```
  printf("Data changed at \"%s\", antenna %d; new value is %d\n",
          name, antenna, sval);

  /* finished */
  status = rm_close();
  if(status != RM_SUCCESS) {
    rm_error_message(status, "rm_close()");
    exit(1);
  }

  return(0);
}
```

4 Execution Times

Execution times for the `rm_read()` and `rm_write()` calls were measured. Execution times vary depending on the size of the allocation list and which member of the allocation list is requested in the `name` parameter, due to the time required to resolve `name` into a memory address offset.

The results shown here were measured using the list of allocation names when it contained 119 entries. The test program ran on an otherwise unloaded PowerPC running LynxOS 2.5.1. Measurements were made for every `name` in the list. The minimum, mean, and maximum execution times were measured separately for 2, 4, and 8 byte accesses.

datum size (bytes)	function	execution time (μsec) minimum	mean	maximum
2	`rm_read()`	4.33	5.37	6.26
	`rm_write()`	3.35	4.38	5.24
4	`rm_read()`	4.82	5.79	6.73
	`rm_write()`	3.71	4.30	4.81
8	`rm_read()`	6.21	7.99	8.63
	`rm_write()`	4.09	4.65	5.09