Caveat: Just because it's documented doesn't mean it works that way.

Date Distributed: 04/02/1990

Requirements

Amiga 1000/500/2000
approximately 50K of CHIP RAM
approximately 400K of additional RAM

About Handebug

Handebug is a debugging tool specifically written for the Handy/Lynx project. It was designed to allow downloading programs and data files, examining and changing memory and registers, breakpointing and stepping through code, and uploading and saving memory. It will also communicate with Handecraft, providing a limited emulation of the Suzy display hardware. Handebug has a public message port allowing other programs to communicate with and control it on a limited basis.

All of the basic debugger functions are available from the main display. Handebug is graphically oriented, and most functions are accessible via the mouse. Keyboard equivilances are also provided where appropriate. Due to their nature, some advanced features are only accessible via either the mouse or the keyboard.

Almost all fields can also be edited via the keyboard. Data can only be entered in the correct format. A field containing decimal data will not accept keydata outside the range of 0-9. Changes made by the keyboard are immediately transmitted to the Lynx, when appropriate. The special keys '-' and '+' or '=' can be used to decrement or increment the value in a field. The cursor can be moved on the display by either clicking on the appropriate position with the mouse, or using the arrow keys to move around. Shift-arrow keys can be used to move between groups of fields.

The mouse can also be used to transfer data from one field to another. Double-clicking on a displayed value with the LEFT mouse button will usually copy that value to an internal paste buffer and temporarily display the selected value at the top of the display. Clicking on a field with the RIGHT mouse button will usually cause the value in the internal paste buffer to be deposited into that field, with any corresponding changes sent to the Lynx. A few fields don't respond to one or both of these actions.

Getting Started

Handebug can only be run from the CLI. To load Handebug, type "Handebug" or "Run Handebug" at the CLI prompt. Either method will work, although the second one is recommended since it will run the debugger as a background CLI task, leaving the CLI free to do other things. If you don't "Run" it, the CLI will be tied up executing Handebug, forcing you to open yet another CLI window to do other things, like editing or assembling files, and windows require screen (CHIP) memory which is at a premium when Handecraft is performing Suzy emulation. In general, the fewer windows open the better.

Handebug takes over the parallel port and opens a custom screen on the Amiga display. The screen can be pushed to the back or slid up or down using the drag bar (title bar) and depth-arranging gadgets at the top of the display. This

allows easy access to the WorkBench screen and the CLI windows, and to Handycraft's screen. At the upper left corner of the Handebug screen is a close box. To exit Handebug simply select this gadget with the pointer by pointing and clicking on it with the left mouse button.

Once the display is up, Handebug attempts to communicate with the Howard board. If it isn't talking yet there may be a delay of up to 3 seconds while the communications times out. At this point the debugger is functional.

If Handebug doesn't run there are a few things you can check. To initially load, Handebug requires about 40K of CHIP memory and about 400K of additional memory. If there is not sufficient memory it will just quit. Once it is running you can shut down the custom screen and recover all but about 4K of the CHIP memory. Also, Handebug needs the parallel port. The current Amiga printer drivers hang onto the port, even if you aren't printing. Once you've used (or tried to use) the printer the port is tied up forever. To get around this you must reboot the Amiga.

Handebug Files

Each time Handebug is loaded it looks for and reads a file called "Handy:Handebug.config". This file contains initial default settings for most of the options available in Handebug. It is possible to save the current state of Handebug in this file. If this file doesn't exist it may be created by saving the current settings via the Shift-F6 key. This will cause the file to be created with every setting established. The current settings can be reset to their default states at any time, also via Shift-F6.

Also read when Handebug starts up is a file called "Handy:Handebug.defs". This file contains the structure definitions used with the Structure Display feature. The current structure definitions can be redefined at any time via the Shift-F7 key.

Booting Handy (Non-Howard board users only

Before Handebug will communicate in any meaningful way with Lynx, Lynx must be bootstrapped with a communications "Monitor Program". This program communicates with Handebug using an established communications protocol. There are several versions of this code, each designed to work with a specific hardware configuration. Before attempting to bootstrap you should make sure that you have set up the correct version.

From the Handebug screen, select the "Bootstrap" gadget with the mouse or press function key F9 on the keyboard. This will cause the Bootstrap window to appear. There may be a delay of up to 3 seconds if Handebug thinks Lynx is trying to communicate, so be patient. At this point RESET the Lynx hardware to make sure it is in a known state.

NOTE to Apple][users: Power cycle the Apple][, or press Ctrl-Reset and type PR#1<ret>.

The Bootstrap window will contain the names and paths of the bootstrap code and monitor program. If your Amiga has been set up in the recommended way then you will not need to change these fields. Otherwise, you may edit these strings to reflect your specific situation.

At this point, select the "OK!" gadget. Handebug will read the bootstrap code and update the display, then try to download it . If it is successful the procedure is repeated with the monitor program. If the display flashes, and does not update beyond the bootstrap code which is loaded at address $0200 then there is a communications problem and procedure should be repeated. If it still fails, there is a definite problem. Check to make sure you are using the correct bootstrap and monitor code.

If you really screwed up the file names, or if you just decide you don't really want to bootstrap (maybe you got here by accident), select the "Cancel" gadget and the operation will be canceled. The path and file names will also be restored to the state they were in when you originally selected Bootstrap.

Primary Command Summary

Gadget	/ Key	
[GO]	F1	Initiate program execution from the beginning.
[Continue]	F2	Continue execution from current PC
[Step]	F3	Single Step the program
[Step Flat	F4	Single Step through function calls
[Download]	F5	Download a program or data
[Upload]	F6	Upload formatted or unformatted data
[Bus Monitor]	F7	Display the Bus Monitor interface
[Fill Memory]	F8	Fill Lynx memory with a value
[BootStrap]	F9	Boot a non-Howard board system
[LoadSymbols]		Load an assembler symbol table file
	F10	NMI
	Shift-F1	"Hide" Handebug - shut down Handebug display
	Shift-F2	Device/Volume name preferences
	Shift-F3	Customize Screen Colors
	Shift-F4	Download ROM image (Game Cart)
	Shift-F5	Execute program from ROM
	Shift-F6	Load/Save Handebug configuration
	Shift-F7	Load new structure definitions
	Shift-F8	Toggle Follow-PC mode of CODE display
	Shift-F9	Toggle internal debugging mode
	Shift-F10	Clear symbols (only when in Symbol mode)
	Ctrl-F1	Display the Search Trace Ram interface
	Ctrl-A	Sort the symbol display alphabetically
[Code]	Ctrl-C	Switch the main display to instructions
[Data]	Ctrl-D	Switch the main display to HEX data
	Ctrl-N	Sort the symbol display numerically
[Symbols]	Ctrl-S	Switch the main display to the symbol table
[Trace]	Ctrl-T	Switch the main display to Trace Ram
	Ctrl-Q	Quit Handebug (immediately)
[Rom]	Ctrl-R	Switch the main display to Lynx ROM

The Handebug Display

Handebug initially creates a display on a custom screen, which seperates it from the Workbench and other screens. This has the benefit of not cluttering up the Workbench display, which will probably be cluttered enough with your own windows for editing, assembling, or whatever. Unfortunately, this also requires

an additional 32K of precious CHIP ram. If the display isn't needed, this can be recovered by "hiding Handebug", as described later.

The display screen consists of Memory Watch fields and Breakpoint fields on the right, a general-purpose data display on the left, a register content display at top-center, a set of fields to control strucure displays at the bottom right, and a set of operation gadgets in the middle. The data display has gadgets at the bottom that can be used to switch it among code, data, symbols, trace-ram, and ROM

The Data display is a multi-purpose area of the display. It can be used to display one of several different sets of data. The cursor keys can be used to scroll the display up or down, and to move around the various fields in the display. The mouse can also be used to scroll the display by clicking and holding the LEFT mouse button inside the borders of the data display area, and then moving the mouse outside the borders. The rate of scrolling varies with the distance the mouse is moved outside the area. Scrolling stops when the mouse button is released.

The types and number of fields varies with the type of display selected.

Code Display...
When the Code option is specified the display will contain a disassembled listing of memory. The address fields may be edited to select particular addresses to disassemble. In addition, the hex opcode and operand fields may also be edited. Any changes are immediately downloaded to the Lynx. If symbols have been loaded they will be included in the disassembly.

The disassembled instruction mnemonics may not be edited. However, the operand field of the disassembly may be selected by double-clicking on it. The expression is evaluated using the current memory and register contents.

If an area of memory is selected that has not been fetched from the Lynx since the last GO or Continue command, then that memory will be read from Handy and the display will be updated. The display is not updated after a Step or Step Flat command. If a self-modifying instruction is single-stepped it will not display correctly until after that section of memory has been displayed as Data or until after the next GO or Continue command.

The display can be made to follow the current PC. If this option is selected, then each time Handebug gets control or switches to the Code display the display range is checked to make sure the current PC is within the displayed range. If it is not, the display is shifted to that address.

The disassembly begins at the address at the top of the display. It does not take into account non-executable data. Because of this it is possible for the display to not always be accurate. It sometimes may not be possible to position a symbol at any screen position other than the top, because the disassembly may not align that location on an opcode. Scrolling the display backwards often changes a considerable portion of the display for the same reason, since almost all values can be disassembled into some instruction.

Data Display...
If the Data option is selected the display area will contain a hex display of Lynx memory. The data is displayed as both hex and ASCII, and either field may be modified. Modified fields are immediately downloaded to the Lynx. When in this mode, the display is continually updated as long as normal communications

are established. This allows monitoring certain hardware registers in real-time. To facilitate editing, the field currently selected for editing (indicated by the presence of the cursor) will not be updated until the cursor is moved from the field. In essence it is "frozen" to allow editing.

Only HEX data will be accepted in the address of HEX data fields. Any 7-bit ASCII value that is not also a Handebug command can be entered into the ASCII portion of the display.

Symbol Display...
This mode is used to display the currently loaded symbol table. If no symbols are loaded the Load Symbols file requester is automatically displayed. Symbols can be either loaded automatically when Handebug is first loaded or when a file is downloaded to the Lynx, or manually by selecting the Symbols display or clicking on the LoadSymbols gadget. When symbols are loaded they can either be merged with an existing set of symbols or the symbol list can be reset.

The display order can be either alphabetic or numeric. When the Symbol display is selected, Ctrl-A causes the display to be sorted alphabetically, and Ctrl-N causes it to be sorted numerically. If possible, the line containing the cursor will remain constant during these operations, with the rest of the display changing around it.

Either field in the display can be selected, with the corrresponding numeric value being stored for subsequent pasting into another field. Neither field can be the recipient of a paste operation. However, the symbol name field can be edited. As the characters are entered the display will position the display to the first symbol to match the characters entered so far. This allows you to position the display to a particular symbol by placing the cursor in the name field and typing the symbol name.

Locating a particular routine is as simple as selecting the symbol from the Symbol display, switching to the Code display, and pasting the value into the address field of the display.

Trace Display...
This mode displays the Trace Ram on the Howard board. The Howard board supports up to 128K cycles of hardware tracing, and traces the Address BUS, the Data BUS, the Control BUS, and up to four external lines.

The cycle numbering convention is relative. Cycle 0 is defined to be the cycle where the first break condition occured. Positive cycles occured after that, and negative cycles before it. If there was no delay involved, such as with a software breakpoint, then cycle 0 is also the last cycle. When the Trace display is first selected after the Lynx has executed the display, it is positioned with cycle 0 at the bottom of the display.

Different cycle types are displayed with different color combinations to make them obvious. A cycle that has a break condition also has the trace display portion of the line highlighted.

Each line is broken into five fields. The first field is the cycle number. The second is the state of the Control BUS bits. The left-most bit indicates whether or not a break condition occured on that cycle. The next two fields are the Address and Data BUS contents. The last field is the cycle type. If it was an instruction fetch then the last field is the

opcode that was fetched.

Currently none of the fields may be selected for the paste buffer. The cycle number field may be edited to allow direct positioning of the display. Also, the standard scrolling mechanisms work.

Trace Ram may be searched for any combination of values by using the Search Trace Ram interface. Invoked via Ctrl-F1, this interface will search any specified range of Trace ram in either direction for any set of conditions that can be specified in the Bus Monitor interface. Searching can be aborted by pressing any key or mouse button, and if continued will start where it was aborted. When a set of conditions is found the display is positioned to that relative cycle.

ROM Display...
The ROM on the Lynx is a peculiar page-mode access system. The Howard board emulates the Lynx ROM, allowing you to download and test ROM data before actually burning ROMs. The default page size is set in the file 'Handebug.config', and may be changed by selecting the appropriate gadget when downloading the ROM data. The size changes take effect even if the download is canceled. To setup the Howard board to use the optional ROM cartridge slot you need to de-select all ROM sizes.

The ROM Display provides access to the Howard board ROM emulation as a linear, contiguous address space. The data may be viewed or modified without regard to page size.

Two ROM ports are available on the Lynx. The display can be switched to the other ROM by selecting the ROM display gadget again. Each time it is selected the display will toggle between the two ROMs.

Memwatch

The Memwatch fields are used to monitor specific memory locations in Handy. The fields may be one or two bytes and may be displayed as a single byte, two individual bytes (low-byte, high-byte), or a 16-bit value (high-byte/low-byte). The data portion is updated each time normal (not "slave") communications is re-established with the Howard board. The data field can be edited, and can contain any valid hex data. When the field is edited the corresponding memory in the Lynx is updated with the new value.

Breakpoints

The Breakpoint fields are used to set strategic breakpoints in Lynx memory. The address field can be manually edited, and can contain any valid hex address. To the right of the address field are two gadgets, Enable and Clear. The Enable gadget is automatically highlighted as soon as you enter or edit the address data. This indicates the breakpoint is enabled, and will be placed the next time the program is executed. Selecting this gadget will toggle the enabled status of the breakpoint, allowing you to temporarily disable a breakpoint without actually clearing it. The Clear gadget will clear the breakpoint and disable it. Entering an address of zero will also clear the breakpoint.

Handebug implements breakpoints by temporarily replacing the instruction at the location to stop at with a special opcode. In non-Howard board systems a BRK instruction is used. With the Howard board an opcode with a value of

x'n3' is used, where 'n' is in the range of 1 - 9. Breakpoints are placed just before continuing execution, after all memory changes are downloaded, and are cleared each time normal communications are re-established.

Conditional breakpoints are not possible with the standard breakpoints. However, elaborate conditions can be specified with the BUS monitor. The BUS monitor has four channels available, allowing up to four independant sets of conditions to be specified.

Registers

The Register fields are used to display and modify the contents of the Lynx registers. These fields are updated whenever normal communications are re-established, and are downloaded just prior to continuing execution. If the Go operation was specified the PC is also loaded with contents of the field to the right of the GO gadget. All of the registers may be modified, and may contain any valid hex data. In addition, there are a set of singe-character gadgets at the bottom of the display that reflect the current state of the status (P) register. The P register may be modified by changing the states of these gadgets.

Structure Display

The Structure Display/Edit creates another window above the normal Handebug display that is used for intelligent displays of structure contents. Structures can be defined before loading Handebug, and the display will format the data and display it with appropriate titles.

Structure types 0 through F are reserved for standard system structures. You are free to define any of your own starting with number 10.

Default structure definitions must be contained in a file called "handebug.defs" in the HANDY: directory. The structure definitions may be reloaded at any time from any file via the Shift-F7 key.

A structure definition consists of a sequence of definition entries. An entry consists of a keyword, sometimes followed by parameter data. Not all entries are required.

A structure definition begins with the keyword STRUCTURE, and terminates when either the next STRUCTURE keyword or the end of file is encountered. A structure consists of one or more fields. Each field definition begins with the keyword FIELD, and terminates when the next FIELD keyword is encountered or the structure definition is terminated. Each field can also contain one of more titles. Each title definition begins with the keyword TITLE, and terminates when the next TITLE keyword is encountered or when the field is terminated.

If a structure definition contains an error, then an error message is displayed in the CLI window that Handebug was started from. The message will indicate the line number containing the error and the name of the structure if possible. That structure definition will be discarded, and everything up to the beginning of the next structure definition will be ignored.

Blank lines are ignored, and may be used freely in a definition. Also ignored are lines that begin with a semi-colon, and anything following the keywords and their occasional parameters. Commenting a definition is thus not only possible, but highly encouraged.

Unless a FIELD definition contains an OFFSET specification it begins immediately after the previous FIELD.

```
STRUCTURE       - designates the start of a structure definition
  NUMBER xx       - specifies HEXADECIMAL structure number
  NAME string     - specifies the name of the structure (optional)
  POSITION x,y    - specifies screen position of window (optional)
  SIZE w,h        - specifies size of window (optional)
  OFFSET n        - a DECIMAL number that is an offset from the specified
                    address to determine the actual starting address

   FIELD          - designates the start of a field definition
     POSITION x,y  - specifies character coordinates of the field
     TYPE HEX      - specifies field to be displayed as HEX data
one  TYPE DEC      - specifies field to be displayed as DECIMAL data
TYPE TYPE BIN      - specifies field to be displayed as BINARY data
only TYPE TEXT     - specifies field to be used as a pointer to a string
     TYPE DBYTE    - specifies 2-byte field displayed in HEX as low/high
     TYPE POINTER  - field to be displayed as HEX data with bytes swapped
     TYPE STRUCT x - field is a POINTER to another structure of type x
one  SIZE 8        - specifies field is only 1 byte long
SIZE SIZE 16       - specifies field is 2 bytes long
only SIZE string length - for TYPE TEXT, the number of characters to display
     OFFSET n      - specifies a DECIMAL offset relative to the starting
                       address of the structure (optional)

     TITLE         - designates the start of a field title (optional)
       POSITION x,y  - specifies the character coordinates of the title
       TEXT string   - the actual title data
```

All fields except TEXT and STRUCT can be edited. STRUCT fields are special in that they are links to other structures. Double clicking on a STRUCT field will not select that value, but will instead cause that value to be used as the new structure address. This allows traversing linked lists of structures by double-clicking on the link fields.

Structure number 1 is a Sprite Control Block. Its definition may be used as an example for creating your own definitions.

Downloading and Uploading Files

Downloading data to the Lynx is a breeze. First, select the "Download" gadget or press the corresponding function key . This will bring up the "Download File" requestor. From this requestor you may select the file to download by selecting names or gadgets or typing names and/or paths.

When the requestor appears it will immediately begin loading file names from a a drive/volume and directory. This may not be the directory or volume you want. You do not have to wait for it to finish. You can specify another volume or directory at any time by selecting gadgets, names, or typing in the appropriate field.

You can get a file from any directory without ever taking your hands off the mouse. You can stroll through the disks by continually selecting the "Next Disk" gadget. Each time you do it will abort what it was doing and start

loading file names again. As soon as you can find the name of the directory in the file name area you can select it. Then when the actual file name appears you can select it also.

If you prefer, you can directly specify any or all of the disk, directory, and filename by selecting the appropriate string fields and typing them yourself. YOU DO NOT HAVE TO WAIT FOR FILE NAME LOADING TO STOP.

Once you have entered or selected the filename, press "OK" or double-click the filename to proceed. Handebug will open the file and transfer it to Handy. If a symbol file was created by using the +S option of the assembler Handebug can also read the symbol file and merge the symbols into the symbol table at this time. If an execution address was specified with the .RUN directive the field to the right of the GO gadget will be changed to the new execution address. If the PC is zero it will also be initialized.

Unformatted files may be downloaded by selecting the "Raw Data" gadget in the "Download Options" window that opens along with the "Download File" requestor. You may also clear the current symbol table before downloading the file by selecting "Clear Symbols".

Unformatted files or formatted files without a .ORG directive will download to the address specified in the requester.

Selecting "Cancel" will abort the download process.

Downloaded data is also stored in Handebug's internal image of Lynx's memory. It may then be changed by editing the data in the Data display window.

Data from the Lynx memory may also be uploaded in a variety of formats. When the Upload command is issued, either via the Upload gadget or fucntion key F6, a file requester is displayed along with a set of options. These options allow you to specify the range of addresses to upload, whether the data should be saved as ASCII or as raw binary, and whether the data should be formatted or unformatted.

Formatted binary data is similar to an assembler output file and is compatible with any program that processes assembler output. Formatted binary upload files may be downloaded just as assembler output files. Unformatted binary is exactly that; just raw binary data. Formatted ASCII is saved in the form of an assembler .SRC file, allowing it to be modified and reassembled. Unformatted ASCII is saved in the form of a HEX dump of memory.

Running (and Stopping) a Program

The whole purpose of Handebug is to allow debugging of programs written for and running on Lynx. You write a program, assemble it, download it, now you need to start it running.

There are several ways to run a program once it has been downloaded. All of them first restore the registers from the display, allowing you to change the registers as desired. Note: After downloading a program, it is good practice to press F10 (NMI) before selecting "Go".

"Go" and "Continue" establish all the breakpoints, begin execution at the current PC location, and continue execution until some mysterious force, such

as a breakpoint or NMI, causes the program to stop. "Go" has the additional feature of first setting the PC from the display field immediately to the right of the "Go" gadget.

The "Single Step" and "Step Flat" gadgets can be used to execute the next instruction and then return to Handebug. "Single Step" will always execute only one instruction and then return. "Step Flat" will also execute a single instruction. However, if the instruction is a JSR it will break on the return from the subroutine, rather than on the first instruction in the subroutine. This can be very useful if you are trying to follow a procedure that calls a lot of subroutines but you aren't interested in following execution through each and every subroutine. The normal breakpoints are established.

Once a program is running it has control of the machine and will continue to run until the hardware dies, the program destroys itself, or something Handebug did causes it to abort. There are several ways for Handebug to stop a program.

The trivial case is executing with "Single Step" or "Step Flat". The reasons should be obvious.

The most common and most useful method is by establishing breakpoints at strategic locations in the program. When a program's execution hits a breakpoint it causes control to return to Handebug, which updates its display to reflect the register contents and any changes to the page of memory currently being displayed. All breakpoints are also temporarily removed so that memory can be freely examined or changed.

Similar to breakpoints, yet totally different, is the Bus Monitor. The Bus Monitor is hardware that monitors the system bus for a variety of conditions that can be specified in Handebug. When the specified condition is met, control is forcibly returned to Handebug. This is incredibly useful when trying to track down random memory trashing, for instance.

Finally, the rudest (and most effective) method of interrupting a program is via the F10 function key, which causes an NMI (Non-Maskable Interrupt). The NMI will stop Lynx in its tracks, no matter what it's doing (hopefully). Generally this is most useful if the program's execution has escaped and gone to the land of no return.

Fill Memory

Handebug can be used to initialize Lynx RAM to a single byte value. The Fill Memory requester allows you to specify a starting address, the number of bytes to fill, and a fill value. All entries are expected to be in HEX. The requester is a standard Amiga requester, and therefore does not work the same as the other fields in Handebug. You cannot select values from these fields or paste values into these fields with the mouse. Also, data entry into these fields is INSERTED, not REPLACED. Therefore, if there is already data in the fields you must delete the old characters before entering the new ones.

Validity checking is not done until 'OK' is selected. If any of the values is not a valid HEX number, or if communications fails during the fill memory process, a cryptic and meaningless message "Ack!" is displayed.

Care must be taken to not overwite some areas. On Howard board systems the

last page of address space should not be filled. On non-Howard board systems care must be taken to not to overwrite the monitor code. If this happens the system will have to be re-booted.

Hiding Handebug

Handebug can be "hidden" to allow temporary recovery of the 40K or so of CHIP ram that it uses for the display. When the user presses Shift-F1 Handebug closes down all open windows and its screen, and opens a small window on the WorkBench screen. This window contains two gadgets that will allow you to re-open Handebug's display or exit Handebug. In addition to the window gadget, the display can be restored by pressing Shift-F1 again.

Most of the basic Handebug features are still available through this window. This window allows keyboard operation of the basic command set. File requesters that are required to upload or download data will appear on the WorkBench screen. Commands that are an integral part of the display, like switching the display mode, will have no effect. However, it is still possible to interrupt the Lynx via an NMI, download a file, and begin execution. Of course, the small Handebug window must be the active window when you attempt this.

The BUS Monitor (Howard board users only)

Handebug is the primary interface to the Howard board, a sophisticated hardware support environment for Lynx development. The Howard board contains ROM emulation, trace RAM, and four channels of extremely flexible bus monitor circuitry.

Function key F7 activates the Bus Monitor interface display. The display allows you to specify practically any set of values for the ADDRESS, DATA, and CONTROL busses, with independant delays for each channel. In addition, register values can be checked at each occurence, and sequences of conditions may be established with conditional looping to any point in the sequence. The types of cycles stored in the Trace RAM is also specified on this screen, and can be varied with each event. Trace RAM specifications from each active channel are 'OR'ed at each event. For instance, if one channel specifies tracing OpCodes and another specifies tracing Suzy and Video cycles, then all three will be traced.

At the upper left of the display are gadgets to manipulate sequences of events. Insert clones the current event, and inserts it immediately after the current event. It then positions you at the new event. All subsequent events are adjusted up one.

Delete will delete the current event, and adjust all subsequent events down one, leaving you at the same relative event. If there are no subsequent events it leaves you at the previous event. For instance, if you are displaying event 5 and there is no event 6, when you delete the current event you are now at event 4. You cannot delete event 1; you can only disable it.

Reset clears all the fields in the current event, and disables it. You MUST enable an event before it will take effect.

Previous and Next move you sequentially through a sequence of events. Additionally, you may edit the event number field to position directly to a particular event.

The channel gadgets A through D at the top select one of the four Bus Monitor channels. Each channel can have a unique sequence of conditions specified. Each channel can also have a delay associated with it. This delay is the number of cycles the hardware will wait before interrupting the Lynx. The delay can be specified as either CPU (Phase-2) or OpCode (Sync) cycles.

For each channel you can specify a set of BUS conditions to watch for. These conditions are expressed via the ramaining gadgets and fields. The OPCODE/DATA/SUZY/VIDEO, READ/WRITE, and LINEA/B/C/D gadgets represent conditions on the CONTROL bus.

Gadgets that conceptually represent different states of the same entity are logically 'OR'ed. Specifications for different entities are 'AND'ed within the same event. For instance, the OPCODE/DATA/SUZY/VIDEO selections are 'OR'ed. In other words, you can specify that a break should occur on an OPCODE -or- SUZY cycle. READ -or- WRITE are similar. However, when you specify both 'OPCODE -or- SUZY' and READ then a bus READ during an OPCODE or SUZY cycle must happen for a break to occur. Note that selecting all bus cycle types is the same as selecting none (a "don't care" condition). The same is true of READ/WRITE.

DATA and ADDRESS bus specifications are a little more complex. These conditions are specified through three fields. The first two fields represent a range of values. Only one range can be specified per event. To specify multiple ranges you will need to use the other channels. When the first data is entered into the range starting field it is copied to the ending value field to simplify breaking on a single value. Note that if you enter data with the keyboard the values will only match for the first character entered. Pasting with the mouse will copy an entire value to both fields.

The third field represents a mask that is applied to the value occuring within the specified range. Each position that has an 'x' represents a bit whose value can be either '0' or 1', as long as the total value falls within the specified range. When a position has either a '0' or a '1' specified, that bit posotion MUST contain that bit value for the condition to be true. For instance, to break on any odd address between 100 and 500 you wold specify a range with a starting value of 100 and an ending value of 500, and the mask would have all 'x's except for the last position, which would have a '1' ('xxxxxxxxxxxxxxx1').

In addition, the range of values can be specified as either inclusive ('In'), meaning within the values specified, or exclusive ('Out'), meaning outside the range specified. Also, the values wrap. If you specify a range of 500 to 100 'Out', it means the same as 100 to 50 'In'.

Each event also has an iteration count associated with it. If the count is non-zero, then it specifies how many times this hardware condition must occur before advancing to the next event. Also, this count must go to zero before any register checking or looping occurs.

When a break condition occurs, if there is an iteration count it is decremented by one. When the count goes to zero, if there are events specified on the same channel after the one that caused the break then the conditions are reset with the next event's conditions and the program

is continued. If there is no next event, then execution is halted and Handebug notifies you that a break condition occurred on that channel.

It is possible to branch to an event other than the next sequential one by using looping. Looping allows you to branch forwards or backwards in a sequence of events based on a TRUE or FALSE evaluation of registers and a single memory location. All the specified conditions must be TRUE for the result to be TRUE. If the result is FALSE, and the LOOP-TO field is non-zero and the LOOP-TO condition is 'Loop-on-FALSE', then the bus monitor will be reset with the conditions specified in the event in the LOOP-TO field. If the LOOP-TO condition is 'Loop-on-TRUE' then it will loop when the register and memory conditions evaluate as TRUE. Otherwise, it advances to the next sequential condition. If there is none, then execution is halted and Handebug notifies you that a break condition occurred on that channel. If no conditions are specified, then the result is always TRUE.

For instance, you can set up the Bus Monitor to only advance to the next event when a certain register contains a specific value by specifying that value in the register's range, and setting the LOOP-TO field to the current event number (the default). When a break occurs, if the register does not have that value the conditions are reset and execution is continued.

To access the register specifications and looping fields you have to re-size the Bus Monitor window to make it larger. The window normally opens too small to display that portion in order to avoid obscuring any more of the main display than necessary.

Communications

Handebug communicates with the development hardware through the parallel port. Data transfer is 8-bits wide. The printer control lines BUSY and POUT are used by Handebug and the monitor code to synchronize. When the system has achieved synchronization, then these lines indicate the current state of the ports. BUSY low indicates Handebug is in output mode. POUT low indicates the LYNX is in inout mode.

Port operation takes place at the interrupt level on the Amiga. Allowing the interrupt system to handle each character limits throughput to about 5K bytes per second. In order to improve this figure the port is polled once a transfer has started. The polling takes place in a tight loop inside the interrupt handler code. The number of iterations is limited to avoid significantly impacting system performance in other areas. The loop count value is configurable to allow tuning communications. This is primarily applicable to non-Howard board systems attached to Amiga 2500's. In these cases the Lynx hardware is much slower and the loop execution on the Amiga is much faster than a standard Amiga and will time out before the next character can arrive. This results in extremely poor performance. By adjusting the loop value it is possible to tune the system to achieve maximum performance.

When Handebug polls the parallel port it directly reads the ICR (Interrupt Control Register) associated with the port. This register is cleared by reading, so other interrupts that use this register may be lost during the polling process. The only one of significance is the keyboard. BECAUSE OF THIS IT MAY BE POSSIBLE TO LOSE KEYBOARD INTERRUPTS DURING DATA TRANSFER BETWEEN HANDEBUG AND THE LYNX.

Public Message Port

It is possible to control Handebug from another program. For specific examples on how to do this, see the example utilites "bootdown.c" and "download.c", and the HandyCraft and HSFXEditor sources. The possible commands are defined in the C language header file "handycmds.h".