Simulation and Modeling of Data Acquisition Systems for Future High Energy Physics Experiments

Alexander Booth, Dennis Black, Don Walsh, Mark Bowden & Ed Barsotti
Fermilab*

Abstract

With the ever-increasing complexity of detectors and their associated data acquisition (DAQ) systems, it is important to bring together a set of tools to enable system designers, both hardware and software, to understand the behavioral aspects of the system as a whole, as well as the interaction between different functional units within the system. For complex systems, human intuition is inadequate since there are simply too many variables for system designers to begin to predict how varying any subset of them affects the total system. On the other hand, exact analysis, even to the extent of investing in disposable hardware prototypes, is much too time consuming and costly. Simulation bridges the gap between physical intuition and exact analysis by providing a learning vehicle in which the effects of varying many parameters can be analyzed and understood. Simulation techniques are being used in the development of the Scalable Parallel Open Architecture Data Acquisition System at Fermilab. This paper describes the work undertaken at Fermilab in which several sophisticated tools have been brought together to provide an integrated systems engineering environment specifically aimed at designing DAQ systems. Also presented are results of simulation experiments in which the effects of varying trigger rates, event sizes and event distribution over processors, are clearly seen in terms of throughput and buffer usage in an event-building switch.

Introduction

At Fermilab a project is underway to produce a Scalable Parallel Open Architecture Data Acquisition System [1][2] for future High Energy Physics (HEP) Experiments. From the outset of the project a goal has been to provide an integrated systems engineering environment in which hardware and software development can proceed in parallel and actually complement one another. To achieve this, it was necessary to bring together a set of tools which not only allow extensive exploration of all aspects of the design, but also to provide building blocks that encourage the close interaction of software and hardware engineers. This approach has had the advantage that valuable information is constantly being communicated between hardware and software groups during the development process. The powerful tools which were set in place included a digital logic simulator and Computer Hardware Description Language (CHDL), a high-speed graphics package and a knowledge-based expert inference system, all running on a very powerful work station. Although all of these tools are very useful when used in isolation, their combined effect is even more powerful and versatile. For example, in order to configure, download, monitor and diagnose the "model" of the data acquisition system, a user interface has been developed which accommodates these functions in a very friendly way. The requirements of this interface are in most cases identical to those of downloading, monitoring, and diagnosing the data acquisition system of an actual physics experiment. If the model is an accurate representation of the actual system, then everything that a user would like to do to the system, he would also like to do to the model. Therefore, as the model is developed, the actual software used to run the experiment is also being developed in parallel in an integrated fashion, thereby providing a universal interface accomodating the model and the "real" system.

Another example of integrated systems engineering is the development of system diagnostics and their integration into the hardware design during the simulation process. Good systems diagnostics are crucial for minimizing downtime in a running experiment. In order to diagnose something, it helps to understand it. Before any hardware is actually built, diagnostic strategies are evolving and being tested on the "model".

Tools

Verilog

The first tool we chose was Veriolog -XL [3], which is a digital logic simulator based on the computer hardware description language Verilog -HDL. It provides advanced simulation capabilities designed to handle complex electronic designs. It has many features which can be summarized as follows:-

(i) Different levels of abstraction; system/architectural, behavioral/algorithmic/register transfer, gate/circuit
(ii) Mixed level modeling
(iii) Stochastic analysis
(iv) Interactive debugging environment
(v) Open Design Environment; library support, programming language interface (C, etc.)

These features make Verilog very useful for modeling data acquisition systems, even those which include multiple ASIC's and complex VLSI devices. Verilog has already been used successfully in the HEP community [4]. Verilog provides three kinds of graphical display; firstly a logic analyzer type display which is very useful for low level

* This work performed under the auspices of the United States Department of Energy.

debugging, secondly a register display where the programmer can display the value of any variable or register in his HDL code, and thirdly a bar graph display for showing such things as queue occupancy and overflows in the stochastic/queue management type modeling.

DataViews

The next tool we chose was Dataviews [5], which is a powerful graphical design environment for developing custom color displays for real-time monitoring and control. This is a very important aspect of the Scalable Parallel Open Architecture Data Acquisition System; that the "User Interface" be state-of-the art.

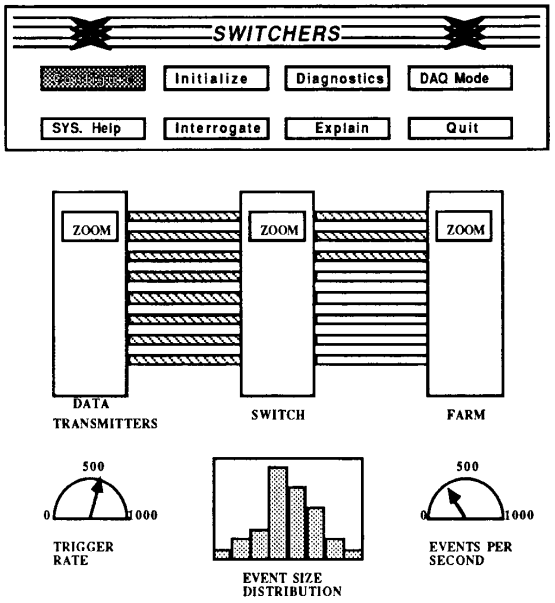


Figure 1 Example of Dataviews graphics

Dataviews is written in C and runs on most 32-bit workstations. It comprises two main components; a powerful drawing editor called DVdraw, and a comprehensive set of utilities called DVtools. DVdraw enables users to create and modify color pictures, and in our case these were system type diagrams of our user interface as wells as menu driven displays for user interaction.

DVtools allows the user to specify the dynamic interactions of all components on each screen and between screens, as well as how to integrate the displays into user application programs. We have also used Dataviews to "front-end" Verilog, so that the infrequent user of Verilog can come along and set up parameters for a simulation run in a user friendly way, without having to know Verilog HDL code.

Nexpert

The third tool we chose was Nexpert [6] which is a powerful "knowledge representation and reasoning" system. It includes a rule and "object" expert system shell and was particulary attractive to us for three reasons; firstly it meets some goals we have in terms of diagnosing data acquisition systems, secondly it includes a unified database bridge which interfaces to a variety of database packages, and thirdly it has a software bridge to Dataviews. This last feature was very attractive to us since the bi-directional relationship between Nexpert and Dataviews means that by clicking on buttons on a Dataviews "view", rules fire which can cause for example, other programs to be invoked (such as reading status registers in the DAQ system), and even other "views" to be displayed (such as a lower level in the DAQ system).

Modeling & Simulation

The purpose of modeling and simulating the switch-based Scalable Parallel Open Architecture Data Acquisition System is to provide a learning vehicle whereby the system designers can experiment with different architectures and control mechanisms to enable them to better understand DAQ design. An improved understanding simplifies decisions such as which operation mode provides for highest throughput, what extra electronics and software should be implemented to more efficiently diagnose failures and fix problems, etc. Modeling and system simulations assist system designers in determining throughput for different configurations, identifying potential bottlenecks, interfacing to "physics data" simulations, identifying busiest channels, selecting proper buffer sizes, determining the number of processors and processing power required, determining data rates, and many other decisions which are normally made using analytical calculations or intuition.

Simulation Experiments

This section describes one of the many simulation experiments which have been undertaken as part of a whole range of simulation studies being performed for the Scalable Parallel Open Architecture Data Acquisition System. Included in what follows is a description of the interaction between different Verilog models, as well as some results which were produced. Some graphs are presented which show throughput and the usage of buffers in Time Slot Interchangers (TSI's), for different trigger rates, event sizes, and event distribution over output channels.

Switch Operation

A detailed explanation of how the switch operates shall not be presented here, but can be found elsewhere [2]. However, an overview of how a switch-based DAQ system operates shall now be given so that the models described later will be more meaningful to the reader. When the trigger system decides that an event is worth forwarding to the farm for further processing, it sends a message containing the trigger type, event id., and processor id. to the "data transmitters" and the input TSI's (see figure 2). All the "data transmitters" in parallel send their "sub-event" to their corresponding input TSI's. Inside the input TSI's there is a logical buffer corresponding to each switch output. When data arrives at the input TSI it goes into a buffer corresponding to the output for which it is destined,

for example, if a sub-event is destined for a processor on o/p channel 3, then it goes in buffer 3 in the input TSI. Although sub-events are of varying size, the switch passes fixed length packets of data (e.g. 512bytes) through the switch at any one time. For an 8 by 8 switch, a packet of data is passed from input 0 to output 0 at the same time as a packet is transferred from input 1 to output 1, etc. Therefore 8 packets are transferred for any one rotation of the switch.. The next rotation of the switch will see input 0 connected to output 1, and input 1 connected to output 2 etc. again for 1 packets worth of time. The output TSI's have in them a logical buffer corresponding to each switch input. When data arrives at the output TSI, it goes into a buffer corresponding to the input from which it came, for example if input 3 is connected to output 5 then the data goes in buffer 3 in the output TSI. When the output TSI has a sub-event from all the input TSI's it can then send a complete event to the processor for which it was destined.

Simulation Goals

It is important at the outset of any simulation experiment to specify clearly the goals of the exercise. For this particular simulation experiment the goals were as follows:-

a) to develop simulation models of the following functional sub-units of the switch-based DAQ system:-

(i) trigger system interface
(ii) test transmitter module
(iii) switch
(iv) switch control
(v) input time slot interchanger
(vi) output time slot interchanger

b) to develop a "system" model of an 8 by 8 switch architecture, using each of the above functional sub-components

c) to observe the behavior of the "system" model as well as each of the functional sub-units when varying certain subsets of the following parameters:-

(i) trigger rate
(ii) event distribution over switch output channels
(iii) event size
(iv) buffer depth in the input TSI's
(v) buffer depth in the output TSI's
(vi) switch packet size
(vii) test transmitter to input TSI data transmission rate
(viii) input TSI to switch data transmission rate
(ix) switch to output TSI data transmission rate

d) to perform analyses and produce meaningful results in terms of summary plots and printouts

Fixed parameters

Although we have experimented with different buffer sizes in the input and output TSI's, in the particular simulation experiment being reported here, they were fixed at 1Mbyte.

Assumptions

The following assumptions were made for purposes of simplicity and clarity:-

(i) The test transmitter modules had an infinite depth in terms of their ability to respond to triggers.

ii)Since no processor farm was modelled, it was assumed that there was always a free processor available to receieve an event from an output TSI

Architecture

The following architecture was modeled. It is an 8 by 8 switch consisting of the following individual functional blocks:-

trigger system interface
test transmitter module
input time slot interchanger
output time slot interchanger
switch
switch control

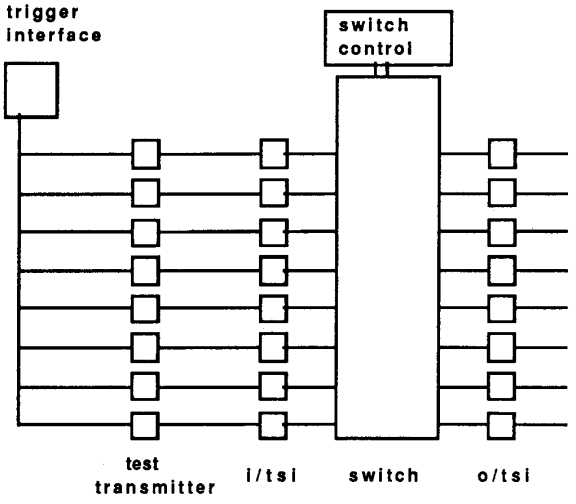


Figure 2: 8 by 8 Switch

Functional Description of models

Trigger system interface model

The "trigger interface" module uses one of several user selectable probability distributions to generate a trigger, which indicates that a digitized event is about to be transmitted from the front-end to the switch. (This trigger is really the amount of time it must wait before generating the next "trigger"). The trigger interface module also generates the destination ID of the processor to which the next event is to be routed; this is done on a "round robin" basis or by using a probability distribution function. In this simple model the processor ID is the same as the channel ID since in this particular simulation experiment it was assumed that there was always an available processor on every channel to take the next event. The trigger interface module passes

event_ID and processor_ID (channel ID), to the "test transmitter" modules on a "trigger accept" link. In this simple model this is facilitated by toggling a "ready" signal but in later models a refinement shall be made so that some time penalty is incurred with this operation.

Test transmitter model

The "test transmitter" module replaces that component in an actual switch-based data acquisition system that would be responsible for gathering all the time-ordered data from multiple data collection chips (DCC's - probably multiplexed at several levels) and transmitting sub-events to the switch. In the simulation experiment being reported here, the "test transmitter" module on seeing the "ready" signal from the "trigger interface" module, uses a "gaussian" (normal) distribution function to generate a sub-event size, and then concatenates event size, processor ID and event ID into a single "header" word which it puts on a FIFO queue. (For simulation purposes the depth of this queue is assumed infinite so that regardless of the trigger rate, events are not lost). In reality the depth of this queue need be only as deep as there are front-end buffers, since "level 1" would have already reduced the number of events arriving at the data transmitters. In the "test transmitter" module another process is responsible for taking a "header" word off the QUEUE, waiting a time equal to transmitting the data to the input TSI, and then toggling a "data valid" signal to indicate to the input TSI that another sub_event has arrived.

Input Time-Slot Interchanger (I/TSI) model

The input TSI has really two main processes, the first one is to see that a sub-event has arrived from the "test transmitter" module and to divide the sub_event size by the packet size to determine how many "packets" of data it must put ON a QUEUE. In each input TSI there is a queue corresponding to each output channel of the switch. If an event is destined for output channel 3, then the data arriving at the input TSI is put on QUEUE 3. The second process is concerned with taking packets OFF a QUEUE, and this is achieved by waiting for the "In_Pkt_En" signal from the "switch control" module, and then taking data from the NEXT queue and presenting it on the inputs of the switch, where NEXT is determinded in a "round robin" fashion.

Switch Control model

This module produces the timing signals needed by the switch and TSI's. It is responsible for issuing "In_Pkt_En" and "Out_Pkt_En".

Switch model

The actual switch module is very simple in that at every r sing edge of "Out Pkt En", it connects certain switch inputs to certain switch outputs for one packet's worth of time. At the rising edge of "Out Pkt En" one packet is passed from the input TSI to the output TSI.

Output Time-Slot Interchanger (O/TSI) model

The output TSI, like the input TSI has really two main processes; one for putting packets ON a queue and another for taking packets OFF a queue. Every time the output TSI sees the rising edge of "Out_Pkt_En" it takes a "packet" of data from the output of the switch and puts it ON the NEXT queue, (again NEXT is determined in a round robin fashion). The output TSI is really the "event builder" since many sub_events have been split into many packets by the input TSI and passed through the switch, they now have to be put back together. This is achieved by the second process of the output TSI model which looks to see whether data has arrived for event 'i' in buffer 0. If it has it transmits this data to the processor and moves to buffer 1. It there checks to see if there is an event fragment for event 'i', and then transmits this fragment to the processor. Any other event ID will cause an error. No data arriving for a long time will cause a "timeout". When the output TSI has been through all it's buffers (remember one buffer per each input channel), the event is complete. No "end of event" flag is needed (although, it may be advisable to put one in for diagnostic purposes).

Diagnostics

In the context of diagnosing a switch-based data acquisition system, we desire to narrow the problem down to a particular functional sub-unit as quickly as possible. One can imagine for example (using the Nexpert product described earlier), a "rule" which tests whether all nodes in the processor farm are "seeing" a particular problem, or whether nodes only on a particular channel are "seeing" the problem. If all nodes are "seeing" the problem, then the problem is probably pre-switch or on the input to the switch itself, whereas if only one channel is experiencing a problem then the problem is post-switch or on the outputs of the switch itself. Either way this simple rule has performed the first "binary chop" on our total DAQ system. At the present time, this is the least developed part of the project, but still remains a positive area for investigation and development.

Validation/Verification Code

The code for each module includes "always" blocks that describe the behavior of the module. It also includes "always" blocks which are there specifically to verify that the code is working and validate any results that are produced. For example, there is verification code in the output TSI to check that it has received all the packets that were put into the switch by the input TSI.

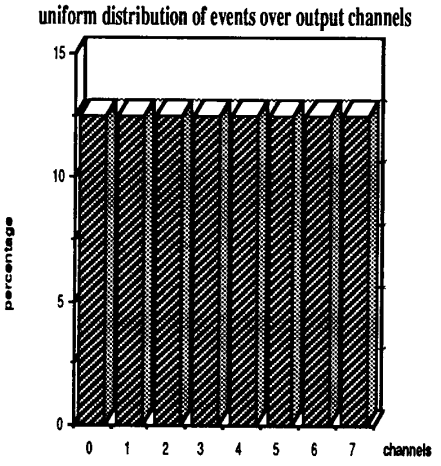
RESULTS

The following list shows some of the kinds of results obtained from the 8 by 8 switch simulation.

(i) a series of graphs showing min. and max. buffer usage at the input TSI for different probability distributions of events over output channels
(ii) a series of graphs showing min. and max. buffer usage at the output TSI for different probability distributions of events over output channels
(iii) a series of graphs showing min. and max. buffer usage at the input and output TSI's for different standard deviations about the mean in terms of sub_event sizes

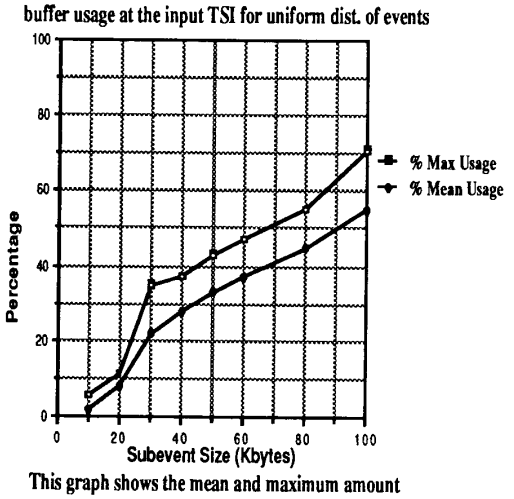
(iv) a series of graphs showing min. and max. buffer usage at the input and output TSI's for different packet sizes
(v) a graph showing throughput in relation to trigger rate and event_size

As a sample of the results, let us consider the following graphs. The first two show that for a uniform distribution of events over output channels (Graph A); i.e. each of the 8 output channels recieved 12.5% of the total



This graph shows what percentage of the total events were sent to each output channel

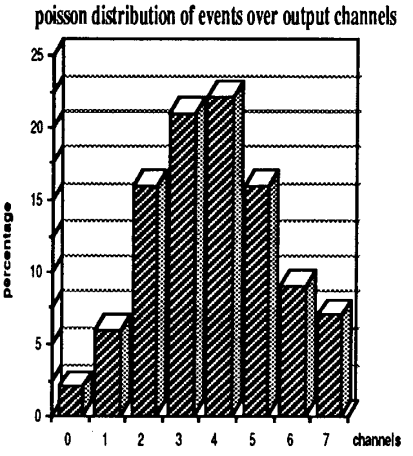
GRAPH A



This graph shows the mean and maximum amount of buffer usage for different subevent sizes

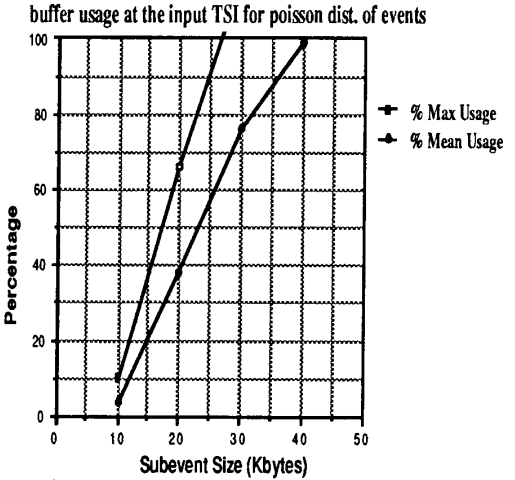
GRAPH B

events through the switch,the percentage of buffer memory used in the input TSI's for sub-event sizes in the range 10Kbytes up to 100Kbytes (Graph B). Here, the switch is seen in a very favorable light since events are evenly distributed over output channels. For example, for sub_event sizes of 60 Kbytes (i.e. event size 480Kbytes) the buffers in the input TSI's were only 38% used on average (mean), and their maximum usage was 48% (still less than half full). The next two graphs show that when events are distributed over output channels according to a "poisson" distribution (Graph C), the switch performs in a much poorer fashion. For example, in Graph D we can see that for a sub-event size of only 30Kbytes (i.e total event size of 240Kbytes), the buffers in the input TSI's were 75%



This graph shows what percentage of the total events were sent to each output channel

GRAPH C



This graph shows the mean and maximum amount of buffer usage for different subevent sizes

GRAPH D

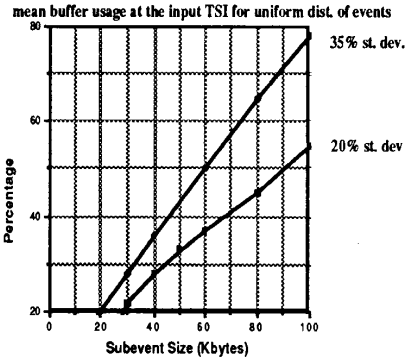
full on average, but what is even worse is that approximately 12% of the events overflowed (as show by the max. trace). The fact that we used a "poisson" distribution (as opposed to say gaussian) is not really important, but just that the events were distibuted

"unevenly" across the output channels. This may be interesting since some in the HEP community have discussed routing certain triggers to certain processors (channels), and doing so obviously degrades the performance of a barrel shifting event builder.

In the case of the "uniform" distribution, a trigger rate was chosen such that for sub-events of 30Kbytes and over the link between the test transmitters and the input TSI's was saturated, i.e. the bandwidth was fully used. In the case of the "poisson" distribution a trigger rate was chosen such that the link was saturated for all sub-event sizes.

Although one might want to extrapolate these results to predict the buffer usage in say a 64 by 64 switch, or a 1024 by 1024 switch, we still intend to simulate these large systems to understand other overheads which come into consideration, such as propogating the trigger information to more input TSI's etc.

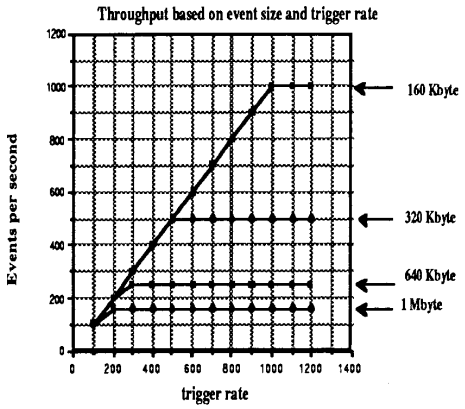
The next graph shows how the switch degrades in performance for a wider distribution of sub-event sizes. In the first simulation runs, the "gaussian" distribution function used by the "test transmitter" model generated event sizes with a standard deviation of +/- 20% about the mean. When we "flattened" the "gaussian" curve to have a "standard deviation" of 35% for example, the buffer usage increased significantly.



GRAPH E

Graph E shows that for sub-events with a mean of 60Kbytes and a st.dev. of 20% the buffers were 38% full (as we saw earlier. However, with the same mean (i.e. 60 Kbytes) and a st. dev. of 35% the buffer usage had increased to 50%. This is significant for designers and their choice of buffer depth in the input TSI's.

The final graph shows the throughput of an 8 by 8 switch for different event sizes and trigger rates. Clearly for detectors producing 1Mbyte events, one would use a much larger switch, but for smaller experiments with events in the 100Kbyte to 300Kbyte range, an 8 by 8 switch would be adequate.



Summary

Some design tools have been brought together and used to demonstrate that it is worthwhile simulating complex systems. Useful things can be learned, and in the context of a switch-based data acquisition system we have been able to identify potential bottlenecks, determine buffer depths, aid system diagnosis and better understand the system as a whole as well as the interaction between different sub-units. We believe this has placed us in a very strong position to build a hardware prototype, with a high confidence of it not only working, but also being able to fix it quickly when it breaks.

Acknowledgements

The authors wish to gratefully acknowledge all members of the design team of the Scalable Parallel Open Architecture Data Acquisition System, as well as Tom Nash and Irwin Gaines , Head and Associate Head of the Computing Division at Fermilab, for their enthusiastic support of this very important project.

We would also like to thank Jheroen Dorenbosch and Michael Botlo of the Superconducting Super Collider (SSC) for their pioneering work with other simulation packages.

References

1. "A Scalable Parallel Open Architecture Data Acquisition System for Low to High Rate Experiments, Test Beams & All SSC Detetcors", E. J. Barsotti, et al, IEEE Nuclear Science Symposium, San Fransisco, 1989.
2. "A High-Throughput Data Acquisition Architecture Based on Serial Interconnects", M. Bowden, et al, IEEE Transactions on Nuclear Science, Vol. 36, No. 1, February 1989, p760-764.
3. "Verilog-XL",Cadence, Lowell,Massachusetts.
4. "Simulation of a Macro-pipelined Multi-cpu Event Processor for Use in Fastbus", M.F.Letheren, A. Marchioro, F Slorath, CERN, Geneva, Switzerland.
5. "Dataviews", V.I. CORP,Amherst,Massachusetts.
6. "Nexpert",Neuron Data Inc.,Palo Alto, California.