CUBE Technical Manual

EuroBEEB Single Board Computer

Control Universal Limited

CONTENTS LIST Revision 5H/1

Introduction

The CUBE Range of Industrial Eurocards

The CUBE range of Industrial Eurocards is designed to provide cost effective solutions to engineering problems.

There is a choice of either 6809 or 6502 CPU cards supported by over 30 peripheral cards. These include the following:

* A/D and D/A conversion
* two/four-channel serial interface
* industrial digital I/O
* high-resolution colour graphics
* Teletext display
* DFS for disk drives
* linear memory extension including DRAM and CMOS RAM
* sideways RAM/ROM
* keyboard, printer and LCD display interfaces
* EPROM programming
* ROM/EPROM simulation

Both 6809 and 6502 CUBE systems use the machine operating concept whereby a software environment is created that allows all the defined peripherals to be called through standard routines with a standard protocol.

EuroCUBE-65 is a 6502-based CPU and single board computer (SBC) designed in Eurocard format (160mm x 100mm). The processor runs at 1 MHz (a 2 MHz option is also available). A standard DIN 41612 bus connector makes the CPU card compatible with the CUBE range of Eurocards.

Features

* 6502 CPU
* on-board real-time calendar clock
* a VIA I/O chip providing 16 I/O channels, 4 control lines and 2 timers
* CUBE/ACORN bus
* four reconfigurable 28-pin memory sockets that take RAM or ROM up to 16 kB per device
* choice of memory maps
* battery back-up for the real-time clock and any CMOS RAM on board
* RS-423 serial interface (RS-232 compatible)
* RS-422 option for noisy environments
* Auto-run feature allowing Turnkey operation (automatic power-up and run)

Machine Operating System

The machine operating system used in EuroBEEB and EuroCUBE-65 provides a software environment similar to that of the BBC Micro and can support BBC BASIC (or other languages) if required. In addition the MOS contains the peripheral drivers (software to drive cards such as the Teletext card) and Control BASIC (an extension of BBC BASIC which provides BASIC-type commands for analog and digital I/O. See Software Section for details). Future versions of the MOS will contain the drivers for Control NET, a low-cost control network in which a number of EuroBEEBs/EuroCUBE-65s may be networked under the control of a 'Master' station.

EuroBEEB and EuroCUBE-65 Differences

EuroCUBE-65 is supplied with a MOS but without a language ROM or RAM unless requested. The three remaining memory sockets can be configured to the user's requirements by wire-wrapping the headers on each 28-pin memory socket. The user can choose the particular memory map which best suits his application. Further details of this can be found in the sections on memory configuration (Appendix 1 & 2).

EuroBEEB has the same hardware features as EuroCUBE-65 except that wire-wrap headers are not required. The PCB tracks are laid so that the four 28-pin memory sockets, M0 to M3, will take:

M0	16 or 8 kB MOS EPROM
M1	16kB BBC BASIC ROM
M2	8kB CMOS RAM or 8kB EPROM (selected by changing a prewired DIL header)
M3	8kB CMOS RAM

EuroBEEB comes supplied with MOSM.3 with machine code monitor and Control BASIC, BBC BASIC, and 8 or 16kB CMOS RAM.

It would, of course, be possible to configure a EuroCUBE-65 card as a EuroBEEB but this would not be very cost effective!

EuroCUBE-65: Hardware Description

Memory

The microprocessor's address and data buses are taken round the card, connecting to all the memories and input/output devices. Address decoding is achieved by using fuse link PROMs. These have a very fast access time and can be programmed to give almost any address map. There are 16 selectable address maps (see Appendix 1). The last map can be programmed to customer specification. (This service is available from Control Universal at extra cost.) A deselect feature is included on memory socket M0 which can allow input/output devices to be placed in part of the space occupied by the operating system EPROM. This is used to create a 256 byte "hole" in the address decoding. Some of this I/O space is reserved for devices on board the CPU, but some is available for external devices on the same bus. Two address decode lines, "HPAGE" and "LPAGE", are output on to the bus connector and greatly simplify the use of locations within this hole. In addition, a "block" signal allows the use of this single line to decode a definable 4kB without further circuitry. Further details of these I/O features can be found in Appendix 1.

The four memory sockets M0 to M3 are configured as 'byte wide'. This means that any of the standard 24 or 28 pin devices can be fitted to the card, including the BBC BASIC ROM.

The memory socket links can be wire wrapped to specify which device is in each socket, while the address decoders give the size of each socket. (Note that a smaller device will work in a larger address space. It will just mirror, i.e. its data will appear within the allocated memory space as many times as its size will go into the allocated space. For example, a 2kB device mirrors four times in an 8kB slot.)

Memory maps are shown in Appendix 1.

Data retention using CMOS devices is provided for by a battery back-up circuit.

I/O

Two parallel ports and one serial port are included. The parallel ports are provided by a 6522 VIA. This includes two timers, one shift register, four control lines and the two 8 bit ports. The serial device is a 6551 which has an internal programmable baud rate generator. Buffers are used to give RS-422 and RS-423 specification lines. The serial output lines can be made to go tri-state, so allowing multiple channel serial communication.

NOTE: In MOSB.2 and MOSB.3, Timer 1 is used to provide the centisecond clock, so this is not available to the user. See also the note on the use of timers in the section on Control BASIC.

Connectors

Three connectors are fitted. The parallel port uses a 26-way IDC connector. The serial lines are on a 7-pin DIN connector, and the bus expansion is through a 64 way Euro connector.

Reset

Power on, warm, cold and serial resets are possible, and the software supplied allows Auto-run operation.

Calendar Clock

A battery-supported calendar clock (M3000/3002) has stop watch and alarm registers as well as the normal calendar register. Both the alarm and the stop watch can generate interrupts.

Single Rail Supply

RS-423 communication requires +/- 5V. If the user has a single rail supply at +5V, then -5V can be generated on board using the optional 5V inverter.

Fig. 1: EuroBEEB - board lay-out

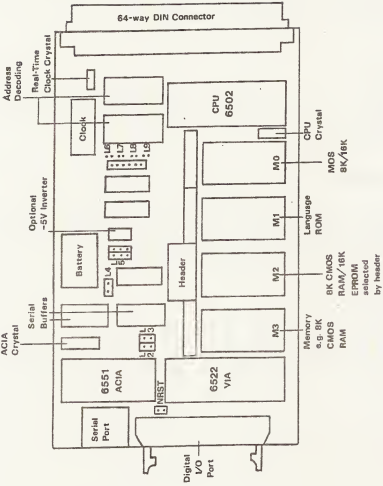
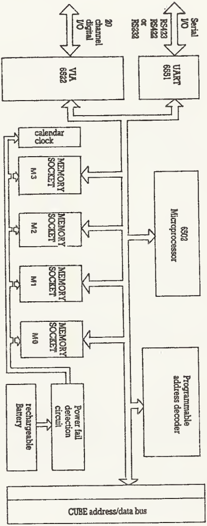


Fig. 2: EuroCUBE-65 - functional diagram



RESET METHODS

1. Power On

This occurs at switch on and is known as a 'cold reset'. It happens approximately 1 second after the supply voltage exceeds 4.5V (+/-0.25V). Before this happens, the VIA (6522) is independently reset by a separate circuit. The software can test the VIA to see whether the registers are in the reset state. If this is the case, the power has just come on and a software 'cold reset' is done (see Appendix 3 for software example).

2. Manual Reset

The reset line is on pin 23 of the parallel port CON 2 (not the VIA itself) or the west pin of link L4. Holding to 0V for longer than 1 second will cause a reset. It causes a system reset identical to a Power On, except that the VIA will not be reset. The VIA registers will now be in their initialised state, and the software can now do a 'warm reset'.

Issue 7 EuroCUBEs have a pair of hard reset pins which cause an instantaneous reset. $\overline{\text{RESET}}$ is taken to two on-board SIL pins located adjacent to Pin 1 of IC11 (6522). To use the hard reset option, connect a momentarily closed (but normally open) push switch between these pins.

3. Serial Reset

The serial port can be configured to cause a reset for remote applications by fitting link L4. The serial input will produce a reset if the receiver input goes active for a time greater than that given by R3,C6. R2 gives a fast discharge for the normal non-active line. This is equivalent to the BREAK key on terminals, but it can also be activated by <shift f9> on a BBC Micro fitted with the CUBE sideways ROM (*EURO). See chapter on using EuroBEEB for more details.

NOTE: When L4 is fitted the manual reset cannot be used.

SERIAL INTERFACE

1. General Description

The serial communication channel is based on a 6551 UART (Universal Asynchronous Receiver Transmitter). Data on this device can be found in a separate Appendix. RS-422/3 drivers and receivers are added to provide a very versatile communications channel. The types of serial communication supported and their attributes are as follows:

Spec	Voltage swing	Maximum distance	Maximum data rate
RS-422	0 to +5V Diff.	4000 feet	10 Mbaud
RS-423	-5 to +5V S/ended	2000 feet	300 kbaud
(RS-232	-12 to +12V S/ended	50 feet	20kbaud)

Further details can be found in Application Note 1 on RS-423/422 (available on request from Control Universal).

NOTE: From Issue 7 onwards, there is a separate on-board oscillator for the 6551.

The output buffers have tri-state outputs so that 'party lines' can be considered. A series of EuroCUBEs using 'Party Line' configuration can achieve multi-channel communication. Full duplex can also be set up. A future development is scheduled to provide an operating system enhancement to allow multi-processor linking called Control NET.

TxD (TRANSMIT DATA) and RxREADY (BUSY) have output buffers with the tri-state condition controlled from RTS (REQUEST TO SEND).

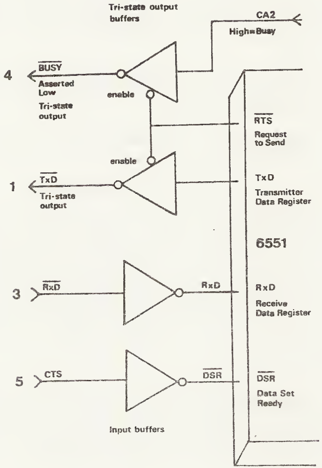
RxD (RECEIVE DATA), TxENABLE (or CTS - CLEAR TO SEND) have input buffers (see Fig. 3).

The serial connector uses a 7 pin DIN socket.

A communications link can be accomplished by simply connecting two serial ports back to back (i.e. Transmit to Receive). If the receiver can process the data faster than the transmitter is sending it, all will be well. At 9600 baud approximately one character will be arriving at the receiver every 1 ms. If this cannot be dealt with, the next character will be missed or the receiver will be switched on in the middle of a character, thus causing bad data.

To avoid this problem, handshaking lines are added. This is effectively a BUSY signal from the receiver telling the transmitter to wait. The transmitter tests whether it is clear to send (TxENABLE or CTS) and flags whether its receiver is ready.

Fig. 3: Functional diagram of the RS-423 serial port.



EuroCUBE uses $\overline{\text{CA2}}$ (from the VIA) for the RxREADY (BUSY) output and $\overline{\text{DSR}}$ (Data Set Ready) for the input (see Fig. 3). The 6551's own RTSxCTS signals are not used directly because the CTS can terminate a transmission in the middle of a byte, and RTS has the side effect of disabling the transmitter interrupt. However, RTS is used as an output enable. See p.18 for 6551 pin usage.

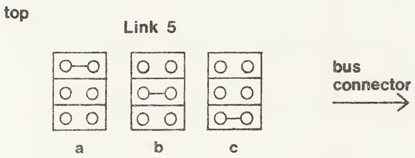
The Operating System provides a fully buffered interrupt driven serial channel. A software example for a very simple serial channel can be found in Appendix 4.

WARNING: $\overline{\text{DSR}}$ sends out interrupts which should be ignored.

Line termination options are shown in Appendix 5.

RS-423/422 Operation

From Issue 7 onwards, Link 5 includes three options for RS-423/422 operation:



(a) Connected to -5V on the backplane:

RS-423 operation (supplied as standard)

(b) Connected to -5V via the optional 7760 voltage inverter:

RS-423 operation (for use with single +5V supply only)

(c) Connected to ground on the backplane:

RS-422 operation

Fig. 4: Using the BBC Micro as a terminal to the EuroCUBE.

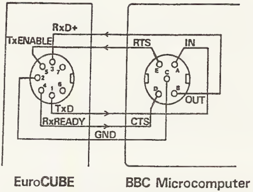
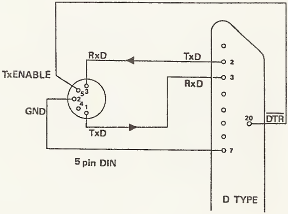


Fig. 5: Linking the EuroBEEB processor card to a standard VDU terminal through the RS-423 (RS-232 compatible) serial port on the EuroBEEB.



NOTES

(1) There are a number of ways of linking the EuroCUBE processor card to a standard VDU terminal through the RS-423 (RS-232 compatible) serial port on EuroCUBE. See Appendix 6 for further details.

(2) When using a terminal other than the BBC Micro, the serial filing system can be switched off by using *FX183,255, so that command LOAD "filename", for example, will now give 'bad command'.

(3) The default conditions for the EuroCUBE serial port match those of the BBC Micro:

9,600 baud
1 Start bit
8 Data bits
1 Stop bit
No parity

(4) ACIA (6551) signal usage on EuroCUBE

TxD	serial output data
RxD	serial input data
CTS	input (tied low)
RTS	enable TxD output buffer (active low)
DSR	TxD enable input
DCD	input (tied low)
DTR	output (no connection)

(5) VIA (6522)

CA2 receiver ready output
Note that all other VIA signals are user available.

(6) RS-232/RS-423 compatibility

Although these two interface standards expect different voltage swings (+/-12V and +/-5V respectively), in practice the RS-423 interface of EuroBEEB will in most cases work quite happily with RS-232.

(7) Baud rate select for input and output

This uses an OSBYTE call entered with the Accumulator set to 7 and the X register as follows (see also under MOSB.3):

1	50	
2	75	
3	109.92	
4	134.58	
5	150	
6	300	
7	600	
8	1200	
9	1800	
10	2400	
11	3600	
12	4800	
13	7200	
14	9600	default
15	19200	

NOTE: OSBYTE 8 (set RS-423 baud rate for data transmission) is not implemented. If the BBC BASIC interpreter is fitted, the equivalent *FX can be used. See "Advanced User Guide for the BBC Microcomputer" for more details of OSBYTES.

VIA TIMER AND CALENDAR CLOCK

1. VIA Timer

The T1 timer on the VIA chip is used for generating the TIME function in BBC BASIC, a facility which is also used by the INKEY command. The user can thus only employ one of the two timers on the EuroCUBE's VIA chip when using BBC BASIC.

2. Calendar Clock

The calendar clock found in the EuroCUBE is based on the MEM M3000/3002 'real-time' clock which features:

(a) A CLOCK which gives the time in hours, minutes and seconds (24-hour clock) and a calendar giving the date, month, year, weekday and week number.

(b) An ALARM which provides an IRQ output when set to a specific date and time (if enabled).

(c) An incremental TIMER which can time events of up to 24 hours duration to the nearest second. The timer can also produce an IRQ output.

The calendar clock is address decoded at &FE18 and appears as a single register. It is driven from a 32.768 kHz crystal and is adjustable by a trimmer capacitor. It will be noted that the clock runs on +5V with power on and from a 2.5V battery with power down. The oscillator is slightly voltage dependent, and when adjusting it, a compromise has to be made. The clock is factory-set assuming a 1:2 on/off ratio of normal use. Clock adjustment is achieved by the time period on pin 15 ($\overline{\text{PULSE}}$). This is set to 3906.25 +/-0.1 µs which gives an accuracy of better than 1 second per day.

The clock register can be regarded in the same way as CMOS RAM. In normal conditions it functions perfectly. However, like CMOS RAM, the clock can be corrupted by power supply transients such as those caused by electrical storms, or by accidental circuit short when the card is removed from the system rack.

Control BASIC, which is part of all MOSB.3 versions, contains the commands CLOCK$, DATE$, DAY, WEEK for interrogating the real-time calendar clock. The section on Control BASIC gives further details. Alternatively, the clock can be accessed using the new OSWORD calls 14 and 15. These calls will read and write to the CLOCK if it is fitted (Issue 5 EuroCUBEs onwards). The parameter block for passing the clock values is described below. Note that a null value, &FF, can be entered when writing to the CLOCK, in order to leave certain values unchanged.

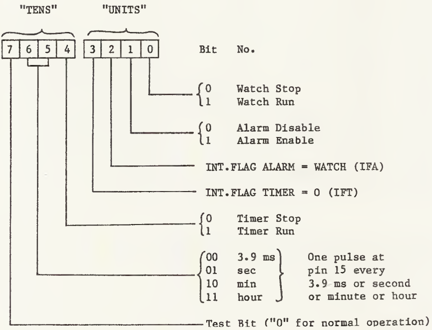
Calendar clock parameter block:

Address	Data	Group	Max.Value	Operations
0 1 2 3 4 5 6 7	Seconds Minutes Hours Date Month Year Week day Week no.	WATCH	59 59 23 28,29,30,31 12 99 07 53	Time date locations incremented by internal timing circuitry under control of status Bit 0
8 9 A B	Seconds Minutes Hours Date	ALARM	59 59 23 28,29,30,31	Alarm data locations preset by user to provide IRQ output at specified time
C D E	Seconds Minutes Hours	TIMER	59 59 23	Timer data locations incremented under control of status Bit 4
F	Status	STATUS		Control

NOTE: The data stored in addresses 0 to 14 is in binary coded decimal (BCD) format.

M3000/3002 Status Word

The M3000/3002 is controlled by the clock status register which in the EuroBEEB is copied into/from parameter block location 15. Its structure is as follows:



If the Monitor peripheral driver (MOSM.3) is fitted, the commands *CLOCK and *DATE will send the current CLOCK time and DATE to the output channel.

If MOSM.3 is not to be used at all in the end application, the data sheet for the M3000 clock chip will be helpful. This is available on request from Control Universal Ltd.

See Appendix 7 for clock software.

EuroBEEB, the BBC-compatible single-board computer, is a 6502-based EuroCUBE with a BBC BASIC II ROM and 8kB or 16kB CMOS RAM memory on board. It is both a low-cost development system and a target system which duplicates the major facilities of the BBC Micro and offers the user the versatility of BBC BASIC in a single Eurocard. The MOSB.3 Operating System supports nearly all the BBC MOS commands, and the BBC BASIC provided on board enables the development and execution of application programs in the familiar software environment of the BBC Micro. The BBC Micro itself can be used as an intelligent terminal and disk file server to EuroBEEB during program development. Files are transferred between disk and the EuroBEEB in target processor mode. Once the application program has been written and tested on EuroBEEB, it can be disconnected for independent operation in stand-alone mode. Communication to the BBC Micro for both second processor and development purposes takes place through the RS-423 serial port. A 256-byte FIFO buffer has been implemented with an interrupt driven system. For example, characters for output are placed in an output buffer which is emptied under interrupt by the transmitter.

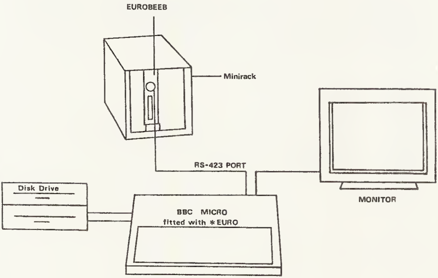
PART II: USING EUROBEEB

DEVELOPING EUROBEEB VIA THE BBC MICROCOMPUTER

Equipment Required

BBC Micro with monitor, DFS and disk drives
*EURO 3.1 EPROM
EuroBEEB with MOSM.3, BBC BASIC and a minimum of 8kB CMOS RAM
Four-slot minirack
RS-423 cable (5-pin DIN to 7-pin DIN)

Hardware Set-up



Procedure

1. Turn OFF the power to the system: The mains switch at the back of the BBC computer should in the DOWN position. The mains switch at the back of the minirack should be in the UP position.

2. Plug the *EURO EPROM into one of the sideways ROM sockets in the BBC Computer. Make sure that the EPROM is in the same orientation as the other ROMs/EPROMs. (The *EURO EPROM contains the software to allow communication between the EuroBEEB and the BBC Computer.)

NOTE: *EURO 3.1 is only compatible with MOSB.3 and its variants.

3. Check the links on the EuroBEEB (this will not be necessary if your card has just arrived from the factory). On Issue 7 boards these should normally be set as follows:

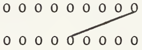
L1	not present
L2	made
L3	made
L4	made
L5	made in the uppermost position only
NRST	NOT made

4. The memory sockets should be configured as follows:

- M0 MOSM.3 (or other series 3 MOS)
- M1 BBC BASIC II
- M2 5565 8kB CMOS RAM (optional but suitable header labelled 5565 should be inserted if this part of the memory is to be used)
- M3 5565 8kB CMOS RAM

NOTE: All series 3 MOS EPROMs are supplied as 27128 devices. From Issue 7 onwards, memory socket M0 on the EuroBEEB is configured for 27128 (2764 compatible). However, if you have an earlier issue of the EuroBEEB board, you will need to make the following modification to M0:

Link for 27128



M0 (See page 50)

5. Push the EuroBEEB firmly into the rack so that the edge connector mates firmly into the backplane.

6. Connect the EuroBEEB to the BBC Micro via the RS-423 lead. Make sure that the cut-out in the metal surround of the 5-pin DIN plug is pointing upwards at the BBC end.

7. Turn on the power to the equipment. (Note: if the EuroBEEB is to be powered from the disk power supply outlet of the BBC Micro, the disk pack must have its own power supply.)

8. The command *EURO (or *EU.) turns the BBC computer into an intelligent terminal, using the RS-423 channel to communicate with the host. Control codes will be interpreted in the same way as the BBC micro's normal VDU channel.

9. When you type in your program on the BBC keyboard it is entered not into the memory of the BBC Micro, but into the memory of EuroBEEB.

10. Other features

(a) Editing facilities

All the BBC Micro's editing commands, e.g. LIST, RUN, AUTO, etc, as well as the cursor control commands are available.

(b) The ESCAPE Key

The ESCAPE key works on a BASIC program in EuroBEEB in the same way as it would on a similar program running in the BBC micro.

(c) RESET

A RESET is transmitted to the BBC by pressing <shift f9>. This will abort the current program in the EuroBEEB and return control to the keyboard. The screen message informs the user that EuroBEEB is still being accessed.

(d) Returning to the BBC Micro

Pressing <shift f0> gives a '*' prompt on the monitor screen. Typing BASIC or B. immediately after the '*' prompt returns control to the BBC micro with BASIC as the current language. This action does not affect the program running in EuroBEEB which will run its 'natural' course whether the user is in EuroBEEB or not.

(e) The BREAK Key

Pressing the BREAK key on the BBC Micro causes a BREAK on the BBC Micro, but this is not transmitted to the EuroBEEB. However, because a 'break' on the BBC micro causes the screen to be cleared, the user may be deceived into thinking that this has had an effect on the EuroBEEB. It has not!

(f) Disk Commands

The standard BBC DFS commands apply to EuroBEEB, i.e. to load a BASIC program into EuroBEEB, simply type LOAD "filename". In fact, all the disk commands are supported in the normal way. However, before issuing a disk command, the user must ensure that he has first accessed EuroBEEB.

For further information on the disk filing system, please consult your DFS manual.

The user may note some delay in loading and saving programs from EuroBEEB which is due to the fact that these programs have to be transmitted along the serial connector.

11. When you are satisfied that the equipment is correctly assembled and switched on, type *EURO, and you should see the following start-up message:

```
MOSB.3 #0  8K

BASIC

>
```

If any other sideways ROMs are present, their start-up titles will also be displayed.

If you now type

<shift f0>

you should see a '*' prompt on the screen. Now type

HELP EURO

and you should see the following information on the screen:

```
*HELP EURO

EURO NET 1

   shift function keys:
   f0  BBC computer *command
   f1  Select new stations
   f2  Poll all stations
   f8  software RESET current stations
   f9  hardware RESET all stations
```

<SHIFT> f0 - is a link to the BBC operating system * command

<SHIFT> f1 - network station select.
Prompts the user to type a new station number, 0 - 31. The default station number is 0.

<SHIFT> f2 - poll all stations
This creates a Mode 7 colour/alphanumeric display showing the status of all stations fitted.

<SHIFT> f8 - send software reset to current station only

<SHIFT> f9 - send hardware reset to all stations

Note on Developing EuroCUBE-65

If you are developing EuroCUBE-65 via the BBC Micro, ensure that memory socket M0 is fitted with the MOS EPROM incorporating the machine code monitor (MOSM.3). Typing *EURO will call up the machine code monitor which will enable you to carry out several diagnostic functions (see chapter on the Machine Code Monitor).

SOME SIMPLE PROGRAMS

When you have connected up your EuroBEEB, try a few simple programs. These programs will only function with EuroCUBE-65 if BBC BASIC is fitted.

1. EuroBEEB (with MOSM.3)

(a) PRINT TIME

Try this a few times. The number currently held in the elapsed time registers will be displayed. Each unit represents 10 ms.

(b) *HELP - This will give the MOS version and the PDs present.

*HELP CUBE - If the CUBE monitor PD is included, this will list the monitor commands and syntax.

*FX 0 - This will give the date and issue number of the MOS-B.

(c) Draw a curve:

```
10 MODE 0
20 FOR X=0 TO 1023 STEP 4
30 MOVE X,0
40 DRAW 1023,X
50 NEXT

>RUN
```

(d) Save this to disk

```
>SAVE "curve"
```

(e) Try to load it:

```
>NEW
>LOAD "curve"
>LIST
```

PROGRAMMING

BBC BASIC is supplied as a 16kB ROM. When fitted to EuroBEEB, all programming facilities are supported, together with such calls to the operating system as can be supported by the hardware available. When used with the EuroBEEB Colour Teletext card, the display is compatible with that of the BBC Microcomputer in Mode 7.

A machine code assembler is included in BBC BASIC. However, we strongly recommend the use of ADE for machine code development. This ROM-based package runs in the BBC Micro and provides a full disk-based Assembler and Screen Editor (SYSTEM Software, Sheffield: ADE - 16K ROM + Manual. Available from Control Universal).

1. BASIC

Program development is carried out in the same manner as on the BBC Micro itself. The main difference is that the program is held in CMOS RAM and will thus be protected during power-down. Nevertheless, it is still good practice to save the program to disk at regular intervals to protect from accidental erasure or overwriting of the program.

Special considerations are necessary when designing for an EPROM-based program. With a RAM-based program, BASIC variables are stored immediately above the BASIC program. Thus the first line of any EPROM-based program must use the BASIC command words LOMEM and HIMEM to assign part of the RAM for use by BASIC. PAGE must point to the first EPROM address. Take an example using a standard EuroBEEB fitted with 8K CMOS RAM (0-&1FFF), 8K EPROM (&2000-&3FFF), BBC BASIC and MOSB.3. On power-up PAGE is automatically set to &0E00. Thus the first commands should be:

```
>PAGE=&2000
>RUN
```

The EPROM-based program will now run. Its first line should be:

```
10 LOMEM=&0E00:HIMEM=&2000
```

This relocates the BASIC storage area to the 8K RAM. However, every time the system is powered down, it will need the start-up commands entered again via the serial link. This is obviously unacceptable in a stand-alone target application. In this case, the commands will be placed in the Autorun line and executed automatically (see below, and also Appendixes 8 and 9 for more details).

In order to LIST the program in EPROM, HIMEM must be relocated above the program text space:

```
>PAGE=&2000
>HIMEM=&8000
>LIST
```

See Appendix 8 for details of EPROMing a BASIC program.

2. Machine Code

The on-board BASIC assembler may be used for small machine code routines. In most cases, however, it will be more efficient to use an Assembler or Cross-assembler running in a host system. The most common configuration for 6502 Assembly language programming would be ADE (SYSTEM Software, Sheffield) fitted to the BBC Micro. The source text would be prepared and assembled in the normal way on the BBC Micro, the assembled object code being sent to a disk file. This may then be down-loaded to EuroBEEB for testing, using *LOAD <file name> [<address>].

The on-board BASIC may be used to run the machine code using CALL <address>. Memory locations may be changed and examined using the BASIC indirection operators, ? and ! (see p. 409 of the BBC Micro User Guide). The optional CUBE Monitor PD provides more extensive facilities for testing and debugging machine code on EuroBEEB, including breakpoint handling. The Monitor also contains the driver routines for the CU-PROM EPROM programmer, as well as the immediate commands *CLOCK and *DATE, which send the current real-time clock value and date to the screen (see chapter on the Machine Code Monitor).

NOTE

EuroCUBE-65 without BBC BASIC fitted will only run in the development phase if the Machine Code Monitor (i.e. MOSM.3) is fitted.

RUNNING EuroBEEB IN STAND-ALONE MODE

After a program has been developed, EuroBEEB can be set up as an independent unit without a terminal. In the independent system it is necessary to have some means of calling the program after switch-on. This can be achieved by using the Autorun facility described below, which allows the system to power-up and run automatically.

Autorun Facility

In stand-alone mode, EuroBEEB can function as a 'Turnkey' system, using the Autorun command line. A string of 24 bytes, resident in the MOSB.3 EPROM, is available for Autorun operation. The required start-up commands, as specified by the customer, can be programmed into this line by Control Universal (for a small charge), or the user can program the commands himself by using an EPROM programmer.

1. Machine Code

The Autorun command line starts at &F000. The first byte provides a start-up control value similar to that provided for the BBC Micro by the 8-way jumper row located on the keyboard PCB of the BBC Micro. The next two bytes provide a vector to the language entry routine in the MOSB.3 which would normally initialise BASIC. This vector may be changed by the user to point to his own machine code routine, in which case control will be passed immediately to that address by the MOS, using an indirect jump instruction. This conforms to the normal 'low byte first' practice, e.g. to point to a user program starting at &2000.

```
F001 00
F002 20
```

2. BBC BASIC

The next 20 bytes, starting at &F003, would be used with a BASIC program. These are executed at switch-on after system initialisation as if they had been entered from the keyboard. The command text must be terminated with the value &FF as an end-of-text marker. For example:

```
PAGE=&2000<CR>  or PA.=&2000<CR>
RUN<CR>
&FF
```

This will point to a BASIC program resident in EPROM at &2000, initialise and then run that program. As a sequence of hex bytes, it would be stored as:

```
50 41 47 45 3D 26 32 30 30 30 0D
(or 50 41 2E 3D 26 32 30 30 30 0D)
52 55 4E 0D
FF
```

Any other start-up commands, such as LOMEM and HIMEM, which are necessary to relocate the BASIC variable storage area, should be contained in the first lines of the user's program.

See Appendix 9 for initialisation tables.

PART III: SOFTWARE

MACHINE OPERATING SYSTEM MOSB.3

Introduction

The EuroBEEB/EuroCUBE Operating System holds the input and output drivers, system subroutine calls, and the serial filing system.

Version 3 has been enhanced to contain the drivers required by Control BASIC. This allows the user to access digital and analog I/O from high level 'BASIC' type commands (see section on Control BASIC for more details). The software has been written to provide the necessary environment for the operation of BBC BASIC in EuroBEEB/EuroCUBE. However, the BBC BASIC is not essential, and the user interested in developing his programs in another high level language or in machine code can use the system subroutines by issuing the appropriate calls.

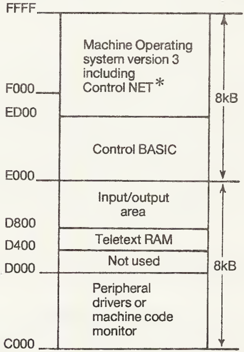
The MOS may also contain peripheral drivers which are software modules interfacing a peripheral to the EuroCUBE CPU card. Examples are:

1. System Video Drivers compatible with the language's graphics commands, i.e. drivers for either the high-resolution graphics card CU-GRAPH or the TELETEXT card.

2. Peripheral Drivers to decode external keyboards

Version 3 of the MOS represents a significant enhancement of MOSB.2 which it replaces. MOSB.3 is supplied in a 16kB 27128 EPROM. A memory map of the 16kB of the address space occupied by MOSB.3 is shown below.

Memory Map arrangement



* Control NET to be released

MOSB.3 Versions

As it is impossible to accommodate all possible peripheral drivers in a 16kB EPROM, several variants of the MOSB.3 exist. These are summarised in the table below.

MOSB.3 is a generic term representing all series 3 versions described below. There is a standard 8K section found in all of the MOS variants, consisting of the Machine Operating System, Control NET and Control BASIC. Unless a particular variant is specified at the time of ordering, the default standard will be supplied. This consists of the standard 8K section at 1 MHz and a machine code monitor (MOSM.3-1).

Notes

1. There are 1MHz and 2MHz versions of MOSB.3 to allow for correct timing of loops and operation of the centisecond clock.

   The following cards do not work at 2 MHz:

   CU-GRAPH, CUBAN-8, CU-DRAM, CY-DRAM

   All the remaining cards can be fitted with 2 MHz 'A' parts.

2. All ROMs are 16kB 27128 devices and include Control BASIC.

3. The new MOSB.3 versions are designed for the new hardware, i.e. there will be a requirement for upgrades of old hardware (CU-KEY 53).

The following table shows the range of firmware available in the MOSB.3 series:

Name	Operation	Peripheral Drivers	Comments
MOSM.3-1 MOSM.3-2	1 MHz 2 MHz	Machine code monitor incl. ATPL EPROM programmer	Facilitates debugging of machine code routines
MOSC.3-1	1 MHz	CU-GRAPH	Contains keyboard drivers for CU-KEY 99 professional keyboard. Machine code monitor only available as a sideways ROM which requires Doublestore or CU-MEM Selecta
MOSJ.3-1 MOSJ.3-2	1 MHz 2 MHz	Jobber, i.e. CU-KEY 25, Rackprint and Viewline	
MOST.3-1 MOST.3-2	1 MHz 2 MHz	Teletext	

NOTE: CU-KEY 25 is a 5x5 matrix keyboard,
Rackprint is a rack-mounting dot-matrix impact printer,
Viewline is a 24 column x 2 row rack-mounting LCD display.

These three items are interfaced to EuroCUBE with the new Jobber interface card.

CU-KEY 99 is a professional keyboard featuring both numeric and alpha-numeric pads.

Compatibility with MOSB.2

If a user wishes to upgrade a MOSB.2 to include Control BASIC (+ NETWORK) while maintaining software compatibility, then the following EPROMs are offered. These are all 27128 EPROMs (16K) and upgrade the old equivalent versions of MOSB.2 C,T,J:

MOSC.3-0	CU-GRAPH and CU-KEY 53
MOST.3-0	Teletext and CU-KEY 53
MOSJ.3-0	Old Jobber drivers

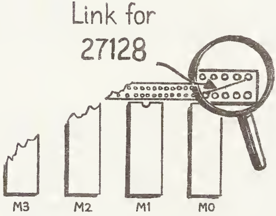
In these versions the word SAMPLE can be used to store readings as 2 byte numbers in RAM or CY-DRAM (sideways RAM).

SIDEMON

Only MOSM.3 contains a machine code monitor. If a monitor is required with MOSC.3, MOST.3 and MOSJ.3, Monitor 3.1 is available as a Paged ROM (2764) called SIDEMON. If the user wishes to dispense with BBC BASIC, the SIDEMON ROM can be plugged into memory socket M1 which is normally occupied by the BASIC ROM. However, if both BBC BASIC and SIDEMON are required, both ROMs must be plugged into sideways slots on either CUMEM Selecta (sideways RAM/ROM board) or Doublestore (FDC card).

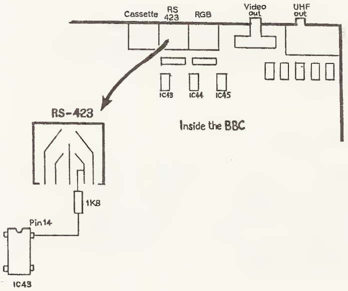
Using 27128 Devices with Older Issue Boards

Users will find that MOSB.3 is resident in a 16kB 27128 device. This means that a link needs to be modified on memory socket M0 of all EuroBEEB boards preceding Issue 7. From Issue 7 onwards, this link is fitted as standard. Both a 2764 and a 27128 device will operate with the link fitted.



Networking

MOSB.3 is a networking operating system. This means that when the serial output is switched off, it goes tri-state. This causes the CTS input to the BBC terminal to go off. In this state, the terminal will not transmit. This problem is easily solved by fitting a biasing resistor to the terminal. A value of 1K8 from the CTS input to +5V causes the input to appear ON. This modification must be made if the BBC Micro is used with MOSB.3. The resistor has no effect on normal operation.



Details of RS-423 port

Extra stations are added by paralleling up the serial cable. All stations should have different station numbers. A station number is programmed into the EPROM at location &F018 (just after the turnkey line). All MOSB.3 are supplied as standard with station number 0. To change this number, the user must reprogram and reblow the EPROM. Full details of the Network commands are provided in the chapter on Networking (in preparation).

*EURO

Users of the BBC Micro as a terminal will require a *EURO 3.1 EPROM fitted to the BBC. Please note that *EURO 3.1 is only compatible with MOSB.3 and its variants.

PROM Decoder

Please note that EuroBEEB must be fitted with PROM decoder M4. MOSB.3 assumes a 27128 socket between &C000 and &FFFF. If a 2764 is present, the systems detects this and does not attempt to enter the peripheral drivers at &C000. PROM decoder M3 was designed for 2764 devices and therefore does not work with MOSB.3.

Operating System Calls

NOTE: The user should refer to the Advanced User Guide for the BBC Micro for a more detailed description of these calls.

*FX Calls

Operating system calls are aimed primarily at controlling the system hardware without resorting to machine code routines. A subroutine call can pass parameters in many ways. The 6502 has A, X and Y 8-bit registers. The simplest subroutine call just uses A, X and Y to pass and return parameters. This would require a large list of predetermined entry points for all the necessary calls. To simplify this, only one call of this type is used, where A specifies which subroutine is required and X and Y form the parameters. In the BBC computer and in MOSB.3 this type of subroutine is called an OSBYTE (operating system byte call).

These 'function' calls take the form:

```
*FX <A> [,<X> [,<Y>]]
```

where X and Y are optional and default to zero.

They are accessed from BASIC as shown above, e.g. *FX 3,1 (enable RS-423 driver). They can also be accessed directly from machine code, using the OSBYTE (&FFF4) passing A, X and Y from the registers of the 6502, e.g. using the same call:

```
LDA#3 : LDX#1 : JSR &FFF4
```

OSBYTE Calls Implemented in MOSB.2/MOSB.3

Summary

dec	hex	function
0	0	Print operating system issue
1	1	User OSBYTE, read/write &281
2	2	Select input channel
3	3	Select output channels
7	7	Select RS-423 baud rate
8	8	as 7
13	D	Disable events
14	E	Enable events
15	F	Flush selected buffers
21	15	Flush specific buffer

0-21 make Y=0

22-116 are not used by OS

dec	hex	function
119	77	Close any SPOOL or EXEC files
124	7C	Clear ESCAPE condition
125	7D	Set ESCAPE condition
126	7E	Acknowledge ESCAPE
127	7F	Check for End of File
128	80	Get buffer status/Analog conversion
129	81	Read input channel within time limit
130	82	Read machine high order address

dec	hex	function
131	83	Read top of OS RAM (PAGE)
132	84	Read top of program RAM (HIMEM)
133	85	as 132
136	88	Perform *CODE
138	8A	Insert character into buffer
142	8E	Enter language ROM
143	8F	Issue PAGED ROM service
145	91	Get character from buffer
150	96	Read from I/O &FE00
151	97	Write to I/O &FE00
152	98	Examine buffer status
153	99	Insert character into input buffer
156	9C	Read/Write 6551 Control reg
166-255	A6-FF	Read/write OS RAM variables &236-&28F

The following is a brief summary of the changes between MOSB.2 and MOSB.3:

*FX22 is a digital bit handler.

*FX23 is digital port initialisation.

*FX155,156 is read/write UART command and control registers respectively.

*FX247 contains the station's own number

*FX248 contains the network's currently selected station number

*FX249,250,251 are CMOS battery-backed variables not affected by resets.

The following variable calls have been modified:

&C0	6551 (ACIA) Control Register
&E7	INT. mask for CLOCK
&EB	CLOCK present flag
&F2	6551 (ACIA) Command Register
&F3	ROM 16 start page (for peripheral drivers)

OSWORD Calls

When subroutines require more than two parameters, the OSBYTE call is inadequate. In this situation the OSWORD call is used. This has a parameter block in RAM which is pointed to by the X and Y registers. The X (low byte) and Y (high byte) form a 16-bit address. The parameter block can contain as much data as is required. The A register again specifies which call is being used.

From BASIC

```
DIM block 15
X% = block
Y% = block DIV 256
A% = 14 : REM read clock
CALL OSWORD
```

Users should consult the User Guide or Advanced User Guide for a full description of OSWORDS.

Calls 0 to 4 are implemented and work as described in the BBC User Guides.

Calls 14 and 15 will read and write to the CLOCK if it is fitted (Issue 5 EuroBEEB/EuroCUBE onwards). The parameter block is described in the section on the Real Time Clock. Note that a null value, &FF, can be entered when writing to the CLOCK, in order to leave certain values unchanged.

In MOSB.3 the following OSWORDs are also implemented:

OSWORD 5,6 fifth parameter specifies paged ROM number.

OSWORD 16 starts sampling sequence.

OSWORD 17 reads back a sample.

OSWORD 18 write to a D/A converter.

Filing System: Available Commands

*SPOOL
*EXEC
*HELP
*FX
*LOAD
*SAVE
*CAT
*OPT
*CODE

Any unrecognised command will be passed back to the BBC Micro for processing. For example, *DRIVE 1 will cause drive 1 to be selected.

Paged ROMs

MOSB.3 supports Paged ROMs. A new version of CUMEM, CUMEM Selecta, has been developed to provide hardware support for ROM paging. This will allow up to 8 ROM/RAM devices to be paged into the 16K language space at &8000 to &BFFF. The use of battery-backed CMOS RAM will also be possible.

Interrupts: IRQ Handler

Before attempting to use interrupts, the user should be thoroughly familiar with Chapters 12 and 13 of the Advanced User Guide for the BBC Micro. In particular, users should note that interrupt service routine which are initiated by intercepting IRQ1V or IRQ2V:

1. should not in general end in RTI.
This is handled by the Operating System, so JSR (OLDVEC) is preferred. OLDVEC contains a copy of the original contents of the interrupt vector.

2. should not in general make OS calls.
If it is essential to use an OS call, check very carefully what it does before using it.

IRQ1V (&204)	Points to MOS-B IRQ handler which carries out:

1. Check ACIA receive, transmit, and handshake
2. Check VIA Timer T1
3. Check CLOCK
4. JMP (IRQ2V)

IRQV2 (&206)	Points to RTI (Vector to user IRQ routine may be placed here)

Events

Events provide the user with a pre-packaged interrupt. The Operating System may generate an event (see list below), and the user is then able to add his own code to the system interrupt service routine. An example of this is given in the Timer program in the Appendix.

Event Number	Cause of Event
0	output buffer empty
1	input buffer full
2	character enters input buffer
3	ADC conversion complete (not implemented)
4	start V. sync.(not implemented)
5	event timer at zero
6	ESC condition detected
7	RS-423 error detected (lower 3 bits of status of ACIA in X register, serial character in Y register)
8	CLOCK event - status register of CLOCK in X
9	User event

Reset

Reset can only be performed on EuroBeeb using <shift f9> which generates a Break condition on the serial input line. The BREAK key on the BBC Micro will have no effect on EuroBeeb. The Reset sequence is as follows:

- RTI is placed at &D00
- SEI is executed
- Reset system vectors in RAM
- Initialise system variables
  - WARMstart : OSBYTEs &A6-&EE
    All flags and pointers except TIME
  - COLDstart : OSBYTEs &A6-&FF
    All flags and pointers
- Initialise OS zero page workspace
- Perform software Reset on VIA
- Initialise ACIA and reset BUSY line to allow serial data input
- Test for CLOCK present and set flag
- Initialise Paged ROMs and load type table to RAM.
- Make service calls to Paged ROMs and claim workspace
- Load startup string to output buffer.
- Make initialisation service call to Paged ROMs
- CLI is executed
- Enter Language ROM
  - WARMstart : Current Language
  - COLDstart : Language with highest socket value

The following are further changes inplemented in MOSB.3:

OSARGS and OSGBPB are serial filing commands added.

PTR#, EXT# now work from BASIC.
FORTH uses OSGBPB to load screens.

Event 5

Y=0	centisecond event
Y=&FF	event timer crossing zero

Keyboard Vector

KEYV which is used by the OS for keyboard access is implemented for CU-KEY 99.

ROMs Currently Supported

BBC BASIC
DDFS
FORTH (modified Skywave FORTH)
EXMON (Beebug)

Control BASIC Extension

Introduction

Control BASIC is supplied as standard in all MOSB.3 versions for EuroBEEB and EuroCUBE-65, i.e. MOSM.3, MOST.3, MOSC.3 and MOSJ.3. It occupies almost 4K of code in the OS EPROM from &E000 upwards.

Simple BASIC type words such as SAMPLE, DELAY, etc, are included in the Control BASIC extension to simplify the interface of high-level language to hardware. Users have appreciated the implementation of "ADVAL", and these extra words go further along the same line.

If the BASIC Interpreter does not recognise a word, it attempts to look it up in the variable list. If the word cannot be found there, it is passed on to the Control BASIC extension. The word is tested to see whether it belongs to the extension. This produces a convenient, if slow, method of extending the BASIC Interpreter.

Channel Identification

Control BASIC words work on "channels" of which there are two types: those concerned with digital access and those concerned with analog access.

1. Digital Access

The digital commands relate to a Versatile Interface Adaptor (VIA) which can be positioned anywhere within the memory. The ports on this VIA are described in Control BASIC as 'channels', with Port A representing channels 0 - 7 and Port B channels 8 - 15. If one or more VIAs are added, up to a maximum of 16, the VIA addressing must be contiguous. The second VIA would take channel numbers 16 to 31, etc. The maximum number of channels is 256 (0 to 255 inclusive).

If the VIAs are not contiguous in memory they can still be used, but the pseudo variable BASE. must be redefined if the VIA to be accessed is changed.

2. Analog Access

The analog inputs/outputs are treated in a similar manner to the digital inputs/outputs.

Analog/digital conversion

			Board decode
channels	0 - 7	for 1st CUBAN-12	&DC00
"	8 - 15	" 2nd CUBAN-12	&DC20
"	16 - 23	" 3rd CUBAN-12	&DC40
"	24 - 31	" 4th CUBAN-12	&DC60
"	32 - 63	(for Integrating CUBAN-12)	
"	64 - 79	" 1st CUBAN-8	&DB00
"	80 - 95	" 2nd CUBAN-8	&DB40
"	96 - 111	" 3rd CUBAN-8	&DB80
"	112 - 127	" 4th CUBAN-8	&DBC0

Digital/analog conversion

channels	0 - 3	for 1st CUBAN-12	&DC00
"	4 - 7	" 2nd CUBAN-12	&DC20
"	8 - 11	" 3rd CUBAN-12	&DC40
"	12 - 15	" 4th CUBAN-12	&DC60
"	16 - 63	(for Integrating CUBAN-12)	
"	64	for 1st CUBAN-8	&DB00
"	65	" 2nd CUBAN-8	&DB40
"	66	" 3rd CUBAN-8	&DB80
"	67	" 4th CUBAN-8	&DBC0

ON/OFF and Logic Levels

At RESET, all VIA channels default to input, and their output lines take the 'logic high state' (+5V). To this effect, 5V is defined as OFF (FALSE) and 0V is defined as ON (TRUE). This represents a logical inversion, but has been implemented in this manner so that all outputs are turned off at RESET. The maximum channel number is recorded, and the error 'Bad channel' will result if this is exceeded.

Summary

Logic state	Output voltage	ON/OFF in Control BASIC	TRUE/FALSE (BASIC)
1	+5V	OFF	FALSE (0)
0	0V	ON	TRUE (-1)

Scaling

All analog values are treated as 16-bit variables and are scaled appropriately. For example, a 12-bit analog input will appear to have a reading between 0 and 65520 (&FFF0) and will increment in steps of 16. The 8-bit converter input will have the same scale but will increment in steps of 256. The same is true of the analog output channels. This means that the language mathematics is the same for 8-bit and 12-bit operations, since the scaling has already been carried out.

Use of the ID Character

Control BASIC words may require an ID character to identify them, depending on the type of statement. This character serves to distinguish the Control BASIC word from a BASIC variable. It can be any non-alphanumeric character such as (, & % # etc. The ID character is ONLY required when the BASIC extension word is on the left-hand side of an assignment statement, i.e. on the left-hand side of an equals sign.

```
Examples    OUT.4  = 1
            DAC#3  = &F0
            DATE$~ = "05:3:85"
```

For simplicity and appearance we suggest that the full-stop character "." is used. Please note that the underline character "_", which is often used in procedure names in BBC BASIC, is also treated as an alphanumeric by the Control BASIC extension. Therefore, it should NOT be used as an ID character.

If the extension BASIC word is NOT on the left-hand side of an assignment statement, the ID character must NOT be used.

```
Examples    C$ = CLOCK$
            PRINT WEEK
```

If the ID character is not used, e.g.

```
            OUT4 = 1
```

BASIC will assign the value 1 to the variable OUT4

Control BASIC Extension Words

Control BASIC Extension Words have four variations:

1. Commands with respect to a VIA stack
2. Analog I/O commands
3. Real-Time Clock commands
4. Timing command

Summary of Extension Words

Digital

BASE.=<address>

OUTCH <channels>

INCH <channels>

TURNON <channels>

TURNOFF <channels>

FLIP <channels>

OUT.<channel>=<num-var>

<num-var>=IN<channel>

Analog

DAC.<channel>=<num-var>

<num-var>=ADVAL<channel>

SAMPLE <number>,<interval>,<address>,<channel>{channel}

<num-var>=SAMPLE <number>,<channel>

Real-time Clock Commands

CLOCK$.=<string-var>

DATE$.=<string-var>

DAY.=<num-var>

WEEK.=<num-var>

<string-var>=CLOCK$
=DATE$

<num-var> =DAY
=WEEK

Timing Command

DELAY <centiseconds>

1. Commands with respect to a VIA stack

BASE

Syntax: BASE. = <address>

set VIA start address

Example:

BASE. = &FE00

Sets the VIA base address at &FE00 which is the VIA on the CPU card. (This is the default value.)

NOTE:

(a) An ID character, in this case ".", must be used.
(b) BASE is a pseudo variable. Statements such as PRINT BASE have not been implemented.

OUTCH

Syntax: OUTCH <channels>

define channels as output

Example:

```
OUTCH 0 TO 3
```

Defines channels 0 to 3 as ouputs.

INCH

Syntax: INCH <channels>

define channels as input

Example:

```
INCH 5,6
```

Defines channels 5 and 6 as inputs.

TURNON

Syntax: TURNON <channels>

turn ON output channels

Example:

TURNON 12

Turns ON channel 12, if defined as an output. This sets the output, physically, to 0V. See note on logical inversion of digital I/O.

TURNOFF

Syntax: TURNOFF <channels>

turn OFF output channels

Example:

TURNOFF 10,13 TO 15

Turns OFF channels 10,13,14 and 15, i.e. these channels will physically output a nominal output of +5V (i.e. logic high) if defined as outputs.

FLIP

Syntax: FLIP <channels>

invert state of output channels

Example:

```
FLIP 2 TO 4
```

Will change the state of channels 2,3 and 4 from ON to OFF or vice versa.

OUT

Syntax: OUT. <channel> = <numeric>

write TRUE or FALSE

Examples:

```
OUT. 12 = TRUE
OUT. 12 = 1
OUT. 12 = &FF
```

All three statements turn ON channel 12 if the channel is defined as an output.

```
OUT. 13 = Valve%
```

Will turn channel 13 ON or OFF depending on the value of the variable Valve%. If Valve% is zero (or FALSE), the channel will be turned OFF (output voltage set high). If Valve% does not equal zero (TRUE), the channel will be turned ON (output voltage 0V).

Note the use of the ID character with OUT.

IN

Syntax: <num-var> = IN <channel>

read TRUE or FALSE

Example:

```
Switch-state% = IN 3
```

The variable Switch-state% takes the value of -1 (TRUE) if the input channel 3 is ON (physically 0V on the input). Alternatively, the variable Switch-state% takes the value of 0 (FALSE) if the input channel is in the OFF state (nominally +5V).

NOTES:

(a) All channels default to inputs.

(b) <channels> can refer to individual channels, a group of contiguous channels, or a combination of both.

(c) <address> refers to a decimal number. Hex numbers must be prefixed with '&'.

(d) Use of the VIA ports
Users should note that when a channel is defined as an output, the output takes the value of whatever happens to be in the corresponding bit of the output register ORA or ORB at the time. The contents of these registers are not changed if the port is changed to an input, since the input registers IRA and IRB are physically distinct from the output registers ORA and ORB. Therefore it is sound practice to define the output registers before defining a channel as an output.

For example, the sequence

```
2000 TURNOFF 0 TO 7
2010 OUTCH 0 TO 7
```

would be preferable to sequence

```
2000 OUTCH 0 TO 7
2010 TURNOFF 0 TO 7
```

because in the latter sequence, after the execution of the OUTCH statement, the outputs are temporarily unknown, unless the program has kept track of the state of the outputs. Depending on the actual hardware configuration, this could be highly undesirable if not actually disastrous.

(e) Only Port A and Port B in the VIA are supported.

Summary of VIA address and channels

Name	VIA offset	Channels
Port A	1	0 to 7
Port B	0	8 to 15

VIAs are assumed to be at &10 intervals up to a maximum of BASE + &F0

2. Analog I/O Commands

DAC

Syntax: DAC. <channel> = <num-var>

write analog value

Example 1:

DAC. 3 = &F0

will send the hex value F0 to the digital to analog converter which in this case will be on the first CUBAN-12. The actual output voltage will be scaled to a 16 bit variable (max &FFF0) as a fraction of the reference voltage.

Example 2:

DAC. 5 = output-voltage

will cause the DAC to output a voltage whose value depends on the BASIC variable output-voltage.

ADVAL

ADVAL is a BBC BASIC statement that allows analog readings to be read into BASIC variables. It has been implemented for CUBAN-8 and CUBAN-12, the CUBE 8-bit and 12-bit analog to digital converter cards, and uses the OSBYTE 128 call.

Syntax: <num-var> = ADVAL <channel>

read analog value

where <num-var> can be any BBC variable and <channel> is the channel number on the device in question.

Example:

```
Depth%= ADVAL(64)
```

will read the scaled value of the analog input to channel 64 (1st channel of CUBAN-8 at &DB00).

ADVAL in BASIC is equivalent to:

OSBYTE 128   read ADC channel

where

X = channel 0 - 127

Y is ignored

This returns a 16-bit converter value in X,Y (X = LSB, Y = MSB) read from the channel specified by X.

Channels 0 to 31 provide access to four CUBAN-12 cards, and channels 64-127 to four CUBAN-8 cards. Channels 32 to 63 are not used.

The address decode on the CUBAN-8 or CUBAN-12 must be set as follows:

CUBAN-12	DC00	channels	0-7
	DC20	"	8-15
	DC40	"	16-23
	DC60	"	24-31
CUBAN-8	DB10	"	64-79
	DB50	"	80-95
	DB90	"	96-111
	DBE0	"	112-127

To write to an analog output channel OSWORD 18 is used.

SAMPLE

Syntax:

SAMPLE <number>,<interval>,<address>,<channel>
{TO <channel>}

defines parameters and starts Sampler

Example:

```
SAMPLE 1000,1024,&2000, 3
```

will read 1000 samples at intervals of 1024 microseconds from channel 3. These values will be stored in memory starting at location &2000. The readings are stored as 2 byte integers.

Syntax: <num-var> = SAMPLE <number>,<channel>

read back a sample

Example:

```
Temperature= SAMPLE 204, 3
```

assigns the value of the 204th sample taken on channel 3 to the variable Temperature.

NOTES

1. Sampling takes place as a background task, i.e. the main program continues and the samples are fed into the appropriate memory locations as they are taken under interrupt. For a complete description of the sampling process refer to the OSBYTE/OSWORD command summary.

2. Samples may be stored in sideways RAM using either CY-DRAM or CUMEM Selecta. Please refer to the OSBYTE summary for further details.

3. Real-Time Clock Commands

CLOCK$

Syntax: CLOCK$.=<string-var>

sets the real-time clock (hours, minutes and seconds)

Example:

```
CLOCK$.= "23:9:30"
```

sets the clock to 23 hours 9 minutes and 30 seconds.

Syntax: <string-var>=CLOCK$

reads the real-time clock (hours, minutes and seconds) to a BASIC string variable

Examples:

```
PRINT CLOCK$
```

prints a string in the form hh:mm:ss representing the current time.

```
PRINT LEFT$(CLOCK$,2);" hours"
```

DATE$

Syntax: DATE$. =<string-var>

sets the current date as date,month, year

Example:

```
DATE$.="04:11:85"
```

sets the date to the 4th of November 1985.

Syntax: <string-var>=DATE$

reads the current date into a string variable

Example:

```
Year% = EVAL(RIGHT$(DATE$,2))
```

DAY

Syntax: DAY.=<num-var>

sets the current day in the week in the real-time clock

Example:

```
DAY.= 3
```

Syntax: <num-var>=DAY

reads a number representing the current day in the week from the real-time clock

Example:

```
10 DIM day$(7)
20 FOR I%=1 TO 7
30 READ day$(I%)
40 NEXT
50 PRINT TAB (20,0);"Today is ";day$(DAY);"day"
60 END
70 DATA Mon,Tues,Wednes,Thurs,Fri,Satur,Sun

>RUN
Today is Friday
>
```

WEEK

Syntax: WEEK. =<num-var>

Example:

```
WEEK.=23
```

makes the current week number 23, as stored in the real-time clock.

Syntax: <num-var>= WEEK

assigns the value of the WEEK register in the real-time clock to a BASIC numeric variable.

Example:

```
This-week=WEEK
```

NOTE: The ALARM function is not implemented.

4. Timing Command

DELAY

Syntax: DELAY <centiseconds>

causes a delay of a given number of centiseconds

Example:

```
DELAY 250
```

causes a delay of 250 centiseconds (or 2.5 seconds) before the next instruction is executed.

Real-time BASIC

For critical applications, we strongly recommend the use of Real-time BASIC which is a fully tokenised extension. The new statements and functions found in Control BASIC are also found in Real-time BASIC so that programs written in Control BASIC can easily be converted to run in Real-time BASIC with the advantage of a much higher speed of execution. The typical execution time of a statement in Control BASIC is 18 ms, whereas in Real-time BASIC this is reduced to 1.8 ms - virtually indistinguishable from BBC BASIC itself.

As the title suggests, Real-time BASIC also offers time-related statements such as

```
WHEN clock$ = "5:30:00" PROCgohome

WHEN KEY    = 13 PROCcarriage-return
```

Real-time BASIC is available only as a sideways ROM. Sideways ROMs may run either on CUMEM Selecta (a sideways RAM/ROM board) or Doublestore (a floppy disk controller card with a sideways facility).

OSBYTE 22 (&16)

digital channel control

X = channel

Y register contains a number specifying the action required:

0 turn OFF
1 turn ON
2 define channel as input
3 define channel as output
4 invert channel
5 read a channel

When asked to 'read a channel', X returns TRUE (1) or FALSE (0).

OSBYTE 23 (&17)

define VIA base

On entry, X,Y point to an address which is the base address of a VIA, allowing the VIA to be placed anywhere in memory.

OSBYTE 128 (&80)

read analog input (equivalent to ADVAL in BASIC)

Syntax: <num-var> = ADVAL <channel>

X = channel 0 - 127

Y ignored

Returns 16-bit converter value in X,Y (X = LSB, Y = MSB).

OSWORD 16 (&10)

start a sample sequence

SAMPLE N%,R%,S%,K%

OSWORD 16 calls up the Sample Vector SAMPLV to initialise the sample sequence when Timer 1 is loaded with the correct interval and interrupts are enabled.

The sampling itself takes place under interrupt. The sample Vector is called with A=2 (take a sample). On return the number of samples is compared to the total required. The working counter is incremented unless it is equal to the number of samples required, in which case the sampling sequence is complete.

The sample data can be read with OSWORD 17.

For example:

```
SAMPLE 1000,100,&4000,3
```

Sample Channel 3 1000 times at an interval of 100 µs and store the results from starting address &4000.

Parameter block		Integer byte no.	
XY →	integer N%	0, 1, 2, 3	number of samples (0 to 32,000 with CY-DRAM)
	integer R%	0, 1, 2, 3	sampling interval (100 - 2↑31 μs)
	integer S%	0, 1, 2, 3	start address for data
	channel specifiers K%	0	first channel no. for ADC
		1	last channel

N% number of samples 0 - N% inclusive

N%, the number of samples, can be from 0 to 2,147,483,647. Please note that each sample requires 2 bytes of RAM for data storage. The maximum number of samples is therefore determined by the total RAM. For example, for 16K of RAM, N% could be 0 to 8192. When using one CY-DRAM, this number could be up to 32K.

R% sampling interval

The interval can be programmed to values from 100 us to 2↑31 µs (approximately 30 minutes). The sample program works in two modes. Intervals greater than, or equal to, 1024 µs produce a background sampling operation using interrupts. For example, if the program specified that 1000 samples be taken at an interval of 1 second, the task would take more than 15 minutes to complete. However, because the task is being performed under interrupt, the main program can continue to run. OSWORD 18 provides a method of testing for end of sampling sequence (see under OSWORD 18).

The 'Bad rate' error is produced on the screen if the interval for a single channel is LESS than 100 µs. 'Bad rate' is also produced if more than one channel is specified AND the sampling interval is LESS than 1024 µs.

Intervals exceeding 1024 µs would probably be used for data acquisition, where data is collected at periodic intervals for later analysis.

If the intervals are programmed to be less than 1024 µs, the Sampler carries out its task immediately, disabling interrupts. For example, if 1000 samples are taken at 1000 µs intervals, the program will stop for 1 second while the samples are being taken. During this time, all other interrupts will be ignored, BASIC TIME will be stopped, and any information coming in from the serial port will be missed.

Also, when the interval is less than 1024 µs, the converter always fills up the memory to a RAM page boundary, starting a new page every time 128 samples have been collected.

Intervals of less than 1024 µs are used for high-speed waveform sampling. A rate of 10 kHz (100 µs) should allow speech to be sampled. For example, 32000 samples collected at an interval of 100 µs would produce 3 seconds of speech.

Minimum Times against Number of Channels

0-1023 µs	1 channel
1024 µs	2 channels
2048 µs	4 channels
4096 µs	8 channels

PLEASE NOTE: Less than 1024 µs will NOT allow more than one CY-DRAM card to be used.

The above restrictions are physical limitations of the 1 MHz 6502 microprocessor, and the speed at which it is able to execute instructions.

S%   start address for data

1. Normal RAM

start address   0-&FFFF   normal 6502 memory space

data saved at address
address is incremented after each sample

For example:

    Save 200 samples into &4000
    N% = 200
    Start address = &4000

The address will increment up to &4000 + 200*2.

2. CY-DRAM

Available only on the following MOS versions: MOSM.3-0, MOST.3-0, MOSC.3-0, and MOSJ.3-0.

CY-DRAM is a paged RAM existing from &4000-&7FFF. For each CY-DRAM card, four 16K blocks are present. These can be individually switched into the above memory space.

The sampling software automatically deals with the CY-DRAM switching. The extra memory is defined as starting at &10000.

```
&10000 - &1FFFF    first CY-DRAM (switch at 0)
&20000 - &2FFFF   second CY-DRAM (switch at 1)
etc.
```

For example, save 32000 readings in CY-DRAM:

```
SAMPLE 32000,100,&10000,3
N% = 32000
R% = 100µs     sampling interval
S% = &10000    start at beginning of CY-DRAM
```

The Sampler always leaves the CY-DRAM pointing to its first 16K block (e.g. &10000-&14000). This can be used as normal RAM (e.g. &4000-&7FFF), if the Sampler is only used from &14000 onwards.

Only one CY-DRAM can be used for intervals of less than 1024 µs.

3. Sideways RAM

Sideways RAM on CUMEM Selecta (sideways RAM/ROM board) exists from 8000-BFFF.

Using option 0, CUMEM Selecta can take eight 8kB RAMs which appear as four 16kB blocks. With SW2 set to position 0, these 16kB blocks appear as Pages 0,1,2,3.

The sampling software automatically deals with the CUMEM Selecta switching. The extra memory is defined as starting at &10000.

```
&10000 - &1FFFF    first 16kB block
&20000 - &2FFFF   second 16kB block
etc.
```

For example, save 32000 readings in CUMEM Selecta:

SAMPLE 32000,100,&10000,3

```
N% = 32000
R% = 100µs     sampling interval
S% = &10000    start at beginning of CUMEM Selecta.
```

K% channel specifiers

For example: channel = 5
SAMPLE 1000,100,&4000,5

(a) K%?0 (byte 0) first channel number for ADC

The peripheral driver actually calls the ADVAL (OSBYTE 128) command when taking a sample. The channel number is the same for ADVAL. ADVAL is a utility that allows the BBC BASIC command ADVAL to operate with CUBAN-8 and CUBAN-12 (see under ADVAL).

(b) K%?1 (byte 1) last channel

For example:

```
First channel = 5
Last channel  = 7
SAMPLE 1000,1024,&4000,5 TO 7
```

If the last channel is less than or equal to the first channel, only the first channel is sampled. For example, if T%?1=0, the last channel is ignored. If the last channel is greater than the first channel and the interval exceeds 1024 µs, multi-channel sampling will occur. The readings from the first to the last channel inclusive will be stored from the start address at sampling times.

NOTE: As discussed, the 'Bad rate' error is produced if more than one channel is specified, and and the sampling interval is less than 1024 µs. Consequently, high frequency multi-channel sampling is NOT possible.

In any case, the 6502 microprocessor would not be able to cope with that speed. In practice, five channels can be sampled at 1024 µs, ten at 2048, etc.

OSWORD 17 (&11)   read back a sample

Sets A = 1 and calls SAMPLV (read a sample). X,Y points to the OSWORD's parameter block.

Syntax: <num-var> = SAMPLE N%, <channel>

```
XY ->   N%        0
                  1
                  2   sample required    0 - 2↑31
                  3

     channel      0

     result%      0
                  1
                  2   reading returned in result%
                  3
```

If a reading has not been taken, result% will equal -1 (&FFFFFFFF).

For example, if 200 samples were being taken at 1 second intervals, one could test for the end of conversion by calling OSWORD 17 with N% = 200, until the result% becomes positive. If N% equals 7 and OSWORD 17 was called, result% would equal -1 for the first 7 seconds. The result would become valid once the seventh sample had been taken.

The readback channel is compared with the specified channels when the sampler was invoked. An error message 'Bad channel' will result if the channels do not match.

Examples:

(a)
```
SAMPLE 1000,100,&4000,3
<nv> = SAMPLE 24,3
```

(b)
```
SAMPLE 1000,2000,&4000,3 TO 4
<nvl> = SAMPLE 50,3
<nv2> = SAMPLE 50,4
```

If the user wishes to call the OSWORD directly, he must set up the parameter block, i.e.:

```
.block  EQUD & 00 00 00 80   ;number of samples

        EQUD & 00 00 06 00   ;sampling interval (µs)

        EQUD & 00 00 20 00   ;start address

        EQUD & 00 40         ;channel 64

.start  LDX # block MOD 256
        LDY # block DIV 256
        LDA # &10
        JSR   OSWORD
        RTS
```

The sampling itself takes place under interrupt.

OSWORD 18 (&12)   analog output channel

Syntax: DAC# <channel> = <num-var>

Parameter block:

```
XY →    0   channel
        1   least significant data to be output
        2   most significant data to be output
```

```
   DAC channels 0 - 3     for 1st CUBAN-12
    "      "    4 - 7     for 2nd CUBAN-12
    "      "    8 - 11    for 3rd CUBAN-12
    "      "   12 - 15    for 4th CUBAN-12
           "   64 - 67    for 1st to 4th CUBAN-8
```

If the OSWORD is used, the user must set up the parameter block. For example, to write to the DAC on CUBAN-8 in 6502 machine code:

```
       LDX  # block MOD 256
       LDY  # block DIV 256

.data  EQUB &64   ;channel number

       EQUB &00   ;low data byte

       EQUB &FF   ;high data byte

       LDA  # &12
       JSR OSWORD
```

SAMPLE VECTOR, SAMPLV

The Sample Vector, SAMPLV (equivalent to INDV1 on the BBC), is used in conjunction with OSWORDS 16 and 17. SAMPLV indirects through &230.

Its action depends on the accumulator contents on entry i.e.

A=1 Read a sample

where X,Y point to a parameter block defined for OSWORD 17.

A=2 Take a sample

X,Y point to SAMBLK (described above in OSWORD 16).

A=3 initialize a sample sequence

X,Y point to SAMBLK (described above in OSWORD 16).

PERIPHERAL DRIVERS

Introduction

The EuroCUBE-65 CPU can drive a number of peripherals, thus giving the user a wide choice for his applications. Examples of popular peripherals include:

CU-GRAPH (high-resolution graphics display)

Teletext (a simpler text display with chunky graphics)

CU-KEY (QWERTY keyboard)

Although this flexibility is desirable from the user's point of view, it poses problems for the designer of the machine operating software. To cater for all the possible hardware combinations, the same number of combinations would have to be provided for in the operating system, which would then require more EPROM space than there is available. This problem can be overcome by standardising the operating system software for the CPU card and separating out the software modules controlling the individual peripherals. These software modules are known as peripheral drivers.

The following are standard options:

1) Machine Code Monitor	(MOSM.3 1 & 2 MHz)
2) CU-GRAPH + CU-KEY 99	(MOSC.3 1MHz)
3) Teletext + CU-KEY 99	(MOST.3 1 & 2 MHz)
4) Viewline/Rackprint/CU-KEY 25	(MOSJ.3 1 & 2 MHz)

All versions include drivers for the CUBAN-8 and CUBAN-12 ADC boards, invoked by the ADVAL(x) command in BASIC.

The operating system and the peripheral drivers together provide the interface between the language (in this case BBC BASIC) and the hardware present. The advantage for the user is that he can write his application software in a manner that is largely independent of the hardware.

Operation

Commands not understood by the operating system are passed to the peripheral driver(s) for action. The following example will serve to illustrate the sequence of events in response to a command to evaluate the voltage in Channel 3:

e.g. volts = ADVAL (3)

Sequence:

1) The language has interpreted ADVAL, evaluated the channel and called the operating system, using the OSBYTE call with
   A=128
   X=channel (in this case 3)

2) The operating system looks up this OSBYTE call; cannot find it and so offers it to the peripheral driver.

3) The peripheral driver recognises it and actions it.

4) The hardware converts the voltage in channel 3 into a digital 16 bit word.

5) The peripheral driver reads the resulting word and returns the word back to the operating system.

6) The operating system returns the words back to the language.

7) The language makes volts = returned word.

The Effect of RESET

When EuroBEEB is reset (using SHIFT f9), the peripherals are also given hardware reset signals. However, they may also be given software reset signals. Peripheral cards are tested for their presence and flagged back to the operating system. The flags are then used to stop calls being made to non-existent cards. The reset gives the peripheral the chance to claim RAM space, set up its variables and modify any operating system vectors, e.g. the VDU Vector (VDUV) in the case of CU-GRAPH.

The peripheral driver software is implemented using the standard Paged ROM structure (see Chapter 15 of the Advanced User Guide for the BBC Microcomputer for more details):

00	No operation - used to accept call
01	RAM claim
02	Private RAM claim
03	Initialise peripheral
04	Unrecognised command
05	Unrecognised interrupt
06	BRK inform
07	Unrecognised OSBYTE
08	Unrecognised OSWORD
09	HELP
0A	Static workspace
0B	NMI release
0C	NMI claim
0D	Onward for filing system

For example, ADVAL uses 3 and 7, the Monitor uses 4 and 9.

Peripheral drivers can also intercept calls from any of the other system vectors, e.g.

1) Call any standard operating functions or calls

2) Divert output stream, e.g. unrecognised VDU channel

3) Insert values into keyboard buffers

4) Modify system variables and vectors

The CUBE Monitor

The CUBE Monitor is designed to help users develop machine code programs. It can be used to examine memory contents and the state of I/O ports. The CUBE Monitor, supplied in MOSM.3, comes as standard with EuroBEEB and EuroCUBE-65 unless another MOSB.3 variant is specified. Only MOSM.3 contains a machine code monitor. If a monitor is required with any of the other MOSB.3 variants, then the Monitor is also available as a Paged ROM (2764) called SIDEMON (see chapter on MOSB.3).

The presence of the Monitor PD can be verified by typing *HELP when you have accessed the EuroBEEB. The following message should be displayed:

```
*HELP

   Control BASIC
   Analogue
   CUBE Monitor 3.1
      CUBE

   MOSM.3 #0  8K
```

The Monitor has a language entry point as well as a service entry point. It uses service calls 03, 04 and 09:

03 Initialise
04 Unrecognised command
09 HELP

Initialise deals with the breakpoints and gives warm and cold breakpoint initialisation.

The HELP call gives access to the Monitor commands. Type *HELP CUBE (or the abbreviation *H..), and you should see the following list of Monitor commands on the screen:

```
HELP CUBE

 MEX <adr>
  SP <hex> [,<hex>]
  "  <string> ["]
  H  hold
  CR next
  -  previous

 RDUMP   <adr>
 CRC    <adr1>, <adr2>
 RTEST / FILL    <adr1>,<adr2>[,<data>]
 RCOPY  / RVERIFY <adr1>,<adr2>,<length>
 GOSUB / GOTO / CALL  [<adr>]
 BREAK  [<no>] [,<adr>]]
 REGS  / BRKS
 REG A X Y P ST PC  <hex>
 MOFF  / MON
 NMI
 PROM
 CLOCK / DATE
```

In order to function, all these commands must be prefixed by *, e.g. *MEX<adr>. All commands can be abbreviated, and the minimum abbreviation is given for each command listed.

Definition of parameters:

<adr>	hex address
<adr1>	source address
<adr2>	destination address
<hex>	1 or 2 digit hex byte
<string>	any number of alphanumeric characters; any character is acceptable but <CR> will terminate the string
<data>	same as <hex>
<length>	4-digit hex length
<no>	single digit number
<reg>	one of the four 6502 Registers

Spaces and commas act as delimiters. The provision of data in square brackets is optional. Whenever incorrect data or non-hex numbers are entered, a 'Syntax?' error message is displayed on the screen. A range specified by two addresses is inclusive of those two addresses.

*MEX <adr>   Memory examine

Minimum abbreviation *ME.

This displays the specified address and its contents in binary, ASCII and hexadecimal, and waits for a subcommand to be given. (NOTE: It is not necessary to include leading zeroes in the address, e.g. you can type in *MEX5A instead of *MEX005A).

For example, if you type *MEX3000 you might see someting like this:

```
3000 01000100 D 44
```

The expected subcommands may be any of the following:

(1) SP <hex> [,<hex>]
    Press SPACE bar and enter new hex byte.

(2) "<string>["]
    Enter the desired string preceded by a quotation mark, and press <RETURN> to terminate it.

(3) - (minus)
    Step back to previous location.

(4) CR (CARRIAGE RETURN)
    Step on to next location.

Press <ESCAPE> to exit from the *MEX command.

(5) H (hold)

The function of hold is best demonstrated on the VIA timer. You can display the location of the VIA timer by typing in *MEXFE05. This will show you the address of the VIA and its status in binary and in hexadeximal, e.g.

FE05 00011001 OB

However, the status of the timer is continually changing, and if you wish to examine this change you can press H to display the current status at that time. If you want to obtain a record of the status changes on the screen, press <RETURN> continuously. Press <ESCAPE> to exit.

*RDUMP <adr>

Minimum abbreviation *RDU.

This displays a screenful of memory from the specified address in hexadecimal and ASCII format. If you change to a higher resolution screen mode, e.g. Mode 0, and then type *RDUMP, this will use the greater area to display more memory. This is because *RDUMP first ascertains the screen mode by sending an OSBYTE to the BBC Micro via the serial port.

*CRC <adr1>,<adr2> Cyclic Redundancy Check

Minimum abbreviation *CR.

This allows you take the 'signature' of an EPROM. For example if you type in

```
*CRC 8000 BFFF      there will be a short delay
                    before you see the signature
=4274               of the BBC BASIC ROM.
```

*FILL <adr>,<adr2>[,<data>

Minimum abbreviation *FI.

This command fills the specified memory block with the specified data. For example

```
*FILL 1000 1FFF 3
*RDUMP 1000
```

fills the specified memory block with threes and then displays the result.

*RTEST <adr1>,<adr2>[,<data>

Destructive RAM test

Minimum abbreviation *RT.

This writes through the whole area of RAM to be tested with the chosen byte and then verifies what it has done. This process is carried out in a total of six write and verify cycles. In the first write cycle, the first and every subsequent third byte are inverted. In the second write cycle, the second and every subsequent third byte are inverted. In the third write cycle the third and every subsequent third byte are inverted. For example:

```
*RT. E00 1FFF 55

AA 55 55 AA 55 55 AA 55 55 etc.   1st write cycle
55 AA 55 55 AA 55 55 AA 55 etc.   2nd write cycle
55 55 AA 55 55 AA 55 55 AA etc.   3rd write cycle
```

The test byte is then inverted and the test is rerun:

```
55 AA AA 55 AA AA 55 AA AA  etc.
AA 55 AA AA 55 AA AA 55 AA  etc.
AA AA 55 AA AA 55 AA AA 55  etc.
```

If no test byte is specified, zero is assumed. Zero is recommended for testing for noise, particularly in dynamic RAM, while test byte 55 is suitable for testing bits.

*RCOPY <adr1>,<adr2>,<length>

Minimum abbreviation *RCOP.

This copies the area of memory (specified by <length>) from the source address to the destination address inclusive.

*RVERIFY <adr1>,<adr2>,<length>

Minimum abbreviation *RV.

This compares the specified area of memory from the source address with that at the destination address. If there is a discrepancy between the two areas of memory, the locations in questions will be displayed on the screen.

*BREAK [<no>[,<adr>

Minimum abbreviation *BR.

This commands sets the specified breakpoint at the desired address. For example:

```
*BREAK 3 3002
```

will set breakpoint 3 at &3002. To clear the breakpoint again, type:

```
*BREAK 3  (i.e. without an address)
```

Breakpoints are placed when the commands *GOTO, *GOSUB and *CALL are entered. They are removed on return from a *GOSUB or on a 'warm' restart. Even if the program 'crashes', the breakpoints will be removed on 'warm' reset to enable testing to continue by reloading the object code file.

*BRKS

Minimum abbreviation *BRK.

This command displays the current saved breakpoints:

```
0   0000
1   0000
2   0000
3   0000
4   0000
```

*REGS

Minimum abbreviation *RE.

This will print out the 6502 saved registers:

A	X	Y	P	ST	PC
00	00	00	00	F4	0000

To specify the registers:

*A <hex>	Accumulator
*X <hex>	Index Register X
*Y <hex>	Index Register Y
*P <hex>	Processor Status Register
*ST<hex>	Stack Pointer
*PC<hex>	Program Counter

*REG <reg> <data>

will put the specified hex data into a specified register, i.e.

    *REG A B5

will put the hex value B5 into the accumulator.

*GOTO [<adr>]

Minimum abbreviation *GOT.

This takes all saved registers, loads them into the microprocessor and starts executing. For example, if there was a machine code routine resident at &3000, *GOTO 3000 would load the processor's registers, go to the specified address and start executing the machine code routine until it reached a breakpoint. At this point the monitor will display the current state of the program registers and return to the monitor prompt. It will be noted that the program counter now contains the address of the breakpoint. The program can be made to continue from here by simply typing *GOTO (without an address). The display registers will be loaded into the microprocessor as before, and the program will continue from the address in the program counter.

The facility to continue after a breakpoint is provided by an intelligent breakpoint handler in the monitor which only places the breakpoints when the command *GOTO is typed. It also removes them when a breakpoint is reached or when the reset button is pressed. In this way the machine code program under test should remain intact. If the GOTO address is the same as a breakpoint address the breakpoint is not loaded, thus making it possible to continue with the program.

*GOSUB [<adr>]

Minimum abbreviation *G.

This works in the same way as *GOTO, except that it does not load the stack pointer (ST). It loads the value required to return to the monitor when the machine code instruction RTS is executed. In this way subroutines can be tested. If a breakpoint is encountered, the monitor behaves the same as with the *GOTO command, and it is possible to continue with the program as described above.

*CALL [<adr>]

Minimum abbreviation *CAL.

This is the same as *GOSUB. However, when a breakpoint is encountered, the monitor displays the registers and automatically continues. In this way all five breakpoints can be displayed in one test. *CALL prints out a breakpoint and continues, whereas *GOTO stops at a breakpoint.

The Use of Breakpoints: Example

```
*MEX1000

1000 . . . . . . . . . . . . _A9,00        ; LDA #0
1002 . . . . . . . . . . . . _A2,01        ; LDX #1
1004 . . . . . . . . . . . . _A0,02        ; LDY #2

1006 . . . . . . . . . . . . _38           ; SEC
1007 . . . . . . . . . . . . _18           ; CLC
1008 . . . . . . . . . . . . _60           ; RTS

<ESCAPE>
```

This simple program loads the A, X and Y registers with 0, 1 and 2 respectively and then sets and clears the carry flag.

```
*A FF        set registers A, X, Y and P to FF
*X FF
*Y FF
*P FF
*PC 1000     PC = 1000

*BREAK 0 1002
*BREAK 1 1004
*BREAK 2 1006
*BREAK 3 1007
*BREAK 4 1008
```

*BRKS	will display set breakpoints
*REGS	will display processor registers
*GOTO	the monitor will display the first breakpoint

A	X	Y	P	ST	PC
00	FF	FF	7F	F4	1002

*GOTO continue

A	X	Y	P	ST	PC
00	01	FF	7D	F4	1004

etc.

Now reset the registers A and X to FF:

*A FF
*X FF

Try *CALL 1000

Note that it is optional whether you specify the address in *GOTO, *GOSUB or *CALL other than by using the program counter.

All the breakpoints will be displayed:

A	X	Y	P	ST	PC
00	FF	FF	7F	F4	1002
00	01	FF	7D	F4	1004
00	01	02	7D	F4	1006
00	01	02	7D	F4	1007
00	01	02	7D	F4	1008

*MOFF/*MON

Minimum abbreviation

Monitor off/Monitor on

This command will rarely be used in practice, but it enables the user to switch off the monitor in cases where the Monitor commands conflict with, say, the DFS commands.

*NMI

If the processor NMI line is connected to ground through a normally open switch, *NMI allows the inspection of registers after a crash. The command *NMI is entered from the keyboard. If a crash subsequently occurs during program development, then making the switch will cause the processor registers to be dumped on to the screen.

*PROM

Minimum abbreviation *PR.

This command calls up the EPROM programmer and should display the following information on the screen:

```
Eprom Programmer 1.0 (C) ATPL
DEVICE?_
```

When you have entered the number of the device you wish to use, e.g. 2764, the screen will display:

```
SELECT 21 VOLTS . . . . fit link K
BLOW : READ : VERIFY : CLEAR
Enter initial letter
```

Further information can be found in the documentation accompanying the EPROM programmer.

*CLOCK/*DATE

These commands send the current time and date to the output channel. If you have set the clock with the BASIC program provided in Appendix 4 or on the utilities disk, try displaying the time and date continuously by typing in the following program:

```
 5 CLS
10 REPEAT
20 PRINTTAB(12,10);:*CLOCK
30 PRINTTAB(12,11);:*DATE
40 IF INKEY 100
50 UNTIL FALSE

>RUN
```

APPENDIX 1: MEMORY DECODING

Standard Maps M4 and I/02

Two fusable link PROMs are used to decode the processor address lines. For the 6502 EuroCUBE these are labelled 'M4' and 'I/02'. Great effort has been made to provide a wide variety of address maps. A table of standard maps is shown overleaf:

Map	M3	M2	M1	M0	I/O Block	I/O Page	Typical Use
0	0000-1FFF 8K RAM	2000-3FFF 8K ROM/RAM	8000-BFFF BBC BASIC ROM	*C000-FFFF MOSB EPROM	D000-DFFF	FE00-FEFF	Development BASIC
1	0000-3FFF 16K RAM	4000-7FFF 16K ROM/RAM	8000-BFFF BBC BASIC ROM	*C000-FFFF MOSB EPROM	D000-DFFF	FE00-FEFF	EPROM BASIC
2	000-7FF 2K RAM	800-FFF 2K RAM/ROM	8000-BFFF BBC BASIC ROM	*C000-FFFF MOSB	D000-DFFF	FE00-FEFF	Minimum BASIC
3	Reserved for future use by Control Universal Ltd						
4	0000-1FFF 8K RAM/ROM	2000-3FFF 8K ROM/RAM	E000-E7FF 2K RAM	E800-FFFF MOSF (6809)	E000-EFFF	EE00-EEFF	FLEX, low cost 6809 only
5	0000-3FFF 8/16K RAM/ROM	4000-7FFF 16K ROM/RAM	E000-E7FF 2K RAM	E800-FFFF MOSF (6809)	E000-EFFF	EE00-EEFF	FLEX 6809 only
6							
	Reserved for future use by Control Universal Ltd						
7							
8	000-7FF 2K RAM	A000-BFFF 8K ROM/RAM	C000-DFFF 8K ROM	E000-FFFF MOSA	000-FFF	FE00-FEFF	ATOM BASIC

9	0000–3FFF 2/8/16K RAM	D000–DFFF 4K RAM	E000–EFFF 4K ROM	F000–FFFF 4K MOS	7000–7FFF	FE00–FEFF	Low-cost EPROM
10	0000–3FFF 2/8/16K RAM	4000–7FFF 16K ROM	8000–BFFF 16K ROM	C000–FFFF 16K MOS	0000–0000	FE00–FEFF	ROM intensive
11	0000–7FF 2K RAM	800–FFF 2K RAM	1000–17FF 2K RAM	8000–FFFF 32K MOS	0000–0000	FE00–FEFF	Low-cost RAM
12	0000–1FFF 8K RAM	2000–3FFF 8K RAM	4000–5FFF 8K RAM	8000–FFFF 32K MOS	7000–7FFF	FE00–FEFF	RAM intensive
13	000–7FF 2K RAM	800–FFF 2K RAM	E000–EFFF 4K ROM	F000–FFFF 4K MOS	D000–DFFF	FE00–FEFF	Minimum configuration
14	0000–1FFF 8K RAM	2000–3FFF 8K RAM	8000–BFFF 16K ROM	C000–FFFF 16K MOS	7000–7FFF	FE00–FEFF	General purpose
15	Spare for user purposes (please contact Control Universal Ltd for details of current charges)						

* C000–CFFF, E000–FFFF, i.e. 16K EPROM will add 4K (C000–CFFF). Block D is reserved for I/O cards.
8K EPROM data at E000–EFFF also appears at C000–CFFF.

NOTES

1. M0 to M3 inclusive refers to the four RAM/ROM/EPROM slots on the board, whereas M4 refers to the fused-link PROM version number.

2. Links 6 to 9 (L6-L9) determine which of the 16 possible maps is selected. L6 is the LSB, L9 is the MSB. For example, to select Map 12 (RAM intensive), links L9 and L8 are made, L7 and L6 are left open.

3. On EuroBEEB, all links are left open giving Map 0.

4. Maps 4 and 5 apply only to 6809 EuroCUBEs.

I/O Map

0,1,2,8-14	VIA	FE00-FE0F
	UART	FE10-FE17
	CLOCK	FE18-FE1F
	LPAGE	FE00-FE7F
	HPAGE	FE80-FEFF
4-5	As above but first digit is E, e.g. VIA EE00-EE0F.	

Note 1: M0 must be deselected for the on-board I/O, i.e. FE00-FEFF.

Note 2: These VIA addresses are NOT the same as those in the BBC Micro since the CRTC, ULA, etc. are not present (see p. 437 of the "BBC User Guide" for comparison).

6502 Input/Output Memory Map:

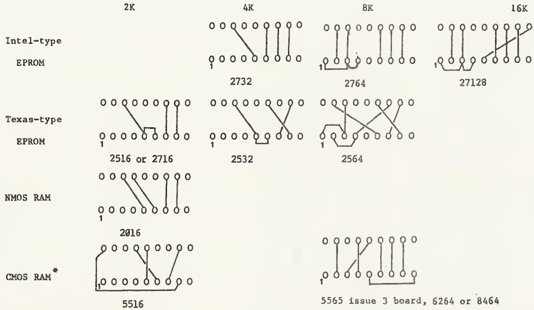
1. CUBAN-8 VIA	-	&DB00	
CUBAN-8 ADC	-	&DB10	
2. CUBAN-12	-	&DC00	
3. RACKPRINT	-	&DE00	
4. VIEWLINE	-	&DF00	
or			
CU-GRAPH	-	&DF00	
5. TELETEXT	-	RAM	&D400-&D7FF
		CRTC	&D800-&D8FF
		Printer	&D900-&D9FF

APPENDIX 2: MEMORY CONFIGURATION

The four memory sockets are designated 'byte wide' and have common address and data buses. The other lines are brought out to a row of nine pins. These can be wire wrapped to nine other signals, thus specifying a certain memory type. Below is a list of the currently most popular memories.

pin signal	$\overline{\text{PD}}$	A12	+5V	R/W	VCB	A11	GND	CS	A13
EPROM pin no.	1	2	28	27	26	23	22	20	PD

PD is a deselect signal on memory socket 0 only.
PD (power down) is an active high CMOS RAM signal on memory sockets 1,2 and 3 (issue 3 boards upwards).



* NOTE: When using CMOS RAMs, e.g. 5516 or 5565, the pin out described above uses a power down signal derived from the address decoding PROM. This is ONLY active in the specified RAM areas of the memory maps.

Issue 5 EuroBEEB

Issue 5 EuroBEEBs do not have wire-wrap pins, but have the standard interconnections (5565 5565 27128 2764/deselect) tracked on to the PCB. Any changes will require tracks to be cut and wire links to be added as appropriate. Fig. 7a shows the modifications that must be made if an EPROM instead of a CMOS RAM is to be used in socket M2. Fig. 7b shows the modification that must be carried out in order to accommodate a 27128 device in socket M0 instead of the standard 2764 device.

Fig. 7a: Modifications to fit 2764 in M2:

1. Link A to B to C
2. Drill out hole D to 1 mm (no larger)
3. Cut track E

top of board

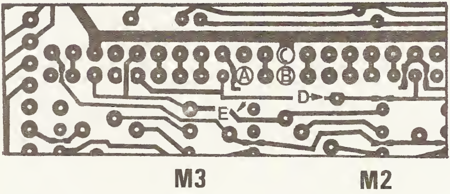
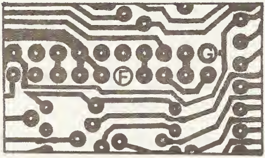


Fig. 7b: Modifications to fit 27128 in MO

1. Link F to G



top of
board

MØ

APPENDIX 3: SOFTWARE 'COLD RESET' EXAMPLE

Example from MOSB.3/EuroBEEB:

```
      VIAIER  EQU  FE0D
      LDA  VIAIER        ; Interrupt enable register
                         ;= &80 after Power On
      ASL  A
      BNE     SOFT

HARD  ; Hard reset
      ; Power just come on.
      ;
      .

SOFT  ; enable some interrupts
      LDA     #&C0
      STA     IER        ; enable timer 1 interrupts
```

APPENDIX 4: SERIAL SOFTWARE FOR EuroCUBE-65

MOSB.2 and MOSB.3 use a 6551 with interrupt control. RAM buffers are used to store the transmitted and the received data. The transmitter interrupt routine empties the transmitter buffer to send data, and the receiver interrupt fills the receiver buffer on reception of data. This has the advantage that the program can be performing a task while the receiver fills the buffer. The program can then return to the data stored in the buffer at a later time.

A simple serial channel can be produced using the three program modules, described below, which allow for the following:

(a) Initialisation of the 6551 UART and the 6522 VIA registers

(b) Reception of serial data

(c) Transmission of serial data

The registers and symbols used in these modules are defined below.

Registers and Symbols

Device	Offset	Symbol	Name/function
6551	0	UTxD	Transmit Data Register - Write Only
6551	0	URxD	Receive Data Register - Read Only
6551	1	USTAT	Status Register
6551	2	UCOMD	Command Register
6551	3	UCTRL	Control Register
6522	-	VPCR	VIA Peripheral and Control Register (PCR)

Program Modules

1. Initialise

Sets up the hardware for RS-423 serial input/output.

```
.INIT  LDA #&0E     Set VIA Peripheral
                    Control Register:
       STA VPCR     CA2 initialised to HIGH

       LDA #&1E     Set 6551 Control Register
       STA UCTRL    : internal clock : 9600 baud
                    : 8 data bits, 1 stop bit

       LDA #&0B     Receiver enabled,
       STA UCOMD    interrupts disabled

       RTS
```

NOTES

(a) This module configures the 6522 PCR as follows:

Bits	Data	Meaning
1–3	110	Set CA2 LOW - i.e. NOT BUSY
0	X	These bits are concerned with CA1,
4–7	XXXX	CB1 and CB2 which are not used in Issue 5 (or later) boards

X = DON'T CARE

(b) The 6551 Control Register (with data &1E) is configured as follows:

Bits	Data	Meaning
0–3	1110	On-chip baud rate generator 9600 baud selected
4	1	Internal baud rate generator selected
5–6	00	Word length : 8 bits selected
7	0	One stop-bit selected

(This format is the same as the default format on the BBC Microcomputer.)

(c) The 6551 Command Register (with data &0B) is configured as follows:

Bits	Data	Meaning
0	1	DATA Terminal Ready : 1 = Enable Receiver/Transmitter ($\overline{DTR}$ = LOW)
1	1	Receiver interrupt : disabled
2, 3	10	Transmitter controls : transmit interrupt disabled : $\overline{RTS}$ asserted low
4	0	Echo mode : disabled
5, 6,	7XX0	Parity check control : parity disabled

As can be seen from Fig. 3 (p.15), the 'BUSY' line from the receiver is asserted LOW. CA2 from the 6522 is inverted to produce this 'BUSY' signal.

6522 CA2	RS-423 'BUSY' LINE	STATE
0	1	NOT BUSY
1	0 (if enabled by RTS)	BUSY

The CA2 line must be set or cleared by software. CA2 can be altered from HIGH to LOW by changing bit 1 of the PCR (Peripheral Control Register) from HIGH to LOW. (Bits 2 and 3 MUST remain high.)

2. Receive

```
REC     LDA USTAT      Receive Data Register full ?
        AND #&08       if so get data, else
        BNE DATAIN     look again

        LDA VPCR       Save copy of VIA PCR
        PHA            (copy has CA2 high)
        AND #&FD       free busy line,
                       i.e. force CA2 low
        STA VPCR       but leave other bits unchanged

SERW    LDA VSTAT      receiver full
        AND #608       if not test again
        BEQ SERW

        PLA            restore old to PCR
        STA VPCR       assert busy

DATAIN  LDA URxD       Read Data
        RTS
```

NOTE: Although CA2 is initialised to 'HIGH', the BUSY line is normally inactive (in tri-state high impedance) when the $\overline{\text{RTS}}$ output is HIGH. RTS is, of course, the BUSY line enable - as shown in the schematic.

3. Transmit Module

This module assumes the data to be transmitted is already in the accumulator.

```
.TRANS PHA           Save Data
       LDA USTAT     Get Status
       AND #&50      Test bits 4 and 6
       EOR #&10      (See explanatory note below)
       BNE TRANS
       PLA           Get Data
       STA UTxD      Send Data
       RTS
```

NOTE: This module tests bits 4 and 6 of the Status Register.

Bits	Data	Use
4	1 0	Transmitter Data Register 1 = empty; 0 = NOT empty
6	1 0	$\overline{DSR}$ HIGH = NOT ready $\overline{DSR}$ LOW = ready

Transmission occurs when:

(a) $\overline{DSR}$ = LOW, i.e. the other (receiver) terminal is ready.

(b) Transmitter Data Register is NOT empty.

In MOSB.1 the receive/transmit software is programmed as above.

In MOSB.2 and MOSB.3 the receive/transmit software is a fully buffered, interrupt-driven module.

USE OF THE Am26LS29, 30, 31 and 32 QUAD DRIVER/RECEIVER FAMILY IN EIA RS-422 AND 423 APPLICATIONS

By David A. Laws and Roy J. Levy

INTRODUCTION

Today's high-performance data processing systems demand significantly faster data communications rates than are possible with the EIA RS-232 specifications in use for the past ten years.

Two new standards prepared by the Electronic Industries Association address this need. EIA RS-423 is an unbalanced, polar voltage specification designed to interface with RS-232C, while greatly enhancing its operation. It permits the communication of digital information over distances of up to 2000 feet and et data rates of up to 300 Kilobaud. EIA RS-422 is a balanced voltage digital interface for communicaton of digital data over distances of 4000 feet or data rates of up to 10 megabaud.

Advanced Micro Devices has developed a family of monolithic Low-power Schottky quad line drivers and receivers to meet the requirements of these specifications.

The Am26LS29 and 30 line drivers and the Am26LS32 receiver meet all requirements of RS-423 while the Am26LS31 differential line driver and the Am26LS32 receiver meet the requirements of RS-422.

A second receiver element, the Am26LS33 is available for use in high common mode noise environments, exceeding the common mode voltage requirements of RS-422 and RS-423.

This application note reviews the use of these devices in implementing the new standards. Emphasis is given to the EIA RS-422 balanced interface.

EIA STANDARD SPECIFICATIONS

Two basic forms of operation are available for transmission of digital data over interconnecting lines. These are the single ended and differential techniques.

The single-ended form uses a single conductor to carry the signal with the voltage referenced to a single return conductor. This may also be the common return for other signal conductors. Figure 1a.

The single-ended form is the simplest way to send data as it requires only one signal lina per circuit. This simplicity, however, is often offset by the inability of this form to allow discrimination between a valid signal produced by the driver, and the sum of the driver signal plus externally induced noise signals.

A solution to soma of the problems inherent in the single-ended form of operation is offered by the differential form of operation. Figure 1b. This consists of a differential driver (essentially two single-ended drivers with ona driver always producing the complamentary output signal leval to the other driver), a twisted pair transmission lina and a differential line receiver. The driver signal appears as a differential voltage to the line receiver, while the noise signals appear as a common mode signal. The two signals, therefora, can be discriminated by a line receiver with a sufficient common mode voltage operating ranga.

The Electronic Industries Association, EIA, has defined a number of specifications standardizing the interface between data terminal equipment and data circuit terminating equipment based on both single-ended and differential operation.

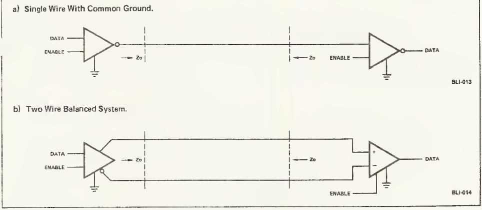


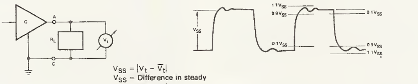
Figure 1. Data Communication Techniques.

The most widely used standard for interfacing between date terminal equipment and data communications equipment today, is EIA RS-232C, issued in August 1969. The RS-232C electrical interface is a single-ended, bipolar-voltage, unterminated circuit. This specification is for serial binary date interchange over short distances (up to 50 feet) at low rates (up to 20 Kilobaud). It is a protocol standard as well as an electrical standard, specifying hand shaking signals and functions between terminal and the communications equipment. As already noted, single-ended circuits are susceptible to all forms of electromagnetic interference. Noise and cross talk susceptibility are proportional to length and bandwidth. RS-232C places restrictions on both. It limits slew rate of the drivers (30V/μs) to control radiated emission on neighboring circuits and allows bandwidth limiting on the receivers to reduce susceptibility to cross talk. The length and slew rate limits can adequately control reflections on unterminated lines, and the length and bandwidth limits are more than adequate to reduce susceptibility to noise.

Like EIA RS-232C, the new EIA RS-423 is also a single-ended, bipoler-voltage unterminated circuit. It extends the distance and date rate capabilities of this technique to distances of up to 4000 feet at data rates of 3000 baud, or at higher rates of up to 300 Kilobaud over a maximum distance of 40 feet.

EIA RS-422 is a differential, balanced voltage interface capable of significantly higher data rates over longer distances. It cen accommodate rates of 100 Kilobaud over a distance of 4000 feet or rates of up to 10 megabaud. These performance improvements stem from the advantages of a balanced configuration which is isolated from ground noise currents. It is also immune to fluctuating voltage potentials between system ground references and to common mode electromagnetic interference. Figure 2 compares the driver output waveforms for the three EIA standard configurations, while Table I compares the key characteristics required by drivers and receivers intended for these applications. Since RS-232C has been in use for many years, RS-422 and 423 parameter values have been selected to facilitate an orderly transition from existing designs to new equipment.

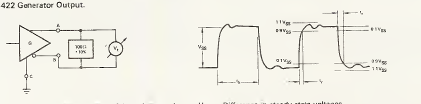
a) EIA RS-232C Generator Output.



$V_{SS} = |V_t - \overline{V_t}|$
V_{SS} = Difference in steady state voltages
R_L = 3KΩ to 7KΩ
V_{SS} min. = ±5V, V_{SS} max. = ±25V

BLI 015

b) EIA RS-422 Generator Output.

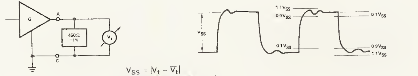


t_D = Time duration of the unit interval at the applicable modulation rate
$t_r \leq 0.1t_D$ when $t_D \geq 200$ns
$t_r \leq 20$ns when $t_D < 200$ns

V_{SS} = Difference in steady state voltages
$V_{SS} = |V_t - \overline{V_t}|$
V_{SS} min. = 2V; V_{SS} max. = 6V

BLI-016

c) EIA RS-423 Generator Output.



$V_{SS} = |V_t - \overline{V_t}|$
V_{SS} = Difference in steady state voltages
V_{SS} min = ±3.6V; V_{SS} max. = ±6V

BLI-017

Figure 2. Driver Output Waveforms.

TABLE I
KEY PARAMETERS OF EIA SPECIFICATIONS

Characteristics	EIA RS-232C	EIA RS-423	EIA RS-422	Units
Form of Operation	Single Ended	Single Ended	Differential	
Max. cable length	50	2000	4000	Feet
Max. data rate	20K	300K	10M	Baud
Driver output voltage, open circuit*	±25	±6	6 volts between outputs	Volts (Max.)
Driver output voltage, Loaded output*	±5 to ±15	±3.6	2 volts between outputs	Volts (Min.)
Driver output resistance power off	Ro = 300Ω	100μA between −6 to +6V	100μA between +6 and −.25V	Min.
Driver output short circuit current I_{SC}	±500	±150	±150	mA (Max.)
Driver output slew rate	30 V/μsec Max.	Slew rate must be controlled based upon cable length and modulation rate	No control necessary	
Receiver input resistance R_{in}	3K to 7K	≥4K	≥4K	Ω
Receiver input thresholds	−3 to +3	−0.2 to +0.2	−0.2 to +0.2	Volts (Max.)
Receiver input voltage	−25 to +25	−12 to +12	−12 to +12	Volts (Max.)

*± indicates polarity switched output

INTEGRATED CIRCUIT CHARACTERISTICS

Most semiconductor manufacturers offer integrated circuits designed to satisfy the old RS-232C standard. A number of them have designs in progress to meet the new EIA specifications. Products available from Advanced Micro Devices to meet these needs are shown in Table II.

The Am26LS29, 30, 31 and 32 are a family of quad drivers and receivers designed specifically to meet the new EIA standards. These products utilize Low-Power Schottky technology to incorporate four drivers or four receivers, together with control logic, in the standard 16-pin package outlines.

The Am26LS29/30 and the Am26LS32 are driver and receiver 's designed to implement the single-ended EIA RS-423 standard. The Am26LS31 is a differential line driver designed for use with the Am26LS32 receiver in a differential mode to meet EIA RS-422.

Am26LS29 AND Am26LS30 QUAD RS-423 LINE DRIVERS

The Am26LS29 and 30 consist of four single-ended line drivers designed to meet or exceed the requirements of RS-423. The buffered driver outputs are provided with sufficient source and sink current capability to drive 50 ohm to a virtual ground transmission line and high capacitive loads. The Am26LS29 has a three-state output control while the Am26LS30 has a Mode Control input that allows it to operate as a dual RS-422 driver (with suitable power supply changes), Figure 3.

Each of the four driver inputs, as well as the $\overline{\text{Enable}}$/Mode Control input is a PNP Low-Power Schottky input for reduced input loading, one-half the normal fan-in. Since there are two inverters from each input to output, the driver is non-inverting. When operating in the RS-423 mode, the Am26LS29 and 30 require both +5V and −5V nominal value power supplies. This allows the outputs to swing symmetrically about ground – producing a true bipolar output. The Mode Control (Pin 4) of the Am26LS30 should be HI or tied to

TABLE II
ADVANCED MICRO DEVICES' EIA COMPATIBLE DEVICES

EIA Standard	Drivers	Receivers
RS-232C	Am1488 Quad Driver	Am1489, 1489A Quad Receivers with response control pin
	Am9616 Triple Driver with logic control	Am9617 Triple Receiver with optional hysteresis
	Am2616 Quad Driver also specified for CCITT V.24 and MIL-188C	Am2617 Quad Receiver specified over MIL range
RS-422	Am26LS31 Quad Differential with three-state control gating	Am26LS32 Quad Differential Driver single-ended Receiver
RS-423	Am26LS29 Quad Driver with three-state output	Am26LS32 Quad single-ended/ Differential Receiver
	Am26LS30 Quad Driver with slew rate control	

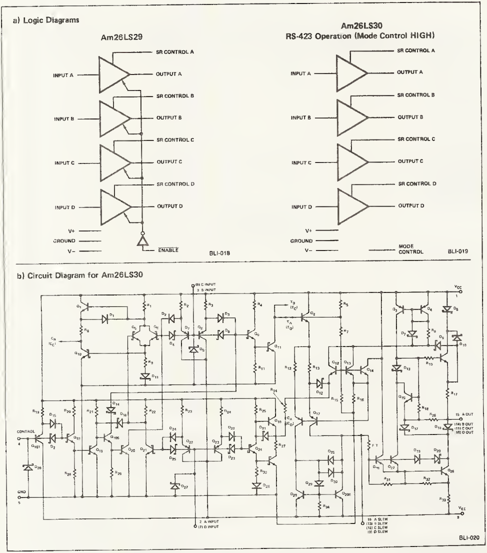


Figure 3. Am26LS29 and Am26LS30 Drivers.

V_{CC} Each output is designed to drive the RS-423 load of 50 ohms with an output voltage equal or greater than +3.6 volts in the HI state and −3.6 volts in the LO state. Each output is current limited to 150mA max. in either logic state. A Slew Rate control pin is brought out separately for each output to allow output ramp rate (rise and fall time) control. This provides suppression of near end cross talk to other receivers in the cable. Connecting a capacitor from this node to that driver's respective output will produce a ramp (10% to 90%) of 50ns typical for each picofarad of capacitance in that capacitor. RS-423 establishes recommended ramp rates versus length of line driven and modulation rate, Figure 4.

The Am26LS30 can be used at low data rates as a dual EIA RS-422 driver with three-state outputs by connecting the V_{EE} supply and the mode control input to ground.

Am26LS31 QUAD RS-422 DRIVER

The Am26LS31 is a quad differantial line drivar designad to meet the RS-422 specification while operating with a single +5 volt supply. A common enable and disable function controls all four drivers, Figure 5. The driver featuras high speed, de-skewad differantial outputs with typical propagation deleys of 12ns and residual skew of 2ns. Both differential lina outputs are designed for three-state oparation to ellow two-way half duplex and multiplex, data bus applications.

Table III is a summary of the essential requirements of the RS-422 standard. Section A describes the key characteristics satisfied by the Am26LS31 driver.

The balanced differential line driver consists of two halves, each of which is similar to a Low-power Schottky TTL gate with equal source and sink current capability. The two helves are emitter coupled in a differential input configuration. One side of the input circuit is tied to a fixed TTL bias thrashold and the other side is tiad to e sink diode in normal DTL/TTL fashion. This configuration offers complemantary outputs with very low skew, dependant only upon component matching, a necessity to meet RS-422.

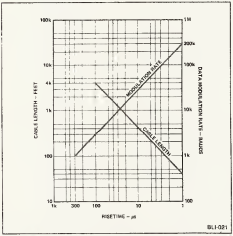


Figure 4. Data Modulation Rate or Cable Length Versus Risetime for EIA RS-423.

The circuit diagram of the driver is shown in Figure 6. The emitter-coupled input circuit is formed by Q2 and Q3, which are biased by a current source. This source is a current mirror, formed by Q1 which supplies the current, and D6 which is diode connected transistor matched to Q1. The fixed bies for Q3, formed by D5 and D6, is $2V_{BE}$. A $2V_{BE}$ bias, less the D2 Schottky diode drop, provides the normal Low-power Schottky TTL threshold, $V_{IL} = 0.7V$. R19 provides a boost to 0.8V for a full 400mV TTL noise margin. The differential outputs of the emitter coupled stage, A and $\overline{A}$, drive emitter followers Q14 and Q15, which provide the required speed and matching characteristics. The emitter followers, drive phase splitters Q4 and Q5, which in turn drive totem-pole outputs. The outputs at tha line interface are of standard Low-power Schottky TTL configuration, except that circuit values are modified to provide high sourcing capability. The outputs are designed to source or sink 20mA each, so that thay can generata a voltage of at leest 2.0V across a 100 ohm loed, as required by RS-422. Additional circuitry has been included to make the line outputs three-state for two-way bus applications. The Am26LS31 meets the RS-422 requirement that the driver not load the line in the powered down condition ($I_X \leq 100\mu A$) or if the power supply to that device should feil.

Am26LS32 QUAD RS-422 AND 423 RECEIVER

The Am26LS32 is a quad line receiver which, operating from a single 5 volt supply, can be used in either differential or single-ended modas to satisfy RS-422 and 423 applications respectively. A complementary eneble and disable feature, similar to that on the driver, controls all four receivers, Figure 7. The device's three-state outputs, which can sink 8mA, incorporete a fail-safa input-output relationship which keeps tha outputs high when the inputs are open.

The Am26LS32 meets tha receiver input specification of Table III, a 200mV threshold sensitivity with common mode rejection exceeding the supply line potentiels, (greater than 7 volts). The same design faature of the input circuit which provides the common mode rejection also insures excellent powar supply ripple rajection, which is important whan switching the high currents involved in a system's interfaces. Furthermore, unlike operational amplifiers, where the DC common moda and power supply rejection ratios roll off with opan loop gain, tha full rejection capability of this line raceiver is maintainad at high frequencies. The receiver hysteresis of typically 30mV, provides differential noise immunity. Signals received on long lines can have slow transition timas, and without hysteresis, a small amount of noise around the switching threshold can cause arrors in the receiver output.

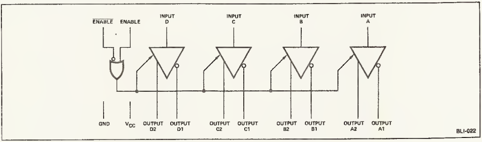


Figure 5. Am26LS31 Logic Diagram.

TABLE III
SUMMARY OF EIA RS-422 STANDARD FOR A BALANCED DIFFERENTIAL INTERFACE

A. Line Driver

Open Circuit Voltage (either logic state)
- Differential — $|V_{do}| \leq 6.0V$
- Common Mode — $|V_{cmo}| \leq 3.0V$

Differential Output Voltage (across 100 ohm load)
- Either logic state — $|V_d| \geq \max(0.5V_{do}, 2.0V)$

Output Impedance
- Either logic state — $R_G \leq 100$ ohms

Mark-Space Level Symmetry (across 100 ohm load)
- Differential — $|V_{dS}| - |V_{dM}| \leq 0.4V$
- Common Mode — $|V_{cmS}| - |V_{cmM}| \leq 0.4V$

Output Short Circuit Current (to ground)
- Either Output — $|I_{SC}| \leq 150mA$

Output Leakage Current (power off)
- Voltage Range — $-0.25V \leq V_x \leq +6.0V$
- Either Output at V_x — $|I_x| \leq 100\mu A$

Rise and Fall Times (across 100 ohm load)
- T = Baud Interval — $(t_r, t_f) \leq \max(0.1T, 20ns)$

Ringing (across 100 ohm load)
- Definitions
  - $V_{dss} = V_d$ (steady state)
  - $V_{SS} = V_{dS} - V_{dM}$ (steady state)
- Limits (either logic state)
  - Percentage — $|V_d - V_{dss}| \leq 0.1V_{SS}$
  - Absolute — $2.0V \leq |V_d| \leq 6.0V$

B. Line Receiver

Signal Voltage Range
- Differential — $|V_d| \leq 6.0V$
- Common Mode — $|V_{cM}| \leq 7.0V$

Single-Ended Input Current (power ON or OFF)
- Either Input at V_x — $|V_x| = 10V$
- Other Input Grounded — $|I_V| \leq 3.25mA$

Single-Ended Input Bias Voltage (other input grounded)
- Either Input Open Circuit — $|V_B| \leq 3.0V$

Single-Ended Input Impedance (other input grounded)
- Either Input — $R_L \geq 4000$ ohms

Differential Threshold Sensitivity
- Common Mode Voltage Range — $|V_{cm}| \leq 7.0V$
- Either Logic State — $|V_T| \leq 200mV$

Absolute Maximum Input Voltage
- Differential — $|V_d| \leq 12V$
- Single-Ended — $|V_x| \leq 10V$

Input Balance (threshold shift)
- Common Mode Voltage Range — $|V_{cm}| \leq 7.0V$
- Differential Threshold (500 ohms in series with each input)
  - Either Logic State — $|V_t| \leq 400mV$

Termination (optional)
- Total Load Resistance (differential) — $R_T > 90$ ohms

Multiple Receivers (bus applications)
- Up to 10 receivers allowed. Differential threshold sensitivity of 200mV must be maintained.

Hysteresis (optional)
- As required for applications with slow rise/fall time at receiver, to control oscillations.

Fail Safe (optional)
- As required by application to provide a steady MARK or SPACE condition under open connector or driver power OFF condition.

C. Interconnecting Cable

Type
- Twisted Pair Wire or Flat Cable Conductor Pair

Conductor Size
- Copper Wire (solid or stranded) — 24 AWG or larger
- Other (per conductor) — $R \leq 30$ ohms/1000 ft.

Capacitance
- Mutual Pair — $C \leq 20pF/ft.$
- Stray — $C \leq 40pF/ft.$

Pair-to-Pair Cross Talk (balanced)
- Attenuation at 150KHz — $A \geq 40dB$

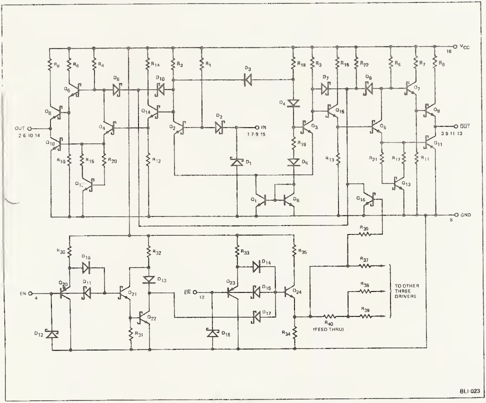


Figure 6. Am26LS31 Circuit Diagram (Only one driver shown).

The balanced differential line receiver is a three-stage circuit. The input stage consists of a low-impedance differential current amplifier with series resistor inputs to convert line signal voltage to current and provide a moderate input impedance. The input resistors provide an impedance greater than 6K on input, power on or power off, which exceeds the requirements of RS-422 and RS-423. This is one advantage of the current amplifier input circuit. Another advantage is that is can operate with immunity to common mode voltages above V_{CC} and below ground. The differential threshold sensitivity of this circuit is 200mV, as required by RS-422. The second stage is a differential voltage amplifier, which interfaces to the single-ended output stage through an emitter follower. The output stage is a standard Low-power Schottky TTL totem-pole output with three-state capability.

The full circuit is shown in Figure 8. Resistors R_{20} and R_{21}, which connect the non-inverting input to V_{CC} and the inverting input to ground, provide the fail-safe feature, which guarantees a HIGH logic state for the receiver output when there is no signal on the line. The differential voltage amplifier in the second stage is formed by Q6 and Q3 which are biased by current source Q9. The hysteresis in the receiver switching characteristic is provided by Q4 and Q5, a differential pair biased by current source Q6, whose collectors are connected in positive feedback to the input pull-up circuits. A small amount of current is switched by Q4 and Q5, which must be overcome by the different voltage signal, resulting in the hysteresis. The output stage is driven from one side of the differential second stage by emitter follower Q17, which is a multiple emitter transistor. the second emitter is the control point for the three-state output.Q17 drives the phase splitter Q12, which in turn drives the three-state totempole output. The remainder of the circuit is the output enable control logic. This three-state capability on the receiver TTL side of the interface is a useful feature for modularizing two-way bus design.

A mask option of the input resistors (R_1, R_2, R_{20} and R_{21}) modifies the receiver characteristics to improve operation in high common mode noise environments. This device, known as the Am26LS33, has these resistors at twice the value of the Am26LS32. An input differential or common mode voltage range of ± 15 volts is achieved at the expense of a minor decrease of input threshold sensitivity, to ± 500mV from ± 200mV.

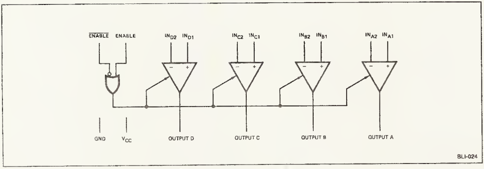


Figure 7. Am26LS32 Logic Diagram.

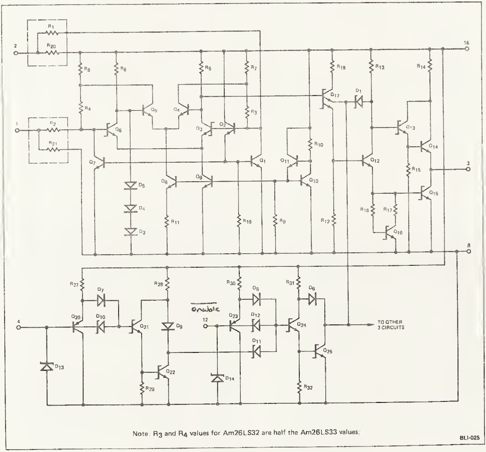


Figure 8. Am26LS32 and Am26LS33 Circuit Diagram (Only one receiver shown).

APPLICATIONS IN MIXED RS-232 AND 422/3 SYSTEMS

A system implemented with the RS-422 differential output cannot be used to drive an RS-232C system directly. An RS-423 single-ended. driver, such as the Am26LS29 or Am26LS30, may be used provided certain precautions are observed.

1. Although the RS-423 driver output specification of between 4 to 5V does not meet the RS-232C specification of 6V, operation is usually satisfactory with RS-232C receivers. This is achieved because the short cable lengths permitted by RS-232C cause very little signal degredation and because of the low source impedance of the RS-423 driver.
2. RS-232C specifies that the rise time for the signal to pass through the ±3.0V transition region shall not exceed 4% of the signal element duration. RS-423 requires much slower rise times, specified from 10% to 90% of the total signal amplitude, to reduce cross talk for operation over longer distances. Therefore, the RS-423 driver in the equipment must be waveshaped. This is achieved by selection of a capacitor value for the Am26LS30 to simultaneously meet the requirements of both RS-423 and RS-232C for data rates covered by RS-232C.
3. RS-423 specifies one common return ground for each direction of transmission, RS-232C requires only one for both directions of transmission. Care must be taken to insure that a return ground path has been created when interfacing between the two systems.
4. RS-232C does not require termination, while it may be necessary for RS-422 and 423. Detailed consideration of termination is covered in the next section.

Note that RS-422 and RS-423 specifies that receivers should not be damaged by voltages up to 12V, while RS-232C allows drivers to produce output voltages up to 25V. The Am26LS32 receiver has been designed to avoid this hazard and can withstand input voltages of ±25 volts.

RS-422 TRANSMISSION LINE FEATURES

Any time a receiver and transmitter are connected with more than a few inches of a wire, problems due to reflections can arise if care is not exercised to terminate the line correctly. RS-422 describes the cable as a twisted pair of approximately 120Ω impedance terminated in a resistor R_T. R_T is not specified because there are two extreme values which may be chosen for the two following general classes of usage: (1) single direction transmission; and (2) multi-direction and multiple source transmission (party line). Considering the cable impedance only, the termination should equal the cable impedance of 120Ω. However this reduces the terminated cable resistance as seen by the driver to only 60Ω, with resulting loading of the output signal. This loading causes a reduction of S/N ratio at the received terminal due to the decrease in signal voltage swing. The solution lies in a compromise between an R_T of 120Ω which providas maximum power transfer at a reduced S/N ratio or R_T of 240Ω which causes a mis-match of 2-to-1 but no S/N reduction. The choice is left to the user as it is system dependent. Both schemes will work for an average line length and should only approach the margins at maximum line length and maximum bit rates.

Electronic Industries Association, when preparing EIA Standard RS-422 conducted their tests with 24 gauge twisted pair wire. The resulting length vs. data rate, is published as a guideline in RS-422 (Figure 9). This shows two important results: (1) Unmodulated baseband (NRZ) signalling is not recommended at distances greater than 4000 feet; (2) At data

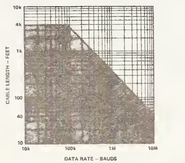


Figure 9. Data Rate Versus Cable Length for Balanced, Twisted Pair Cable (From EIA RS-422).

rates above about 100KHz, the maximum cable length for acceptable signal quality is inversely proportional to data rate.

Result (1) above is due to the DC resistance of the cable. For a 4000 foot cable with a DC resistance of 30 ohms/1000 feet, the DC series loop resistance is 240Ω. The minimum allowable terminated differential load impedance is 90Ω. The DC voltage attentuation is 90/(90 – 240) = 1/4(6db), which is arbitrarily chosen as the maximum allowable limit.

Result (2) is due to line losses. Laboratory tests using the 26LS31 Line Driver connected to the 26LS32 Line Recaiver by 800 feet of ordinary 20 AWG twisted pair (Beldon #8205 plastic-jacketed wire), terminated in its characteristic impedance of 100Ω were evaluated. The input waveform was a 500KHz square wave with (10% to 90%) rise and fall times of less than 10ns. The output waveform produced rise and fall times which together accounted for approximately one-half the period ($t_r + t_f$ = 500ns). This was due to line loss and constant capacity. The energy per cycle of the output waveform is approximately 25% lower than that of the input. The input rise and fall times are not a function of line length, assuming matching termination. The output rise and fall times are dependent upon length in a complex manner. Furthermore, it can be shown by observation that they build up along the line.

Many good reference sources are available on the subject of transmission lines (References 1, 2, 3 and 4). Thesa will provide background information to the following discussion.

Seshadri in Reference (1) has analyzed a line with series resistance losses and has shown that rise time varies with the square of the length. This shows series resistance to be a function of the square root of frequency. Howevar when one tries to use this result in combination with the previous result, it becomes apparent just how difficult the problem is. In Reference (2), the authors point out that skin depth implies a frequency dependent series inductance as well as resistance, and that one cannot be considered without the other.

They go on to show how this leads to the same result; namely that rise and fall times vary with the square of distance.

No attempt will be made to explain here why Figure 5 shows maximum length varying inversely with frequency rather than with the square of frequency. Certainly many complex factors are involved. Our laboratory observations showed a dependence somewhere in between linear and square law.

The Am26LS31 Quad Line Driver and the Am26LS32 Quad Line Receiver are capable of good, clean operation to the distance limits and data rate limits of RS-422.

SYSTEM APPLICATIONS

The Am26LS30, 31, 32 and 33 can be combined in various signaling networks. Using Am26LS29, Am26LS30 and Am26LS32, Figure 10, a unidirectional RS-423 communication can be constructed. Allowing for the voltage variation described earlier, RS-232C requirements can be satisfied. It should be noted that the Am26LS29 or Am26LS30 is used above to meet the bipolar requirements. If a single-ended line, Figure 11, is required without a bipolar requirement, the Am26LS31 can be used by biasing the reference terminal of the receiver to approximately 1.5 volts. Note that additional resistors will enhance fail safe operation.

Figure 12 shows the use of the Am26LS31 and Am26LS32 to meet a balanced line, single direction RS-422 application. If bidirectionality is required, an additional termination should be added as shown in Figure 13.

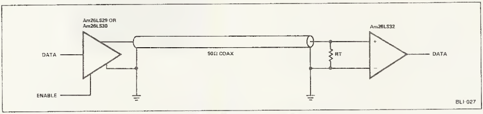


Figure 10. Unidirectional RS-423 (partial RS-232C).

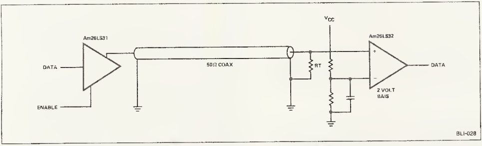
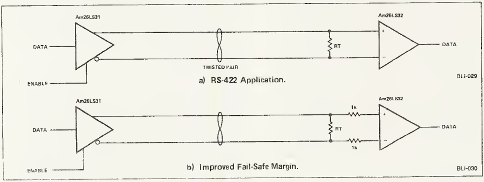


Figure 11. Single-Ended Line Without Bipolar Requirement.



a) RS-422 Application.

b) Improved Fail-Safe Margin.

Figure 12.

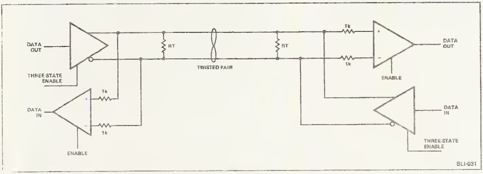


Figure 13. Bidirectional RS-422.

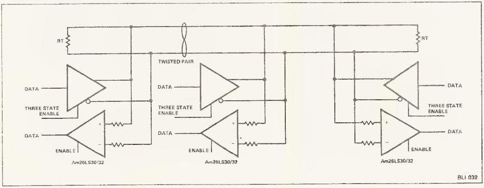


Figure 14. Party Line Configuration.

a) Full Duplex Four-Wire Data Communication RS-422 Interface (with Data Modem).

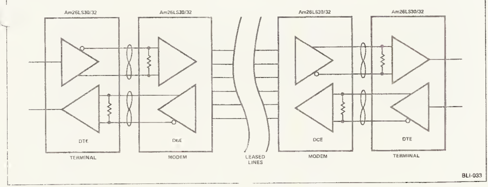


Figure 15.

b) Full Duplex Four-Wire Data Communication RS-422 Interface (without Data Modem).

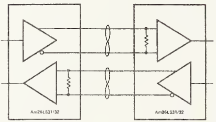


Figure 15. (Cont.)

The high speed capability of RS-422 has attracted the interest of many computer designers for use in the party line mode (Figure 14). The most common usage is that of a four wire full duplex exchange system (Figure 15). This mode of operation involves two pairs of wires each handling a single direction of traffic. The outgoing direction consists of one driver (Am26LS30 or Am26LS31) and n receivers (Am26LS32 or Am26LS33). The incoming direction consists of one receiver (Am26LS32 or Am26LS33) and n drivers (Am26LS30 or Am26LS31). This seems extremely simple to organize. However, problems arise when system ground is considered. If the network of receiver and driver span a moderate to long physical distance, ground loop noise or differences are developed changing the voltage that appears at the terminals of all receivers and drivers except for the one driver that is active. It remains the system reference as long as it is active. This induced or system developed voltage is referred to as Common Mode voltage (CMV) and as such must be considered as a device parameter. All manufacturers specify CMV capability of their receiver in compliance with RS-422 (approx. 7 volts plus signal) but there is no specification for drivers. If the dimensions of the system are short compared to 1/4 wave length of the maximum date rise and fall times, the CMV can be assumed to be minimal and drivers with single voltage supply and limited negative CMV can be used, i.e., Am26LS31. If the system dimensions are large, the CMV will cause problems in that the driver will clamp to the ground the moment the collective or apparent voltage swings below minus 0.5 volts relative to the driver ground, causing a short in the line and increasing level shift and noise. The clamping is caused in part by conduction of the I/C substrate diode. The problem can be avoided by using a driver with an output common mode range (Am26LS30). The Am26LS30 guarantees an output CMV range of ±10 volts about the driver ground reference. New international standards are under consideration to specify this mode of operation. In conclusion, a good system of 4 wire full duplex for data communication would use as an outgoing pair an Am26LS30 line driver and up to 12 – Am26LS32 line receivers, with a termination at the near and far ends of the cable. The same system would use as an incoming pair an Am26LS32 line receiver and up to 32 – Am26LS30 line drivers with only one enabled at a time and all others in three-state mode with cable termination at both near and far ends of the cable.

Many other applications are possible using this family of devices. Although the designs are based on the requirements of the EIA data communications specifications, they are not limited to these situations. Aircraft buses and internal equipment interconnections will benefit from the features offered by these products.

REFERENCES

1. Seshadri, S. R., Fundamental of Transmission Lines and Electromagnetic Fields, (U. of Wisconsin), Addison-Wesley, Reading, Mass., 1971.
2. Adler, R. B., L. J. Chu, and R. M. Fano, Electromagnetic Energy Transmission and Radiation, (MIT), John Wiley & Sons, New York, 1963.
3. Matick, R. E., Transmission Lines for Digital and Communication Networks, (IBM), McGraw-Hill, New York, 1969.
4. Reference Data for Radio Engineers, (ITT), Fifth Edition, Howard W. Sams & Company, Indianapolis, 1974.
5. Electronic Industries Association, 2001 Eye Street, N.W. Washington, D.C., RS Standard Proposal, RS-232C, August, 1969
6. Electronic Industries Association, 2001 Eye Street, N.W. Washington, D.C., RS Standard Proposal 1220, Rev. RS-422, September 21, 1976.
7. Electronic Industries Association, 2001 Eye Street, N.W. Washington, D.C., RS Standard Proposal 1221, Rev. RS-423, September 21, 1976.

APPENDIX 6: INTERFACING EIA TERMINAL EQUIPMENT TO THE EuroCUBE-65 SERIAL PORT

Introduction

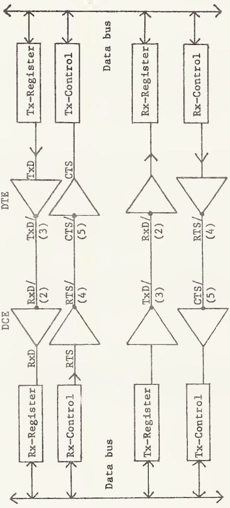
The following information will be of use to Control Universal customers who wish to utilise the integral serial port to generate customised asynchronous serial links to their own EIA terminals, or terminal equipment.

EuroCUBE-65 may be used in RS-423 mode, which is interconnectable with RS-232. The protocol is asynchronous, i.e. one character at a time using start and stop bits, and an optional parity bit.

The ACIA may be reprogrammed for various baud-rates, character length, and parity. Due consideration must be given to the ACIA software drivers on EuroCUBE-65, whether the drivers are specialised designs or whether the user elects to utilise the standard routines provided in the MOSB.3 EPROM.

Serial communications software drivers will be the subject of a future application note. This document explains the physical/functional interface (level 1 protocol).

Fig. 8: Typical EIA interface with use of handshake lines



EIA Serial Communication with Handshake

1) The RECEIVER CONTROLLER is ready to receive data, activating its RTS line (Request to Send - pin 4 on the 25-pin EIA RS-232 D-type Connector).

2) The TRANSMITTER CONTROLLER, at the other end, senses the received CTS line (Clear to Send - pin 5 on the EIA Connector) and sends out the subsequent data character on its TxD line.

3) Upon receiving the data, the RECEIVER CONTROLLER disposes of it in a buffer. It then decides whether it is ready to receive another character, and signals to the transmitter at the other end, using its RTS line.

Care should be taken to ensure that RTS-CTS is not deactivated while a character is being sent out. If it is deactivated, the transmitter may stop the character in the middle and cause a framing error.

This condition cannot ordinarily be ensured by the receiver during 'continuous' transmissions, and may only be prevented by the transmitter completing the 'byte in process' prior to stopping the next byte.

Application to EuroCUBE Serial Port

The EuroCUBE design uses the 6551 ACIA. Although a CTS input is provided, this may cut off the current transmitted character, so that it is less than useful as an interactive handshake line. However, our design engineers have remedied this problem by connecting the external CTS input (pin 5 on EIA D-connector equivalent to pin 5 on the 7-pin serial port DIN connector on EuroCUBE) to the DSR line on the ACIA.

The effect of this is to flag a status bit on the ACIA and to generate an interrupt request (IRQ). In this way, the MOS software receives an indication that the receiver at the other end of the line is either busy or ready to receive the next character, prior to loading that character into the ACIA transmit register.

A similar process - but in reverse - takes place between the terminal equipment and EuroCUBE, when EuroCUBE is receiving data. Once again the ACIA provision, namely RTS, is inadequate. This is because manipulating the control register RTS line (bits 2 and 3) has the undesirable side-effect of disabling interrupts. The solution offered by EuroCUBE is CA2 from the VIA (pin 39 on IC11), which is connected to pin 4 of the DIN connector and acts as an RTS line for the EuroCUBE receiver mechanism. This may be connected to pin 4 on EIA connectors.

Communications with By-passed Handshake

Interactive handshake mechanisms are particularly useful when transferring large blocks of characters at high baud rates, using simple asynchronous protocols. For lower baud rates or short transfers, the interactive handshake lines may be by-passed for simplicity. This must be done for the controllers to work, simply by shorting pins 4 and 5 locally on the DIN connector (similarly pins 4 and 5 on the 25-pin EIA DIN connector).

In EIA applications other pins are often encountered, namely:

```
DSR - (pin 6)  - Data Set Ready       - (input)
DCD - (pin 8)  - Data Carrier Detect - (input)
DTR - (pin 20) - Data Terminal Ready - (output)
```

The common solution is to by-pass pin 20 to both pins 6 and 8, or simply to activate pins 6 and 8 in a constant manner.

These signals are not required by EuroCUBE externally, although similar signals are used on the ACIA internally.

NOTE: EIA signals are 'Active Negative' by convention, i.e. MARK is a negative voltage and SPACE is a positive voltage on either RS-232 or RS-423 standards.

Summary

1. Interconnection with handshake

1.1

Terminal Equipment (Using RS-232 or RS-423)		EuroCUBE (In RS-423 mode)	
Signal Name	EIA Connector	DIN Connector	Signal Name
Rx-Data	PIN 2	PIN 1	Tx-Data
Tx-Data	PIN 3	PIN 3	Rx-Data
CTS	PIN 5	PIN 4	BUSY(RTS)
RTS	PIN 4	PIN 5	CTS
Signal Ground	PIN 7	PIN 2	0V
DSR	PIN 6	Not Used	
DCD	PIN 8	Short as	
DTR	PIN 20	required	
Protective Ground	PIN 1		

1.2

EuroCUBE (In RS-423)		EuroCUBE (In RS-423)	
Signal Name	EIA Connector	DIN Connector	Signal Name
Rx-Data	PIN 3	PIN 1	Tx-Data
TX-Data	PIN 1	PIN 3	Rx-Data
CTS	PIN 5	PIN 4	BUSY (RTS)
BUSY (RTS)	PIN 4	PIN 5	CTS
OV	PIN 2	PIN 2	OV

2. Interconnection with by-passed handshake

2.1

Terminal Equipment		EuroCUBE	
Signal Name	EIA Connector	DIN Connector	Signal Name
Rx-Data	PIN 2	PIN 1	Tx-Data
Tx-Data	PIN 3	PIN 3	Rx-Data
CTS	PIN 5	PIN 4	BUSY (RTS)
RTS	PIN 4	PIN 5	CTS
Signal Ground	PIN 7	PIN 2	0V
DSR	PIN 6		
DCD	PIN 8		
DTR	PIN 20		

2.2

EuroCUBE		EuroCUBE	
Signal Name	EIA Connector	DIN Connector	Signal Name
Rx-Data	PIN 3	PIN 1	Tx-Data
TX-Data	PIN 1	PIN 3	Rx-Data
CTS	PIN 5	PIN 4	BUSY (RTS)
BUSY (RTS)	PIN 4	PIN 5	CTS
0V	PIN 2	PIN 2	0V

NOTE: Pin 1 on RS-232 EIA connectors is termed 'Protective Ground', and is normally connected to the cable shield (when available) and to the equipment 'Chassis Ground', hence to 'Mains Earth'.

APPENDIX 7: CLOCK SOFTWARE

Control BASIC contains high-level commands to write to and read the real-time clock. If the user wishes to access the clock via the Operating System, the OSWORDs 14 and 15 can be used.

For a clock READ operation the accumulator must contain 14 (&0E), and the X and Y registers must contain the low and high bytes respectively of the parameter block address.

A typical READ operation in BASIC would be:

```
10  DIM data 15:OSWORD=&FFF1
20  X%= data MOD 256
30  Y%= data DIV 256
40  A%= 14
50  CALL OSWORD
```

The parameter block itself may be read using the indirecton operators, i.e.

```
100 PRINT "seconds ",~?data
110 PRINT "minutes ",~data?1
120 PRINT "hours   ",~data?2
etc.
```

If you are unfamiliar with these indirection operators, refer to the BBC User Guide p.409. If you are using assembly language routines, these parameters can be accessed by way of indexed addressing.

For a WRITE operation the OSWORD call would be made in the same way, except that

(1) the equivalent of line 40 in the READ program would become A%=15,

(2) the new values to be written into the clock would be put in the parameter block before the OSWORD call.

In a program to change the minutes and seconds but not the hours or date, the parameter block loading routine might look like this:

```
 90 REM load parameter block
100 ?data=00:REM 0 seconds
110 data?1=&30 :REM 30 minutes
120 FOR D%=2 TO 15
130 data?D%=&FF :REM rest unchanged
140 NEXT
150
160 X%=data MOD 256
170 Y%=data DIV 256
180 A%=15
190 CALL OSWORD
```

NOTE: In the M3000 an update cycle occurs every second and lasts for a maximum of 6 ms. If a READ is performed during the update cycle, the data on the four output pins of the M3000 go high, giving 'F' (1111). The software in the MOS allows for this and waits for the update to be completed. Interrupts are enabled during this waiting period. The clock chip has a complicated data hold time specification during write cycles. During factory testing, any chips failing to write after 20 successive attempts are rejected. For a valid WRITE operation, the data should be read back after the WRITE to ensure that the data has actually been transferred to the clock. See demonstration programs "Clock" and "Timer" for the implementation of this.

Interrupts

There are two possible sources of interrupts from the clock, namely the Timer and the Alarm.

The Timer produces an interrupt when it changes from 23:59:59 (hours, minutes and seconds) to 00:00:00.

The Alarm produces an interrupt when the Watch time coincides with that of the Alarm time. This interrupt has to be enabled by setting bit 1 in the status word.

Both interrupts are indirected through the event vector (EVNTV) at &220. The relevant event number for both interrupts is 8, enabled using OSBYTE 14 with X set to 8.

The response of MOSB.3 to these interrupts is:

1. to copy the 4 least significant bits from the status register into the X register.

2. to clear the interrupt flags in the status word.

Control then passes to the user's routine via EVNTV. The user can then identify the source of the interrupt from the X register.

Demonstration Software

Two programs are supplied to illustrate the programming of the real-time clock. These programs will, of course, only run successfully on the EuroBEEB, not on the BBC Micro.

'Clock' allows the watch to be set and then displays the date and time, updating the time every second. This program will run with MOSB.2 or any version of MOSB.3.

'Timer-plot' draws a graphics display and at the same time displays a real-time clock which is updated every 5 seconds. This program will NOT run on MOSB.2.

```
10REM:TITLE    ......Clock......
20
30REM CONTROL UNIVERSAL DEMONSTRATION PROGRAM
40REM Program displays calendar and clock
50
60X=9:Y=11
70OSWORD=&FFF1 :X=9:Y=11: @%=1
80DIM day$(7),month$(12),data (15), check (15)
90
100FORC=1TO7:READ day$(C):NEXT
110FORC=1TO12:READ month$(C):NEXT
120 CLS
130INPUT"CHANGE TIME "ans$
140trys=0
150IF LEFT$(ans$,1)="Y" PROCsettime
160IF trys>19 PRINT"Write data fault":END
170
180VDU 23;11,0;0;0;0:REM cursor OFF
190CLS
200A%=14:REM read clock
210X%=data:Y%=data DIV 256
220REPEAT
230S=?data:REPEATCALL OSWORD:UNTIL?data<>S
240
250PRINTTAB(X,Y)"Time  ";
260FORC=2TO0STEP-1:PROCtime(data?C)
270IF C PRINT" : ";
280NEXT
290
300PRINTTAB(X,Y+1)day$(data?6)
310
320PRINTTAB(X,Y+2);
330PROCtime(data?3)
340PRINT" ";month$(VALSTR$~(data?4));" 19";
350PROCtime(data?5)
360
```

```
370PRINTTAB(X,Y+3)"Week number "~data?7
380UNTILFALSE
390END
400
410DEF PROCtime(X)
420PRINTX DIV 16;X MOD 16;
430ENDPROC
440
450DEF PROCsettime
460FORC=0TO14
470wr=&FF:IF C>7 wr=&80
480C?data=wr:NEXT
490data?15=1
500PRINT"Week no. ";:PROCinput(7)
510PRINT"Week day ";:PROCinput(6)
520PRINT"Year     ";:PROCinput(5)
530PRINT"Month    ";:PROCinput(4)
540PRINT"Date     ";:PROCinput(3)
550PRINT"Hours    ";:PROCinput(2)
560PRINT"Minutes  ";:PROCinput(1)
570PRINT"Seconds  ";:PROCinput(0)
580
590REPEAT
600A%=15:X%=data:Y%=data DIV256
610CALL OSWORD
620A%=14:X%=check:Y%=check DIV256
630CALL OSWORD
640F%=1
650FOR C=0 TO 15
660IF data?C<>255 AND data?C<>check?C F%=0
670NEXT
680trys=trys+1
690UNTILF% OR trys>19
700ENDPROC
710
720DEF PROCinput(Z)
```

```
730INPUTans$
740IF ans$<>"" data?Z=EVAL("&"+ans$)
750IF Z=4 data?Z=EVAL(ans$)
760ENDPROC
770
780DATA Monday,Tuesday,Wednesday
790DATA Thursday,Friday,Saturday,Sunday
800DATA Jan,Feb,Mar,Apr,May,Jun
810DATA Jul,Aug,Sep,Oct,Nov,Dec
```

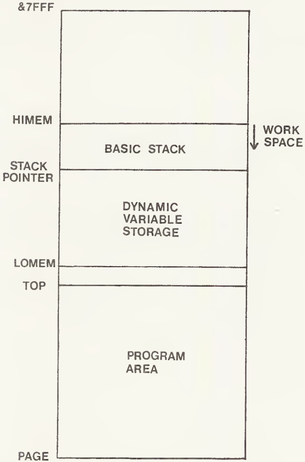
```
 10 REM TITLE ...... Timer-Plot
 20
 30 REM CONTROL UNIVERSAL DEMONSTRATION PROGRAM
 40 REM Program produces a graphics display
 50 REM and displays the time under real-time
 60 REM clock interrupt control.
 70 REM Program runs on Euro-BEEB
 80 REM (with MOS C.3,J.3,M.3 or T.3)
 90 REM with BBC as a terminal.
100 REM Will run on CU-GRAPH if PLOT 85,..
110 REM is changed to PLOT 5,.....
120
130 MODE0
140 PROCdefine
150 PROCassemble
160 CALL init :CALL start
170 CLS : PROCtime
180 PROCdisplay
190 END
200
210
220 DEFPROCdefine
230 REM Enable event #8 (Real-time clock event)
240 *FX14,8
250 OSWORD=&FFF1 : evntv=&220
260 VDU23,1,0;0;0;0;:  REM cursor off
270 @%=2 : REM define number field
280 DIM day$(7),month$(12),code 80
290 FOR C=1 TO 7: READ day$(C):NEXT
300 FOR C=1 TO 12: READ month$(C):NEXT
310 :
320 DATA Mon,Tues,Wednes,Thurs,Fri,Satur,Sun
330 DATA January,February,March,April
340 DATA May,June,July,August,September
350 DATA October,November,December
360 ENDPROC
```

```
370
380
390 DEFPROCassemble
400 FOR PASS% = 0 TO 2 STEP 2
410 P%=code
420 [ OPT PASS%
430         \ Set up clock parameter block
440.block  EQUD &FFFFFFFF
450        EQUD &FFFFFFFF
460        EQUD &FFFFFFFF
470        \ Time-out at 5 seconds
480        EQUD &11235955
490.flag   EQUB 0
500
510        \ Load event vector
520.init   LDA #intrv MOD 256
530        STA evntv
540        LDA #intrv DIV 256
550        STA evntv+1
560
570        \ Set up timer in the Real-time clock
580.start  LDX #block MOD 256
590        LDY #block DIV 256
600        LDA #15
610        JSR OSWORD
620        RTS
630
640       \ Interrupt service routine
650       \ X contains the 4 lsb's of
660       \ the clock status reg.
670
680.intrv TXA
690       AND #&08  \ Timer event?
700       STA flag
710.retrn RTS:]
720 NEXT
730 :
```

```
740 ENDPROC
750
760
770 DEFPROCtime
780 CALL start :REM reset timer
790 time$=CLOCK$
800 hours=VAL(LEFT$(time$,2))
810 minutes=VAL(MID$(time$,4,2))
820 seconds=(VAL(RIGHT$(time$,2)) DIV 5)*5
830 afternoon=hours DIV 12
840 hours =hours MOD 12
850 IF hours=0 hours=12
860 :
870 PRINT "              ": REM Remove last time
880 REM Now send time to screen
890 PRINT TAB(0,0);hours;":";
900 PRINT minutes;":";seconds;
910 IF afternoon PRINT " PM " ELSE PRINT " AM "
920 PRINT day$(DAY);"day "
930 PRINT (LEFT$(DATE$,2));" ";
940 PRINT month$(VAL(MID$(DATE$,4,2)));
950 PRINT " 19";VAL(RIGHT$(DATE$,2))
960
970 ?flag=0 : REM Clear the BASIC flag
980 ENDPROC
990
1000
1010 DEFPROCdisplay
1020 W=PI/15:V=PI/8
1030 GCOL3,1
1040 MOVE 640,932
1050 :
1060 :
1070 FOR Cycle%=0 TO 4
1080 READ A,B:W=PI/A:V=PI/B
1090 K%=A*B*2
1100 FOR Times%=0 TO 1
```

```
1110 MOVE 640,932
1120 FOR I%=0 TO K%
1130 PLOT 85,640+480*SIN(W*I%),512+420*COS(V*I%)
1140 IF ?flag PROCtime
1150 NEXT
1160 NEXT Times%
1170 NEXT Cycle%
1180 RESTORE
1190 :
1200 :
1210 DATA5,7,15,16,7,13,19,7,17,6
1220 ENDPROC
1230 END
```

Memory Usage with BBC BASIC:



PAGE, LOMEM and HIMEM are standard BASIC pseudo-variables.

RAM Storage Definition

It can be seen that all variable space sits between LOMEM and HIMEM and furthermore that by default LOMEM is immediately above TOP.

Both LOMEM and HIMEM can be easily changed by standard BASIC statements:

```
LOMEM=&xxxx

HIMEM=&yyyy
```

This is necessary when the BASIC program is in EPROM since LOMEM will otherwise exist in the EPROM space.

When planning to relocate the variable space it is necessary to know how much RAM will be required. As this is difficult to calculate, the following method of determination is suggested.

Let us assume that RAM exists between &E00 and &1FFF. The program should be run with the first statement (during the development of the application program):

```
HIMEM=LOMEM+&1200
```

This limits the RAM usage to that available when the program is in EPROM. If the ′no room′ error is not encountered on running the program, all will be well. However, if the ′no room′ error does appear, it will be necessary to compact the BASIC program by using the minimum number of variables, integer numbers instead of floating-point numbers wherever possible, and multi-statement lines separated by ′:′.

EPROM Definition

The M2 memory socket on EuroBEEB is addressed at:

&2000 - &3FFF using Map 0 (normally supplied)
&4000 - &7FFF using Map 1

The Maps are memory decoding options and these are described fully in Appendix 1.

By default the BASIC variable PAGE is set to &E00. This is the normal program start. If the application program in EPROM is now located at &2000, PAGE must first be changed to &2000 before the program can RUN. So that this can happen automatically, MOSB.3 allows the user to program an Autorun line. This is equivalent to typing an instruction on the keyboard. A maximum of 20 characters can be programmed into this line. To provide automatic power-up and run, for example, the Autorun line should contain:

PAGE=&2000<CR>
RUN<CR>

Programming the EPROMs

1. Application program

When the application program is finally developed and de-bugged, it is advisable to commit it to EPROM. Two types of EPROM programmer are available from Control Universal: CU-PROM and 'Softlife'. To program the EPROMs the following procedures are used:

(a) CU-PROM

The following description applies both to the CU-PROM used with the BBC Micro and to the System CU-PROM which works in conjunction with EuroBEEB. The latter uses software contained within the MOSM.3 which must, of course, be present.

First of all the application program must be loaded from disk into either the BBC micro or the EuroBEEB. This will normally be located at &0E00 on a EuroBEEB and at &1900 on the BBC micro. This is the 'buffer address' requested by the EPROM programmer software. Full instructions on how to use this software are provided in the User Manual accompanying the CU-PROM.

(b) Softlife

This EPROM programmer only works in conjunction with the BBC micro. The only prerequisite with this programmer is that the required application program is held on disk. For full instructions consult the User Manual accompanying the Softlife programmer.

2. Programming the Autorun line

This can again be done on the EuroBEEB or the BBC micro and requires 8K of RAM to hold the MOS-B.2 copy.

Initially the MOS-B.2 ROM software should be saved to disk using the normal *SAVE command, e.g.

```
*SAVE MOSB.3 C000 0000
```

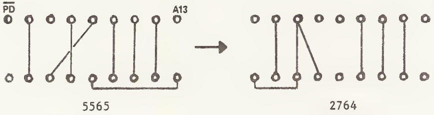
Then, in either the EuroBEEB or the BBC Micro, run the program called SFTLFE1 which is included on the utilities disk. This program amends the EPROM file so that it now contains the Autorun line.

Finally, the EPROM programmer is used to blow an EPROM copy of L.MOSB.3 which contains the Autorun line.

Installation of the EPROM

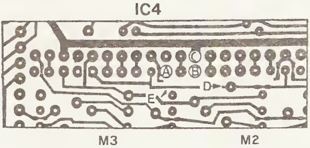
The application EPROM should be fitted to memory socket M2. In Issue 5 EuroCUBEs/EuroBEEBs, M2 is supplied linked for a 5565 8K RAM. The following modifications must be carried out to convert to a 2764 8K EPROM:

(a) Issue 5 with wire-wrap pins



(b) Issue 5 with the standard interconnections tracked on to the board

Modifications to fit 2764 in M2



1. Link A to B to C
2. Drill out hole D to 1 mm (no larger)
3. Cut track E

(c) In Issue 7 boards, M2 is fitted with an 18-pin socket into which the user can plug one of two headers supplied with the board. This obviates the need for board modifications and allows the user to change from CMOS RAM to EPROM with only a moment's work.

Special Consideration for the Serial Port

At switch-on, the operating system sends the following ASCII codes to the serial output port:

&0C (clear screen)
MOSB.3
&0D (carriage return)

(Peripheral driver software may extend this message.)

This is the normal reset message, and this will be placed in the RS-423 serial output buffer. If the serial handshake line is active high (+5V on pin 5 of the 7 pin DIN connector), this message will be transmitted. If the line is off (i.e. low -5V), the message will remain in the buffer.

If this start-up message needs to be suppressed, there are two ways to remove it. The first is to use a 'flush buffer' command at the start of the program using *FX15 ('flush all buffers'). In practice, the system may have already transmitted the 'clear screen' and probably the 'M'.

(This page has been left blank intentionally)

9. INITIALISATION TABLES

The initialisation tables provided below are designed for users with advanced applications. They enable the user to change the values of the variables so that their value at reset will be different. Probably the two main variables are the input type found at &F05A and the output type found at &F095.

For example, an application using the serial port to connect to another machine may require the reset message to be suppressed. This can easily be achieved by specifying no output channels in the table, i.e. &F095 = 06 (normally 07), which will cause any message to be sent nowhere. Remember to switch on an output channel in the application program if you want to use the serial port with *FX3,7 (for output definition of OSBYTE 3 see BBC User Guide and Advanced User Guide).

Address in EPROM	OSBYTE decimal	Initial value	Implemented	Description
F000		0F		Start-up byte
F001		F1A7		Language entry vector
Turnkey Line				
F003		FFFF, ...,FF		Turnkey line
Vector Table				
F019			Y	USERV user vector
F01B			Y	BRKV BRK vector
F01D			Y	IRQ1V primary interrupt vector
F01F			Y	IRQ2V Unrecognised interrupt vector
F021			Y	COMM command line interpreter
F023			Y	BYTEV FX/OSBYTE vector
F025			Y	WORDV OSWORD vector
F027			Y	WRCHV write character vector
F029			Y	RDCHV read character vector
F02B			Y	FILEV load/save file vector
F02D			Y	ARGSV file argument vector
F02F			Y	BGETV byte get vector

F031	Y	BPUTV byte put vector
F033		GBPBV group byte put/get vector
F035	Y	FINDV open or close file vector
F037	Y	FSCV file system control vector
F039	Y	EVNTV event vector
F03B		UPTV user printer vector
F03D		NETV network vector
F03F		VDUV unrecognised VDU vector
F041		KEYV keyboard vector
F043	Y	INSV insert into buffer vector
F045	Y	REMV remove from buffer vector
F047	Y	CNPV count/purge buffer vector
F049		IND1V
F04B		IND2V
F04D		IND3V

Variable table

F04F	166	0190	Y	Read start address of OS variables
F051	168	0D9F	Y	Read address of ROM pointer table
F053	170	02A0	Y	Read address of ROM information table
F055	172	0000		Read address of key translation table
F057	174	0300	Y	Read start address of OS VDU variables
F059	176	00		Read/write CFS timeout counter
F05A	177	01	Y	Read/write input source
F05B	178	00		Read/write keyboard semaphore
F05C	179	0E	Y	Read/write primary OSHWM
F05D	180	0E	Y	Start of language RAM
F05E	181	00	Y	Read/write RS-423 mode
F05F	182	00		Read key explosion state
F060	183	00	Y	Read/write cassette/ROM filing system switch
F061	184	0000		Read RAM copy of video ULA control register

F062	185			Read RAM copy of video ULA palette register
F063	186	00	Y	Read/write ROM number active at last BRK (error)
F064	187	FF	Y	Read/write number of ROM socket containing BASIC
F065	188	04		Read current ADC channel
F066	189	04		Read/write current maximum ADC channel number
F067	190	00		Read ADC conversion type
F068	191	FF		Read/write RS-423 use flag
F069	192	1E	Y	Read UART control register
F06A	193	19		Read/write flash counter
F06B	194	19		Read/write mark period count
F06C	195	19		Read/write space period count
F06D	196	32		Read/write keyboard auto-repeat delay
F06E	197	08		Read/write keyboard auto-repeat period
F06F	198	00	Y	Read/write *EXEC file handle
F070	199	00	Y	Read/write *SPOOL file handle

F071	200	00	Y	Read/write ESCAPE, BREAK effect
F072	201	00		Read/write keyboard disable
F073	202	20		Read/write keyboard status byte
F074	203	09	Y	Read/write RS-423 handshake extent
F075	204	00		Read/write RS-423 input enable
F076	205	00		Read/write cassette/RS-423 selection flag
F077	206	00		Read/write Econet OS call interception status
F078	207	00		Read/write Econet OSRDCH interception status
F079	208	00		Read/write Econet OSWRCH interception status
F07A	209	50		Read/write speech suppression status
F07B	210	00		Read/write sound suppression status
F07C	211	03		Read/write BELL channel
F07D	212	90		Read/write BELL envelope number/amplitude
F07E	213	64		Read/write BELL frequency

F07F	214	06		Read/write BELL duration
F080	215	81	Y	Start up error
F081	216	00		Read/write length of soft key string
F082	217	00		Read/write number of lines printed since last page
F083	218	00		Read/write number of items in VDU queue
F084	219	09		Read/write TAB character value
F085	220	1B	Y	Read/write ESCAPE character value
F086	221	D001		
	222	8001		
	223			
	224			
F08A	225	8001		Read/write function key status
	226			Read/write SHIFT + function key status
	227			Read/write CTRL + function key status
	228			Read/write CTRL + SHIFT + function key status
F08E	229	00	Y	Read/write ESCAPE key status
F08F	230	00	Y	Read/write flags determining ESCAPE effects

F090	231	FFFF	Y	Read/write IRQ bit mask for clock
	232		Y	Read/write IRQ bit mask for 6551
	233		Y	Read/write IRQ bit mask for system 6522
F093	234	00		Read flag indicating Tube presence
F094	235	00	Y	Read flag indicating clock
F095	236	07	Y	Output type RS423
F096	237	00	Y	Read/write cursor editing status
F097	238	0000		Read/write location &27E, not used by OS 1.20
	239			Read/write location &27F, not used by OS 1.20
	240			Read/write location &280, not used by OS 1.20
	241			Read/write location &281, used by *FX 1
F09B	242	05		UART command register
F09C	243	E0	Y	ROM 16 Page address
F09D	244	FF		Read/write soft key consistency flag
F09E	245	01		Read/write printer destination flag

F09F	246	0A		Read/write character ignored by printer
F0A0	247	0000		Read/write first byte of BREAK intercept code
	248			Read/write second byte of BREAK intercept code
	249			Read/write third byte of BREAK intercept code
F0A3	250	0000		Read/write location &28A, not used by OS 1.20
	251			Read/write location &28B, not used by OS 1.20
F0A5	252		Y	Read/write current language ROM number
F0A6	253		Y	Read/write last BREAK type
	254		Y	Read/write available RAM
	255		Y	Read/write startup options

APPENDIX 10: Link and Line Termination Options

(1) Links L2,3 connects zero voltage to the negative inputs of the diffential serial inputs. This allows the positive inputs to work in 'single ended' mode for RS-423 specification (for RS-232 operation, see Appendix 6).

(2) Fail safe resistors:
1kB input resistors are fitted on all inputs. These improve the internal biasing of the input devices so that an open circuit line will present a non-active signal to the UART (6551).

(3) Slew rate limiting:
C10 to C13 can be fitted to provide slew rate limiting of the output drivers. In most applications these can be ignored. On badly 'ringing' lines slower edges will stop the 'ringing' at the expense of a reduced baud rate.

(4) The board has provision for line terminating resistors. As supplied, these load resistors are not fitted. If you are using long lengths of serial cable, you will need to consider fitting them.

R9	terminates	CTS	TxENABLE
R14	terminates	RxD	Receiver input (RECEIVE DATA)

For a cable with a characteristic impedance of 120R, Duplex connection requires a resistor at the receiving end (e.g. 120R). However, party line configuration requires a resistor at both ends (e.g. 240R).

These values are for guidance only, and in a large system experimentation may be required. The solution lies in a compromise between 120R which provides maximum power transfer at a reduced signal to noise ratio, or 240R which causes a mismatch of 2:1 but no signal to noise reduction.

NOTE

Link 1 is not used on Issue 5 and later boards.

-5V INVERTER

EuroCUBE is normally supplied with link L5 made and no -5V inverter (7660) fitted. L5 connects the expansion bus pin 4b to the serial buffers. If the -5V inverter is fitted with its two capacitors C24 and C25, L5 will need to be broken. The maximum output current of the inverter is 20mA, at which point the voltage will drop to -4V (see p.16 for more details).

APPENDIX 11: Connectors

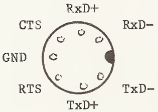
Bus Connector

B		A
+5V	1	+5V
+12V	2	A15
−12V	3	A14
−5V	4	NWDS
+15V	5	NRDS
−15V	6	RESET
HPAGE	7	A8
A23	8	A7
A22	9	A6
A21	10	A5
A20	11	A4
A19	12	A3
A18	13	A2
A17	14	A1
A16	15	A0
D15	16	D7
D14	17	D6
D13	18	D5
D12	19	D4
D11	20	D3
D10	21	D2
D9	22	D1
D8	23	D0
	24	A13
	25	A12
	26	A11
RDY	27	A10
IRQ	28	A9
NMI	29	phase 2 clock
SYNC	30	R/W
LPAGE	31	BLK
AGND	32	DGND

VIA Connector

GND	26	25	GND
nc	24	23	RESET
CA1	22	21	CA2
PA0	20	19	PA1
PA2	18	17	PA3
PA4	16	15	PA5
PA6	14	13	PA7
PB0	12	11	PB1
PB2	10	9	PB3
PB4	8	7	PB5
PB6	6	5	PB7
CB1	4	3	CB2
+5V	2	1	+5V

Serial Connector



socket view

APPENDIX 12

Brief History of EuroCUBE-65 (Issues 1 - 7)

Issue 1

This was a production prototype of which very few are in existence. The board had no solder resist, and there was a paging latch where the clock chip now resides. The serial buffers used were different from the current issue and would only work single ended, not differential. The memory decoder M1 provided only one map for MOS-B.1. Issue 1 EuroCUBEs cannot be upgraded to the current standard.

Issue 2

Main production started with Issue 2 boards. The boards were basically the same as Issue 1 but with solder resist. Issue 2 boards used CTS and RTS from the UART and different serial buffers (26LS30 and 26LS32). Boards were modified to use CA2 on the VIA for handshaking output and DSR for handshaking input. Wiring for the MOSTEK clock was present, but no chip could be fitted because the MOSTEK clock chip was suddenly withdrawn. The memory decoder was changed to M2 I/01 with eight maps. There was a fault with the CMOS 5565 RAM. Issue 2 EuroCUBEs cannot be upgraded to the current standard.

Issue 3

This production board replacing Issue 2 corrected the fault with the CMOS 5565 RAM and used the M3000 clock chip instead of the MOSTEK one. However, modifications still had to be made to the board in order to make the clock function correctly. The CB2 language select was cut and linked to disable. The memory decoder M2 I/01 remained the same as for Issue 2 boards. Issue 3 EuroCUBEs can be upgraded to the current standard.

Issue 4

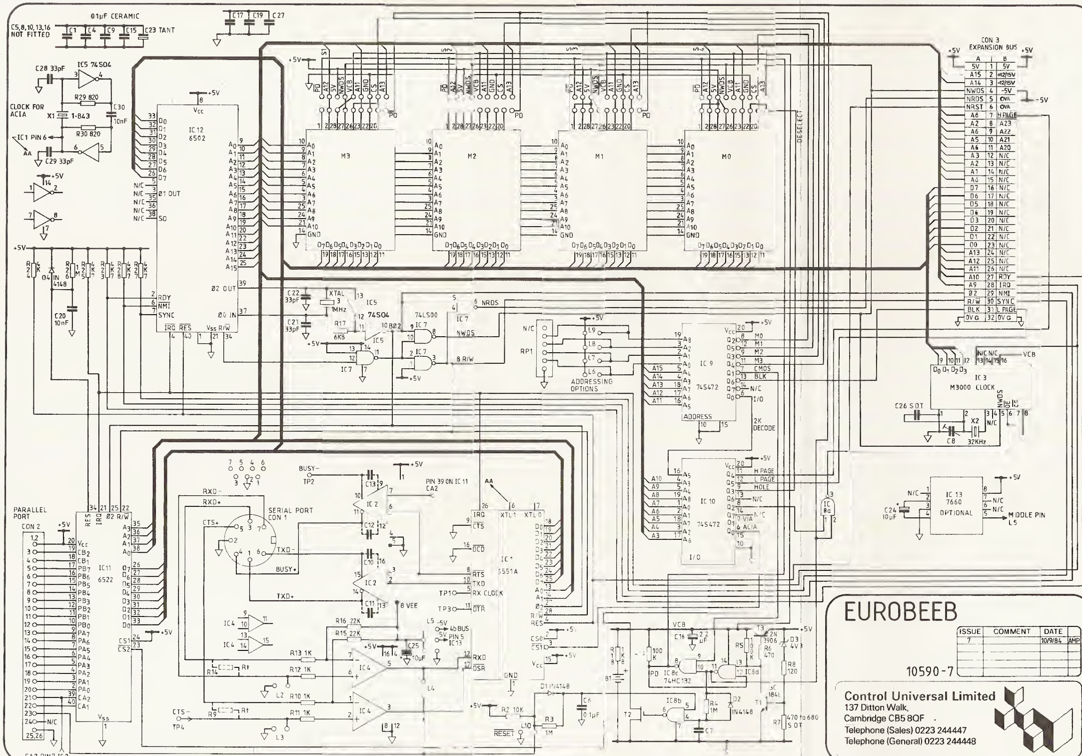
Issue 4 EuroCUBEs were pre-production boards only and never went into full production. The fault which had previously necessitated modifications for the proper functioning of the clock was corrected. The memory decoder was upgraded to M3 I/02. M3 had some different internal maps, and I/02 had an active high clock strobe.

Issue 5

The current issue went into production in May 1984. The memory map decoder M4 I/02 represents a complete redefinition with 16 maps. The CPU divider has been removed and replaced with a -5V inverter. EuroBEEB, the EuroCUBE-65 with BBC BASIC on board, was released simultaneously; it represents an Issue 5 board with an overlay for 5565 5565 27128 2764 memory pin-outs.

Issue 7

Issue 7 boards are fitted with two hard reset pins, giving the user the option to fit a hard reset button. Link 5 now includes three options for RS-423/422 operation. Memory socket M0 is configured as standard for a 16kB 27128 device. Memory socket M2 is supplied with an 18-pin header socket to facilitate the change between CMOS RAM and EPROM. A separate on-board oscillator is provided for the 6551 ACIA.


EUROBEEB
10590-7
ISSUE
COMMENT
DATE
7
10/9/84
AMP
Control Universal Limited
137 Ditton Walk,
Cambridge CB5 8QF
Telephone (Sales) 0223 244447
Telephone (General) 0223 244448
CON 3
EXPANSION BUS
PARALLEL PORT
CON 2
SERIAL PORT
CON 1
CLOCK FOR ACIA
ADDRESSING OPTIONS
DESELECT
2K DECODE
0.1µF CERAMIC
C5,8,10,13,16 NOT FITTED
IC 12
6502
IC11
6522
IC 9
74S472
IC 10
74S472
IC 3
M3000 CLOCK
IC 13
7660
OPTIONAL
M3
M2
M1
M0