PHILIPS

Interactive Systems Group

Philips Research Laboratories, Redhill, UK

Enhancements to Hardware Debugging System: A Discussion Document

Paul Clarke

Distribution

P.	Clarke	PRL, Redhill
J.	Piesing	
J.	Jacobs	IMS, Eindhoven
T.	van Oorsou	
C.	Golvin	PIMA, Los Angeles
R.	Moore	
M.	de Krock	PIMC, Hasselt
R.	van de Laarschot	
T.	Tabruyn	
T.	DiDomenico	Optimage, Chicago
T.	Wahlers	Optimage, Des Moines

H94/46 *Company Confidential* 6 May 1994

Authoring Workstations Project

Discussion Document
Release 1.0A-Alpha

Philips Research Laboratories, Crossoak Lane, Redhill, Surrey, RH1 5HA
Phone +44 293 815000 Fax +44 293 815500
Unix mail: paulc@prl.philips.co.uk

~/debugger/docs/features.xim

1. Introduction

Primary release CD-i Hardware Debugging Systems have recently been installed at a number of sites in Europe and the United States. During the installation visits the opportunity was taken to discuss future requirements of the system.

The CD-i Hardware Debugging System consists of a cartridge that connects to any consumer CD-i player through the Digital Video expansion bus, and a suite of software that runs on a host debugging PC. The system currently provides a single debugging facility known as the "inquest", which allows programmers to obtain a disassembly of player processor activity up to a given bus cycle. This trigger cycle could be an access to the error exception region of the player's memory map (to spot errors and null pointer accesses) or a breakpoint in the form of a specific bus cycle (given by address, data, etc). Data concerning bus cycles up to the trigger cycle can be up-loaded to the host PC where it is disassembled and presented to the user.

The inquest debugging facility constitutes a small subset of the total capabilities of the Hardware Debugging System. During the installation of the prototype systems a number of desired additional features were identified. These fall into three categories:

★ Extensions and enhancements to the inquest facility.

★ Debugging facilities not related to the inquest facility.

★ Miscellaneous support facilities.

This informal note forms a discussion document describing enhancements that have been suggested by the alpha sites. A work schedule will soon be specified covering development of the Hardware Debugging System through to the end of 1994. In order to produce a development strategy that satisfies the needs of as many alpha sites as possible feedback is needed on the contents of this discussion document. Are there any features that you would like to have that are not described? Do you disagree strongly with the proposed priority or time-table of a task? There is more work described in this discussion document than there is time to implement in the period under consideration so specification of the work schedule is very important.

The alpha period extends through to the summer of 1994. During this time some of the simpler but never-the-less valuable enhancements could be made. The period from the end of the alpha period to the end of 1994 could be spent implementing some of the more advanced features described in this discussion document. Your feedback concerning these types of features is especially important.

A questionnaire is included with this document - please complete it and fax back to me.

NB: This work is managed by Philips IMS, Eindhoven. All decisions regarding direction of work and time-tables for implementation are made by IMS management.

2. Extensions to the inquest facility

2.1 Show trigger cycle

When viewing inquest data an indication could be given of which cycle is the trigger cycle through a trigger mark in the inquest data. This need arises from the ability to hold off the trigger for a variable number of cycles leaving the trigger cycle somewhere other than the end of the inquest data.

This would require an enhancement to the disassembly code to pick out the trigger cycle from the rest of the data, and an extension to the inquest file format whilst maintaining backward compatibility.

I believe that this simple task is of high priority and could be completed in one of the first upgrade alpha releases.

2.2 Extend hold-off capability

The current system provides a hold-off choice during trigger specification. Hold-off allows the trigger conditions to take effect after they occur. In the current system either no hold-off can be selected or a hold-off period of 128 cycles can be selected. An extended hold-off facility would allow the trigger to be delayed by various amounts up to a maximum of 16384 cycles.

This would require extensions to the filter design. The filter circuit is very complex and squeezing more functionality into the filter device will be difficult.

I believe that task is of medium priority and increasing hold-off functionality could be introduced incrementally over a number of alpha releases.

2.3 Verified error exception trigger

The current filter design allows triggering on any access to the error exception region of the player's memory map. It would be useful to be able to trigger on a 'real' error exception occurring as the vector access trigger also detects null pointer accesses.

A new filter design derived from the current filter would enable a real error exception to be detected by looking for a long word read from a vector followed by a read from the address given by the contents of the vector.

I believe that this task is of high priority as it would enable easier debugging of titles which produce "blue screen" crashes as well as null pointer references. Currently if a disc makes regular null pointer references then they constantly trigger the cartridge preventing the crashes from being debugged. Verified error triggering could be supported in an early alpha release.

2.4 Printer output file format

The current system allows inquest data to be printed to a simple text file. This could be extended to include PostScript output and/or a rich text format so that cycle differentiation can be made (using bold, italics, etc).

I believe that this task is of low priority although some form of improvement over the plain ASCII currently produced could be implemented in an early alpha release. Thereafter increasingly elaborate output formats could be implemented in subsequent releases.

2.5 Look and feel enhancements

The look and feel of the host user interface will be enhanced based on use by programmers. This includes disabling unavailable options in menus, double click facility, etc.

I believe that this task is of low priority and enhancements are likely to be made in an incremental way through all alpha releases.

2.6 Preferences

The host software could allow programmers to set up their own preferences about definable features of the host software. User preferences such as sound prompts, inquest view options and trigger conditions could be saved when exiting the utility and automatically loaded when entering it.

I believe that this task is of low priority although a basic implementation could be included in an alpha release.

2.7 Player Module Directory file location

Currently the Player Module Directory (PMD) files are stored in the current directory. A typical test setup may use a number of different players, with and without Digital Video, resulting in a collection of PMD files. With the current system each project directory requires a copy of the appropriate PMD file which is both wasteful of disc space and setup time copying files. The player module directory files could be tied to a set directory (such as \INQUEST\MDIRS) and creating and accessing files in the correct subdirectory could be automatically supported by the host utility. The subdirectory could be a modifiable preference.

I believe that this simple task is of high priority and a single PMD directory location could be supported in an alpha release.

2.8 Help facility

The current system provides no on-line context sensitive help. Pressing the PC standard help key (F1) could provide the user with appropriate help depending on their position within the system. Additionally, the help menu could provide an index into all help areas. This could provide information concerning CD-i specifics to the user, including data concerning interrupt vectors, system call function codes and parameters, OS-9 global variable locations, and 68070 internal addresses, etc.

I believe that this task is of medium priority and could be implemented in an incremental way over releases of the software.

2.9 Project management

A programmer will frequently work on a title over a period of time. Each title testing session will use a CD-i player with a given module directory, and the title will have associated application modules and symbol tables. A project facility would enable assets relating to the testing of a given title to be grouped and loaded easily.

I believe that this task is of low priority and need not be supported for some time.

2.10 Raw inquest data viewing

A facility could be provided to view the raw data up-loaded by the host PC before it is disassembled. This would be particularly useful during the alpha phase of the Hardware Debugging System as it would provide programmers with access to their data even if the disassembly part of the debugging system fails due to a bug.

I believe that this task is of high priority. A facility could be supplied in the form of a separate .EXE file which converts a .DAT file up-loaded to the PC into a .TXT ASCII file for viewing with the editor of the programmer's choice. This conversion utility ("DAT2TXT.EXE") could be implemented for an early alpha release.

2.11 Interpret parameters passed to/returned from system calls

Currently system calls are identified by detecting the function code fetch made within the system call code. At the start of this code the contents of all registers are pushed onto the stack. Some of these registers (depending on the system call) are used to pass parameters. The parameters for a given system call could be identified and displayed automatically by the host software. Similarly, at the end of a system call the registers are popped off the stack and some of these will contain return values from the system call. These too could be identified and displayed.

I believe that this task is of high priority and could be implemented in the alpha release period.

2.12 C-Source tracing

The inquest data could allow interactive C-source tracing by making use of the .DBG file produced during compile-time. Stepping through the inquest data could produce an interactive trace through all available source files as found in standard debugging systems.

I believe that this is a high priority task but it would require considerable programming. There is no proposed time-table for this work.

2.13 Code memory masking

The user could be able to define regions of the memory map that they wish to see during code analysis. This would enable them to skip over interrupts and the contents of system call code and concentrate on the application code (or vice-versa for system programmers). Alternatively, the user would be able to just 'shrink' system calls and interrupts into their start and return code, seeing where they occur but not the detail of their implementation.

I believe that this is a medium priority task, the timetable for which has not been defined.

2.14 Index window

A window could be provided in the host software that displays an overview of the inquest data being viewed. This could contain a summary of information such as interrupts and returns, subroutine calls and returns, etc. Clicking on an entry in this window would make the inquest data viewing window move to display this state. The data for this window would form the basis for implementing other additional features (such as 2.13 Code memory masking described above).

I believe that this is a medium priority task, the timetable for which has not been defined.

2.15 Register display

A window could be provided which shows the value of every 68070 register as the user moves through the data. This information can be determined at disassembly time and presented automatically as the user steps through the inquest data.

Traditional intrusive debugging systems obtain register contents by saving all the registers on the stack on every trace interrupt occurrence. The non-intrusive debugging system would have to evaluate each register for each bus cycle during disassembly time.

I believe that this is a reasonably complex high priority task which could be introduced during the alpha period.

2.16 Full module/label description

The current system can display columns of information concerning module name and label name using data concerning the module directory and symbol stables of modules. These columns are limited to eight characters width each which can cause confusing truncation. An additional status line at the top of the inquest data window could display the full module and label names.

I believe that this is a simple medium priority task that could be implemented during the alpha period.

2.17 Full screen inquest data window

At present the maximum amount of inquest data viewable at one time is 29 states. A full-screen window would enable more than 40 states to be visible at one time. The default could remain at 29 states as this would allow viewing of other windows at the same time (such as module directory lists and register value windows).

I believe that this is a simple medium priority task that could be implemented during the alpha period.

2.18 Symbol decoding for globals

The current system uses the symbol table for program labels but not global variable references. All references to global variables (offsets from the global base address register) could be decoded.

I believe that this is a simple high priority task that could be implemented early in the alpha test period.

2.19 Trigger on address range access

A new filter design derived from the current filter could enable triggering on an access to an address range and not just a specific address.

I believe that this is a medium priority task that could be implemented during the alpha period.

3. Non-inquest enhancements

3.1 Intrusive debugging

The cartridge has the facility to generate interrupts, supply vectors and map ROMed OS-9 modules into the player's memory map. This could enable intrusive debugging functions to be performed.

I believe that this is a high priority task that would take considerable development. It could be done in an incremental way over a long period of time. It may be desirable to have some intrusive functionality implemented during the alpha test period.

3.1.1 Memory manipulation

A programmer would be able to manipulate (read and write) areas of a player's memory.

3.1.2 Module patching

A programmer would be able to perform binary patches of RAM-based OS-9 modules on the fly.

3.1.3 Tracing

Tracing through code an instruction at a time and setting breakpoints could be supported.

3.2 Memory area watching

A region of player memory could be defined which is watched for manipulation and displayed in a window. Format support would enable known structure definitions to be applied to the memory block to describe, for example, status blocks for real-time plays in fields that the programmer will recognise.

i believe that this is a high priority task that would require considerable development. It is unlikely that this work could be done during the alpha period.

3.3 Awareness of loaded modules

The loading and linking of modules by an application could be determined by the debugging system. This could be done by either watching for DMA loads of modules, or watching for the system calls which link a loaded module into the player's module directory.

I believe that is a high priority task that could be implemented early in the alpha test period.

3.4 Trace support

Programmers frequently want to be able to store trace information (rather like 'printfs') at various points in their program. A library of highly optimised subroutines would enable a write-port to be defined through which a programmer could send unformatted data with a format specifier to the host PC for displaying and storing in a text trace file. For example, a routine in the library called during title initialisation would define a write port. After initialisation the programmer could send various types of data to the PC through appropriate function calls into this library. Typical facilities could include sending ASCII data, character variables, integers, pointers, etc. Since formatting for display (such as converting a pointer to a string) would be done at the PC rather than the player, these library routines would be very fast.

I believe that this is a medium priority task, the timetable for which has not been defined.

3.5 Real-time system call tracing

The exact requirements for this are difficult to define. As a starting point and to provide an important debugging facility the real-time detection and analysis of memory allocation and deallocation system calls could be attempted. The user would be presented with information concerning what code requested which amounts of memory in what order, and the pointer to the resulting memory block.

I believe that this is a high priority task, the timetable for which has not been defined.

3.6 Profiling

By picking out every subroutine call and return the debugging system could provide valuable profiling information concerning which subroutines are called when, how often, and how long they take to execute.

I believe that this is a low priority task, the timetable for which has not been defined.

3.7 Path finding

Similar to profiling, the cartridge could produce a trace of how a user interacts with a title by determining which subroutines are called. Saving the formatted data to disk could provide a useful tool for those concerned with software testability.

I believe that this is a low priority task, the timetable for which has not been defined.

3.8 Bus usage analysis

The cartridge could determine and report in real-time the amount of bus usage being spent in DMA. Furthermore, it may be possible to detect (with a certain degree of accuracy) the proportion of time spent in user state and the proportion spent in system state. This could be done by designing a filter that detects interrupts (and system calls) and their returns and check the status register value stored and retrieved on the stack as a result of these operations (the status register has a bit determining CPU state).

I believe that this is a low priority task, the timetable for which has not been defined.

4. Miscellaneous support facilities

4.1 Parallel port communication

The current system uses a slow serial communications link to up-load data to the host PC. A capability has been built into the debugging cartridge to allow bidirectional parallel communication and this would vastly reduce the time for data up-load. Software for the cartridge and PC would produce the required functionality.

I believe that this is a high priority task that could be implemented early in the alpha phase.

4.2 Communications port selection

The serial port device driver could enable installation on either COM1 or COM2. Currently COM1 has to be used.

I believe that this is a simple low priority task that could be implemented during the alpha phase.

4.3 Host memory management

The current system only makes use of memory in the lowest 1 MByte on the PC. There are facilities to use extended or expanded memory where available. The memory management section of the host code could be enhanced to use this memory wherever possible. This would enable more elaborate debugging functionality to be perfomed.

I believe that this is a medium priority task that could be implemented during the alpha phase.

4.4 DOS shell facility

The host software could enable the user to exit to a DOS shell to perform some task.

I believe that this is a medium priority task that could be implemented during the alpha phase.

4.5 Load modules over comms link

Consumer players have the facility to load OS-9 modules over a serial link. Host PC's will usually have the cartridge connected to one communications port and a mouse connected to the other leaving no free serial ports. The debugging system could allow modules to be quickly down-loaded to the cartridge using the parallel link and then sent to the player from there (possibly using the unused serial port on the cartridge).

I believe that this is a low priority task, the timetable for which has not been defined.

4.6 Hardware development

Future hardware development could allow the inclusion of an area of dual-ported non-volatile static RAM on the cartridge where OS-9 modules could be down-loaded from the PC and included into the memory map. This would be especially valuable to people developing system software as they could quickly map a patched kernel or driver module into a player without burning any EPROMs.

Priority for this is unknown. A timetable for future hardware development has not yet been defined.

5. Conclusions

This report has described a large number of enhancements to the system that have been suggested during the installation visits. No doubt as people use the system a large number of additional ideas will be suggested. It is important that the order of these tasks is optimised so that as many useful features can be added as quickly as possible without neglecting some of the larger and more complex facilities that are required.

This is a discussion document that has indicated the perceived importance of these features and the expected time-frame for their development. I would appreciate as much feedback from as many users and potential users as possible before the timetable is defined in detail. If necessary a second document will be written incorporating new ideas and suggesting new time-scales.

A work plan for this project covering the rest of 1994 must be written very soon and so your prompt response to this report would be appreciated.

A questionnaire is included with this document - please complete it and fax back to me.

Enhancement Priority Questionnaire - Sheet 1 of 2

Task	Task Description	Low	Med/Low	Medium	Med/High	High
2.1	*Show trigger cycle*					
2.2	*Extend hold-off capability*					
2.3	*Verified error exception trigger*					
2.4	*Printer output file format*					
2.5	*Look and feel enhancements*					
2.6	*Preferences*					
2.7	*Player module directory location*					
2.8	*Help facility*					
2.9	*Project management*					
2.10	*Raw inquest data viewing*					
2.11	*Interpret system call parameters*					
2.12	*C-Source tracing*					
2.13	*Code memory masking*					
2.14	*Index window*					
2.15	*Register display*					
2.16	*Full module/label description*					
2.17	*Full screen inquest data window*					
2.18	*Symbol decoding for globals*					
2.19	*Trigger on address range access*					

Enhancement Priority Questionnaire - Sheet 2 of 2

Task	Task Description	Low	Med/Low	Medium	Med/High	High
3.1	*Intrusive debugging*					
3.2	*Memory area watching*					
3.3	*Awareness of loaded modules*					
3.4	*Trace support*					
3.5	*Real-time system call tracing*					
3.6	*Profiling*					
3.7	*Path finding*					
3.8	*Bus usage analysis*					
4.1	*Parallel port communication*					
4.2	*Serial communication port selection*					
4.3	*Host PC memory improvements*					
4.4	*DOS shell*					
4.5	*Module loading awareness*					
4.6	*Hardware development*					