3.3 BADEDAS-E

BADEDAS-E is the set of all BADEDAS software, which executes on the Eclipse. This software conducts the down-line loading of the front-end processor and provides the user interface to it.

3.4 USER INTERFACES

The user controls BADEDAS using a simple set of commands entered from an Eclipse user console or by calling BADEDAS subroutines from a Fortran 5 program.

The four BADEDAS commands are summarised in table 2. Devices and user routines are always referenced in these commands by way of their three character BADEDAS mnemonic (see table 1).

TABLE 2: BADEDAS COMMAND SUMMARY

COMMAND	DESCRIPTION
X BAD/I	Initialise a BADEDAS device or user routine eg. X BAD/I AD0,4096
X BAD/P	Set up and start a BADEDAS pipe eg. X BAD/P AD0,UR0,FILE.DAT,3
X BAD/T	Terminate a BADEDAS pipe eg. X BAD/T AD0
X BAD/R	Reset BADEDAS eg. X BAD/R

The initialise command is used to set up the device drivers and user routines. Examples of this would be to set the size of blocks to be input by a source device or to set the values of the arguments for a transformation to be performed by a user routine. The example shown for the initialise command in table 2 sets the input blocksize of the A/D converter to 4096 words.

The pipe command is used to establish a BADEDAS pipe between a source and a destination device and to start the flow of data blocks along that pipe. The pipe command may include the mnemonic of a user routine which is to perform some sort of processing on the data blocks passing through the pipe.

In the case where the host computer is to be the destination or source of the pipe, an AOS disk file name must be supplied in the pipe command in place of the host's mnemonic. When the host computer is the pipe source, the file must be an existing AOS file. This file must contain the data which is to be sent down the pipe. When the host is to be a pipe destination, the file must not exist in the current AOS user directory. The pipe command will then cause a file of that name to be created and all data received by the host will be stored in the file.

The pipe command example shown in table 2 specifies the A/D converter as the pipe source, UR0 as the user routine which will process data blocks input by the A/D converter, FILE.DAT as the AOS disk file in which the data blocks will be stored on being received by the Eclipse and it sets the number of blocks to be input by the A/D converter to three.

The terminate command is used to terminate a BADEDAS pipe before the number of blocks specified in the pipe command have been input by the pipe source. Since a BADEDAS pipe is identified by its source device, the terminate command must refer to the mnemonic of the source device of the pipe which is to be terminated. The example shown in table 2 terminates the pipe whose source device is the A/D converter.

The reset command is used to force a BADEDAS system reset. In effect this command just makes the Micronova execute a halt instruction. Immediately after the Micronova CPU executes a halt instruction, control is transferred to the console debug monitor. As a result of the reset command, the next BADEDAS command entered by the user will cause BADEDAS-M to be re-down-line loaded into the Micronova (see "Down-line Loading" in this chapter). The reset command has the same effect as manually resetting the Micronova. An example of a reset command is provided by table 2.

The BADEDAS Fortran 5 interface comprises six user-callable subroutines. The calls to these subroutines are summarised in table 3.

TABLE 3: BADEDAS FORTRAN 5 CALL SUMMARY

CALL	DESCRIPTION
BADI	Initialise a BADEDAS device or user routine
BADP	Set up a BADEDAS pipe
BADW	Write a block of data into a pipe
BADR	Read a block of data from a pipe
BADT	Terminate a pipe
BADX	Reset BADEDAS-M

Unfortunately, it is not possible to provide examples of calls to these subroutines without entering into a detailed explanation of the parameters. A detailed description of each subroutine call is given in the BADEDAS User's Manual.

The initialise, terminate and reset subroutine calls correspond in function to the BADEDAS commands of the same name. The pipe call is similar to the BADEDAS pipe command with the exception that, following a pipe subroutine call in which the host computer is the pipe destination, a read call must be issued, by the Fortran 5 user program on the host, for each block of data to be input. This enables the user program to be in control of the data flow across the serial link between the host and the front-end. All blocks of data which are destined for the host are therefore queued and are only transmitted to the host when the user's program issues a read call. If the host is the pipe source then each block of data is transmitted to the front-end by the user program issuing a write call.

The user-callable subroutines and BAD, the program which executes BADEDAS commands, both form part of the collection of software known as BADEDAS-E, and they both have basically the same job of translating the parameters passed to them into a terse command which is then transmitted to the Micronova. This translation consists of replacing the three byte BADEDAS mnemonics with one byte internal codes (see table 1), converting ASCII numeric strings to their binary equivalent and omitting any delimiters. Because of the similarity of the two interfaces, they make use of the same routines for the validation of parameters, and for I/O with the Micronova (see figures 5A and 5B).

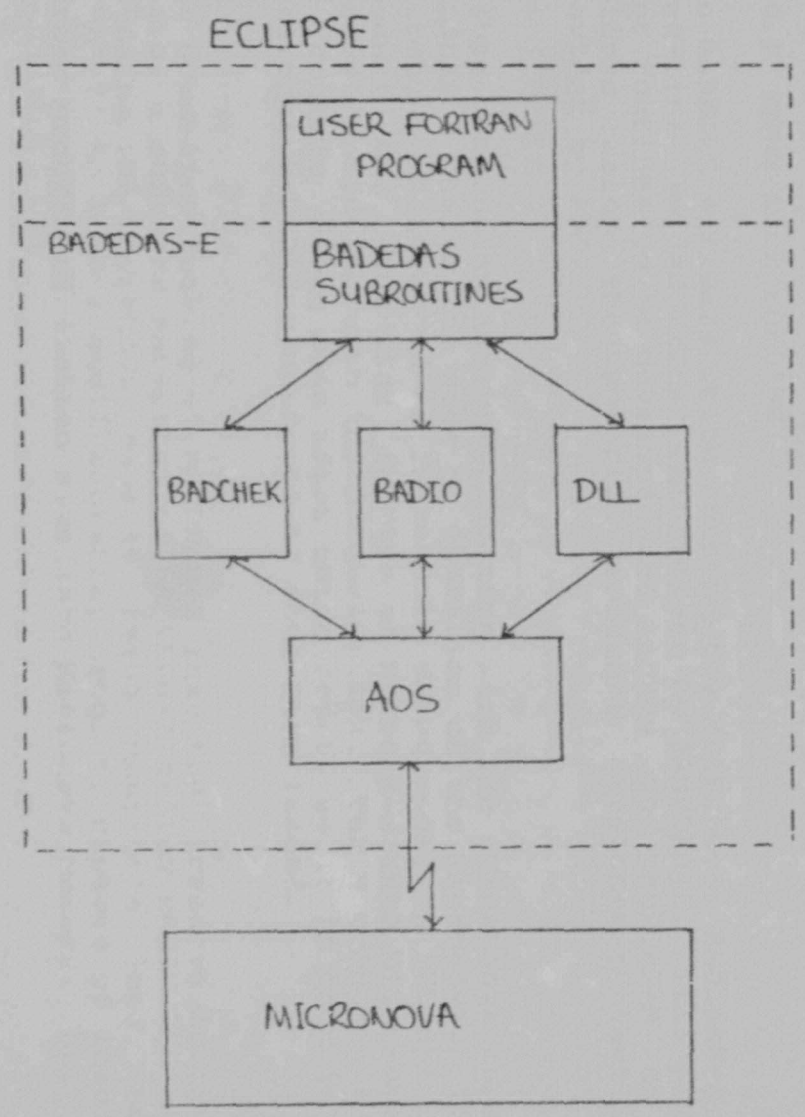

FIG 5A : BADEDAS-E FORTRAN 5 INTERFACE

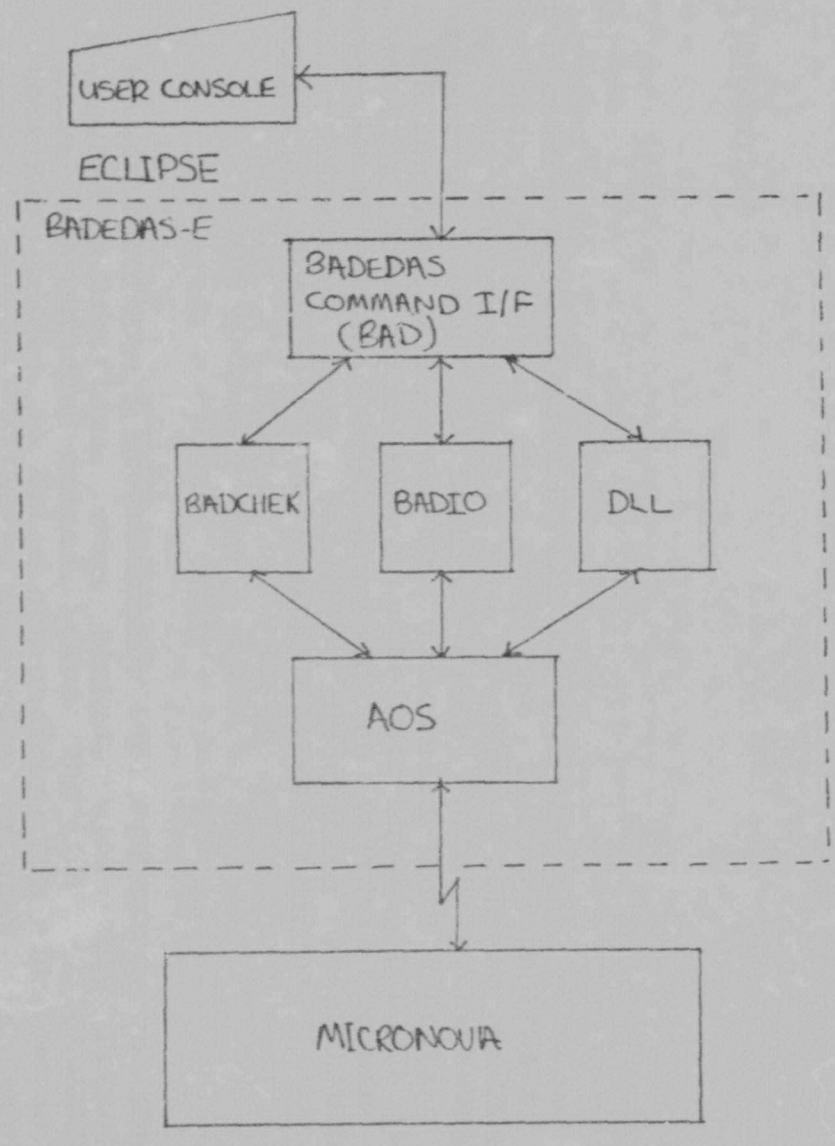

FIG 5B : BADEDAS-E COMMAND INTERFACE

With reference to figures 5A and 5B, DLL is the subsystem of routines responsible for down-line loading BADEDAS-M, the Micronova control software. More details of DLL are provided in the section "Down-line Loading" below. BADIO is the subsystem of I/O routines, which uses AOS I/O calls for inputting and outputting data from and to the Micronova. BADCHEK is the subsystem of routines which are used to validate command and call parameters. More detailed information about BADIO and BADCHEK can be found in the BADEDAS Technical Manual.

NOTE

A user's Fortran 5 program which includes calls to the BADEDAS subroutines, when linked, will only include those subroutines called, plus the other BADEDAS routines needed by those subroutines. In other words, only the software which is essential for what the user is doing is linked into his program. Similarly, the program BAD consists only of the BADEDAS-E routines essential to it.

3.4.1 DOWN-LINE LOADING

The BADEDAS software which runs on the Micronova, BADEDAS-M, is down-line loaded from the Eclipse by BADEDAS-E. This permits user routines to be developed and linked to BADEDAS-M on the Eclipse using the powerful software development facilities provided by AOS.

Before commencing any communication with the Micronova system, BADEDAS-E checks the mode of operation of the Micronova. If the Micronova is found to be in console debug mode, BADEDAS-E will conduct the down-line loading of BADEDAS-M into Micronova RAM before continuing with the communication. The Micronova is only in console debug mode after being reset, so it is only in this case that BADEDAS-M needs to be down-line loaded.

The down-line loading of BADEDAS-M into the Micronova is carried out by a software subsystem of BADEDAS-E called DLL. DLL begins by using the Micronova console debug monitor's load command ("L") to load a small bootstrap program, capable of loading a larger program of limited size into Micronova memory. DLL uses

the bootstrap to load a powerful binary loader, which can load a program of any size into Micronova memory while at the sar time performing error checking. DLL then makes use of this binary loader to down-line load and execute BADEDAS-M.

The bootstrap loader, the binary loader and BADEDAS-M all reside as AOS disk files on the Eclipse system disk.

3.5 BADEDAS-M

BADEDAS-M is the set of routines which BADEDAS-E down-line loads into the Micronova to conf'gure and control it as a front-end processor for data acquisition and dissemination.

BADEDAS-M consists of a kernel, I/O device driver routines and user routines.

3.5.1 KERNEL

The BADEDAS kernel is a software subsystem of BADEDAS-M which provides the environment within which the device drivers and user routines execute.

The kernel provides the following services:

* handling of all interrupts
* memory management
* command interpretation
* user routine scheduling
* message logging
* timeouts
* wake-ups
* handling of software traps

3.5.1.1 Interrupts: - All BADEDAS I/O devices in the Micronova are interrupt-driven. The servicing of all I/O device interrupts is performed by a multiple priority level master interrupt handler, which makes use of two tables, one to obtain the address of the interrupting device's service routine the other to obtain its interrupt priority mask. The master interrupt service routine has the job of identifying the interrupting device and then calling that device's interrupt service routine.

The real-time clock is also a source of interrupts, but these interrupts are handled directly by the real-time clock interrupt routine not via the master interrupt routine. Real-time clock interrupts are used to ensure that the kernel gains control on a regular basis for its periodic system management functions, such as memory allocation and command interpretation. When enabled, the real-time clock causes an interrupt once every 2,4 milliseconds. The real-time clock interrupt routine passes control to the kernel for its management functions after every five real-time clock interrupts.

The servicing of interrupts on the Micronova is a time consuming process in comparison to the same activity on most modern microprocessors, in which much more is done by the hardware. Much effort was, however, applied to ensure that the BADEDAS interrupt service routine were optimised in efficiency.

The order of priority among interrupting devices is as follows:

* A/D and D/A converters
* real-time clock
* host communication input interface
* host communication output interface
* message logging device

3.5.1.2 Memory Management: - The primary unit of data in BADEDAS is the data block. In order to effectively control the allocation of memory blocks from the pool of available memory to house the data blocks, an efficient dynamic memory management system was essential. This system had to meet the following requirements:

* variable blocksizes (ie. The system should not be limited to the use of fixed length blocks, or multiples of fixed length blocks)

* blocks which are no longer in use must become available for re-use as soon as possible

* adjacent blocks of available memory must be collapsed into a single block to prevent unnecessary fragmentation

* fragmentation in general must be limited

* CPU and memory overheads should be kept at a minimum .

Various memory management schemes were considered, and in particular those discussed by Knuth (1972) were studied. Knuth's first-fit boundary tag system was found to meet all the above requirements, and so it was adopted.

The idea behind Knuth's algorithm is to maintain a doubly linked list of the blocks of available memory from which blocks can be conveniently removed. The algorithm assumes that each available memory block has the following format:

WORD	BITS	DESCRIPTION
0	0	tag bit = 0
0	1 to 15	blocksize in words
1	0 to 15	pointer to next available block
2	0 to 15	pointer to previous available block
3 to N	0 to 15	available memory
N+1	0	tag bit = 0
N+1	1 to 15	blocksize in words

In order to reserve a block, the available memory list is searched until a block is found of size equal to or larger than that required. If the block found is larger by more than the minimum allowed remnant size (10 words), it is split into two blocks, one of the required size and the other equal to the remainder. The latter is placed back into the linked list. Bit 0 of the first and last words of the acquired block are set to 1 to indicate that it has been reserved. The format of a reserved memory block is as follows:

WORD	BITS	DESCRIPTION
0	0	tag bit = 1
0	1 to 15	blocksize in words
1	0 to 15	data count in words
2	0 to 15	queue pointer
3 to N	0 to 15	data area
N+1	0	tag bit = 1
N+1	1 to 7	block type: C=command, D=data
N+1	8 to 15	block ID = internal code of source device

The tag bits at each end of a memory block serve to remove the large overheads involved in releasing memory blocks, which are common to most other memory management systems. In order to release a block of memory, the memory management system tests bit 0 of the memory location following the block for release to determine whether the block below is free. If it is, it is removed from the linked list and added to the block for release. Next, bit 0 of the memory location preceding the block for release is tested. If the block above is reserved, the block for release is added to the linked list, otherwise, if the block above is free, the block for release is added to it.

One alternative to first-fit memory allocation is best-fit allocation, in which the available memory blocks are searched for one which is exactly of the size requested. If one of the exact size is not found, the block which is closest to the requested size is used to satisfy the request. The first-fit approach is quicker and has been found experimentally to cause less fragmentation (Knuth, 1972) than the best-fit approach. In the first-fit approach, although the overheads associated with acquiring a block are significantly less than with the best-fit approach, they still depend on the number of blocks in the linked list which must be inspected, before one large enough is found.

In order to prevent the delay associated with acquiring a block everytime one is needed, the BADEDAS source device drivers request blocks for input buffers before they actually need them. For example, the A/D converter driver requests its first buffer when it is initialised. By the time a pipe command is issued, the buffer is available so the conversion may be started immediately and, if the pipe has a block count of more than one, the next buffer is requested immediately after the input of the first one is started. By the time the A/D converter interrupts to indicate the data block has been input, the next buffer, if needed, is available, and so on.

Whenever a task needs to acquire or dispose of a memory block it issues a request to the memory management system. All requests are queued until the kernel gains control clock driver for performing its system management functions. Firstly all blocks to be released are returned into the pool of available memory, then all block acquisition requests are handled. Any requests which cannot be satisfied are requeued in the hope that they can be satisfied the next time around.

3.5.1.3 Command Interpreter: - The BADEDAS command interpreter has an input queue, which it checks whenever the periodic system management is performed. Commands in this queue are terse versions of the commands or subroutine calls issued by the user or user program, in which descriptive mnemonics have been replaced by single byte internal codes, delimiters omitted and numeric strings replaced by their binary equivalents. When the command interpreter finds a command in the queue, it is interpreted and, if valid, is executed. Invalid commands are aborted and an explanatory message is displayed on the monitor console.

The command interpreter executes the commands as follows.

An initialise command causes a device's or user routine's initialisation routine address to be extracted from a look-up table and control transferred to it.

A pipe command causes one or two entries to be made in a table, which effectively links the standard output of a source device driver to the standard input of a destination device driver or the standard output of a source device driver to the standard input of a user routine and the standard output of the user routine to the standard input of a destination device driver and then it causes the source device driver to be started.

A terminate command causes the number of blocks still to be input by the pipe source device driver to be tested and, if this number is not zero, it is set to zero, effectively terminating the pipe.

A reset command causes the Micronova to execute a halt instruction, which in turn results in the Micronova entering console debug mode.

3.5.1.4 User Routine Scheduling: - The user routine scheduler is that part of the kernel which runs in the background. Whenever BADEDAS-M is otherwise idle, the scheduler checks the user routines' input queues, on a round robin basis. When a data block is found in a user routine's input queue, the user routine is given control to process that block. Each scheduled user routine is permitted to process only one block before returning control to the scheduler. The user routine scheduler then queues the processed block for output by the destination device.

3.5.1.5 Message Logging: - Any task in the system may display messages on the monitor console at any time, so some form of discipline is required for queueing messages.

The kernel provides a message logging facility whereby a task may issue a message log request using the address of the start of the message as a parameter. If the monitor console is idle when the request is made, the output of the message is started immediately, otherwise it is queued. The message queue may be up to ten message addresses long before a message queue overflow occurs. If this occurs, it is reported on the monitor console and all messages in the queue are lost. Should this tend to occur frequently, the queue length may easily be increased (see BADEDAS Technical Manual).

The monitor console is used purely as an output device and its driver, which also forms part of the kernel's message logging subsystem, is interrupt driven. An interrupt from this device indicates that the output of a character is complete. The message logger then extracts the next character from the message for output. If this character is non-null it is displayed. If it is the null character, this indicates that the complete message has been output and the queue is checked for the next message. If there is one, its first character is extracted and output, otherwise the monitor console is idled.

3.5.1.6 Timeouts: - The timeout facility provided by the kernel forms a functional part of the communication protocol between BADEDAS-E and BADEDAS-M. The main purpose for the use of timeouts is to prevent incomplete packets from hanging up the communication link. If BADEDAS-M is expecting a n byte packet and less than n bytes arrive, BADEDAS-M keeps the incomplete packet until the timeout period has expired and then discards it. The timeout period has been chosen to be long enough to avoid being affected by the simple delays between characters

caused by the Eclipse being heavily loaded.

NOTE

BADEDAS-E performs the output of each packet using a single AOS call, thereafter the AOS peripheral manager PMGR handles the output of each byte. After the output of the first byte, subsequent bytes are output by PGMR's interrupt level, which is a high priority task and is therefore less affected, than most other tasks, by the loading of the system. So heavy loading of the system will not generally cause timeouts between the bytes of a packet If it did, however, no data would be lost, since a packet sent would not be acknowledged, and as a result BADEDAS-E would retransmit the it (see "Communication Protocol" in this chapter).

In BADEDAS-E all I/O timeouts are handled by AOS.

3.5.1.7 Wake-ups: - If a source device driver requests a memory block and the block has not been allocated by the time it is needed, the device driver goes to sleep. Before going to sleep, the driver issues a wake-up request to the kernel. This request tells the kernel to monitor that driver's memory request and, when it has been honoured, to restart the driver.

3.5.1.8 Traps: - A Micronova trap instruction, in effect, causes a software generated interrupt. At various places in BADEDAS-M, trap instructions have been inserted to handle exceptions, which should only occur if a hardware fault, an undetected system software bug, or a bug in a user routine is encountered.

When a trap instruction is executed, control is passed to the trap handler. If the hardware permits (in the case of a hardware malfunction), the trap code together with a descriptive message are displayed on the monitor console, after which a halt instruction is executed, causing the Micronova to return to console debug mode.

Software trap codes are documented in the BADEDAS Technical Manual.

3.5.2 I/O DEVICE DRIVERS

As stated above, the primary unit of data in BADEDAS is the data block. Source devices produce data blocks and destination devices consume data blocks. Each source device driver has a standard output and each destination device driver has a standard input. In a pipe construct the blocks produced by a source device will pass via that device's driver output to the associated destination device's driver input, to be consumed. An I/O device driver consists of all the software needed to control the device, such as initialisation, start-up and interrupt service routines.

The standard input and standard output interfaces of the device drivers are implemented in a very simple manner. A source device's standard output consists simply of an entry in a look-up table and a destination device's standard input is a data block queueing routine. When a pipe command is encountered, which links a particular source device to a particular destination device, the command interpreter takes the destination device's queueing routine address and places it in the source device's look-up table entry. Data blocks produced by the source device are then queued for consumption by the destination device, simply by the source device driver calling the queueing routine whose address is in its entry in the look-up table.

3.5.2.1 A/D And D/A Device Drivers: - The A/D and D/A converters in the BADEDAS system are operated in data channel mode. In this mode these devices are set up with the size and address of the memory block into which or from which data is to be input or output. The actual transfer of data then takes place under direct memory access. No software participation is needed until the transfer is complete.

An interrupt from an A/D converter indicates that it has completed the input of a data block. The data block is then queued for processing by the user routine or output by destination device driver, whichever is connected to the source device's output. The pipe block count is decremented and if it is not equal to zero, the A/D converter is restarted for the input of the next block of data. When the block count reaches zero the A/D converter is idled.

An interrupt from a D/A converter indicates that it has completed the output of a data block. The memory block which housed the data block is queued for release by the memory management system. The D/A's standard input is then checked for data blocks which have been queued for output. If there are any, the output of the first in the queue is started immediately. When the queue is empty, the D/A converter is idled. If while the D/A is idle, a block is placed in its queue, the queueing routine will automatically start the D/A outputting that block.

3.5.2.2 Host Communication Driver: - There are two drivers, one input and one output, associated with the communication link to the host computer. As with the A/D and D/A drivers, these drivers, respectively, produce and consume data in terms of blocks. A major difference, however is that these devices cannot be operated in data channel modes so, software participation is required for each byte of data handled.

All communication with the host is achieved using a secure, efficient protocol, which shall be described below (see "Communication Protocol").

The communication input driver receives both data and command blocks in the form of one or more packets. Data blocks are queued for processing by a user routine or output by a destination device driver, whichever is connected to the communication input drivers standard output. Command blocks are queued for the attention of the command interpreter.

In BADEDAS, a packet is a message of a convenient length for transmission over the communication link. The maximum length of a packet sent to the host via this link is limited to the length of the AOS ring buffer, as explained below (see "Communication Protocol"). If the ring buffer restriction did not exist, it would still be necessary to limit the length of packets in order to keep the error detection and retransmission mechanisms effective and efficient. Since data blocks are of variable length, it is necessary to divide them up into packets for transmission.

The communication output interface outputs all data blocks which have passed through a pipe for which the destination is the host computer. Each block is output as a set of one or more packets.

3.5.2.3 Adding Extra Device Drivers: - BADEDAS has been designed to allow for the addition of extra device drivers. To add an extra A/D or D/A driver, the existing A/D or D/A driver routines can be copied and the device codes and routine names changed. The names of routines in the existing drivers have a suffix "01", so the next driver could use suffix "02". BADEDAS internal codes and mnemonics have been reserved specifically for one extra A/D and one extra D/A converter (see table 1, AD1 and DA1). The addresses of certain of the device driver routines need to be entered into look-up tables and the device's priority mask needs to be entered in the priority mask table.

A further four extra BADEDAS internal codes have been reserved, two for additional source devices and two for addition destination devices. Any type of device may be added by providing the standard driver routines as described in the BADEDAS Technical Manual and entering their addresses in the necessary tables.

It is not likely that more device drivers will be added than those catered for, but should this prove necessary, it would be possible by reallocation of the BADEDAS internal codes and rearranging the look-up tables.

More detailed information for the adding of extra device drivers is given in the BADEDAS Technical Manual.

3.5.3 USER ROUTINES

BADEDAS provides the capability of linking user-written routines into BADEDAS-M, which can then be included in BADEDAS pipes for processing data en-route between the pipe source and destination. The user routine is called by the user routine scheduler with the address of a data block as a call parameter. The user routine performs some transformation on the data block and then returns control to the scheduler, which then queues the processed block for output by the destination device.

User routines are intended to be simple data transformations. Typically the transformation will be a one-to-one function, with each data element being transformed from one value to another. A typical application of a user routine is for the implementation of a digital filter. An example of a user routine appears in Appendix C.

3.5.2.3 Adding Extra Device Drivers: - BADEDAS has been designed to allow for the addition of extra device drivers. To add an extra A/D or D/A driver, the existing A/D or D/A driver routines can be copied and the device codes and routine names changed. The names of routines in the existing drivers have a suffix "01", so the next driver could use suffix "02". BADEDAS internal codes and mnemonics have been reserved specifically for one extra A/D and one extra D/A converter (see table 1, AD1 and DA1). The addresses of certain of the device driver routines need to be entered into look-up tables and the device's priority mask needs to be entered in the priority mask table.

A further four extra BADEDAS internal codes have been reserved, two for additional source devices and two for addition destination devices. Any type of device may be added by providing the standard driver routines as described in the BADEDAS Technical Manual and entering their addresses in the necessary tables.

It is not likely that more device drivers will be added than those catered for, but should this prove necessary, it would be possible by reallocation of the BADEDAS internal codes and rearranging the look-up tables.

More detailed information for the adding of extra device drivers is given in the BADEDAS Technical Manual.

3.5.3 USER ROUTINES

BADEDAS provides the capability of linking user-written routines into BADEDAS-M, which can then be included in BADEDAS pipes for processing data en-route between the pipe source and destination. The user routine is called by the user routine scheduler with the address of a data block as a call parameter. The user routine performs some transformation on the data block and then returns control to the scheduler, which then queues the processed block for output by the destination device.

User routines are intended to be simple data transformations. Typically the transformation will be a one-to-one function, with each data element being transformed from one value to another. A typical application of a user routine is for the implementation of a digital filter. An example of a user routine appears in Appendix C.

Up to three user routines may be linked into BADEDAS-M at a time. A user routine is allocated a mnemonic UR0, UR1 or UR2 (see table 1) depending on its position in the user routine look-up table (for more details on this consult the BADEDAS Technical Manual).

Along with each user routine it is also possible for the user to provide an initialisation routine. The initialisation routine is called by the command interpreter when it encounters an initialise command for the user routine. The command parameters are passed to the initialise routine by the command interpreter.

A typical application of an initialisation routine for a user routine would be to provide the coefficients of a transformation function, such as a digital filter.

3.6 COMMUNICATION PROTOCOL:

The communication requirements between the host and the front-end made it necessary to use a fairly sophisticated communications protocol.

Inter-processor communication protocols can become very complicated unless one of the processors is made the communication master and the other or others adopt the position of slaves. Mainly to simplify the data flow control aspects of the protocol, the front-end processor is regarded as a slave, which only responds to prompts from the host Eclipse and never transmits unsolicited packets on the link.

The following is a summary of the communication protocol requirements:

* It must be possible for data to be transmitted in its original binary format (ie. it should not be necessary to convert data from binary to a code such as ASCII). This so-called data transparency improves the transmission efficiency by minimising the number of bytes that need to be transmitted per data block.

* All blocks must be transferred in packets, which may be of limited but variable length. This will enable blocks of any length to be transmitted in spite of the limited size of the AOS ring buffer.

NOTE

In AOS, one ring buffer is allocated to each console port. All data arriving at a port is stored in its ring buffer until an AOS read command is issued for that port. The ring buffer is typically 255 bytes long and if more than this number of data bytes arrive at the port before a read call is made, the buffer overflows causing an AOS error and loss of data.

* Each packet must contain a form of error detection, such as a checksum.

* Each packet must contain an identification field to prevent the duplication of a packet should an acknowlegement get corrupted.

* Each packet must contain a field which identifies its source.

* There must be positive and negative acknowlegement of packets. Acknowledgement from the host is essential, as it is an indication, not only of whether the packet was successfully received, but also that the AOS ring buffer has been cleared. This prevents the loss of data through overflow of the ring buffer.

* A timeout facility must be provided to prevent the receiver from hanging-up or getting confused following the arrival of an incomplete packet or the transmitter hanging-up when an acknowledgement fails to arrive.

* There must be a means of preventing acknowledgements from becoming out of synchronism with the packets for which they are intended, especially following a timeout. It is possible for such a problem to occur, due to the buffering of packets by AOS. An acknowledgement arriving after a timeout (an unlikely but possible occurrence in BADEDAS) will be buffered. This acknowledge would then be mistaken for the acknowledgement of the subsequent packet transmitted.

* When data is being transferred from the host to the front-end and the front-end runs out of memory, the front-end must be able to signal the host to stop transmission and later, when memory is available, signal the host to resume transmission.

The above requirements were met by adopting a protocol based

NOTE

In AOS, one ring buffer is allocated to each console port. All data arriving at a port is stored in its ring buffer until an AOS read command is issued for that port. The ring buffer is typically 255 bytes long and if more than this number of data bytes arrive at the port before a read call is made, the buffer overflows causing an AOS error and loss of data.

* Each packet must contain a form of error detection, such as a checksum.

* Each packet must contain an identification field to prevent the duplication of a packet should an acknowlegement get corrupted.

* Each packet must contain a field which identifies its source.

* There must be positive and negative acknowlegement of packets. Acknowledgement from the host is essential, as it is an indication, not only of whether the packet was successfully received, but also that the AOS ring buffer has been cleared. This prevents the loss of data through overflow of the ring buffer.

* A timeout facility must be provided to prevent the receiver from hanging-up or getting confused following the arrival of an incomplete packet or the transmitter hanging-up when an acknowledgement fails to arrive.

* There must be a means of preventing acknowledgements from becoming out of synchronism with the packets for which they are intended, especially following a timeout. It is possible for such a problem to occur, due to the buffering of packets by AOS. An acknowledgement arriving after a timeout (an unlikely but possible occurrence in BADEDAS) will be buffered. This acknowledge would then be mistaken for the acknowledgement of the subsequent packet transmitted.

* When data is being transferred from the host to the front-end and the front-end runs out of memory, the front-end must be able to signal the host to stop transmission and later, when memory is available, signal the host to resume transmission.

The above requirements were met by adopting a protocol based

broadly on the Digital Equipment Corporation's Radial Serial Protocol (see Appendix B). For the purposes of communication, the host is regarded as the master and the front-end as the slave. All communication is solicited by the master. Messages take the form of single or multibyte packets. Single byte packets are used to manage data flow and multibyte packets, to convey data and commands. An alternating acknowledgement scheme was implemented to overcome problems caused by the intermediate buffering of packets by AOS, as described above.

Because BADEDAS-M does not make use of an operating system, it handles all I/O with the host at byte level. It would have been convenient to also have BADEDAS-E handle its I/O with the Micronova at byte level. However, because of the presence of an operating system on the Eclipse, all I/O has to be performed using operating system calls. All consoles are controlled by the AOS peripheral manager, which can, according to the AOS Performance and Tuning Guide, only handle 80 I/O requests per second. Each AOS I/O call results in an I/O request. If BADEDAS-E performed byte level I/O, each byte input or output would require an AOS call. As a result, the peripheral manager would be saturated and BADEDAS would cause the performance of the Eclipse to degrade significantly for other users. In addition to this, I/O with the Micronova would be limited to an unacceptably low 80 bytes per second.

The packet format was chosen with a view of optimising the use of I/O calls provided by AOS. BADEDAS-E requires three AOS calls to read a multibyte packet. The first read is used to determine that it is a multibyte packet, the second to determine the packet length and the third to read the remainder of the packet.

AOS allows various I/O record formats and each was considered before adopting the above approach.

CHAPTER 4

PROJECT PROCEDURE

4.1 OVERALL APPROACH

The BADEDAS project was conducted using formal project management and software engineering procedures. The project lifecycle was divided into the following phases (Metzger,1981):

* system definition
* system design
* programming
* testing and integration
* acceptance testing

4.2 SYSTEM DEFINITION

The purpose of the System Definition Phase was to collect as much information as possible about the problem, to analyse it in order to obtain a thorough understanding of the problem, and then to formally define the problem and to establish a plan for its solution.

The System Definition Phase used as its baseline the Preliminary Specification compiled by Professor Hanrahan, the project supervisor. This document provided a concise statement of the general requirements of the proposed system. The Preliminary Specification was supplemented by personal communication with Professor Hanrahan.

Once information about the proposed system had been carefully studied and the detailed requirements of the project identified, a Problem Specification was drawn up. The Problem Specification, much of which has been incorporated into this dissertation, was intended as a detailed statement of the BADEDAS project requirements as seen by the author.

Based on Metzger's (1981) guidelines, a Project Plan was compiled. The Project Plan laid down a formal framework within which the project could be pursued.

4.3 SYSTEM DESIGN

Whereas the System Definition Phase served to identify what had to be done, the System Design Phase had to provide a solution as to how it should be done. The purpose of the System Design Phase was to define how the requirements stated in the Definition Phase were to be met.

The design was approached using a process of stepwise refinement. Firstly all the high-level functional areas of the proposed system were identified. The system is divided naturally into two sections, the user interface, which has to run on the Eclipse, and the data acquisition system, which has to run on the Micronova.

Figure 6 shows the functional areas into which the system was divided.

Each functional area was decomposed into its separate functions. Complicated functions were subdivided into sequences of less complicated functions. Each function was then scheduled for design and implementation as a software routine. A routine corresponding to a complicated function would contain calls to the routines corresponding to that function's subfunctions.

To illustrate this the decomposition of the memory management funtional area into its separate functions is shown in figure 7.

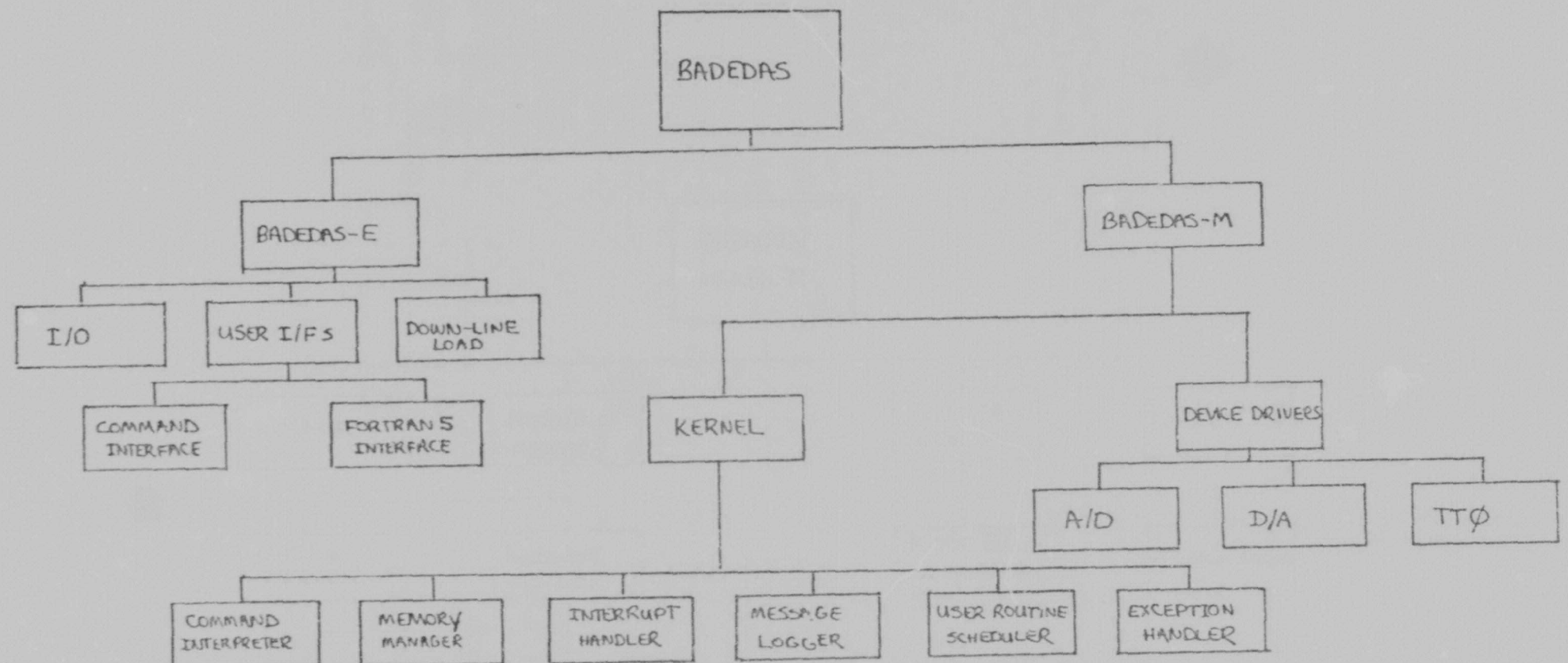

FIGURE 6: BADEDAS FUNCTIONAL BREAKDOWN

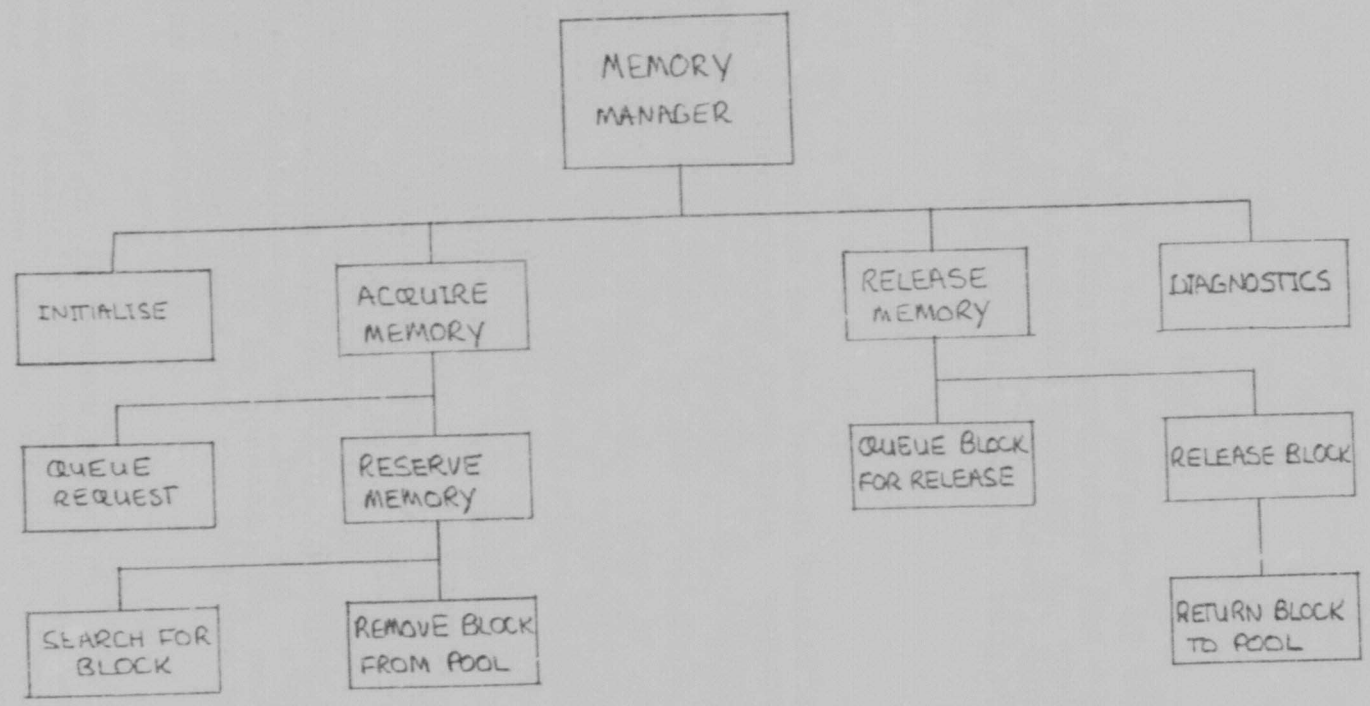

FIGURE 7 : MEMORY MANAGEMENT SYSTEM - FUNCTIONAL BREAKDOWN

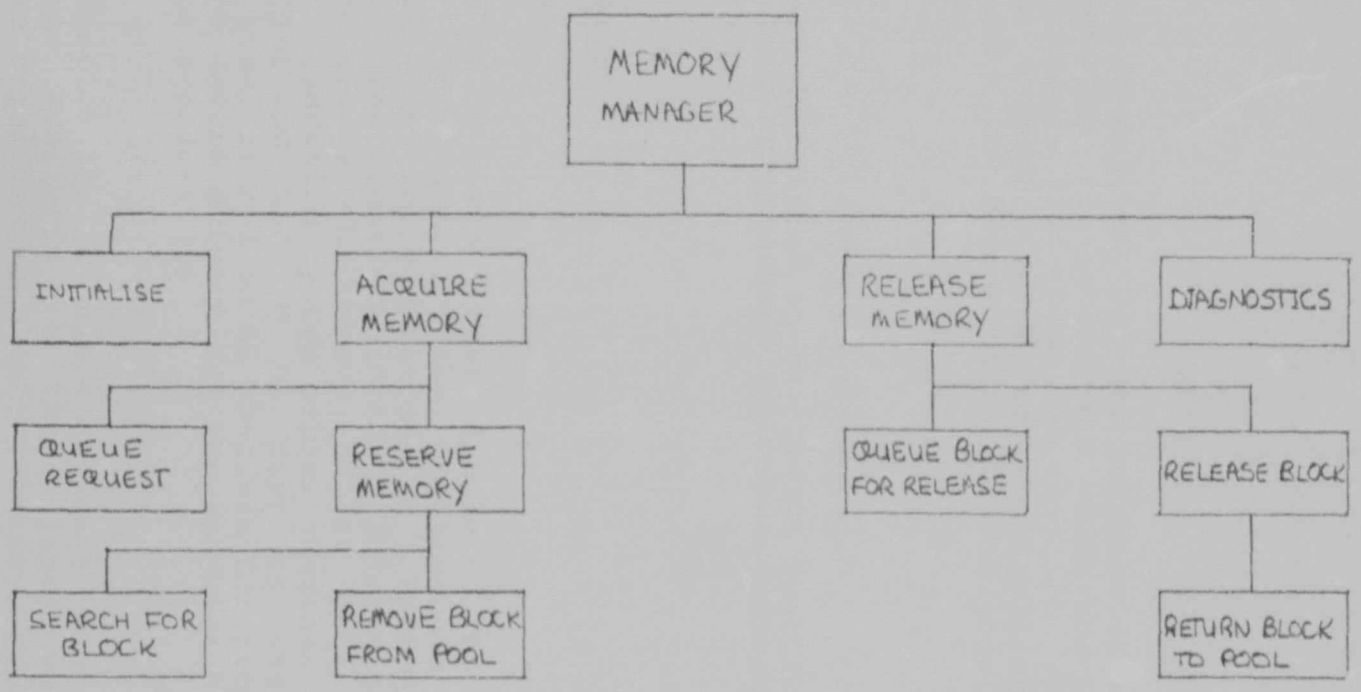

FIGURE 7 : MEMORY MANAGEMENT SYSTEM - FUNCTIONAL BREAKDOWN