Minicomputer Controlled System Simulation Considering Input/Output Interrupting Timing

Yukoh KOBAYASHI*

Abstract

A general principle that any devices and computers embedded in a system should be considered in equal rights is established. This principle suggests a good simulation technique of a computer controlled system, offering simplicity and flexibility. For demonstration, simulation of minicomputer system is implemented for a purpose of developing and debugging real programs, especially those sensitive to input-output interrupting timing. The simulation technique described can be applied for large computer system to investigate the system efficiency.

Key Words and Phrases: simulation, principle, debugging tool, computer controlled system, interface, interruption.

CR Categories: 4.42, 4.49, 6.21, 6.35, 6.39, 8.1.

1. Introduction

Recently, minicomputers have been so popularized in wide and various uses mainly because of their easy manipulations and low price. One of the remarkable characteristics and advantages of minicomputers is that they can control many and various kind of input-output devices such as special hand-made apparatuses for scientific experiments, data transmission lines, etc. as well as usual teletypewriters or paper tape readers, whereas big machines are not suitable for such special uses because of severe hardware and software restrictions and complexity. Moreover, minicomputers may stand exclusive uses, but big computers may not from economic point.

In order to utilize fully these advantages, we can make up a sophisticated system which is constructed out of devices and minicomputers, as processors, being connected organically, where the whole system is controlled by the minicomputers. The system, for example, may be scientific measuring one with unique equipments as input or output devices which are combined with each other, data transmission system with modems, process control system, and so on. This idea seems to be one of the proper applications of minicomputers, and will become more important in near future.

In those cases it is necessary to develop software or programs which run on minicomputers to control the whole complex system. However, it is very uncommon that the system involves such I/O-devices that may be powerful for program debugging, like card readers, line printers or disk storage. In addition the amount of memory is so limited that it is practically impossible to keep both debugged program and debugging packages reside in the memory concurrently. Therefore the task of developing the soft-

* Department of Electrical Engineering

ware is terribly troublesome and takes long time when the actual minicomputer is employed. This is one reason why simulation on middle or large machine with more powerful I/O-devices, debugging packages, and large memory is remarkably helpful for program debugging.

Second advantage derived from simulation lies in that we can easily affix to devices optional functions which are not fixed on real devices but may be very convenient for testing or debugging, e.g., display organization of buffered data, data storing organization, alarming organization and so on.

Third advantage, which is more essential and important, is accessibility and repeatability. In a real device or machine it will be very difficult or almost impossible to see or modify data in some registers and buffers during execution without disturbing any situation, whereas this is so easy on simulation. Furthermore, to repeat exactly the same system operation or mal-function any times we want is impossible on real system because machines and devices have more or less timing fluctuation. Thus simulation dealing with clock tick can provide effective debugging means especially concerning complex I/O-interrupting timing problems.

The great advantage of simulation, which is briefly described above, is explained in more detail by R. M. Supnic [3]. He also reports some ideas and the implementation of our concerns, but we have made what he conceived clear and generalized it to give us simplicity of theory and flexibility of simulator. For example, he especially concerns program debugging, but we intend hardware debugging of I/O-devices as well.

There are few papers on our subjects. J. Kontons, et al. [2] utilize two computers which interrupt each other: one is a control computer operating under almost actual conditions, the other a central computer that simulates a plant. In this case flexibility is hardly secured because of the restricted environment of these computers.

In this paper a principle regarding simulation of computer controlled system is established, which suggests successful simulation technique in the sense that the simulator can have much flexibility in order to be used as a convenient debugging tool, as well as system analyzer. And in addition a new implementation of system simulation is shown according to the principle. As is evident from above description, our simulation concerns not the every structual details of computers and I/O-devices, but the interactions among those system components.

The implementation has been made for a system controlled by a minicomputer, HITAC–10, Hitachi Worker Inc., to make sure our philosophy. The simulator runs on FACOM 270–20, small machine. It should be noticed that our technique could be applied for much larger computer systems in order to debug, test or examine the system, programs or system performance.

2. Basic Concept of Computer Controlled System

We consider a system, where one or more computers and devices are organically connected to proceed automatically their operations according to the proceduers given by program on the computers. This situation is illustrated in Fig. 1 schematically.

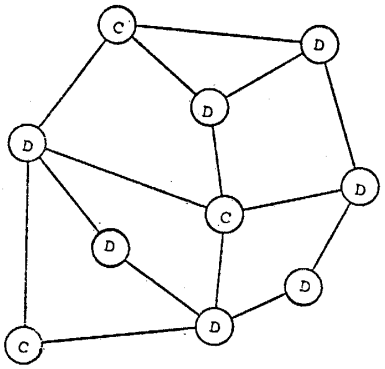

Fig. 1 A system composed of computers (processors) and devices.
C denotes a computer and D a device.

In this system any devices connected to a computer can be considered as either an input or an output, or both if we think much of the part of the computer. However, now that the computer is embedded in the system in this way, we no more need to emphasize its role particularly in the sense that the computer gets data from input and puts out some data to output just in the same manner as other usual devices. Therefore we should not distinguish computers, or processors, and other devices, rather we had better consider that they are identical constituents of the system, having input and output terminals to be called "black boxes" or automata. All the components together with computers and devices interact on each other.

We shall call each constituent of the system as an element in order to identify a computer with a device. Consequently, we can establish a principle that all elements of the system should be regarded as automata having same equality; we should not distinguish computers in the system from other devices like conventional I/O-devices essentially. On this thought the system shown in Fig. 1 can be rewritten as Fig. 2. Each element or automaton denoted by A may be a computer, I/O-device, an equipment, a plant or other system which is constituted by another some automata likewise.

This principle is very important and useful since it gives us obvious thinking way on different type of devices and makes the logical design of the simulation significantly

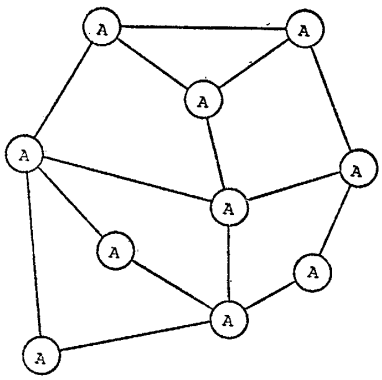

Fig. 2 A system composed of many automata.

simple. The supervising part of the system simulator which supervises the entire simulation should treat each element without any distinction whatever kind of automaton it may be. Hence the flow diagram of the supervisor part of the simulator is simplified to give same flow for all elements.

3. Structure of Simulator

We briefly describe below the implementation of a minicomputer system simulation following our principle shown in previous section. Though one or more computers can be contained in a system, present simulation concerns only one computer and some I/O-devices for simplicity. The simulator is organically parted into three main blocks;

(1) SUPERVISOR block,
(2) CLOCK block, and
(3) ELEMENT block.

These three blocks are mutually related as illustrated in Fig. 3. The SUPERVISOR block supervises the whole system, i.e. the other two blocks; to motivate or stop the clock block to control the execution of simulation, to see or modify the data in every system element of ELEMENT block, and to gather momentary data of the simulating system during executing simulation so as to be printed out latter in appropriate form. The CLOCK block is responsible for ticking the time being simulated, having a counter which may be called clock. Fig. 3 implies that we had better think the CLOCK block

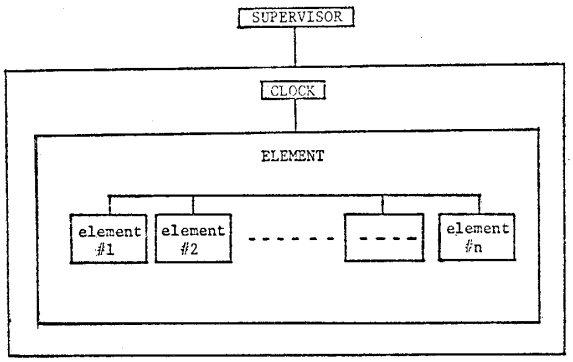

Fig. 3 Logical structure of a computer controlled system simulator.

should not stand outside of SUPERVISOR, but rather should be involved in it. The last ELEMENT block means the group of all system elements, each of which simulates an actual device or computer. For every system element CLOCK block commonly serves whenever requested since time is universal over the system.

In Fig. 4 the main flow of the simulator is shown, which is just the one of SUPERVISOR block. In the following we will give more detailed explanations for each block.

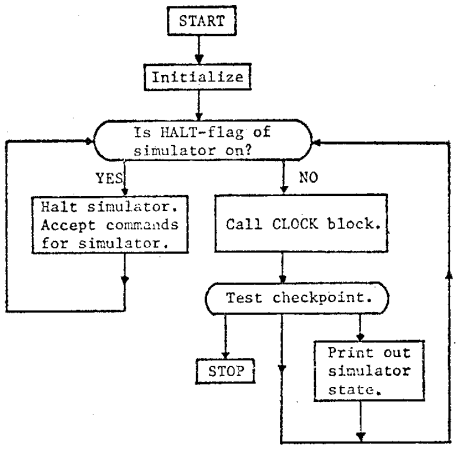

Fig. 4 Main flow chart of a computer controlled system simulator. (SUPERVISOR block).

3.1 SUPERVISOR Block

As HALT-flag of the simulator, which is a sign to halt simulation in temporary, is initialized in on-state, the simulator goes to halt-state immediately after first starting and accepts commands to set every element in initial condition. When start-command is given to the SUPERVISOR, then HALT-flag is set to off-state and CLOCK block is invoked in order to proceed simulating time. Every time CLOCK block is called out, the counter is put forward, and then an element is activated to operate only one event as long as any event is registered in tables managed by CLOCK. That is, only an event can be executed during one cycle of SUPERVISOR-CLOCK-SUPERVISOR loop shown in Fig. 4.

The HALT-flag is set to on or off by SUPERVISOR either sensing a special key or switch which is manipulated manually by the user at any time, or checking trace conditions specified in advance. Many of the trace will appear in section 5.

In this way command for the simulator may be given not only in a batch processing way but also in interactive through a typewriter. Obviously, the latter allows more flexible debugging means rather than the former especially in the case that it is too difficult to guess bugs beforehand because of complexity of the system being simulated.

The command received by SUPERVISOR involves the one for SUPERVISOR itself, which contains manipulation of the simulator, for CLOCK, and for ELEMENT in order to examine and modify data or state of some elements. Further details of the command oriented to debug will be shown in following section 5.

3.2 CLOCK Block

As mentioned in section 3, CLOCK block plays the function of ticking time and alarming in the simulator. It is just like a usual alarm clock; it accepts a request of any system element demanding to be alarmed, and when the demanded time comes CLOCK calls out the element with a sign to do its operation. The correspondence between CLOCK and an element in ELEMENT block may be accompanied with some

informations such as an element number, the time length necessary for its operation, an operation-specification number, and so on. On practical implementation those informations should be managed with sufficient efficiency in order to speed up the execution of simulation since CLOCK is passed through every time one event is carried out as described in section 3.1.

To determine the sequence in which an alarm is sent out, CLOCK has a time table and a counter which indicates the current time being simulated. The time table can be managed in order in the form of either list or linear structure. In many cases where the number of registration arranged on the table is not so large almost always, or there is smaller number of events with short time length on the table than that with long time length, the latter structure will be better than the former from the viewpoint of running speed. With this choice Fig. 5 explains how time, which is held by counter (C), is put forward.

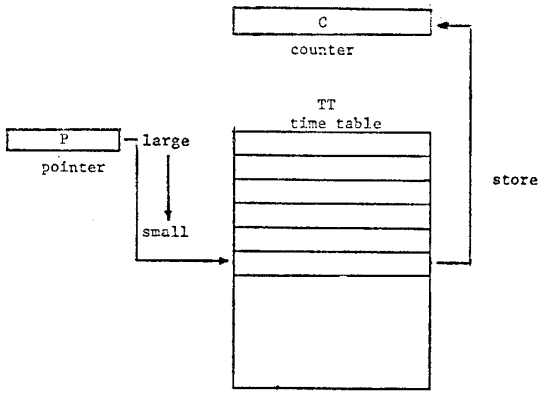

Fig. 5 Schematic representation of the management of time by CLOCK block. The content of the counter C indicates present time being simulated.

The time table (TT) is arranged in large to small order when request is accepted. The pointer (P) points the head of the table and indicates the time when next event is to be executed. This method of keeping the table stands on the reasonable consideration that of all operations of any system element program instructions executed by a computer is nearly shortest since other many I/O-operations take long time and moreover, the number in the table is small. Then it seems to be ordinary phenomenon that only a small part of top of the table is processed and replaced with a new event for one cycle of simulation, while large part of the bottom remains unchanged for long time. For these reason and another reason that usual computer system under which the simulator can run takes much longer execution time of list structure owing to somewhat complicated programming procedure, linear table will be very advantageous to us. It should be noticed that as is clear from above explanation the sequence of events to be processed is determined not just before an event is processed, but when CLOCK receives the request except that later new events may be inserted into the sequence.

The counter (C) is advanced simply by being stored the content of the time table pointed by the pointer (P). Hence time is skipped till next event. When we want to

count much longer time than a counter can hold, additional counters can be employed just as one for minute, for hour, for day, etc..

It will be very interesting to mention that such state of the system that nothing occurs in real system not stopping, which means there is no communications among system elements, corresponds to the state in the simulator that CLOCK has no events to call out on the time table, but the simulator is not stopping, only to make worthless loop of SUPERVISOR-CLOCK-SUPERVISOR.

3.3 ELEMENT Block

ELEMENT block consists of all system elements, each of which may be divided into several modules based on task; a main body simulating principal operation of a real device, a module to accept user commands, etc.. The element is called out either by CLOCK block during normal execution of simulation, not in halt-state, or by SUPERVISOR block to allow manual manipulation such as pushing buttons, which is made by commands through typewriter or switches, and to offer debugging aid. In such a way as is described in section 3.2 communications among system elements can be done only through CLOCK block, not directly each other.

For an example of element simulation we will show the frame of a module simulating a computer in next section since it has much complicated organization and I/O-interface rather than other kind of elements.

4. An Element Simulating a Computer

The organization of a minicomputer is rather simple, and may be simulated as its flow chart is shown in Fig. 6. However, this chart is to be slightly changed depending on the machine organization [1].

At the first step of this module in Fig. 6 previous status is checked if halt- or running-state, and when the former case the module is switched to running state to begin fetching next instruction, otherwise, in running state, the interrupt line is sensed. At next step, halt-switch, which simulates a stop button fixed on the real computer panel being simulated, determines whether next instruction is to be fetched or not. After decoding and executing the instruction, the module asks CLOCK to register the operation time length.

When an I/O-control instruction is decoded, informations are transferred between the computer and other system elements connected to it via I/O-interface. Thus each system element must have its own interface routine corresponding to a real hardware interface circuit. As we greatly concern ourselves about I/O-interrupting timing problems, much attention has been paid to simulate those interfaces so as to perform real hardware operations as exactly as possible. Of course it is not sagacious to simulate its details beyond needs for it causes additional simulation time too much.

In Fig. 7 I/O-interface module which belongs to the minicomputer is exhibited. There I/O-devices are connected in series, but informations are in parallel transmitted just like real situation, which provides users with easiness of attachment or detachment

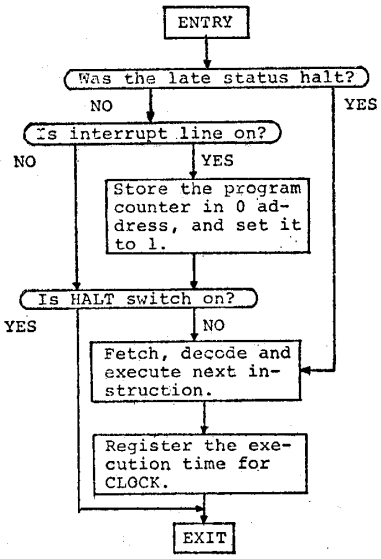

Fig. 6 Main flow chart of a minicomputer.

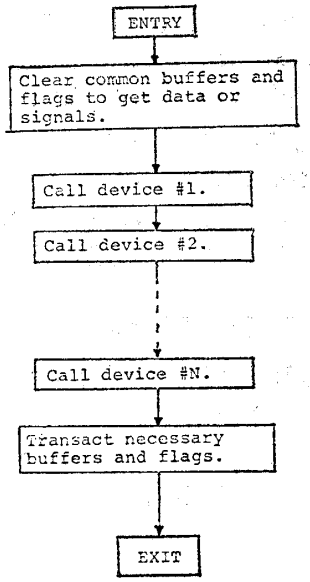

Fig. 7 I/O interface modulefor a minicomputer.

of devices without any change of other circumstances. Though all devices are transmitted to same informations, only a device whose device number coincides with the one accompanied can receive the data. Of course these depend on actual hardware.

Using our technique, it is easily possible to treat various kinds of control pulses which are exchanged between a computer and I/O-devices associated with data transmission. This enables us to test I/O-interface circuits by means of simulation.

Other elements than a computer can be simulated almost in similar way except hardware operations.

5. Commands Used for Debugging

Commands, all of which are given to the simulator through typewriter or switches manually, are conveniently divided into two categories; the one oriented to a system element, the other to SUPERVISOR.

The first category involves mainly such commands as simulate real button or switch manipulations, which contain those for unreal, additional function helpful for debugging. These are as follows:

(1) to start or stop the operation.
(2) to reset the state.
(3) to define the operation time length.
(4) to display the internal status including buffered data.

The second concerns two kinds of command; the one is for simulation control,

and the other for tracer which is a module belonging to SUPERVISOR block and traces all simulation stages to provide debug data. Simulation control command is:

(1) to start, halt temporarily and finish the execution of simulation.
(2) to attach and detach an element to/from the system.
(3) to reset clock counter.
(4) to print out the internal state of the system.

On the other hand the tracer control command concerns both management of trace table and specification the condition when trace should be begun or finished during simulator's normal running, and has relation with CLOCK and ELEMENT block. Conditions may be specified simultaneously by clock time, that is, system's time or some special status of system elements. For instance, the moment when some element gets or put out data, or when some element makes interruption may be used for trace condition. For a computer of system element the address held by its program counter or some kind of instructions, such as branch, store, load, etc., may be used to specify trace condition. The tracer checks all of the given conditions during normal running of the simulator, and when at least one of those is satisfied at some simulation point either of the two actions will be taken; to print out appropriate data of the system, e.g., the current time of the system, the contents of buffers and registers of elements, status flags of elements, and so on, or to halt the simulator to enable user to manipulate it in interactive way. Following examples are of these commands:

(1) to reset tracer table.
(2) to display the content of the tracer table.
(3) to update the tracer table.
(4) to specify time or time period.
(5) to specify an address or range of address for a particular system element.
(6) to specify both an address and the number of times to pass through it for a particular system element.
(7) to specify special instructions, as described above, for a particular system element.
(8) to specify both a special instruction and the range of address where the instruction exists or accesses for a partiular system element.

These checkpoint commands are managed in a table, named trace table, by tracer-module of SUPERVISOR.

The data printed out by the tracer-module of SUPERVISOR when trace condition is satisfied may involve the current time of the system, the contents of buffers and registers of elements, status flags of elements, and so on. And it is possible without difficulty that each user modifies this print-out module so as to print out what he wants according to his own need. For printing the list, SUPERVISOR does not directly get data from the system elements, but only offers a storage buffer to have each element put

in its own data by itself in order to secure modulality of each block.

6. Conclusion

A general principle that any devices and computers embedded in a system should be treated in the same light in terms of system elements regarding computer controlled system simulation has been proposed in this paper. Besides from this point of view the implementation of a minicomputer system simulation has been described in order to demonstrate and test the validity of the principle, where debugging of control program is intended. This has confirmed that our principle provides simplicity and flexibility of simulator structure, and moreover this simulation technique is very effective for debugging such programs as may concern with even the delicate interrupting timing.

Here we summarize several advantages of our simulator from the viewpoint of implementation, as following;

(1) the simulator-program is divided into many modules in order to allow users easy modification in compliance with their own requests.
(2) the simulator-program is written mainly by FORTRAN, which offers high portability.
(3) I/O-interrupting timing for CPU is successfully simulated by establishing CLOCK block.

It will be instructive to mention that although our implementation has been confined to a system controlled by only one computer, our principle can also apply for a system with two or more computers.

There is still a problem in the implementation. The sequence in which events are carried out is determined according to the magnitude of the time on the time table managed by CLOCK block. Thus when there are different events with same time, the result of simulation may be differ from that of real system operation, but not always. In usual case the processing sequence of events with same time will not cause serious problem, but when such order may significantly sensitive to result special consideration should be paid to arrangement of the time table.

Acknowledgment. The author wishes to express his appreciation to Professor H. Takahashi, Associate Professor T. L. Kunii, Mr. N. Hasebe, Mr. M. Imori, and Mr. N. Ohbo for their useful discussions.

References:

1) Chu, Y. Introduction to computer organization. Prentice-Hall, Inc. New Jersey, 1970
2) Kontons, J., and Papakonstantinou, G. A simulation method for computer control systems. IEEE Transactions on Computers C–20, 1 (Jan. 1971), 98–100
3) Supnic, R. M. Debugging under simulation. PDP–10 Applications in Science, Vol. No. 2 (1973), 103–126, Digital Equipment Corporation