HEWLETT PACKARD

Software Performance Analyzer

MODEL 64310A

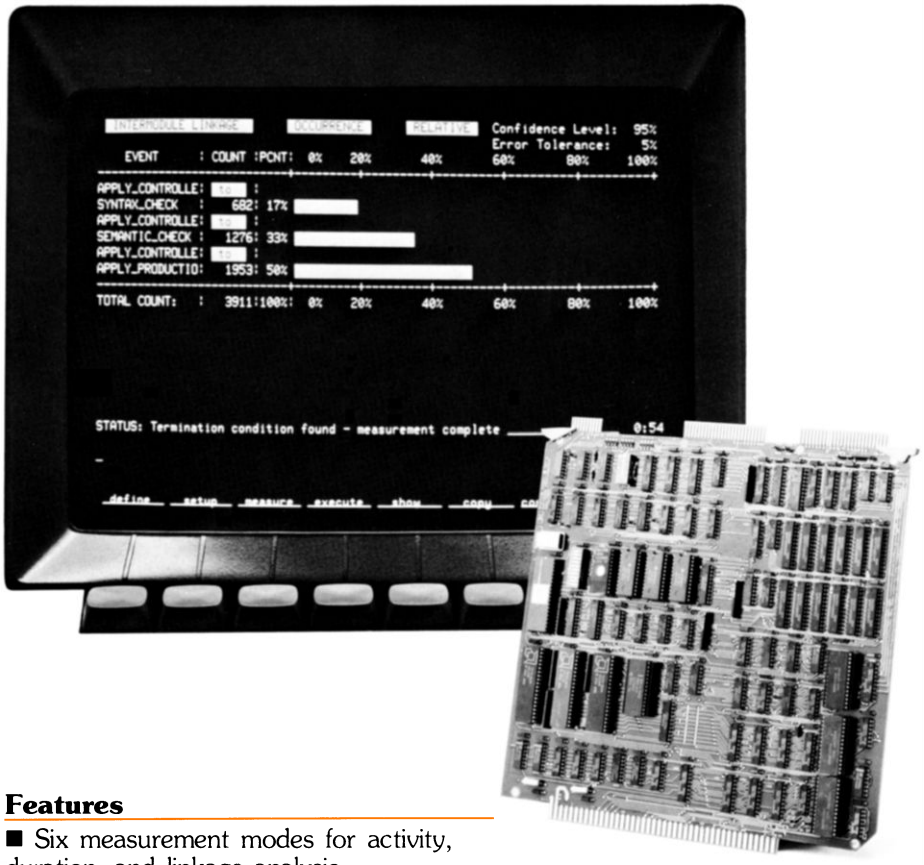
TECHNICAL DATA 1 MAY 83

Description

Model 64310A Software Performance Analyzer provides nonintrusive performance analysis of executing software. A subsystem of the 64000 Logic Development System, the Software Performance Analyzer is installed with a 64000 Emulation Subsystem for either 8-bit or 16-bit microprocessors. An innovative software development tool, the 64310A is used for software characterization, testing, debugging, and optimization.

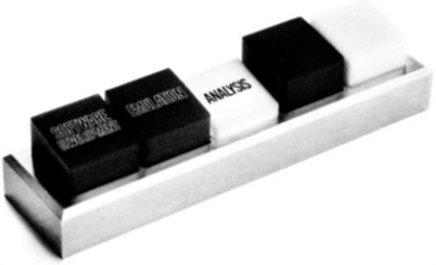
Six measurement modes offer a variety of perspectives on program execution. All modes monitor emulator-generated bus activity, allowing performance analysis to be conducted at any point in design, development, maintenance, and upgrading of processor-based systems, even when no target system hardware exists. Activity, Duration, and Linkage measurements are displayed as histograms or in tabular form. Displays and related sampling statistics are continuously updated during the measurement. The display may be halted for inspection while the analyzer continues the current measurement.

Data collection parameters are entered quickly and easily with directed-syntax softkeys. Symbols and labels generated in program assembly or compilation can be used directly in defining measurements. Measurement configurations are flexible, meeting a variety of application requirements. The analyzer can be specified to provide, for example, a global view of the entire memory space divided into 12 address ranges, or more detailed analysis, as in showing how frequently a subroutine is called by another subroutine. Measurements may be initiated and terminated manually, or automatically by setting enable/disable conditions. Alternatively, windows can be defined for repetitive data collection in a defined code segment for situations requiring context recognition.



Features

- Six measurement modes for activity, duration, and linkage analysis.
- Simple connection to 64000 Emulation Subsystems for 8-bit and 16-bit processors.
- Interactive measurements with 64620S Logic State/Software Analyzer, 64600S Logic Timing/Hardware Analyzer, and 64300A/64302A Emulation Bus Analyzers.
- Uses symbols, module names, and labels from programs written in Pascal, C, or assembly language.
- Global and detailed views of software operating in real time.
- Cumulative statistics for standard deviation, mean, confidence level, and error tolerance.
- Windowing capability for context recognition.
- Automatic instruction prefetch correction where possible.
- Analyzer specifications may be stored, then recalled for immediate return to a previous analyzer setup.



The Software Performance Analyzer becomes an integral part of a powerful set of design and development aids for processor-based systems. Model 64310A may be used interactively with other 64000 analysis and emulation subsystems. Compilers, assemblers and linkers contribute to fast, efficient and productive software development.

The modularity and configurability of the entire 64000 System provides for rapid acceleration of the design cycle. Convenience and ease-of-use combined with the measurement power of the 64000 System enhances design productivity, resulting in superior products and significant competitive advantages.

Memory Activity Measurement

The Memory Activity measurement mode provides software engineers with an indication of memory activity intensity in user-defined areas of memory. Activity is displayed either as a percentage relative to overall system memory activity, or as a percentage of activity relative only to monitored memory segments. Information may be displayed as a 12-bar histogram, or in a tabular data list with statistics such as standard deviation and mean.

Up to 12 memory areas may be defined using address ranges, single address values, module names, or program symbols. Data collection may be qualified further by including only specified bus activity. This qualification may include, singly or in combination, memory reads, memory writes, stack operations, opcode fetches, I/O or DMA activities, or other bus activity, depending upon status indications available from the processor being monitored.

Memory activity measurements can be displayed in a variety of formats. The measurement can be displayed in terms of activity count, or elapsed time and may be displayed in tabular as well as histogram form. Additionally, displays may indicate activity percentages relative to the events being monitored, or as a percentage relative to total system memory activity. Information from Memory Activity measurements gives the software designer a basis for allocating available memory more efficiently. As shown in figure 1, such a measurement can point to areas where memory allocation may be too large, or too small. The Memory Activity mode is also extremely valuable in evaluating code space requirements, or code space vs algorithm optimizations.

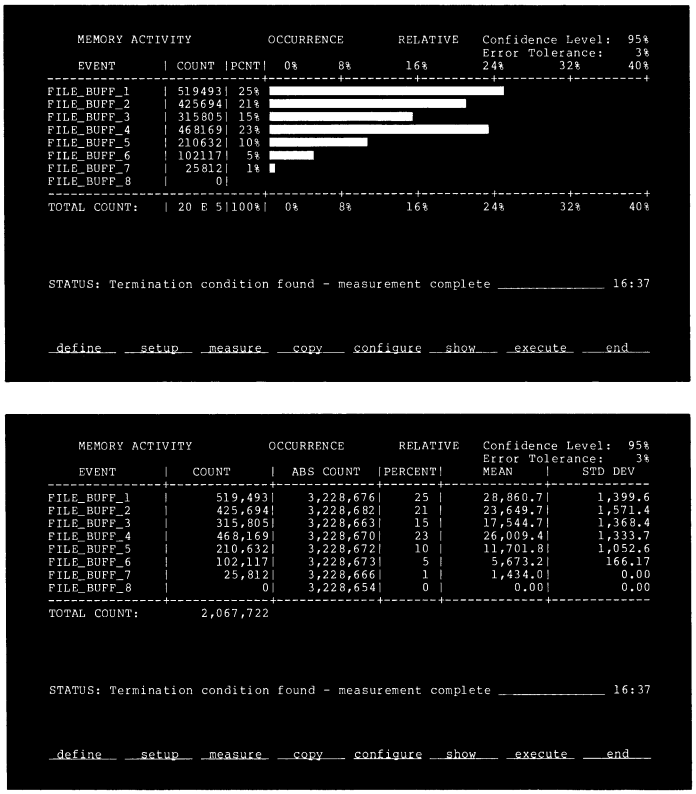


Figure 1. Memory Activity histogram (top) and the corresponding tabular form (bottom) showing file buffer usage during program execution.

Program Activity Measurement

The Program Activity measurement mode monitors opcode/instruction fetch activity. In contrast to the Memory Activity measurement, it also records I/O, stack, memory, and other activities initiated by these instructions (figure 2).

In the Program Activity mode, an instruction fetch cycle is recorded when it falls within one of the modules or address ranges being monitored. Other bus cycles (e.g. memory reads/writes, I/O operations, stack pushes/pops, etc.) are included in Program Activity measurements if they occur after the starting address for the specified module is encountered, and before the opcode at the ending address for that module is executed.

Up to 12 modules, or address ranges may be monitored during a Program Activity measurement. Both accumulated time and program-initiated bus cycle counts are measured and displayed as histograms or tables.

The Memory Activity measurement, discussed previously, is primarily directed to providing utilization analysis of nonprogram memory segments. Application of this measurement to program segments can, however, provide meaningful results, especially in the pursuit of module code size or algorithmic optimizations. Information gained from the Memory Activity measurement, as it may be specified to encompass time and bus cycle count information for instruction fetch cycles only, is often used in conjunction with the Program Activity measurement to determine the nonprogram memory reference intensity of a given module. This often results in algorithmic improvements reducing or eliminating possible memory reference bottlenecks.

The difference between Memory Activity and Program Activity measurements is sharply delineated in analyzing a procedure dedicated to memory-swapping functions. Such procedures are often implemented on processors having single instructions capable of performing block memory transfers. Thus, a single module may contain only a few instructions, resulting in a relatively low instruction count or time representation when viewed with a Memory Activity measurement. The same module when viewed with a Program Activity measurement may be seen to account for a major portion of the total program activity.

Trace List — State 6,120 channel, 8085 interface

Label:	ADDRESS	8085 Mnemonic	STATUS	time count
Base:	hex	hex	hex	rel
Map:	ADDR MAP	ADDR MAP	STAT MAP	
trigger	TEST+0000	XRA A	Opcode	1.48 usec
+001	TEST+0001	OUT 28	Opcode	2.00 usec
+002	TEST+0002	28 memory read	Mem_read	2.00 usec
+003	abs 2828	00 i/o write	IO_write	1.52 usec
+004	TEST+0003	MOV A,M	Opcode	1.48 usec
+005	abs 0BFD	88 memory read	Mem_read	2.00 usec
+006	TEST+0004	OUT 38	Opcode	1.52 usec
+007	TEST+0005	38 memory read	Mem_read	2.00 usec
+008	abs 3838	88 i/o write	IO_write	1.48 usec
+009	TEST+0006	MOV A,B	Opcode	1.52 usec
+010	TEST+0007	OUT 28	Opcode	2.00 usec
+011	TEST+0008	28 memory read	Mem_read	2.00 usec
+012	abs 2828	08 i/o write	IO_write	1.48 usec
+013	TEST+0009	RET	Opcode	1.52 usec

STATUS: Awaiting state command - userid TEST1 — 11:34

display <LINE #> disasmb show execute ---ETC---

Trace List — State 6,120 channel, 8085 interface

Label:	ADDRESS	8085 Mnemonic	STATUS	time count
Base:	hex	hex	hex	rel
Map:	ADDR MAP	ADDR MAP	STAT MAP	
trigger	TEST+0000	XRA A	Opcode	1.48 usec
+001	TEST+0001	OUT 28	Opcode	2.00 usec
+002	TEST+0002	28 memory read	Mem_read	2.00 usec
+003	abs 2828	00 i/o write	IO_write	1.52 usec
+004	TEST+0003	MOV A,M	Opcode	1.48 usec
+005	abs 0BFD	88 memory read	Mem_read	2.00 usec
+006	TEST+0004	OUT 38	Opcode	1.52 usec
+007	TEST+0005	38 memory read	Mem_read	2.00 usec
+008	abs 3838	88 i/o write	IO_write	1.48 usec
+009	TEST+0006	MOV A,B	Opcode	1.52 usec
+010	TEST+0007	OUT 28	Opcode	2.00 usec
+011	TEST+0008	28 memory read	Mem_read	2.00 usec
+012	abs 2828	08 i/o write	IO_write	1.48 usec
+013	TEST+0009	RET	Opcode	1.52 usec

STATUS: Awaiting state command - userid TEST1 — 11:34

display <LINE #> disasmb show execute ---ETC---

Figure 2. The highlighted area in the logic analyzer trace (top) is the information recorded in a Memory Activity measurement of module TEST, with status qualification set to record only opcode fetch cycles. The highlighted area of the trace (bottom) represents information acquired in a Program Activity measurement of module TEST.

Since the Program Activity measurement monitors all activity generated by a module, the designer can quickly isolate modules that cause large amounts of activity within a program (figure 3). These modules, then, are principal targets for the most beneficial optimization efforts.

Module Duration Measurement

The Module Duration measurement mode generates time distribution histograms representing execution times of a specified module or block of code. Time distribution measurements allow characterization and verification of best-case and worst-case execution times. By highlighting modules consuming inordinate amounts of processing time, sources of overall system degradation are identified. Spurious execution times, as a result of passing faulty parameters or improper algorithms for example, become highly visible in this measurement.

Up to 12 time events may be specified in Module Duration measurements. These events may represent time ranges with minimum lower limits of 1 μs (depending upon the bus cycle speed of the processor being monitored) to maximum upper limits of over 11 minutes. Initial Module Duration measurements often span a relatively large total time interval. As the the investigation proceeds, it is possible to focus on specific time events, by using smaller time ranges defined more closely around pertinent time values. The resulting magnification of data provides better measurement resolution.

In interrupt-driven systems, for example, module timing measurements are often difficult to interpret depending on the regularity and frequency of interrupts and their associated service routines. Under these conditions, it is useful to employ the option of either including or excluding activity external to the module of interest. This feature allows the designer to consider the time spent in subroutines or functions referenced by the module, as well as time spent in the execution of interrupt-activated code. Subsequent measurements of module duration excluding, then including such activity, allows the engineer to gain a more complete understanding of external procedures, functions, interrupts, etc., and their effect on system activity at the module level.

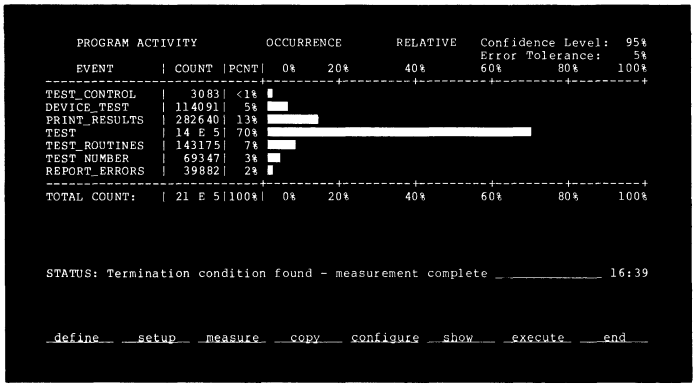


Figure 3. Program Activity measurement indicating that module TEST represents 70% of the overall system activity.

Another application is the extrapolation of hardware performance information from software performance data. For example, assume that a single module is responsible for correctly reading a data block from a disc after the head has been positioned. If this assumption is the case, the principal factor responsible for a variance in the module execution time is the number of read retries necessary to correctly transfer the requested information. Module Duration measurements can easily bring such information to the attention of the engineer via histogram or data list displays. Implications so derived can result in the discovery of faulty media or improper hardware communication, not only in the lab, but in production and testing environments as well.

Module Usage Measurement

The Module Usage measurement mode provides a distribution of the time available for execution of other tasks after a specific module executes. It indicates the intensity of demand for the services of a module. This results in an extremely useful measurement for identifying program areas where optimization efforts can be most effective.

The Module Usage measurement is the complement of the Module Duration measurement, in that it measures the time from a module completion to the time that same module is used again. Figure 4 contrasts the two measurements.

In a typical application, the Module Usage measurement may reflect low demand for a specific module, allowing other scheduled tasks to occur normally. A small percentage of the time, the measurement may indicate heavy module usage, preventing other system tasks from being performed at all. This usage measurement is a valuable pointer to task scheduling problems, indicating the need for operating system level modificatons and task optimization. Program modifications can then produce more effective and efficient task scheduling, resulting in greater overall system throughput.

Up to 12 time ranges from 1 μs (depending on the processor being monitored) to 11.18 minutes may be specified.

Intermodule Duration Measurement

The Intermodule Duration measurement produces a distribution of the time intervals between successive executions of two specified modules. The time duration is measured between execution of the last instruction of the "from" module, and the first instruction of the "to" module (figure 5).

Here, as in other Duration modes, up to 12 time range events may be specified ranging from 1 μs to 11.18 minutes. The lower limit of 1 μs, is dependent upon the bus cycle rate of the processor being monitored.

Consider, for example, a program whose execution eventually requires a new software overlay. In this scenario, the main program might call a subroutine used to set up an external hardware transfer mechanism (module XFER). Module XFER will then transfer the overlay code from a relatively slow mass storage device into a high-speed memory buffer, while the main program continues its task. As the main program completes the task in progress, another module is called to load the overlay from the high-speed memory buffer into program memory (module LOAD_NEW). After these steps, the main program can then jump into the newly overlayed code and continue execution.

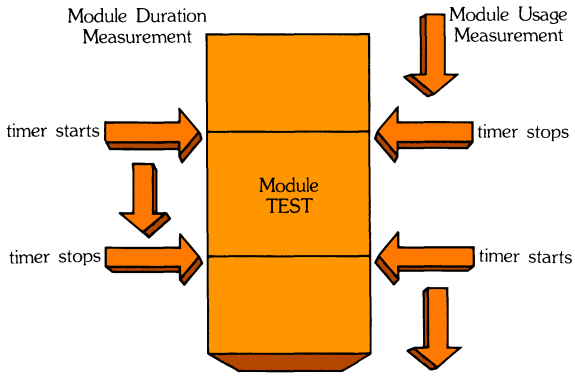


Figure 4. The Module Usage measurement is the complement of the Module Duration measurement.

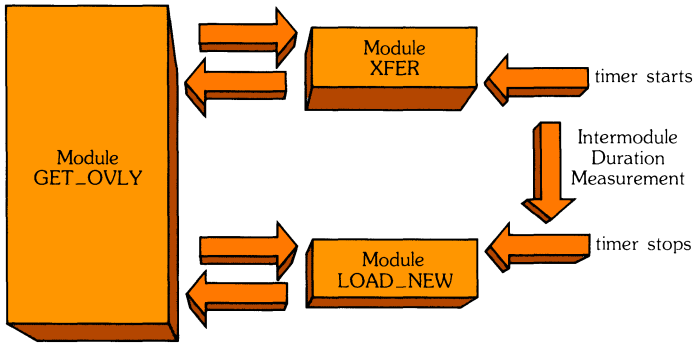


Figure 5. Diagram of an Intermodule Duration measurement from Module XFER to Module LOAD_NEW.

High-speed, parallel-activity, memory-swapping mechanisms are used in situations where program execution cannot remain idle for the time necessary to transfer directly between disc and program memory. But, by executing the slow portion of the transfer in parallel to essential program activity (in effect, transparently) the total impact on system performance is relatively minor.

For overall system performance considerations, the time interval between the start of the transfer from disc, and the request for the high-speed load into program memory is often critical. If the interval is too small, the main program must idle; if the interval is too long, the main program may require optimization. From another standpoint though, if optimization is not possible, hardware cost reductions and associated software simplification may be realized by reducing the speed at which the transfer mechanism must operate, possibly eliminating the need for such a capability at all.

This type of analysis provides a strong basis for resolving module interaction problems. Transfer timing between modules is often critical, especially when software is interacting with other software or external hardware. Overall system performance can be markedly improved by first identifying worst-case program paths, and then optimizing the interaction involved.

Intermodule Linkage Measurement

The Intermodule Linkage measurement mode provides direct visibility of module-to-module transfers for the analysis of program flow paths versus path usage intensity. Intermodule Linkage measurements monitor program control flow between a base module and its subordinate modules.

Up to six module pairs may be specified for measurement in a "from module/to module" form. The Linkage measurement indicates the number of direct program transfers from a specified module to another selected module as a percentage of all transfers from the "from module(s)" specified or as a percentage of transfers between the module pairs specified.

The Intermodule Linkage measurement shown in figure 6, displays A to B transfers and A to C transfers as a percentage of only those transfers from A to B or A to C. Alternatively, the measurement could show A to B and A to C transfers as a percentage of all transfers from A. This is especially helpful in spotting unexpected transfer conditions that should not be occurring.

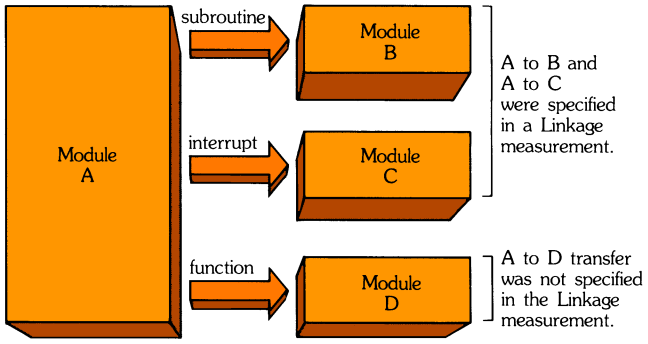


Figure 6. An Intermodule Linkage measurement specifying that transfers from module A to B and A to C are to be analyzed.

Intermodule Linkage measurements show which program flow paths are used most extensively. Modules interacting with subroutines too often may be targets for program restructuring to avoid such heavy interaction. When the interaction is valid however, the measurement points to areas for possible code optimization — in the subroutines for example — to enhance overall system performance.

Data Acquisition Operation

The 64310A Software Performance Analyzer employs a scanning method for data acquisition. During program execution, the analyzer first monitors a randomly selected event from the list of events for that measurement. The time spent monitoring this event, the "event period", may be defined in terms of time (40 μs to 671 s) or in terms of an event occurrence count (4 to over 4 billion). After monitoring the first event, the analyzer then cycles through remaining events, monitoring each for the specified event period. A scan is complete when all events have been monitored, at which time the analyzer randomly selects another starting event, and the next scan is begun.

This scanning algorithm, while statistically accurate, imposes large amounts of analyzer "dead time" with respect to individual events. Depending on the measurement mode, and analyzer setup, a particular event may be monitored only every 4 ms. Some measurements require continuous monitoring, and cannot tolerate large amounts of analyzer dead time. In analyzing stack usage, for example, some portions of the stack are used infrequently. The normal scanning method, by virtue of its imposed dead time, may not capture such sporadic behavior. In cases such as this, the alternate "real-time" acquisition mode may be applied. Allowing nearly continuous monitoring of two program or memory events, data acquisition is interrupted in this mode for only 40 μs every second. This short interruption, accounting for only 0.004% of total acquisition time, allows capture of statistically insignificant, yet, perhaps, extremely important information.

Measurements Involving Multiple Analyzers

Interactive measurements involving two or more 64000 analysis modules are extremely useful in tracking hardware/software, software/software, and hardware/hardware interaction problems. Supervised and controlled on a global level by the 64000 Measurement System, these measurements are conducted via the high-speed Intermodule Bus (IMB). The IMB carries signals between analysis subsystems, providing extensive and advanced measurement capability. Subsystems that may operate interactively include:

64310A Software Performance Analyzer
64620S Logic State/Software Analyzer
64600S Logic Timing/Hardware Analyzer
64300A Emulation Bus Logic Analyzer
64302A Emulation Bus Logic Analyzer
64XXXS Emulation Subsystems

Model 64310A Software Performance Analyzer interacts with other analysis subsystems with two IMB signals. The "master enable" signal coordinates measurement starts with other analyzers and emulators. Furthermore, if this signal is false, the analyzer will not capture measurement information until the signal again becomes true. By this means, interactive measurements with the 64620S Logic State/Software Analyzer, for example, can provide complex windowing capability to software performance measurements.

The "trigger enable" signal is received to accomplish much the same goal. This signal may also be driven by the Software Performance Analyzer on measurement start, or measurement complete conditions, allowing the 64310A to control other analysis subsystems. As long as the trigger enable signal is specified to be received by the Software Performance Analyzer, it must be true before data is acquired. The 64310A analyzer cannot start nor continue a measurement until both trigger enable and master enable signals are true.

An application illustrates the analysis power derived from interactive measurements. In many systems, power failure detection circuits are responsible for quickly switching the system from normal power to battery backup in the event of main power failure. In large systems though, battery backup may be impractical. Power failure detection in these cases provides an early warning that line power has been interrupted. This allows the CPU to immediately take appropriate measures to assure that no data is lost and that the CPU can automatically restart where it left off when power is eventually restored.

The power failure detector may be responsible for asserting an appropriate CPU interrupt. The associated interrupt service routine may then copy vital areas of the volatile memory to disc by evoking disc transfer routines. If these transfer conditions are too slow, the memory to disc transfer will be incomplete when CPU power is interrupted, causing a loss of system integrity. Providing further complication are transfer routines that function well for normal operations, but are deficient under power failure conditions. This necessitates performance analysis under power failure conditions only.

There are several analysis approaches to isolate performance deficiencies resulting from this class of problem. It may be possible to isolate the power failure software by program flow alone. In this case, the two-level measurement enable sequence of the Software Performance Analyzer may be sufficient. In more complex program flow environments, the 15-level sequencer of the 64620S Logic State/Software Analyzer is useful. Program flow satisfying the sequencer specification can then be directed to start the Software Performance Analyzer. Program tracing, and performance analysis features of the Logic State/Software Analyzer expand the measurement with instruction flow information before, during, or after program events. This is of great benefit in the analysis of conditions surrounding performance degradation. If, however, the value of a parameter is required to isolate the power failure software, it is necessary to perform a multiple analysis measurement using one of the 64000 Logic Analyzers. Here, the Software Performance Analyzer is disabled until the Logic Analyzer "sees" the correct parameter value, at which time the Software Performance Analyzer is enabled, allowing it to capture performance data for the power failure operation alone.

If power failure service routines are dual function, it may not be possible to isolate the power failure condition by program or data flow. For example, the information saved by the power failure mechanism may be the same information periodically updated on disc for another purpose. An external timer circuit can periodically interrupt the CPU, and may use the same power failure software. Here, the 64600S Logic Timing/Hardware Analyzer can be used to detect the power failure interrupt by monitoring hardware interrupt signals. The trigger enable signal from the 64600S can then be used to control the Software Performance Analyzer.

Measurements of this type are extremely advantageous in relating software performance to various other software and hardware events. Multiple subsystem measurements extend the capabilities of analysis modules, and when performed with the 64310A Software Performance Analyzer, provide alternative, enhanced views as to the hardware or software conditions responsible for performance degradation. This information is beneficial in carrying out accurate and rapid characterization of such problems and directs engineers to efficient and correct solutions.

Measurement Enable/Disable

In many cases, software performance measurements are meaningful only after specific events have occurred. A "measurement enable" condition may be specified, to define the point at which data acquisition should begin. These enabling conditions may be specified in a number of ways as outlined below:

```
address PORT_1A
address 762DH
address range STACK_1 thru STACK_2
address range 0EFFH thru 0FFFH

module READ_PORT
module range TEST_1A thru TEST_1B

line_number 13 in_file IO_UTIL
line_number range 45 thru 77 in_file PORT
```

Note: A file name identifier is optional on all but line number specifications.

It is possible to enable a measurement based on the number of times an event occurred. For instance, the measurement could be initiated after location PORT_1A is written to 12 times.

Further measurement starting point qualification is available through use of the two-term measurement enable sequence. Suppose system performance is normal as long as module A is never followed by module C. With the use of the two-term measurement enable sequence, the 64310A can remain idle until module C is entered immediately after module A is executed.

Of similar importance is the ability to specify a measurement disable condition, after which the analyzer will no longer collect data. The conditions listed previously may also be used in this specification. For example, a system may exhibit poor performance characteristics until a specific program module is executed. As degraded performance may exist only for a short period of time, the associated performance information may become statistically insignificant when viewed in relationship to overall system activity. Judicious selection of the measurement disable condition in this situation causes the analyzer to stop acquiring data and presents the user with performance information relative to the specific problem being tracked.

Measurement enable and disable conditions, used separately or together, focus measurements on specific areas of concern, and prevent data from being lost in a much larger, overall measurement.

Measurement Windowing

The windowing function is used primarily in situations requiring context recognition. In systems using overlay structures for example, it may be desirable to carry out performance analysis only when a specific overlay is in use. In other applications, general purpose routines may fail to satisfy performance criteria when evoked in a specific sequence. The windowing capability of the 64310A allows engineers to isolate, quickly analyze, and resolve problems in such situations.

Conceptually similar to the measurement enable/disable function, windowing provides performance measurements relative to a particular program area. Specified by enable and disable conditions, this feature also allows isolation of code segments important to a specific measurement.

When the window disable condition is met, the analyzer is only temporarily disabled and resumes data collection as soon as the enable condition is once again encountered. The window may transition from enable to disable and back many times during a measurement, providing, in a sense, a filter for the information acquired by the analyzer. Figure 7 illustrates the windowing concept.

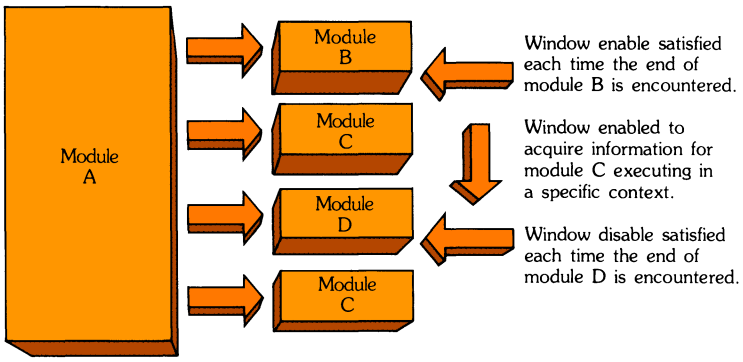


Figure 7. Execution of modules B and D can window data collection concerning module C. Note that module C executed again, but since the context requirements defined by the window specification were not met, no information was collected by the anlayzer.

Characteristics

ACQUISITION

Clock Rate 5MHz max.

Width 24 bits of Address, 8 bits of Status.

MEASUREMENT MODES

Memory Activity	Displays the intensity of activity in user-defined areas of system memory.
Program Activity	Indicates instruction and related activity generated by specific program modules.
Module Duration	Shows the execution time distribution of a selected module, or program segment.
Module Usage	Provides a time distribution measurement from when a module is exited to when it is subsequently reentered.
Intermodule Duration	Indicates a transfer time distribution from the exit point of a procedure to the entry point of another procedure.
Intermodule Linkage	Monitors direct program transfers from a selected module to another module.

TERMINATION CONDITIONS

Time Duration	1 μs to 671 s.
Occurrence Count	1 to $2^{32}-1$.
Scan Count	1 to $2^{32}-1$.
Confidence Level	51 to 99 percent.
Error Tolerance	1 to 99 percent.

MEASUREMENT QUALIFICATION

Measurement Enable (can use a 2-term sequence).
Window.
Measurement Disable.
IMB Trigger Enable (can be received).
IMB Master Enable (always received).

MEASURED EVENTS

Activity and Duration Measurements	12 (max).
Intermodule Linkage Measurement	6 (max).
Real Time Activity Measurements	2 (max).
Maximum Occurrence Count	$2^{32}-1$.

DEFINITIONS

Maximum Number of Event Definitions	99.
Maximum Number of Group Definitions	16.
Allowable Time Event Range	1 μs to 671 s.

INTERMODULE BUS (IMB) FUNCTIONS

Trigger Enable	can be received or driven on measurement start or measurement complete.
Master Enable	received always.

SUPPORT

Languages	Pascal, C, and Assembly.
Processors	All 8-bit and 16-bit processors supported by HP 64000 Emulation Subsystems.

ENVIRONMENTAL

Temperature	operating	0° to +40° C (+32° to +104° F).
	nonoperating	−40° to +75° C (−40° to +167° F).
	operating survival	−20° to +50° C (−4° to +122° F).
Altitude	operating	up to 4600 m (15 000 ft).
	nonoperating	up to 15 300 m (50 000 ft).

Relative Humidity 5% to 80%, noncondensing.

POWER REQUIREMENTS

+5.0V	5.80 A (max).
−5.2V	0.12 A (max).

Ordering Information

COMPONENTS

Model 64310A Software Performance Analyzer Card — $3400

Model 64310AF Software Performance Analyzer
Operating Software on Flexible Disc — $100

Model 64310AT Software Performance Analyzer
Operating Software on Tape Cartridge — $100

Model 64960A Emulation Bus cable, 2-position — $70
(2 cables required)
Option 001: replaces 2-position cable with 3-position cable — no charge
Option 002: replaces 2-position cable with 4-position cable — no charge
Option 003: replaces 2-position cable with 5-position cable — no charge

Model 64964A Intermodule Bus cable, 2-position — $80
(One cable is required only for performing interactive measurements with other 64000 analysis subsystems.)
Option 001: replaces 2-position cable with 4-position cable — no charge
Option 002: replaces 2-position cable with 6-position cable — no charge
Option 003: replaces 2-position cable with 8-position cable — no charge

Note: The 64310A Analyzer must be used with a 64000 Logic Development System and a 64000 Emulation Subsystem.

Your HP Field Representative can help you determine the best configuration to meet your needs.

U.S.A. list prices only.

5953-9210 (D) Data subject to change. PRINTED IN U.S.A.

For more information, call your local HP Sales Office or nearest Regional Office: Eastern (301) 258-2000; Midwestern (312) 255-9800; Southern (404) 955-1500; Western (213) 877-1282; Canadian (416) 678-9430. Ask the operator for Instrument Sales. Or, Write: Hewlett-Packard, 1501 Page Mill Road, Palo Alto, CA 94304. In Europe: Hewlett-Packard S.A., 7, rue du Bois-du-Lan, P.O. Box CH-1217 Meyrin 2, Geneva, Switzerland. In Japan: Yokogawa-Hewlett-Packard Ltd., 29-21, Takaido-Higashi 3-chome, Suginami-ku, Tokyo, 168.