Lect 6

Example 1
(Illustrating the measurement procedure)

Question: Consider a system with one CPU and n channels. Show the setup for measuring the fraction of time the CPU and k channels (for a given k □ 1...n) are busy simultaneously.-vedio

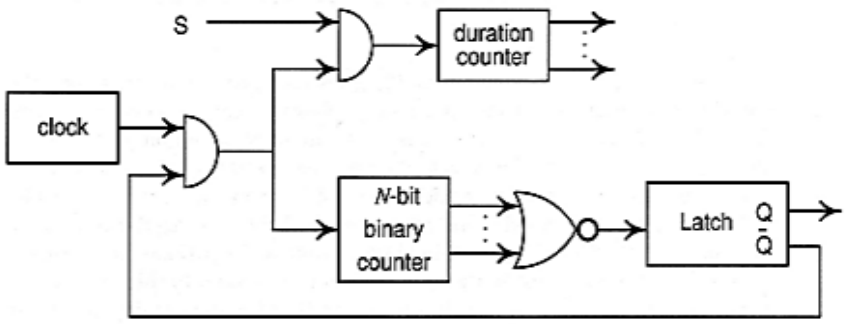


Fig 1: Setup for CPU-Channel overlap measurement

Solution:

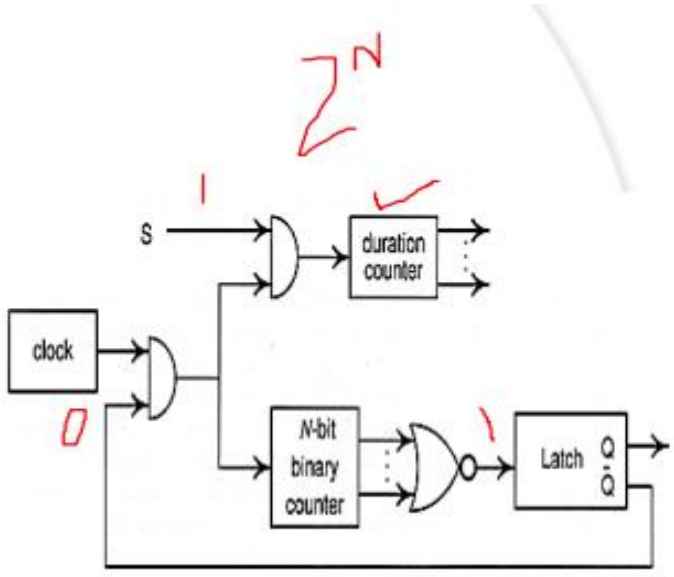


Fig 1: Setup for CPU-Channel Overlap Measurement

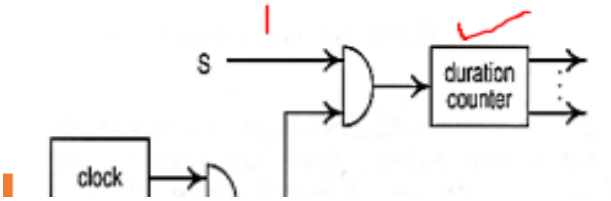
• suppose that The required input signals are CPU-busy bit

(CpB) and the channel-busy bits (ChB_i for channel i).
ChB_1 for channel 1 , ChB_2 for channel 2 .. so on

• Assume these are available from the backplane.

• Let $f_{k/n}$ denotes the boolean function that generates the output 1 if exactly k out of its n boolean inputs at logic 1.
Infact from the given problem statement we are actually interested in the same mechanism.
Lets suppose there are total 5 channels & 2 of them are busy at the moment so the value of n is going to be 5 & the value of k is going to be 2. So $f_{2/5}$ will be 1 at this point in time.
• The boolean function S computed as follows:
• $S=CpB \wedge f_{k/n} (ChB_1,... ,ChB_n)$ ---------- (1)

• Refer to Fig 1, for each clock pulse, if S is high (if value of S is 1) and the measurement interval is not over, the duration counter will be incremented by 1.



• The N-bit binary counter along with the NOR gate acts as an interrupt generator.
How N-bit binary counter & NOR gate are acting as an interrupt generator.
just try to recall the functionality of a 3 bit binary counter, let say its a down counter so a 3 bit binary counter will actually counts from 7 till 0 (like 7 6 5 & so on).
so initially, the output of NOR gate will be 0 & once the counter overflows, the output of NOR gate will be 1.

means at the point when all these values will become 0, the output of NOR gate will be 1.

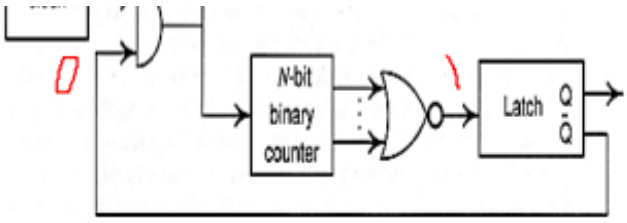
• Obviously, the NOR gate output will make a 0-to-1 transition when the counter overflows.

What will happen when this output will be 1?

It will actually set the latch

• This, in turn, will set the latch and freeze the duration counter.

the latch will be set nd this value will become 0 & it will freeze the duration counter. & the value contained in the counter at this point can be read by the interrupt handler.



The pupose of our experiment is to measure the fraction of time so it is desirable to choose the measurement interval as 2^N clock pulses where N is the number of bits in the counter. & it is preferable to let the measured system interact with the monitor at the start & end of the measurement section.

• The measurement can be controlled at the program level by providing a system call such as IO_OVERLAP(k, X)

• where X indicates the memory location in which the measurement result is returned.

• The system call could spawn a surrogate process as follows:

The system call can spawn a process it can perform various task like

• It can Determine the function code to load into the EDR using the parameter k.

• It can Initialize the EDR, clear both counters, and reset the

interrupt latch (should be done last).

& also then monitoring would start automatically

- The monitoring would then start automatically.
- & the process would Block until the monitor posts an interrupt. Then read the value from the duration counter, convert it to a real number, put it in location X, and exit.

from this example we can conclude that

- Hardware monitoring may be inconvenient for measuring software-level characteristics.-vedio

EDR is the Event Definition Register & we have already discussed that the instrumentation for the sampled hardware monitoring infact we have concludede that for hardware monitoring of type C sampled monitoring is the most appropriate type so the EDR the boolean function synthesizer the backplane all these things are already in place.
so for every problem they are already there in the setup & you will assume that the required signals will be obtained from the backplane i.e. the computer bus.

SOFTWARE MONITORING

Software Monitoring uses some measurement codes i.e. either embedded in the existing software or as a separate set of routine but it requires some support from the OS & the hardware.

Software monitors are used to monitor operating systems and higher level software such as networks and databases. At each activation, several instructions are executed, and therefore, they are suitable only if the input rate is low. For example, if the monitor executes 100 instructions per event, each activation would take 0.1 millisecond. Thus, to limit the overhead to 1%, it should be activated at intervals of 10 milliseconds or more. That is, the input rate should be less than 100 events per

second.-book 5 pg 109

• It is Used to monitor operating system and higher level software such as networks and databases.
• At each activation, several instructions are executed therefore are suitable only if input rate is low.
what is input rate?

• **Input Rate:** The maximum frequency of events that a monitor can correctly observe is called its input rate. Generally, two input rates are specified: burst mode and sustained.
Burst-mode rate specifies the rate at which an event can occur for a short duration. It is higher than the sustained rate, which the monitor can tolerate for long durations.-book 5 pg 109

Example:
• On a 1-MIPS machine, if a monitor executes 100 instructions per event then each activation would take 0.1 millisecond.
• To limit the overhead to 1%, it should be activated at intervals of 10 milliseconds or more.
since the overhead is higher, we actually want the monitor instructions to be executed at a some what longer interval So instead of executing or activating the monitor instruction at every 1 milisecond we actually enhancing that interval nd we want that it should be activated at the interval of 10 milisecond or more than the 10 milisecond in this way the input rate ...
• That is, the input rate should be less than 100 events per second.-vedio

Issues In Software Monitor Design-book 5 pg 111

This section covers some of the design issues that occur during the design of

software monitors.

1. *Activation Mechanism*: The first decision that a software monitor designer has to make is how to trigger the data collection routine. Three mechanisms have been used in the past:
(a) Trap instruction
(b) Trace mode
(c) Timer interrupt
The first mechanism is to instrument the system software with trap instructions at appropriate points in the code. The trap instruction is a software interrupt mechanism that transfers control to a data collection routine. The monitor measures elapsed time for various operating system services using trap instructions at the beginning and at the end of the service code. Thus, to measure I/O service time, a trap instruction placed at the beginning of the I/O service-call-handling routine enables the monitor to record the time clock. After finishing the I/O, another trap instruction reads the clock, subtracts the beginning value, and thus obtains the time spent in the service routine.

The second mechanism is that of changing the processor to the trace mode. In this mode, which is available on many processors, the instruction xecution is interrupted after every instruction, and the control is passed on to a data collection routine. This method has a very high overhead and is used only for those monitoring applications where time between events is not to be measured.

The final mechanism is that of a timer interrupt. A timer-interrupt service provided by the operating ystem is used to transfer control to a data collection routine at fixed intervals. This mechanism, called sampling, is specially suitable for frequent events since the overhead is independent of the event rate.

2. *Buffer Size*: Most software monitors record data in buffers, which is later written onto disk or magnetic tape for storage. The size of the buffers should be large so that the frequency of writing onto the secondary storage is minimized. The size should be small so that the time lost per writing operation is not too large and so the effect of reduced memory available for system usage is not perceptible. Thus, the optimal buffer size is a function of the input rate, input width, and emptying rate.

3. *Number of Buffers*: Buffers are usually organized in a ring so that the recording (buffer-emptying) process follows the monitoring (buffer-filling) process as

closely as possible. If there is only one buffer, the two processes cannot proceed simultaneously, and monitoring may have to be stopped while recording is in progress. Thus, a minimum of two buffers is required for continuous, simultaneous operation.

4. ***Buffer Overflow***: In spite of multiple buffers per ring, there is always a finite probability that all buffers become full. The monitoring process is required to either overwrite a previously written buffer or stop monitoring until a buffer becomes available. In either case, some information is lost. If a buffer is overwritten, the relatively old information is lost. Whereas if the monitoring is blocked, new information is lost. Thus, the choice between the two alternatives depends upon the value of old versus new information. In either case, the fact that the buffer overflow occurred should be recorded.

5. ***Data Compression or Analysis*****:** It is possible for the monitor to process the data as it is observed. This helps reduce the storage space required as the detailed data need not be stored. However, it adds to the monitor overhead.

6. ***On/Off Switch*****:** Most hardware monitors have an on/off switch that enables/disables the monitoring operation. A software monitor should similarly have conditional (IF ... THEN ...) statements so that the monitoring can be enabled/disabled easily. Since monitoring does add to the system overhead, it should be possible to disable the monitor when it is not being used. Also, a software monitor usually requires modification of system codes, which may introduce bugs. The on/off switch helps during monitor development and debugging.

7. ***Language*****:** Most monitors are written in a low-level system programming language, such as assembly, Bliss, or C, to keep the overhead at a minimum. Since a software monitor is usually a part of the system being monitored, it is better to write both in the same programming language.

8. ***Priority***: If the monitor runs asynchronously, its priority should be low so that key system operations are least affected. However, if timely observation and recording of events is important, the priority should be high so that the delay in its execution does not cause a significant skew in the time values recorded.

9. ***Abnormal-Events Monitoring*****:** A monitor should be able to observe normal as

well as abnormal events on the system. Examples of abnormal events include system initialization, device failures, and program failures. In fact, if both cannot be accommodated, users may often prefer to monitor abnormal events at a higher priority than normal events. This is because abnormal events occur at a lower rate and impose less monitoring overhead than normal events. The abnormal events also help the user take preventive action long before the system becomes unavailable.-book 5 pg 111 & 112

SOFTWARE MONITORING DESIGN ISSUES		
S. No.	***Design Issue***	***Description***
1.	***Activation Mechanism***	a) Trap: The trap instruction is a software interrupt mechanism that transfers control to a data collection routine. The monitor measures elapsed time for various operating system services using trap instructions at the beginning and at the end of the service code.
		b) Trace Mode: the instruction execution is interrupted after every instruction, and the control is passed on to a data collection routine. It is used only for those monitoring applications where time between events is not to be measured.
		c) Timer Interrupt: A timer-interrupt service provided by the operating system is used to transfer control to a data collection routine at fixed intervals. This mechanism is called sampling.
2.	***Buffer Size***	Optimal buffer size is a function of the input rate, input width, and emptying rate.
3.	***Number of Buffers***	A minimum of two buffers is required for continuous, simultaneous operation.
4.	***Buffer Overflow***	The monitoring process is required to either overwrite a previously written buffer or stop monitoring until a buffer becomes available.

SOFTWARE MONITORING DESIGN ISSUES (Cont'd)

S. No.	Design Issue	Description
5.	*Data Compression or Analysis*	Monitor processes the data as it is observed. This helps reduce the storage space required.
6.	*On/Off Switch*	Most hardware monitors have an on/off switch that enables/disables the monitoring operation. A software monitor should similarly have conditional (IF ... THEN ...) statements so that the monitoring can be enabled/disabled easily.
7.	*Language*	A software monitor is usually a part of the system being monitored, it is better to write both in the same programming language.
8.	*Priority*	If the monitor runs asynchronously, its priority should be low so that key system operations are least affected.
9.	*Abnormal-Events Monitoring*	A monitor should be able to observe normal as well as abnormal events on the system. Examples of abnormal events include system initialization, device failures, and program failures.

-vedio

Software monitoring may seriously interfere with the normal functioning of the system & they can not be used to capture the fast occuring events. They are most appropriate for obtaining user program & OS related information such as time spent executing a particular routine, frequency of page fault or average number of processes in each possible state. -vedio

Software monitoring, as with hardware monitoring, still requires that we know ahead of time where the sampling measures are to occur within the system and the frequency of this sampling if our measurements are to have meaning.-book 2 pg 115

So if we talk about Sampled Monitor we need to know ahead of time where sampling measure are to occur with in the system & the frequency of this sampling as well.

The most imp facility that can be needed for controlling the time duation is Programmable timer that can be loaded with desired time interval & counts down & generates an interupt when the remaining time reaches 0 let suppose also we can use interupt

handling routine that can take appropriate action & they can process collected data & closing down the experiment & the imp facility could be a virtual clock that can measure some process specific time duration & so on.

All these facilites can be used to perform software sampled monitoring.-vedio

In trace monitoring, the analyst adds additional code to a code sequence so that the code's run time can be monitored. Typically we would be interested in how often a code segment is entered, how long the code segment runs, or how much of the total systems time the code segment utilizes.-book 2 pg 115

In trace monitoring, the analyst can add some extra code to a program to record some certain information whenever events of interest occur. For example how often a code segment entered, how long it run or how much of the total system time the code segment utilizes. Also the average time taken for I/O operation on a given device etc.
The main advantage is that such codes are flexible & it involves the inclusion of appropriate statements in the software.
However this flexibility can also be a drawback for some reasons like one must clearly understand the functioning of program to instrunment them.
Also the edit code could contain bugs and make the program misbehave & the edit code could also interfere with the program in unexpected ways.-vedio

Task

A newly developed compiler for C was found to be much slower than expected. After some preliminary investigation, it was suspected that the compiler was spending too much time

manipulating the symbol table.

Show how the fraction of time used for symbol table manipulations, denoted frac, can be measured accurately using software monitoring.-vedio

if the fraction of time spend in manipulating symbol table depends on the program to be compiled.

solution ?

either you can use sampled monitoring technique or you can use trace monitoring

HYBRID MONITORING

• A monitor using a combination of software & hardware is a hybrid monitor.

• Involves transferring relevant information under software control from system under measurement to a set of interface registers where it can be picked up by a measuring system.

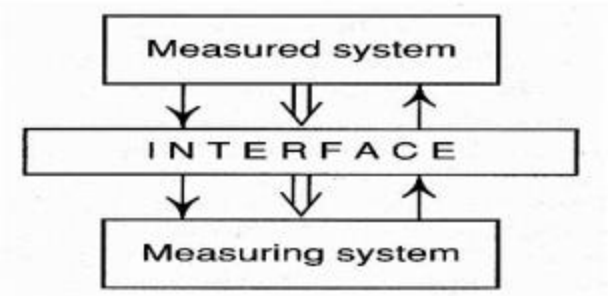


Fig 2: Setup for hybrid measurement

This is a setup of hybrid measurement. There are two systems; Measured system & a Measuring system.
under the software control the system that is under measurement can sent relevant information to the set of interface registers & those information can be picked up by the measuring system.
The main idea is to allow complex measurements without intolerable perturbations in the workload of the measured system. It means that we dont want any external disturbances in the normal operation of the system to be measured.

Task

Consider the measurement of the relative frequency of instruction usage.

a) Discuss pure hardware monitoring and software monitoring solutions for the given problem.

do it by yourself

b) Discuss the drawbacks (if any) of both of these methods.

first suggest solution for pure hardware & software monitoring & then discuss the drawback of both of these methods

c) Suggest the solution using hybrid monitoring.

d) Highlight the advantages of hybrid monitoring for the given problem.

Solution (Part c)

Consider the setup of hybrid measurement

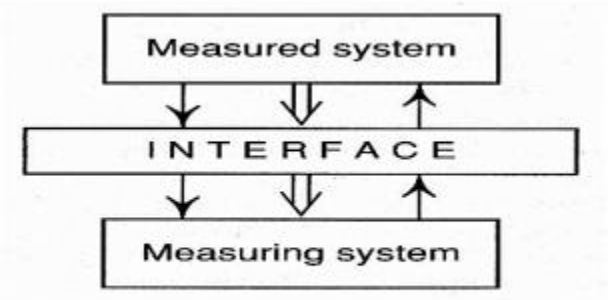


Fig 2: Setup for hybrid measurement

• Hybrid monitoring suggests to do half of the job in hardware and half in software.

• For that scheme, we will use two general purpose computers: one under measurement and one to measure.

Between these two system we have got an interface.
let suppose we want to measure the frequency of load instruction usage whenever the load instruction is used
the first ..

• The first system outputs the opcode to an interface register.

• The second system picks up the op-code from this register and uses it to increment the COUNT array, which is now stored in the main memory of the measuring system.

Solution Part (d)

Advantages of Hybrid Monitoring:

1) One clear advantage over the hardware technique

• we have a full-fledged system to do the measurement;

• So change in measurement objectives will require no new hardware. & the setup for hybrid monitoring will remain the same.

2) A second advantage:

• The control software on measured system can be designed to place any relevant information on system bus, from where it can be picked up by the interface.

• In the context of hardware monitoring, this approach can be used to expose information in main memory, machine registers, and even devices.

• It is also possible to provide DMA capability in the interface.

3) Advantage over software technique: the measurement now has very little effect on functioning of the measured system; and is fast, hardware level signals can be captured.

Set-up of Hybrid Measurement

1) The control program(s), if any, to be run on the measured system.

2) What information the monitoring interface should capture and when?

- A number of general purpose registers and flags, and

• the setup phase will decide what information goes into what register/flag, when it is written and read, etc.

3) How the measuring system should retrieve information from the interface and what to do with it (display, store, etc)?

4) How the measured and measuring system would synchronize?

• Use some flags in the interface and

• a protocol be set up for setting and clearing these flags.-vedio

Measured System	Measuring system
loop Process next instruction; busy-wait until $F_2 = 1$; Reset F_2; R ← instruction opcode; Set F_1; forever;	loop Wait until $F_1 = 1$; Reset F_1; Read opcode from R; Increment COUNT[<opcode>]; Set F_2; forever;

Fig 4: Two-way synchronization in hybrid measurement

-vedio