The memory bus monitor—A new device for developing real-time systems

by RICHARD E. FRYER

Naval Weapons Center
China Lake, California

INTRODUCTION

The memory bus monitor was designed to assist in program development on dedicated computers. A dedicated computer is defined here to be one that is used for only one major application and is available to the programmer in blocks of time as needed to complete the development and checkout of his problem. Because of high cost, few large scale computers can be operated in this manner. However, many medium scale and virtually all minicomputers (including airborne and other process control computers) are operated this way, at least when assembly language code is being written. Most systems classified as real-time also fall in this category.

The cost of such systems hardware is decreasing at such a rate that their number is dramatically increasing. Progress in reducing software costs is virtually nonexistent, even though it is generally regarded as being the limiting factor in the development of these systems.\(^1\) These reasons give economic justification for the development of tools to assist in the programming task.

Most instrumentation and monitoring techniques have been developed since 1965.\(^2\) A survey of bibliographies on systems measurements\(^3\) also reveals that software evaluation and measurement techniques are being explored at a much more intense level than hardware approaches—probably because computer hardware designers and users continue to be distinct varieties with relatively little cross-pollination. The predominance of measurement techniques has been developed to assist in improving time-sharing systems or to aid comparison of large systems. Very little emphasis has been placed on the needs of programmers, and even less on applications for medium and small scale systems.

Some instrumentation aids have been in use since the early 50's. The breakpoint register has been implemented on many systems at least as old as the IBM 650\(^6\) and as large as the UNIVAC 1108. This register allows the programmer to execute a program at full speed until the instruction register (or perhaps a general register) agrees with the breakpoint value; then halt. The user can then step through the program one instruction at a time (a practice that is probably discouraged at 1108 installations). In spite of the simplicity of this device, it can be a great help to the programmer. It has been reinvented a number of times in the last few years by vendors of small machines.

REAL-TIME SYSTEMS DEVELOPMENT

Development of software systems for real-time applications typically proceeds through definition and specification, design, implementation and checkout stages. Due to difficulties in predicting subprogram size and timing for various functions, it isn’t usually known if overall size and timing goals will be met until the complete system is benchmarked. When original design is incomplete or system specifications are revised, the goals may be exceeded by a substantial margin. Then program cycle time or memory requirements (sometimes both) must be reduced without adversely affecting the other parameter. And this optimization must take place while program debugging continues.

Machine software simulators and assembly language debugging programs (such as Digital Equipment Company’s ODT) are among the tools available at this stage of system design. Many aspects of system design can be verified with these tools—logical flow, variable scaling and correctness of computation, for example—but they are rather useless for measuring cycle time, locating faults that occur during peak I/O bus loading, detecting race conditions in a multiple asynchronous interrupt environment and similar problems. A program trace recorded on tape can yield data on the actual program path taken (at least over short time periods), but is expensive to reduce, is not interactive, and can only approximate real execution times.

The memory bus monitor is intended to supplement these and other development tools. It is useful for initial testing, checkout, optimization, and system validation. Timing and instruction mix data obtained with the system are also expected to be useful in future system designs.

MEMORY BUS INFORMATION

The information carried on the memory bus of a typical system has a simple format. Address lines specify the
memory location to be accessed. Data lines carry the data read or to be written, and control information includes a Read/Write (R/W) line and sometimes a split cycle line for read-modify-write operations. This information is adequate for the operation of the memory bus monitor. An Instruction/Data (I/D) status line is also sometimes available. It specifies how the current bus data word will be interpreted; its use is described in the operation section. Other lines sometimes available include the Direct Memory Access control line. While this and other control lines can reveal still more information for the user, they are not discussed further in this article.

The stream of addresses and data traveling the memory bus, though simple in form, contains much information about the execution of a program; information not ordinarily available to a programmer. The rich content of bus information can be extracted using time and address correlation of bus traffic. Several specific operations on bus data and the program parameters that they measure are listed below.

1. The behavior of any selected variable is easy to follow by identifying when its address is upon the bus. Actual instead of intended behavior is observable.
2. The address(es) holding any specified data word that is accessed may also be obtained by monitoring data values and treating the address as an unknown.
3. One of the most valuable artifacts available from the bus is the history of addresses just prior to the time when a specific location is written. This instruction stream tells the programmer what code was executing when an instruction or program constant is accidentally destroyed.
4. Another artifact that is often referred to but rarely measured is the instruction mix for a particular section of code. If the code is short enough, a "static" mix may be determined by hand or via a program editor. However, large sections of code become unmanageable and generally unknown branching ratios at decision points make "dynamic" (actual) mixes difficult to predict. Dynamic mixes are easily obtained with the bus monitor.
5. Branching ratios are also readily extracted from the bus.
6. Accurate and detailed timing information provides the most effective way to optimize program cycle time. Measuring the actual execution time of a section of code is easily accomplished using the bus monitor, in contrast with conventional techniques that, for example, require changing existing instructions (such as NOPs) to uncommitted output instructions to generate timing sentinels.

HARDWARE DESIGN

A basic system

The block diagram in Figure 1 shows the essential elements of a minimum bus monitor. It can be used to display the contents of any memory location as it is read or written. The user enters the address of the location to be monitored and selects Read, Write, or both. The data values read or written are then displayed in the readout. A static/dynamic switch (not shown) allows the operator to stop the display if the update rate is too fast. A read status indicator can be set to blink or latch each time the selected location is read. The trigger to the readout register is also connected to a BNC connector on the front panel so that an oscilloscope or digital counter may be used for frequency, period, or counting operations. Interrupt and I/O rates can be monitored this way, for example. This signal may be ORed with the halt signal in some computers to effect the breakpoint function.

An intermediate system

A diagram of the first system constructed appears in Figure 2. It was designed for a Varian 622A computer in 1969, and has one primary feature missing from the previous system. The address stack is added consisting of an 8 word shift register. Each new address that appears on the bus is pushed onto this stack.

The control logic that is used to latch the display register can also be routed to freeze the stack. When the address compare signal halts the stack, the user can manually revolve the stack past the display to inspect memory addresses (and thus instruction activity) prior to

From the collection of the Computer History Museum (www.computerhistory.org)
the selected read or write operation. This feature was commonly used to locate the section of code responsible for destroying a known location.

**The current design of the memory bus monitor**

A simplified block diagram of the Memory Bus Monitor appears in Figure 3. In addition to the functions that existed in the earlier design, the current design has the following features.

1. The shift register stack is replaced with a 16 word Content Addressable Memory (CAM) that is used as a stack, a learning memory, or as a 16 word comparator depending on mode.

2. A 16 word data stack is also added with 2 extra bits for the R/W status line and the I/D line when available. The data register (number 4 in Figure 3) and comparator can generate a trigger when a data word equal to the register contents appears on the bus. Bits set in the mask register (number 5) inhibit comparison of corresponding data bits. That is, they become "don't care" bits.

3. The address detector is expanded to 4 registers (numbers 0-3). Register 1 is used with register 0 to bracket sections of code and to straddle branches. An output is also generated that combines registers 0 and 1 to detect any address falling between (0) and (1), where (n) denotes the contents of register n.

4. The next two address registers, numbers 2 and 3, are enabling registers. They are used for arming conditions. Comparator pulses from these two are also used to set and reset an RS flip-flop. The output of this flip-flop can act as a gate to pass other events.

5. The base address register, number 6, and the associated adder are used to add a constant to keyboard addresses and to subtract that constant from address readouts. This feature is nearly essential for effective use with relocated code.

6. A watchdog timer function is implemented with register 7 and an up-counter. The internal 20MHz monitor clock is counted down to supply a 1 microsecond and a 1 millisecond trigger rate for the timer. The timer may be operated in either "gate mode" that compares the time interval of the RS gate with (7) or in a "cycle mode" that compares the time duration between pulses. In either case, failures (measured intervals greater than the register value) generate a trigger.

7. Finally, though not shown in Figure 3, an octal/hexadecimal switch controls the keyboard and readout mode to match the form used by the computer under test.

The current monitor is considerably more sophisticated than the earlier design, but is of no more than moderate complexity (simpler than a disc controller, for example). The unit is constructed using wire-wrap and TTL technology. Approximately 150 SSI and MSI packages are required for the design, excluding the interface to the memory bus. The bus interface is on a separate board that can be easily replaced for transportability between various 16 bit computers. For machines with a larger address or data word only the corresponding registers and display need to be enlarged. The device is mounted in a 19"×12"×3 1/2" case—one selected to fit along with a general purpose counter into a small suitcase.

The operators panel of the monitor has the following features. A hexadecimal keyboard, 2 digit thumbwheel switch, and an Address-Data display allow the user to load and verify the contents of the 8 registers. Also, the CAM storage appears at thumbwheel addresses 10, to 27; the data RAM at 30, to 47. Each comparator output and the two derived signals are routed to connectors on the panel. Toggle switches adjacent to the connectors allow the user to select a combination of these signals to form a "master" trigger that is routed to the display and stack. The selected signals may be NANDed or ORed to form a master trigger. This composite trigger is also routed to both a pulse and latching indicator. Control is also provided for the watchdog timer modes and for the condition to be used to halt the operation of the monitor. A 7 position selector switch gives access to the various modes of operation. Status lights reflect the condition of the timer and halt logic.

**OPERATION MODES AND THEIR USES**

The Register R/W mode is used to load or modify registers at the beginning of a run, and to review CAM and RAM contents after a run. Pushbuttons on the panel set the R/W and I/D bits in registers 0 through 4, and these bits are compared with the bus status lines along with the address bits.
The **Bus Monitor** mode routes the contents of the memory bus address and data registers to the display registers. This mode is useful when operating the computer in single step mode.

The **Load Stack** mode uses the trigger source(s) selected by the user to transfer current address and data memory bus address and data registers to the stacks. These registers are also transferred to the display. Typical monitoring operations in this mode are described below.

1. The activity of a specific location is extracted from the bus by putting its address in one of the 4 address registers and enabling the comparator output to generate the master trigger with the corresponding toggle switch. More than one register can be used if desired (4 is the maximum). The trigger generated pushes the bus address and data registers onto their respective stacks and also transfers this data to the display. The R/W enabling bits for the 4 registers need not be the same. The stack continues to load until the halt flip-flop is set by the stack full trigger or any other trigger selected by the user.

2. Addresses containing a specific data word may be located as this word is read or written by loading and enabling register 4. These addresses are displayed and pushed onto the stack. The mask register is used to limit the comparison to, for example, the op code field.

3. Using the data and mask registers in conjunction with an address register allows a limited study of variable scaling.

The **Halt Stack** mode allows all address and data information to push the stacks, and halts this operation when the master trigger occurs. Recent history of bus traffic is the most important artifact in this mode.

1. The instruction stream that references a given word can be recovered—particularly useful if a data word is destroyed as mentioned for the earlier design.

2. Registers 2 and 3 are used to set and reset the RS flip-flop. The output then enables any other triggers, allowing the measurement mentioned above to take place only when an event arms the device, or while within (or outside) a defined section of code. The $(0) \leq \text{ADDRESS} \leq (1)$ trigger can be similarly used for routines or data arrays instead of single words.

3. When a certain number of tasks must be completed in a specific time, the watchdog timer is used to halt the stack, allowing the task currently executing to be identified.

**System Timing** and **Counting** operations are done in both of the above two modes. The connectors on the front panel can be connected to a general purpose counter (a Hewlett Packard 5325 has been used). Common timing operations are described next.

1. Program loop cycle time is determined by monitoring the frequency or period of the trigger from an address comparator.

2. Subroutine (or any section of code) timing is done by measuring the period of the RS flip-flop with registers 2 and 3 set to the limits of the routine.

3. When a section of code calls another routine, it is itself interrupted, or has multiple exit points the time measured using the RS flip-flop as described above is in error. The correct value can be determined by using the $(0) \leq \text{ADDRESS} \leq (1)$ signal, since it falls to zero when the address falls outside the limits.

4. The effective cycle time is determined from the memory cycle trigger that is routed from the bus interface to the front panel. This measurement can aid the user in determining if faster memory would speed up his overall system.

Several specific counting operations that are useful are now described.

1. The branching ratios at a decision point are readily determined by setting one register to the branch instruction and another to an instruction in one of the branches. The counter is operated in the ratio mode to give a direct ratio reading. All branches can be quickly checked to insure that the sum of all ratios is 1.

2. The number of occurrences of an instruction with a given op code may be counted by using registers 5 and 6 to isolate the op code part of a word. A wait loop is often a part of real-time programs, and if the idle time is significant, the instruction mix becomes contaminated. The address range can then be limited to exclude the wait loop in a manner already described. This same technique is used to determine a mix for a specific task.

The monitor trigger that is derived for timing is also often useful for synchronizing an oscilloscope during hardware checkout. Events within the computer and on the I/O bus that result from instruction execution can be ‘anticipated’ by the monitor.

The **Address Sieve** technique operates the CAM as a learning memory. It is used to locate all instructions that refer to a specific location. The user derives a trigger in the usual way, and it causes the previous address (saved in the holding register shown in Figure 3) to be applied to the CAM. If that address is not already in the memory, it is added.

The **Selective Dump** mode uses the CAM as a 16 word comparator. The user loads the CAM with up to 16 locations that he wishes to capture; then sets up a halt trigger. When the CAM detects a match, the address of the match in the CAM (that is, 0 to 15) is supplied to the RAM and the data register is stored in the corresponding RAM location.
The **Watchdog Timer** runs in all modes and generates triggers that may be combined for the master or halt triggers. When the timer mode is selected, however, measured time intervals are pushed onto the stack. The RAM and CAM are both loaded to provide 32 values. The user may store all counter values or only those that exceed the value in register 7. The timer values are also routed to the display.

**DIRECTIONS FOR DEVELOPMENT**

When a user wants to know how long a given section of code takes to execute, he would really like to know the extremes and something about the distribution. The current monitor is unsuitable for collecting all measurements for statistical analysis since it is a manually operated device. Anticipating this problem, some effort was made to ease the transition to computer control of the monitor. Also, the counter mentioned allows for remote programming and data collection. There are many modes and instances, however, when the increased cost and complexity of a general purpose computer would not be justified. Developments in microcomputers make the inclusion of such a device in an advanced monitor an attractive option to consider, especially if compilers are written for them.

We envision that both manual and automatic monitors will find their way into the programmers toolkit.

**SUMMARY**

This paper has briefly reviewed the software development problem for real-time and process control applications, and has shown how memory bus information may provide data to aid the development of these systems. A device is described that can extract many different artifacts from the memory bus with relative ease. The device is inexpensive, small, and can be switched from machine to machine with little effort.

**REFERENCES**


From the collection of the Computer History Museum (www.computerhistory.org)