ALEPH 88-155
DATACQ 88-29
A. Miotto
15.11.1988

The eeprom utility program

Alessandro Miotto.

This is the description of an OS-9 program that allows changes in the system parameters contained in EEPROM on Aleph Event Buiders and VIP processors.

• The program will abort on attempts to run it under OS-9 versions different from 2.2. An updated version will be released along with the EPROMS containing the next system.

• Any attempt to run the program on machines different from AEB's and VIP's will likely produce, in the best case, a bus error.

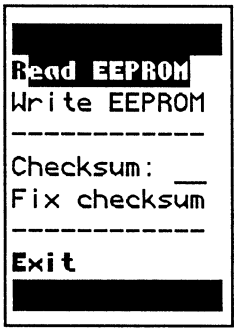
THE INTERFACE

The program interface is UPI_menu-like: cursors can be used to displace through menus, while the '-' and '+' key on the auxiliary keypad will change page in the current menu. While editing parameters, only legal key are accepted (e.g. only numbers in case of decimal values), and the DELETE key can be used to delete input characters. To clear an alphanumeric entry (i.e. to enter a null string) CRTL-SPACE has to be pressed.

THE COMMAND MENU

The first menu contains the following commands:

- **Read EEPROM** The system parameters are displayed in the second and third menu. The "Write EEPROM" item is enabled. If the EEPROM checksum is wrong the "Fix checksum" item is enabled.
- **Write EEPROM** The content of the EEPROM are updated. Only parameters that had their value changed are actually written. After the execution the cursor is put on the "Exit" item.
- **Fix checksum** Normally OS-9 cannot be started if the checksum is not right, but in case something has been written in the EEPROM by mistake, this can fix it.
- **Exit** Exit to the OS-9 shell.



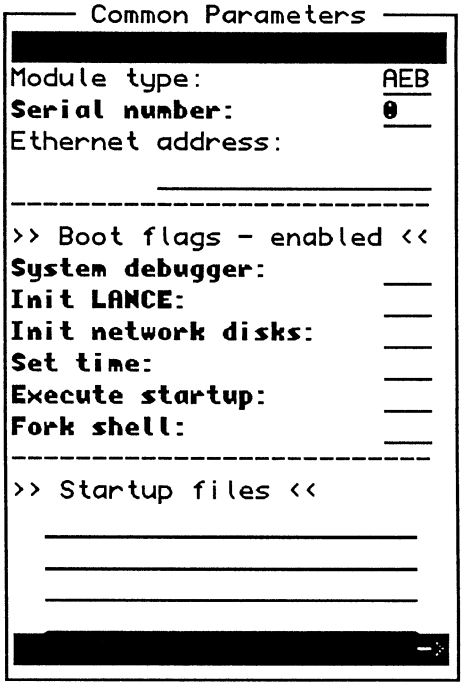
THE COMMON PARAMETER MENU

The second menu contains parameters that are common to AEs and VIP's (this is not completely true in the present version of the system, but the program will make it transparent):

- **Module type** This parameter is not editable. The program will recognize automatically the module type and update this field.
- **Serial number** For AEBs this parameter must be the same as the corresponding one in the AEB parameter menu. Allowed serial number for VIPs are 0 to 63.
- **Ethernet address** The ethernet address cannot be choosen arbitrarily: this parameter is automatically built form the serial number.

• **Boot flags** These flags are checked during the system startup. For all the following parameters YES (not case sensitive) has to be entered to enable the flag and NO to disable it.

- **System debugger** The system debugger is enabled at startup. A terminal must be connected to the main port to start OS-9 when this flag is enabled.



- **Init LANCE** The LANCE chip is initialised.
- **Init network disks** The network disk driver is initialised.
- **Set time** The time is read from the host and the clock is started.
- **Execute startup** The OS-9 commands displayed under "Startup files" are executed.

- **Fork shell** An OS-9 shell is forked to the main terminal.

• **Startup files** Up to five commands or command files can be specified. The maximum length for each command is 23 characters. Any semicolon in the string will be translated to a carriage return, so even if more than 5 commands can be entered, only the first five can be edited.

THE AEB PARAMETER MENU

This menu contains parameters that are used by Buggy. You may wish to update the basic identifiers when new versions are released, but it is better not to modify other things unless you are quite sure of what you are doing.

- **Serial number** The actual number is printed at the bottom of the front panel.
- **Hardware revision** Current revision is 2.
- **Buggy revision** Current revision is 1.10.
- **Microcode version** Last µcode version released is 2.29.
- **OS-9 version** Current version is 2.
- **OS-9 revision** Current revision is 2.

• **Buggy boot pointers** They are set by the SO command in Buggy.

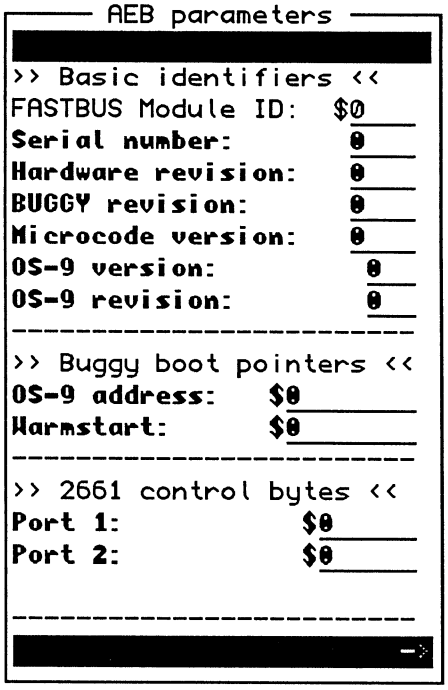
- **OS-9 address** Entry point of the operating system (currently $E12462).
- **Warmstart** Entry point of Buggy (currently $E000E8).

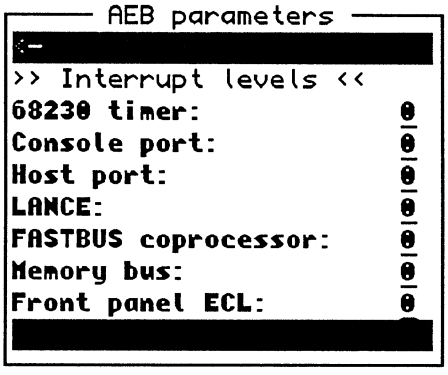
• **2661 control bytes** Refer to the Event Builder documentation.

- **Port 1/Port 2** Default is $6E7E26 for both ports.

• **Interrupt levels** Refer to the Event Builder documentation.

- **68230 timer** Default level is 6.
- **Console port** Default level is 5.
- **Host port** Default level is 5.
- **LANCE** Default level is 4.
- **FASTBUS coprocessor** Default lvl is 3.
- **Memory bus** Default level is 2.
- **Front panel ECL** Default level is 1.



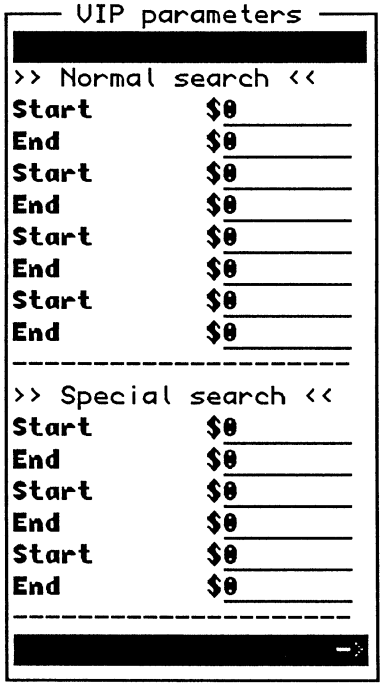


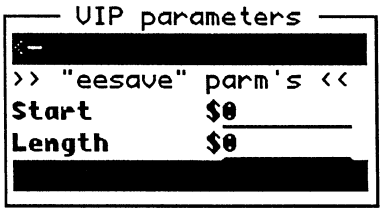
THE VIP PARAMETER MENU

• **Normal search** These parameters specify start and end addresses of normal OS-9 search areas, i.e. the unitialised RAM seen by the operating system. The first area is internal memory: it can be 192 or 768kbytes, depending on the RAM chips size. If the smaller chips are used, the search must end at $30000 (because the VIP does not produce a bus error in accessing higher addresses) otherwise the end address is $C0000. In any case the start address must be $5000. The user can specify up to 3 other different VME search areas.
The board is delivered with two default entries: $0–$30000 (internal) and $400000–$800000 (VME).

• **Special search** These addresses specify areas in which the operating systems looks for OS-9 modules, i.e. ROM or initialised RAM. The first area is set to internal memory ($800–$2000), and it is used to load at startup user modules contained in EEPROM. The second area is the internal ROMs space ($C0000–$1000000). A third area can be specified by the user.
Both special and normal entries are valid until the first null entry ($0–$0).

• **"eesave" parameters** These parameters are used to load OS-9 modules in the user part of the EEPROM. They are used by other utilities that will be delivered and documented separately.





eeprom.h

```
/* module_type */
#define AEB     1
#define VIP     2

typedef     unsigned char     byte;
typedef     unsigned short    word;
typedef     unsigned int      long;

                                    /* OFFSET */
struct      param_vip
{
    long    mem_list [16],          /* 0x00 */
            usr_prg,                /* 0x10 */
            usr_plen,               /* 0x14 */
            spare [14];
};

struct      param_aeb
{
    word    fastbus_id,             /* 0x00 */
            aeb_id,                 /* 0x02 */
            hw_rev,                 /* 0x04 */
            buggy_rev,              /* 0x06 */
            os9_rev;                /* 0x08 */
    byte    spare [6],
            eth_add [6],            /* 0x10 */
            eth_filter [8],         /* 0x16 */
            null_0 [2],
            vax_add [6],            /* 0x20 */
            null_1 [10];
    word    ucode_rev;              /* 0x30 */
    byte    null_2 [2];
    long    os_start,               /* 0x34 */
            warm_reset;             /* 0x38 */
    byte    null_3 [4],
            epci_1 [3],             /* 0x40 */
            epci_2 [3],             /* 0x43 */
            null_4 [10];
    struct
    {
        byte    timer,                  /* 0x50 */
                console,                /* 0x51 */
                host,                   /* 0x52 */
                ethernet,               /* 0x53 */
                coprocessor,            /* 0x54 */
                memory,                 /* 0x55 */
                 front_panel;           /* 0x56 */
    } intpt_lvl;
    byte    terminator [4], /* not aligned on longword boundary! */
            null_5 [37];
};
```

```
/*
** currently EBs are using only string_offset and alephgo_flags parameters
*/
struct  param_com
{
    byte    eeprom_id,                  /* 0x80 */
            module_type,                /* 0x81 */
            module_sn,                  /* 0x82 */
            string_offset,              /* 0x83 */
            eth_add [6];                /* 0x84 */
    struct      init_flags
    {
        unsigned
                    : 3,
        fork_shell  : 1,
        ex_startup  : 1,
        init_r0     : 1,
        init_disks  : 1,
        set_time    : 1,
        init_lance  : 1,
        start_dbg   : 1;
    }   alephgo_flags;              /* 0x8A */
    byte    spare_1 [112];
    long    checksum;               /* 0xFC */
};

typedef struct
{
    union
    {
        struct param_aeb    paraeb;
        struct param_vip    parvip;
    } mod_dep;
    struct param_com    parcom;
} sys_par;
```