DAΦNE TECHNICAL NOTE

INFN - LNF, Accelerator Division

Frascati, March 31, 1994

Note: **C-12**

The DANTE Mailbox system

G. Di Pirro, M. Masciarelli, M. Verola

The DAΦNE control system, DANTE, uses a direct memory access mechanism to retrieve data and to send messages between the different levels, instead of a standard network.

This note explains the details of the communication method and describes the routines implemented to exchange messages in the system.

The advantages of using a direct memory access in the system are speed and simplicity. We don't use any kind of communication protocol and we don't need any sophisticated mechanism to check the correct data transmission: we write and read back or read twice the same message.

The only messages used in DANTE are:

- commands from the first level (PARADISE) to the third level (HELL) to generate some action in the system;

- error messages from HELL to second level (PURGATORY) and from this to PARADISE.

In the system every VME CPU and every console has two memory areas: one to send messages and one to receive from the nearest levels. Send and receive areas are relative to the single CPU. For example, in the communication between the first and second level the address of the send memory area of PURGATORY is the same of the receive memory area of PARADISE.

In Fig. 1 we describe a general mailbox area.

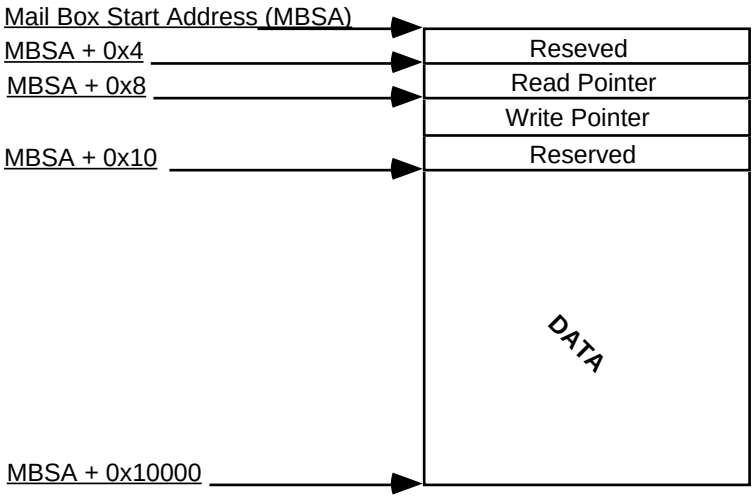

Fig. 1 - Mailbox Structure

The read and write mechanisms for a mailbox are completely asynchronous and independent, since they are performed by two separate CPUs.

The header of the mailbox contains two pointers, one relative to the read operation, RDPTR, and the other to the write operation, WRPTR. The pointers are managed independently by the appropriate CPU. For example, when a console sends a message to PURGATORY the console changes the WRPTR after writing the message and the CPU in PURGATORY changes the RDPTR after reading the message.

The bus error is managed by the routine. This prevents local CPU system crashes and lets the program handle the error without losing control of the execution.

The message structure is show in Fig. 2.

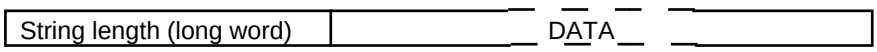

Fig. 2 - Message Structure

We have written two routines to implement this mechanism.

WriteMailbox.

Fig. 3 shows the schematic diagram of the write mailbox operation. The memory area is a circular memory 0x10000 bytes long. When the WriteMailbox routine does not have enough space to write the current message between the write pointer and the physical end of the memory, it writes a zero in the first location that follows the last message and wraps around to write the message from the beginning of the memory. This zero indicates to the ReadMailbox routine that a wrap has occurred.

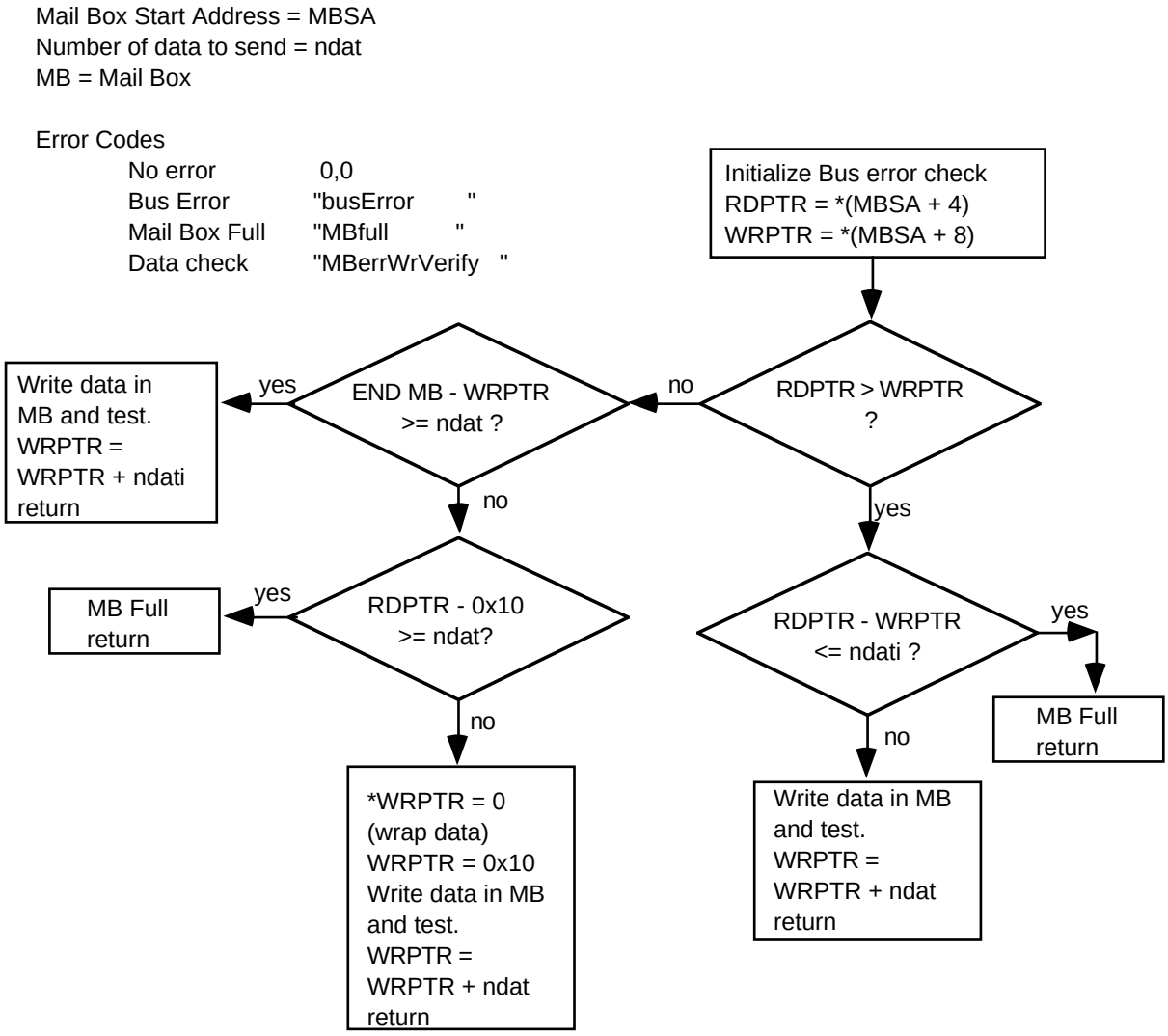

Fig. 3: WriteMailbox Schematic Diagram

ReadMailbox.

Fig. 4 shows the schematic diagram of the read mailbox operation. When a CPU is performing a read operation, a message dimension equal to zero indicates that it must wrap the pointer and restart from the first position in the mailbox memory area.

Mail Box Start Address = MBSA
Number of data to read = ndat
MB = Mail Box

Error Codes

No error	0,0
Bus Error	"busError "
Mem Full	"MacMemFull "
Data check	"MBerrRdVerify "

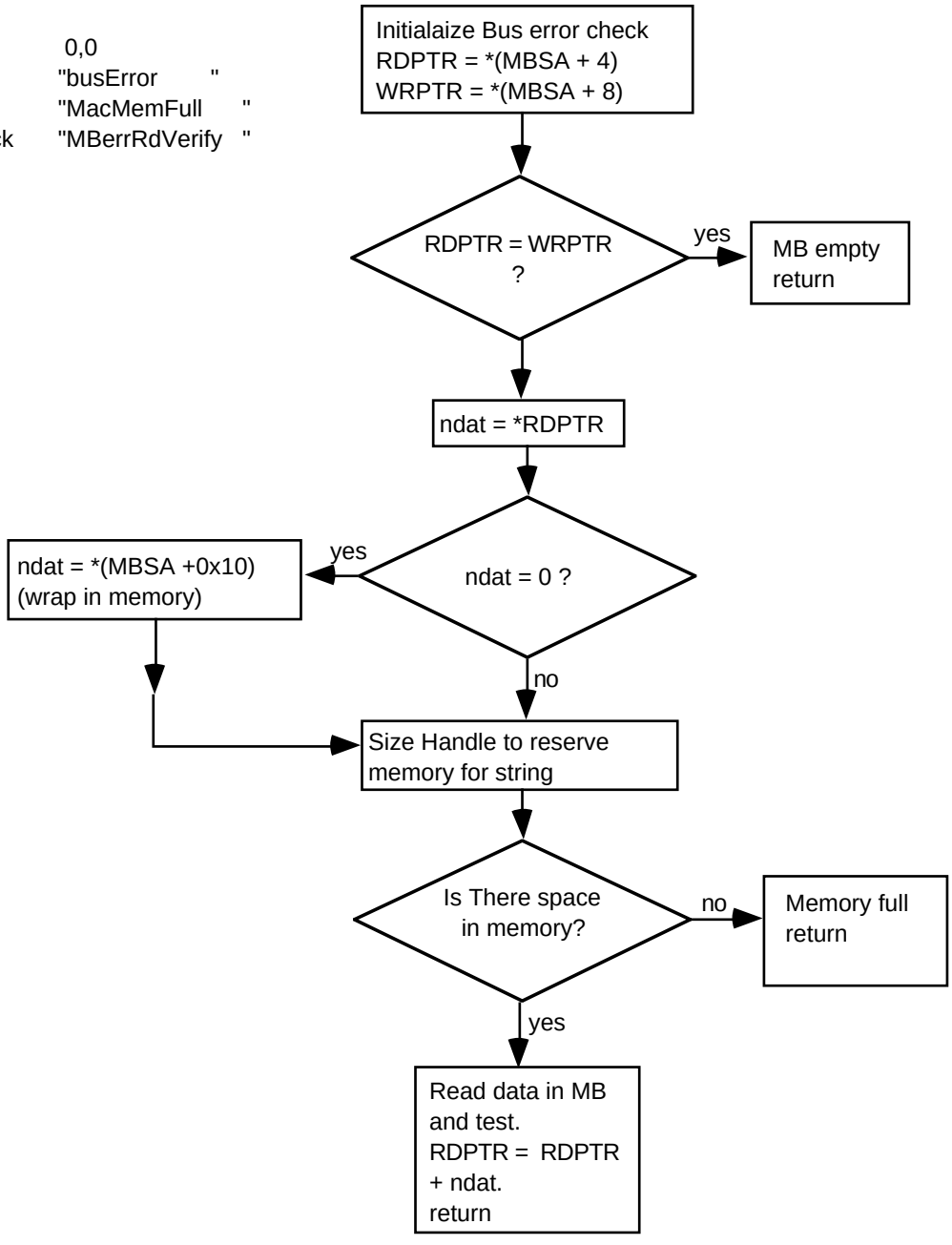

Fig. 4: ReadMailbox Schematic Diagram

Time measurement

We have measured the time performance of the routines. This measurement was executed on a Devil with two different message lengths and in two conditions, mailbox empty and mailbox empty but at the end of the mailbox memory area. In Table 1 we report some results.

ReadMailbox		**WriteMailbox**	
64 Bytes	128 Bytes	64 Bytes	128 Bytes
top of MB			
303	365	160	218
bottom of MB			
305	368	167	222

Table 1 - Time measurements in µs

As we expected, these times are very short when compared with data exchange on a network. The read operation is longer than the write operation because we need to reserve space in memory to accommodate the received data.