SSCL-SR-1211

SSCL-SR-1211

Superconducting Super Collider Laboratory

Workshop on Data Acquisition and Trigger System Simulations for High Energy Physics

April 23–25 1992

Superconducting Super Collider

WORKSHOP ON DATA ACQUISITION AND TRIGGER SYSTEM SIMULATIONS FOR HIGH ENERGY PHYSICS

April 23 - 25, 1992

ORGANIZING COMMITTEE

A. W. Booth, SSC Laboratory

M. Botlo, SSC Laboratory

J. Dorenbosch, SSC Laboratory

P. Hale, SSC Laboratory

C. Milner, SSC Laboratory

R. Partridge, Brown University

P. Sinervo, University of Toronto

Operated by Universities Research Association, Inc.
Under Contract with the United States Department of Energy

TABLE OF CONTENTS

Agenda ..... *I*

Participants List ..... *IX*

E-Mail Addresses ..... *XI*

DAQSIM: A Data Acquisition System Simulation Tool - *A. W. Booth* 1

Front End and DCC Simulations for the SDC Straw Tube System - *A. Hölscher* 33

Simulation of Non-Blocking Data Acquisition Architectures - *R. Partridge* 61

Simulation Studies of the SDC Data Collection Chip - *E. Hughes* 87

Correlation Studies of the Data Collection Circuit &
The Design of a Queue for this Circuit - *G. Tharakan* 97

Fast Data Compression & Transmission from a Silicon Strip Wafer - *K. Lackner* 119

Simulation of SCI Protocols in Modsim - *A. Bogaerts* 139

Visual Design with vVHDL - *Diana Miller-Karlow* 165

Stochastic Simulation of Asynchronous Buffers - *H. Kasha* 187

SDC Trigger Simulations - *S. Dasu* 199

Trigger Rates, DAQ, & Online Processing at the SSC - *G. Sullivan* 229

Planned Enhancements to MODSIM II & SIMOBJECT - an Overview - *R. Belanger* 265

DAGAR - A Synthesis System - *V. Raj* 277

Proposed Silicon Compiler for Physics Applications - *G. Vanstraelen* 347

Timed - LOTOS in a PROLOG Environment: an Algebraic
Language for Simulation - *M. L. Ferrer* 359

Modeling and Simulation of an Event Builder for
High Energy Physics Data Acquisition Systems - *V. Kapoor* 397

A Verilog Simulation for the CDF DAQ - *R. Grindley* 445

Simulation to Design with Verilog - *T. Ekenberg* 453

The DZero Data Acquisition System: Model and Measurements - *J. A. Wightman* 477

DZero Trigger Level 1.5 Modeling - *M. E. Johnson* 495

Strategies Optimizing Data Load in the DZero Triggers - *M. Fortner* 507

Simulation of the DZero Level 2 Data Acquisition System - *D. Nesic* 521

A Fast Method for Calculating DZero Level 1 Jet Trigger Properties - *A. Milder* 541

Physics Input to DAQ Studies - *E. Wang* 551

Some Summarizing Remarks (From a Tools Perspective) - *A. Booth* 567

Workshop on Data Acquisition and Trigger System Simulations for High Energy Physics

Wednesday, 22-April-92

5:00 - 7:00 pm, reception and registration, Cafeteria, SSCL

Thursday, 23-April-92

8:30- 9:00 coffee and registration, main lobby, SSCL

9:00-10:30 **Morning Session**

(10 minutes) **Workshop Overview**, Cas Milner, *SSCL*

(45) **A Tool for Understanding Data Acquisition Systems**, A. W. Booth, M. Botlo, J. Dorenbosch, R. Idate, V. Kapoor, C. Milner, V. Raj, C. C. Wang, E. Wang, *SSCL*

At the Superconducting Super Collider a tool for studying the behavior of data acquisition systems has been developed. It is based on the MODSIM II object-oriented programming language. The tool allows system designers to evaluate alternative DAQ architectures in terms of deadtime, throughput and buffer usage etc. The user can specify dynamically the number of chips, the size of buffers, the amount of processing time, as well as the bandwidth of the links, etc., for various interconnection topologies. The type of input data is also user selectable, e.g. fixed-size, GEANT extracted, zero-suppressed or not, or random, according to a number of probability distributions. A network of Data Collection Circuits (DCC's) has been used as a test case for the tool, where "push" and "pull" control strategies have been compared.

(30) **Simulations of the Data Collection Circuit for the SDC Straw Tracker**, A. Holscher, G.Stairs, P.K.Sinervo, *University of Toronto*

A VERILOG model for the front end readout of the SDC Straw Tube Tracker system is presented. The model investigates the buffer and bandwidth requirements on the front end card and on the carte interface card, which is housed outside the detector. Furthermore the required pipeline lengths for the L1 and L2 buffer are presented.

10:30-10:45 Break

10:45-12:15 **Morning Session Continued**

(30) **Simulation of Non-Blocking Data Acquisition Architectures**, Richard Partridge, *Brown University*

A generic simulation model has been developed to study the throughput and deadtime for a broad class of data acquisition architectures using the VERILOG simulation language. The results should be applicable to any non-blocking architecture where the throughput between a particular source and destination is not affected by the data rate among other sources and destinations. The effect of varying various parameters in the model is studied in terms of event throughput and deadtime.

(30) **Deadtime Studies of the SDC Data Collection Circuit**, Eric Hughes, *University of Illinois*

The SDC (Solenoidal Detector Collaboration) Data Collection Chip (DCC) reads data from the Front Ends (FEs) of the different detector subsystems. The DCC assembles event information from a number of input channels, so a non-overlapping tree structure of DCCs can be used to build appropriately-sized event fragments. We will present modeling and simulation studies which examine the effect of DCC design parameters on the deadtime of the detector. In the first study, three possible architectures of the DCC are examined in detail, each of which uses a different protocol for communication. The relative effects of trigger throttling and data loss are examined. In the second study, a configurable model of the DCC is developed. This model is used to construct trees of DCCs. The results demonstrate the feasibility of a parameteric DCC model for the SDC DAQ. The DCC design can accomodate different data formats and data rates, and is appropriate for a variety of detector subsystems.

(30) **Correlation Studies of the Data Collection Circuit**, George Tharakan, *University of Illinois*

The SDC (Solenoidal Detector Collaboration) Data Collection Chip (DCC) is responsible for reading out the Front Ends (FEs) of the different detector subsystems. One concern with the DCC design model is the interdependence of input data. We have studied the effect of input data correlation on buffer requirements for the DCC. The result suggests a simple linear relationship between the correlation of data on input channels and the buffer space needed for normal operation. This implies that physics motivated stimuli (e.g., Monte Carlo generated events) will be necessary for reliable DCC simulations. Another problem with the present state of the DCC design is a lack of candidate implementations and prototypes, which would be useful for upcoming test installations. We will present some preliminary implementations and discuss a collaborative effort with a FE design group. One goal of this collaboration is to ensure the feasibility of the DCC design.

12:15- 2:00 Lunch

2:00- 3:30 **Afternoon Session**

(45) **Efficient Data Transmission from Silicon-Wafer Strip Detectors**, Klaus S. Lackner, *Los Alamos National Laboratory*

An architecture for on-wafer processing is proposed for central silicon-strip tracker systems as they are currently designed for high energy physics experiments at the SSC. The data compression achievable with on-wafer processing would make it possible to transmit all data generated to the outside of the detector system. The band-width requirements are obtained from simple numerical simulations of the wafer occupancy. A set of data which completely describes the state of the wafer for low occupancy events and which contains important statistical information for more complex events can be transmitted immediately. This information could be used in early trigger decisions. Additional data packages which complete the description of the state of the wafer vary in size and are sent through a second channel. By buffering this channel the required bandwidth can be kept far below the peak data rates which occur in rare but interesting events.

(45) **Simulation of SCI Protocols in MODSIM§**, Andre Bogaerts, *CERN*

Modern system design relies more and more on computer modelling and the Scalable Coherent Interface (SCI) is no exception. Different tools are used to simulate different aspects. The standard is defined by the IEEE as executable C-code, which allows conformance verification. VERILOG has been used by Dolphin Server Technology (Oslo, Norway) to design the SCI "node chip" interface. The behavioural simulation of the node chip in VERILOG has been cross checked against results obtained from the IEEE C-code using identical stimuli. The latter required a multi-thread environment which was provided by the SUN/SPARC implementation of "light weight processes". The Department of Informatics of the University of Oslo has used models written in both SIMULA and C++ to model small SCI based systems. At CERN we have used ModSim II to model large (> 1000 nodes) SCI based Data Acquisition Systems for LHC. The model simulates accurately SCI protocols at the packet level. Since SCI packets vary in size, the simulation is necessarily asynchronous with a typical time resolution of ~ 50 ns (the average packet length at transmission speeds of one Gbyte/s). The simulation program (SCIMP, SCI Modelling Program) typically calculates the traffic on SCI interconnects, the flow of data in and out processors or memories in a network consisting of SCI rings and interconnects. The model takes into account SCI protocols for bandwidth allocation (roughly the equivalent of arbitration in bus based systems) and retransmission of packets to overloaded SCI memories or processors. Recently, work has started to include cache coherency in the simulation.

The different simulations are complementary. SCIMP does not simulate the content of packets which are exchanged between SCI nodes, although it could easily be extended to do this. It does not simulate the exact details of how the hardware emits or strips symbols of the interconnect. SCIMP models the load on an SCI network, but does not generate test patterns. Recently, the Physics Department of the University of Oslo has acquired ModSim with the aim of providing more functionality for the simulation of LHC Data Acquisition systems. The experience using ModSim, the design of SCIMP, results obtained so far will be presented.

(§ModSim is a trademark of CACI, La Jolla, Ca.)

3:30- 3:45 Break

3:45- 5:30 **Afternoon Session Continued**

(30) **Demonstration: Visual VHDL**, *University of Illinois*

(20) **Stochastic Simulation of Asynchronous Buffers**,
Henry Kasha, *Physics Department, Yale University*

Events accepted by the first-level trigger of a collider detector, while no longer synchronous with the beam bunch crossing frequency, may still occur at time intervals which are shorter than the decision time of the next level trigger processor. An asynchronous buffer must therefore be used to derandomize the arrival times at that level. A MODSIM II simulation is used to study the required depth of such buffers in SSC environment.

(35) **SDC Level 1 Calorimeter Trigger Simulation Study,**
S. Dasu, T. Gorski, J. Lackey, W. H. Smith, W. Temple, *University of Wisconsin* (Department of Physics and Astronomy, Chamberlin Hall, 1150, University Avenue Madison, WI 53706)

Level-1 trigger electronics for the SDC detector is required to reduce the interaction rate of 100 - 1000 MHz, down to 10 - 100 kHz. The key to triggering the SDC involves tracking, calorimetry and muon identification, as well as correlation of information from these systems. The trigger system starts by making local decisions about the presence of objects such as photons, electrons, muons, and jets, as well as making global sums of ET and missing ET. It uses this global compilation of information to decide whether to trigger a particular 16 ns beam crossing or not. Extensive simulations of electron, photon and jet triggers have been made in order to optimize various important trigger parameters. These studies have been performed, using hundreds of thousands of ISAJET events, and, reconstructing the calorimeter energy deposits and tracking using a parametric Monte Carlo. The ISAJET data set included both Drell-Yan, QCD and min-bias events. We conclude that the electron trigger threshold can be reduced to as low as 20 GeV while maintaining high trigger efficiency, using, comparison of energy deposited in electro-magnetic and hadronic parts of the calorimeter, and transverse isolation. We conclude from the jet study that the overlapping grid of 1.6 x 1.6 (ϕ, η) towers yields best jet energy trigger threshold. Additional work involved study of logarithmic versus linear digital energy scale, and other details. Further improvements to the Monte Carlo, including the addition of muon system performance, study of trigger efficiencies for various physics processes, etc. are in progress. Results from these studies will be discussed in this paper.

(20) **Trigger Rate Simulations for the SDC**, Greg Sullivan, *University of Chicago*

A fast detector simulation of the SDC is used to estimate trigger rates for the SDC detector. The rates are discussed along with the implications for an overall trigger strategy in the SSC environment.

Discussion, (J. Dorenbosch, *agent provocateur*)

6:30 Informal Dinner (Dutch Treat), Mercado Juarez

Friday, 24-April-1992

9:00-10:30 **Morning Session**

(45) **MODSIM II Simulation Tools**, Ron Belanger, *ErgoSoft*

(45) **Dagger: A Design Automation Tool,**
Vijay Raj, *University of Texas at Arlington*

10:30-10:45 Break

10:45-12:15 **Morning Session Continued**

(20) **Proposed Synthesis Tool for Physics Applications,** Guy Vanstraelen, SSCL

A "Silicon Compiler" can be defined as a software system supporting chip layout synthesis starting from a behavioral description at the algorithmic level. Important is the observation that there is no such thing as a "general" silicon compiler. Only application-specific silicon compilers can be realized, because different applications require different specification languages and the designer may want to pass structural hints to the compiler to guide the synthesis process. The required characteristics of a synthesizer for physics problems will be examined in more detail. These problems are characterized by high throughput requirements, memory management needs and an interdependent incomming data stream.
An important strategy in the simulation-synthesis process should be the "meet in the middle" strategy, in which the top-down approach of system designers meets with the bottom-up approach of silicon designers at the module level. This strategy will be explained more in detail.

(30) **Timed-LOTOS in a PROLOG Environment: an Algebraic Language for Simulation,** M. L. Ferrer, *Laboratori Nazionali di Frascati dell'INFN, Frascati, 00044, Italy,* and G. Mirabelli, *Dipartimento di Fisica, Universita' di Roma I and INFN, Roma, 0185, Italy*

The time inclusion in LOTOS allows its use for protocol and system simulations.A tool will be presented, implementing a timed-LOTOS in PROLOG environment performing the following functions: evaluation of PROLOG clauses via guardmode, menu computation and selection and automatic traversal of the communication tree. An example of use of this language to define and simulate some aspects of the FUTUREBUS+ protocol will be given.

(30) **Modeling and Simulation of an Event Builder for High Energy Physics Data Acquisition Systems,** Vishal S. Kapoor, *University of Texas at Arlington*

In data acquisition systems for high energy physics experiments, the physical connection between individual data sources (data collection chips) and data destinations (processors) can be made using an event builder. An event builder assembles the data associated with an event, and sends it to a processor. Specifically, this involves funneling and routing a large volume of data to a large number of processors. In essence, an event builder is an interconnection network.
An integrated systems engineering approach was used to model, at both the systems level and the component level, the behavior and functionality of an event builder. This provided a vehicle to study high level considerations such as system architecture, efficiency, dead time, event sizes, event distributions, data rates, and transmission rates, as well as component level issues such as functionality and relative signal timing. System behavior was studied using MODSIM II, an object-oriented discrete-event simulation language. Some design issues were studied in greater detail using VERILOG, a hardware description language.
The study was done by varying data volumes, trigger frequencies and buffer sizes. The system was evaluated by analyzing the buffer usage and throughput of the event-building interconnection network.

12:15- 2:00 Lunch

2:00- 3:30 **Afternoon Session**

(45) **CDF DAQ Simulation in Verilog**, Robin Grindley, *University of Toronto*

A Verilog simulation of the CDF DAQ system for the 1992 run was developed to investigate the dependence of throughput on the system geometry. The goal was to test certain proposed enhancements to the system to see whether they would be effective or not in achieving the design event-taking rate. The talk will center on the purpose, structure and implementation of the simulation, but will briefly touch on the results obtained and the general usefulness of such simulations in the design of DAQ systems.

(45) **Modeling the SDC Straw Tube Front End Interface in Verilog**,
Tor Ekenberg, *University of Pennsylvania*

We are using verilog to simulate the read-out of the front end-chips, i.e. the interface between the DCC and the FE, for the straw tube read-out system. Since we are working on a design of this interface for use in the "full-density" straw tube test, it is important that the models can easily be transfered to a hardware implementation. This is accomplished under the CADENCE IC design environment by tying behavior-level verilog code to blocks in the schematics, and then replacing the behavioral level blocks with actual transistors as we proceed with the design.
Advantages of this simulation path are, 1) that we have can have higher confidence in our design since the topology of the schematics haven't changed between simulation and implementation, and 2) that we relatively easily can lift certain blocks out of the half-done design and replace them with different structures (written in verilog) to investigate the impact of design changes.
We have designed a four channel version of the read-out logic that can handle the TMC or the TVC/AMU. This design is done inside CADENCE on a block-diagram basis, where all blocks have verilog code tied to them, and then simulated to verify correctness in the logical design. This phase is almost complete. In parallel we have been filling in the blocks we know we need (like counters, comparators, flip-flops, latches, etc) with gate level schematics, and in most cases also with lay-outs. For parts of the design we are doing verilog simulations where some blocks only have high-level behavioral level verilog code, while other blocks have complete gate level designs associated with them.
For the first version of the actual TVC/AMU we made a 10,000 transistor hSpice job simulate. We simulated 20 ms of detector time on a 1 ns time step in a three week simulation.

3:30- 3:45 Break

3:45- 5:00 **Afternoon Session Continued**

(30) **The DZero Data Acquisition System: Model and Measurements§**,
J. A. Wightman, *Texas A&M*, R Angstadt, M.E. Johnson, and I.L. Manning, *FNAL*

We have developed a queueing theory model of the DZero Data Acquisition System based on IBM's RESQ Version 2 modeling package. This model was used in the design of our DAQ system. As a check of the simulation, an analytical solution was developed which has the advantage of providing a self-consistency check. We have used the analytical solution to refine the model and find hidden queues which had not been previously considered. The analytical solution provides a simplification of the more complex RESQ model and has been a very powerful tool not only in discovering bottlenecks in the DAQ system but also in comprehending why they exist. Finally, we have made some measurementsof the

performance of the DZero DAQ system in its current configuration. These are consistent with both the simulation and calculation. As the DZero DAQ system undergoes upgrades to its final configuration, we make projections of its performance.
§Supported in part by US DoE.

(30) **Simulating the DZero Intermediate Level Hardware Trigger§,**
M.E. Johnson, R Angstadt, and I.L. Manning, *FNAL*, and J. A. Wightman, *Texas A&M*

The DZero trigger consists of three levels. The first is a hardware trigger which makes a decision within the time frame of a single beam crossing. The last level of triggering is a software trigger which makes a decision within about 100 ms. Residing between these two is the intermediate level hardware trigger which operates within the time frame of approximately ten beam crossings. We have used our queueing theory model to study the effects of this level of triggering on the DAQ system. The processing rate for this trigger as well as the rejection factor supplied by the algorithm have served as parameters in our study. With the intermediate level hardware trigger still under construction, we have not been able to make any measurements to verify the predictions of the simulation, but the success of the simulation in modeling the rest of the DAQ system gives us great confidence in our results.
§Supported in part by US DoE.

(20) **Strategies Optimizing Data Load in the D0 Triggers,**
Mike Fortner, *Northern Illinois University*

The DZero trigger is a multistage object consisting of a synchronous hardware trigger, an asynchronous hardware trigger and a software trigger. Conceptually all three of these stages have the same types of cuts, but with increasingly better resolution. The effects of simulating background rates and data aquisition performance has led to specific strategies of triggering that involve spreading the trigger load between the three stages.

Discussion, (J. Dorenbosch)

6:30 **Informal Dinner**

Saturday, 25-April-92

9:00-10:30 **Morning Session**

(30) **Simulation of the D0/Level-2 Data Acquisition System,** D. Nesic, D. Cutts, J. Hoftun, M. Mattson, *Brown University*
The D0/Level-2 data acquisition system provides a high level software trigger for the D0 experiment. Data flows directly from digitizing crates over parallel high speed paths to a selected farm node for the filter analysis. We have modelled this system using the Verilog simulation package, and compared our model with measurements on the partial system that has been running in the installation/commissioning phase of D0. Using the simulation, possible upgrade paths for the Level-2 system are studied; results of different upgrade scenarios will be presented.

(30) **Fast Calculation DZero Jet Trigger Properties,**
Andy Milder, *University of Arizona*

Level 1 trigger efficiencies and their dependence on η and Pt have been calculated using single-jet events. Sytematic η effects which have been missed in other studies become apparent and represent a potentially important angular correlation efficiency problem. A method for quickly simulating the trigger efficiencies of specific physics processes has been developed and applied to two-jet QCD events as a check against other, more time-consuming means.

10:30-10:45 Break

10:45-12:00 **Morning Session Continued**

(30) **Physics Simulation for Input to Behavioral Simulations,**
Ed Wang, *SSCL*

At the SSC, triggering and data acquisition problems are particularly acute. It is ultimately physics goals that drive both the accelerator and detector design. We use these physics goals as input to behavior modelling of data acquisition systems. The present discussion describes features of the physics input to the behavior simulations presented at this workshop.

Workshop Summary and Discussion

12:00 **Adjourn**

DAQ / Trigger Simulation for HEP

PARTICIPANTS LISTING

LAST NAME	FIRST NAME	INSTITUTION
Becker - Szendy	R.	Stanford Linear Accelerator Center
Belanger	Ron	ErgoSoft
Bird	Fred	SSC Laboratory
Black	Dennis	Fermilab
Bogaerts	Andre	CERN
Booth	Alex	SSC Laboratory
Botlo	Mike	SSC Laboratory
Bowden	Mark	SSC Laboratory
Briggs	D.	Stanford Linear Accelerator Center
Dasu	Sridhara Rao	University of Wisconsin - Madison
Dingus	Peter	SSC Laboratory
Dorenbosch	Jheroen	SSC Laboratory
Ekenberg	Tor	University of Pennsylvania
Ferrer	Maria Lorenza	INFN-LNF
Forden	Geoffrey	University of Arizona
Fortner	Mike	Northern Illinois University
Frederiksen	Soren	SSC Laboratory
Grindley	Robin	University of Toronto
Halsall	Rob	Rutherford Appleton Laboratory
Holscher	Andreas	University of Toronto
Hughes	Eric	University of Illinois at Urbana-Champaign
Hunt	Steven	SSC Laboratory
Iyer	Raja	University of Texas at Arlington
Johnson	Marvin	Fermilab
Kapoor	Vishal	University of Texas at Arlington
Kasha	Henry	Yale University
Kouzes	Richard	Battelle Pacific Northwest Laboratory
Lackner	Klaus S.	Los Alamos National Laboratory

DAQ / Trigger Simulation for HEP

PARTICIPANTS LISTING

LAST NAME	FIRST NAME	INSTITUTION
Mathieson	Derek	SSC Laboratory
Milder	Andrew	Fermilab
Miller-Karlow	Diana	University of Illinois at Urbana-Champaign
Milner	Cas	SSC Laboratory
Nesic	Dusan	Brown University
Partridge	Richard	Brown University
Ragan	Kenneth	McGill University
Raj	Vijay	University of Texas at Arlington
Regan	Thomas	SSC Laboratory
Roe	Byron	SSC Laboratory
Sarkis	Joseph	University of Texas at Arlington
Shaevitz	Mike	Columbia University
Shao	Beibei	SSC Laboratory
Sullivan	Greg	University of Chicago
Thaler	Jon	University of Illinois at Urbana-Champaign
Tharakan	George	University of Illinois at Urbana-Champaign
Vanstraelen	Guy	SSC Laboratory
Vo	Ha	SSC Laboratory
Walsh	Donald	Fermilab
Wang	Ed	SSC Laboratory
Wang	Chung-Ching	SSC Laboratory
Wightman	Jay A.	Texas A & M University
Williams	Brig	University of Pennsylvania

```
!list of addresses of workshop attendees:
jnet%"atiya@bnldag"                          ! Maged Atiya, BNL
jnet%"barsotti@fnal"                         ! Ed Barsotti, FNAL
smtp%"bebek@csa.lbl.gov"                     ! Chris Bebek, LBL
smtp%"ronb@cacici.cts.com"                   ! Ron Belanger, ErgoSoft
smtp%"vanderby@vxcern.cern.ch"               ! Erik van der Bij, CERN
jnet%"dblack@fnal"                           ! Dennis Black, FNAL
jnet%"bogaerts@cernvax"                      ! Andreas Bogaerts, CERN
booth                                        ! Alex Booth, SSCL
smtp%"k.bos@nikhef.nl"                       ! Kors Bos, NIKHEF
botlo                                        ! Michi Botlo, SSCL
jnet%"Briggs@slacvm"                         ! Don Briggs, SLAC
jnet%"buchholz@cernvm"                       ! Peter Buchholz, CERN
jnet%"myron@fnal"                            ! Myron Campbell, U. Michigan
jnet%"Sergio_Cittolin%macmail@cernvm"        ! Sergio Cittolin, CERN
smtp%"bcourtn@dalvmic1.vnet.ibm.com"         ! Bill Courtney, ibm dallas
brhep1::cutts                                ! Dave Cutts, Brown U.
dorenbosch                                   ! Jheroen Dorenbosch, SSCL
jnet%"rwd@uiuchepa"                          ! Bob Downing, U. of Illinois
smtp%"ekenberg@upenn5.hep.upenn.edu"         ! Tor Ekenberg, U. of Pennsylvania
jnet%"ellisn@cernvm"                         ! Nick Ellis, U. of Birmingham
smtp%"ferrer@lnf.infn.it"                    ! Maria L. Ferrer, INFN
jnet%"foreman@lampf.lanl.gov"                ! Will Foreman, LAMPF
fnald0::fortner                              ! Mike Fortner, N. Illinois U.
fry                                          ! Alan Fry, SSCL
jnet%"gaines@fnal"                           ! Irwin Gaines, FNAL
smtp%"grindley@physics.utoronto.ca"          ! Robin Grindley, U. of Toronto
jnet%"dbg@slacvm"                            ! David B. Gustavson, SLAC
phyllis                                      ! Phyllis Hale, SSCL
jnet%"mjh@uiuchepg"                          ! Mike Haney, U. of Illinois
jnet%"hoehn@lampf.lanl.gov"                  ! Martha Hoehn, LANL
smtp%"holscher@physics.utoronto.ca"          ! Andreas Holscher, U. of Toronto
smtp%"Hughes@cs.uiuc.edu"                    ! Eric Hughes, U. of Illinois
jnet%"jaquez@fnal"                           ! Mari Jaquez, FNAL
jnet%"mjohnson@fnal"                         ! Marvin Johnson, FNAL
yalphy::kasha                                ! Henry Kasha, Yale
smtp%"rt_kouzes@ccmail.pnl.gov"              ! Dick Kouzes, Pacific Northwest Lab
jnet%"kozl@lampf.lanl.gov"                   ! Tom Kozlowski, LAMPF
jnet%"ksl@lanl.gov"                          ! Klaus Lackner, LANL
lankford                                     ! Andy Lankford, UC Irvine
vxcern::ledu                                 ! Patrick Ledu, Saclay
jnet%"let@cernvm"                            ! Mike Letheren, CERN
jnet%"levi@slacvm"                           ! Mike Levi, LBL
bnlhi0::levine                               ! Michael Levine, BNL
jnet%"cal@slacvm"                            ! Connie Logg, SLAC
fnald0::ianm                                 ! Ian Manning, FNAL
jnet%"mapelli@cernvm"                        ! Livio Mapelli, CERN
jnet%"marchior@cernvax"                      ! Alessandro Marchioro, CERN
jnet%"jym@slacvm"                            ! John Matthews, U. New Mexico
mcfarlan                                     ! Ken McFarlane, SSCL
milner                                       ! Cas Milner, SSCL
smtp%"mirabelli@roma1.infn.it"               ! G. Mirabelli, INFN
jnet%"muller@cernvax.cern.ch"                ! Hans Muller, CERN
brhep1::nesic                                ! Dusan Nesic, Brown U.
aptp                                         ! Andrea Palounek, LANL
fnal::partridge                              ! Richard Partridge, Brown U.
jnet%"patrick@fnald"                         ! Jim Patrick, FNAL
jnet%"ruth@fnal"                             ! Ruth Pordes, FNAL
smtp%"ragan@physics.mcgill.ca"               ! Ken Ragan, McGill U.
smtp%"raj@cse.uta.edu"                       ! Vijay K. Raj, U. of Texas, Arlington
jnet%"rimmer@cernvm"                         ! Peggy Rimmer, CERN
jnet%"ritchie@utaphy"                        ! Jack Ritchie, U. of Texas, Austin
roe                                          ! Byron Roe, U. of Michigan/SSCL
jnet%"hfws@slacvm"                           ! Hartmut Sadrozinski, U.C. Santa Cruz
jnet%"sandberg@lampf.lanl.gov"               ! Vern Sandberg, LAMPF
yalphy::schmidt                              ! Michael Schmidt, Yale
```

smtp%"ses@asd470.dseg.ti.com"	! Steve Schulz, Texas Instruments
jnet%"phs@ukacrl"	! Peter Sharp, Rutherford Lab
jnet%"schurecht@fnald"	! Kurt Schurecht, FNAL
smtp%"pekka@cepheid.physics.utoronto.ca"	! Pekka Sinervo, U. of Toronto
wishep::wsmith	! Wesley Smith, U. Wisconsin
smtp%"stairs@hokuto.yorku.ca"	! Gavin Stairs, U. of Toronto
jnet%"jjt@uiuchepa"	! Jon Thaler, U. of Illinois
smtp%"gth@eng3.hep.uiuc.edu"	! George Tharakan, U. of Illinois
jnet%"thooris@frsacl2.bitnet"	! Bruno Thooris, Saclay
jnet%"turner@fnald"	! Kathy Turner, FNAL
smtp%"vanberg@penndrls.upenn.edu"	! Rick Van Berg, U. of Pennsylvania
jnet%"walsh@fnal"	! Don Walsh, FNAL
jnet%"wightman@fnal"	! Jay Wightman, FNAL
smtp%"williams@penndrls.upenn.edu"	! Brig Williams, U. of Pennsylvania
smtp%"sandro@online.cern.ch"	! Sandro Vascotto, CERN
jnet%"ziock@lampf.lanl.gov"	! Hans Ziock, LANL

DAQSIM
A Data Acquisition System Simulation Tool

SSCL

A. W. Booth, M. Botlo, J. Dorenbosch,
R. Idate, E. C. Milner,
V. S. Kapoor, C. Wang, E. Wang

SSC Laboratory

V. Raj UTA

DAQSIM Talk Outline

SSCL

DAQSIM Features

Selection of a Simulation Language - MODSIM II

Simulation Objects - General and Specific

DCC - A Study, Push versus Pull

Results

Future Studies and Conclusions

DAQSIM Introduction

SSCL

At the SSC we have developed a tool for studying the behavior of Data Acquistion Systems

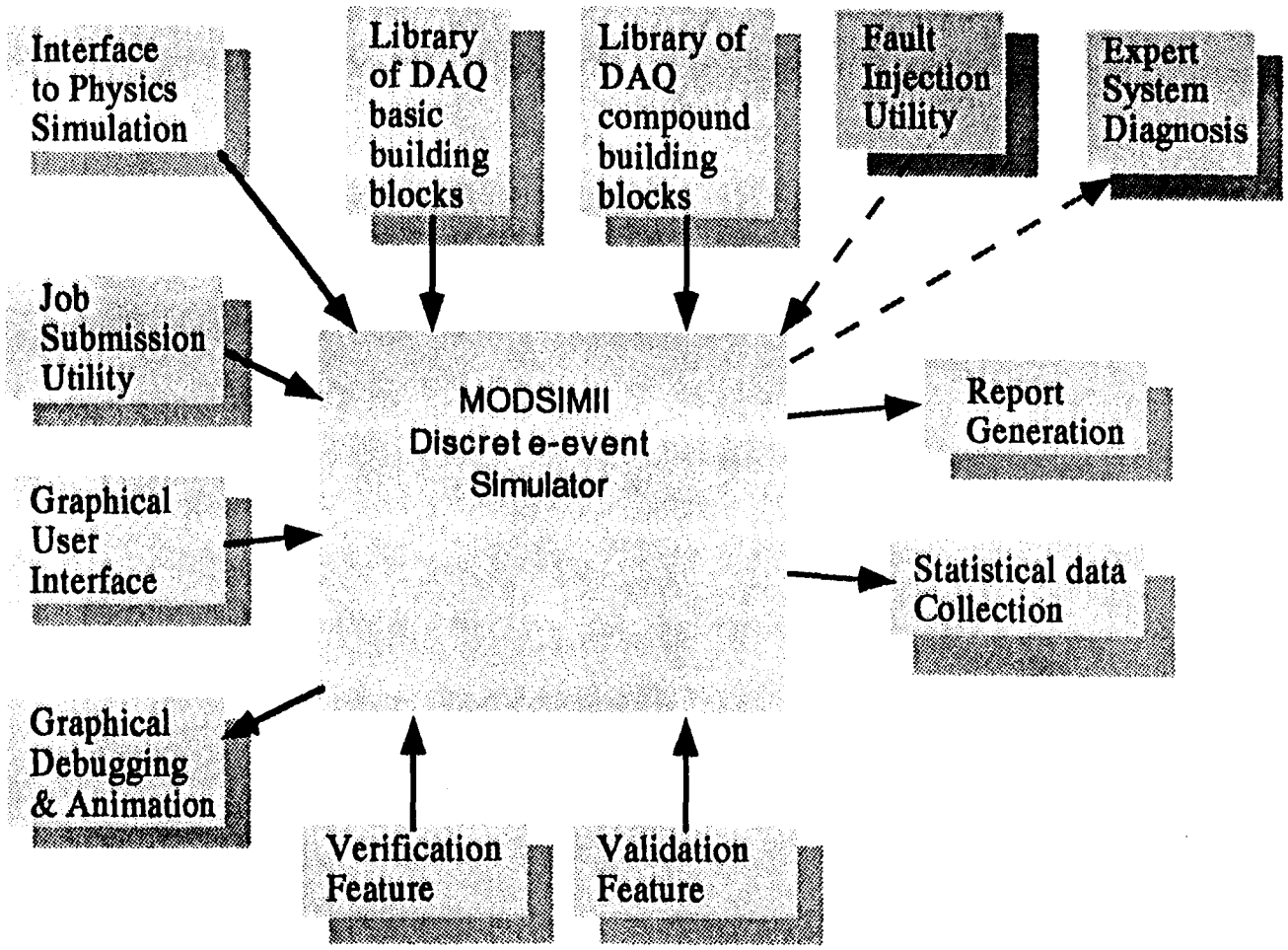
Based on the MODSIM II object-oriented programming language

Allows designers to evaluate alternative DAQ architectures in terms of deadtime, throughput and buffer usage, etc.

Specify dynamically the number of chips, buffer sizes, processing time, bandwidth of links, for various interconnection topologies

Graphical User Interface, Various types of input data

SSCL


Interface to Physics Simulation
Library of DAQ basic building blocks
Library of DAQ compound building blocks
Fault Injection Utility
Expert System Diagnosis
Job Submission Utility
MODSIMII Discrete-event Simulator
Report Generation
Graphical User Interface
Statistical data Collection
Graphical Debugging & Animation
Verification Feature
Validation Feature

Selection of a Simulation Language

SSCL

Most important step in developing a simulation model

Basically four choices that can be made:

- a simulation package, extension of a general-purpose language
- a simulation language, general-purpose language

We chose a simulation language for [redacted] the following reasons:-

It has built in facilities for
- advancing time, scheduling events
- manipulating objects, generating random numbers
- collecting statistical data ,generating reports

Less distractions on issues general to all simulations

Provides readable modular code with error detection capabilities

MODSIM II

SSCL

Object-oriented programming language

General purpose, modular

Block-structured high-level programming language

Discrete-event simulation

Used to build large process-based simulation models

Simulation Objects

SSCL

General simulation objects

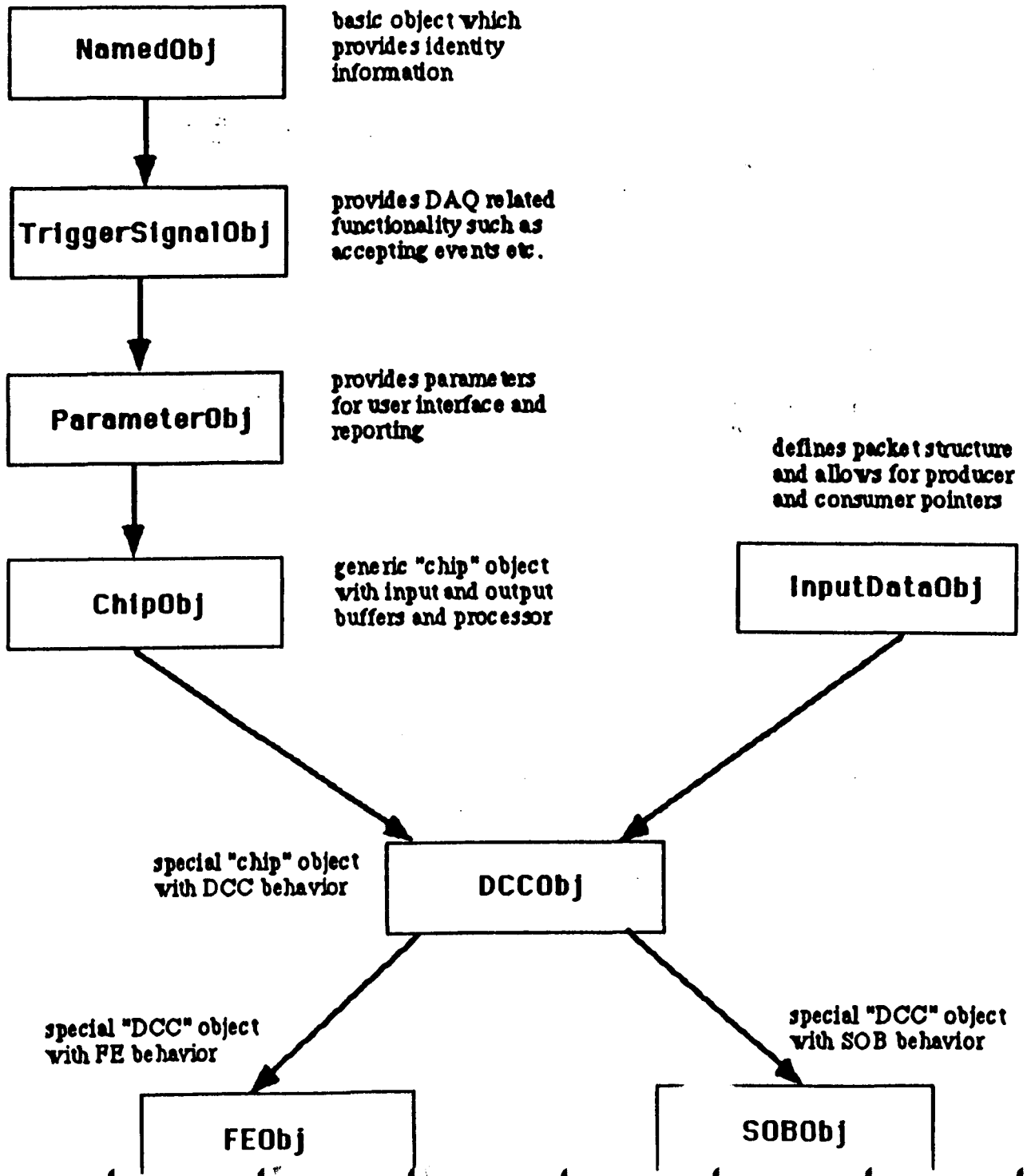
- NamedObj
- ParameterObj

General DAQ objects

- StandardActionObj
- TriggerSignalObj
- ChipObj
- InputDataObj

Specific DAQ objects

- DCCObj
- FEObj
- SOBObj


NamedObj
basic object which provides identity information
TriggerSignalObj
provides DAQ related functionality such as accepting events etc.
ParameterObj
provides parameters for user interface and reporting
ChipObj
generic "chip" object with input and output buffers and processor
defines packet structure and allows for producer and consumer pointers
InputDataObj
special "chip" object with DCC behavior
DCCObj
special "DCC" object with FE behavior
special "DCC" object with SOB behavior
FEObj
SOBObj

SSCL

Data Collection Circuit (DCC) - A Study

SSCL

Clear Statement of Simulation Goals

- To determine the throughput, deadtime and efficiency of different DAQ architectures
- To identify potential bottlenecks by varying buffer sizes, link transmission speeds, etc
- To compare different data flow control strategies, specifically push and pull and make recommendations about their use
- To produce a model of a DCC network, including any algorithms, which could be used as the basis for an implementation to readout an SCC calorimeter for example
- To provide feedback to institutions which are involved in DCC development in terms of the impact of design decisions such as whether or not "to order the data", or "to throttle the trigger(s)", or "when to perform zero suppression"

DCC - A Study

SSCL

Other DAQ Questions

- Do our results depend on assumptions made about triggering, and if so how adaptable is it to alternative triggering strategies ?
- Do we zero-suppress or not ?
- What data transportation to the second level trigger ?
- How much processing can be done on line and what is the effect on throughput of doing such processing ?
- How to deal with errors and adverse conditions, can we make some statements about if/when to "drop" bad events ?
- What are the implications in terms of technology, i.e. what technologies make an architecture unreasonable (in terms of cost, complexity)
- To which proposed real detectors could our studies be applied ?
- Are there some approaches that lend themselves better to reliability (and redundancy) than others, what are the trade-offs ?
- What are the implications for interfacing to an event-building switch, or to the scalable coherent interface

DCC - A Study

SSCL

Why DCC ?

Push

- Nocheck Push
- Spacecheck Push
- Imaging Push

Pull

- Pull In Order
- Pull When Ready

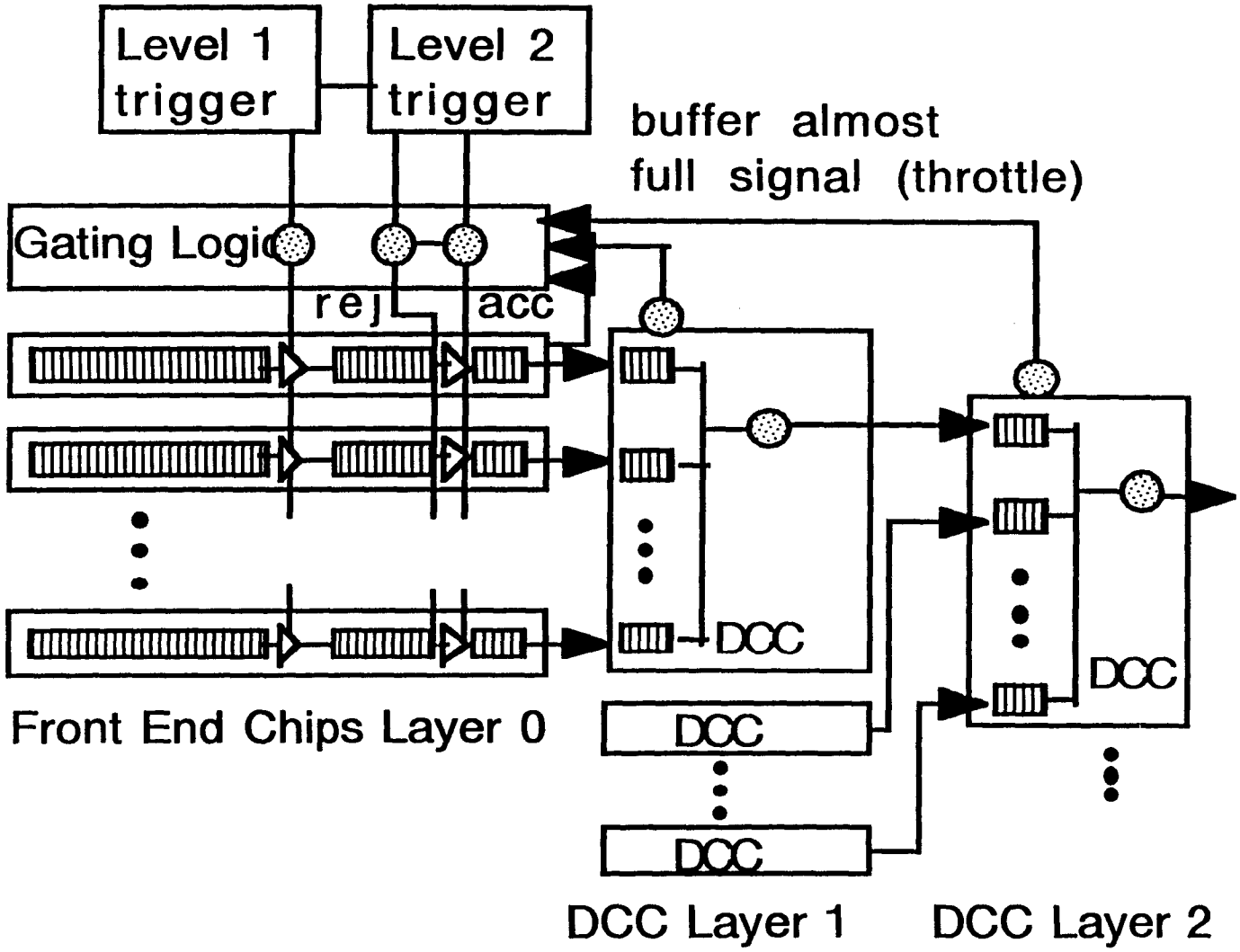
Processing functions

- checking level2 id's and wordcounts
- checking error-flags
- stripping off headers, adding new headers
- manipulation pointers, imaging consumer buffer
- running special algorithms

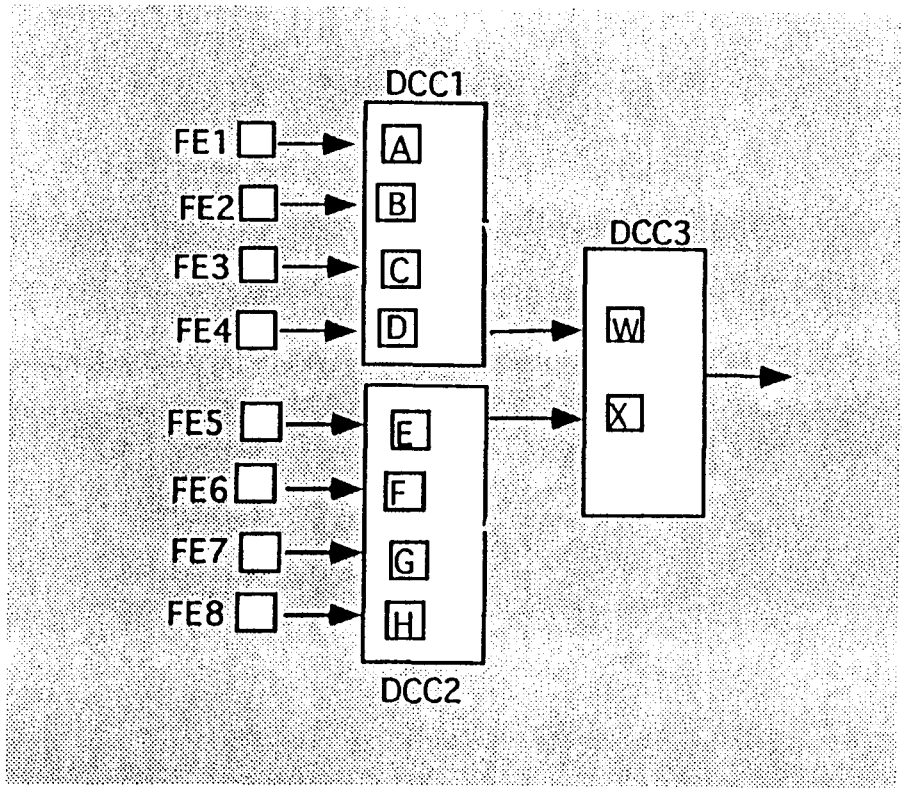
Error Conditions

- Dropping Packets
- Losing a Link

DAQ System

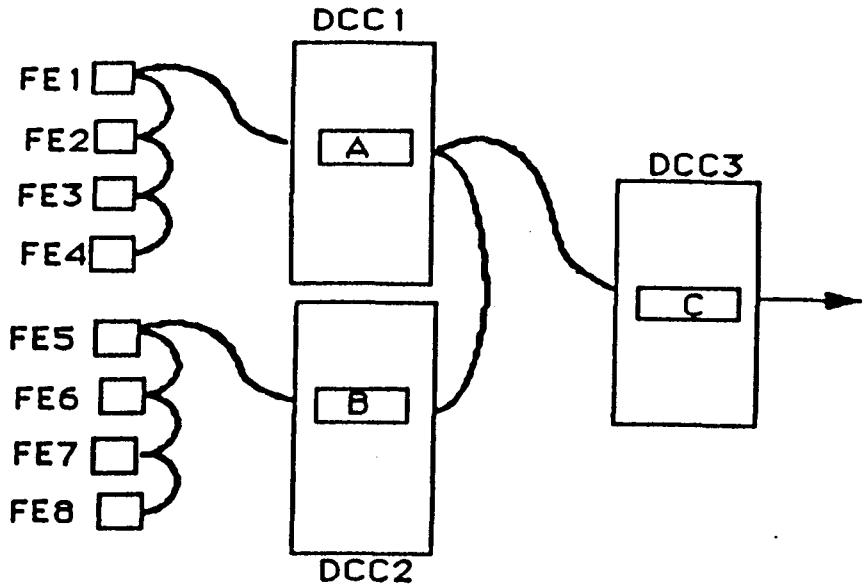


SSCL



SSCL

SSCL



Different Kinds of Input Data

SSCL

Fixed-size data

Data According to an Exponential Distribution

Data According to an Exponential Distribution and with correlations applied

QCD and Electron Physics data extracted from GEANT Simulation

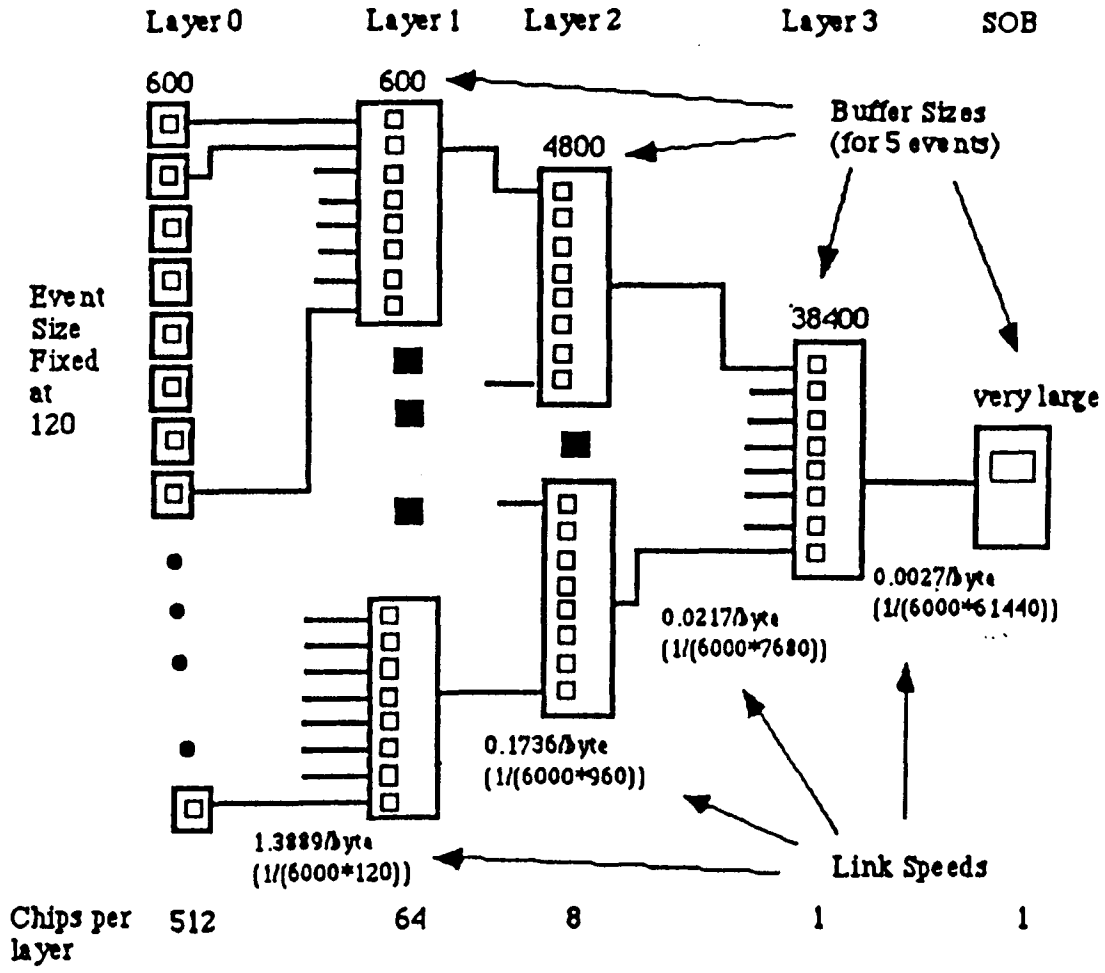
Simulation of DAQ Architectures

SSCL

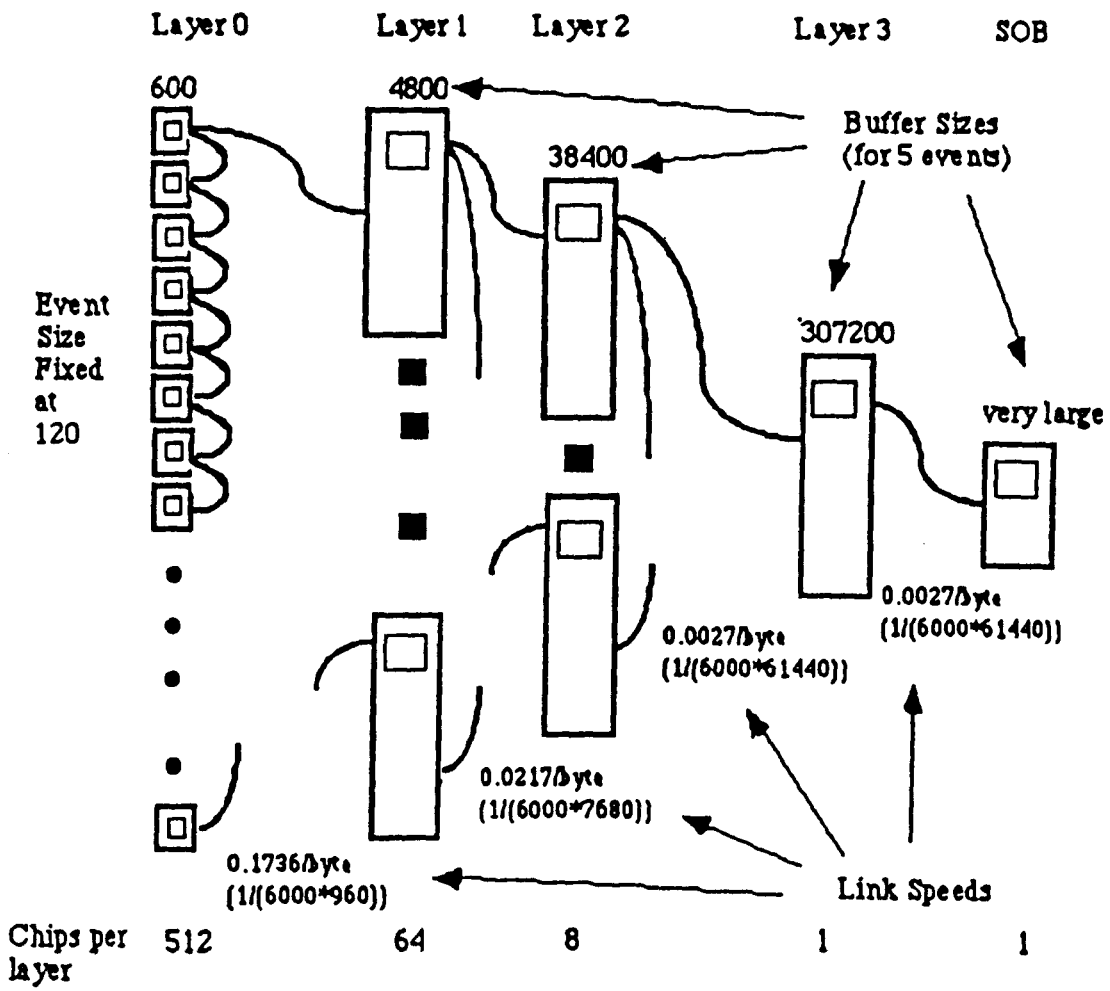
Architectures

- Push, Pull & Mixed
- Large & Small
- Varied the number of Layers
- Throttle or Nothrottle
- Varied the buffer sizes and link speeds

LARGE PUSH DCC NETWORK

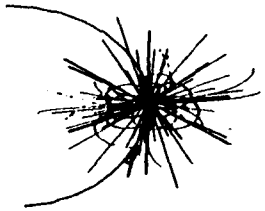


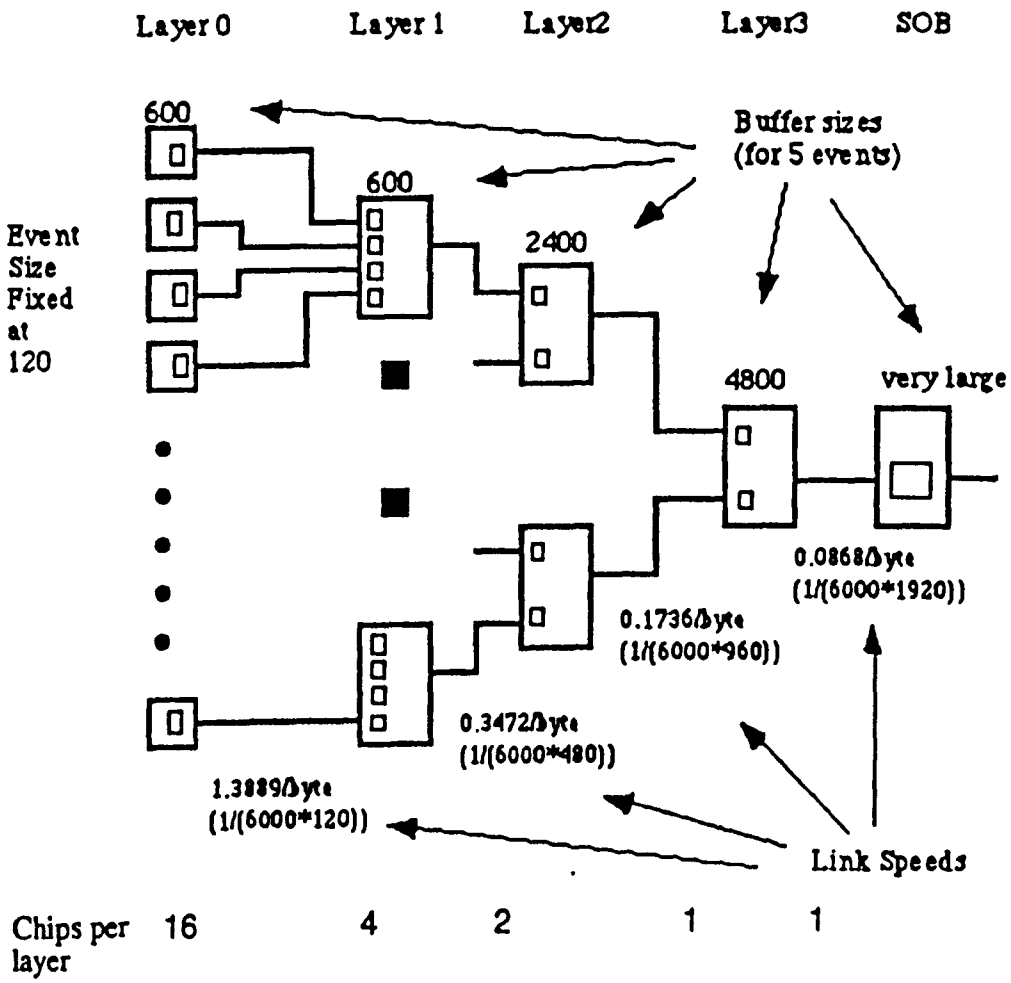
LARGE PULL DCC NETWORK



SSCL

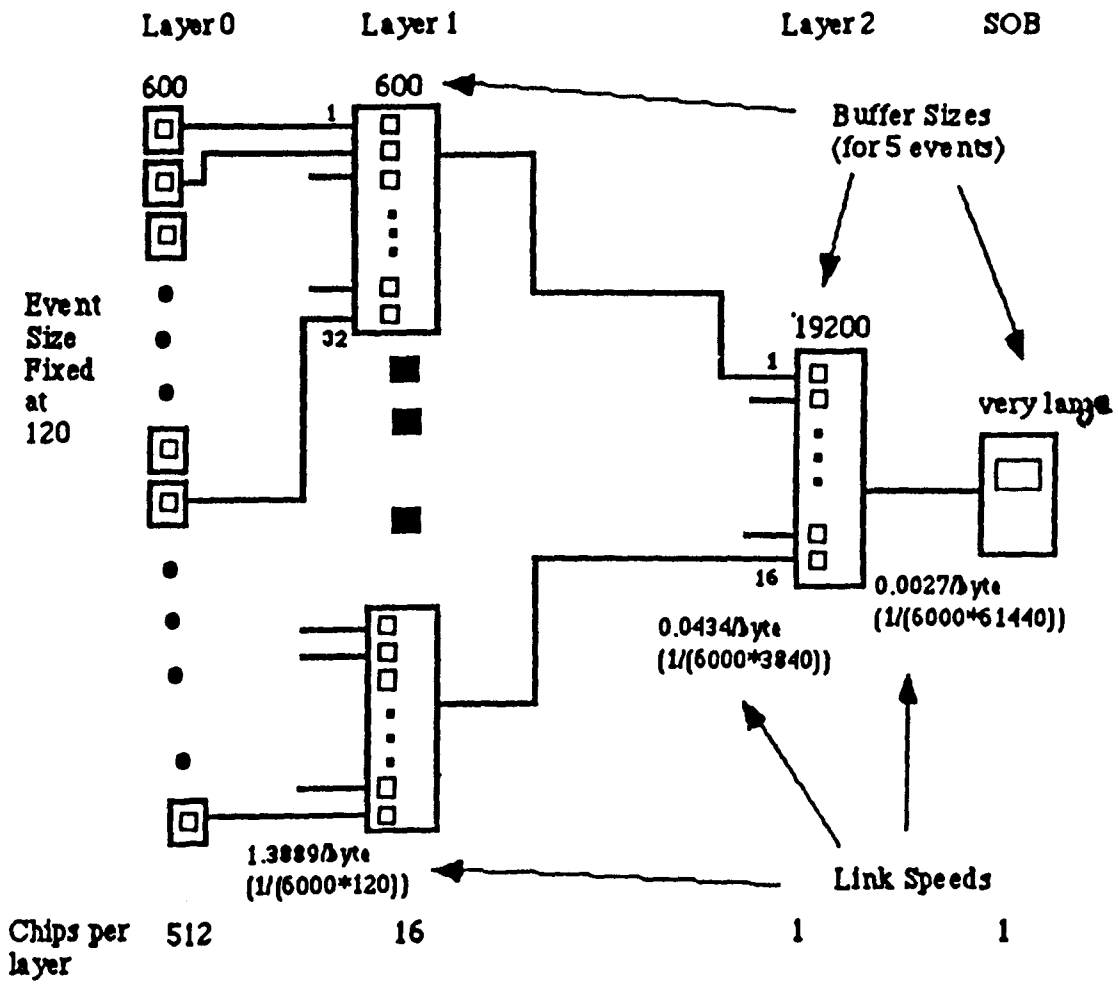
SMALL PUSH DCC NETWORK





LARGE PUSH DCC NETWORK WITH ONLY TWO LAYERS

SSCL



Simulation Run Parameters and Assumptions

SSCL

Assumptions

Front-ends could provide a wordcount and a level2 id

DCC contained processing ability to perform the functions described in this talk

The systems were indeed warmed up after 100 events

Static Parameters

Each run was for 1100 events, first 100 warm-up

Transmissions links between layers were set up to have 1 erlang at 6KHz of accepted level 2 triggers

Variable Parameters

Level 2 rates, # of FE's, # of DCC's, buffers sizes, # of layers, input data type,

Interactive parameters

SSCL

Simulations are done over a wide range of the values that describe system architectures and running conditions.

ParameterObj allows the values of variables to be changed interactively during run time. It provides default values, initialization procedures and descriptions of the variables

Each time a parameter value is changed, a procedure is called to update all variables that must be kept consistent with that value

It is possible to print the current values of all parameters of a model. (e.g. done in batch runs to document the simulation conditions)

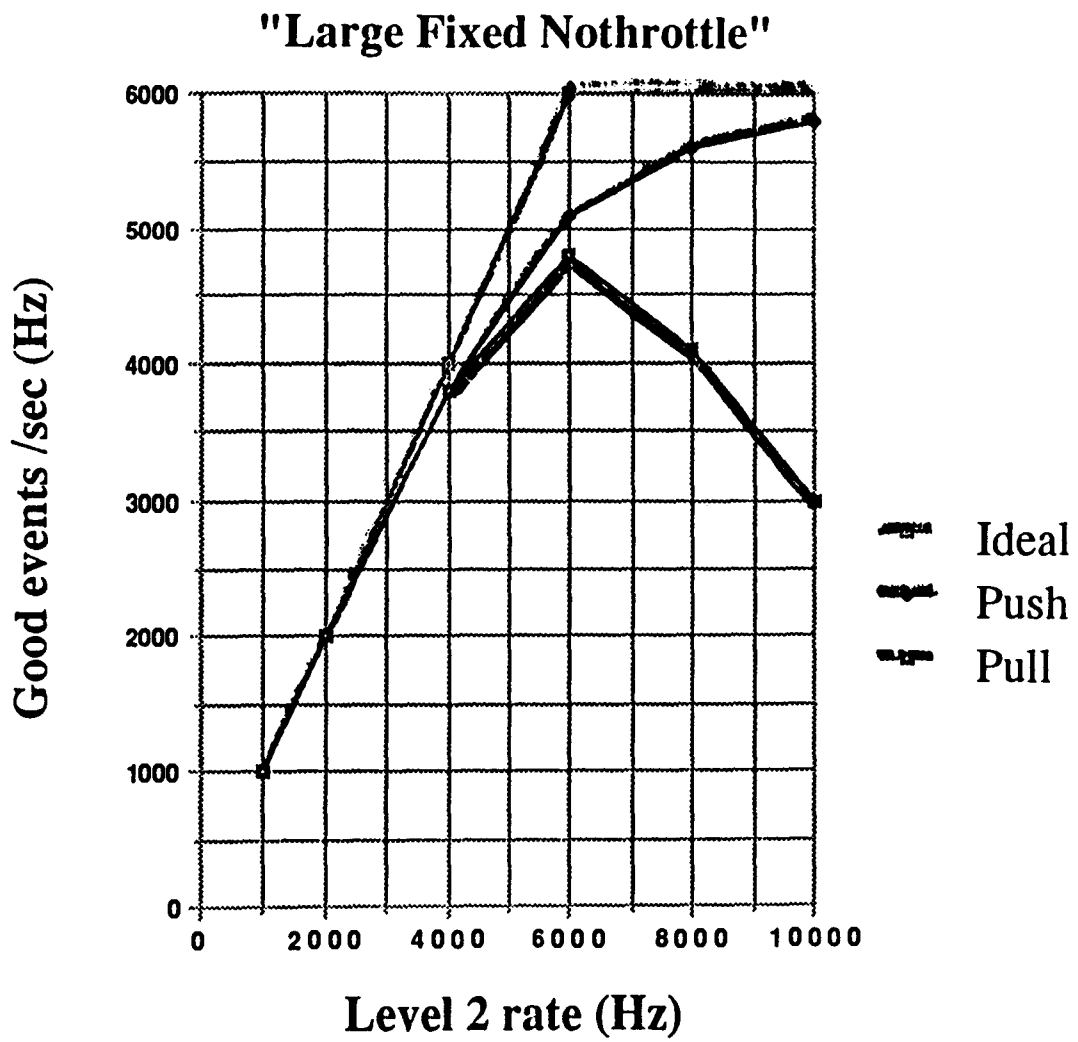
Parameter values can also be dumped to a file, which can be used for initialization of future runs.

List of Simulation Runs

SSCL

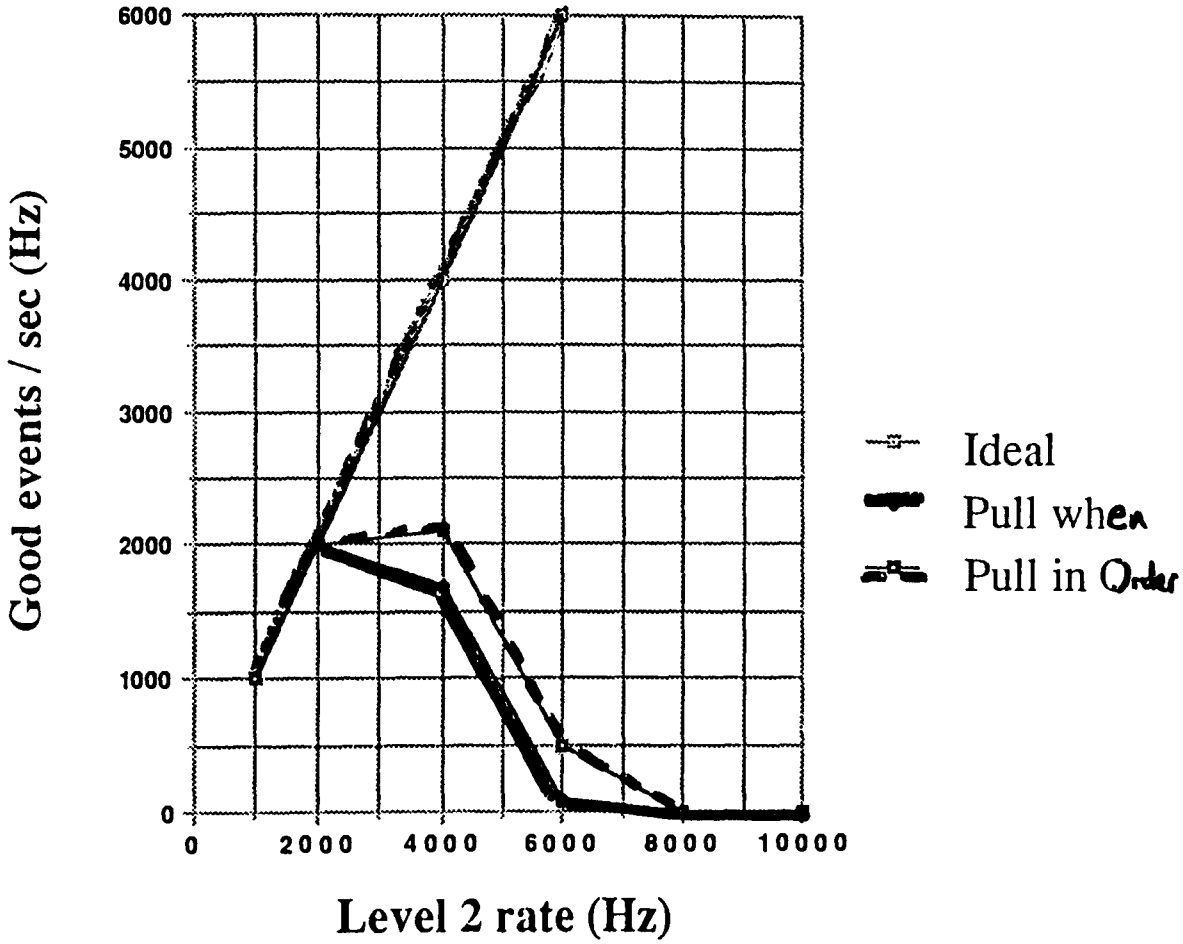
Small Fixed Data Runs	(100-105)
Small QCD100 Data Runs	(106-111)
Small Exponential Dist. Runs	(112-117)
Small Electron Data Runs	(130-135)
Large Fixed Data Runs	(200-205)
Large QCD100 Data Runs	(206-211)
Large Exponential Dist. Runs	(212-217)
Large Correlation Run	(218-223)
Large Electron Data Rus	(230-235)
Reducing the number of Buffers	(400-504)
Reducing the bandwidth out of Layer 3	(600-610)
Reducing the number of Layers	(800-)
Zero-suppression in Layer 1	(900-)

SSCL

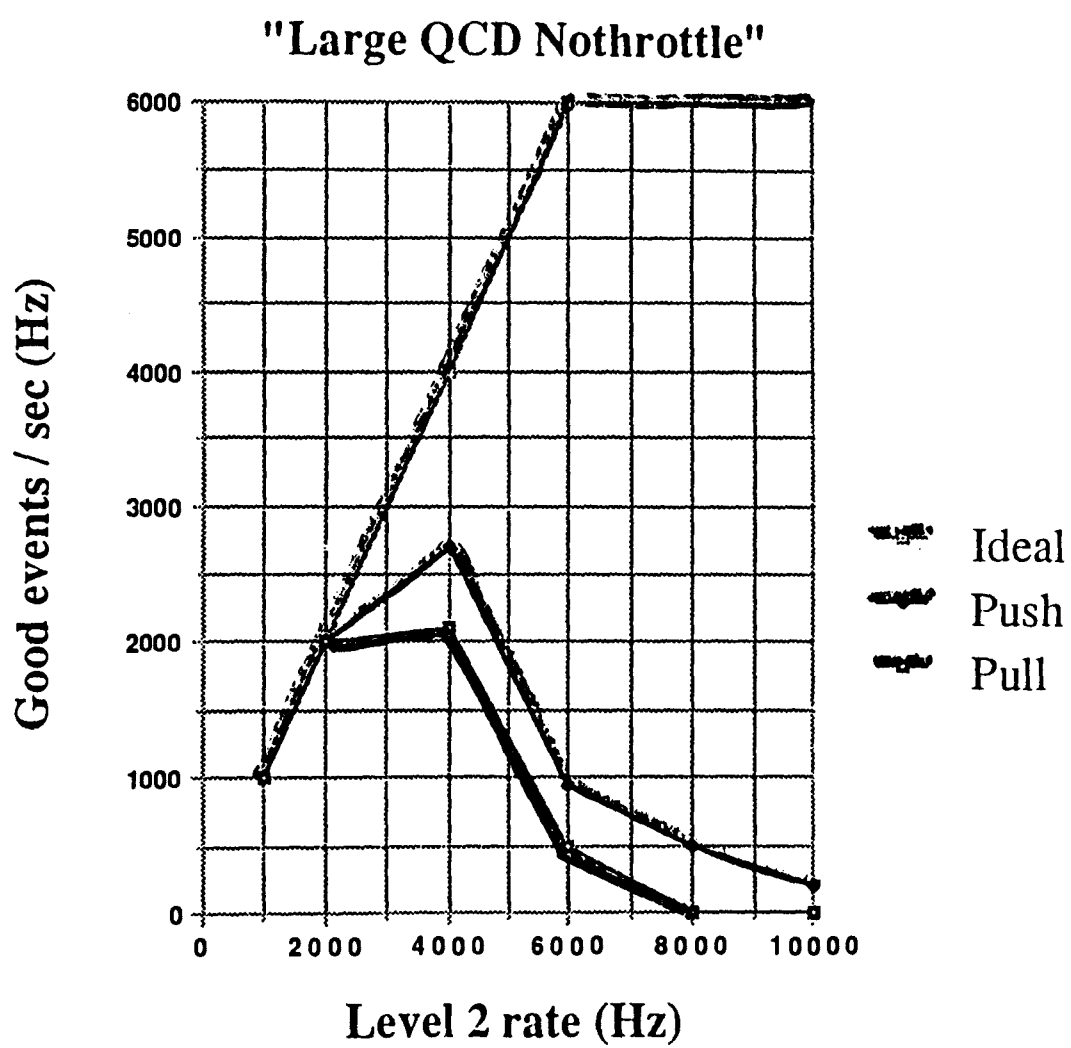

"Large Fixed Nothrottle"
Good events /sec (Hz)
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
Level 2 rate (Hz)
Ideal
Push
Pull

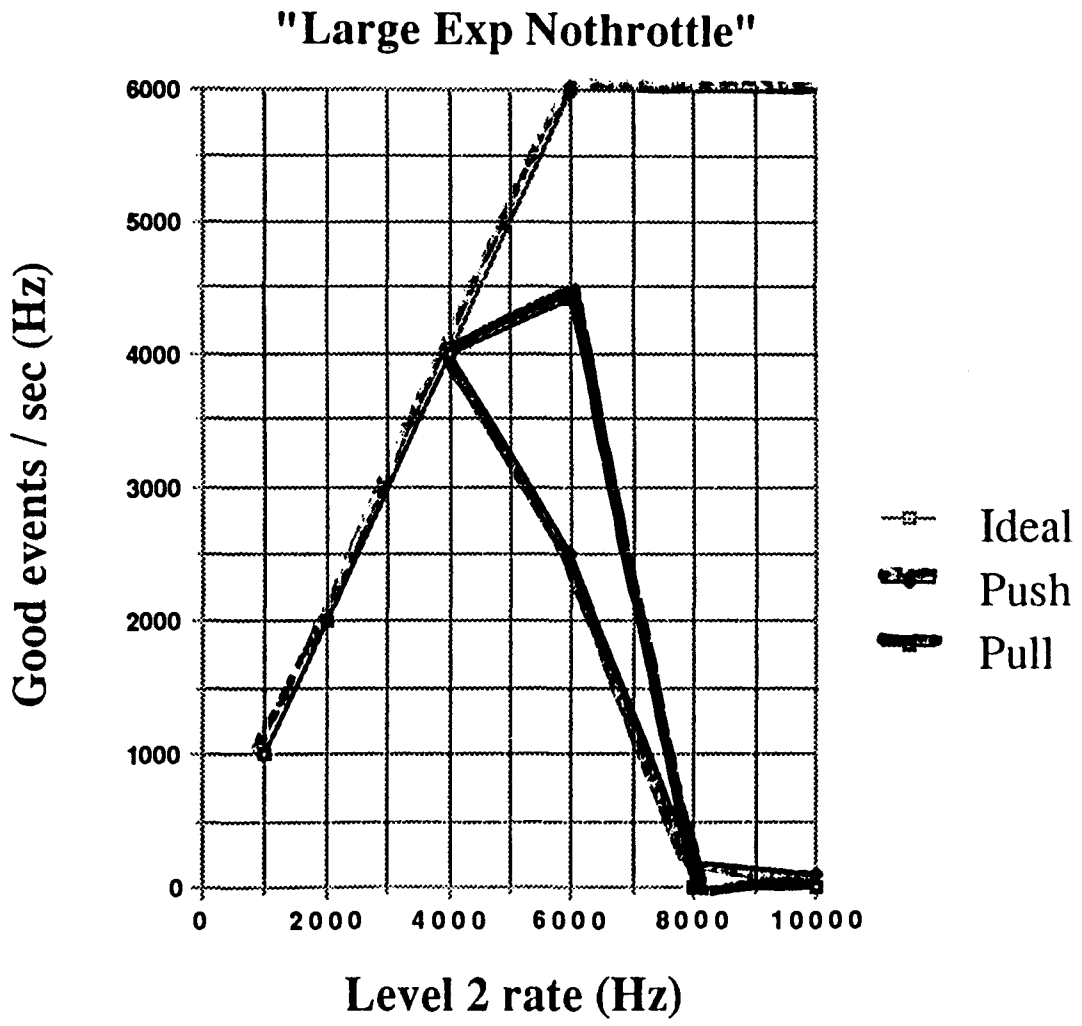
SSCL

Comparison of different "Pull" strategies



SSCL

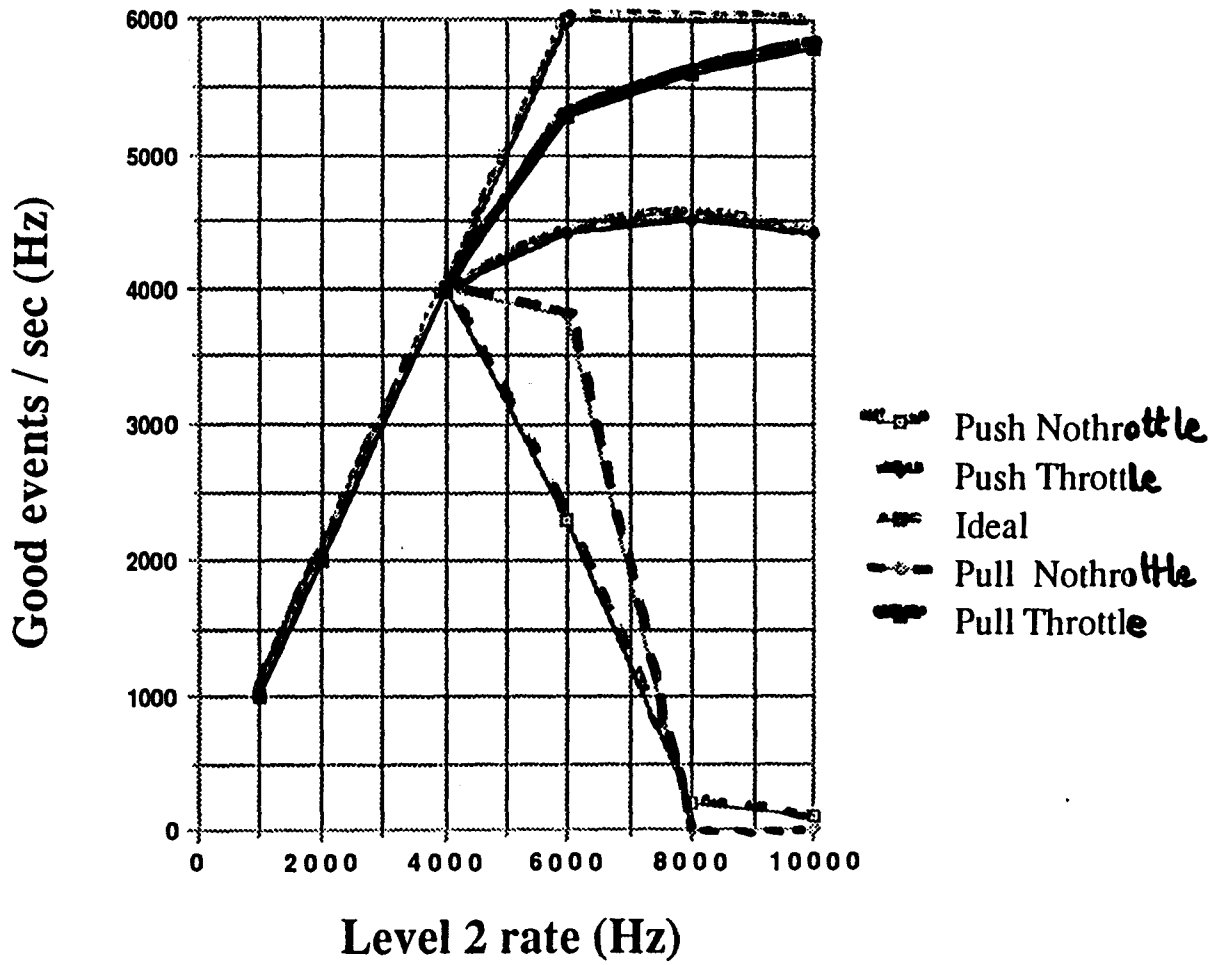

"Large QCD Nothrottle"
Good events / sec (Hz)
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
Level 2 rate (Hz)
Ideal
Push
Pull


"Large Exp Nothrottle"
Good events / sec (Hz)
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
Level 2 rate (Hz)
Ideal
Push
Pull

SSCL

SSCL

Comparison of Throttle/No throttle



Results - Interesting Observations

SSCL

- Good balancing of the data load over the data collection channels is essential.
- Data collection schemes that use throttling to limit buffer overflows, show a much more graceful degradation at high rates than systems that handle high rates by discarding part of the data.
- There is little difference in performance between 2 and 3 layers of DCC's
- Pull is somewhat better than push in throttled systems with non-fixed data sizes
- Under normal running conditions the Imagecheck protocol does not perform significantly better than the simpler SpaceCheck protocol.
- The buffer sizes in the deeper layers of the system are larger than required for smooth running.
- Performing 'zero suppression' in the first data collection layer rather than in the front-ends has little impact on throughput, provided that sufficient bandwidth is available for data transport to layer 1.

Future Studies

SSCL

Using DAQSIM for Proposed "Real" Detectors

Research into Architectures which Incorporate Redundancy

Refining some of the models

Provide Animation

Integrate Expert System and Fault Insertion Utility

Gee Whizz Slide

SSCL

Number of lines of MODSIM code = 12,000

Number of objects = 50

Number of cpu hours per run = between 4 - 7 hours
(one point on a plot)

Number of runs so far = 1000

Using one SparcStation II = 200 days continuous running

However, we have 20 workstations in our group and 30 workstations as front-ends to the physics detector simulation facility (pdsf)

Conclusions

SSCL

- DAQSIM - A tool for the understanding Data Acquisition Systems
- DAQSIM used to study the behavior of a data collection circuit
- DAQSIM has been used to compare push andpull control strategies
- Graphs showing deadtime, throughput and buffer usage plotted
- The study of DAQ systems is complex - has generated two Master's Theses at the University of Texas at Arlington
- DAQSIM can be used to understand "real" SSC detector readout systems

Front End and DCC Simulations for the SDC Straw Tube System

A. Hölscher, G. Stairs and P. K. Sinervo
University of Toronto
Toronto, Canada

April 21, 1992

contact: A.Hölscher
e -mail: holscher@cerebus.physics.utoronto.ca

This document is a collection of three notes, which describe the buffer and bandwidth requirements at various stages of the SDC straw tracker front end system.

1 Simulations of Front End Board Occupancies for the SDC Straw Tube Tracker

This note describes the results of a study of the front end board occupancies for the different superlayers of the SDC straw tracker.

2 Front End Buffer Requirements for the SDC Straw Tube Tracker

This note describes estimates made of the buffer requirements for the L1 and L2 buffer.

3 Buffer Occupancies for the SDC Straw Tube DCC System

This note describes a study of the buffer and bandwidth requirements on the front end boards and at the DCC level.

Simulation of Front End Board Occupancies for the SDC Straw Tube Tracker

A. Hölscher, P. K. Sinervo and G. Stairs
University of Toronto
Toronto, Canada

Penny Estabrooks
Carlton University
Ottawa, Canada

March 11, 1992

Abstract

In this note we present simulations of the occupancies of the Front end boards for the straw system using SDCSIM generated events.

1 Introduction

The straw system consists of about 110,000 straws, which are arranged in 5 superlayers. For an exact description of the geometrical arrangement we refer to [1]. In this note we determine occupancies of different layers and of the electronic front end boards. We use physics events generated by the SDC simulation program. No attempt has been made to generate electronic noise.

2 Superlayer occupancies

First we show the occupancies of different event types alone. However, at the nominal luminosity of the SSC of $L = 10^{33} cm^{-2} s^{-1}$, there are 1.6 minimum bias events per crossing. For the straw system the detector integrates over three bunch crossings per each trigger in order to catch all signals. This implies that for each physics events one triggers on, one has roughly 5 underlying minimum bias events, which of course add to the occupancies. We

generated 5 minimum bias events, 5 events containing a Higgs decaying in the following decay chain: Higgs $\rightarrow$ ZZ $\rightarrow 4\mu$ and 5 dijet events with a p_t in the range of 1 - 100 GeV/c. Table 1 shows the occupancy per channel per trigger, first for the different event types alone and then for the event types of interest plus the underlying 5 minimum bias events. Figure 1 shows these distributions for the higgs + minbias, dijet + minbias and 1 minbias cases represented graphically.

	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
1 minbias	0.83 %	0.35 %	0.28 %	0.16 %	0.16 %
1 Higgs	6.0 %	3.0 %	1.9 %	1.75 %	1.75 %
1 dijet	2.7 %	1.3 %	0.85 %	0.64 %	0.63 %
1 Higgs + 5 minbias	10.0 %	4.7 %	3.3 %	2.5 %	2.5 %
1 dijet + 5 minbias	6.8 %	3.1 %	2.3 %	1.4 %	1.4 %

Table 1: Layer occupancies for different types of events

3 Front end board occupancies

We are also interested in correlations in specific Front end boards. As shown in Figure 2 each superlayer consist of 6 or 8 layers, which are bundeled together into modules and read out by one Front end board (FEB). It is assumed that each front end board contains 160 straw channels. This implies that there are about 850 FEBs for the whole straw system. Each FEB consists of five so called FMUX's, each reading out 32 channels [2]. Therefore there will be about 4250 FMUX's in the whole straw system. The FMUX's are assumed to be radially arranged.

A particle transversing one superlayer will leave several hits in the particular front end board. Figure 3 and 4 [1] show the occupancies of the FMUX's for 1 minbias event and 1 dijet event alone. One clearly sees peaks at an occupancy of 6, which indicates that the transversing particle has a hit in all of the layers of one superlayer. These 6 hits mostly belong to one FEB, which is why there is a clear peak in the occupancy distribution of the FEB. The 6 hits may also belong to one FMUX, but here the correlation is not so strong. In comparison we show the occupancy distribution assuming that the modules would be arranged in the ϕ direction, where one does not have this correlation effect. Many more

[1] Note that the mean of the occupancy distributions was only calculated for FEB's or FMUX's, which contain at least one hit.

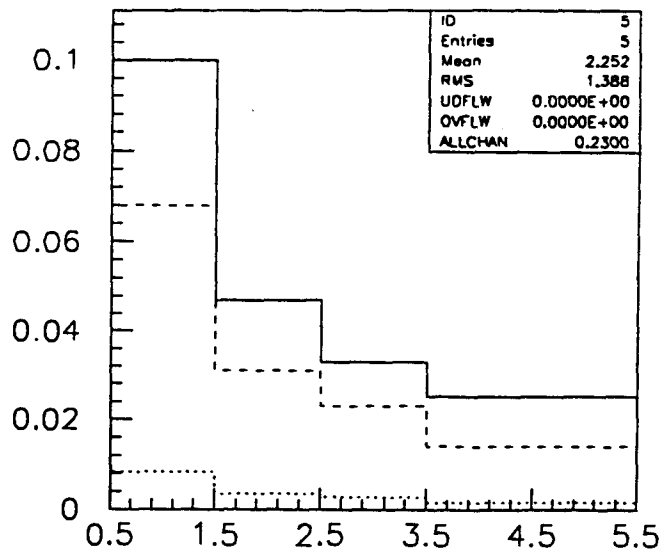


Figure 1: Occupancies of the different superlayers of the straw system for a higgs event (solid histogram) and a dijet event (dashed histogram). These events include 5 underlying minimum bias events. For comparison the occupancy for one minimum bias event (dotted histogram) is shown.

modules contain a hit, but the average occupancy of a hit module or FMUX is much smaller. Note that the average FEB and FMUX occupancies are very similar for the minbias and dijet events. However, in the minbias case fewer modules actually have one or more hits. Figure 5 finally shows the occupancy distribution of the Higgs or dijet events plus 5 underlying minbias events. The shapes don't change very much since the additional particles from the minbias events mostly transverse other modules. This is also illustrated in table 2, which shows the percentage of FMUX's or front end boards that don't contain any hit for all the different events categories. Including the minbias events clearly decreases the number of empty FMUX's and FEB's, implying that they hit different modules. We remind the reader however, that we didn't generate any electronic noise, which might change this picture.

The occupancy distributions, however, show long tails. For the generated 5 events of each category an occupancy of 105 for one FEB was observed and an occupancy of 41 for one FMUX.

	empty FEB's	empty FMUX's
1 minbias	95 %	98 %
1 higgs	68 %	85 %
1 dijet	84 %	93 %
1 higgs + 5 minbias	55 %	77 %
1 dijet + 5 minbias	66 %	84 %

Table 2: Fraction of FEB boards and FMUX's that do not contain any hits.

4 Conclusions

The maximal inner superlayer occupancy was found to be 10%. This translates into the following average datarates from the front end boards, assuming that each hit contains 4 bytes and an L2 trigger rate of 10 kHz and assuming that each front end board contains 160 straws. Again this table ignores the electronic noise. These datarates, however, are

Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
0.16 hits/μs	0.08 hits/μs	0.05 hits/μs	0.04 hits/μs	0.04 hits/μs
0.64 Mbyte/s	0.32 Mbyte/s	0.2 Mbyte/s	0.16 Mbyte/s	0.16 Mbyte/s

Table 3: Datarate from one FEB for the Higgs + 5 minbias events, assuming 4 bytes/hit and excluding electronic noise.

the average datarates. For any event the hits are correlated. When a module is hit by a transversing particle, it usually contains at least 6 hits. It can contain many more hits, when it is hit by a jet. In most cases, however, the Front end boards are empty.

References

[1] SDC detector notes, SDC-91-125, SDC modular straw outer tracking system conceptual design report.

[2] G.Stairs et al., *Front end Collector Bus Design Description*, UofT-DCC-03, Jan. 1992 (unpublished).

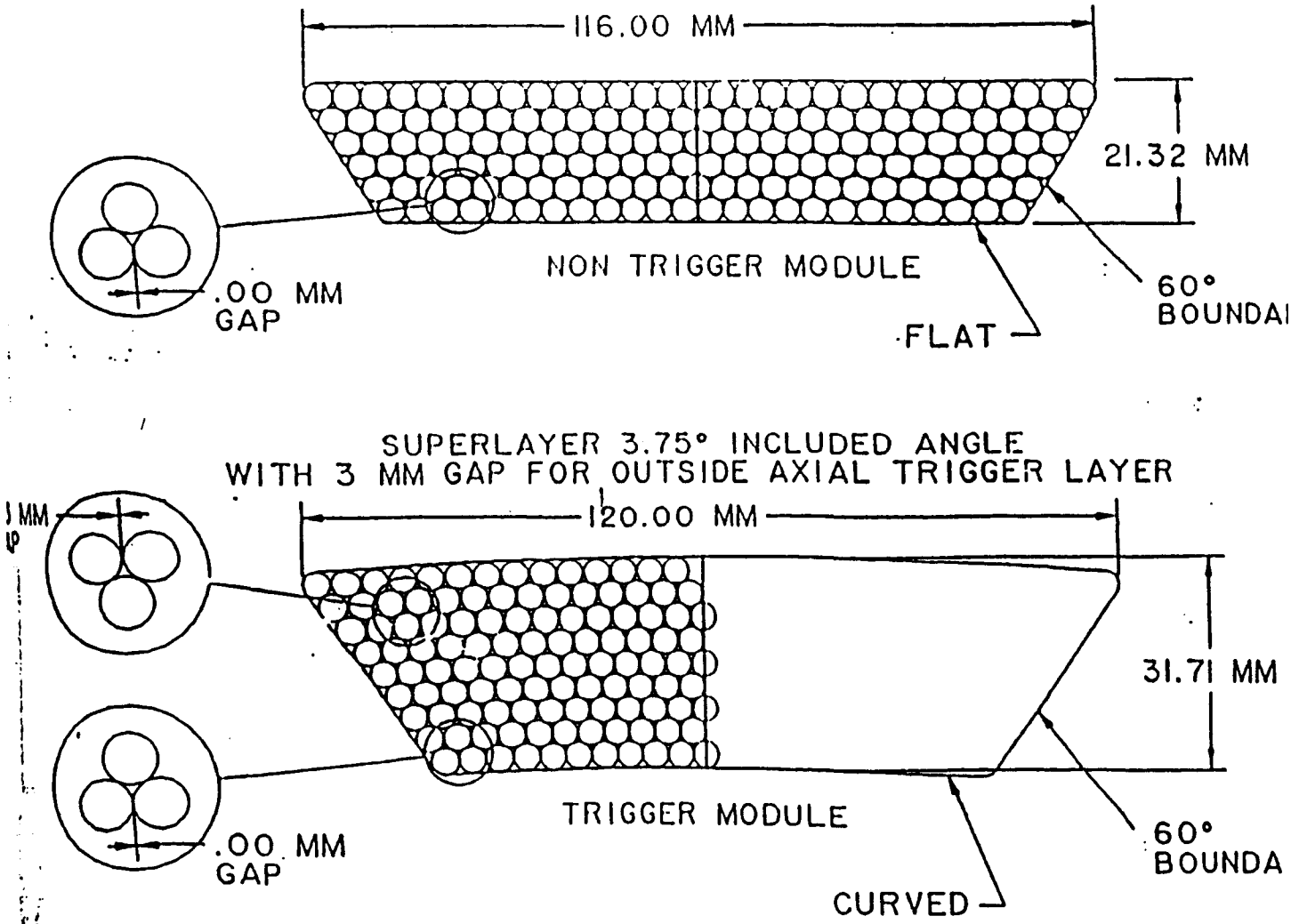


Fig. IV.2. The basic module design is shown here schematically. The two basic designs are shown; a 6 layer module for non-trigger layers and stereo layers, and a 9 layer trigger module that has all straws on a radial line. The present design will probably use an 8 layer trigger module.

Figure 2: Module design within the superlayers. For details see [1].

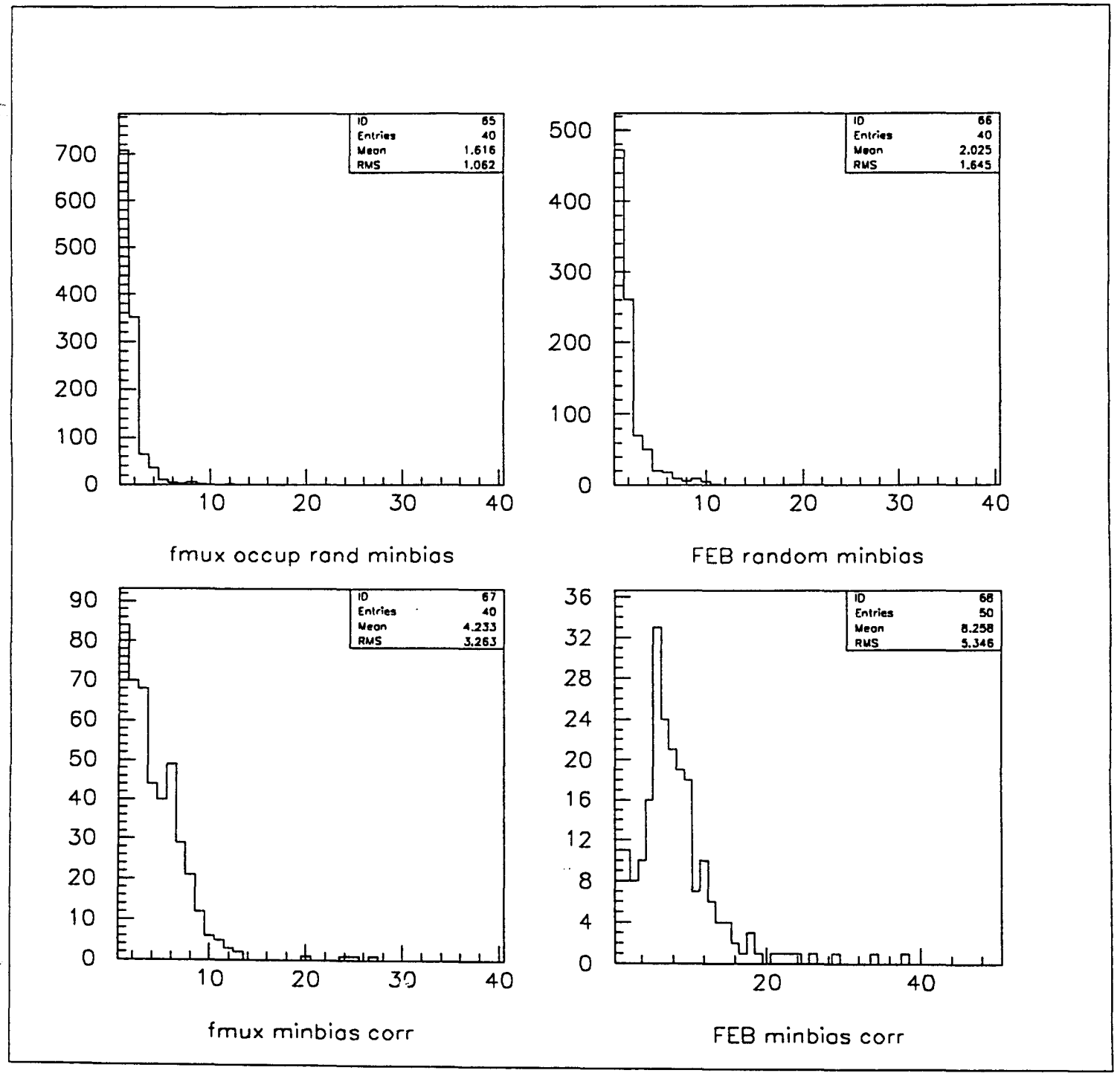


Figure 3: FEB and FMUX occupancies for one minimum bias event. The upper two plots show the modules arranged in ϕ, the lower two like in the real design.

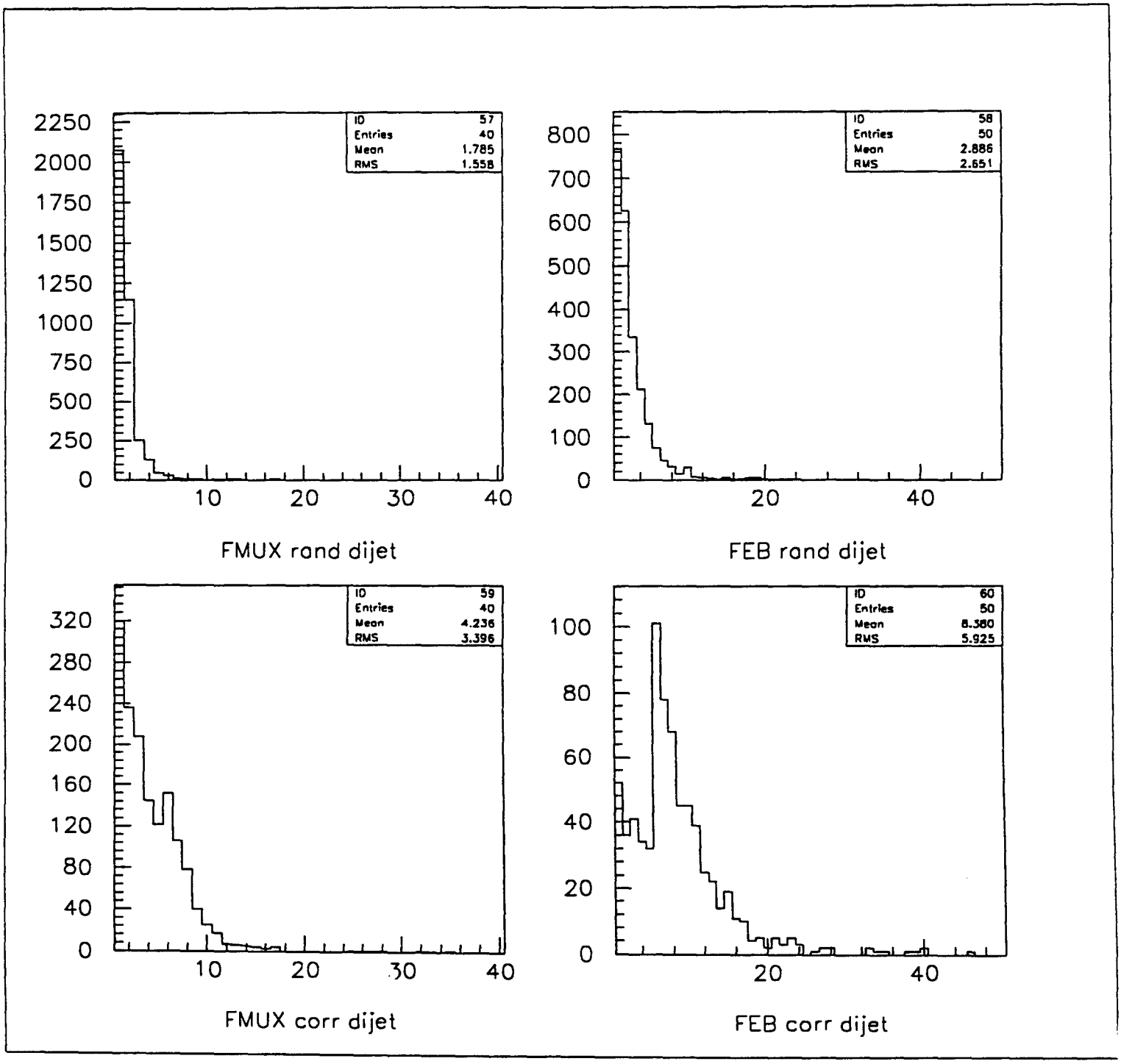


Figure 4: FEB and FMUX occupancies for a single dijet event without any underlying minimum bias events.

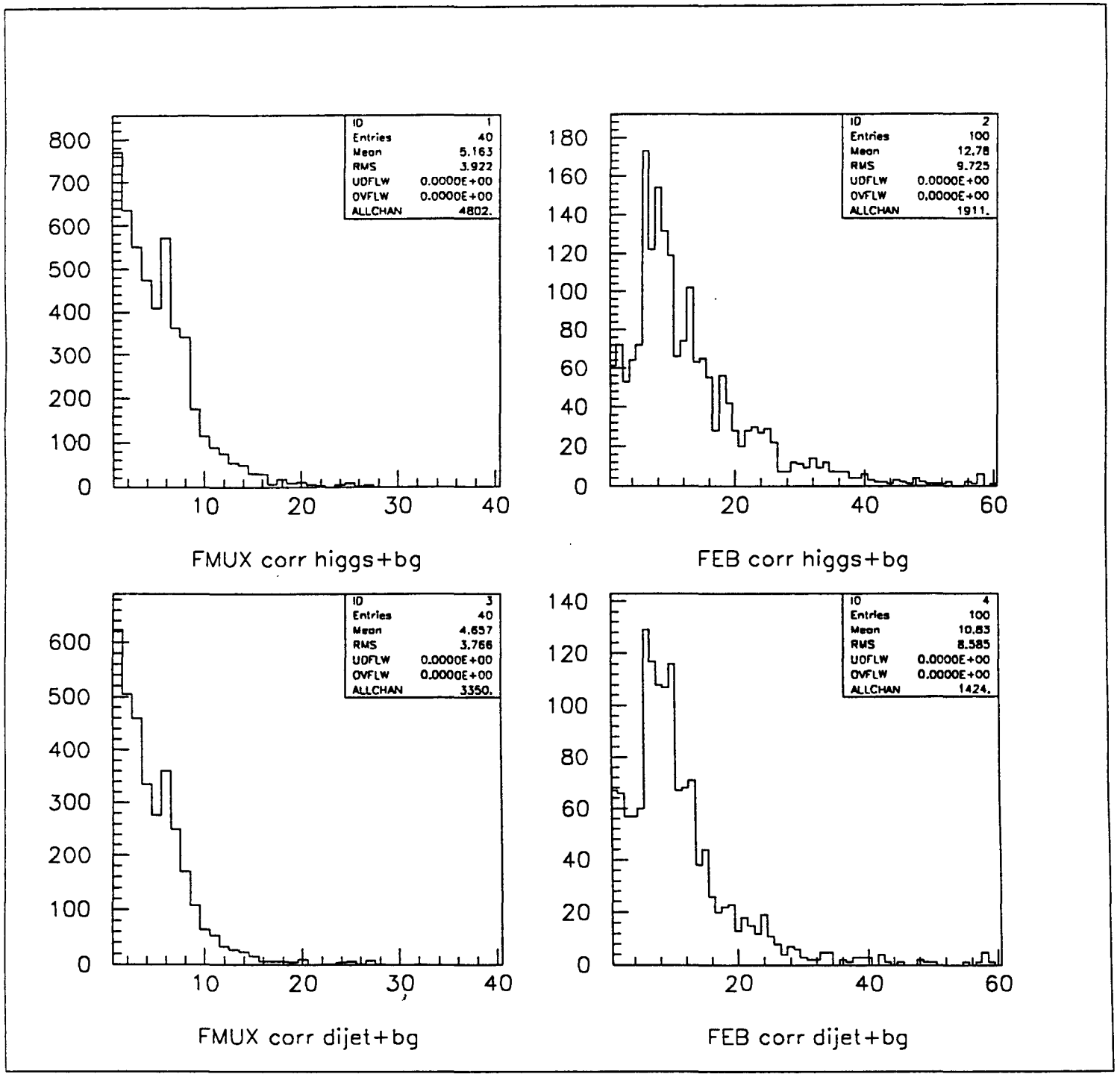


Figure 5: FEB and FMUX occupancies for Higgs events (upper two histograms) and dijet events (lower two histograms). Both events contain 5 underlying minimum bias events.

Front End Buffer Requirements for the SDC Straw Tube Tracker

A. Hölscher, G. Stairs and P. K. Sinervo
University of Toronto
Toronto, Canada

March 16, 1992

Abstract

In this note we present the results of calculations of the buffer occupancies at the front end card for the straw system. For an occupancy per crossing of 5%, the L1 buffer would need 18 storage locations and the L2 buffer 8 storage locations. One can impose a minimal distance requirement between two L2 trigger accepts of up to 50 μs with only little additional burden to the L2 buffer.

1 Introduction

We simulate the Front end TVC/AMU for the straw system to determine the buffer requirements of the L1 and L2 buffer. The system is supposed to meet the following requirements:

- sustain a L2 trigger accept rate of up to 10 kHz, with less than 10 % losses.
- work for occupancies of up to 5% / crossing and integrate the signal over 3 crossings.

Model simulations show that the occupancuies of the inner layers of the straw system are about 10% per trigger without any electronic noise. The above assumption contains some safety margin to incorporate this noise.
The model used is a simplified version of the front end simulation model of Sinervo et al. [2]. It uses the beam clock as the minimal time step and starts generating events for any L1 accept. The L1 triggers are generated with an $L1$ accept rate between 62MHz*(0.001-0.003). For L1 trigger accepts, hits are generated with a probability/crossing, occup=0.05. For each trigger 3 beam crossings are read out. These hits then enter the L2 pipeline. The

L2 delay ($L2lat$) is parametrized by a a flat uniform distribution between 10 and 50 μs (a) or 10 and 100 μs (b). The trigger proposal [1] requires the L2 latency time to be between 10 and 50 μs. Here we will usually allow for a latency of up to 100 μs. L2 trigger accept signals are generated with the probability $L2$. We further require a minimum time $L2min$ between two L2 triggers. This time is the same for L2 accepts or L2 rejects.

2 L1 buffer occupancy

The L1 buffer occupancy depends only on the delay of the L1 trigger decision, which we take to be 240 crossings, and the occupancy of one channel per crossing. It follows a binomial distribution:

$$prob(n) = \frac{240!}{(240-n)! * n!} * occup^n * (1 - occup)^{240-n}$$

Simulations werde done with a detailed model of P. Sinervo et al. [2]. Figure 1 shows the L1 buffer occupancy distribution for an occupancy of 2% and 5% per crossing. The maximal buffer length to contain 99 % of the hits, is 10 and 18 for the occupancies of 2% and 5% per crossing, respectively.

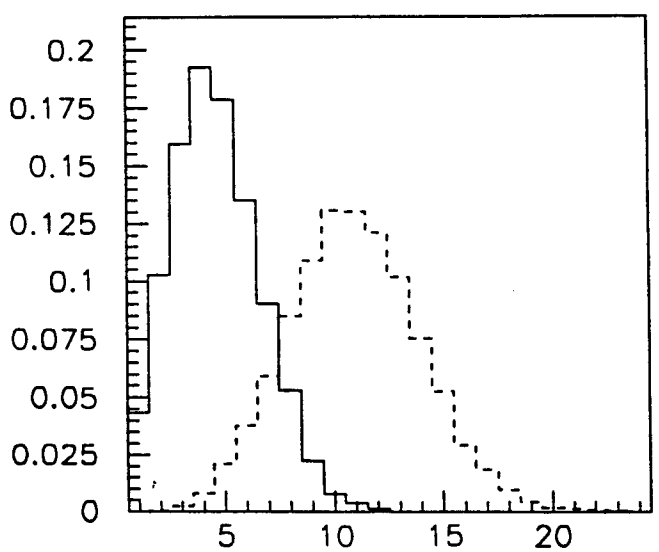


Figure 1: L1 buffer occupancy distribution for an occupancy of 5% per crossing and 2% per crossing.

3 L2 buffer occupancy

We test the required L2 buffer size in a simplified version of the front end simulation model of Sinervo et al. [2]. We note the following parameters for the buffer: the average occupancy n (averaging excludes empty buffer conditions), the variance of the occupancy distribution σ and the maximal occupancy $nmax$ of the L2 buffer. The maximal occupancy is defined as the number of buffers needed to contain more than 99% of the hits. Table 1 gives the results for various parameters.

L1	occup	L2lat	L2min(Xing)	n	σ	nmax
0.001	0.05	a	250	1.3	0.6	4
0.001	0.05	b	250	1.4	0.7	5
0.001	0.1	a	10	1.5	0.8	5
0.001	0.1	b	250	1.9	1.1	6
0.002	0.05	a	250	1.6	0.9	6
0.002	0.05	b	10	1.9	1.1	7
0.002	0.1	a	600	14	7	>24
0.002	0.1	b	10	3.1	1.9	11
0.003	0.05	b	250	2.9	1.6	8
0.003	0.1	b	250	5.0	2.2	13

Table 1: L2 buffer occupancies.

From this table one can infer that the variables n, σ and nmax scale approximately with the square root of L1, occup and L2lat.

$$\mathrm{n}, \sigma, \mathrm{nmax} \propto \sqrt{\mathrm{occup}} * \sqrt{\mathrm{L1}} * \sqrt{< \mathrm{L2lat} >}$$

The introduction of minimal distance between L2 triggers of 250 Xings=4μs does not have a big influence on the performance of the system. This parameter, however, starts to have a large influence as soon as this distance comes close to the mean time between L1 trigger accepts (60 MHz*L1=10-20 μs).
Figure 2 shows the maximal occupancy of the L2 buffer, nmax, as the function of the L1 trigger rate for occupancy/crossing of 0.1, 0.05 and 0.02. For an occupancy of 5% per crossing and a L2 latency time distribution between 10 and 100 μs, a maximal L2 buffer length of 8 hit locations is sufficient.

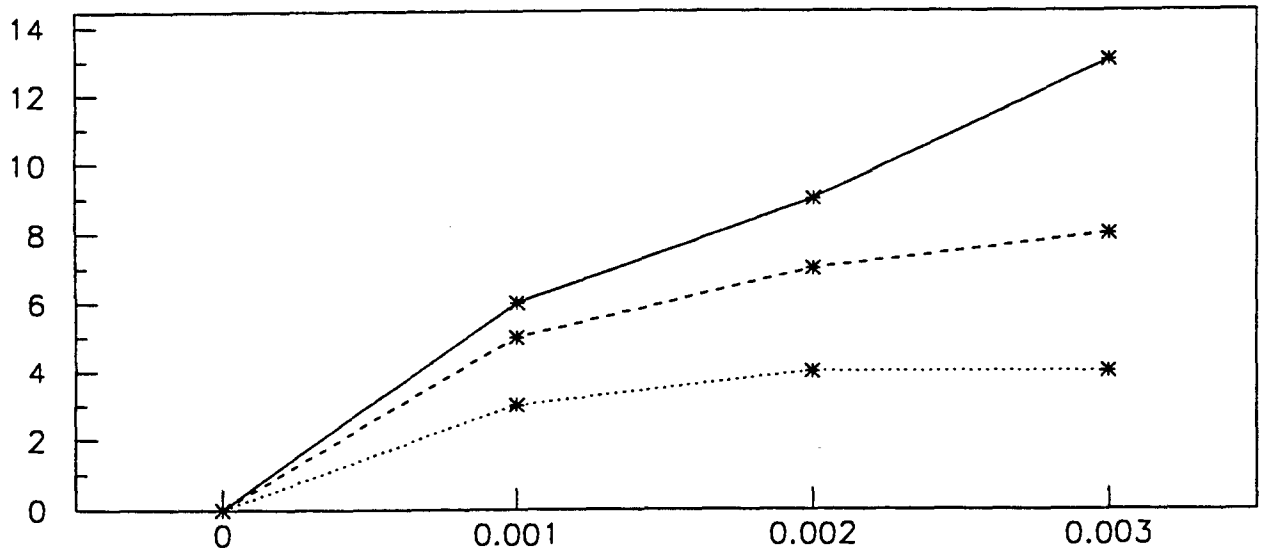


Figure 2: nmax as a function of the L1 trigger accept rate for an occupancy/crossing of 0.1 (upper curve), 0.05 (middle curve) and 0.02 (lower curve).

4 L2 accept separation

For some reasons like event ordering one is interested in the influence of an additional latency between L2 trigger accepts on the performance of the front end buffers. Let τ be the minimal distance between two *L2 trigger accepts*. This is not to be confused with the L2 trigger latency, which is the time the system needs for an L2 trigger decision. This additional latency would require the L2 buffer to take the burden of this additional buffering. For an occupancy of 5% per crossing we show in the following table the L2 buffer occupancies, for different latency times τ.

The last column in the table gives the percentage of L2 trigger accepts that had to be buffered so that the minimal distance criterion was satisfied.
For the L2 latency time we used a uniform distribution between 10 and 100 μs, so that the typical latency was usually much larger, than the additional latency caused by the minimal distance requirement.
The increase in required L2 buffer space on the L2 buffer side seems to be modest, if noticable at all for the assumed parameters. However, as soon as τ approches the L2 accept

L1	L2	L2rate(kHz)	$\tau(\mu s)$	n	σ	nmax	(%)
0.002	0.02	2.4	1	2.1	1.2	5	0
0.002	0.02	2.4	32	2.1	1.2	5	6
0.002	0.02	2.4	64	2.1	1.2	5	11
0.003	0.05	9	1	2.9	1.6	7	0
0.003	0.05	9	32	3.0	1.7	8	25
0.003	0.05	9	64	5.1	2.1	> 8	53

Table 2: L2 buffer occupancies with an minimal L2 trigger accept distance, for an occupancy of 5% per crossing.

rate, the required buffer size increases dramatically. For the last entry, 16 % of the hits are lost for the L2 buffer length of 8.

The small increase in buffer requirement can be understood in the following way: The minimum L2 accept requirement introduces *on the average* a latency of L2 * τ for each event, which is quite small compared to the L2 decision latency, since L2 is small. This effect, however, becomes very important when the probability that during this latency another L2 accept trigger occurs, becomes important. At this point the L2 buffer queue is not emptied any more.

Figure 2 compares the L2 buffer occupancy distribution for (a) L1=0.002 and L2=0.02 and (b) for L1=0.003 and L2 =0.05 for minimal L2 trigger accept separations of 1, 32 and 64 μs. In the case a) the difference in the distributions is nearly imperceptible. In case b) the distributions for $\tau = 1\mu s$ and $\tau = 32\mu s$ are very similar, while the case of $\tau = 64\mu s$ would need more bufferspace. 16 % of the hits are lost for an L2 buffer length of 8.

Figure 3 shows the percentage of L2 accept triggers which had to be delayed, to enforce the L2 trigger accept seperation.

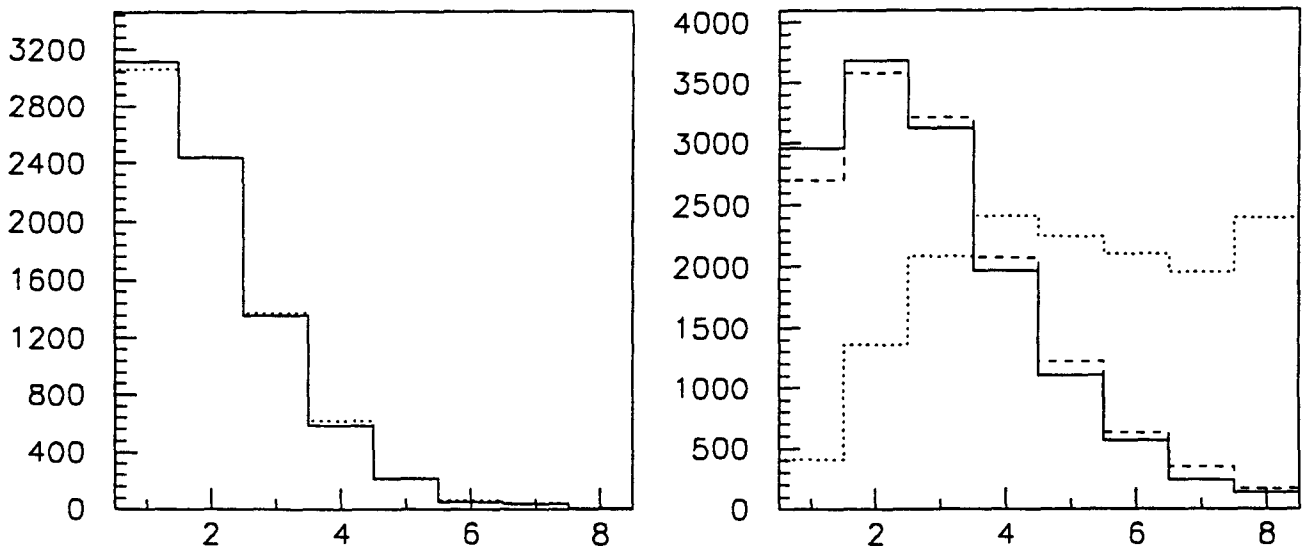


Figure 3: L2 buffer occupancy for a) L1=0.002 and L2 =0.02 and b) L1=0.003 and L2=0.05. The three cases $\tau = 1, 32, 64 \mu s$ are superimposed on each other.

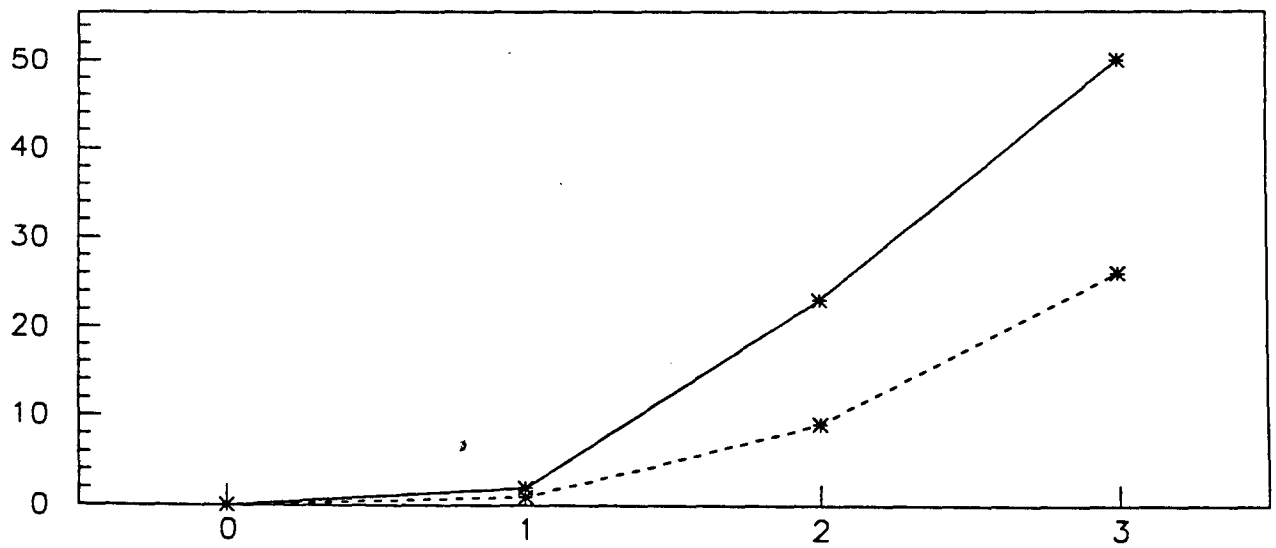


Figure 4: % of L2 trigger accepts which have to wait in order to enforce the minimal distance condition, for separation times of 32 and 64 μs.

5 Conclusions

In this note we have evaluated the buffer requirements for the L1 and L2 buffers, which are located on the TVC/AMU. In order to collect more than 99% of the hits, the L1 buffer needs to have 18/12 storage locations for an occupancy of 5% or 2% per crossing, respectively. The L2 buffer needs 8/4 storage locations for the two different occupancies, assuming the L2 latency to be uniformly distributed between 10 and 100 μs. Without too much additional burden a minimal distance of 30-60 μs between 2 L2 accept triggers can be imposed. The buffer needs are only increased for very large rates (L2=9 kHz) and long minimal distances of 64 μs.

References

[1] SDC detector notes, SDC-91-89, SDC Trigger preliminary conceptual design.

[2] P. Sinervo et al., *SSC Front End Simulations* Forth Worth Symposium 1990.

Buffer Occupancies for the SDC Straw Tube DCC System

A. Hölscher, G. Stairs and P. K. Sinervo
University of Toronto
Toronto, Canada

March 23, 1992

Abstract

In this note we present the results of simulations of the buffer occupancies at the front end card and at the crate interface card for the SDC straw tube system.

1 Introduction

The general architecture of this system as it is assumed in this simulation is as follows [1] (see figure 1). On each front end board reside 5 FMUX's, which are connected to 8 four channel TVC/AMU's (or TMC's) each, so that we have 160 straw channels per front end module. The FMUX's have one buffer for each TVC/AMU they are connected to. After each L2 accept trigger the data are first digitized on the TVC/AMU, which might take 1-3 μs, and then transmitted to the FMUX. This link is clocked at about 1 MHz and each clock cycle 4 bits are transferred to the FMUX. One hit from the TVC/AMU generates 3 bytes of data. On the FMUX a 4th byte for the geographical address is added. The 5 FMUX's residing on one front end board are read out one after the other by one FETX that transmits the data to a buffer on the CIC (crate interface card), which resides on a data acquisition crate outside the calorimeter. Each CIC reads 20 front end boards out in this scheme. For the whole straw system we forsee 32 CICs. We are interested in the occupancies of the buffers and bandwidth requirements at various stages.

The system is supposed to meet the following requirements :

- sustain an L2 accept rate of up to 10 kHz, with less than 10 % losses, and
- work for occupancies of up to 15%/trigger/channel.

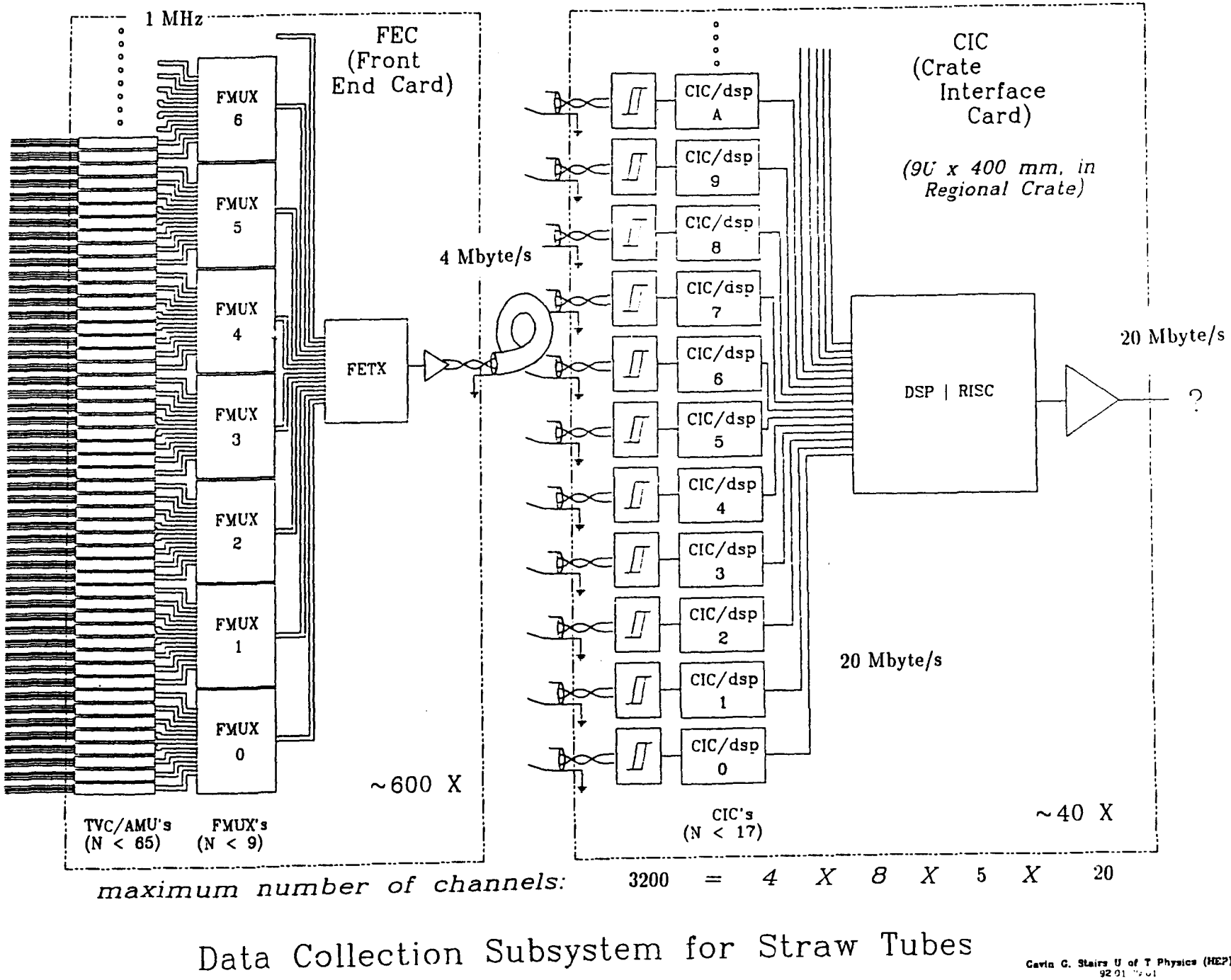


Figure 1: Architecture of the SDC Straw Tube DCC System.

Model simulations [2] show that the occupancies of the inner layers of the straw system are about 10% per trigger without any electronic noise. The above assumption contains some safety margin to incorporate this noise.
In this note the occupancies were generated, unless otherwise stated, with an L2 trigger accept rate of 9 kHz and an occupancy of 5% per crossing with 3 crossings per trigger being read out.

2 Bandwidths

From these requirements one can deduce the following average rates:

$$\text{datarate(TVC/AMU} \rightarrow \text{FMUX)} = 4 * 0.15 * 10\text{kHz} = 0.006 \text{ hits}/\mu\text{s}$$

Our protocol between the FE chip and the FMUX requires the FE to generate an "end-of-event" datum for every L2 trigger (see section 4). Including this additional effect, we obtain an average rate of 0.016 hits / μs.
Assuming the connection TVC/AMU - FMUX is clocked at 1 MHz and that the data are transferred in 1 nibble=4 bit, it will take time = N $*$ 6μs (N number of hits on one TVC/AMU) until the data from one TVC/AMU arrives at the FMUX.
For the datarate from the FMUX to the CIC one gets the following values:

$$\text{datarate(FMUX} \rightarrow \text{CIC)} = 160 * 0.15 * 10\text{kHz} = 0.25 \text{ hits}/\mu\text{s}$$

Assuming one hit is 4 bytes long and allowing for a safety margin of at least a factor 4, the minimal required bandwidth for the bus to the TVC/AMU is :

$$\text{bandwidth(FMUX} \rightarrow \text{CIC)} > 4 \text{ Mbyte/s}$$

For each front end board there is a separate input buffer on the CIC. The bus which reads out these input buffers needs to carry the following rate:

$$\text{datarate(CIC} \rightarrow \text{CIC)} = 20 * 160 * 0.15 * 10\text{kHz} = 5.0 \text{ hits}/\mu\text{s}$$

Assuming again that one hit contains 4 byte and requiring a safety factor of at least 2 times the required bandwidth one obtains:

$$\text{bandwidth(CIC} \rightarrow \text{CIC)} > 40 \text{ Mbyte/s}$$

This is also the rate with which the data are transmitted to the DAQ.
The further one is away from the front end module, the smaller the safety factor needs

to be, since the fluctuations become smaller and smaller. This will be confirmed by the subsequent simulations. Generally, as long as one stays some reasonably safe factor away from the limiting bandwidth, the system performs very well.

3 Correlations

For each L2 trigger accept signal, we generate hits for each channel according to an average occupancy. In a previous note [2] we have shown, however, that the front end boards typically contain either no hit or at least 6 hits, due to the fact that a transversing particle sets a hit in all layers of one superlayer. To simulate this type of correlations, we assign to each event a different average occupancy. The distribution of this average occupancy is assumed to be exponential

$$f(occup) \propto exp(-occup/ < occup >)$$

with a mean occupancy of 5% per crossing. Three crossings are read out for each trigger.

4 FMUX occupancies

The FMUX's reside on the front end card and read out 8 TVC/AMU's (or TMC's) each. On each front end card there are 5 FMUX's, which are read out by one FETX. The FETX reads the FMUX's one after the other and transmits the data to a buffer on the CIC. SDC mandates that all data in the DAQ system to be event and channel ordered. The two issues concerned with this ordering are:

- data ordering by the geographical channel address, and
- event ordering by time and disentangling data from different events.

There are two possibilities to order the data from different events:

- Introduction of a latency between L2 trigger accepts, so that one event can be read out completely from the FMUX before the next one enters it.
- Let the TVC/AMU send an empty hit to indicate the end of an event for one trigger and the beginning of the next trigger. The FMUX would then continued to be read out until the empty hit for each L2 trigger.

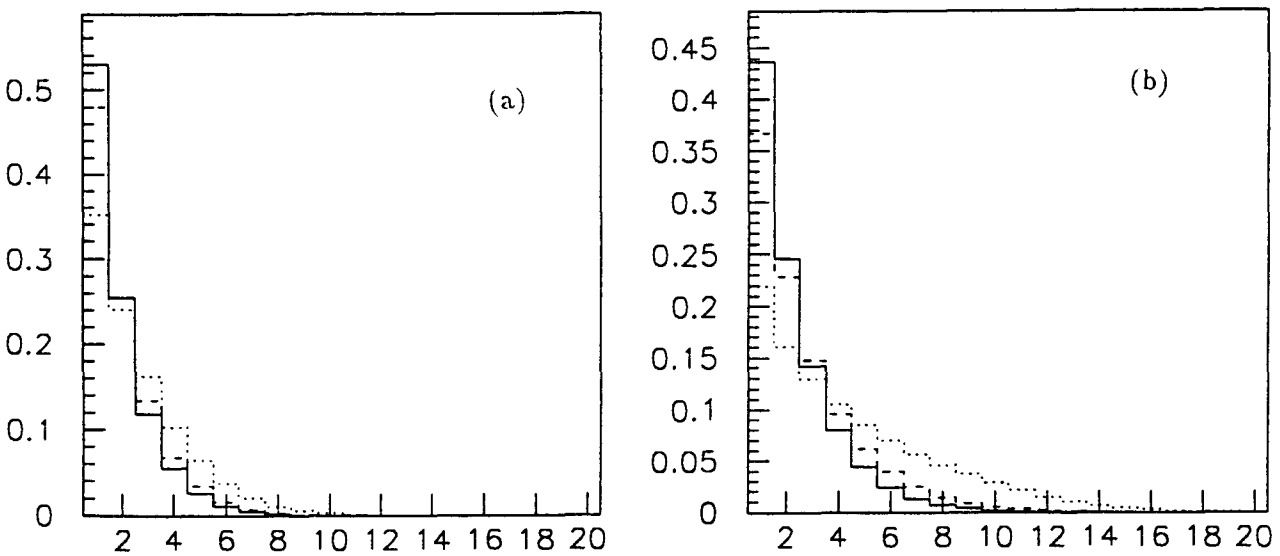


Figure 2: FMUX occupancy (in terms of hits) for one four-channel TVC/AMU without (a) or with (b) one empty hit for each event. The occupancy is shown for three different FMUX-FETX-CIC bandwidths of 2 (dotted), 4 (dashed) and 8 Mbyte/s (solid histogram).

The introduction of a minimal distance between L2 trigger accepts of up to 50 μs introduces a sustainable load on the the L2 buffer as was shown previously [3]. However, problems might occur for events that have a high occupancy in one FMUXR. This scheme also requires some additional logic in order to enforce this minimal L2 trigger accept distance.
In the second solution, the TVC/AMU adds an empty hit to the end of the data for each event. The FMUX is then read out until the occurence of an empty hit and one would naturally get the hits ordered after their event number. The disadvantages of this method are an increased TVC/AMU - FMUX bandwidth and an increased buffer requirement on the FMUX. Both disadvantages don't seem to be severe, however, and we use this method for the subsequent simulations.
The ordering by geographical address can be done by reading out the FMUX's in the same order for each trigger. This reduces the effective bandwidth somewhat, since one might have to wait for the data to arrive at a particular FMUX before the readout can continue. This degradation, however, is acceptable. Figure 2 shows the occupancy of an FMUX buffer for one four-channel TVC/AMU for three different FMUX-FETX-CIC bandwidths of 2, 4 and 8 Mbyte/s and for the case where one empty hit at the end of each event is not included (a) or is included (b). Figure 3 shows the required FMUX buffer length in order to contain 99.9% of the hits for the various cases. For a FMUX-CIC bandwidth of 4 Mbyte/s, the hits

are well contained in a buffer of length 15.

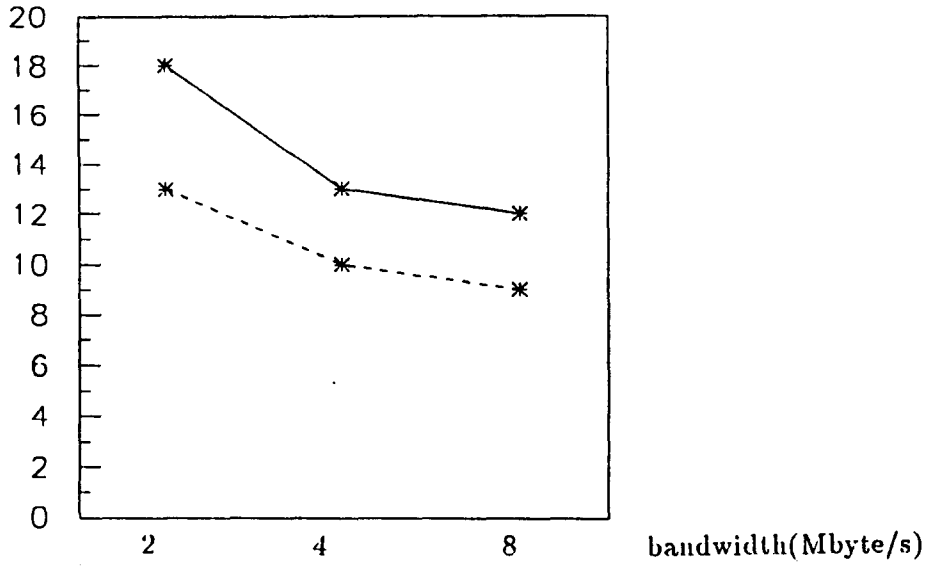


Figure 3: max FMUX buffer occupancy (99.9%) as a function of the FMUX-CIC bandwidth.

4.1 Recovery fom buffer overflow

Special care has to be taken when the FMUX buffer overflows, in order not to loose the control information provided by the empty hit. When the FMUX buffer is full, we propose to stop the data transmission from the front end chip to the FMUX. This will cause first the L2 buffer and then the L1 buffer on the chip to be filled up and eventually cause the chip to disable its input when no space is left for hits in the L1 buffer. As soon as the FMUX has space again, the data transmission from the TVC/AMU (or TMC) to the FMUX buffer can resume.

5 CIC occupancies

The data are transmitted from the front end boards to buffers on the crate interface card (CIC), which is located outside the calorimeter. Each CIC has one data input buffer for each of the 20 front end board it is connected to. It then puts the data belonging to one trigger together in one output buffer, from where the data are transmitted to the DAQ. Apart from collecting the data from the front end board, the CIC card has some control

and monitor functions. The input buffer needs on the CIC are of course dependent on the bandwidth with which theses buffers are emptied. Figure 4 shows the occupancy distribution of the CIC input buffers for a CIC bandwidth of 24 Mbyte/s (again assuming that each hit consists of 4 bytes). In this figure, we compare the case, where (a) the event data are not geographically ordered, but the input buffers on the CIC are read out in the order that they arrive in the CIC, (b) second where we assume a fixed occupancy for each event, neglecting correlations as they were assumed in section 3 and (c) assume the type of correlations presented in section 3 and reading out the CIC front end buffers always in the same order. The percentage of hits lost with a buffer length of 1000 hits per front end card is 0.6%, 0.03% and 1.6%, for the three cases, respectively. The occupancy distributions don't show big differences between the three cases. Reading the front end boards out in the order of their availability (a) gives a slightly larger throughput than always reading them out in the same order, whereas neglecting correlations within one event leads to smaller losses (b) compared to the nominal case where we get ordered event and were we put in some correlations. Although these differences are expected the difference between the distributions is much smaller than one probably would have anticipated.

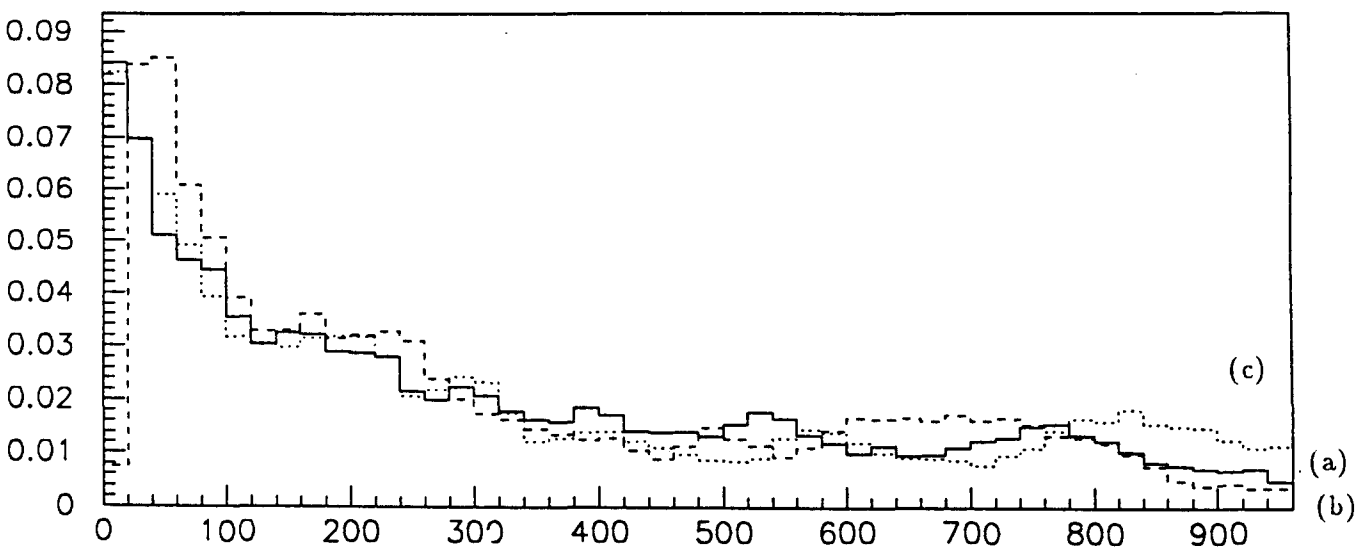


Figure 4: CIC input buffer occupancy (in terms of hits) for one front end board. Case (a) solid, (b) dashed and (c) dotted histogram).

Figure 5 shows the CIC buffer occupancy distribution dependence on the CIC bandwidth. The occupancies are shown for a bandwidth of 24, 36 and 48 Mbyte/s. Also shown are the maximal occupancy for different bandwidths. For a CIC bandwidth exceeding 40 Mbyte/s

the hits are well contained in a buffer length of 400 hits.

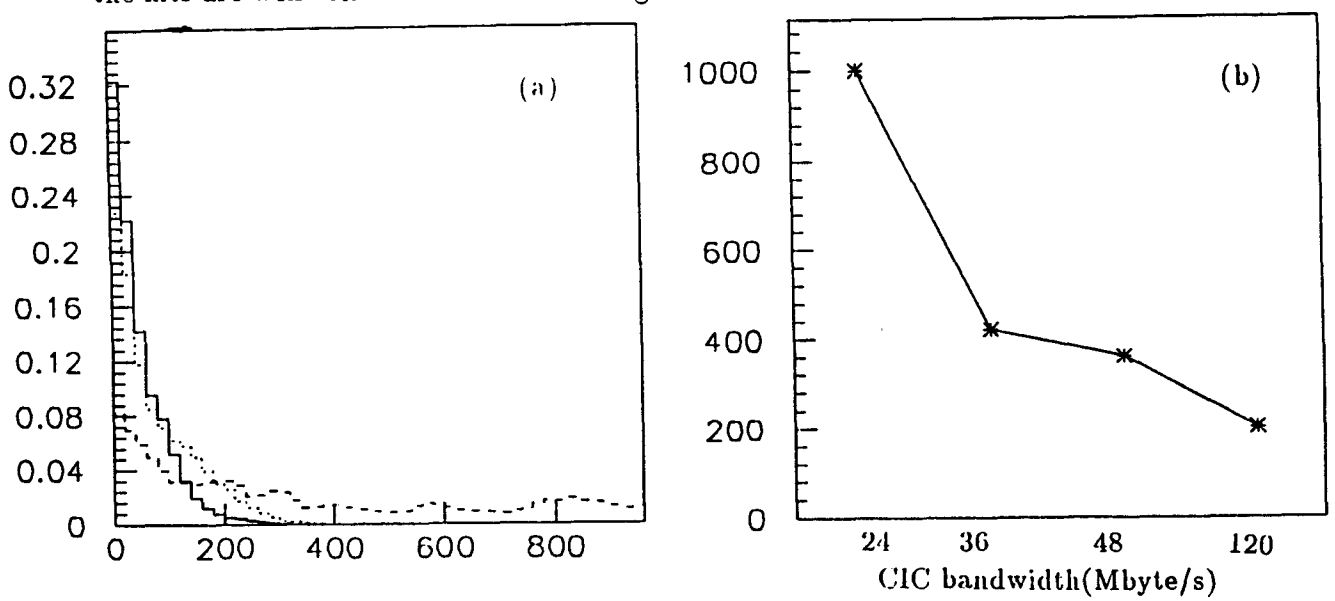


Figure 5: CIC input buffer occupancy for one front end board, for three different bandwidths on the CIC: 24 (dashed), 36 (dotted) and 48 Mbyte/s (solid histogram). The histogram in (b) shows the maximal (99.9%) occupancy.

5.1 Influence of CIC delay times

Since the CIC has also monitor and control functions, we are also interested in the buffer requirements assuming that the hits have to stay a minimal time τ in the input buffer. This would give the a DSP (see figure 1) some time for monitor and control functions. Figure 6 shows the CIC input buffer occupancy distribution for an CIC bandwidth of 48 Mbyte/s for 3 different minimal times $\tau = 0$, 500 and 1000 μs in the CIC buffer. Even for a delay time of 1000 μs, the additional burden on the CIC buffers is small. In this case one would like to have a somewhat larger storage of 500 hits.

5.2 Load balanced Front End Readout

Simulations of the occupancies of front end modules [2] show that the occupancies have a strong dependence on the superlayer number in the sense that the occupancies of the outer layers are much smaller than the inner layer. Since the CICs are located just outside of the calorimeter in different ϕ positions, geometrical arguments and arguments of load balance suggest, that each CIC reads out front end modules of different layers. This lowers the

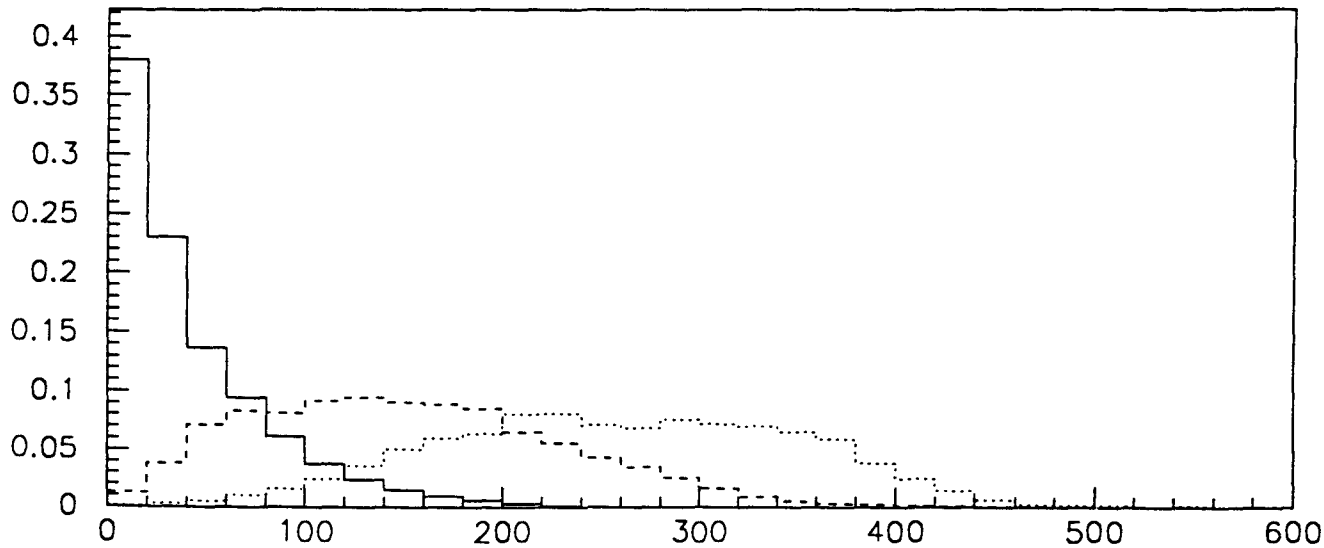


Figure 6: CIC buffer occupancy for one front end board for a CIC bandwidth of 48 Mbyte/s for three different CIC minimal stay times of 0 (solid), 500 (dashed) and 1000 μs (dotted histogram).

bandwidth requirements of one CIC, since the occupancies of the outer layers are much smaller. We assume that the 20 front end boards of one CIC contain 4 front end module of each superlayer. We assume the following average occupancies per trigger for the 5 superlayers of 15%, 9%, 6%, 4.5% and 4.5%, which is still well above the results of the simulation of 10%, 4.7%, 3.3%, 2.5% and 2.5% [2]. Figure 7 shows the CIC buffer ocupancy distributions for different CIC bandwidths of 24, 16 and 12 Mbyte/s. Here no difference is made between CIC reading out superlayers with higher or lower occupancies. For this load balanced front end readout, the required CIC bandwidth is only 20 Mbyte/s.

5.3 CIC output buffer and bandwidth to DAQ

The data from different front end boards are now assembled and added to one output buffer, where they are to be shipped off to the data aquisition system. The buffer size needed here, of course, depends on the specification of this connection, which at this stage is not defined. Assuming the load balanced case (section 5.2), a CIC - DAQ bandwidth of 24 Mbyte/s and no additional delays, one gets the output buffer distribution of figure 8, showing that in this case the buffer can contain as many as 1000 hits. This requirement will certainly grow.

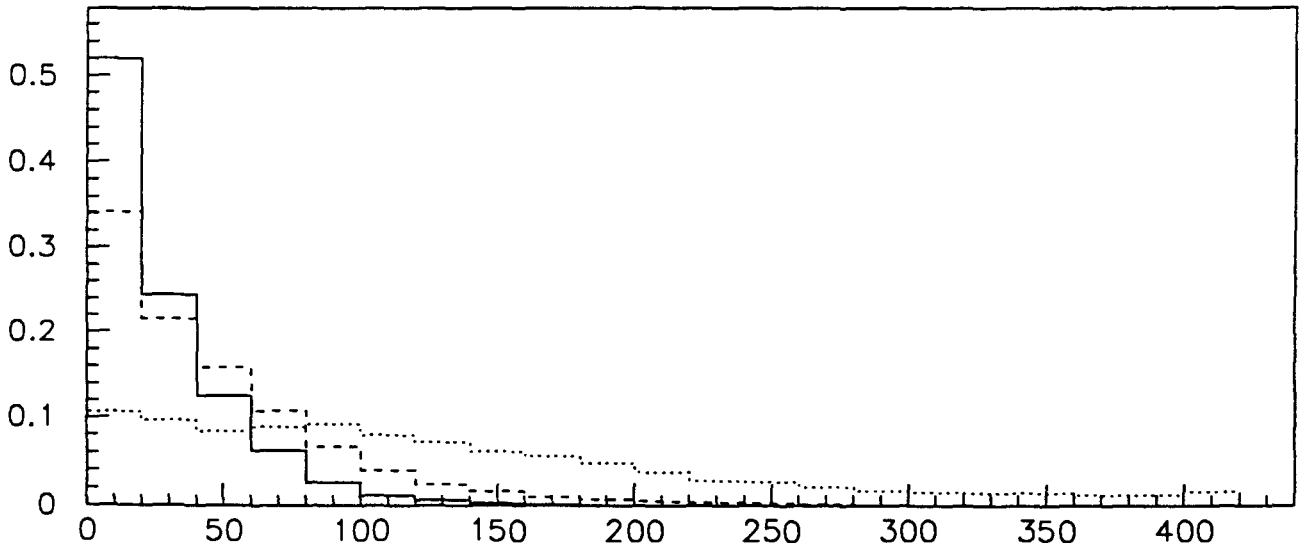


Figure 7: CIC buffer occupancy for one front end board for a CIC bandwidth of 24 (solid), 16 (dashed) and 12 Mbyte/s (dotted histogram), with the CIC reading out 4 front end boards of each superlayer.

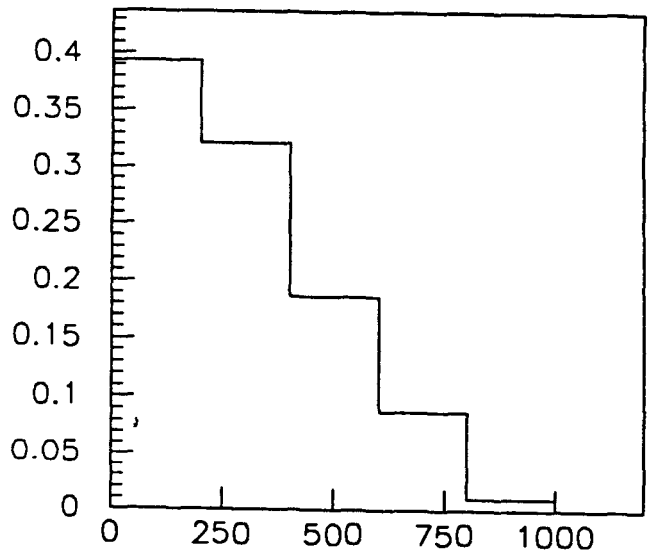


Figure 8: CIC output buffer occupancy for the load balanced front end readout (section 5.2) and a CIC-DAQ bandwidth of 24 Mbyte/s and no additional delays.

6 Conclusion

In this note we discussed the buffer needs and bandwidth requirements of the front end boards and of the crate interface card. Adding the conclusion from a previous study [3] we arrive at a set of minimal parameters for a front end system, assuming the requirements that:

- the system has to sustain an L1 accept rate of 100 kHz
- the system has to sustain an L2 accept rate of 10 kHz
- allow the straw tubes to have an occupancy of 15 % per trigger
- and allow for an L2 latency time of 10 - 100 μs uniformly distributed

The system would have the following parameters:

- L1 buffer length >= 18 hits
- L2 buffer length >= 8 hits
- TVC/AMU (or TMC) - FMUX link clocked by at least 1 MHz with 4 bits transferred/clock
- FMUX buffer length per 4 channel TVC/AMU of at least 15 hits. Assuming each hit contains 4 bytes, one needs 8*4*15 =480 bytes storage on each FMUX.
- the bandwidth FMUX - FETX - CIC exceeding 4 Mbyte/s.
- CIC input buffer length per front end board exceeding 1.6 kbyte. Assuming that each CIC reads out 20 front end boards one would need at least 32 kbyte on the CIC as an input buffer.
- requiring the data to stay'some minimal time of 500 μs to 1000 μs in these CIC input buffers to allow for some data monitoring operations, imposes only a small additional burden on the buffer.
- bandwidth on CIC with which the 20 front end board buffers are read out exceeding 40 Mbytes. If one CIC reads out front end boards roughly equally distributed over the superlayers (load balanced), a lower bandwidth exceeding 20 Mbyte/s is sufficient.

- the CIC output buffer for the load balanced case with a CIC-DAQ bandwidth of 24 Mbyte/s is required to be at least 1000 hits. This requirement will certainly grow, depending on the specifications of the CIC-DAQ link.

References

[1] G.Stairs et al., *Front end Collector Bus Design Description*, SDC-92-242.

[2] A. Hölscher, P. K. Sinervo and G.Stairs, Univ. of Toronto,
Simulation of Front End Board Occupancies for the SDC Straw Tube Tracker

[3] A. Hölscher, P. K. Sinervo and G.Stairs, Univ. of Toronto,
Front End Buffer Requirements for the SDC Straw Tube Tracker

Richard Partridge
DAQ and Trigger Simulation Workshop
April 23, 1992

SIMULATION OF NON-BLOCKING DATA ACQUISITION ARCHITECTURES

- Simulation Goals
- Non-Blocking DAQ Architectures
- The Simulation Model
- Baseline Simulation
- Event Generator Parameters
- Bandwidth Scaling
- Latency in the Event Builder
- Conclusions

Simulation Goals

Simulation goals include:

- Develop generic DAQ simulation that includes essential features of a broad class of DAQ architectures
- Model large system such as will be used at SSC or LHC
- Avoid detailed simulation specific to a particular architecture
- Focus on global behavior of DAQ
- Compare simulation results with scaling predictions

Detailed design and simulations are required at a later stage to:

- Determine bandwidths, latencies, buffering, and other inputs used in the generic model
- Study architecture specific fault conditions
- Determine characteristics of input data distributions
- Verify performance of actual system

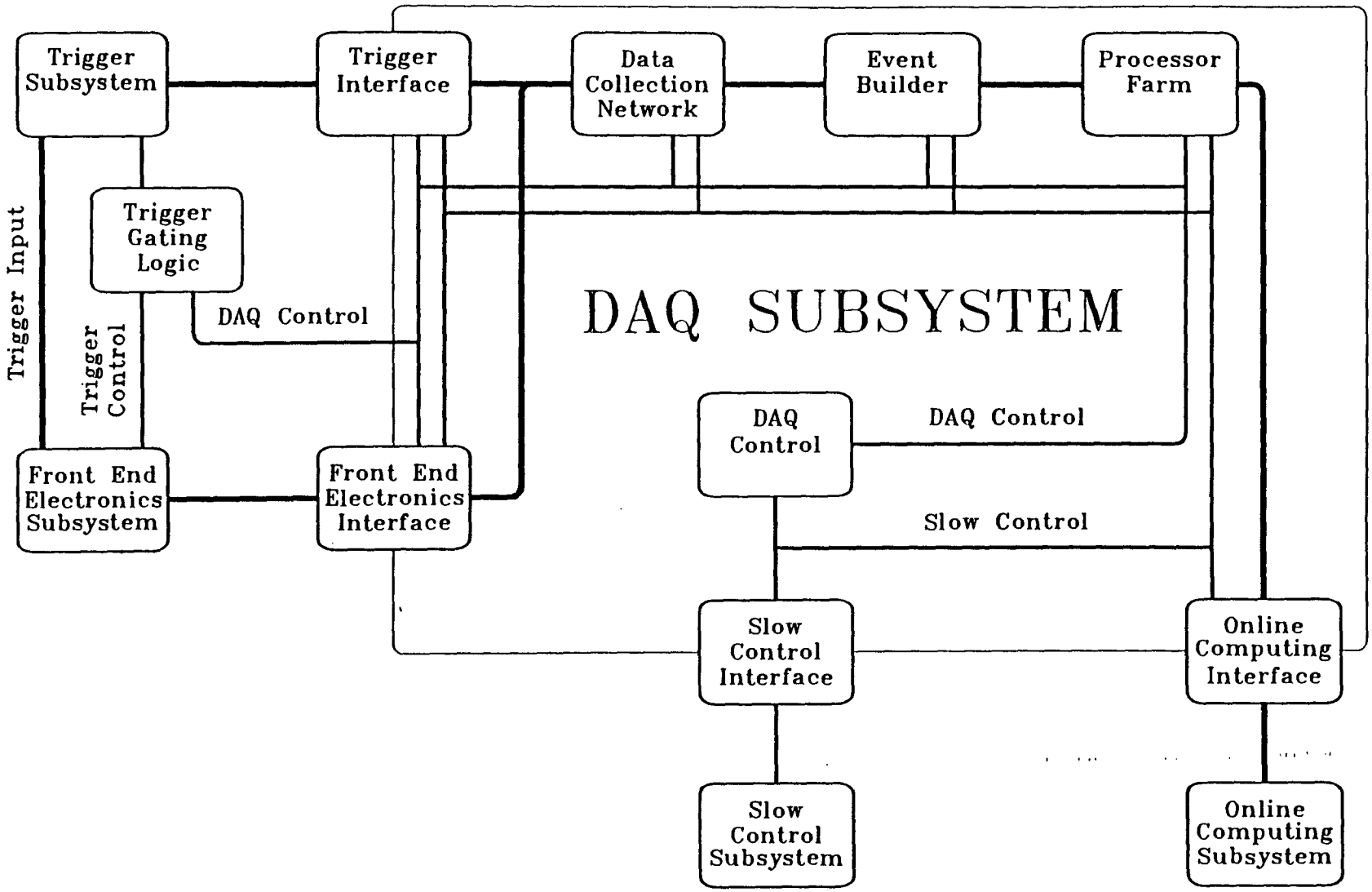


Figure 1: DAQ functional components and connections to other subsystems

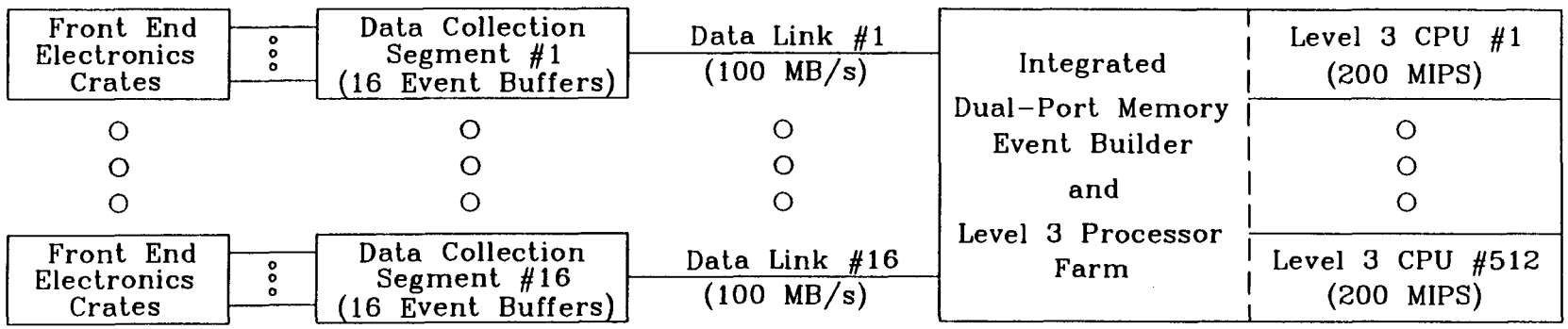
Non-Blocking DAQ Architectures

A non-blocking DAQ architecture is defined to be a data acquisition architecture with a set of data sources of fixed bandwidth. The bandwidth that can be delivered from a given data source is uncorrelated with activity on the other data sources. A non-blocking architecture should, in principle, have a much more predictable behavior due to the lack of correlations.

Examples of non-blocking architectures:

- Dual-port memory architecture
- Cross-point switching network
- Network with non-blocking packet switch/router

Example of a blocking architecture: Several data sources on a shared bus or network segment


Front End Electronics Crates
Data Collection Segment #1 (16 Event Buffers)
Data Link #1 (100 MB/s)
Front End Electronics Crates
Data Collection Segment #16 (16 Event Buffers)
Data Link #16 (100 MB/s)
Integrated Dual-Port Memory Event Builder and Level 3 Processor Farm
Level 3 CPU #1 (200 MIPS)
Level 3 CPU #512 (200 MIPS)

The Simulation Model

The simulation model was written in VERILOG, and includes the following elements:

DATA SOURCE

- Generate Gaussian distribution for data lengths, L_i, on each data source
- Average length for a source, $\langle L_i \rangle$, has a Gaussian distribution about the nominal length $\bar{L}$
- Apply Gaussian normalization factor to each data length to provide correlations between data sources
- Add fixed length to each segment for headers, noise, etc.
- Negative values from Gaussian distributions forced to 0 with distribution corrected to give proper mean
- An input queue buffers a number of events prior to sending the data through the DAQ
- Deadtime occurs if one or input buffers are full

DATA TRANSMISSION

- Data is sent over fixed bandwidth links in order of generation
- Each data link operates independently of the other links, so different links may be transmitting different events

EVENT BUILDER

- Event builder waits until all data for a given event has been received and then signals the processor assigned to the event to begin processing

- Two types of event builders considered: dual-port memory architecture with no latency and network/switch architecture with random latency time between data source and destination

PROCESSOR FARM

- A processor is assigned to an event at the time of its generation
- If no processor is available, deadtime is incurred
- The CPU time required for making a trigger decision is modeled as a fixed time for all events plus a variable time that depends on the nature of the event and has an exponential distribution
- Logging of accepted events is assumed to not incur additional deadtime

SIMULATION PARAMETERS

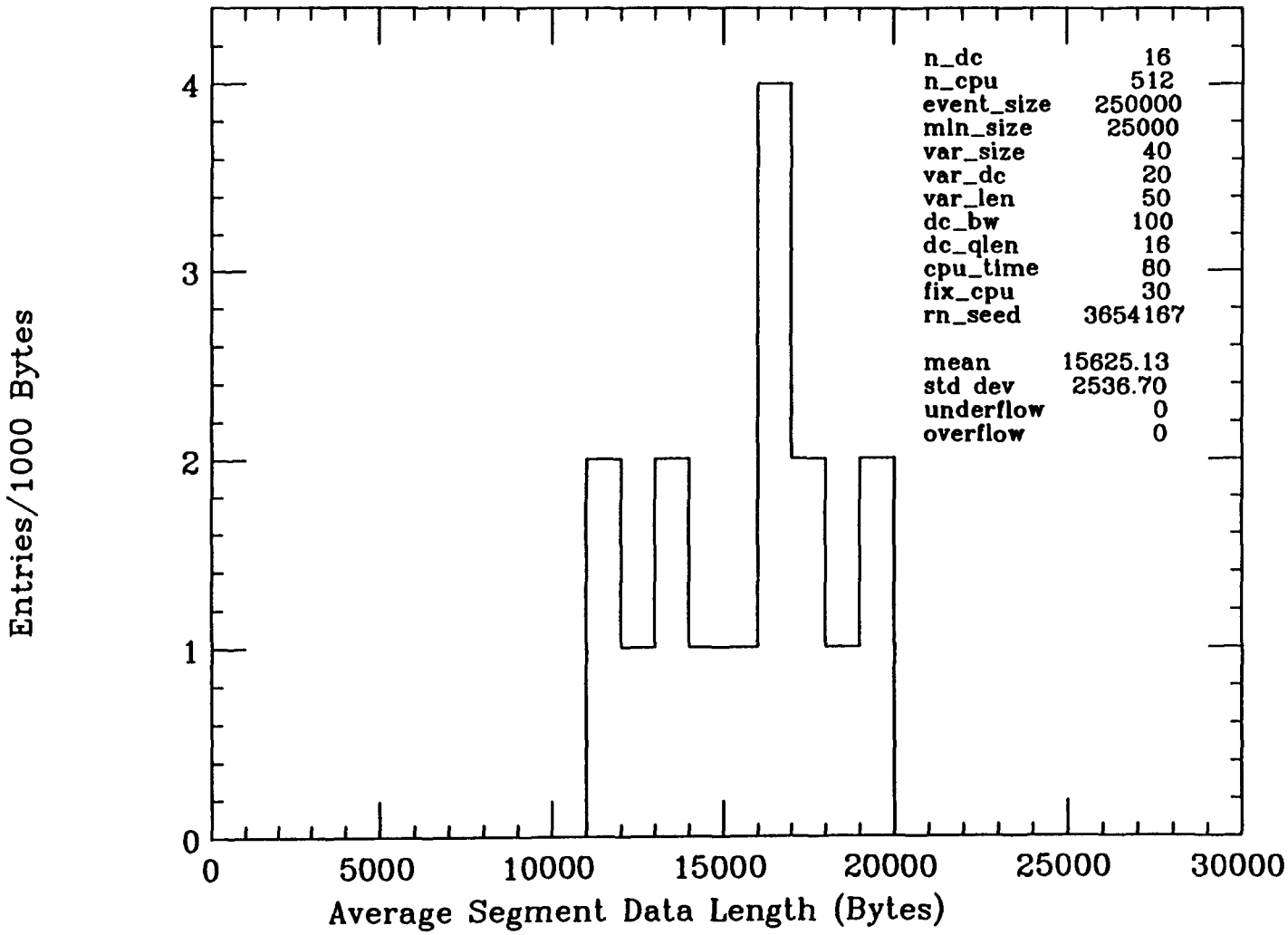
The following is a list of parameters used in the DAQ model:

n_{dc}	Number of data cables
n_{cpu}	Number of farm CPU's
min_size	Minimum event size
event_size	Average event size ($\equiv$ min_size $+ n_{dc} \cdot \bar{L}$)
var_size	σ of event size (% of $n_{dc} \cdot \bar{L}$)
var_dc	σ of $\langle L_i \rangle$ (% of $\bar{L}$)
var_len	σ of L_i (% of $\langle L_i \rangle$)
dc_qlen	Number of buffers in the data collection network
eb_lat	Average latency in event builder (ms)
eb_qlen	Number of event builder buffers per CPU
cpu_time	Average CPU time for an event (ms)
fix_cpu	Minimum CPU time (% of cpu_time)
rn_seed	Random number seed

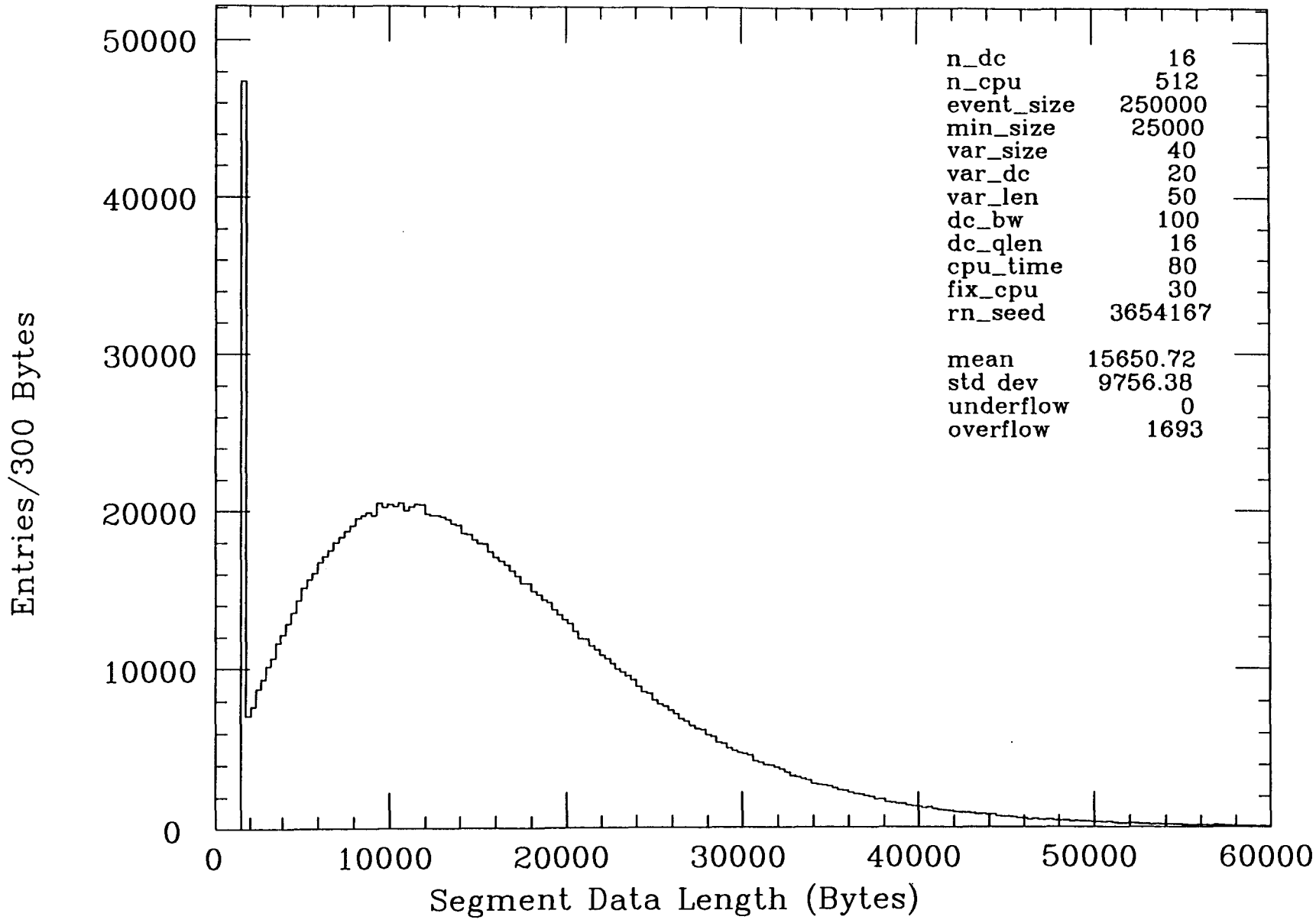
Baseline Simulation

Simulation Procedure:

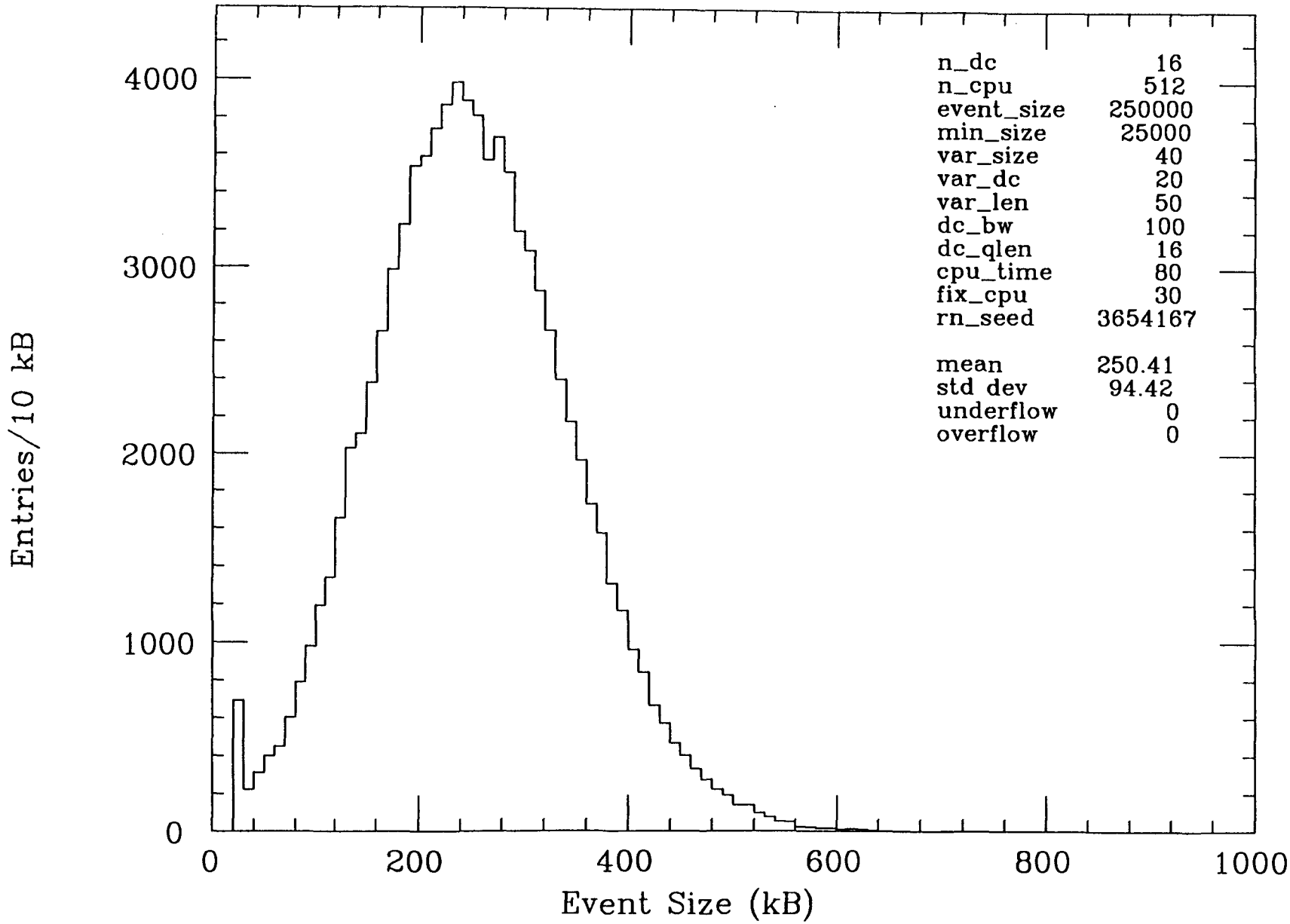
- Measure the deadtime for each run in a series with differing hardware trigger rates
- Fit dead-rate versus trigger rate
  - Determine asymptotic event rate @100% deadtime
  - Determine deadtime at crossover where the trigger rate is equal to the asymptotic event rate
- Plot deadtime versus event rate for live events
- Compare asymptotic rate with predicted rate
- Study variation of model parameters around a set of baseline parameters



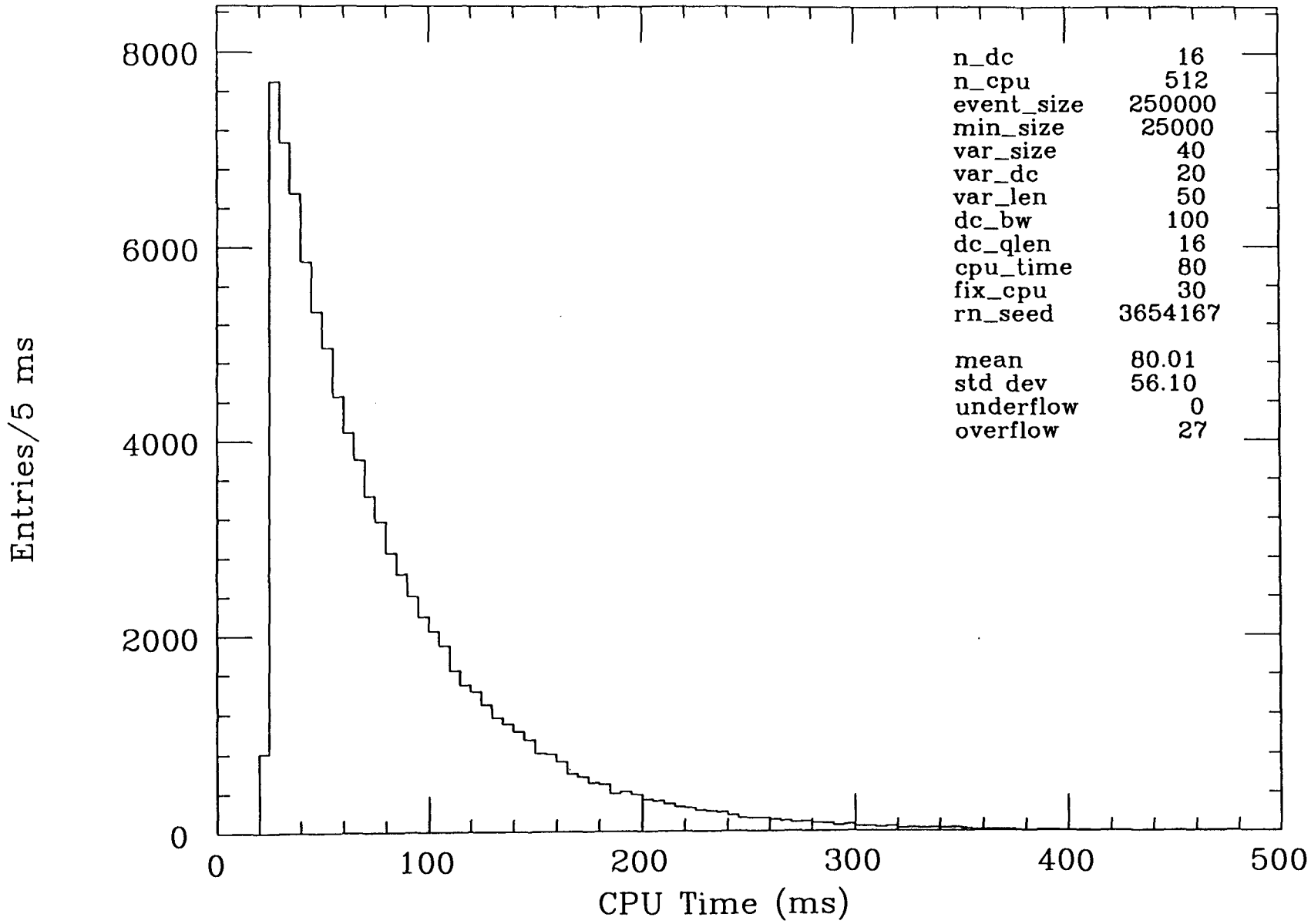
DPMDAQ V1.0 cpu80



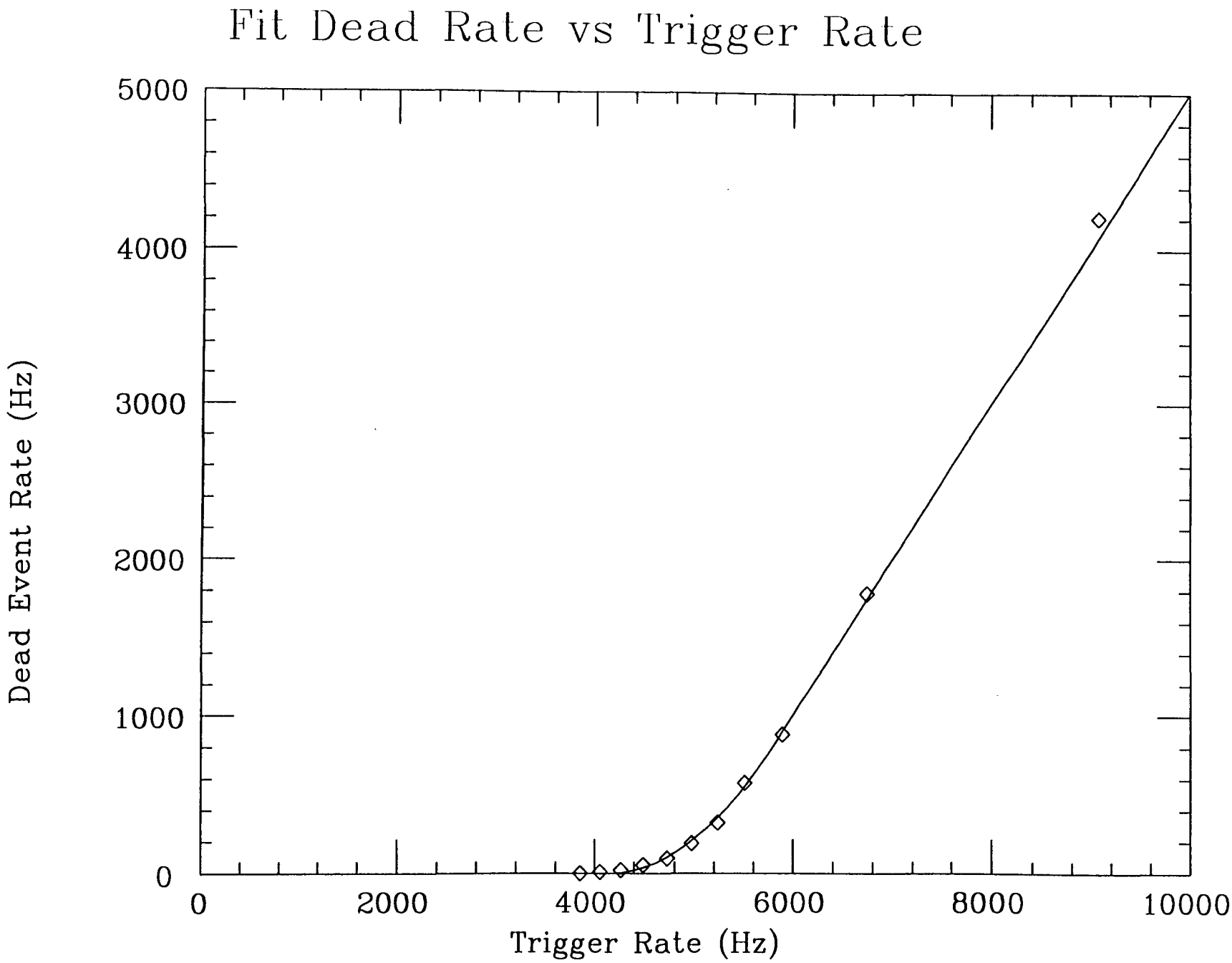
DPMDAQ V1.0 cpu80

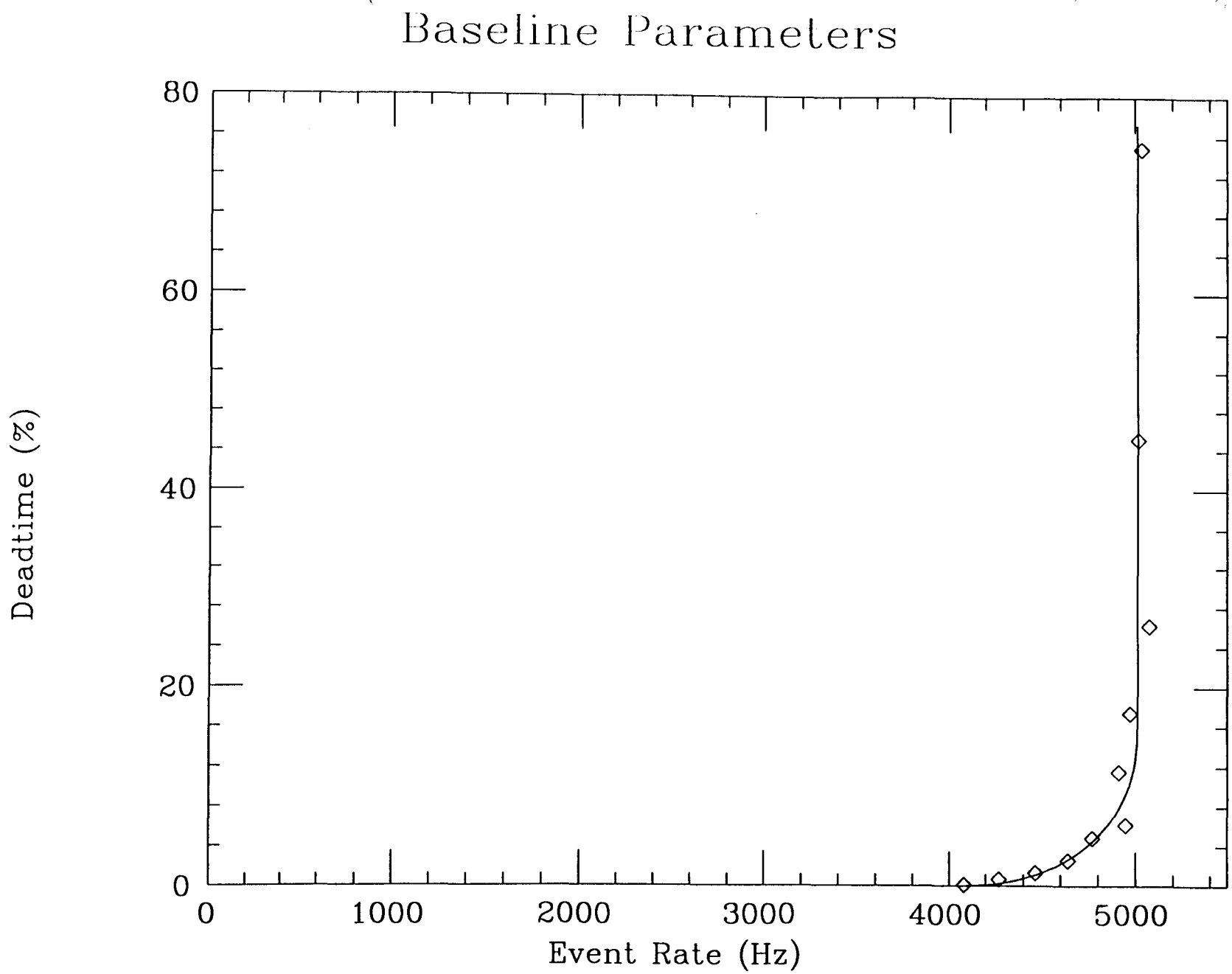


DPMDAQ V1.0 cpu80



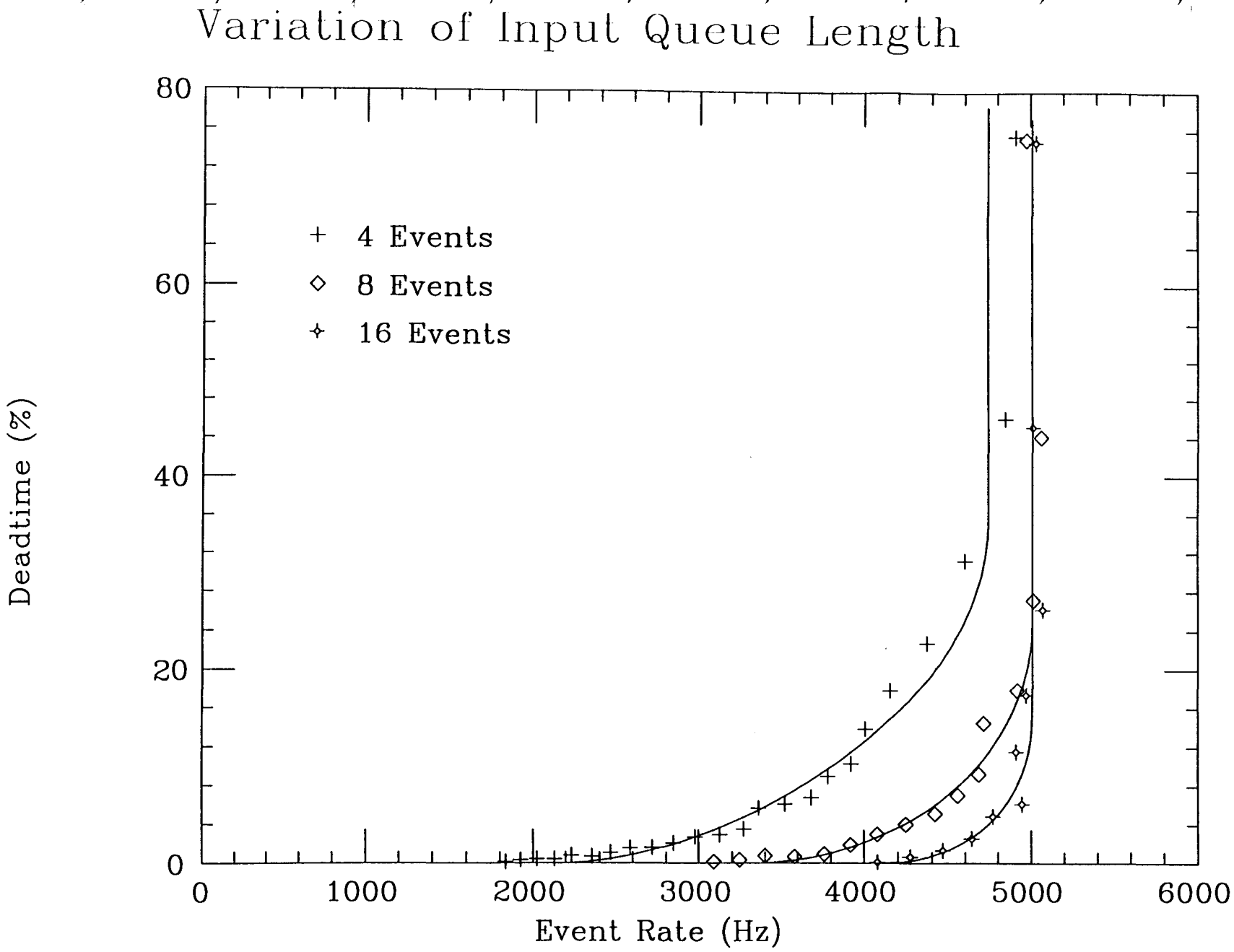
DPMDAQ V1.0 cpu80

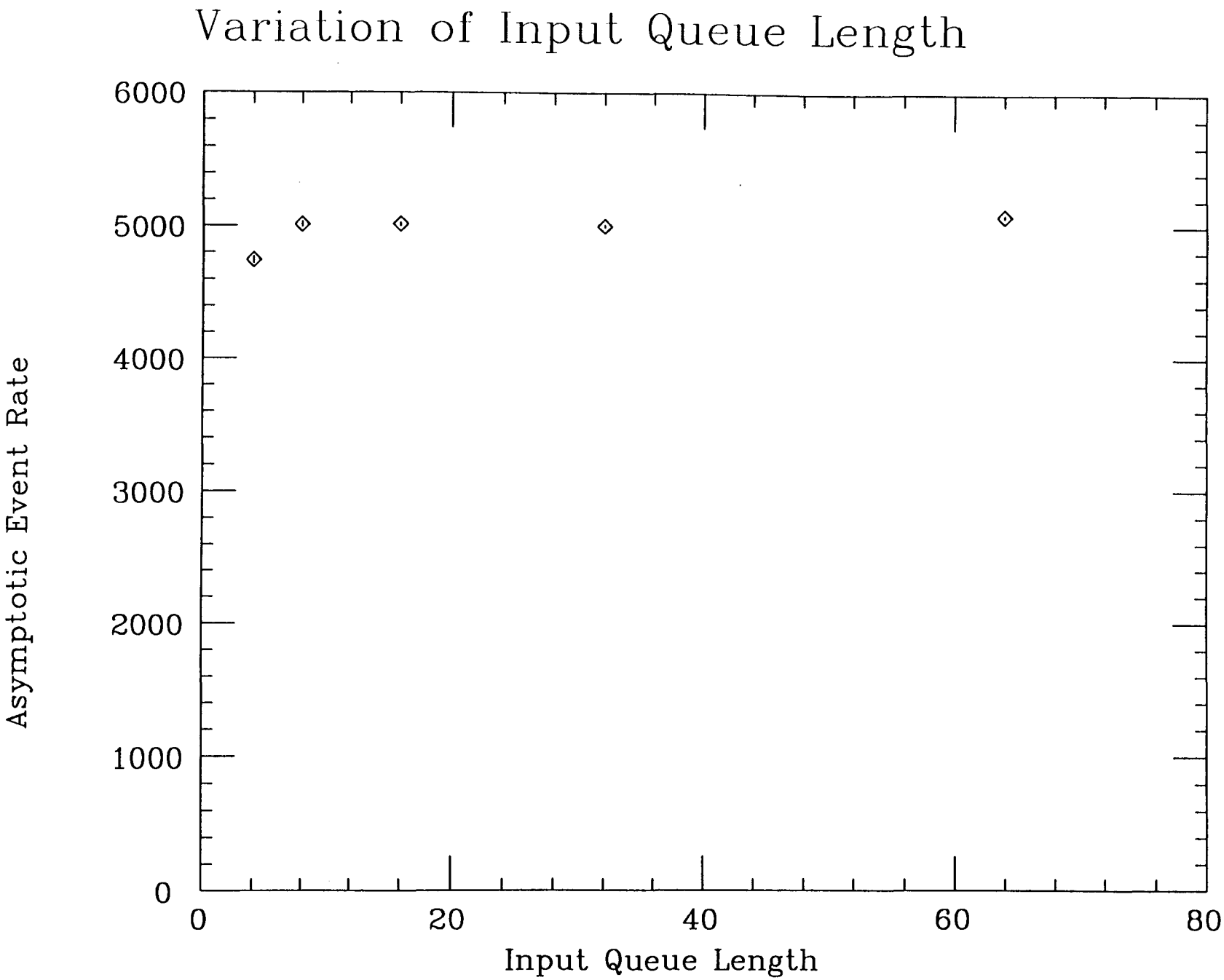

Fit Dead Rate vs Trigger Rate
Dead Event Rate (Hz)
Trigger Rate (Hz)
0
1000
2000
3000
4000
5000
0
2000
4000
6000
8000
10000

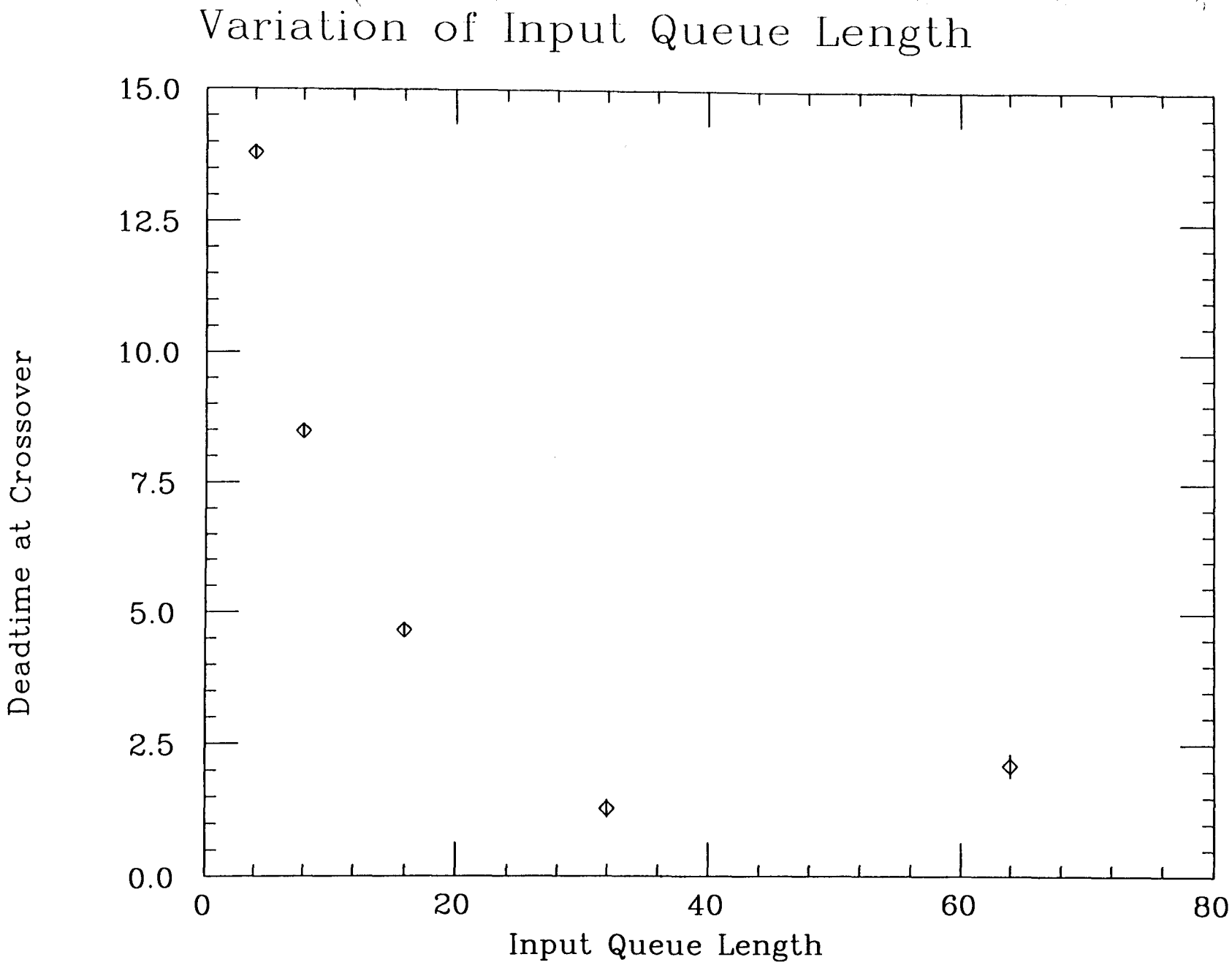

Baseline Parameters
Deadtime (%)
80
60
40
20
0
0
1000
2000
3000
4000
5000
Event Rate (Hz)

Event Generator Parameters

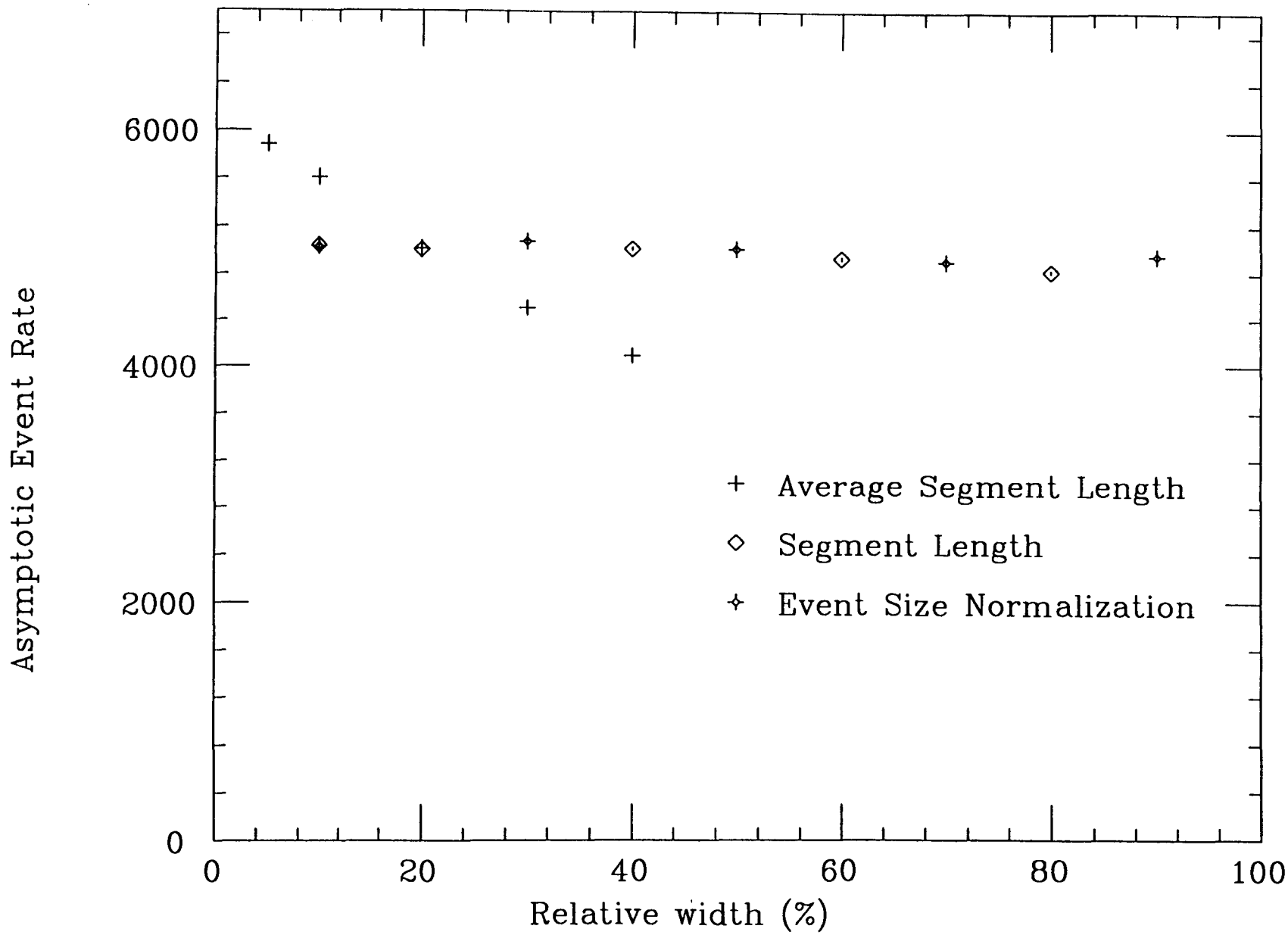
- Varying the number of input queue elements strongly affects deadtime at crossover but has no significant effect on the asymptotic event rate
- Increasing the variation in average data length among data sources reduces the throughput
- Changing the fluctuations in data length has no significant effect


Variation of Input Queue Length
4 Events
8 Events
16 Events
Deadtime (%)
Event Rate (Hz)
0
20
40
60
80
0
1000
2000
3000
4000
5000
6000


Variation of Input Queue Length
Asymptotic Event Rate
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
Input Queue Length


Variation of Input Queue Length
Deadtime at Crossover
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
Input Queue Length

Variation of Event Generation Parameters



Bandwidth Scaling

The asymptotic event rate can be predicted based on a simple model of the DAQ bandwidth:

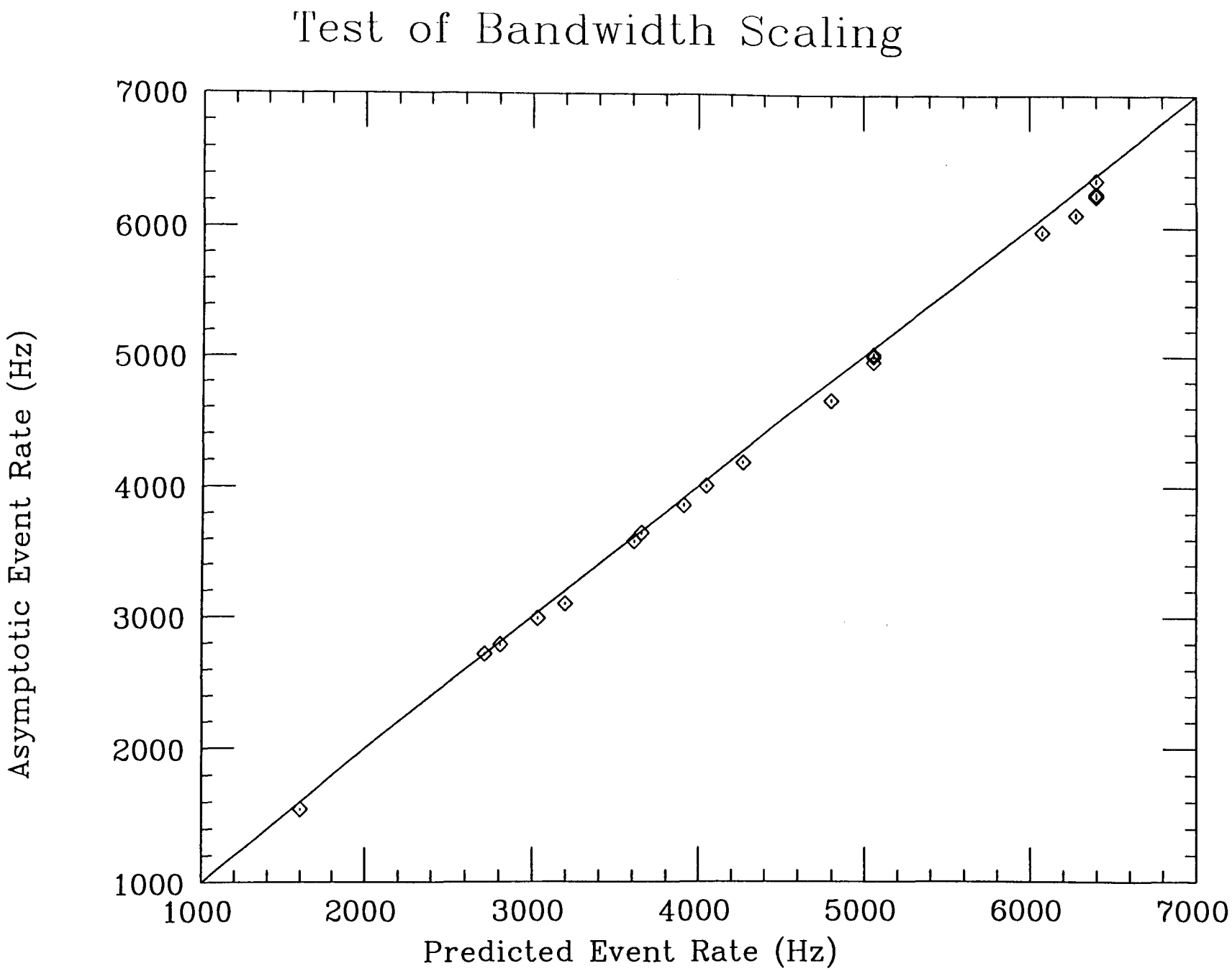
Event rate is limited by bandwidth from data source,

$$ER \leq \frac{BW}{\langle Data_Length \rangle_{max}}$$

or by CPU capacity,

$$ER \leq \frac{N_CPU}{\langle CPU_Time \rangle}$$

Comparison of predicted event rates and asymptotic event rates from simulation gives good agreement


Test of Bandwidth Scaling
Asymptotic Event Rate (Hz)
Predicted Event Rate (Hz)
1000
2000
3000
4000
5000
6000
7000

Latency in the Event Builder

To simulate architectures with significant latency in event building, we modified the simulation program as follows:

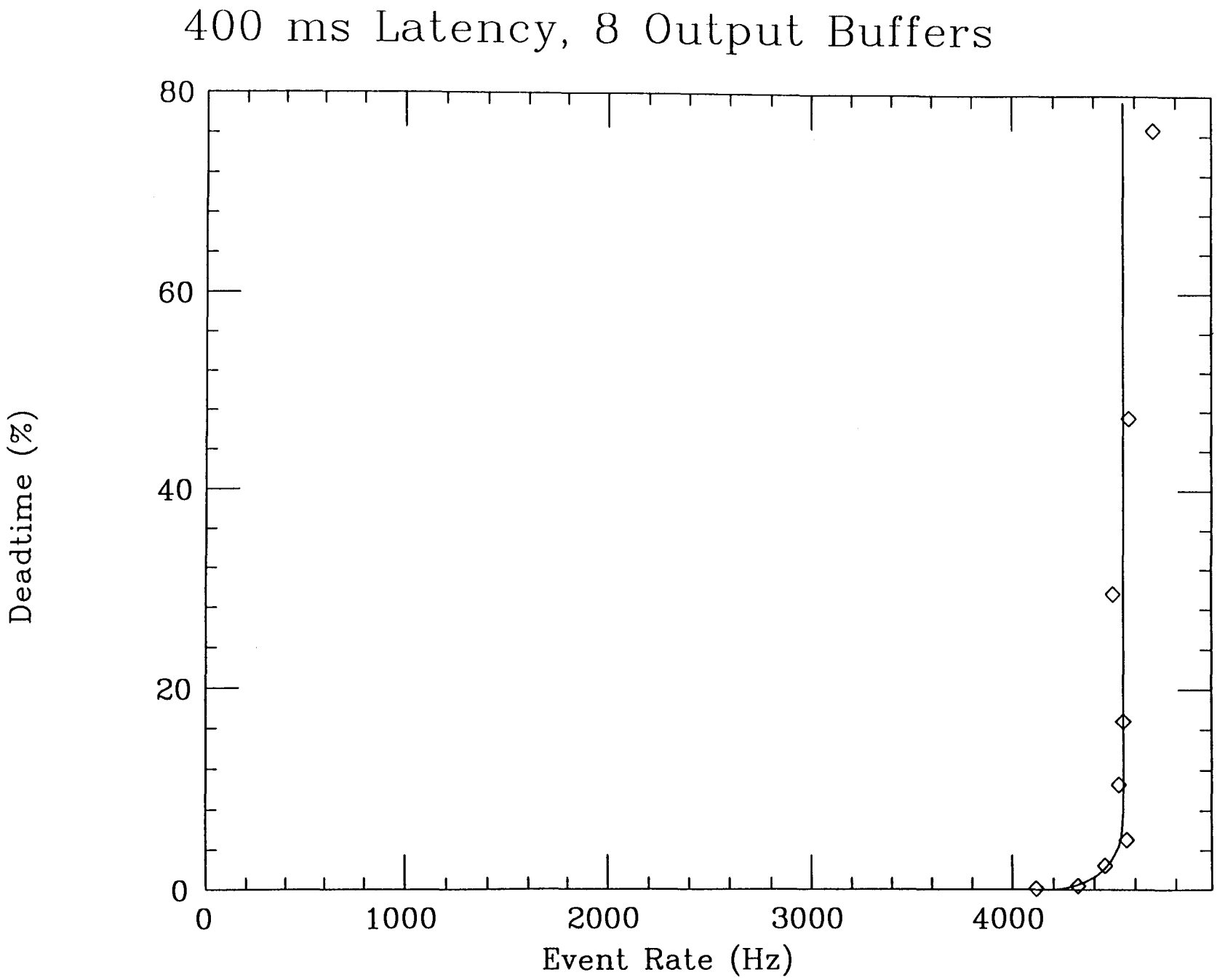
- Added random latency time between end of data transmission and arrival of data at CPU buffer
- Incorporated a set of event buffers in front of each CPU to hold data as it makes its way through the event builder
- CPU processes events in order of generation
- Uniform distribution of latency times
- Deadtime incurred if no CPU buffer available

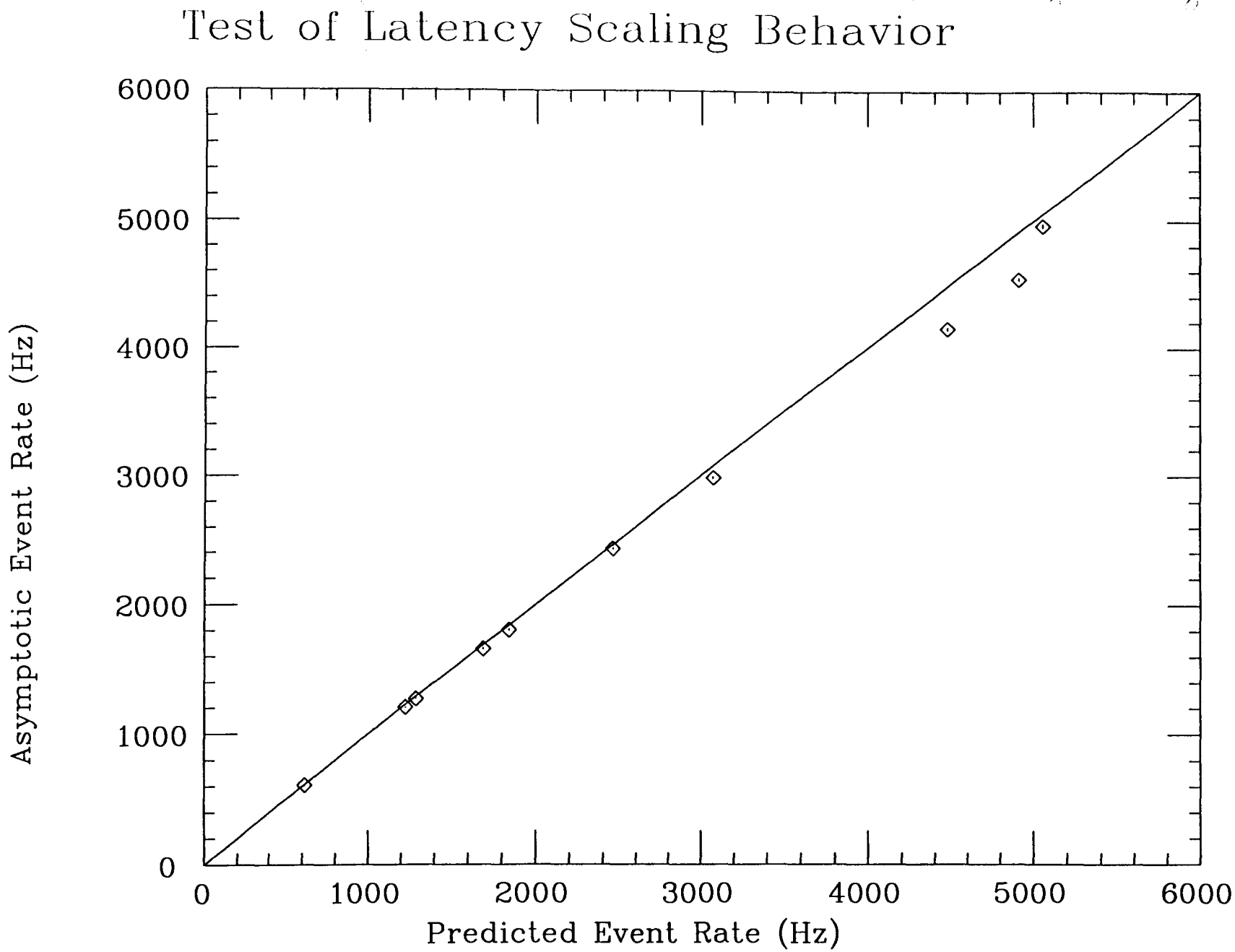
New scaling law applies:

$$ER \leq \frac{N_CPU \cdot N_Buf}{\langle Max._Latency \rangle}$$

For a uniform latency distribution,

$$\langle Max._Latency \rangle = \frac{2N}{N+1} \langle Latency \rangle$$


400 ms Latency, 8 Output Buffers
Deadtime (%)
80
60
40
20
0
0
1000
2000
3000
4000
Event Rate (Hz)


Test of Latency Scaling Behavior
Asymptotic Event Rate (Hz)
Predicted Event Rate (Hz)
0
1000
2000
3000
4000
5000
6000

Conclusions

- Simulations performed for a variety of model parameters
- Parameters that effect throughput or deadtime of generic architecture have been identified
- Scaling laws established
- DAQ architectures with large latency times simulated

Simulation Studies of the SDC Data Collection Chip

E. Hughes[α], G. Tharakan[β], M. Anania[α], R. Dames[α], S. Danz[α], R. Downing[γ], M. Haney[αγ], E. Golin[α], E. Hoffmann[α], L. Huang[α], L. Jones[α], S. Larison[α], S. Raman[β], D. Miller-Karlow[α], J. Thaler[γ]

[α] University of Illinois Department of Computer Science, 1304 W. Springfield Ave., Urbana, IL 61801 USA

[β] University of Illinois Department of Electrical Engineering, 1406 W. Green St., Urbana, IL 61801 USA

[γ] University of Illinois Department of Physics, 1110 W. Green St., Urbana, IL 61801 USA

Abstract

This paper describes simulation studies of the Data Collection Chip (DCC) design for the Solenoidal Detector Collaboration (SDC)[1]. The DCC assembles event information from a number of input channels. A non-overlapping tree structure of DCCs can be used to build large event fragments. Modeling and simulation studies are described which examine the effect of DCC design parameters on the deadtime of the detector.

In the deadtime study, three possible DCC architectures are examined in detail, each of which uses a different protocol for communication. The relative effects on trigger throttling and data loss are examined.

In the correlation study, the effect of input data correlation on buffer requirements for the DCC is examined. This study addresses the interdependence of input data, which is an important concern. The results suggest a linear relationship between the correlation of data on input channels and the buffer space needed for normal operation.

Based on our studies of the DCC and the requirements of the SDC data acquisition system (DAQ), the design of a cascadable, fault tolerant queue is described.

In the multilevel study, a flexible model of the DCC is developed. The model includes a *protocol* parameter, and can be used to construct arbitrary trees of DCCs. The results demonstrate the feasibility of a parametric DCC design for the SDC DAQ. The DCC design can accommodate different data formats and data rates, and is appropriate for a variety of detector subsystems.

I. INTRODUCTION

A. The SDC Data Collection Chip

The Solenoidal Detector Collaboration (SDC) Data Acquisition System (DAQ) is an ongoing design that is intended to be part of the detector system for the Superconducting Supercollider (SSC). The Solenoidal Detector is a general purpose system that is optimized for high-p_t physics. The expected operating luminosity of this facility is 10^{33} $cm^{-2}s^{-1}$, and the detector is designed to address specialized issues at higher luminosities. The SDC DAQ collects digital data from the Front End systems (FEs) on the detector and moves the data to stable storage.

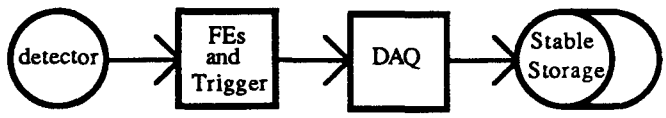


Figure 1. The SDC architecture

The Data Collection Chip (DCC) reads detector data from the Front End (FE) and Trigger subsystems. A non-overlapping tree structure of DCCs is used to collect data from approximately $8 * 10^6$ detector channels [SDC92] and send the data to an array of microprocessors. A flexible DCC design has been developed to meet the needs of the DAQ. The DCC reads data from a variable number of sources, and can be optimized for each detector subsystem without significantly affecting detector data.

B. Modeling and Simulation

Designers of large electronic systems use modeling and simulation to improve the reliability of a system while reducing its design time. The *model* of a system is a partial description of the design that can be simulated to check correctness of behavior and compare alternatives. A system can be described at several levels, corresponding to the amount of detail in the parts of the description. For example, *behavioral* models describe a system in an abstract manner, and *structural* models use a detailed interconnection of components [Hugh90]. The *simulation* of a model is the execution of the model for a given set of inputs, called the *stimulus*. DCC models have been written in *Verilog* [Vlog91] and *VHDL* [VHDL88]. Both languages provide for mixed-level model descriptions.

C. Assumptions and Parameters

A block diagram of the SDC DAQ is given in Figure 2. In this architecture, a tree of DCCs gathers data from a large portion of the detector and groups the data by *event*. In this context, an event is a group of particles resulting from a collision of particles. Resulting data is sent to an array of

[1] Supported in part by DOE contract: DE-AC02-76ERO1195 and by TNRLC contract: RGFY9147.

processors, which store interesting events on magnetic media. This diagram includes only data flow, omitting trigger and control signals.

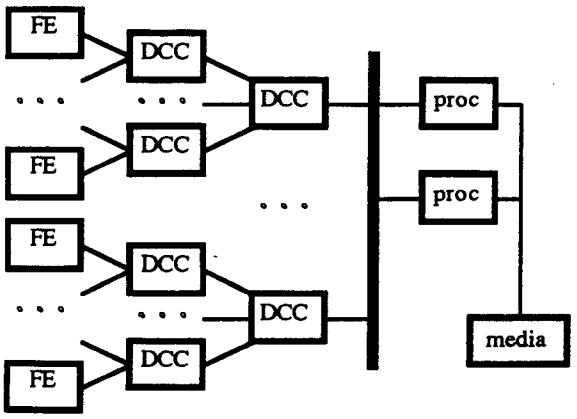


Figure 2. SDC DAQ architecture

The DCC model uses a queue to store data from each source and a *workspace* to store assembled events. In addition, each FE model has a queue for storing detector data. The queue sizes can be adjusted by the model parameters fe_qsize, dcc1_qsize, and dcc2_qsize. In the block diagram, dcc1 parameters apply to the left-most DCCs and dcc2 parameters apply to the second layer of DCCs. In addition, the maximum data rates for the data connections are controlled by the fe_rate, dcc1_rate, and dcc2_rate parameters.

The models discussed in this paper use randomly generated input data. The parameters guiding data generation are derived from preliminary data for several detector subsystems [SDC92]. A uniform distribution is used to generate *L2accepts*, which are occurrences of interesting events. The average number of L2accepts per crossing is given by the L2_rate parameter. Next, the FE model generates a data packet for each L2accept based on the number of data channels in each FE (num_chan), the fraction of active channels (occupancy), and the amount of data for each active channel (event_size).

The SDC DAQ will have a large data throughput. In this work, we assume that the DAQ will require data to be transferred without handshake during normal operation. The DCC model includes two kinds of control signals, *busy* and *throttle*, which warn of impending overflow. A busy signal is asserted when one of the input queues exceeds a preset level, given by the dcc1_busy or dcc2_busy parameters. The throttle signal is asserted when any of the queues exceed the dcc1_thr or dcc2_thr throttle levels. Typically, the throttle signal represents a more urgent situation than the busy signal. Finally, the daq_wait parameter models the delay between events read from the DCC.

The DCC model includes a "processing" delay for assembled events, given by dcc1_proc and dcc2_proc. The number of FEs connected to a DCC is given by num_FEs and the number of first-level DCCs is given by num_DCCs. The fe_delay, dcc1_delay, and dcc2_delay parameters model the propagation delay of control feedback signals.

The sim_time parameter specifies the amount of detector time to simulate. In addition, the clock rate of the system is controlled by clock_pd. Typical values for the parameters are given in Table 2. In the simulation experiments, parameters will take nominal values unless otherwise specified.

clock_pd	16 ns	num_chan	12
L2_rate	5.5 E-5	num_FEs	16
occupancy	10%	num_DCCs	12
event_size	5 words	fe_delay	340 ns
fe_qsize	160 words	dcc1_delay	16 ns
dcc1_qsize	80 words	dcc2_delay	70 ns
dcc2_qsize	160 words	fe_busy	40 words
fe_rate	180 KB/sec	dcc1_busy	20 words
dcc1_rate	3.0 MB/sec	dcc2_busy	40 words
dcc2_rate	18 Mb/sec	dcc1_thr	15 words
dcc1_proc	100 ns	dcc2_thr	20 words
dcc2_proc	500 ns	daq_wait	100 μ.sec

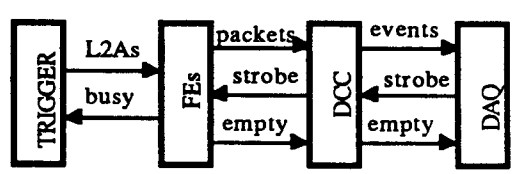
Table 2. Nominal values of parameters.

II. DEADTIME STUDIES

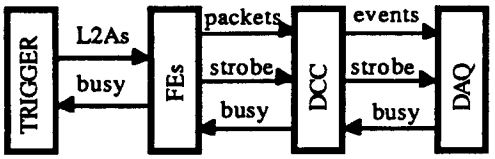
A. Goals

Abstract simulation studies of the DCC began with VHD' and Verilog models [Hugh92]. The Verilog model was used compare candidate architectures, with different values of DC design parameters, for *deadtime*, which is the fraction of tin. the detector cannot take data. In this study, two phenomen. result in deadtime: (1) *throttled triggers*, which are suppressed by control signals to the TRIGGER, and (2) corrupted events due to queue overflow. For this study, the occupancy parameter was set to 32% to cause a statistically significant number of corrupted and throttled events.

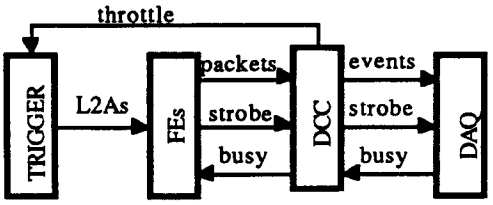
Figure 3 gives the architectures of the models compared in this study. In this figure, the 16 FE models are represented by a single block. The TRIGGER generates L2accept signals (L2As). The FEs generate variable-sized packets, which are assembled into events by the DCC. A DAQ model is included to communicate with the DCC.



(a) pull architecture



(b) ripple architecture



(c) direct architecture

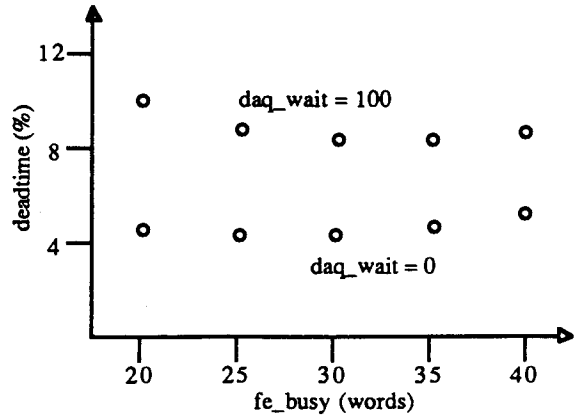
Figure 3. Model architectures for deadtime study.

In the pull architecture, the FE-DCC data link is controlled by the DCC. The FE asserts the empty signal to warn of impending queue underflow. The architecture is called pull because the receiver controls the data flow. In the ripple and direct architectures, the FE controls the FE-DCC link and the DCC sends busy and throttle signals when overflow is likely. These are both called *push* architectures because the sender controls the data flow. In the ripple architecture, a busy signal is sent to an FE when the DCC queue for that FE is nearly full. In the direct architecture, an additional throttle signal is sent to the TRIGGER when any of the DCC queues are nearly full. The throttle signal contributes directly to deadtime by suppressing triggers (L2As). The DCC busy signal contributes to deadtime indirectly by slowing transmission from the FE.

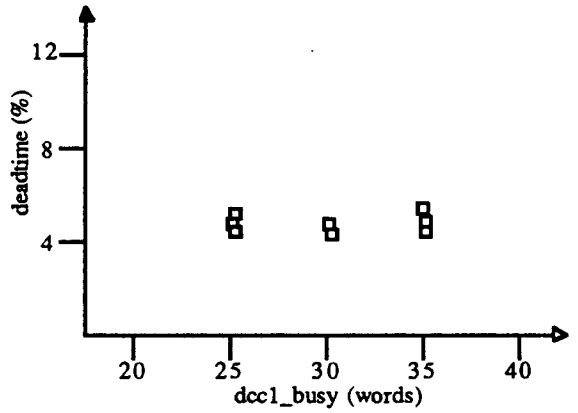
B. Deadtime Study

The first experiment measures the effect of the feedback parameters on deadtime. In this experiment, the busy and throttle depth parameters and the daq_wait parameter are varied. Each point on the graphs is the average of five simulations of two seconds of detector time with different seeds to the random number generator. Each simulation takes approximately four hours on a Sun Sparc station using Cadence's Verilog running under Sun OS 4.1.1.

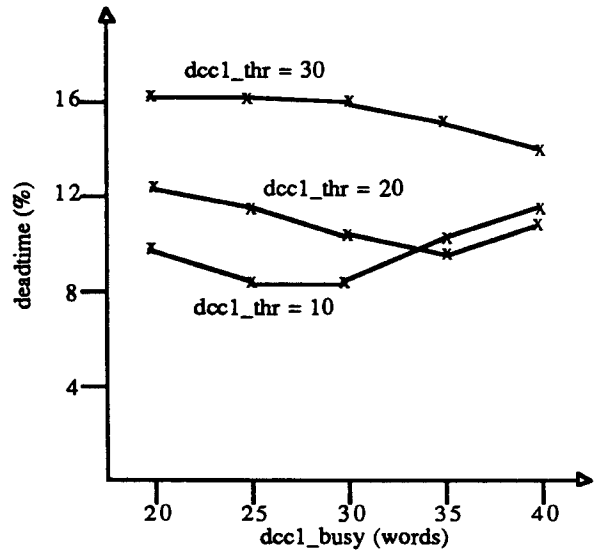
For the pull architecture, fe_busy is varied from 20 to 40 words and daq_wait is either 0 or 100 µsec. The results for this experiment are plotted in Figure 4(a). For the ripple architecture, fe_busy is varied from 20 to 35 words and dcc1_busy is varied from 25 to 35 words. The results are shown in Figure 4(b). For the direct architecture, dcc1_busy takes values from 20 to 40 words and dcc1_thr varies from 5 to 35 words. The results for this architecture are given in Figure 4(c). The curves for dcc1_thr = 5, 15, 25 and 35 have been omitted for clarity. The curve for a throttle depth of 5 words is slightly higher (more events lost) than the curve for 10 words. The composite result is given in Figure 4(d).



(a) pull architecture

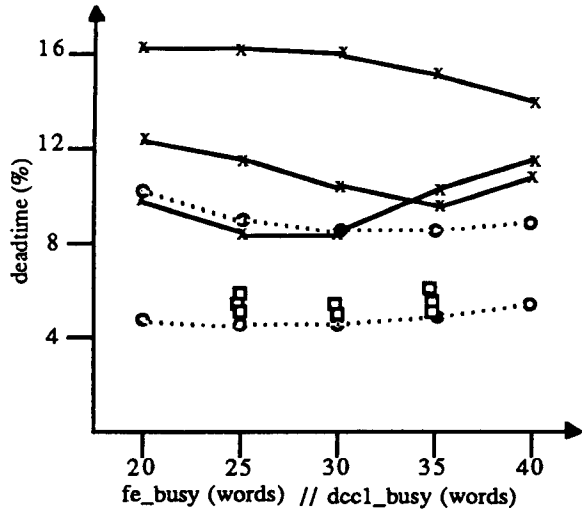


(b) ripple architecture



(c) direct architecture

The results for the pull architecture demonstrate the importance of the daq_wait parameter, which is one measure of the ability of the second level DCC to read out the data of the first level DCCs. A similar test could be applied to the push architectures by randomly setting busy signals from the DAQ model, presumably with similar results. It is interesting to note that the two curves are relatively flat.



(d) composite result

Figure 4. Deadtime results.

The results for the ripple architecture are tightly grouped around 5% deadtime. In this architecture, neither the fe_busy or dccl_busy parameters have a significant effect on deadtime. The components of deadtime in this experiment varied substantially. For example, the deadtime for fe_busy = 25 words and dccl_busy = 25 words is composed of 81% throttled triggers and 19% corrupted events, for which some data was lost. In contrast, the deadtime for fe_busy = 35 words and dccl_busy = 35 words is composed of 99% throttled triggers and 1% corrupted events. This tradeoff is more closely examined in the next section.

The results for the direct architecture vary between 8% and 20% deadtime. The results show a clear tradeoff between throttled triggers and corrupted events. As dccl_thr is increased, dccl_busy should also be increased. The results show that the busy depth should be set about 15 words higher than the throttle depth. The direct architecture will require more buffer or communication resources to handle a given input data rate. One reason for this result is that the direct architecture uses many fewer control signals. For 192 FEs read by 12 first level DCCs and one second-level DCC, the pull and ripple architectures use 396 control signals and the direct architecture uses 217 signals. This experiment also tests the overflow recovery mechanisms of the model. The simulations suggested many test situations which are extremely rare in practice and which can cause the system to deadlock.

The measured deadtime was incurred by one DCC and the associated FEs. Thus, deadtime in excess of 4% should be unacceptable, because the SDC detector will have thousands of DCCs.

C. Reconstruction Study

One weakness of the deadtime study is that the model architectures are only compared with respect to total deadtime. This comparison is not always ideal, because deadtime is composed of corrupted events and throttled triggers. The reconstruction study examines the components of deadtime individually. Realistic physics detectors can reconstruct an event if a small portion of the data has been lost. In this study, the results of the deadtime study are analyzed in detail. The problem is simplified by assuming that the DCC can rebuild an event if the amount of data lost is less than a constant. In a real detector, the distribution of lost words would affect the ability of the DAQ to reconstruct an event. The results for this study are given in Figure 5.

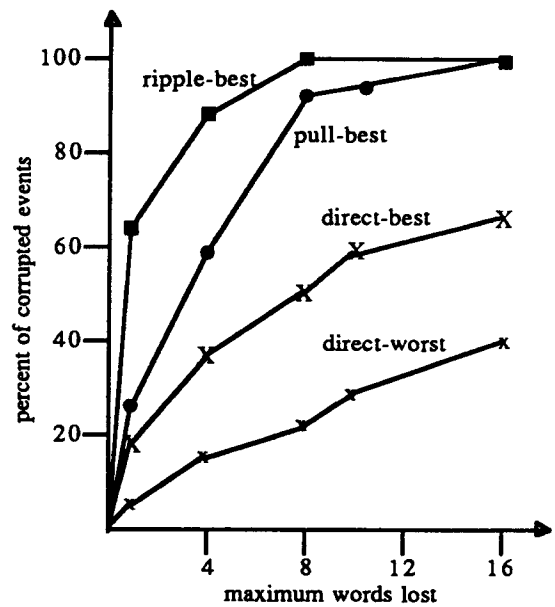


Figure 5. Reconstruction results.

In this study, the average size of a complete event in the DCC is 307 words. For each architecture, the best results are reported. In addition, the worst result for the direct architecture is included. For example, when the reconstruction algorithm was applied to one of the ripple simulations, over 60% of the corrupted events can be reconstructed if the constant limit is one word per event. The worst result for the ripple architecture is slightly worse than the best result for the pull architecture. Reconstruction is most effective for the ripple architecture, which lost just one word of many of the events. The direct architecture lost large amounts of data, and less than 20% of the events can be reconstructed with a limit of one word lost.

D. Extendibility Study

In the deadtime study, the resulting curves for the pull and ripple architectures are relatively linear, while the curves for the direct architecture are not. One possible explanation is that an architecture which is relatively capable of handling the input data rate (i.e. pull and ripple) is mostly insensitive to the parameters, while an overtaxed architecture (i.e. direct) is not insensitive to DAQ parameters. If true, this would have important implications for upgrading the operating luminosity of the SDC.

In an attempt to validate the above hypothesis, the ripple architecture was simulated with a 30% higher input data rate. The result was a set of curves varying from 16.5% to 19.3% deadtime. These curves are not linear, so optimization is important. A remaining question is: Will both an increase in input rate and a decrease in buffer space have the same effect? Another experiment was run with 12.5% less buffer space in both FEs and the DCC. The result was a set of curves varying from 6.9% to 8.0% lost events, which are not as linear as the curves around 5% but are more linear than the curves above 10%, as expected.

III. CORRELATION STUDIES

A. Goals

The previous studies assume a uniform distribution of data among the FEs. Each channel has the same probability of having data for an event. Situations can arise where one Front End has much more data than the others for an event even though the average amount of data entering the DCC is the same. Such situations arise when there is a strong correlation or dependency between the channels in a Front End. Due to the spatial location of the channels on the detector, an event that causes a particular channel to be active may activate many neighboring channels. These channel dependencies produce a skewed distribution of data among the FEs. This study addresses these issues and determines the effect of correlation on queue space in the DCC.

B. Experiments and Results

The distribution of data among the FEs, for two different amounts of correlation, is shown in Figure 8. In this figure, there are two parameters of interest, the *probability* that an FE has active channels for an event and the *channel dependency* (fraction of channels in an FE that are active for any event). Hence, a 100% channel dependency implies that all twelve channels of an FE are active for any event. The product of these two quantities (*channel occupancy*) is kept constant so that the average amount of data entering the DCC per event is unchanged. The data is localized in a few FEs when the channel dependency is high, but is distributed evenly among many of the FEs when the dependency is low.

Figure 9 shows a plot of the maximum DCC queue depths as the DCC *data rate* (dcc1_rate) is varied. The experiment is performed for two values of *FE data rate* (fe_rate) and two values of channel dependency. Two important observations can be made from the figure. First, the maximum queue depth increases as the FE data rate is decreased. When the input rate is low, the DCC takes more time to assemble a complete event, which causes queues that contain larger events to grow deeper. Second, the queue depth increases as the channel dependency is increased. The second observation is explained with the help of Figure 8. When the channel dependency is high, the data for an event is concentrated in a few FEs, which increases the depths of the corresponding queues of the DCC.

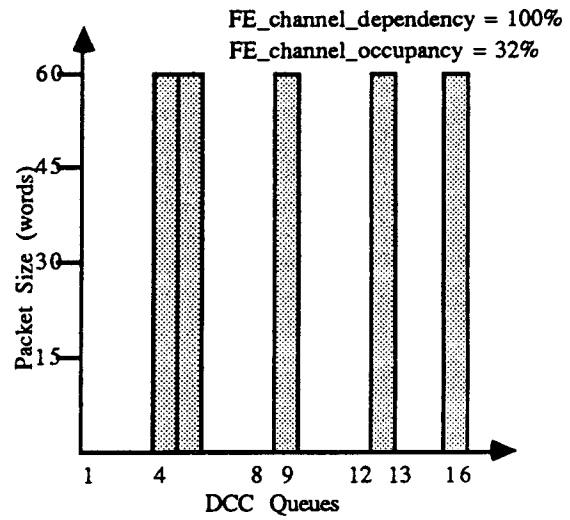


Figure 8a. Data Distribution for a typical event (high dependencies)

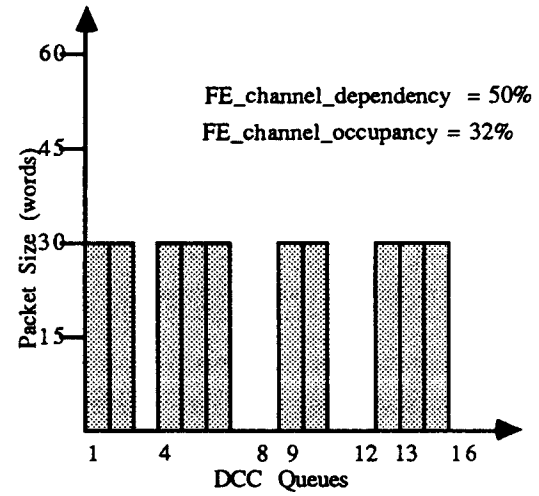


Figure 8b. Data Distribution for a typical event (low dependencies)

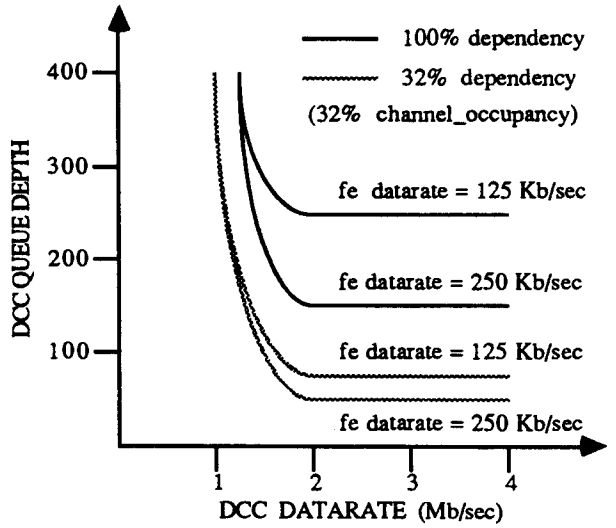


Figure 9. Maximum DCC queue depths.

A plot of the maximum depth of the DCC queues as the dependency is varied is shown in Figure 10. This result explains the effect of channel dependency on queue depths. The input and output data rates are kept constant and the experiment is performed for two values of channel occupancy. The DCC queue depth varies almost linearly with channel dependency. Also, the queue depth is increased when the channel occupancy is higher. This is because the total amount of data entering the DCC per event is proportional to the channel occupancy.

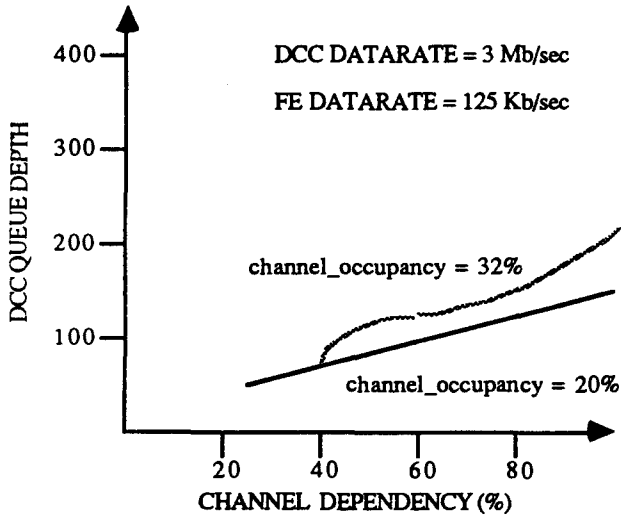


Figure 10. Queue Depth Vs Dependency

From the above discussion, it can be seen that moderate data dependencies cause a skewed distribution of data among the queues in the DCC. All of the event data is concentrated in a few queues while the remaining are mostly empty. This leads to poor utilization of resources and can cause queue overflow. The design of a queue which addresses some of these shortcomings is proposed below. Since the queue is a very crucial and widely used module in data acquisition systems, this design could also be incorporated into other subsystems of the SDC.

IV. DESIGN OF A CASCADABLE QUEUE

A. Requirements of the DCC Queue

The design of a queue for the data acquisition system of the SDC must satisfy the following requirements:

(i) it must be highly fault tolerant

(ii) it must be easily testable

(iii) it must be very flexible, to fit different subsystem requirements

(iv) it must be easily reconfigurable in "real time", to prevent loss of event data

(v) it must be fast (high throughput and low latency)

Fault tolerance can be achieved in three ways: first, the system can be built using "gold" components, second, triple modular redundancy (TMR) techniques can be applied, and third, the system can be tested frequently to detect faulty components. Of the three techniques, the first is impossible to implement with current technology and the second causes a tremendous waste of system resources. If the system is designed to be easily testable, very few test vectors will be needed to detect and locate faults. Hence, the third technique can result in high utilization of silicon area with low system downtime and is therefore best suited to SDC requirements. The DCC must be very flexible because it communicates with numerous subsystems having different characteristics. Specifically, the queues must be designed such that their depth and/or width can be easily varied. Reconfigurability must be built into the queue design for fault tolerance requirements and to accommodate bursts of data from the Front Ends. Finally, all these requirements must be attained with minimal loss of performance.

B. Possible Designs

One possible way to satisfy the first four requirements with minimal waste of silicon area is to break up the queue into smaller *subqueues*. The subqueues are then cascaded together according to system requirements to build a larger queue. Two commonly used cascadable queue designs are described below.

1) SERIAL SHIFT REGISTER DESIGN

A serial shift register design is one of the simplest implementations of a queue. A diagram of this design is given in Figure 11. The data is strobed into the shift register by the *strobe in* signal and is strobed out by the *strobe out* signal. Each input or output strobe causes the data in the register to be shifted towards the output by one slot. The *input ready* and *data ready* signals are useful for cascading the registers to increase the depth of the queue. One obvious drawback of this design is that the data is moved from one location to the next in the queue, which reduces throughput and increases latency.

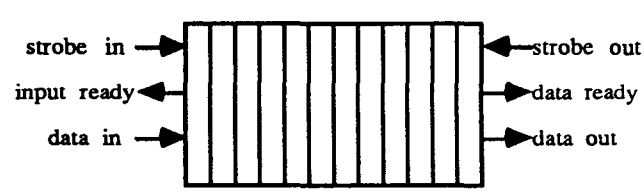


Figure 11. Shift Register Design

2) RAM BASED DESIGN

RAM based queues eliminate the drawback of the previous design by changing address pointers rather than moving data. A diagram of this design is shown in Figure 12. The data is stored in a RAM which is addressed by the read and write

pointers. In this design, each output or input strobe causes the read or write pointers respectively to be incremented. The empty and full signals are asserted when the RAM is empty or full. The queues can be cascaded to increase the depth and/or width with the help of the handshake signals. This is shown in Figure 13. Two factors limit data throughput in this design: (1) the physical movement of data from one queue to another when two or more queues are cascaded, and (2) the handshaking mechanism between the queues. Both drawbacks are eliminated by the design proposed in the next section.

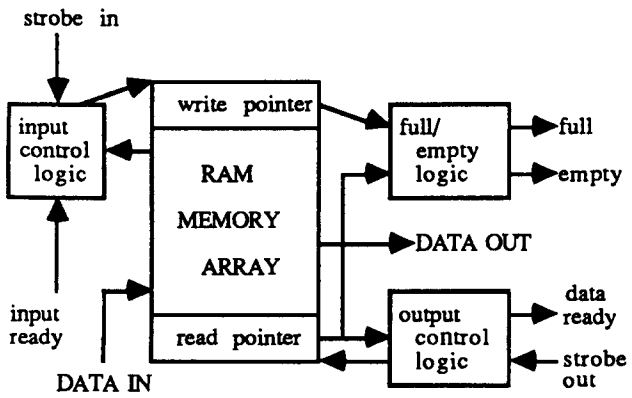


Figure 12. RAM BASED DESIGN

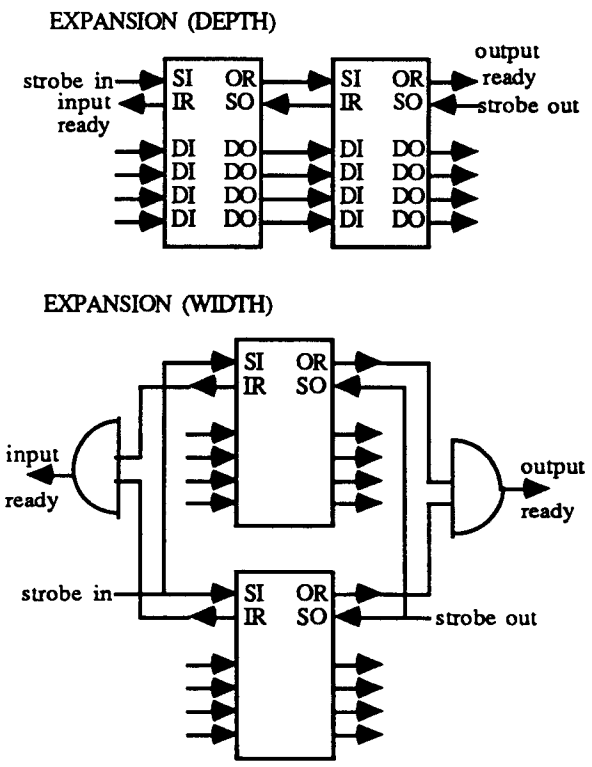


Figure 13. Queue Expansion

C. A New Design

The proposed design is very similar to the RAM based design. The most important difference between the two designs is the way queues are interconnected to make a deeper queue. In the previous design, queues were cascaded *serially* to increase the depth which caused the data to move from one queue to another. In the proposed design, queues are connected in *parallel* to increase their depth or width. Hence, data does not move from one queue to another and no handshake signals are required. Figure 14 shows how two queues are cascaded to increase the depth or width. The input data bus is connected to all the queues in the cascade and the outputs of the queues are tied together. Two signals are required for cascading purposes. The *FS* signal indicates the first queue in the cascade. The *Xout* signal of a queue is connected to the *Xin* signal of the following queue. The Xout signal produces a pulse each time the queue has been completely emptied or completely filled. This pulse causes the transfer of read/write activity to the following queue in the cascade. Both FS and Xin are tied high when the queues are cascaded to increase the width.

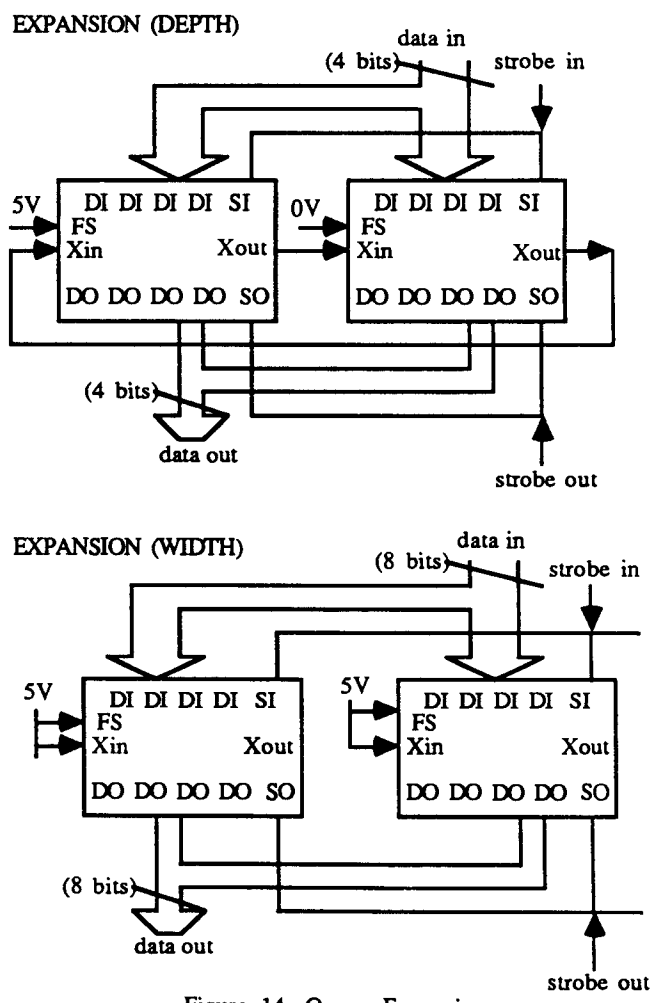


Figure 14. Queue Expansion

A block diagram of the design is shown in Figure 15. The *read & write active logic* generates the *RA* and *WA* signals which are asserted when the queue is being read from or written to, respectively. The *full/empty logic* indicates when the queue is empty or full. Both logic blocks are implemented by finite state machines (FSMs). These FSMs are clocked by two input signals: *strobe in* (SI) and *strobe out* (SO). The implementation of a two-clocked FSM is shown in Figure 16. This implementation requires the following steps to be performed. First, the state transition diagram (STD) of the FSM is drawn showing all the transitions due to both the clocks. Second, from the STD, two state transition diagrams are generated showing the transitions due to each clock. Finally, the combinational logic to implement each STD is built separately and interconnected as shown in Figure 16.

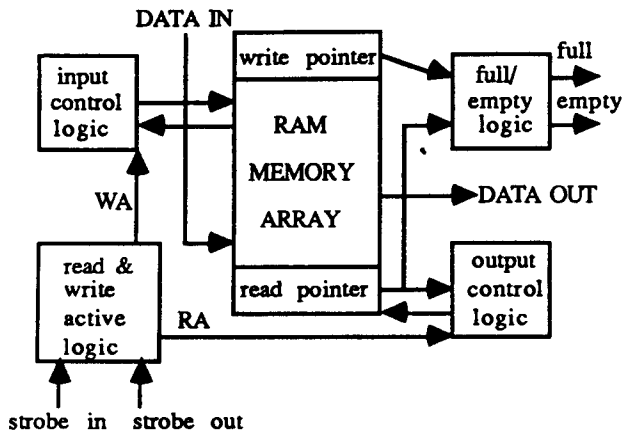


Figure 15. THE NEW DESIGN

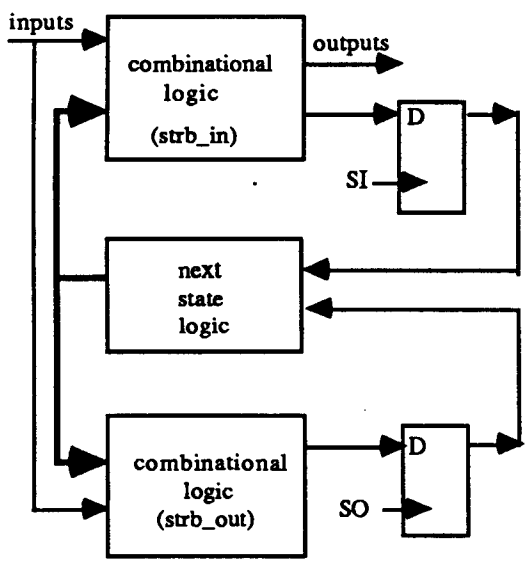


Figure 16. State Machine with 2 clocks

The queues can be reconfigured very easily and quickly because only one signal is required to connect the queues in a cascade (from Xout to Xin). This technique can be used to increase flexibility and fault tolerance with minimal loss in speed. The flip-flops used in the FSMs, the read and write counters and the input and output data registers can be connected in a scan path for testability purposes. The proposed queue design allows the buffer space of the DCC design to be changed with minimal side effects on the timing of the DCC's behavior.

V. MULTILEVEL DCC STUDIES

A. The Model

One problem with the previous studies is that the DAQ model is not a realistic representation of a second level DCC. This study addresses the problem by incorporating two or more levels of the DCC network into the model. The multilevel model improves upon the previous models by including a GATING component, which decides when to throttle L2accepts. A diagram of this model is shown in Figure 17.

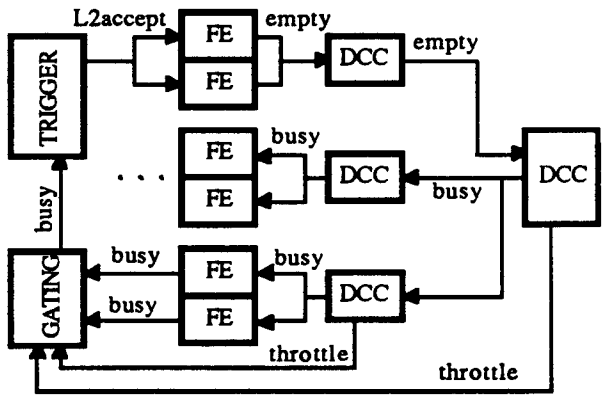


Figure 17. Multilevel DCC control flow.

In the block diagram, only some of the connections are shown. In the multilevel model, each FE component receives the L2accept signal from the TRIGGER model and returns a busy signal to the GATING model. In addition, each DCC receives empty signals from each source and a busy signal from the next DCC, and returns busy signals and a throttle signal. A single GATING component models the overall throttle decision.

The FE/DCC interface is based on the architectures used in the deadtime study. The multilevel model includes a *protocol* parameter, which is used to select pull, ripple, or direct. Figure 17 shows all the control signals (empty, busy, throttle) necessary for the different values of protocol. For example, if the protocol is pull, then the empty signals are used and the busy and throttle signals from the DCC are not used. The FE busy signals are used for all protocols.

The multilevel DCC study also improves upon the previous models by incorporating realistic parameters for the delay of control feedback signals. The proposed SDC detector will be a cylinder about 50 meters in length and about 20 meters in diameter [SDC92]. Thus, the potential propagation delay of a signal includes a nontrivial component for transmission over cable lengths. First, the detector is

approximated by a bounding rectangular solid. Next, we assume that the propagation delay for a signal is 10 nsec plus 6 nsec/meter of cable. An FE is assumed to be within one meter of the corresponding DCC, resulting in 16 nsec delay. The longest path from any point on the detector to the central GATING component is 55 meters, resulting in 340 nsec delay. Finally, we assume that the second level DCCs are within ten meters of the corresponding first level DCCs and within 45 meters of the GATING, resulting in delays of 70 nsec and 280 nsec, respectively.

B. Modeling Techniques

A VHDL model of the DCC was constructed for this study. This model uses a record type to represent packet information, which simplifies changes to the packet definition. The models are *parametric*, so that readout structures of arbitrary size can be built without changing the DCC or FE models. The propagation delays of control signals are modeled by VHDL signal assignments with *transport* delays. This technique allows signal delays to be changed without affecting the number of visible signal pulses.

The most important characteristic of the multilevel model is the use of a parameter to select the communication protocol. In this study, the FE and DCC models adhere to the protocol parameter by using selected processes for communication and flexible processes for data storage.

In the first technique, the protocol parameter is used to *generate* different processes for the pull and push protocols. The model is efficient, because unused processes are not included during model compilation (much like macro expansion).

In the second technique, the process which stores data in a DCC queue asserts the throttle signal only if the protocol is direct, and asserts the empty signal only if the protocol is pull. This technique reduces the size of the model at the expense of minor inefficiency.

An important characteristic of the multilevel model is that control signals which are not relevant for the chosen protocol never change value. This reduces the run time of the simulation by reducing the number of signal changes to be handled. This technique also simplifies the results to be interpreted by the designer.

VI. CONCLUSIONS AND FUTURE WORK

A. Summary

We have demonstrated a top-down approach to the design of large, complicated systems like the data acquisition system of the SDC. A high level specification is first developed, debugged, simulated, analyzed, and then used to judge the correctness of the lower level implementation. A behavioral model of the DCC was built whose behavior could be varied with simulation parameters. High level studies were performed on this model to understand the behavior of the system and to suggest which components must be optimized for throughput and reliability. Studies to determine the resource requirements (queue depths) were conducted for various scenarios. The effect of channel dependencies on the maximum queue depths of the DCC was also studied. From these studies, a detailed design of the DCC queue was begun. The queue was designed to be cascadable, testable and fault tolerant. It was also built to be very flexible and easily reconfigurable so that it could communicate with a wide variety of detector subsystems. The design was verified by extensive simulations using the Vantage VHDL Spreadsheet [Vant91].

The models presented in this work demonstrate many useful modeling strategies. Key features of the simulations are controlled by parameters, which are easy to adjust. With VHDL, it is even possible to choose a communication protocol by setting a parameter. The model hierarchy matches the design, so models can be reused, especially in mixed-mode simulations.

B. Future Work

One important requirement for the DCC design is preliminary testing with existing Front End systems. This effort includes both simulations and prototypes. We have begun a collaboration with one FE group at Pennsylvania [VanB92]. This kind of experiment is a natural predecessor to future SDC DAQ test beam experiments.

In the near future, we intend to simulate the detailed DCC design to study the fault tolerance of the queues. Real time adaptive reconfigurability techniques to prevent loss of data in the queues will be studied. Structural design of the other modules in the DCC will be done and a prototype of the DCC will be built and tested.

In addition, the abstract VHDL DCC models will be used in simulations with other DAQ models. For example, Ancor has developed a model of the FDDI which could be used to read out data from the DCC model.

Finally, we hope to investigate some specialized design issues for the DCC. First, detailed models can be used to examine the failure modes of the DCC tree structure. Next, abstract models can be used to evaluate techniques for *logging* non-physics detector data. These studies will give a good degree of confidence in the DCC design before expensive prototypes are built.

VII. REFERENCES

[GoRe90] E. Golin and S. Reiss, "The Specification of Visual Language Syntax," *Journal of Visual Languages and Computing*, 1, pp. 141-157, 1990.

[Hoff92] E. Hoffmann, "Implementation of an Incremental CAD Environment," MS Thesis, University of Illinois at Urbana-Champaign, 1992.

[Hugh90] E. Hughes, "A Behavioral Model of the CDF Muon Trigger," MS Thesis, University of Illinois at Urbana-Champaign, 1990.

[Hugh92] E. Hughes, et. al., "Modeling and Simulation of the SDC Data Collection Chip," IEEE Trans. Nucl. Sci., Apr. 1992.

[Jone89] L. Jones, "Fast Incremental Netlist Compilation of Hierarchical Schematics," *Proceedings of the 1989 IEEE International Conference on Computer-Aided Design* (November 1989), pp. 326-329.

[Mill92] D. Miller-Karlow, "The Design and Development of Visual VHDL," MS Thesis, University of Illinois at Urbana-Champaign, 1992.

[Rudn90] E. Rudnick, "The Bluestem Design Framework," MS Thesis, University of Illinois at Urbana-Champaign, 1990.

[SDC92] "Solenoidal Detector Collaboration Technical Design Report," G. Trilling, ed., Draft SDC-92-201, April, 1992.

[Thar92] G. Tharakan, "Design of a flexible Data Collection Chip for the SDC," MS Thesis, University of Illinois at Urbana-Champaign, 1992.

[VanB92] R. Van Berg and T. Ekenberg, personal communication, March, 1992.

[Vant91] Vantage Spreadsheet, Release 3.1. Vantage, July 1991.

[VHDL88] IEEE Standard 1076-1987, "IEEE Standard VHDL Language Reference Manual," IEEE Standards Board, New York, March 1988.

[Vlog91] Verilog-XL Reference Manual, Release 1.6. Cadence Design Systems, 1991.

Refer questions to :

Eric Hughes or George Tharakan
441 Loomis Laboratory of Physics
1110 W. Green, Urbana, IL 61801 USA

Phone : (217) 244-8353
Email: {hughes, gth}@eng3.hep.uiuc.edu

CORRELATION STUDIES OF THE DATA COLLECTION CIRCUIT

&

THE DESIGN OF A QUEUE FOR THIS CIRCUIT

By

GEORGE M. THARAKAN

UNIVERSITY OF ILLINOIS AT URBANA - CHAMPAIGN

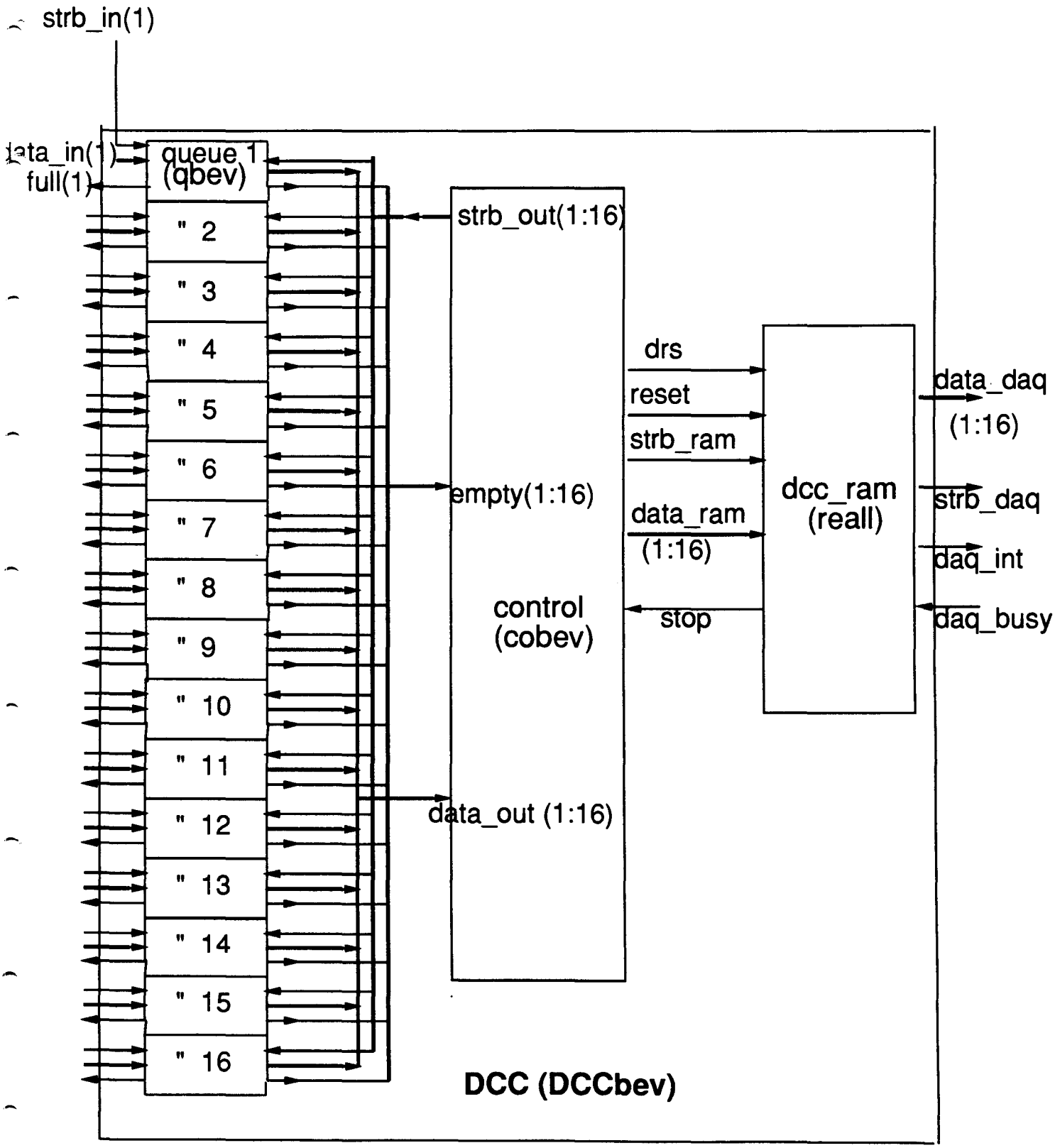
OUTLINE

1) **DATA COLLECTION CHIP (DCC) DESIGN**

2) CORRELATION STUDIES, RESULTS AND LESSONS

3) DESIGN OF A RECONFIGURABLE QUEUE

4) SUMMARY AND FUTURE WORK


strb_in(1)
data_in(1)
full(1)
queue 1
(qbev)
" 2
" 3
" 4
" 5
" 6
" 7
" 8
" 9
" 10
" 11
" 12
" 13
" 14
" 15
" 16
strb_out(1:16)
empty(1:16)
control
(cobev)
data_out (1:16)
drs
reset
strb_ram
data_ram
(1:16)
stop
dcc_ram
(reall)
data_daq
(1:16)
strb_daq
daq_int
daq_busy
DCC (DCCbev)

OUTLINE

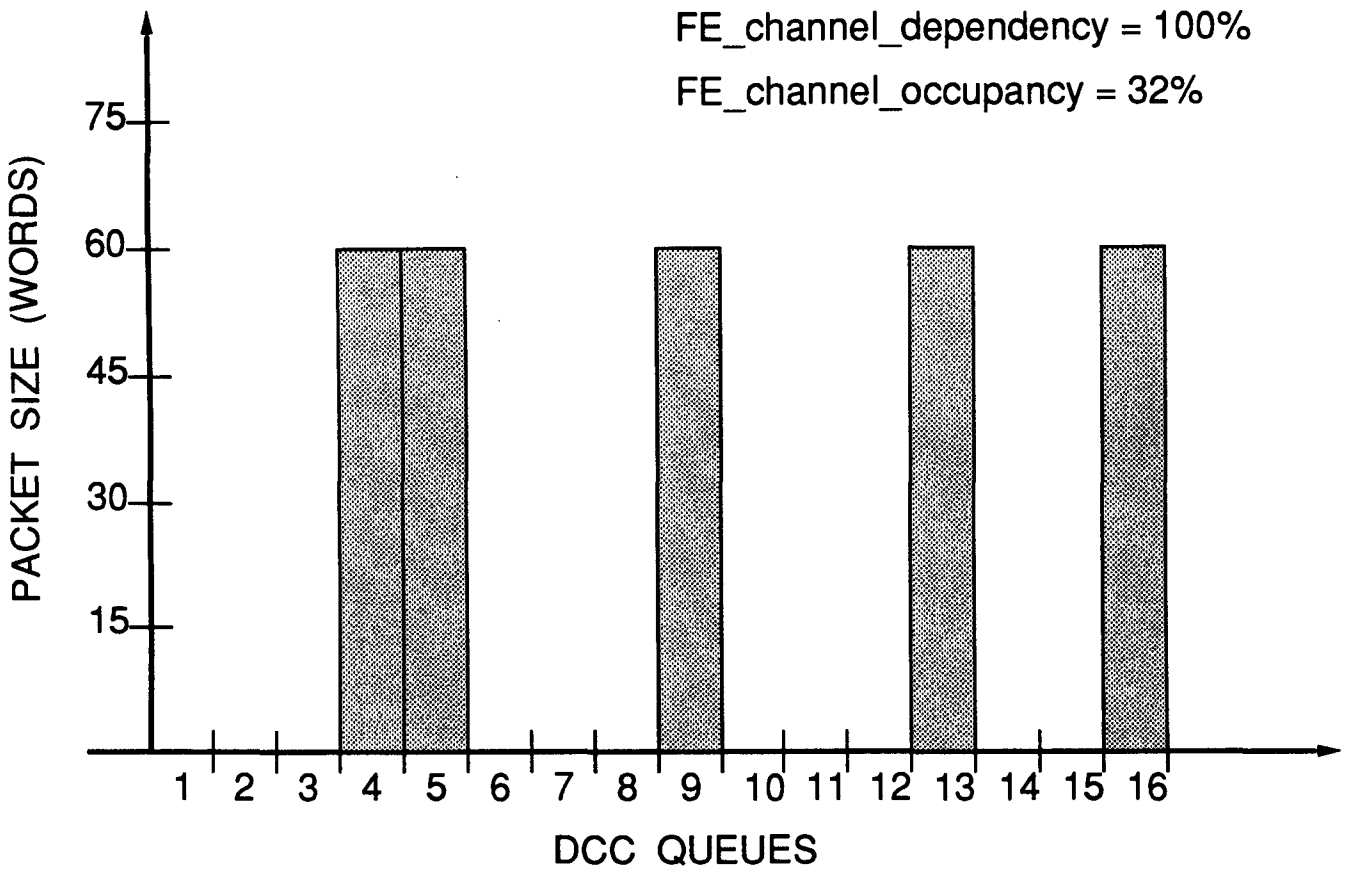
1) DATA COLLECTION CHIP (DCC) DESIGN

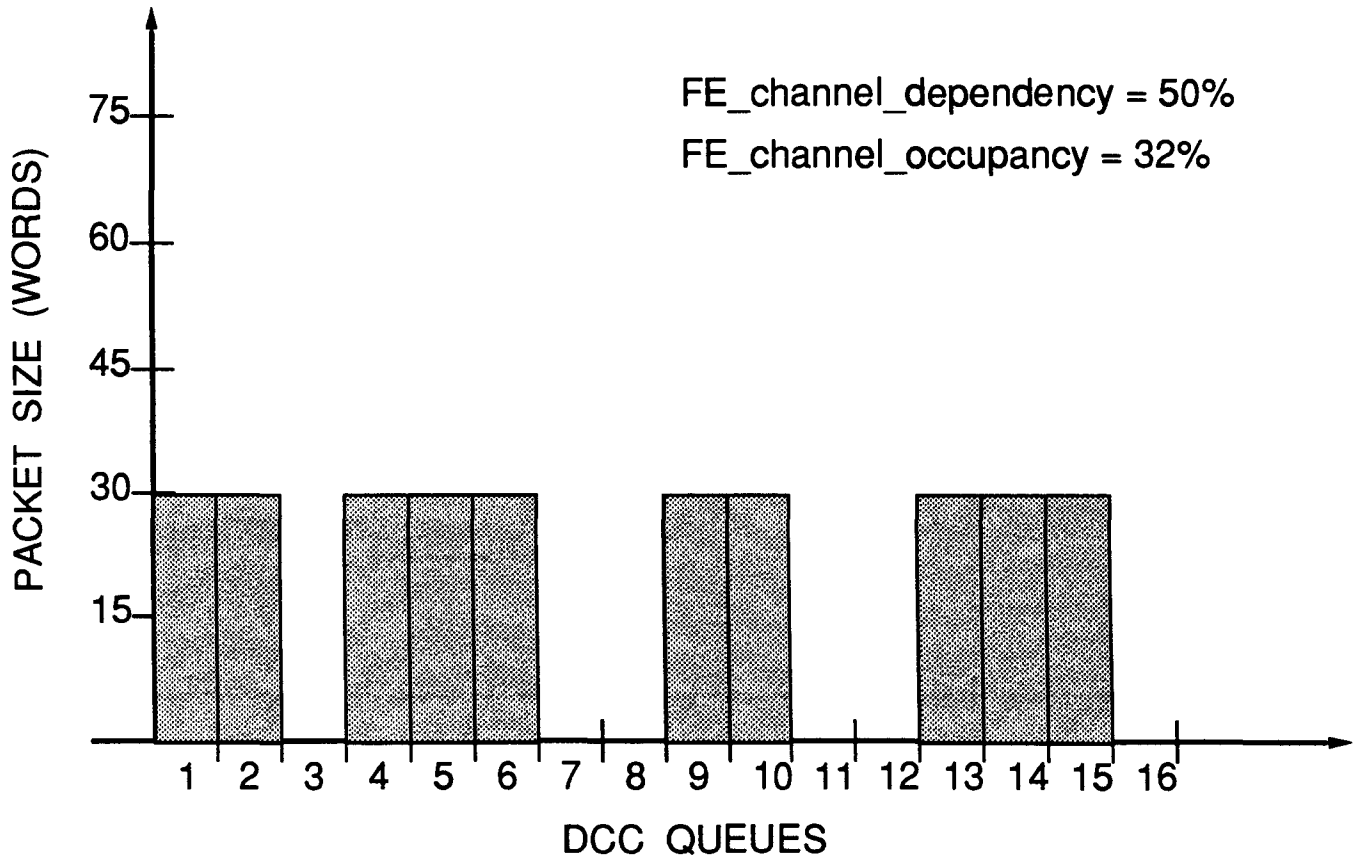
2) CORRELATION STUDIES, RESULTS AND LESSONS

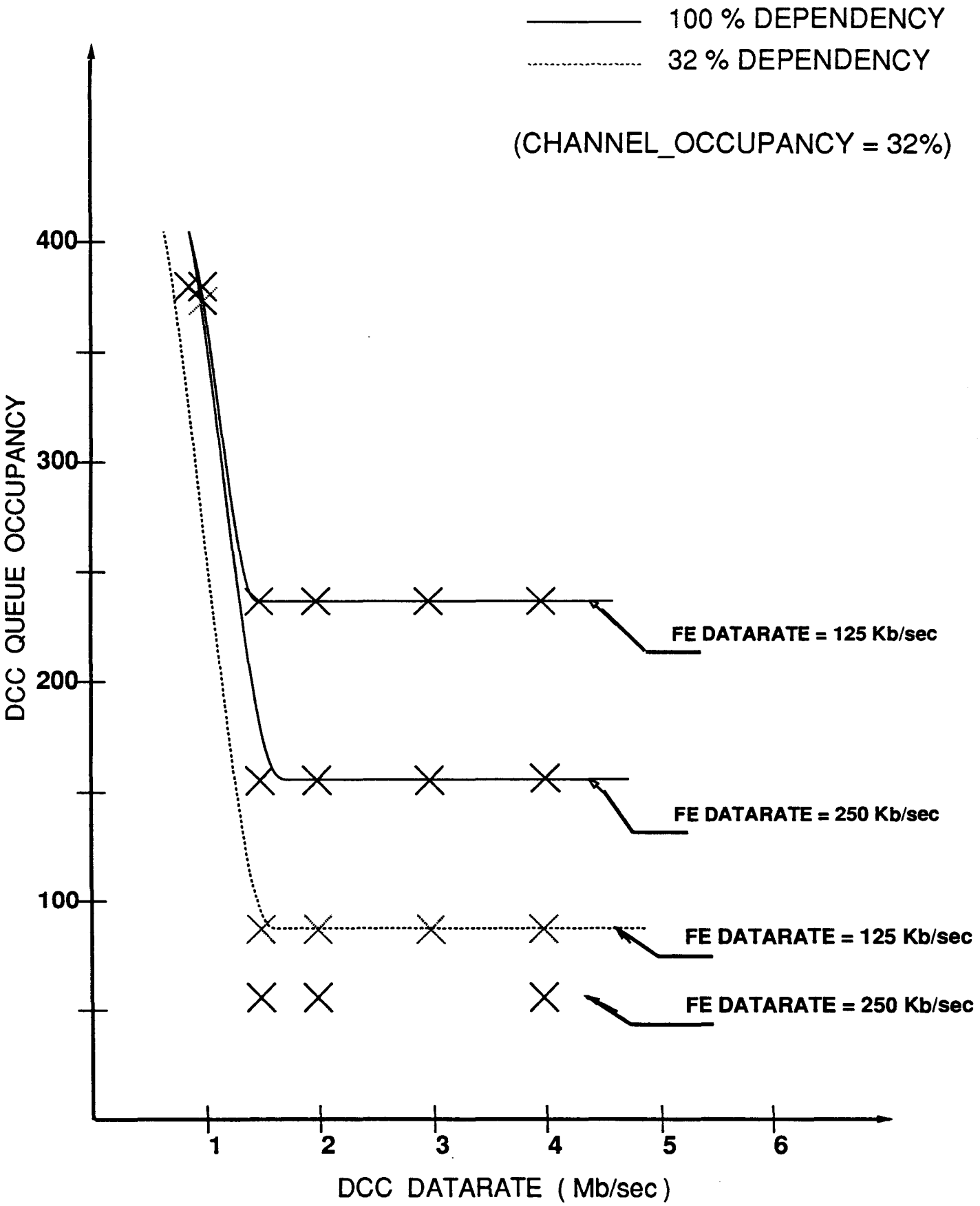
3) DESIGN OF A RECONFIGURABLE QUEUE

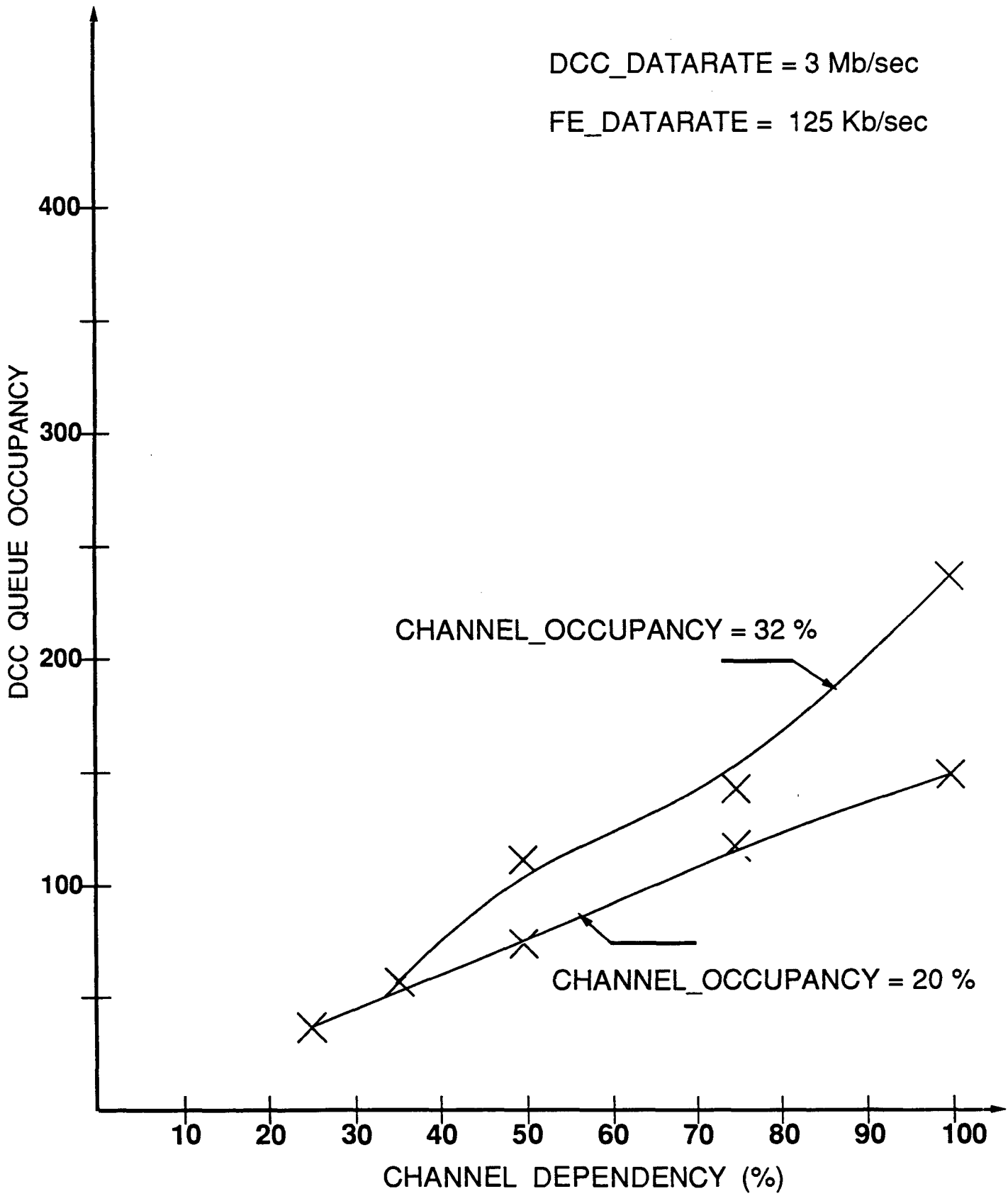
4) SUMMARY AND FUTURE WORK

DATA DISTRIBUTION FOR A TYPICAL EVENT






100 % DEPENDENCY
32 % DEPENDENCY
(CHANNEL_OCCUPANCY = 32%)
DCC QUEUE OCCUPANCY
400
300
200
100
FE DATARATE = 125 Kb/sec
FE DATARATE = 250 Kb/sec
FE DATARATE = 125 Kb/sec
FE DATARATE = 250 Kb/sec
1
2
3
4
5
6
DCC DATARATE (Mb/sec)


DCC_DATARATE = 3 Mb/sec
FE_DATARATE = 125 Kb/sec
DCC QUEUE OCCUPANCY
400
300
200
100
CHANNEL_OCCUPANCY = 32 %
CHANNEL_OCCUPANCY = 20 %
10
20
30
40
50
60
70
80
90
100
CHANNEL DEPENDENCY (%)

OUTLINE

1) DATA COLLECTION CHIP (DCC) DESIGN

2) CORRELATION STUDIES, RESULTS AND LESSONS

3) DESIGN OF A RECONFIGURABLE QUEUE

4) SUMMARY AND FUTURE WORK

WHY DESIGN THE QUEUE ?

1) ESSENTIAL COMPONENT OF THE DCC

(others may change depending on application etc.)

2) THE DCC RAM (WORK SPACE) IS VERY SIMILAR TO THE QUEUE

3) WILL BE USED BY MANY OTHER SUBSYSTEMS IN THE SDC

CHARACTERISTICS OF THE QUEUE ESSENTIAL FOR THE SDC

1) HIGHLY FAULT TOLERANT (RELIABLE)

2) VERY FLEXIBLE (interface with many subsystems of the SDC)

3) EASILY RECONFIGURABLE (in "real time" to prevent queue overflow)

4) FAST (HIGH THROUGHPUT, LOW LATENCY)

5) EASILY TESTABLE

WHY BREAK UP A QUEUE INTO SMALL PIECES ?

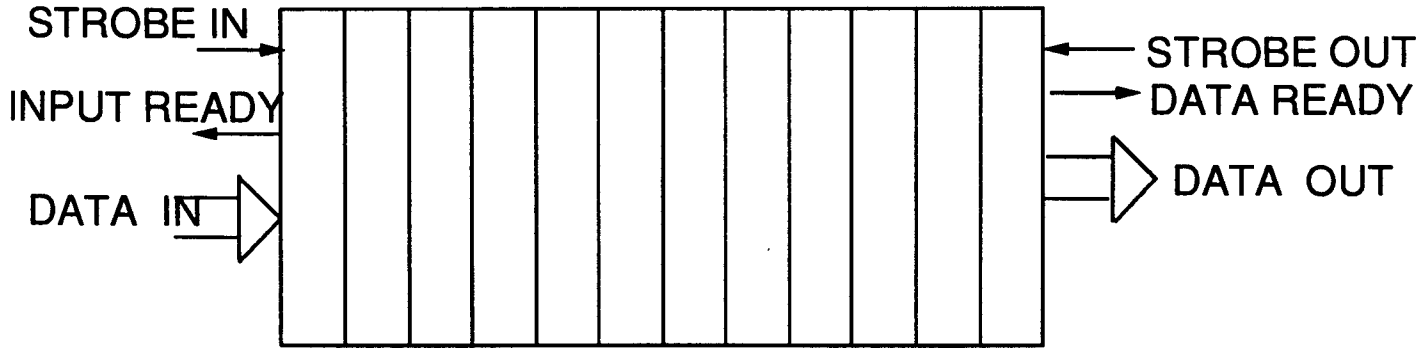
1) LESS WASTAGE OF SILICON TO ACHIEVE

HIGH FAULT TOLERANCE,

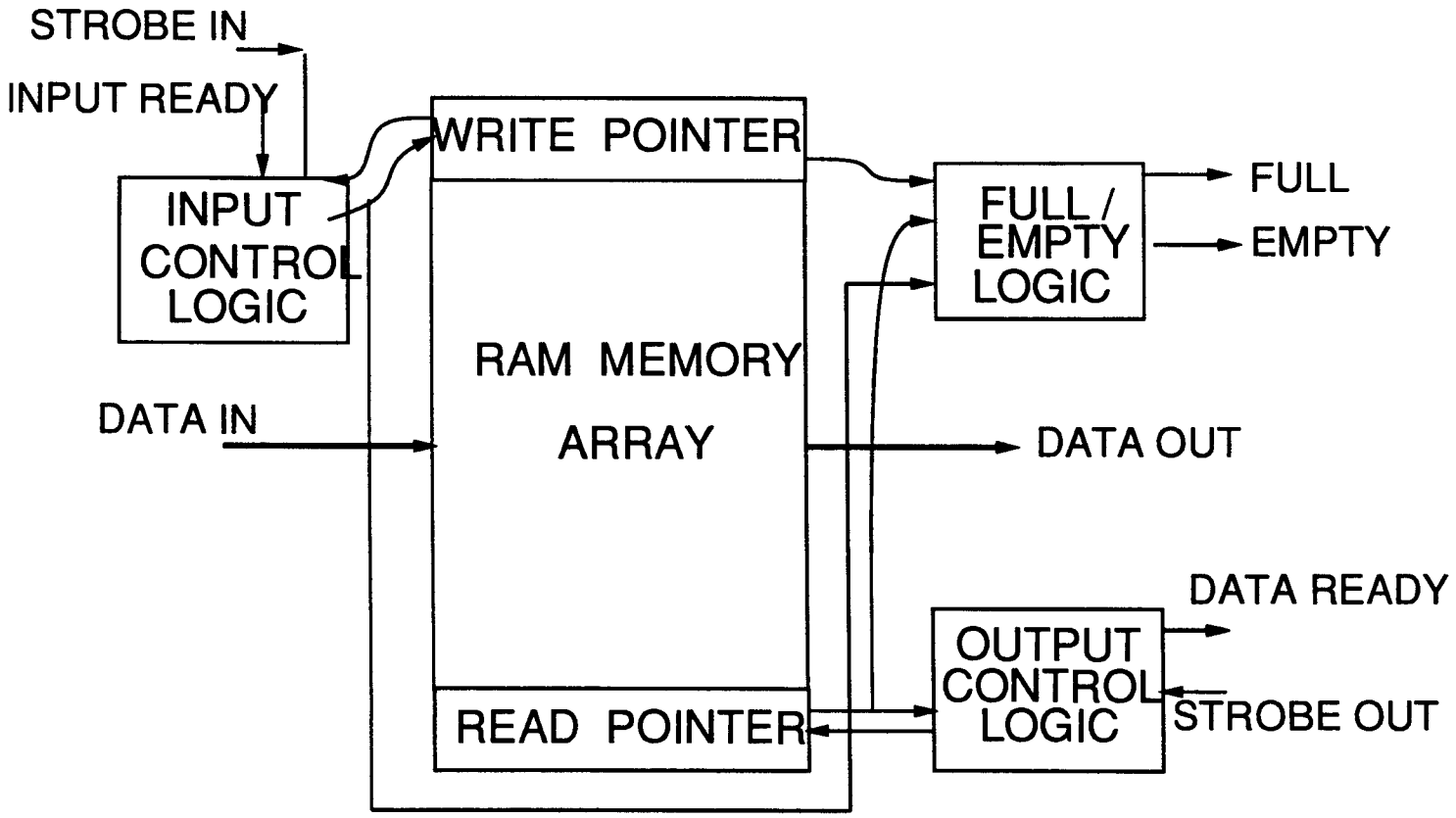
FLEXIBILITY & RECONFIGURABILITY

PREVIOUS DESIGNS

A) SERIAL SHIFT REGISTER

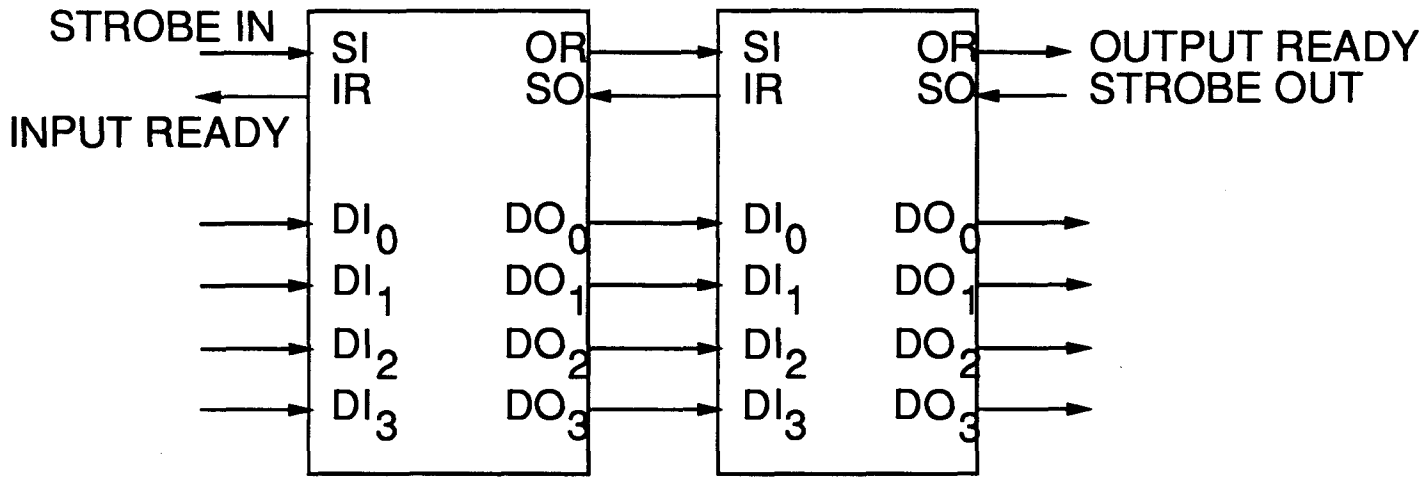


B) RAM BASED DESIGNS

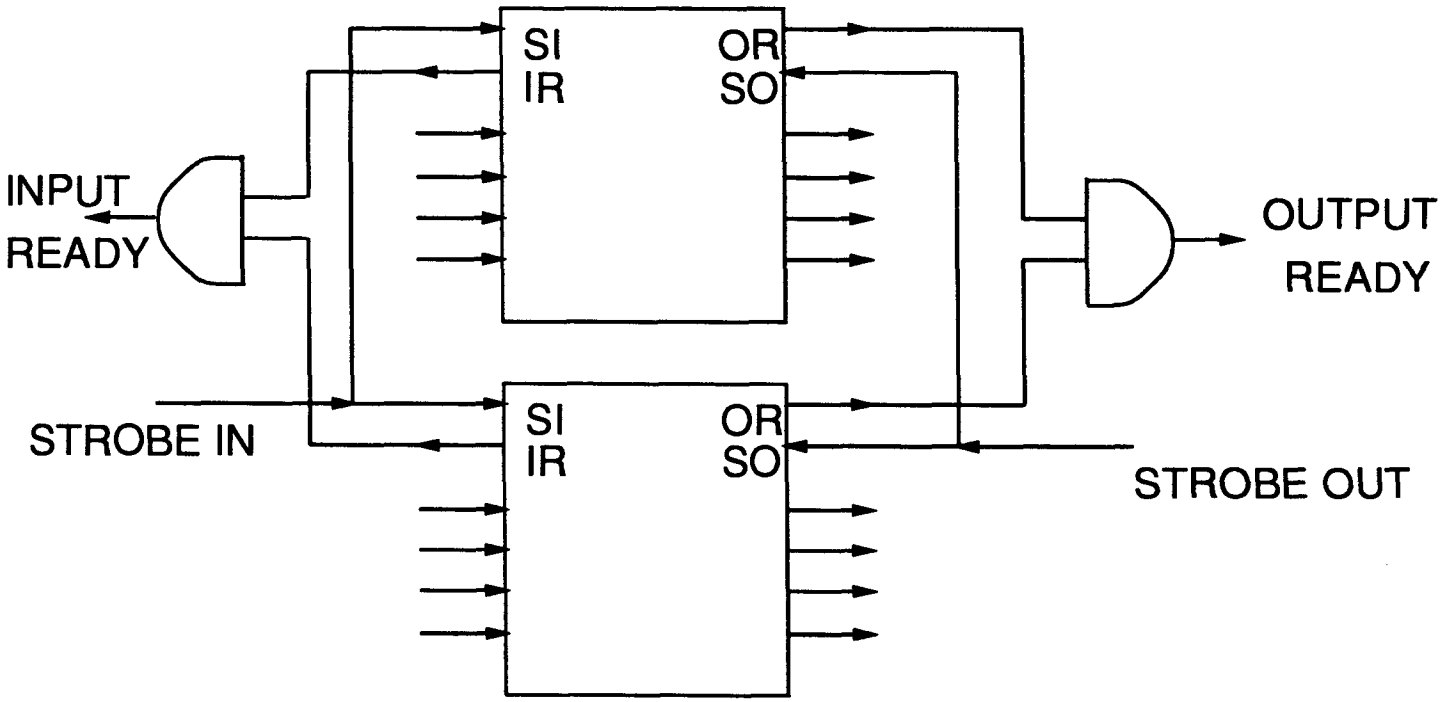


RAM BASED DESIGNS (contd.)

EXPANSION (DEPTH)



EXPANSION (WIDTH)



FACTORS LIMITING PERFORMANCE

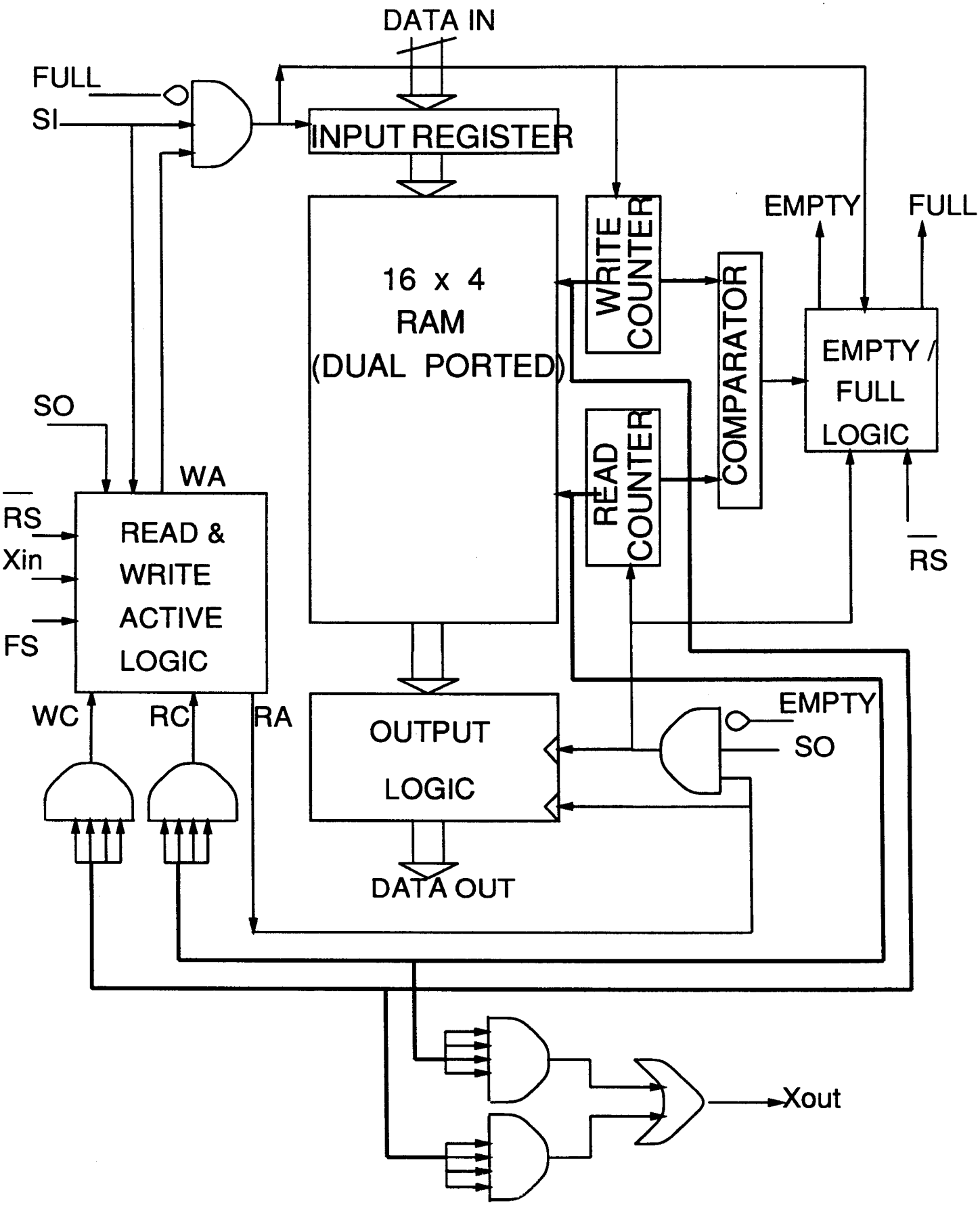
IN THE EARLIER DESIGNS

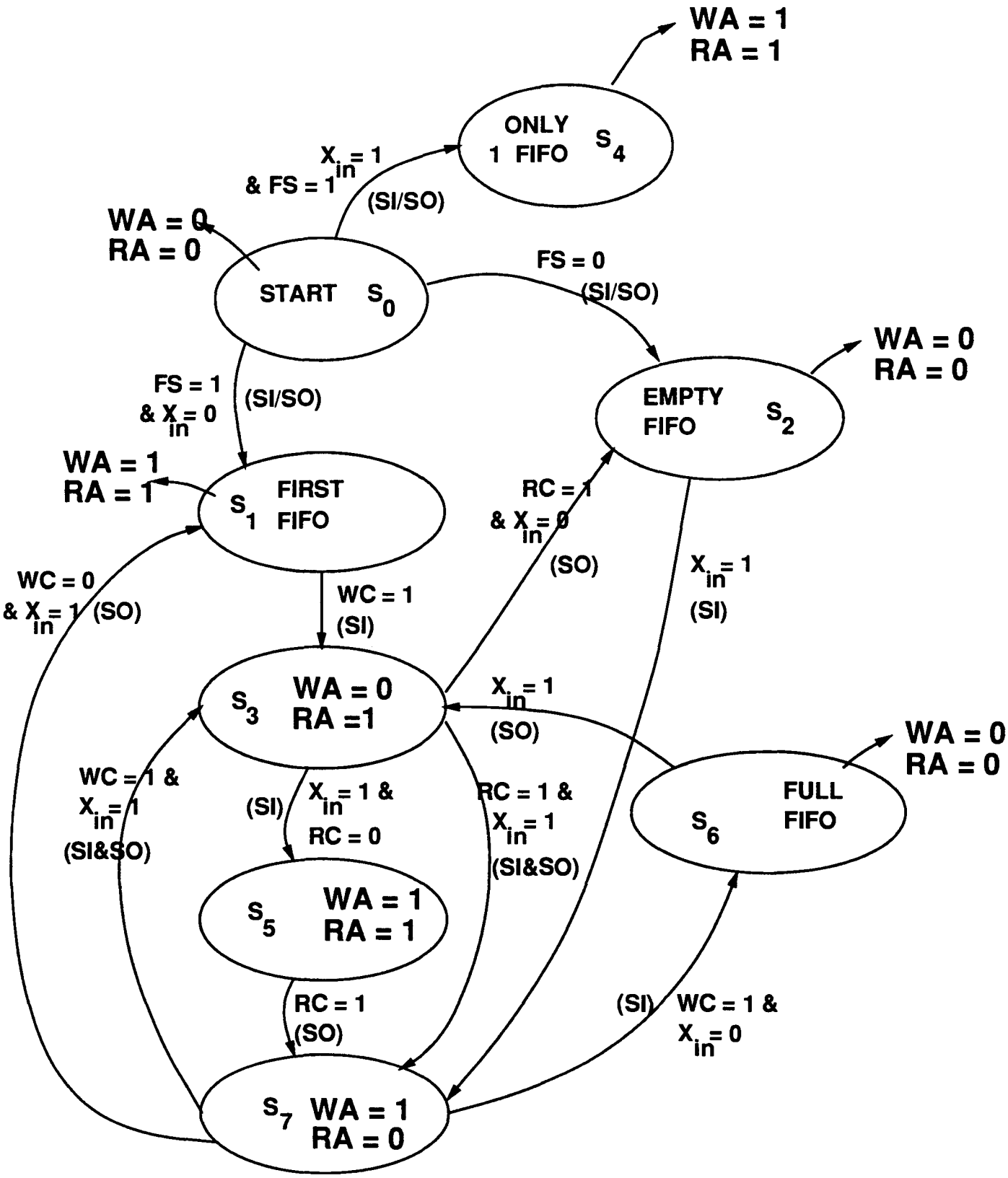
1) PHYSICAL MOVEMENT OF DATA FROM FIFO TO FIFO

   -- CAUSED DUE TO SERIAL CONNECTION OF FIFO's

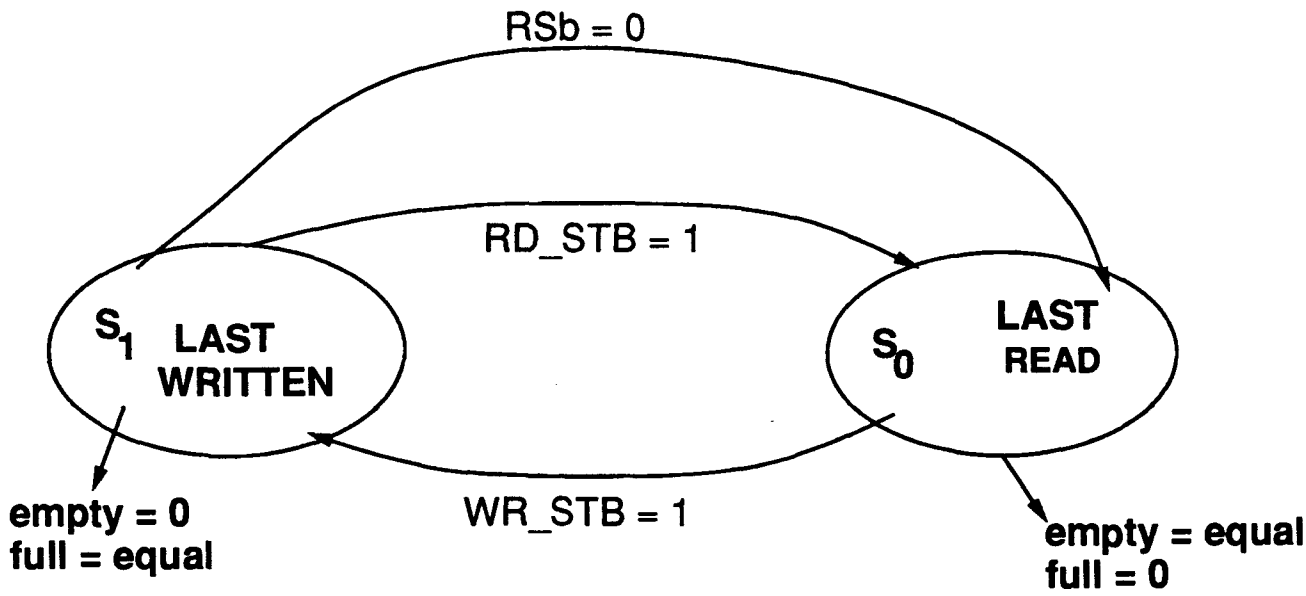
2) HANDSHAKING MECHANISM BETWEEN FIFO's

BLOCK DIAGRAM OF THE NEW DESIGN

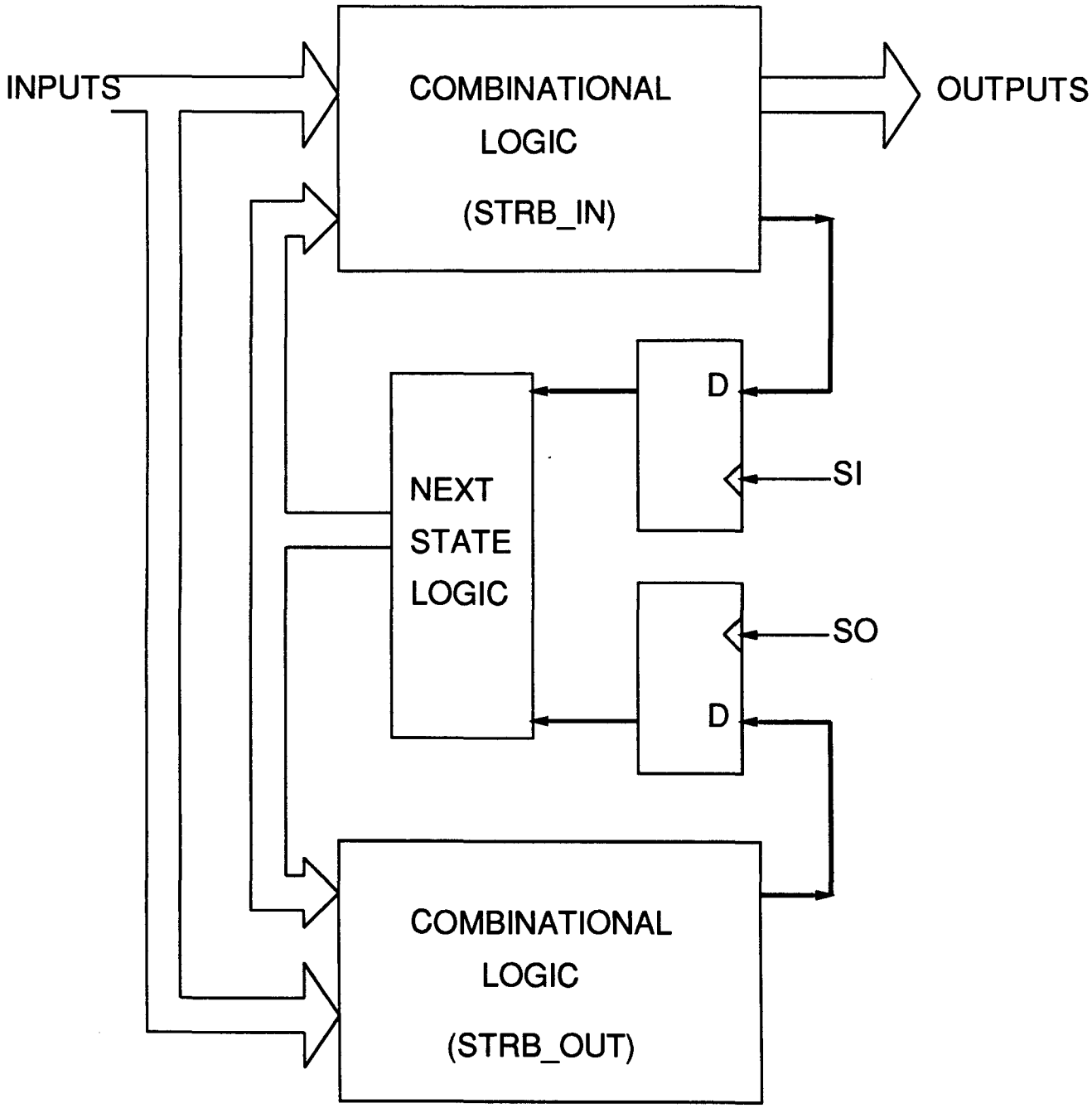




STATE DIAGRAM OF READ & WRITE ACTIVE LOGIC



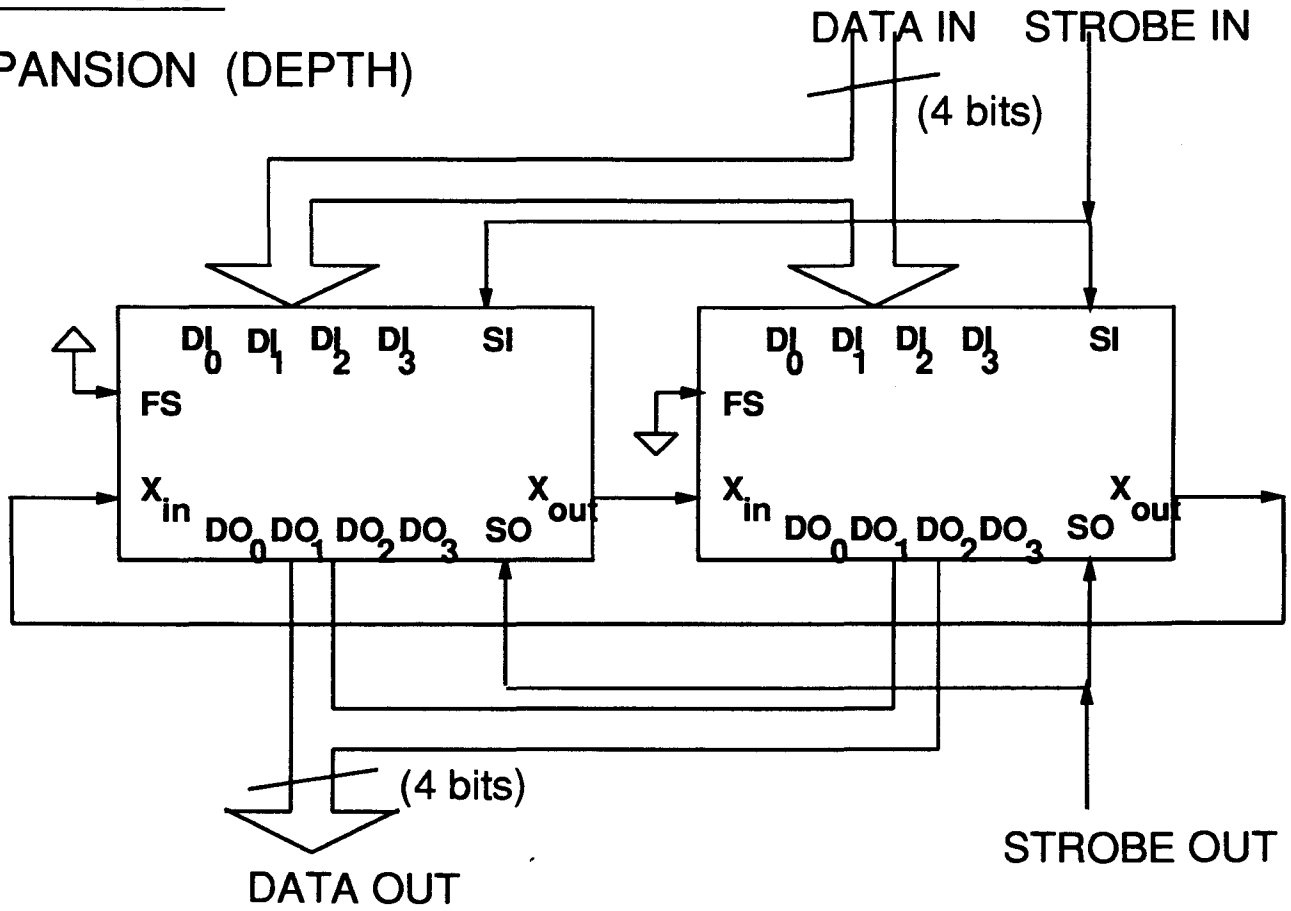
STATE DIAGRAM OF THE FULL / EMPTY LOGIC



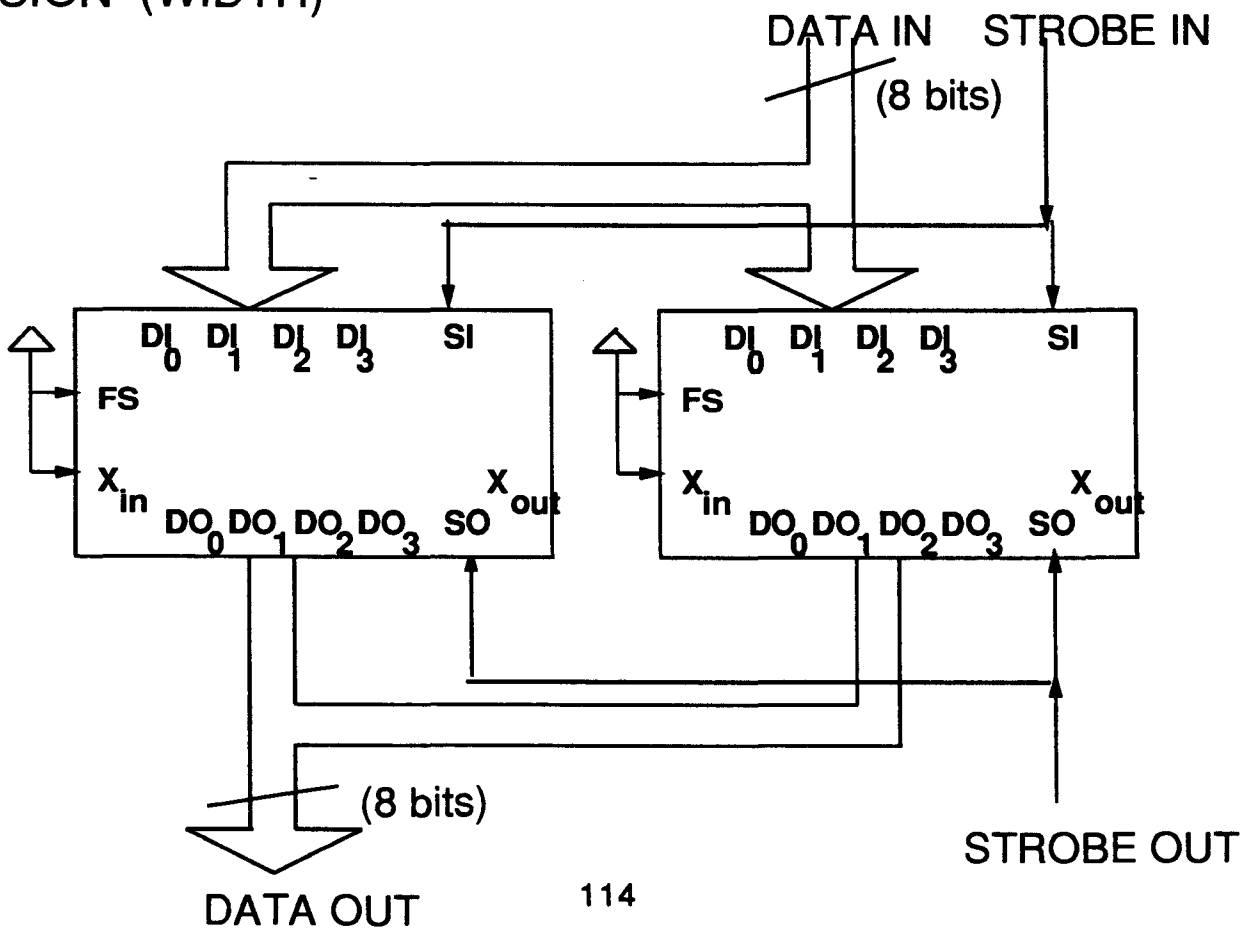
STATE MACHINE CLOCKED BY TWO INPUTS

NEW DESIGN

EXPANSION (DEPTH)



EXPANSION (WIDTH)



ADVANTAGES OF THE NEW DESIGN

1) NO HANDSHAKING SIGNALS

2) CASCADABLE (BOTH IN DEPTH & WIDTH)

-- WITHOUT MOVEMENT OF DATA FROM FIFO TO FIFO

3) TESTABLE

4) EASILY RECONFIGURABLE

-- COULD BE DONE IN "REAL TIME"

5) FAULT - TOLERANT

OUTLINE

1) DATA COLLECTION CHIP (DCC) DESIGN

2) CORRELATION STUDIES, RESULTS AND LESSONS

3) DESIGN OF A RECONFIGURABLE QUEUE

4) SUMMARY AND FUTURE WORK

SUMMARY

1) BEHAVIORAL DESIGN OF THE DATA COLLECTION CHIP

2) HIGH LEVEL STUDIES OF THE BEHAVIORAL MODEL

3) STRUCTURAL DESIGN FROM THE HIGH LEVEL STUDIES

4) DESIGN VERIFICATION BY SIMULATION

FUTURE WORK

1) SIMULATE THE FAULT TOLERANT NATURE OF THE QUEUES

2) ADAPTIVE RECONFIGURABILITY TECHNIQUES

3) STRUCTURAL DESIGN OF OTHER MODULES OF THE DCC

4) BUILD AND TEST A PROTOTYPE

FAST DATA COMPRESSION & TRANSMISSION FROM A WAFER

BRAD COOKE
KLAUS LACKNER
ANDREA PALOUNEK
DAVE SHARP
HANS ZIOCK

GOAL: A SELF CONTAINED SYSTEM WHICH DOES NOT REQUIRE TRIGGER INFORMATION TO TRANSMIT ITS DATA

REDUCE DATA TRANSMISSION RATE

ADVANTAGES OF A SELF CONTAINED SYSTEM

- ELIMINATES A POTENTIAL BOTTLENECK
  ⟹ GREATER FLEXIBILITY FOR LEVEL I TRIGGER

- ON LINE SELF CALIBRATION
  ⟹ SPECIAL SUBFILTER

- CONTRIBUTES TO LEVEL I FILTER

- STIFF JETS
- STIFF PARTICLE TRACKS
- BACK TO BACK JETS
- SEMILEPTONIC EVENTS
  SINGLE TRACK NEAR JET
- MISSING MOMENTUM
- VERTEX RECONSTRUCTION
  TAGGING HEAVY QUARK EVENTS

DATA FUSION

- ELECTROMAGNETIC CALORIMETER EVENT

  DISTINGUISH PHOTON EVENT FROM ELECTRON BY LOOKING FOR TRACK

- TIMING UNCERTAINTY

  CORRELATION WITH MUON DATA CAN NARROW DOWN THE TIME WINDOW OF THE EVENT

DATA COMPRESSION

- NECESSARY FOR TRANSMITTING ALL DATA OVER GHZ FIBER OPTIC CHANNELS

- EVEN AFTER LEVEL I FILTER DATA RATES ARE HIGH (AT 100 kHz TRIGGER RATE ~ 60 ~ 120 Mhz/wafer)

  (TRANSMISSION TIME FOR JET EVENT 10μs)

DATA COMPRESSION CAN LEAD TO AN IMPORTANT SAVINGS IN FIBEROPTIC CHANNELS AND/OR TRANSMISSION SPEED EVEN AFTER A LEVEL I FILTER

BASIC ARCHITECTURE

TWO DATA CHANNELS PER WAFER

PROMPT CHANNEL		DELAYED CHANNEL
FAST WITH CONSTANT DELAY	⟷	SAME TRANSMISSION RATE VARIABLE DELAY
FIXED LENGTH DATA STRING	⟷	VARIABLE LENGTH
INFORMATION IS RELEVANT	⟷	INFORMATION IS COMPLETE

DATA RATES

$\mathcal{L} = 10^{33}$

$N_{tracks} \sim \frac{60\,cm^2}{r_\perp^2}$ PER WAFER PER CYCLE

$r_\perp = 9\,cm \Rightarrow 0.75$ tracks/wafer/cycle

Range of Interest: 0 – 2 TRACKS

HIGHER RATES? WAFER LIFETIME?

640 strips per side
5 milliradian stereo angle

ESTIMATING BANDWIDTH REQUIREMENTS

TIME SCALES: 16 ns EVENT CYCLE
4 μs LEVEL I TRIGGER

50-100 EVENT CYCLES FOR INTERNAL BUFFERING

MEAN DATA RATE: 0.75 events
STATISTICAL FLUCTUATIONS $\lesssim 20$
JET FLUCTUATIONS ~100

DATA COMPRESSION

ASSUMPTIONS:

- POISSON DISTRIBUTIONS FOR TRACKS 0.75
- CLUSTERWIDTH FROM MONTE CARLO 2

ENCODING:

TRANSMIT "OPTIMAL" ENCODING
OF "EDGE" INFORMATION

MONTE CARLO

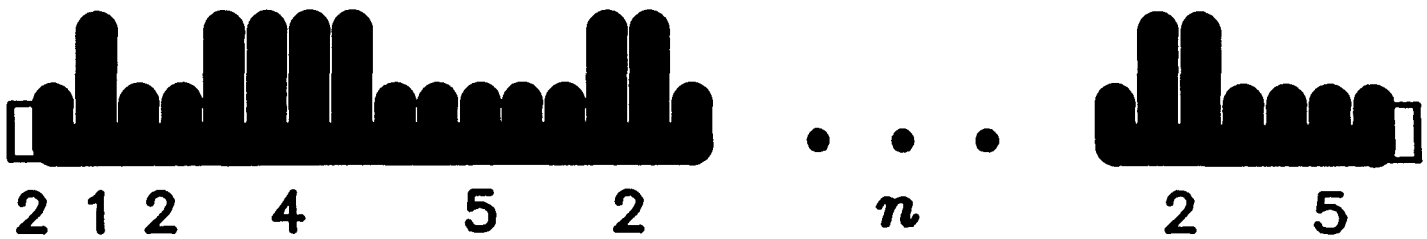
C++ OBJECT ORIENTED

OBJECTS

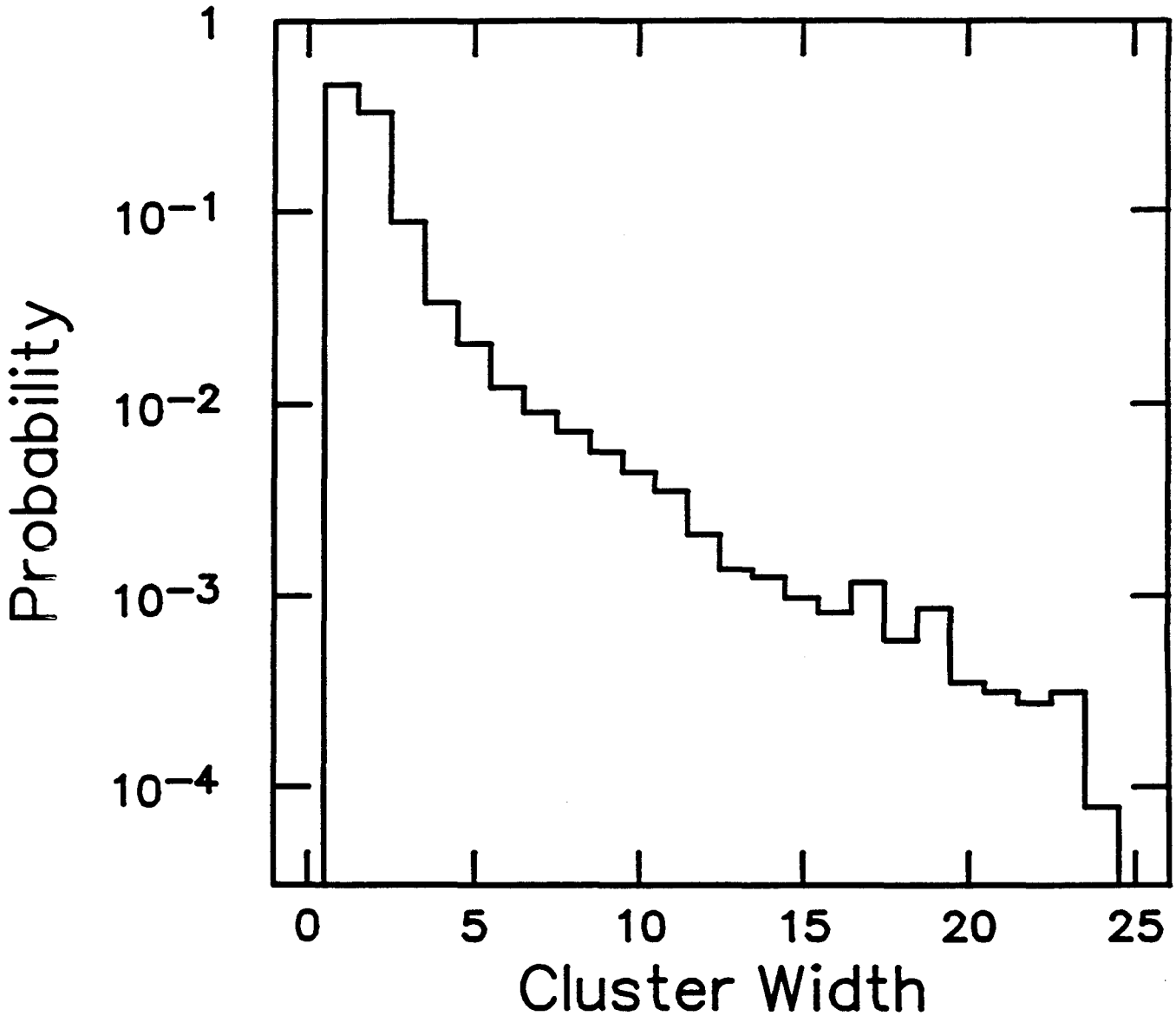
WAFER
CLUSTERLIST
CLUSTER

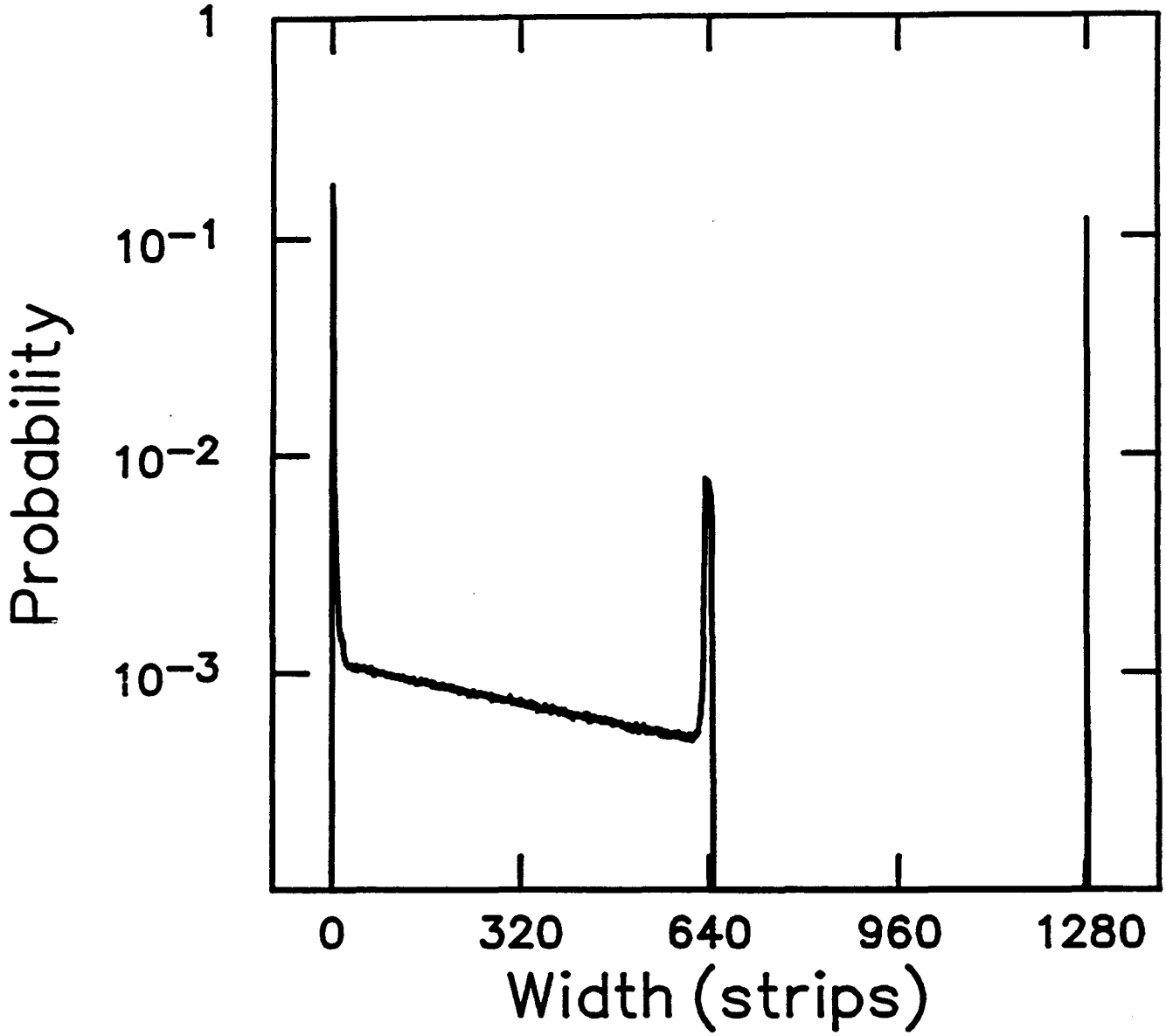
EVENT
(HITS)

RANDOM DISTRIBUTION GENERATORS



NUMBERS ARE THE NUMBERS OF CONSECUTIVE STRIPS IN THE SAME STATE


1
10^{-1}
10^{-2}
10^{-3}
10^{-4}
Probability
0
5
10
15
20
25
Cluster Width


1
10^{-1}
10^{-2}
10^{-3}
Probability
0
320
640
960
1280
Width (strips)

OPTICAL ENCODING

0.75 tracks: 6.6 bits per edge

26.5 bits per event

$4\langle \text{tracks} \rangle + 1$

1.5 tracks: 6.8 bits per edge

48.3 bits per event

0.75 tracks ~ 1.7 Gbit/channel

MINIMUM PROMPT CHANNEL

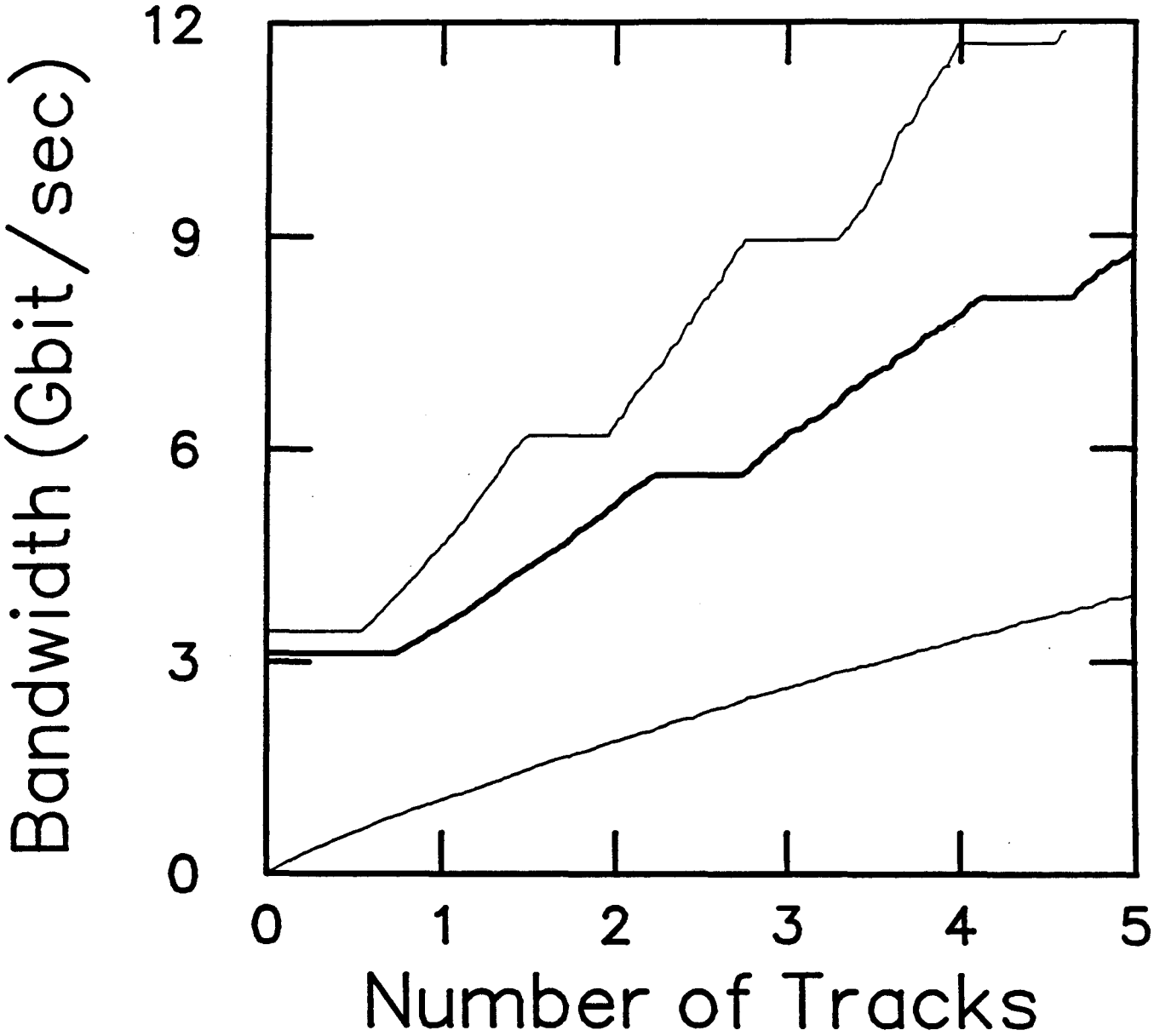
0 TRACKS	1+3 bits for N_{track}	4 bits
1 TRACK	1+3 bits for N_{track} 4x10 bits for edges	44 bits
2 TRACKS	1+3 bits for N_{track} 4x10 bits for edges on one side	44 bits
3 TRACKS	1+3 bits for N_{track} 3x10 bits for centroids 10 bits for # of hit strips	44 bits
4 TRACKS	1+3 bits for N_{track} 4x10 bits for centroids	44 bits
> 5 TRACKS	1+10 bits for N_{track} 10 bit for centroid 20 additional bits leftmost & rightmost coarse grained map	41 bit

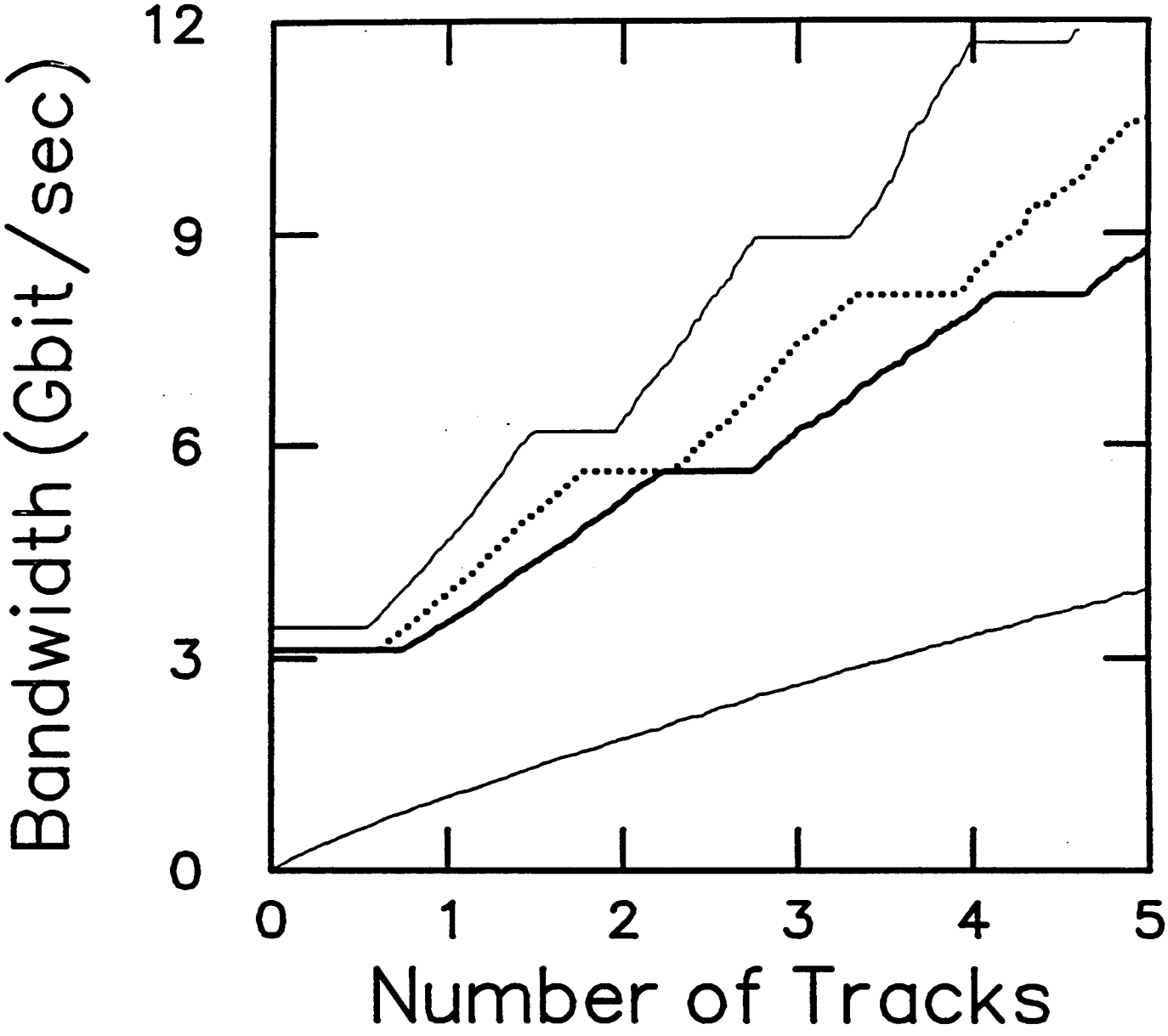
- PROMPT CHANNEL CARRIES INFORMATION
- CALCULATE DELAYED CHANNEL
  WITH OPTIMAL ENCODING
  ASSUMING PRESENCE OF PROMPT
  CHANNEL

  — 0, 1 TRACK EVENTS ARE OMITTED

  — CHECK SUM IS OMITTED

  — 2.5 Gbz for buffered events


Bandwidth (Gbit/sec)
12
9
6
3
0
0
1
2
3
4
5
Number of Tracks


Bandwidth (Gbit/sec)
12
9
6
3
0
0
1
2
3
4
5
Number of Tracks

NON OPTIMAL ENCODINGS

- FIXED LENGTH 11 bit per edge

- TWO FIXED LENGTH

  1+3 bit
  1+11 bit

  NEARLY OPTIMUM FOR ≥2 TRACKS

- SIGNIFICANT BIT ENCODING

  TWICE THE NUMBER OF SIGNIFICANT BITS FOR EACH NUMBER

1 / 15 / 7
1 / 1111 / 111
1 / 0001 / 001

SIGNIFICANT BIT ENCODING

- SIMPLE ALGORITHM
- PIPELINED ARCHITECTURE
  PARALLELIZATION IS POSSIBLE
- EVERY STEP COMPRESSES
- MEMORY REQUIREMENT

  $100 \times 2 \times 1280 = 250\,\text{kbit}$

- BANDWIDTH
  4 – 6 Gbit/s per channel

A. Bogaerts (CERN)

Simulation of SCI Protocols in MODSIM

presented at

Workshop on Data Acquisition and Trigger System Simulations for High Energy Physics

SSC, Dallas 23 - 25 April 1992

Authors, contributors:

SCI Transactions: Jan. - June 1991

A. Bogaerts (CERN), J.-F. Renardy (CEA, Saclay)

Cache Coherency: Jan. 1992 - not yet finished

A. Bogaerts (CERN), D. Samyn (CERN)

LHC DAQ modeling: June 1990 - not yet finished

CERN RD24: Applications of the Scalable Coherent Interface to Data Acquisition at LHC

A. Bogaerts, J. Buytaert, R. Divia, H. Muller, C. Parkman, P. Ponting, D. Samyn (CERN)

B. Skaali, G. Midttun, D. Wormwald, J. Wikne (University of Oslo)

S. Falciano, F. Cesaroni (INFN and University of Rome)

P. Creti (INFN Lecce)

K. Lochsen, B. Solberg (Dolphin SCI Technology, Oslo)

A. Guglielmi, A. Pastore (DEC/CERN Joint Project)

F.-H. Worm, J. Bovier (CES, Geneva)

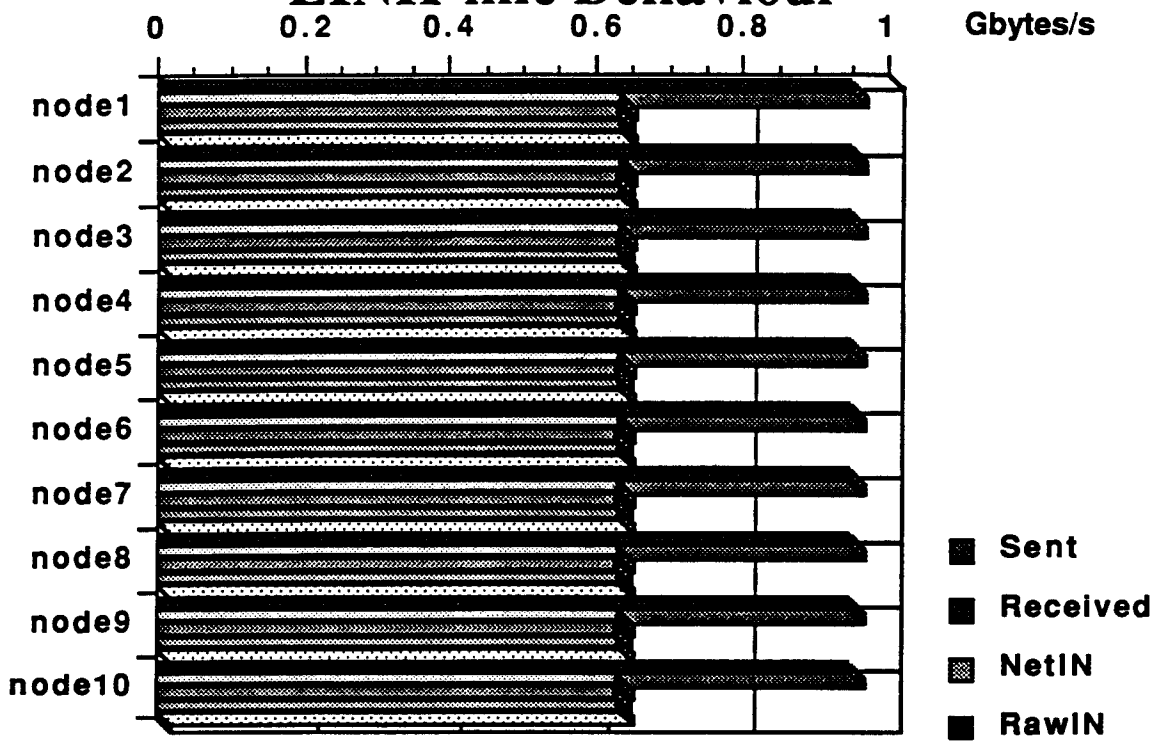
C. Davis (Radstone Technology, Towcester)

SCI Simulation Overview

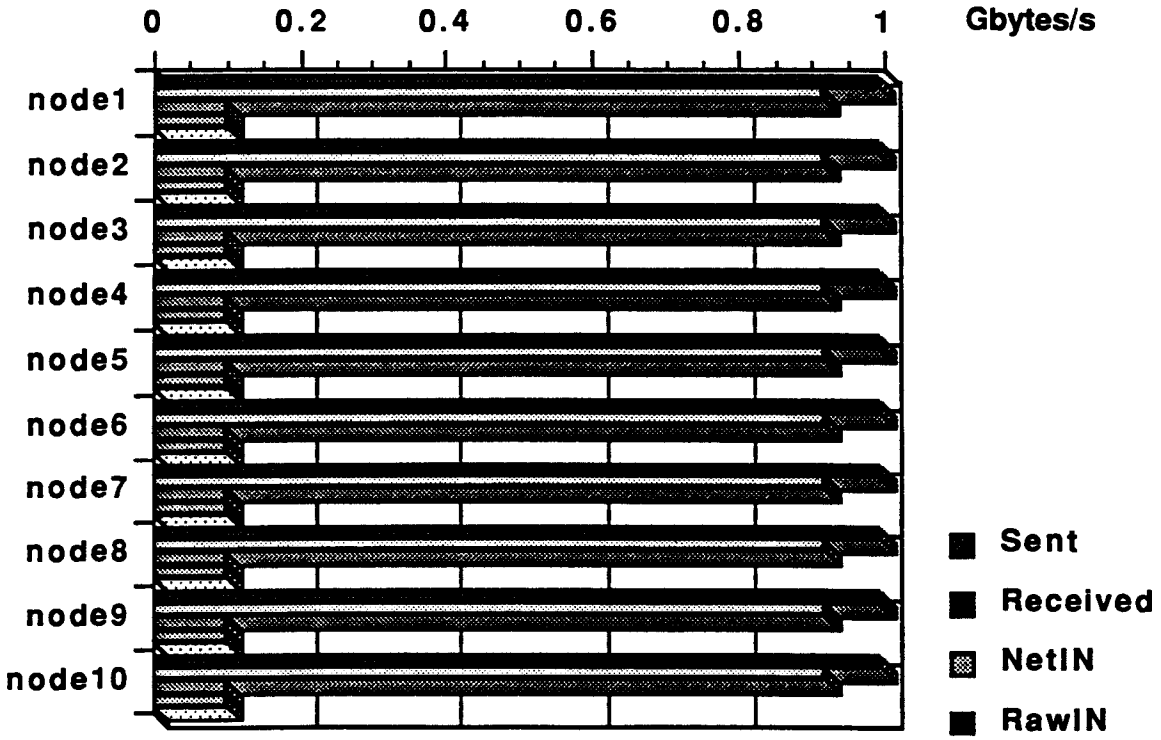
Spec	SCI specification, C-code IEEE committee	
A. Hardware Simulation		
VLSI	Behavioural models for VLSI parts, VERILOG Dolphin Server Technology A.S.	sub-nanosec
Board	Board level devices (nodes, processors, interfaces), VERILOG Dolphin Server Technology A.S.	
Ringlet	SCI transmission protocols (packets, ringlets), VERILOG Dolphin Server Technology A.S.	
B. Architecture Simulation		
Ringlet	SCI transmission protocols (packets, ringlets), SIMULA Informatics Department, University of Oslo	symbol (2ns)
Multiple Ringlets	SCI Systems (cache coherence, network, bridges), SIMULA Informatics Department, University of Oslo	
Large System	Large networks (> 1000 nodes), DAQ Systems, MODSIM CERN	packet (≥100ns)

Regular Traffic

LINK-like Behaviour

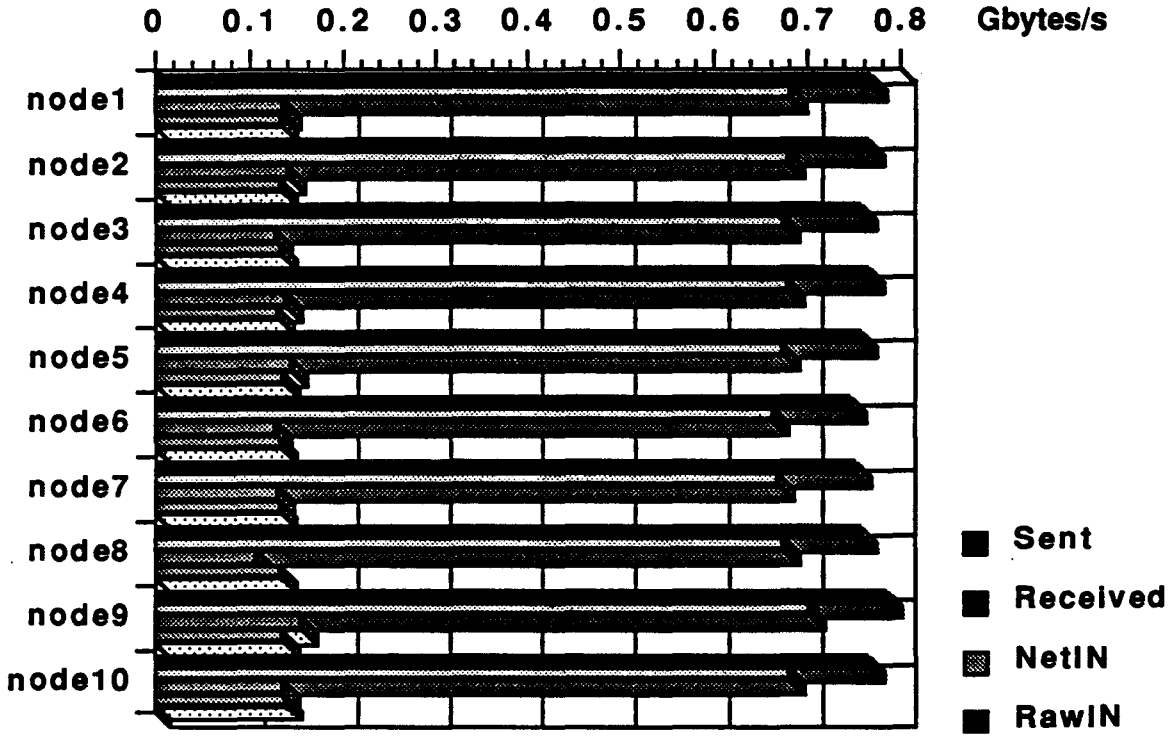


BUS-like Behaviour

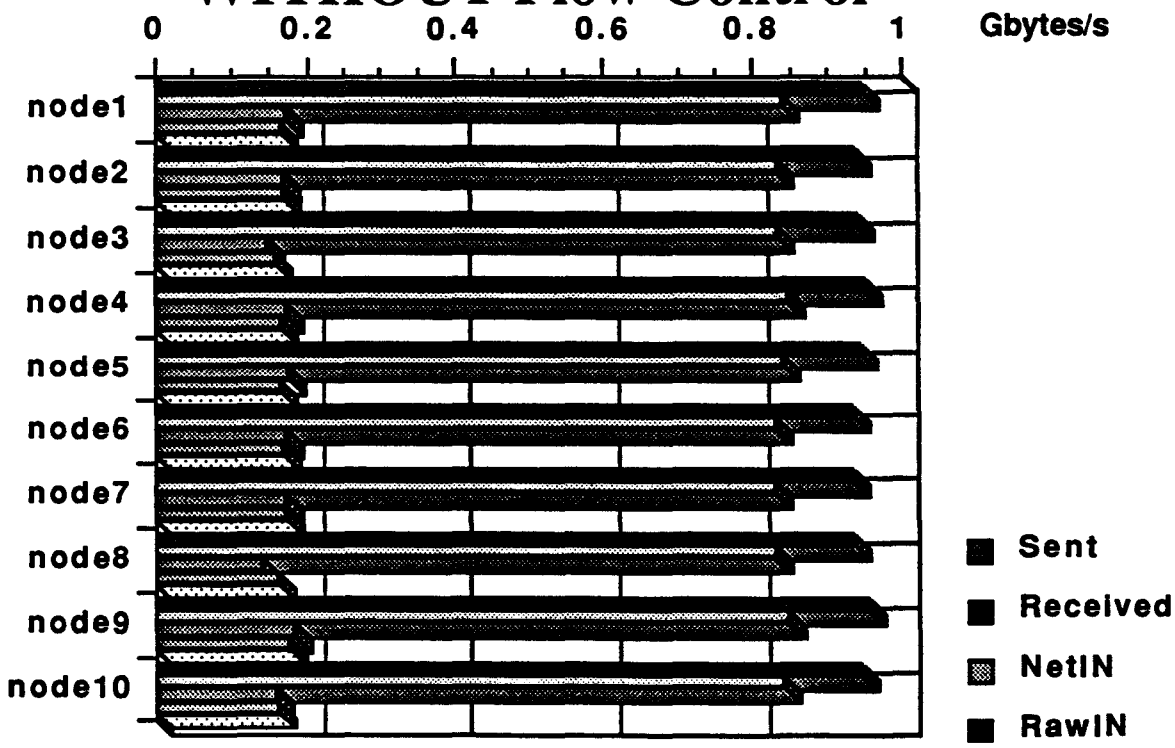


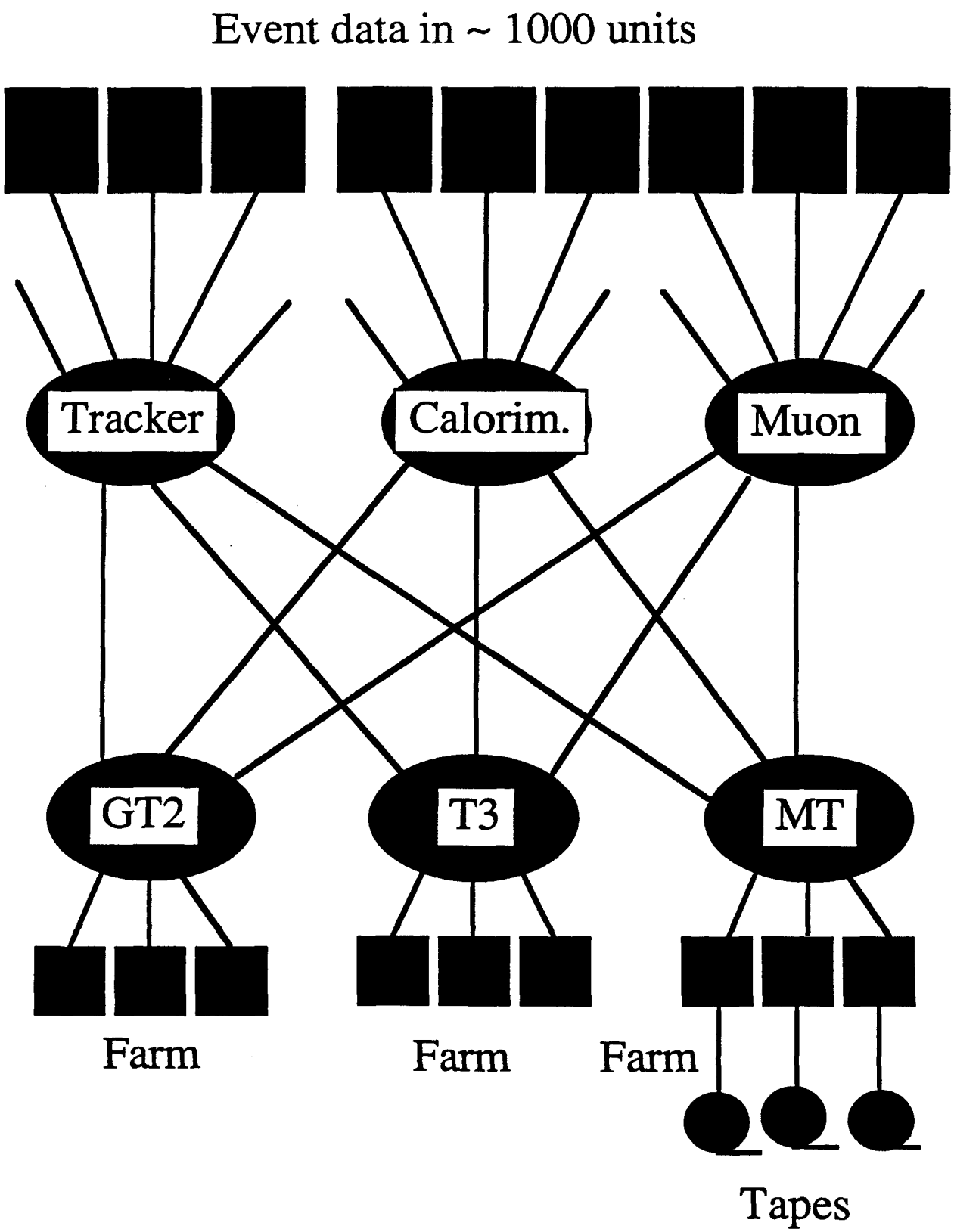
Random Traffic

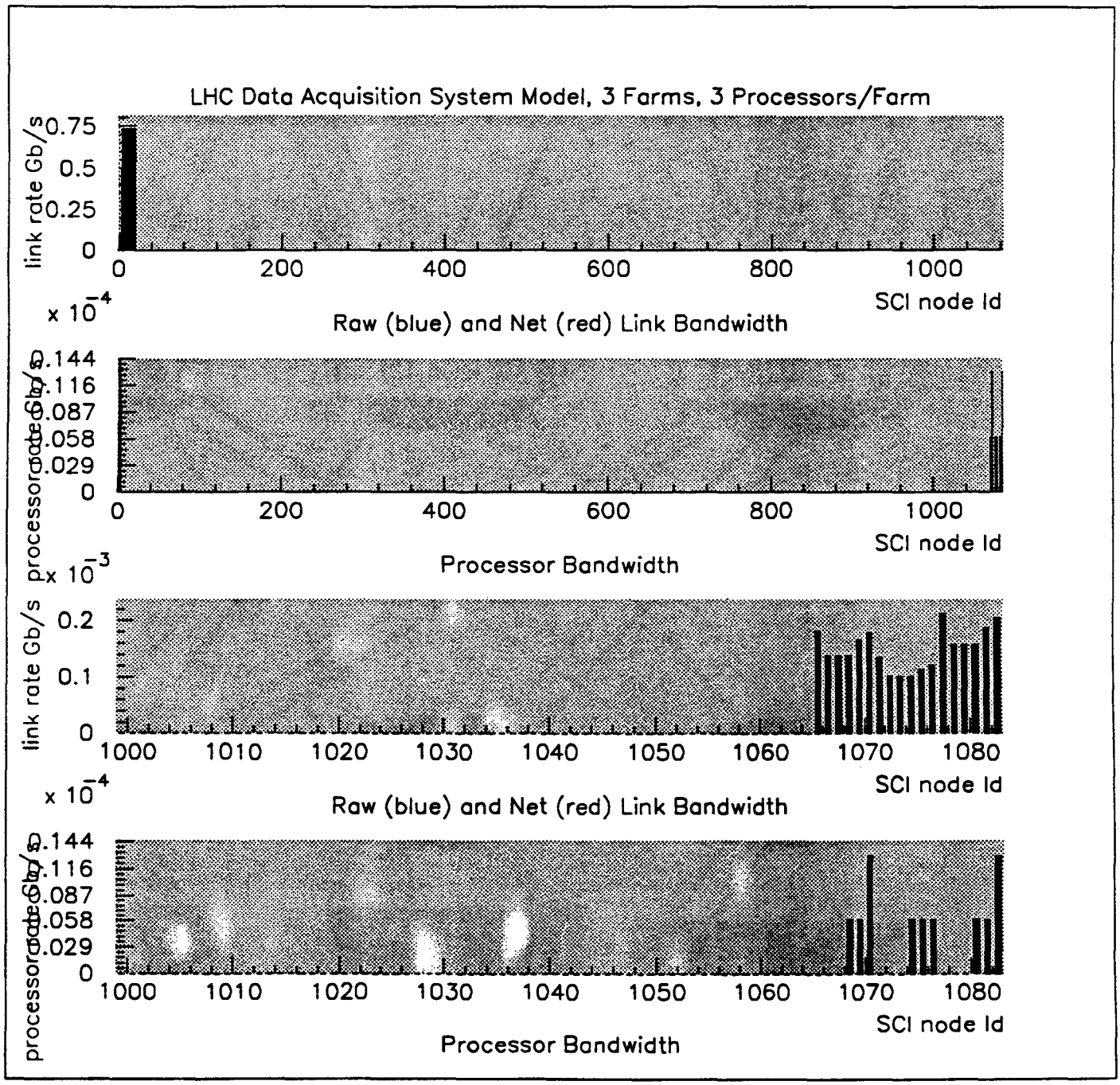
WITH Flow Control



WITHOUT Flow Control




Event data in ~ 1000 units
Tracker
Calorim.
Muon
GT2
T3
MT
Farm
Farm
Farm
Tapes

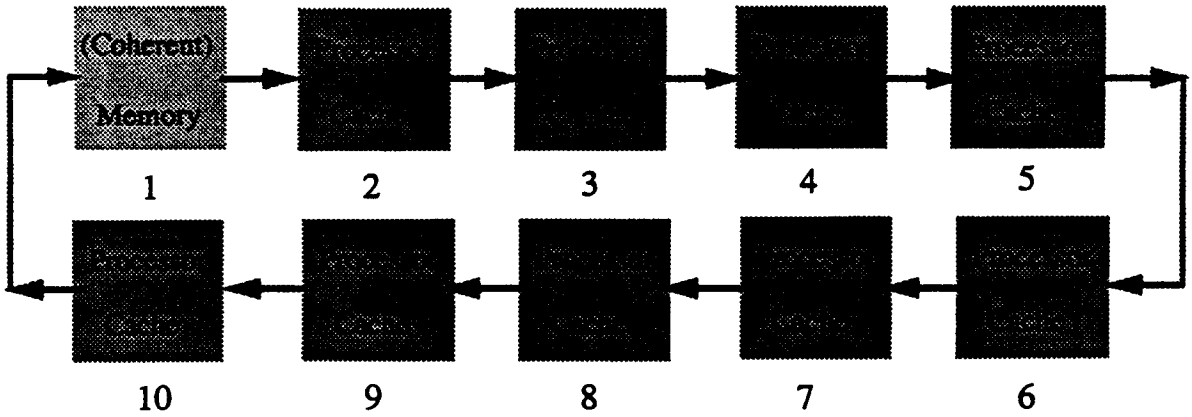

LHC Data Acquisition System Model, 3 Farms, 3 Processors/Farm
link rate Gb/s
0.75
0.5
0.25
0
0
200
400
600
800
1000
SCI node Id
Raw (blue) and Net (red) Link Bandwidth
x 10^-4
0.144
0.116
0.087
0.058
0.029
0
processor rate Gb/s
0
200
400
600
800
1000
SCI node Id
Processor Bandwidth
x 10^-3
link rate Gb/s
0.2
0.1
0
1000
1010
1020
1030
1040
1050
1060
1070
1080
SCI node Id
Raw (blue) and Net (red) Link Bandwidth
x 10^-4
0.144
0.116
0.087
0.058
0.029
0
processor rate Gb/s
1000
1010
1020
1030
1040
1050
1060
1070
1080
SCI node Id
Processor Bandwidth

Cache Coherency Simulations

- **Purpose**
  - study impact on HEP DAQ Systems
  - investigate trade-offs non/coherent data access
  - investigate and understand efficiency of different options (e.g. IEEE minimal, typical, full set)
- **Implementation strategy**
  - use existing MODSIM infrastructure for LHC/DAQ simulation, as well as future extensions
  - use existing MODSIM simulation of basic SCI transactions (this is fast, and allows simulation of > 1000 node systems)
  - combine this with IEEE C-code for cache coherency (> 7000 lines of complex, but efficient code).
- **Status**
  - Bugs in C-code are being corrected rapidly
  - still some fundamental problems with MODSIM integration: MODSIM does not allow calls to asynchronous TELL methods from within (nested) C-functions.
  - provisional, low performance implementation available for SUN/SPARC.

Cache Coherency: case study

Ring with 1 Memory and 9 Processors



- **Compare noncoherent nread64 with coherent mread64**
  - Both transactions read 64 bytes from memory. Noncoherent nread64 may cache data "locally".
- **Processor operations**
  - All processors do successive read operations to memory addresses 0, 64, 128, ... etc. (to avoid repeated access to already cached data)
- **Three cases:**
  - lightly loaded ringlet: wait 5 μs after each read operation
  - moderately loaded ringlet: wait 1 μs after each read operation
  - heavily loaded ringlet: no wait between successive read operations
- **Simulation parameters**
  - Assume 10 ns propagation delay for an idle node + cable
  - All internal, memory and cache buses operate at 50 Mhz, 32 bits parallel (or, equivalently, 64 bits @ 25Mhz)
  - 1 kbyte cache size

Expected latency of nread64

Operation	Requester 32 bits @ 50MHz	Responder 32 bits @ 50MHz	Interconnect 16 bits @ 500MHz
copy request header interface -> node chip	80		
transfer request processor -> memory			70[1]
copy request header node chip -> interface		80	
copy response header interface -> node chip		80	
copy response data interface -> node chip		320	
transfer response memory -> processor			134[2]
copy response header node chip -> interface	80		
copy response data node chip -> interface	320		
subtotal	480	480	204

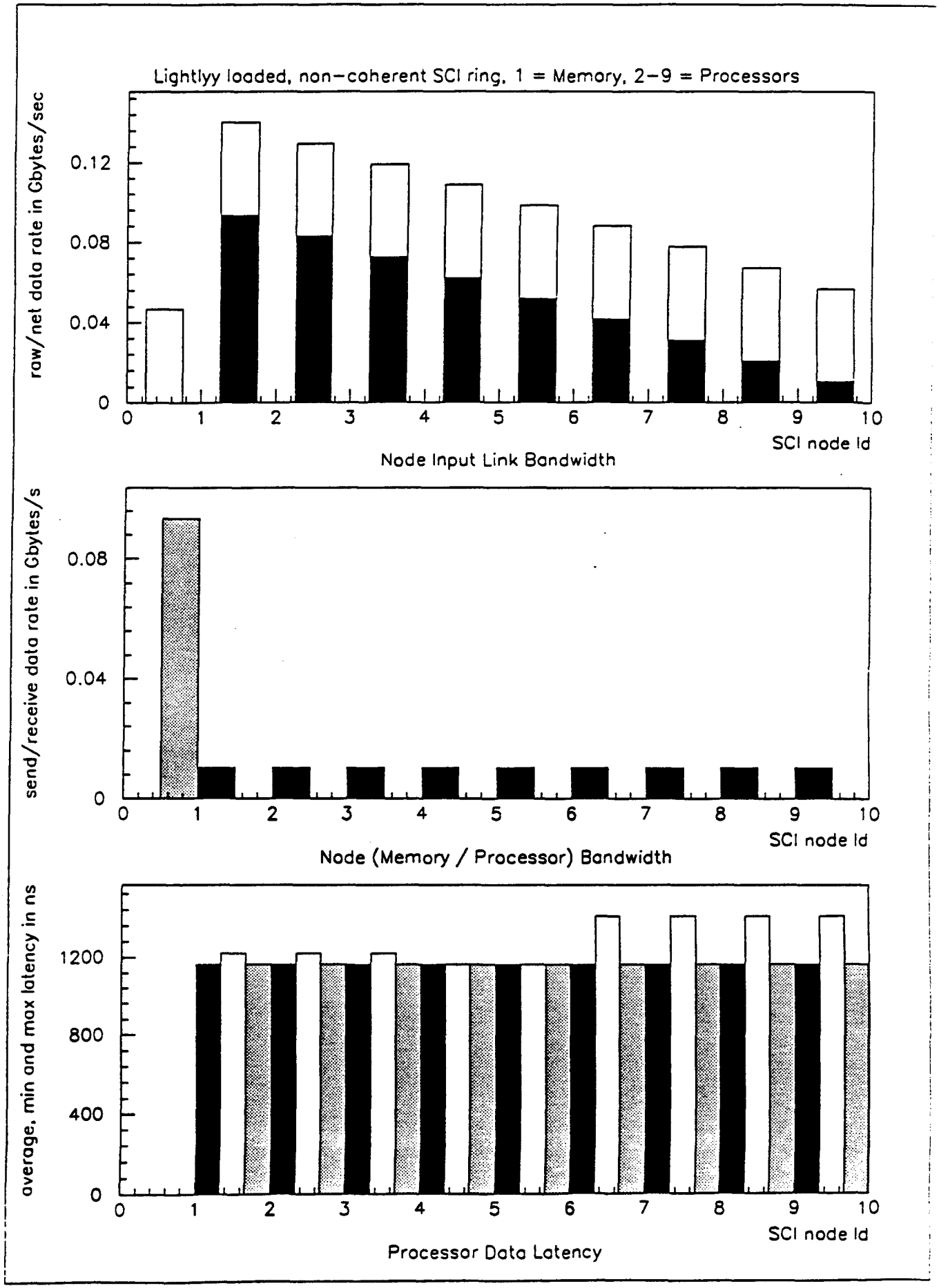
total transaction (ns) 1164

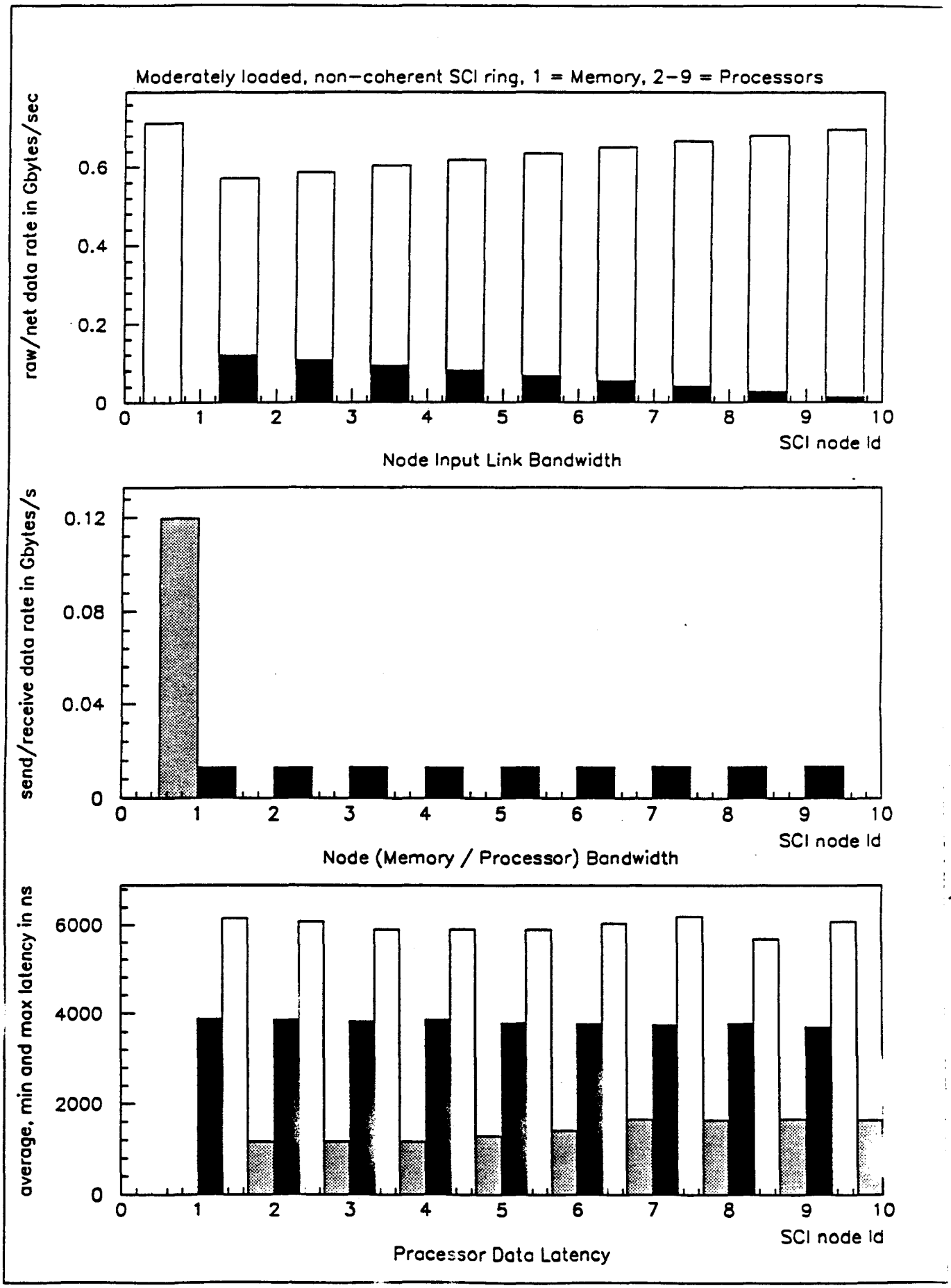
[1] header (20) + 5 link/node delays (5*10)

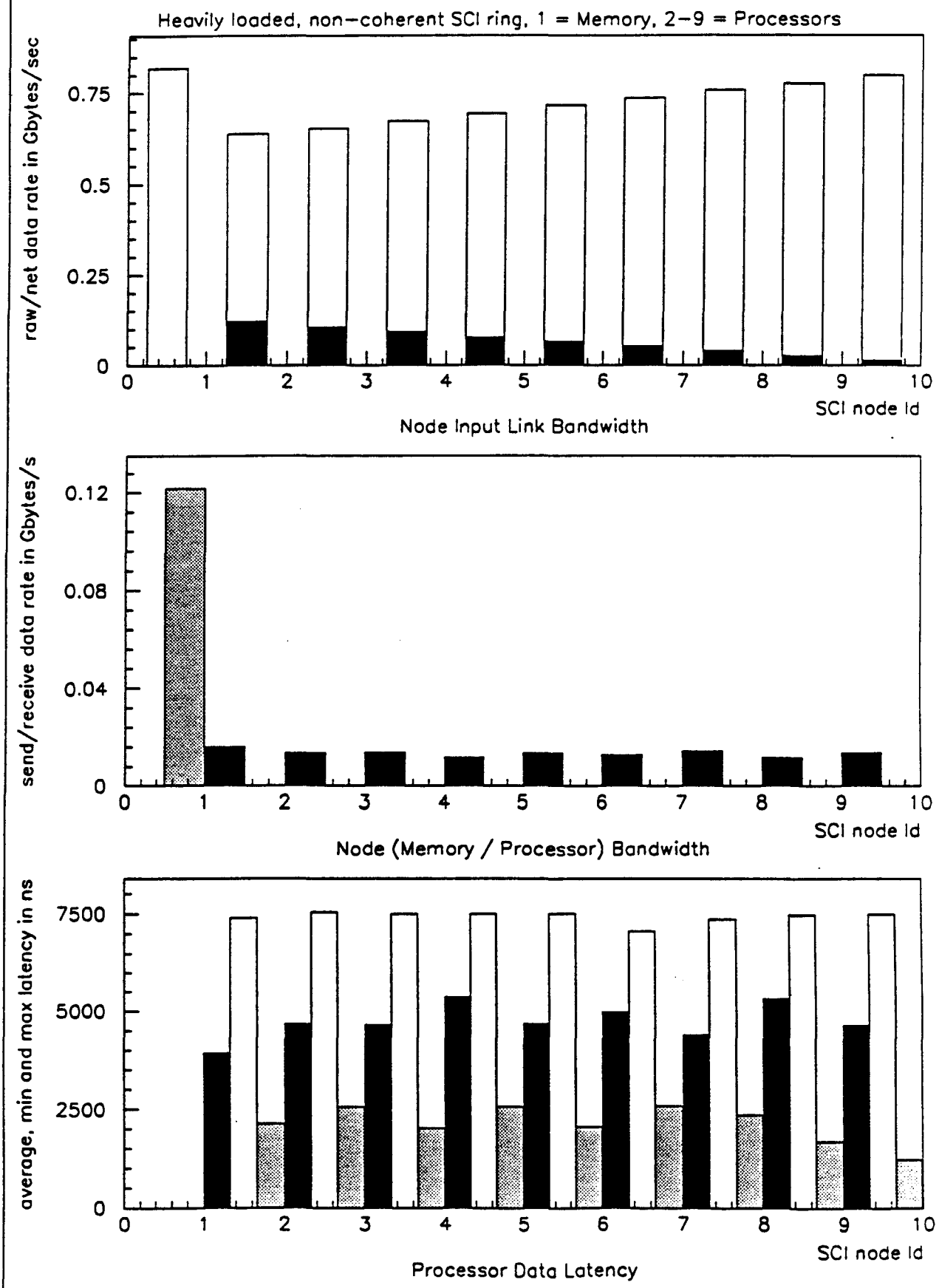
[2] response (84) + 5 link/node delays (5*10)

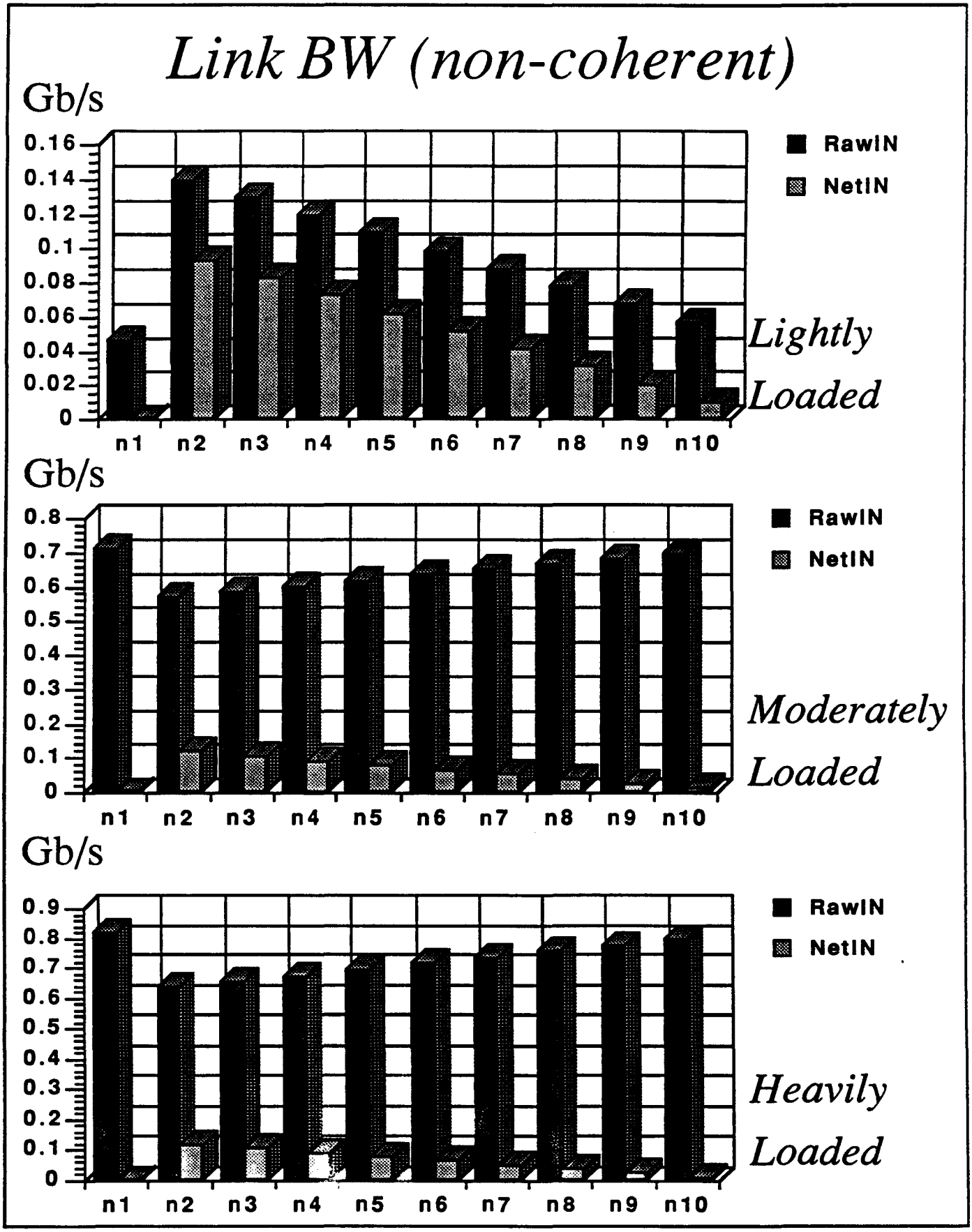
nread64: simulation results

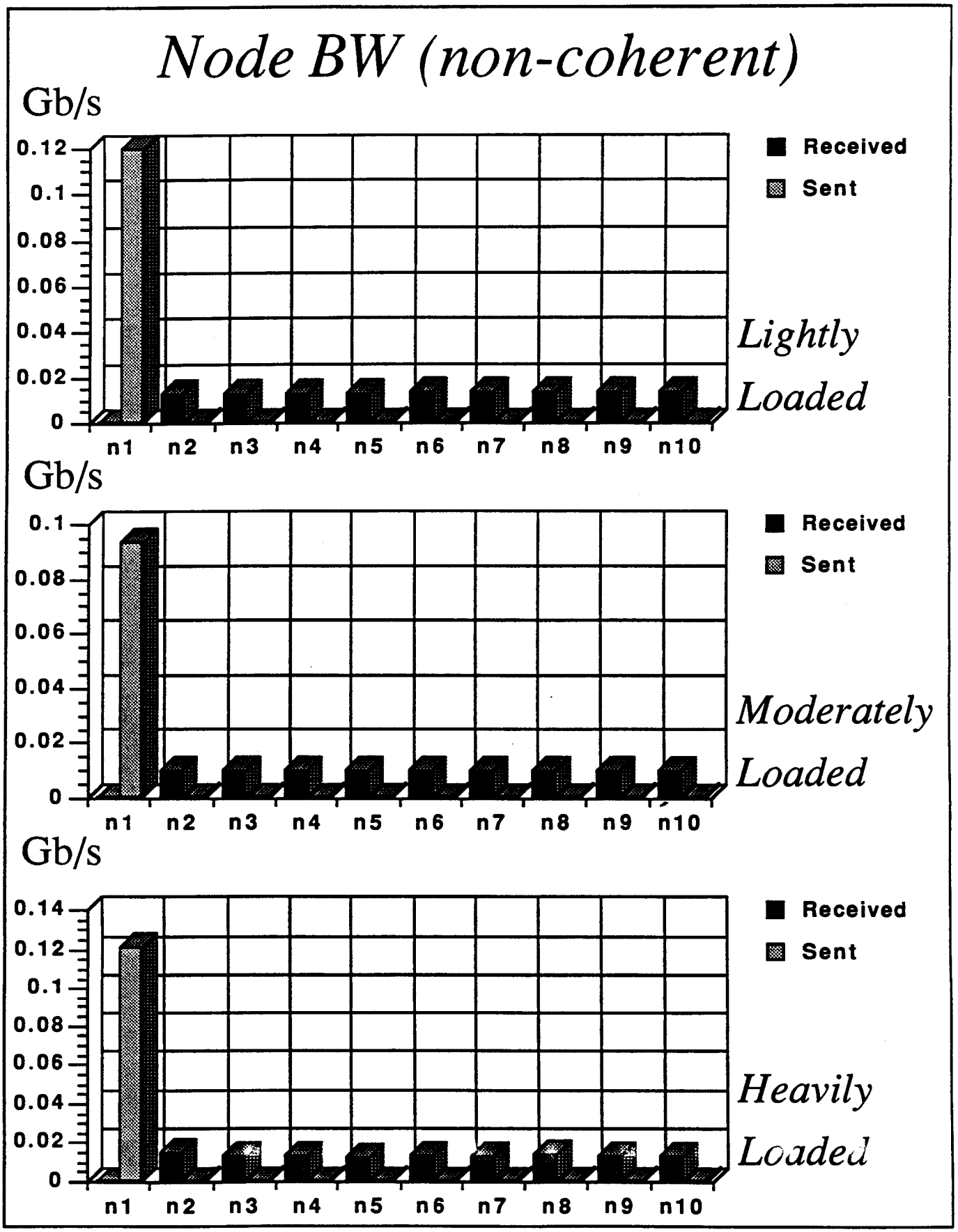
- **lightly loaded (5 μs between reads)**
  - Minimum latency: 1164 ns (as expected). This is almost equal to the average latency, indicating low SCI utilisation (no "collisions").
  - Processor input bandwidth of ≈ 10 Mbytes/s follows from delays between successive operations (64 bytes every 1.164 + 5 μs).
    Memory output bandwidth of ≈ 90 Mbytes/s is below its theoretical limit of 120 Mbytes/s as determined by its dead-time: 480 (interface) + 20 (header) + 50 (ring delay for echo reception) ns.
  - SCI link utilisation (≈ 140 Mbytes/s on memory output link) consistent with loading; absense of "retry" load.
- **moderately loaded (1 μs between reads)**
  - Average latency well above minimum latency, indicating loading of the interconnect ("collisions")
  - Memory output bandwidth (≈ 120 Mbytes/s) at its theoretical maximim; this restricts the processor input bandwidth to ≈ 13 Mbytes/s (1/9nth of memory bandwidth)
  - SCI link utilisation high. Peak rate (≈ 700 Mbytes/s) on input link of memory, caused by processor requests; many of these are "retries" because of memory saturation.
- **heavily loaded (no delay between reads)**
  - Average latency large (≈ 5 000 ns) and larger spread indicating heavy loading.
  - No change in memory / processor bandwidth with respect to a moderately loaded system (saturation)
  - SCI interconnects close to saturation at ≈ 800 Mbytes/s.


Lightlyy loaded, non-coherent SCI ring, 1 = Memory, 2-9 = Processors
raw/net data rate in Gbytes/sec
0.12
0.08
0.04
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.08
0.04
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
1200
800
400
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency

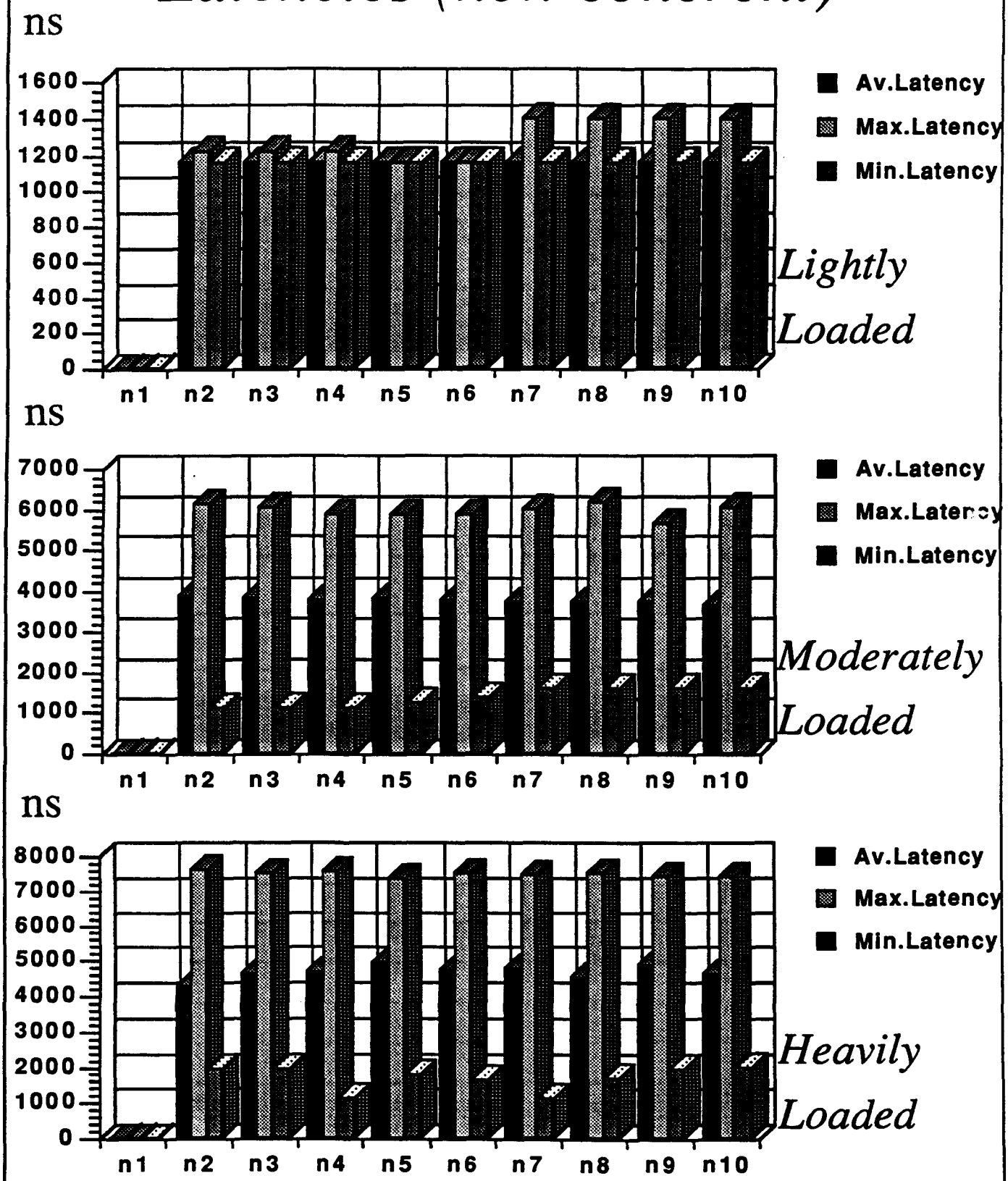

Moderately loaded, non-coherent SCI ring, 1 = Memory, 2-9 = Processors
raw/net data rate in Gbytes/sec
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.12
0.08
0.04
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
6000
4000
2000
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency


Heavily loaded, non-coherent SCI ring, 1 = Memory, 2-9 = Processors
raw/net data rate in Gbytes/sec
0.75
0.5
0.25
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.12
0.08
0.04
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
7500
5000
2500
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency


Link BW (non-coherent)
Gb/s
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
RawIN
NetIN
Lightly Loaded
Gb/s
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
RawIN
NetIN
Moderately Loaded
Gb/s
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
RawIN
NetIN
Heavily Loaded


Node BW (non-coherent)
Gb/s
0.12
0.1
0.08
0.06
0.04
0.02
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Lightly Loaded
Gb/s
0.1
0.08
0.06
0.04
0.02
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Moderately Loaded
Gb/s
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Heavily Loaded

Latencies (non-coherent)



Coherency Simulation Implementation

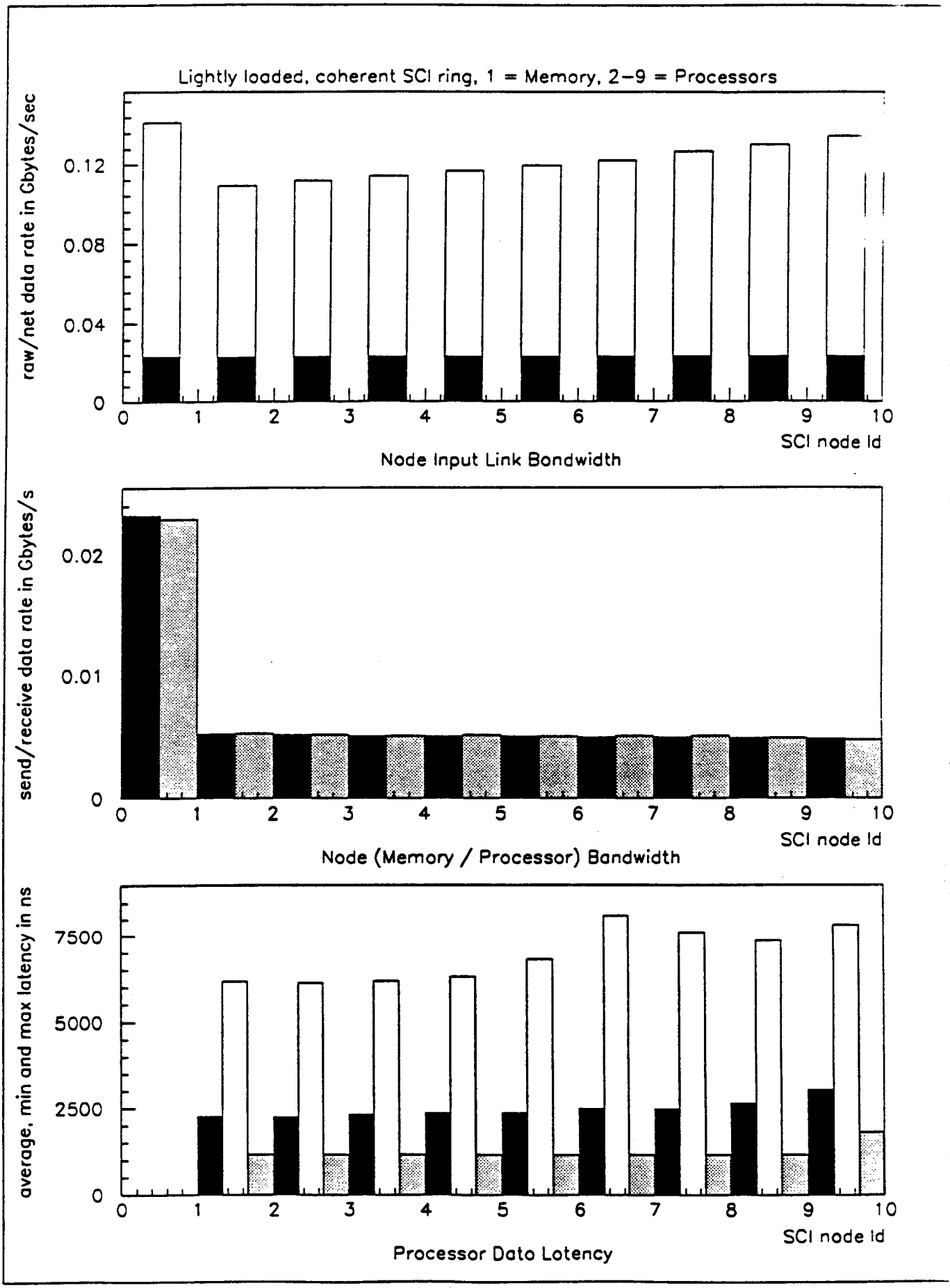
- **Transactions are implemented in MODSIM as asynchronous "TELL" methods**
- **Coherent transaction generate several elementary SCI transactions**
- **IEEE C-code ("synchronous") determines transaction sequence as a function of:**
  - cache state (defined by cache tags)
  - memory state (defined by memory tags)
  - optimisation level (standard defines minimal, typical and full set. Vendor specific optimisations are foreseen).
- **Coherent cache operations 4 phases:**
  - find a cache line (no transactions)
  - setup phase (may generate multiple transactions)
  - execute phase (no transactions)
  - cleanup phase (may generate multiple transactions)
- **measured latencies**
  - do not include the cleanup phase (assumed to be in parallel with processor execution)
- **memory states, cache states, transistions**
  - memory states: 12 (full), 3 (typical)
  - cache states: 137 (full), 24 (typical)
  - memory commands: 13 (full), 5 (typical)
  - cache commands: 49 (full), 9 (typical)
- **processors accessing a cache in a transient state go into a "spin loop"**

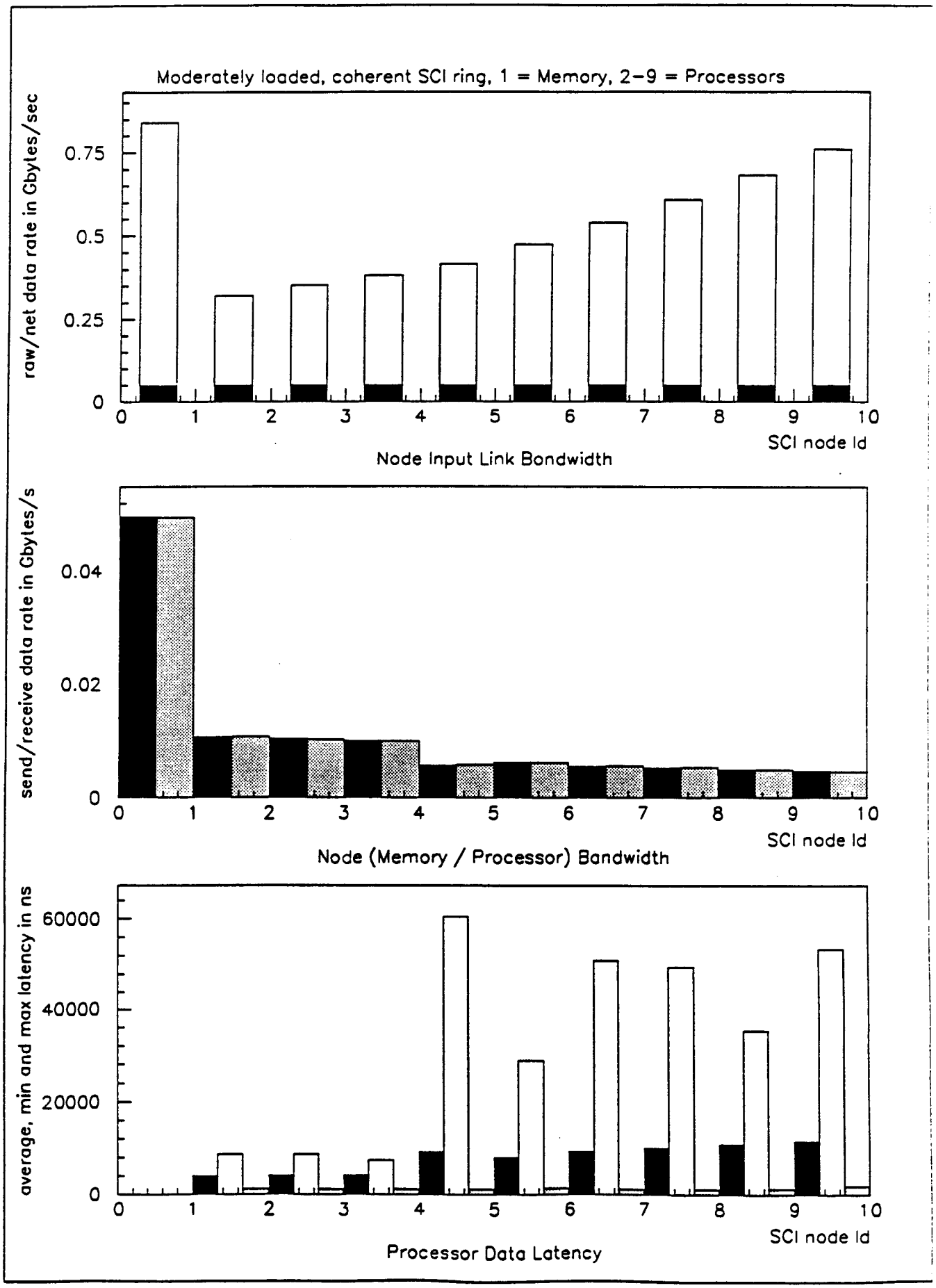
"Typical" Cache Load

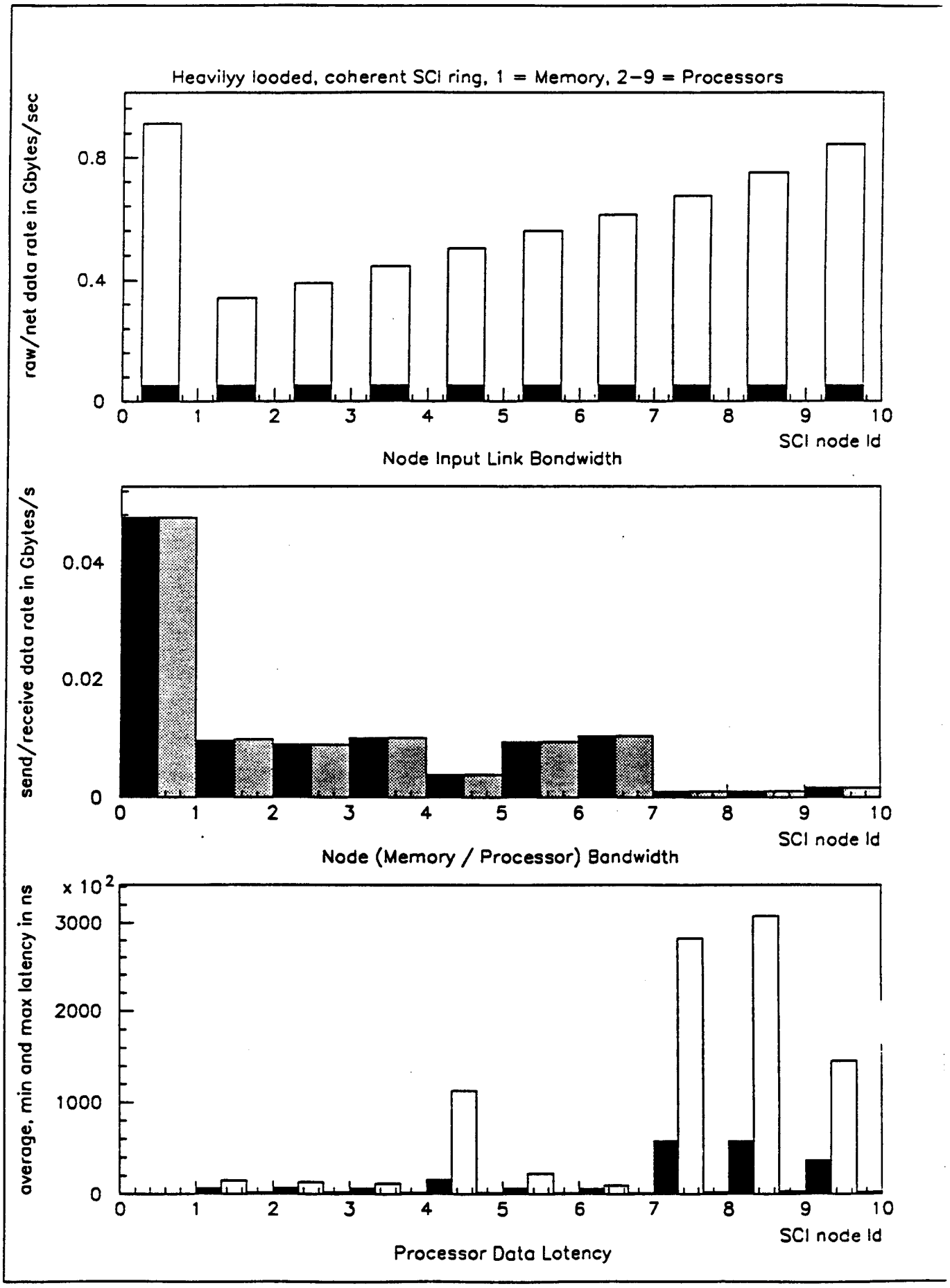
- **TypicalFindLine**
  - Find a cache entry. May result in a transaction TypicalRolloutEntry.
- **TypicalLoadSetup**
  - Request a cacheline from memory (mread64). Memory either returns the cacheline, or refers the requester to another cache, resulting in another transaction (cread64).
- **TypicalLoad**
  - Does not involve other transactions. data is now available to the processor or cache controller. latencies are measured from the beginning (Typical FindLine) till this point (data available).
- **TypicalCleanup**
  - Sharing list insertion (establishing the correct pointer chain) may requires another transaction, contributing to cache "dead time".

mread64: Preliminary Results

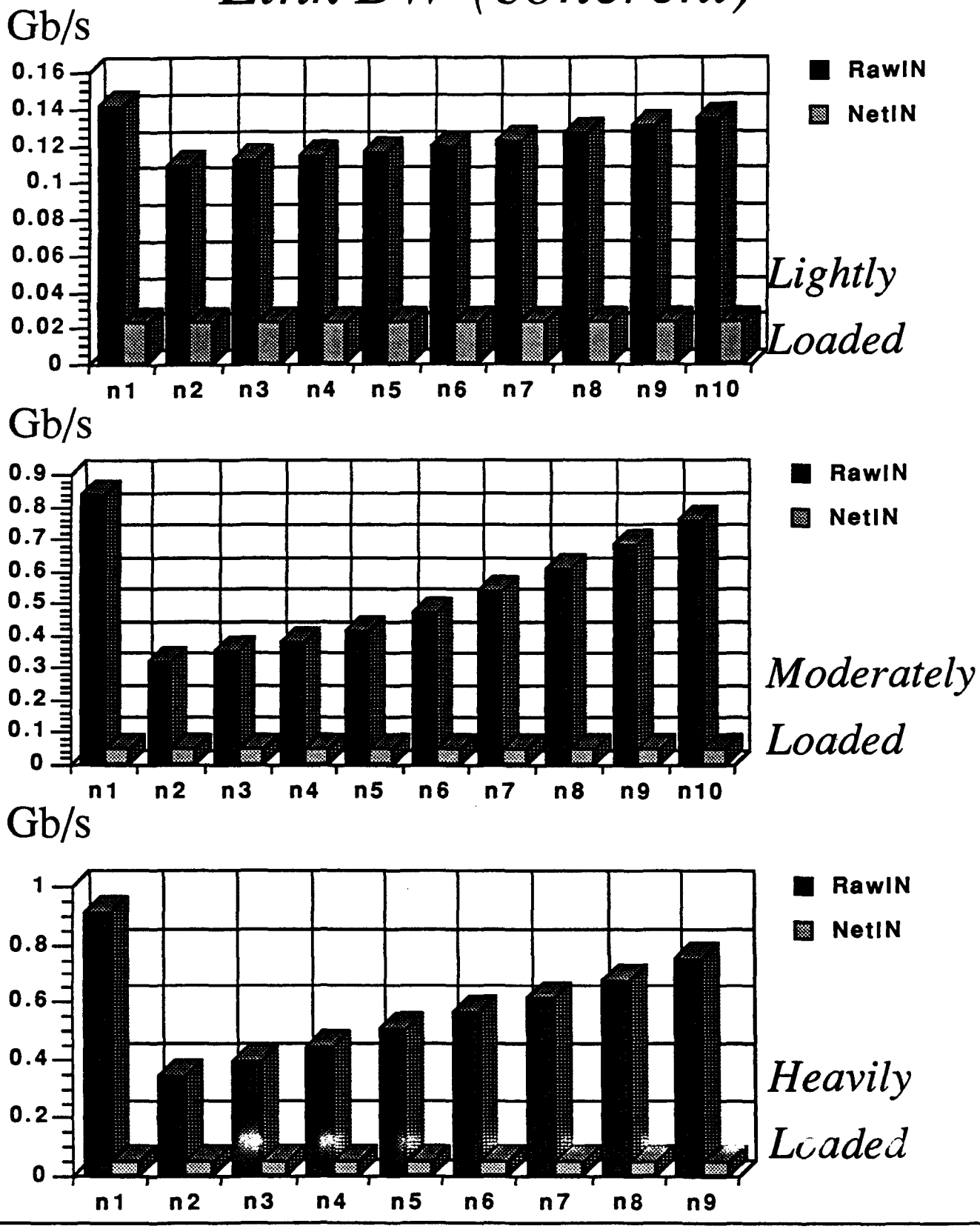
- **latencies**
  - Minimum latency is identical to the noncoherent latency
  - Average latency determined by 3 transactions (one to roll out a used cacheline, one to memory to identify the data source, one to another cache to obtain the data), resulting in ≈ 2300 ns
  - Maximum latency can be much higher, due to transient cache states (cache not yet settled) resulting in "spin loop transactions" (≈ 6500 ns).
  - For moderately and heavily loaded systems, these loops are excessively long (60 - 300 μs ! ! !). Needs further investigation.
- **memory/processor bandwidth**
  - As expected, the load on the memory is diminished by a factor 2-3 and spread over the caches.
  - There are read and write operations to memory/caches, due to cache flushing.
  - Because of longer latencies, processor input bandwidth is reduced (there may be additional dead-time after a cache has received its data during the cleanup phase).
  - Anomalous latencies cause processor starvation. Needs further investigation ! ! !
- **interconnect bandwidth**
  - peak rates on links is reduced, but the average over all links is higher, due to "spreading" of memory bandwidth. "Spin loops" may generate some "retry" traffic. This is confirmed by the total traffic on all links:
    - noncoherent system: 0.93 Gb/s raw, 0.47 Gb/s net
    - coherent system: 1.23 Gb/s raw, 0.23 Gb/s net.

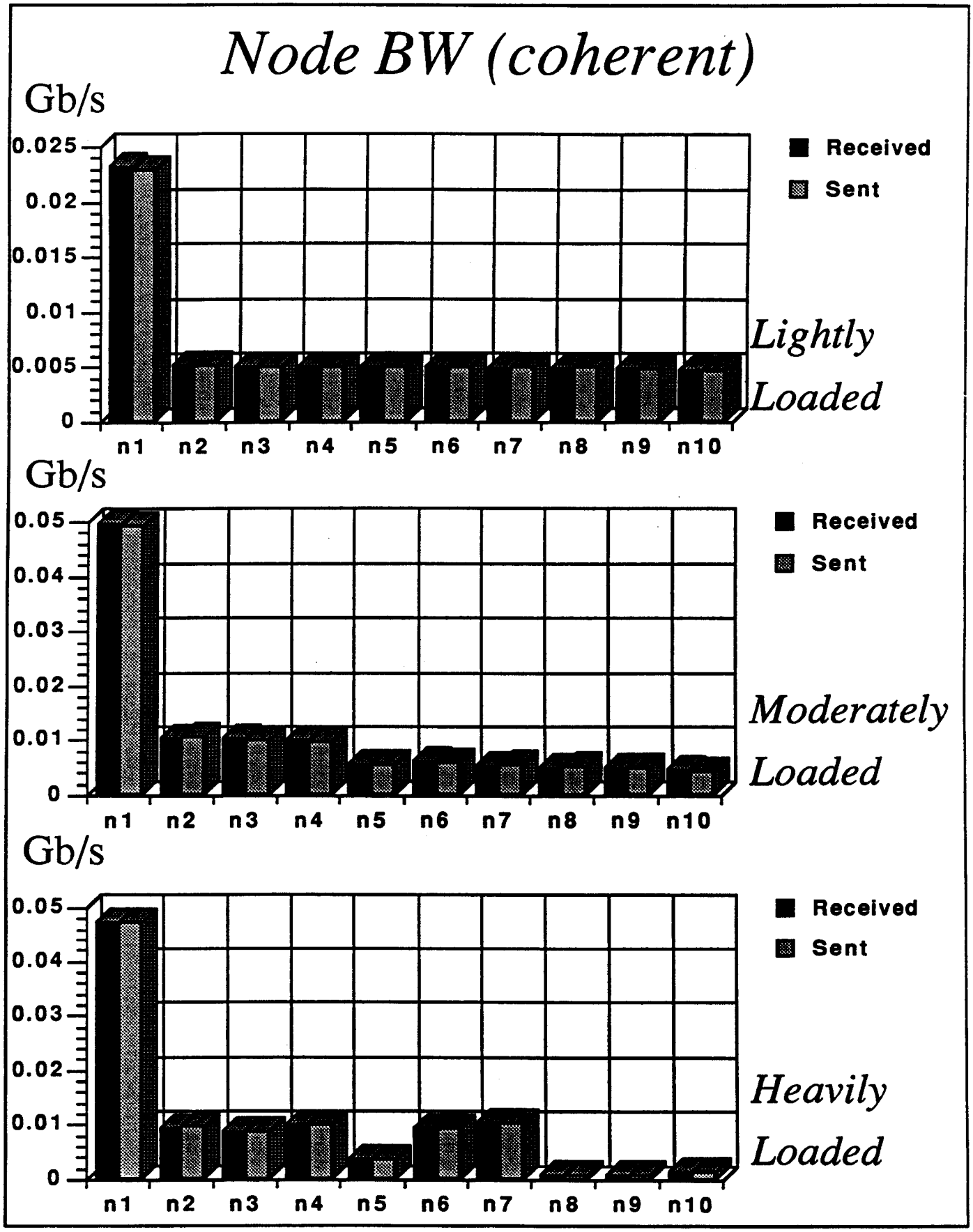

Lightly loaded, coherent SCI ring, 1 = Memory, 2–9 = Processors
raw/net data rate in Gbytes/sec
0.12
0.08
0.04
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.02
0.01
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
7500
5000
2500
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency


Moderately loaded, coherent SCI ring, 1 = Memory, 2–9 = Processors
raw/net data rate in Gbytes/sec
0.75
0.5
0.25
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.04
0.02
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency

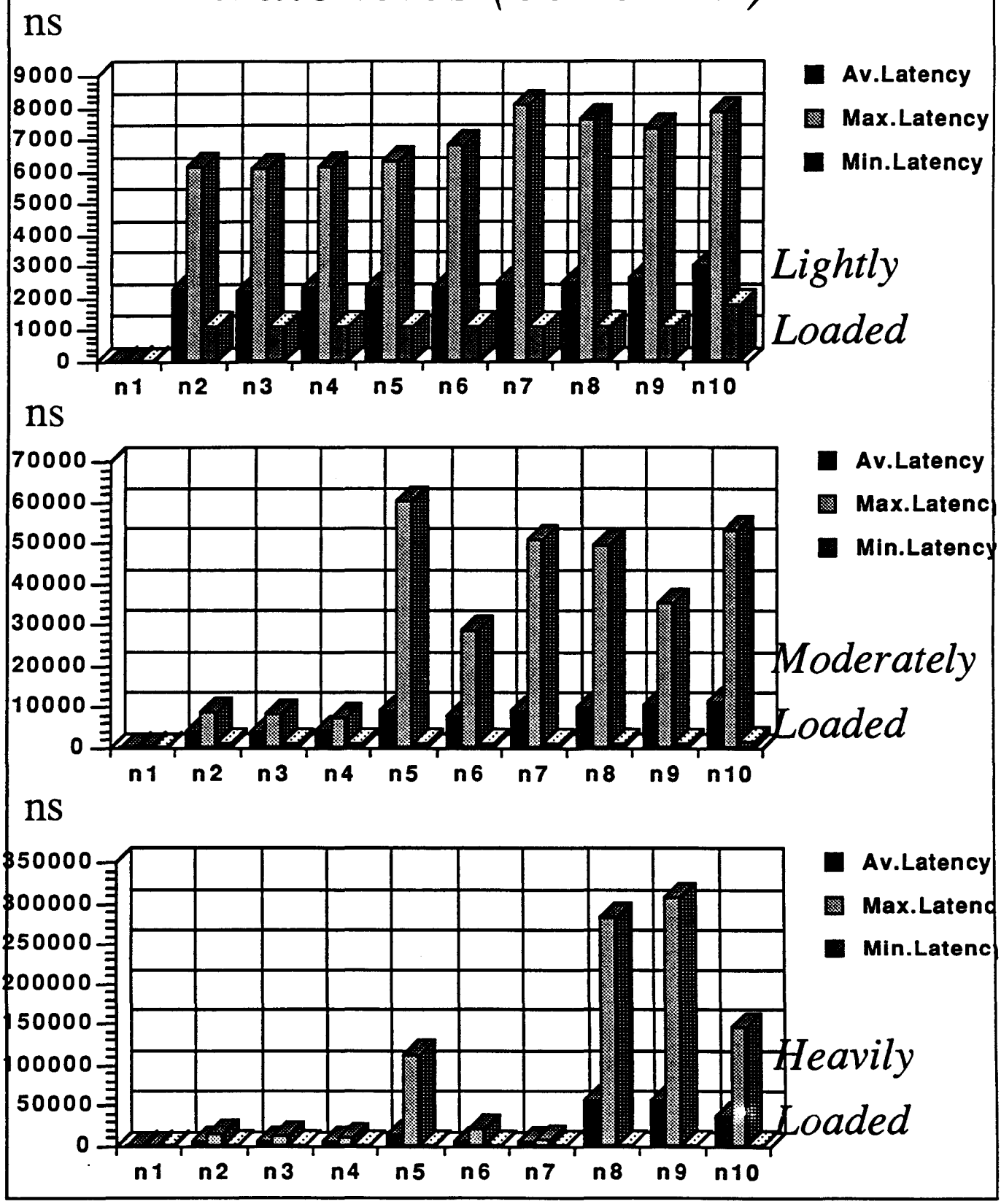

Heavilyy looded, coherent SCI ring, 1 = Memory, 2–9 = Processors
raw/net data rate in Gbytes/sec
0.8
0.4
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node Input Link Bandwidth
send/receive data rate in Gbytes/s
0.04
0.02
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Node (Memory / Processor) Bandwidth
average, min and max latency in ns
x 10^2
3000
2000
1000
0
0 1 2 3 4 5 6 7 8 9 10
SCI node Id
Processor Data Latency

Link BW (coherent)




Node BW (coherent)
Gb/s
0.025
0.02
0.015
0.01
0.005
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Lightly Loaded
Gb/s
0.05
0.04
0.03
0.02
0.01
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Moderately Loaded
Gb/s
0.05
0.04
0.03
0.02
0.01
0
n1 n2 n3 n4 n5 n6 n7 n8 n9 n10
Received
Sent
Heavily Loaded

Latencies (coherent)



Simulation of SCI protocols in ModSim II

Modern system design relies more and more on computer modelling and the Scalable Coherent Interface (SCI) is no exception. Different tools are used to simulate different aspects. The standard is defined by the IEEE as executable C-code, which allows conformance verification. VERILOG has been used by Dolphin Server Technology (Oslo, Norway) to design the SCI "node chip" interface. The behavioural simulation of the node chip in VERILOG has been cross checked against results obtained from the IEEE C-code using identical stimuli. The latter required a multi-thread environment which was provided by the SUN/SPARC implementation of "light weight processes". The Department of Informatics of the University of Oslo has used models written in both SIMULA and C++ to model small SCI based systems.

At CERN we have used ModSim II to model large (> 1000 nodes) SCI based Data Acquisition Systems for LHC. The model simulates accurately SCI protocols at the packet level. Since SCI packets vary in size, the simulation is necessarily asynchronous with a typical time resolution of ~ 50 ns (the average packet length at transmission speeds of one Gbyte/s). The simulation program (SCIMP, SCI Modelling Program) typically calculates the traffic on SCI interconnects, the flow of data in and out processors or memories in a network consisting of SCI rings and interconnects. The model takes into account SCI protocols for bandwidth allocation (roughly the equivalent of arbitration in bus based systems) and retransmission of packets to overloaded SCI memories or processors. Recently, work has started to include cache coherency in the simulation.

The different simulations are complementary. SCIMP does not simulate the content of packets which are exchanged between SCI nodes, although it could easily be extended to do this. It does not simulate the exact details of how the hardware emits or strips symbols of the interconnect. SCIMP models the load on an SCI network, but does not generate test patterns. Recently, the Physics Department of the University of Oslo has acquired ModSim with the aim of providing more functionality for the simulation of LHC Data Acquisition systems.

The experience using ModSim, the design of SCIMP, results obtained so far will be presented.

ModSim is a trade mark of CACI Products Company (La Jolla, Ca.)

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT NO. UIUCDCS-R-92-1745 UILU-ENG-92-1725

VISUAL DESIGN WITH vVHDL

by

Eric J. Golin, Michael J. Haney, Eric Hughes, Diana Miller-Karlow, and George Tharakan

April 1992

REPORT NO. UIUCDCS-R-92-1745

VISUAL DESIGN WITH vVHDL

by

Eric J. Golin, Michael J. Haney, Eric Hughes, Diana Miller-Karlow and George Tharakan

April 1992

DEPARTMENT OF COMPUTER SCIENCE
1304 W. SPRINGFIELD AVENUE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, IL 61801

Supported by Texas National Research Laboratory Commission, project #047
National Science Foundation - Grant CCR-9108931

Visual Design with vVHDL

1 Introduction

Computer-aided design (CAD) tools revolutionized the design of electronic components by allowing convenient management of large amounts of data. Hardware description languages (HDLs) are an important component of CAD tools. An HDL description of a design is a precise representation that can be used for documenting, communicating, and simulating the design. Modern HDLs allow electronic designs to be described structurally and/or behaviorally and thus provide an effective mechanism for developing designs as they evolve from abstraction to reality.

One difficulty in using HDLs to develop designs is the cumbersome nature of the HDL syntax. An HDL describes a complex, multifaceted system using a one-dimensional textual representation, which results in the obfuscation of many aspects of the design. One way to improve the usability of HDLs is to employ a visual language that represents HDL constructs graphically, rather than textually. A visual language "program" is a two-dimensional arrangement of pictorial elements. A visual hardware description language makes an HDL easier to use by providing a graphical interface to the language.

A visual approach to hardware design is very natural. Traditionally, engineers have developed hardware descriptions based on schematic circuit diagrams, which are a visual notation. Although these diagrams are very useful for describing gate-level designs, complex systems also require modeling components in terms of behavior, to allow for simulation and testing throughout the design process. Our goal is to develop a visual language for hardware design that allows the graphical specification of both structure and behavior of hardware components.

We have developed a visual programming language based on the VHSIC Hardware Description Language [1] (VHDL), which may be thought of as providing a visual syntax for the VHDL language. The visual syntax makes it easier for engineers to develop designs by easing the burden of programming in VHDL, and by providing a uniform visual approach to creating models combining structure and behavior. The underlying computational model is VHDL, and the semantics of the visual language are defined by a translation into textual VHDL code.

The development of visual VHDL (vVHDL) is the consequence of research in complex system design for the SDC[2] of the Superconducting Super Collider project. This paper demonstrates the use of vVHDL for an example taken from the SDC. The visual representation of a part of a data collection circuit, along with the VHDL code generated from the pictures, are described. Our experience in developing this design has shown that a visual approach to hardware description languages can have clear benefits in the development and understanding of system designs.

This paper presents the use and syntax of the vVHDL language. Section 2 gives the motivation and goals of vVHDL as well as an introduction to visual programming languages. Section 3 introduces the vVHDL language and its syntax. Section 4 contains an example of a FIFO designed using vVHDL. Finally, Section 5 discusses some conclusions and future directions for vVHDL.

2 A Visual Language Approach to Hardware Description

2.1 vVHDL Motivation and Goals

A great deal of research effort has been devoted to providing an integrated programming environment for VHDL [3, 4, 5]. In addition, several commercial CAD vendors (Vista Technologies, Vantage Analysis Systems, Ascent Technologies, i-Logix, Inc., etc.) have released improved VHDL tools that address many of the difficulties inherent to textual VHDL. However, all of these attempts share the common failing that they lack a formal visual grammar and do not allow a complete visual representation of VHDL. vVHDL endeavors to rectify these shortcomings.

The prototype of vVHDL was developed with the following goals in mind:

- To be easier to use and understand than textual VHDL.
- To provide for visual mixed-level models at all levels of the design process that schematic capture tools do not allow.
- To create a visual representation for electronic designs that will be understandable to the VHDL non-user.
- To alleviate the burden of VHDL syntax constraints.

2.2 Visual Programming Languages

Visual programming languages form programs by combining graphical symbols such as lines, circles, rectangles, and text fragments. A *visual program* is a picture specifying a computation. A visual program is not an arbitrary picture, but rather a diagram constructed according to a set of rules describing the class of valid diagrams. A visual programming language is a family of diagrams. The rules specify the syntax of the language.

Classical examples of visual programming languages include flowcharts, dataflow diagrams and finite state diagrams. Recent research has investigated visual programming languages in a wide variety of application domains, including object-oriented programming [6], concurrent programming [7], image processing [8] and data structures [9]. Two key factors distinguish visual programming languages from traditional textual programming languages:

- Programming languages such as C use text to form the lexical elements (e.g., identifiers, operators, constants). Visual languages use graphical elements such as shapes, icons, and lines, as well as pieces of text, as the basic constituents of a program.
- In a textual language, a program is a one-dimensional string, while in a visual language a program is arranged as a multidimensional picture (typically two- or 2.5-dimensional).

2.3 The VHDL Model

Before vVHDL could be designed, a consistent classification of the available constructs had to be developed. We use the term *design units* to refer to the most important VHDL constructs: entity, architecture, package, package body, and configuration. An *entity* is a "black-box" description of a component. An *architecture* is an implementation of the function of an entity. There can be many

architectures for a given entity. A *package* describes a related set of definitions. The *package body* gives implementations for the functions and procedures included in the package. A package has only one package body. Finally, a *configuration* can be used to specify the architectures to be used for components in an entity. An entity may have any number of configurations.

The remaining VHDL constructs occur within the scope of a design unit. The constructs have been prioritized, based on user experience and a study of commercial products. Constructs with higher priority are more easily accessible to the designer and have a simpler representation in the visual language. Constructs will be added to the prototype visual VHDL tool in order of priority.

3 vVHDL - A Visual Design Tool

3.1 The vVHDL Language

In designing a visual language, it is important that the language both *redundantly record* the important information and *reveal* the underlying meaning [10]. vVHDL uses the Shape and Flow paradigm to achieve this goal. In this paradigm, similar VHDL constructs have similar shapes, and flow in sequential statements is shown with arrows in very much the same way as a flow chart.

The Shape and Flow paradigm represents encoded information in a way that is *relevant* by choosing shapes and icons suggestive of the objects they represent. It *redundantly records* the important constructs by representing similar objects with similar shapes. The representation *reveals* the underlying meaning by representing concurrent and sequential statements with different shapes, and the prominence of the primitives reveals the hierarchical nature of VHDL constructs. Thus, the function of a vVHDL program is contained in its topology, and it is not necessary to refer to the textual details. The following sections describe the relationship between the vVDHL primitives.

3.2 Declarations and High Level Primitives

There are several constructs in VHDL that are used for declarations. These include the design unit that is used for declaring a primary unit, the entity declaration that is used for declaring the interface to a design, and the package declaration that is used for declaring the interface to a package. Because of their similarities, declaration units in vVHDL have a thick rectangle as their main component as shown in Figure 1. These particular components are referred to often in the design process, and the thick box makes them stand out so that they are easily located.

The most important abstract structures in VHDL are the entity declaration and the architecture body. Therefore, these two primitives should possess a distinctive shape. As shown in Figure 2, both constructs are composed of a large rectangle topped by a smaller rectangle, and both are identified by the entity name in the upper rectangle.

Another pair of related primitives, the architecture body and the package body, are used to specify the behavior of items declared previously. Figure 3 shows these two constructs. Both the architecture body and the package body primitives are drawn with a thin rectangle as their dominant shape. Thus, these constructs have similar shapes and are less obvious than their respective declaration units. The hierarchical relationship between the primitives is emphasized by making them less conspicuous.

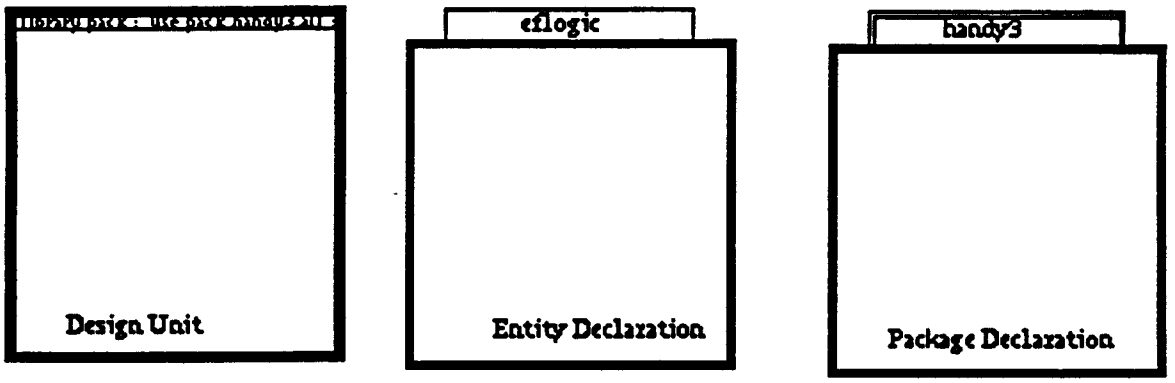


Figure 1: Declaration Units in vVHDL

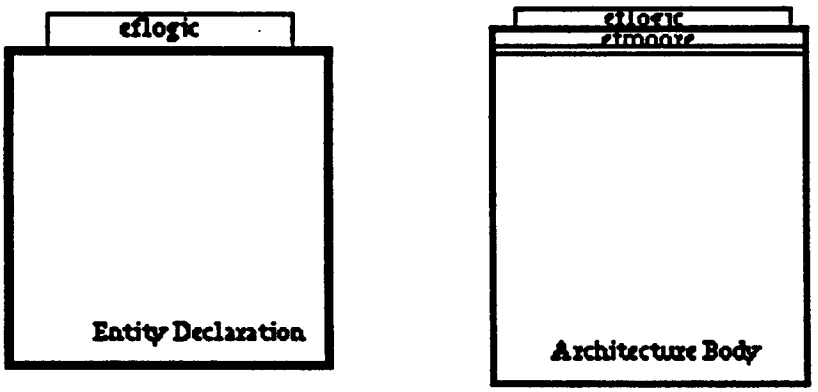


Figure 2: An Entity Declaration and Architecture Body in vVHDL

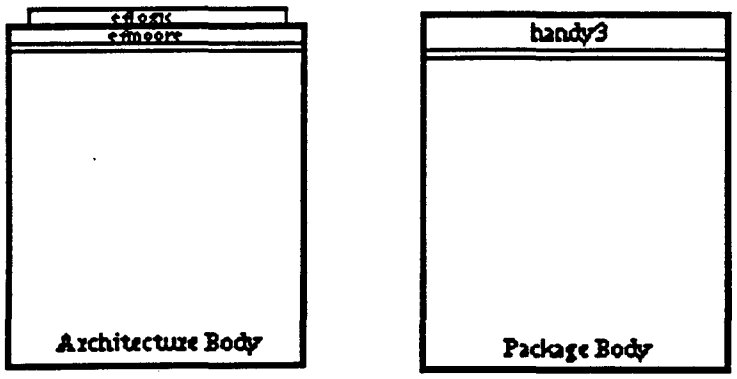


Figure 3: An Architecture Body and Package Body in vVHDL

3.3 Sequential and Concurrent Statements

Another group of related primitives are the concurrent statements shown in Figure 4. There are six concurrent constructs supported: the process statement, the block statement, the concurrent procedure call, the internal and external signal assignments, and the component instantiation. The dominant shape of these figures is a rectangle with rounded corners. The lines are thinner than the rectangles used in the architecture body which shows that concurrent statements must be contained by architecture bodies. An optional label may be placed in the rectangle atop the concurrent statements.

The sequential statements, in Figure 5, are similar to the concurrent statements. Sequential statements have a rectangle as their dominant shape to distinguish them from the concurrent statements. This rectangle has the same thickness as concurrent statements to show that sequential and concurrent statements are comparable.

The final set of similar primitives are the assignment statements. VHDL allows assignment to both internal and external signals and variables, and both sequential and concurrent signal assignments are permitted. The assignment statements are shown in Figures 4 and 5. Each of these constructs has three internal components: an upper rectangle, a lower rectangle and an icon. The upper rectangle holds the name of the signal or variable that is being assigned to, the lower rectangle holds the value that is being assigned, and the icon represents the kind of the assignment. An internal signal is represented by a bus icon, an external signal is represented by an airplane icon, and a variable is represented by a bucket icon. The shape of the external part of the assignment statement depends upon whether it is a concurrent statement or a sequential statement.

3.4 The vVHDL Programming Environment

A *visual programming environment* is a tool for the construction, manipulation and execution of visual programs. One problem in the development of new visual programming languages is the difficulty in producing environments for visual programming. Traditional tools for processing programming languages, such as editors, compilers and programming environments, are text-based.

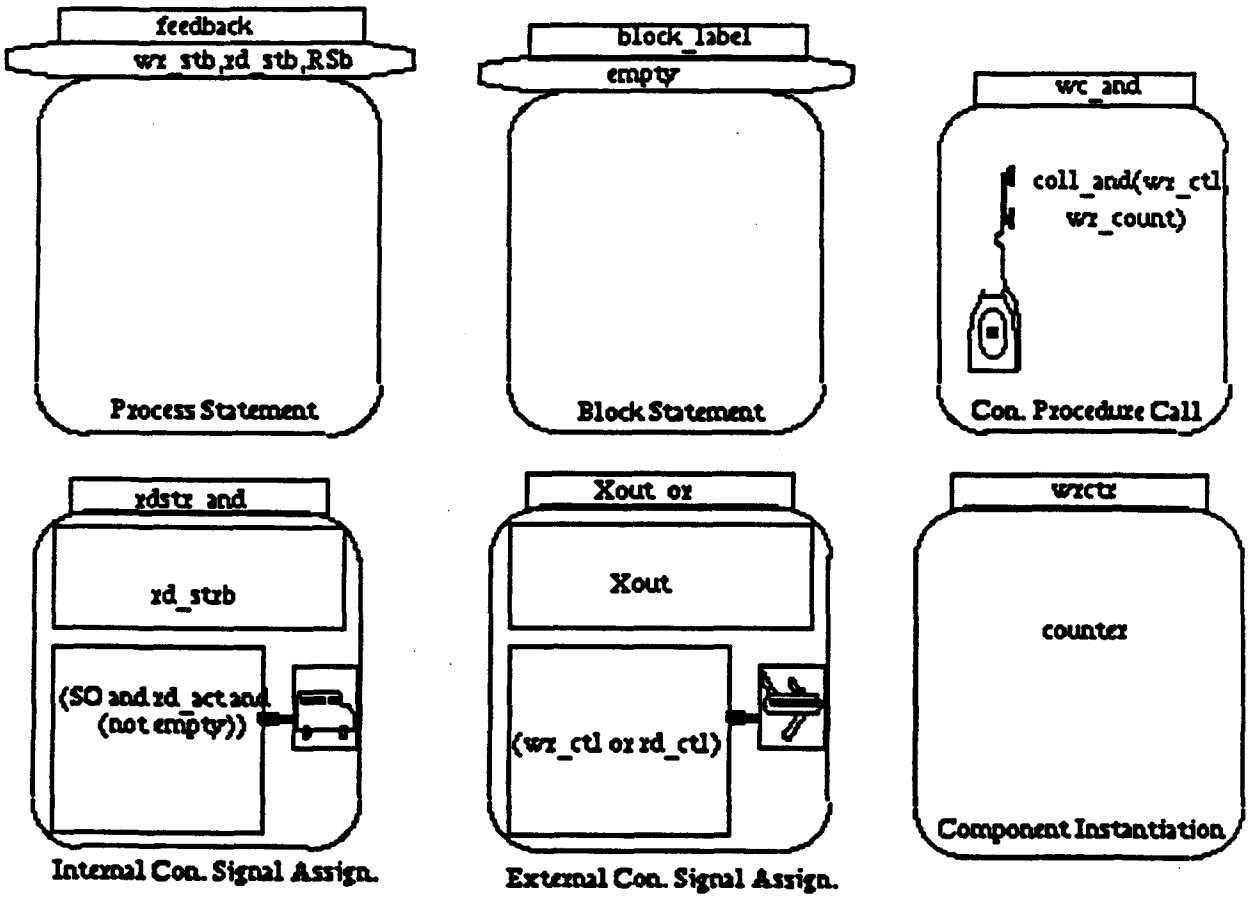


Figure 4: Concurrent Statements in vVHDL

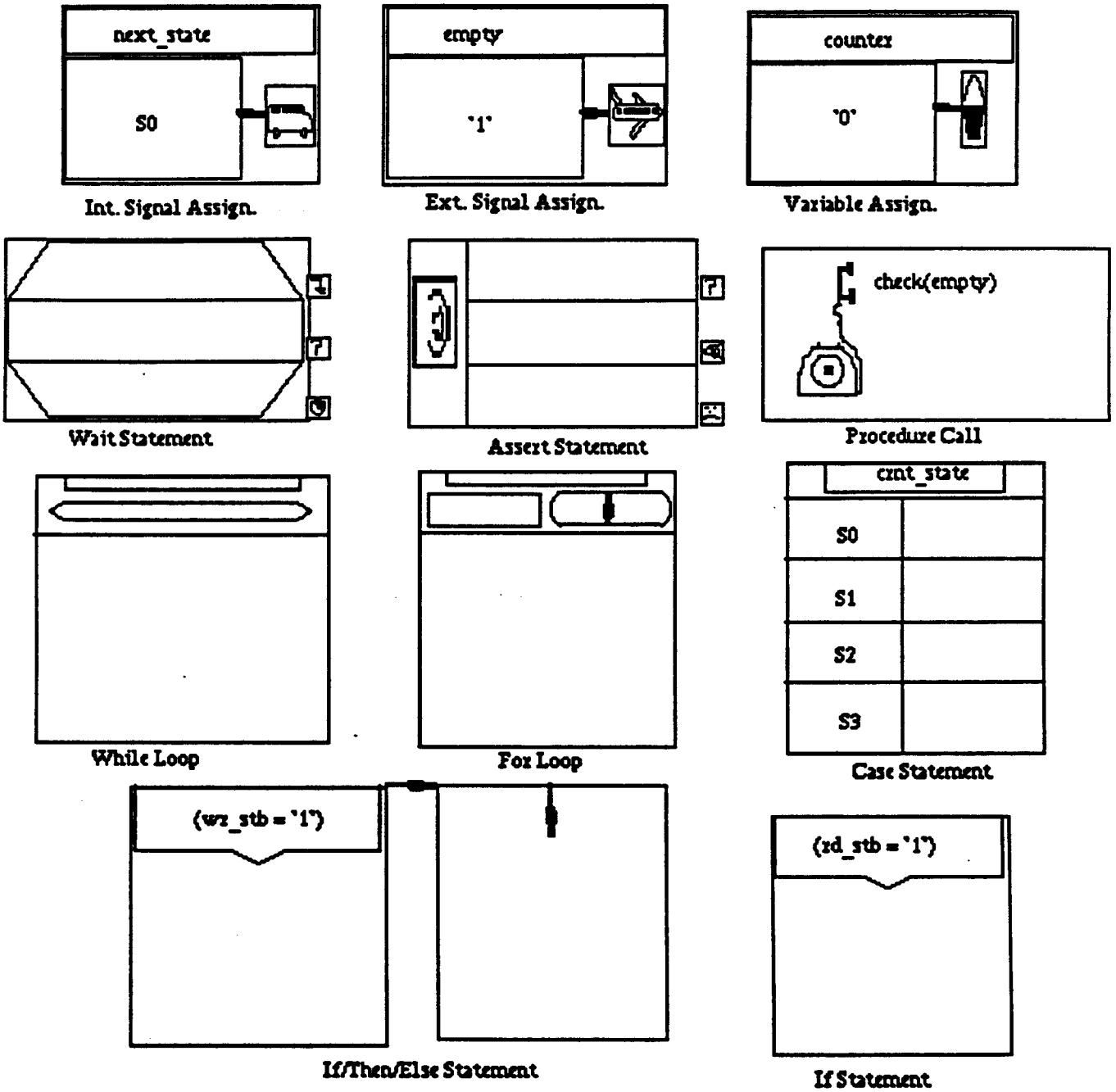


Figure 5: Sequential Statements in vVHDL

Visual programming languages, by nature, require the use of graphical interfaces for manipulating programs.

We have constructed an environment that allows the visual specification of a component design with a picture and that generates textual VHDL code corresponding to the picture. The overall approach to building the environment is based on combining a graphical editor tailored to vVHDL with a compiler for the vVHDL language. The architecture of the system is shown in Figure 6.

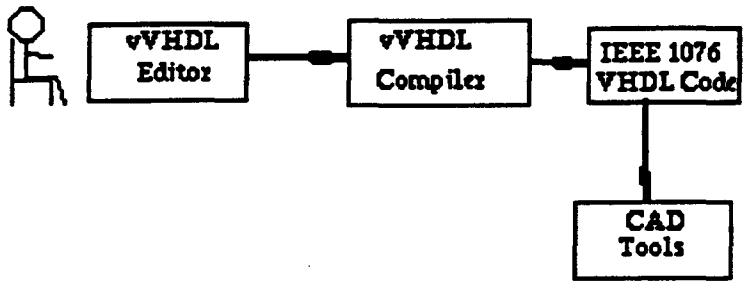


Figure 6: The vVHDL Design System

The vVHDL editor is constructed using the Palette system [11], which is a *reconfigurable* editor for visual programs. It provides the underlying capabilities of a graphical editor, including the overall editor framework, automatic display facilities and simple editing functions. Palette also defines a framework for customizing an editor to a specific visual language. The editor for vVHDL, which is referenced in Section 4.1, was constructed by defining the graphical shapes that are found in the vVHDL language.

In addition to the editor, a compiler for the vVHDL pictures was constructed. The vVHDL compiler is responsible for analyzing the picture to determine the syntactic structure (i.e., parsing the picture) and then translating the picture into IEEE 1076 VHDL code. This code can then be used by most commercially available CAD tools.

4 Designing with vVHDL - a FIFO

The use of vVHDL for electronic design is best demonstrated by a realistic example. In this section, an original design of a *cascadable FIFO*[12] is presented. The vVHDL tool has been used to create a VHDL model of a large portion of the design. This example illustrates some of the concepts of the visual design environment, and shows its advantages over conventional electronic CAD environments.

4.1 The Example Design

A FIFO allows systems with different clock rates to be interconnected. In large, high speed data acquisition systems the FIFO is used frequently, and it is essential that it be fast as well as reliable.

This cascadable FIFO was designed to be fully testable, easily reconfigurable in real time and highly fault tolerant.

A block diagram of the cascadable FIFO is shown in Figure 7. The figure includes components for the `empty/full logic` and the `read/write logic` finite state machines, the tristate output `RAM`, and the `write` and `read counters`. The diagram shows the connectivity of the components, but does not describe the nature of the signals transmitted between components. For example, the `data_in` and `data_out` signals of the `RAM` are tristate, while all other signals are two state signals. In addition, it is not clear that the `write` and `read counters` are identical components.

A mixed-level model of the FIFO has been constructed using vVHDL. Figure 8 shows the vVHDL `fifo` entity declaration which describes the interface of the design to other elements. The nature and the direction of the interfacing signals are clearly shown by this picture. This figure also shows the vVHDL editor. Figure 9 shows the vVHDL representation of the architecture body corresponding to the FIFO. This architecture includes several component declarations and instantiations (eflogic, counter, RAM16x4, and rwlogic) along with their incoming and outgoing signals. In vVHDL, signals are bound to ports by their position on the component in the following way: the signals in the component declaration are bound to the signals the component instantiation in order from top-left to bottom-left to top-right to bottom-right. While this is somewhat limiting, it greatly simplified the prototype design. In future versions of vVHDL, components will be more flexible. In addition to components, concurrent procedure calls (wc_and, rc_and), concurrent assignments to internal signals (compar, wrstr_and, rdstr_and), and concurrent assignments to external signals (Xout_or), are shown in this example.

One design requirement of the FIFO is the capability to indicate to the external interface when it is full or empty. A state machine is used to implement this function. The state transition diagram for this machine is shown in Figure 10. The outputs are *empty* and *full*, and the inputs are *wr_stb*, *rd_stb*, and *equal*. The equal input is true when the outputs of the read and write counters are equal. The state transitions occur at the falling edges of wr_stb and rd_stb. The vVHDL model for the state machine entity is shown in in Figure 11. The vVHDL case statement representing the state machine is shown in Figure 12. It is important to note that in all of the vVHDL pictures the designer is freed from having to worry about syntax details such as semicolons and begin/end statements.

4.2 Compiling to VHDL

Once a vVHDL picture has been drawn, it can be translated into VHDL code. The visual syntax of the vVHDL language and its translation to VHDL code are defined by an object-oriented picture layout grammar. C++ classes and functions are defined for each of the vVHDL primitives. These are used in the grammar to specify how the picture should be parsed. The grammar is processed by a spatial parser generator to produce the vVHDL compiler. Finally, the vVHDL compiler is run on the picture. The compiler produces a parse tree from which textual VHDL code is generated. This code is equivalent to conventional VHDL code and serves all of the same purposes.

The state machine is modeled by a `case` statement, with branches for each value of the current state. In each branch, the possible transitions are modeled by `if-then-else` statements which check the value of some inputs and assign the correct next state. Each branch also contains assignments to the outputs empty and full. The vVHDL model is a clear representation of the states of the design, the possible transitions, and the outputs.

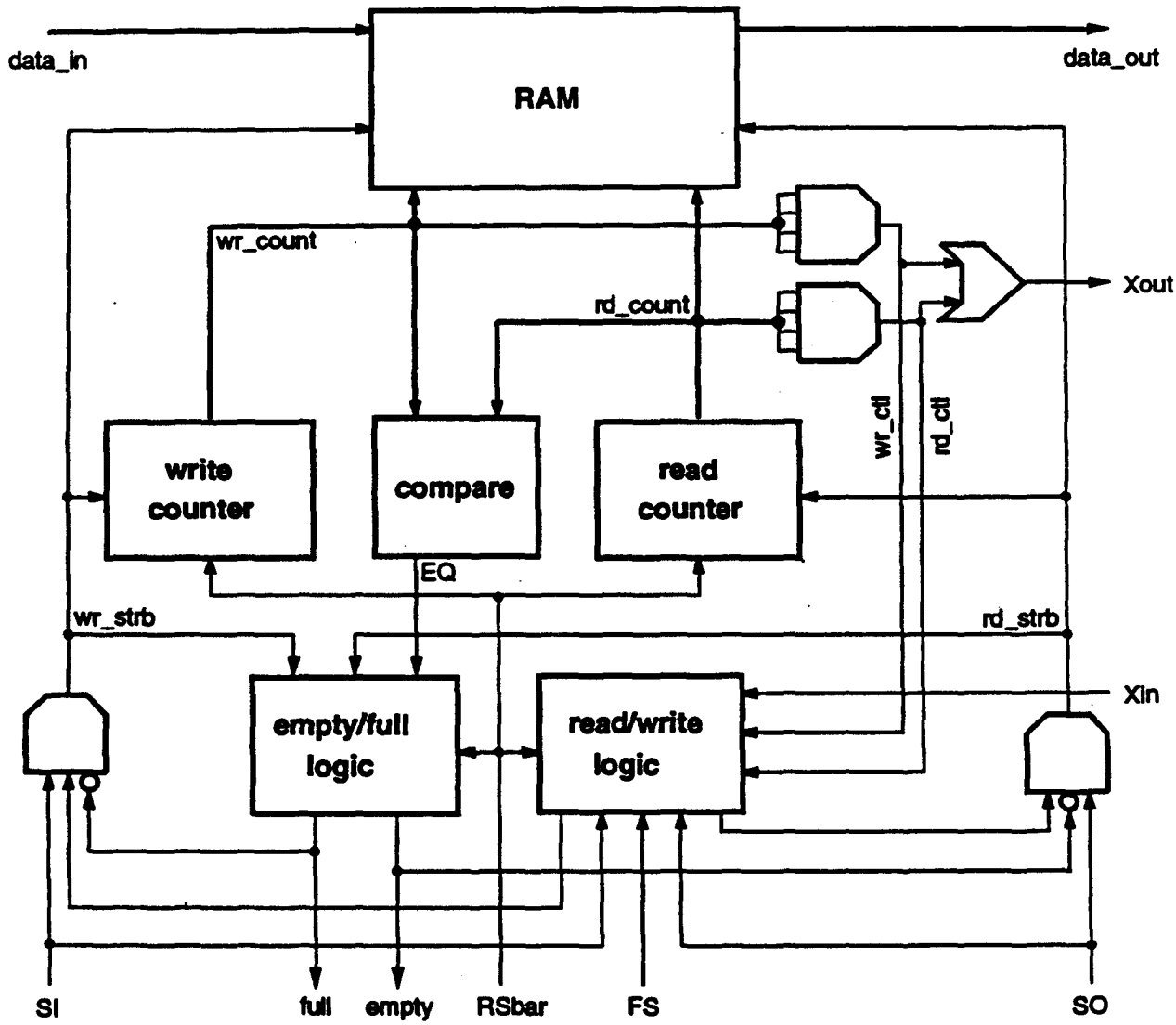


Figure 7: Block Diagram of the Cascadable FIFO

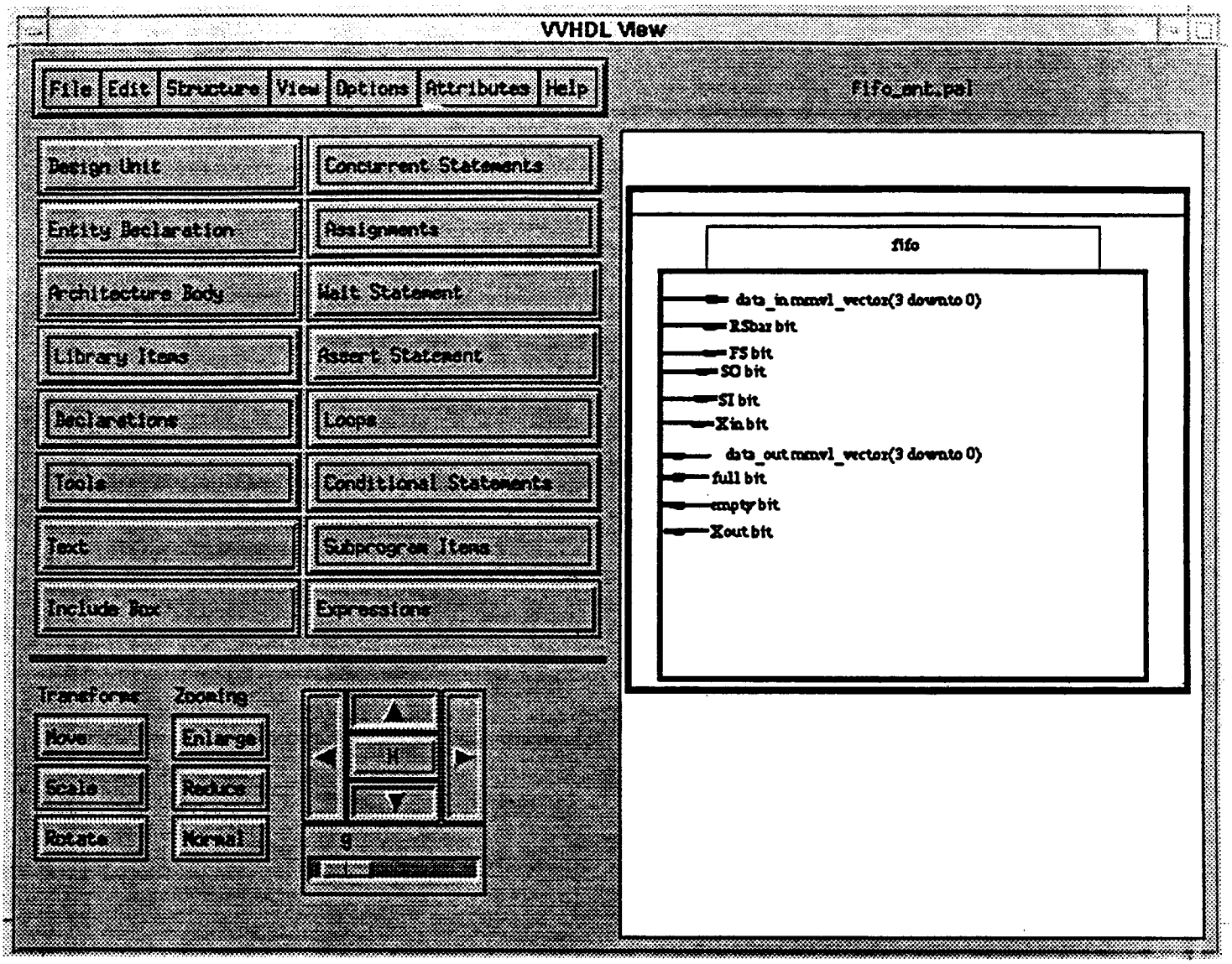


Figure 8: vVHDL Model of the FIFO Entity

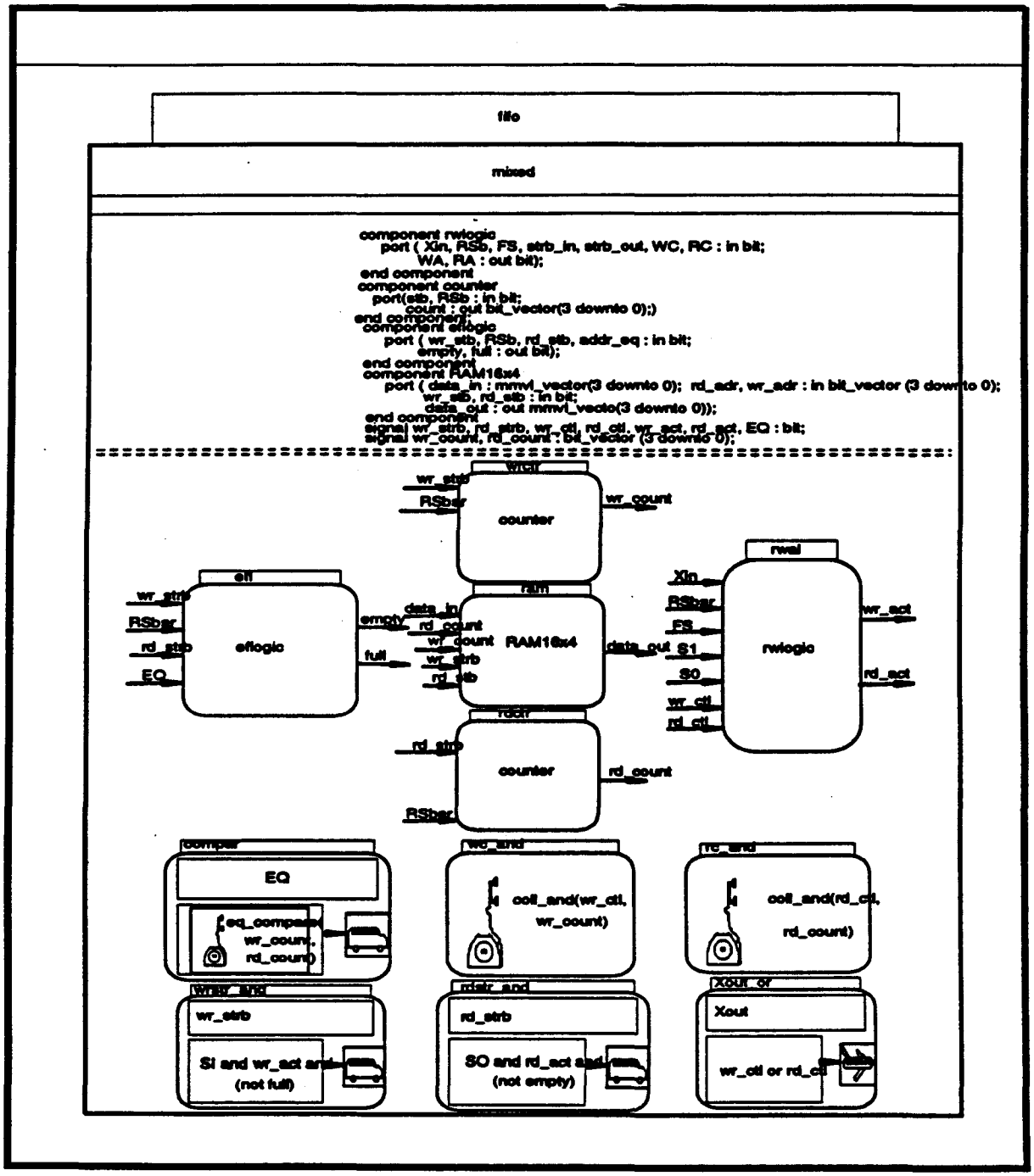
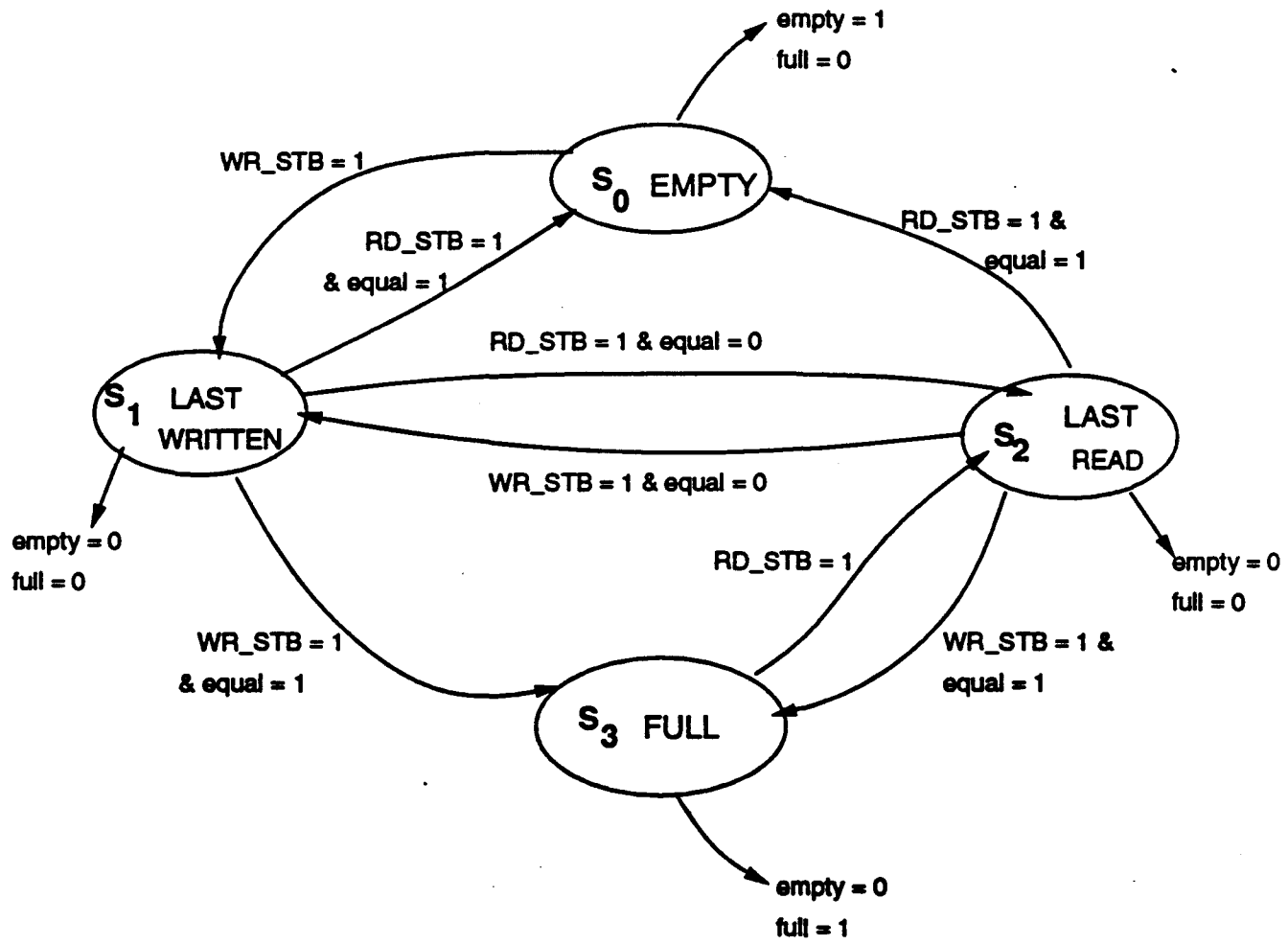


Figure 9: vVHDL Model of the FIFO Architecture



STATE DIAGRAM OF THE FULL / EMPTY LOGIC

Figure 10: State Diagram of the eflogic

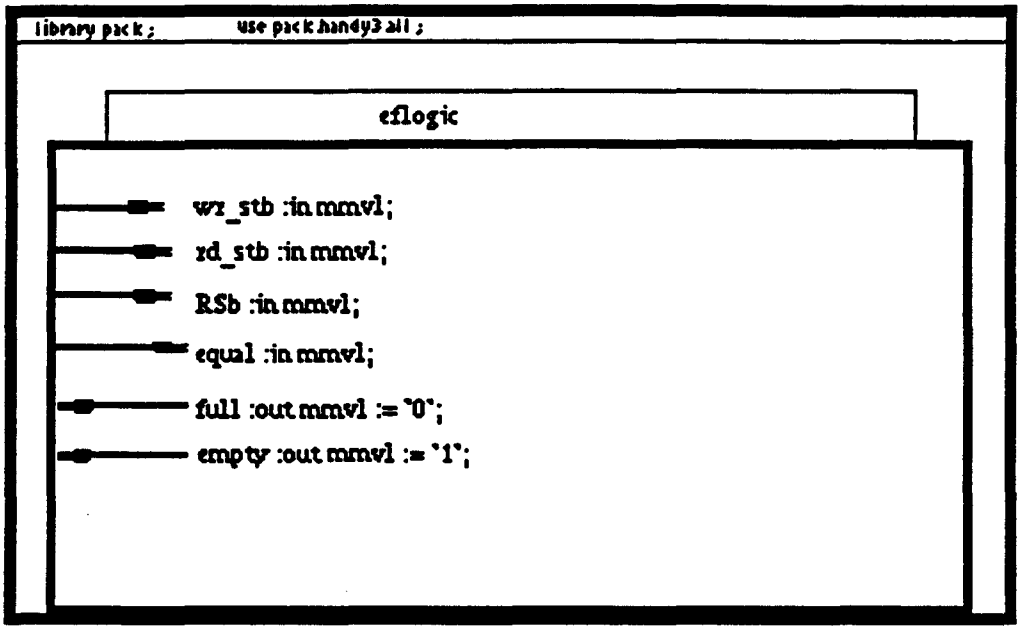


Figure 11: vVHDL Model of the eflogic Entity

4.3 Simulation

The state machine implementing the empty/full logic was designed and compiled to VHDL using vVHDL as described in the previous section. This code can be found in Appendix A. The VHDL model was then simulated using the Vantage Spreadsheet software[13]. The test vectors were constructed to show the various state transitions. As shown in Figure 10, not all transitions occur at the same clock edge. For example, the transition from S0 to S1 occurs at the negative edge of wr_stb, while the transition from S3 to S2 occurs at the negative edge of rd_stb. Thus, if a wr_stb occurs during state S3 then no transition is made. One way to implement this is to treat the clock signals as inputs to the next state logic of the state machine.

The resulting waveforms are shown in Figure 13. *Rst* is the external reset signal conncected to the RSb port which initializes the state to S0. The results verify the state transitions shown in Figure 10. The results also show that the transitions occur at the correct clock edges. In this example, we have shown that the vVHDL tool can be used to construct a VHDL model of a realistic design. The VHDL code generated from the vVHDL model can be compiled and simulated using existing CAD tools.

5 Discussion

A visual programming language for VHDL has been developed that significantly reduces the syntactic burden of textual VHDL, retains the inherent multi-dimensional aspects of hardware

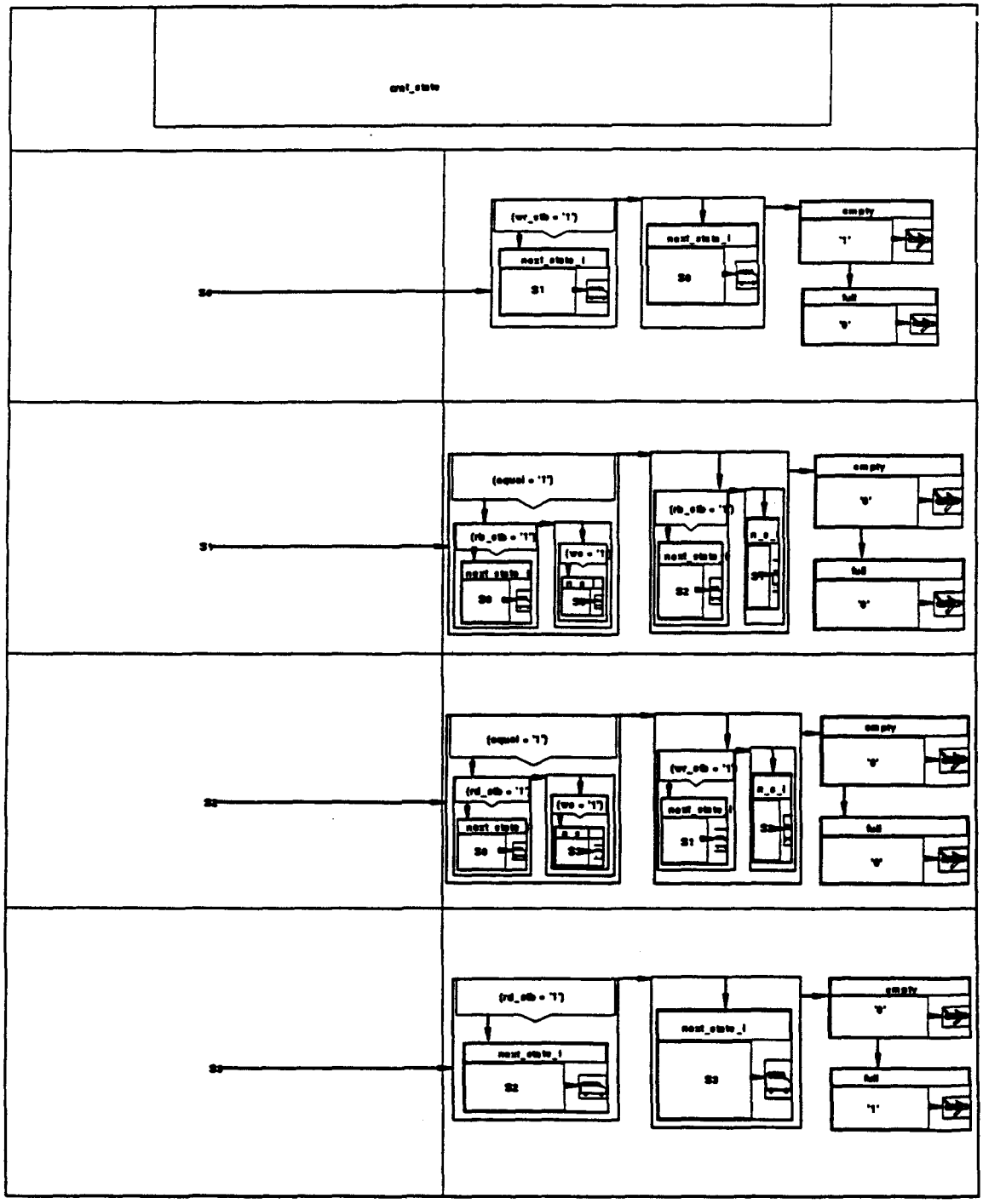


Figure 12: vVHDL Model of the case statement in the eflogic Architecture

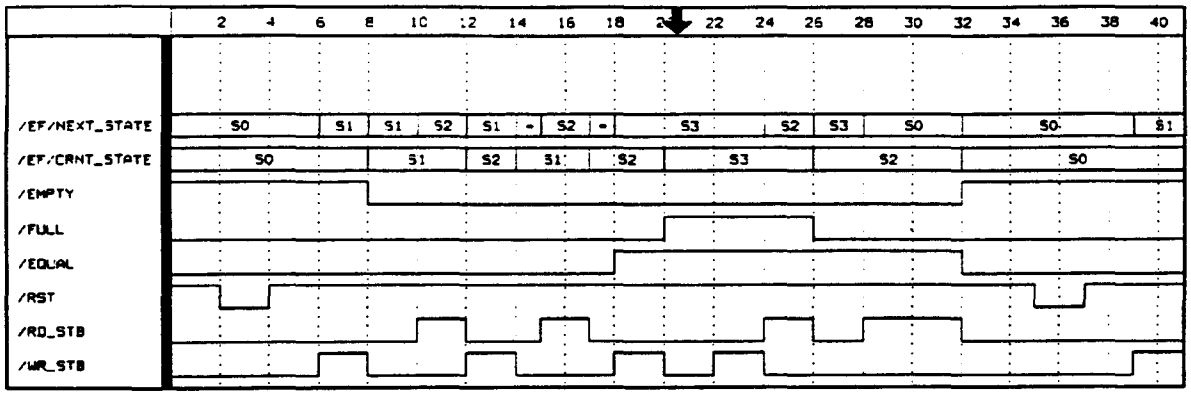


Figure 13: Simulation Output for the eflogic

description, and allows for mixed-level designs. In addition, it provides a clear, overall picture of how various components in a design are interconnected. This is especially useful in designs with a large number of strongly interconnected components. It helps in understanding the functionality of the design and makes it easier to debug and/or modify existing designs. An example was shown of the use of vVHDL to design and simulate a low level component of a large data acquisition system. We intend to use vVHDL to design the entire system and to communicate and share models with the other members of the SDC collaboration.

The most significant difficulty with the current vVHDL prototype is handling large designs. Large designs are problematic because their visual representations are hard to view and manipulate on normal computer screens, and because they are difficult to print on standard printers. The viewing problem will be resolved by allowing modular design and by hiding the details in an existing picture. The printing problem will be addressed by allowing multi-page printing.

The current version of vVHDL is only a prototype. In the future, we plan to extend vVHDL to cover the complete VHDL language, rather than just a subset. In addition, the vVHDL prototype is closely tied to the textual representation of VHDL, and while this simplified the initial implementation, future versions should rely more heavily on visual representations. For example, visual declarations, which are not currently supported, should be implemented. In order to make vVHDL look more like a schematic editor, future versions could allow the use of standard engineering symbols for structural descriptions. Finally, the modifications discussed concerning components should be implemented.

6 Acknowledgements

The authors wish to thank Robert Downing, Larry Jones, Dave Knapp, Jon Thaler and the other members of the SSC DAQE Design Environments project for their contributions to the design of vVHDL. Steven Danz and Susan Larison were instrumental in implementing the Palette environment. This research is supported by the Texas National Research Laboratory Commission, project #047, and by National Science Foundation grant CCR-9108931.

7 Appendix A: VHDL Code for eflogic

```
library pack;
use pack.handy3.all;
entity eflogic is
   port(wr_stb : in mmvl;
        rd_stb : in mmvl;
        RSb : in mmvl;
        equal : in mmvl;
        full : out mmvl := '0';
        empty : out mmvl := '1');
end eflogic ;

architecture efmoore of eflogic is

   type ef_state is (S0, S1, S2, S3);
   signal crnt_state : ef_state := S0;
   signal next_state_I : ef_state;

begin

   ef: process(crnt_state,equal,RSb,wr_stb,rd_stb)
   begin
      if (RSb = '0') then
         next_state_I <= S0;
      else
         case crnt_state is
         when S0 =>
            if (wr_stb = '1') then
               next_state_I <= S1;
            else
               next_state_I <= S0;
            end if;
            empty <= '1';
            full <= '0';
         when S1 =>
            if (equal = '1') then
               if (rd_stb = '1') then
                  next_state_I <= S0;
               else
                  if (wr_stb = '1') then
                     next_state_I <= S3;
                  end if;
               end if;
            else
               if (rd_stb = '1') then
                  next_state_I <= S2;
               else
                  next_state_I <= S1;
               end if;
```

```
            end if;
            empty <= '0';
            full <= '0';
         when S2 =>
            if (equal = '1') then
               if (rd_stb = '1') then
                  next_state_I <= S0;
               else
                  if (wr_stb = '1') then
                     next_state_I <= S3;
                  end if;
               end if;
            else
               if (wr_stb = '1') then
                  next_state_I <= S1;
               else
                  next_state_I <= S2;
               end if;
            end if;
            empty <= '0';
            full <= '0';
         when S3 =>
            if (rd_stb = '1') then
               next_state_I <= S2;
            else
               next_state_I <= S3;
            end if;
            empty <= '0';
            full <= '1';
         end case;
      end if;
   end process ef;

   feedback: process(wr_stb,rd_stb,RSb)
   begin
      if (RSB = '0') then
         crnt_state <= S0;
      else
         if ((not wr_stb'stable) and (wr_stb = '0')) then
            crnt_state <= next_state_I;
         end if;
         if ((not rd_stb'stable)  and (rd_stb = '0')) then
            crnt_state <= next_state_I;
         end if;
      end if;
   end process feedback;
end efmoore;
```

References

[1] *VHDL Language Reference Manual (IEEE Standard 1076-1987)*. New York, March 1988.

[2] E. Hughes, G. Tharakan, R. Downing, M. Haney, E. Golin, L. Jones, D. Knapp, and J. Thaler. Modeling and simulation of the SDC data collection chip. *IEEE Transactions on Nuclear Science*, 39(2), April 1992.

[3] Larry F. Saunders. The IBM VHDL Design System. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 484–490, 1987.

[4] Paul R. Jordan and Ronald D. Williams. VCOMP: A VHDL Composition System. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 750–753, 1989.

[5] Joe DeGroat, Kevin Berk, Douglas Pompilio, and Stephen Matechik. The AFIT VHDL Environment. In *1988 Frontiers in Education Conference Proceedings*, pages 324–329, 1988.

[6] P. T. Cox, F. R. Giles, and T. Pietrzykowski. Prograph: a step towards liberating programming from textual conditioning. In *1989 IEEE Workshop on Visual Languages*, pages 150–156, Rome, Italy, October 1989.

[7] Kenneth M. Kahn and Vijay A. Saraswat. Complete visualizations of concurrent programs and their executions. In *1990 IEEE Workshop on Visual Languages*, pages 7–15, October 1990.

[8] Carla S. Williams and John R. Rasure. A visual language for image processing. In *1990 IEEE Workshop on Visual Languages*, pages 86–91, Skokie, IL, October 1990.

[9] Gregory Scott Rogers. *Visual Programming Using Graphics, Relations, and Classes.* PhD thesis, University of Illinois at Urbana-Champaign, 1990. (available as Report No. UIUCDCS-R-90-1632).

[10] M. Fitter and T. R. G. Green. When do diagrams make good computer languages? In *International Journal of Man-Machine Studies*, pages 236–261, 1979.

[11] Eric J. Golin, Steven Danz, Susan Larison, and Diana Miller-Karlow. Palette: An Extensible Visual Editor. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 1208–1216, 1992.

[12] G. Tharakan. A Fast, Reliable, Cascadable FIFO for the SDC. *submitted to IEEE Transactions on Nuclear Science*, April 1992.

[13] Vantage Analysis Systems. *Vantage Spreadsheet User Guide*, May 1991.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-92-1745	2.	3. Recipient's Accession No.
4. Title and Subtitle VISUAL DESIGN WITH vVHDL			5. Report Date April 1992
7. Author(s) Eric J. Golin, Michael J. Haney, Eric Hughes, Diana Miller-Karlow, and George Tharakan			8. Performing Organization Rept. No. R-92-1745
9. Performing Organization Name and Address Department of Computer Science University of Illinois 1304 W. Springfield Avenue Urbana, IL 61801			10. Project/Task/Work/Unit No. 11. Contract/Grant No. See front page.
12. Sponsoring Organization Name and Address Texas National Research Laboratory Commission National Science Foundation, Washington DC			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

A prototype visual design environment for electronic computer-aided design is presented.
The design environment, vVHDL, is based on a visual implementation VHDL.
Unlike schematic capture tools, vVHDL allows the designer to draw pictures of arbitrary VHDL code.
A realistic example design is presented to demonstrate the capabilities of the tool.
The visual model captures the spatial aspects of the design, which are difficult to extract from the corresponding textual model.
A discussion of the prototype tool is included and future directions are described.

17. Key Words and Document Analysis

Computer-Aided Electronic Design
Visual Programming Languages
CAD Tools

17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement unlimited	19. Security Class (This Report) Unclassified	21. No. of Pages 19
	20. Security Class (This Page) Unclassified	22. Price

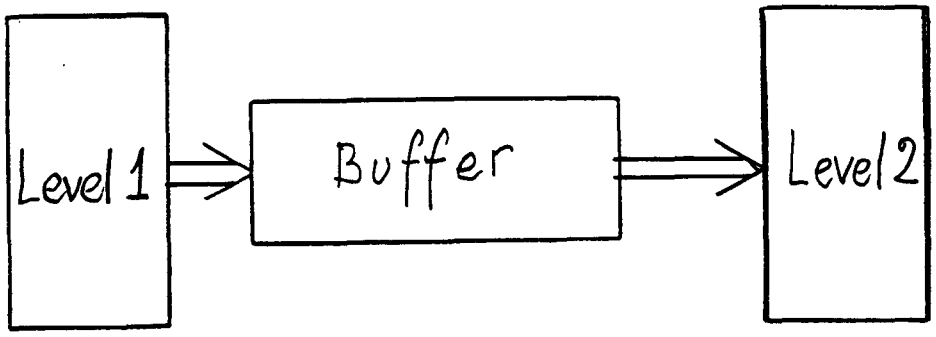
FORM NTIS-35 (10-70) USCOMM-DC 40329-P71

pkr 4/11/90

Stochastic Simulation of Asynchronous Buffers

Henry Kasha, *Yale University*

Let us consider the path of trigger data flow from a collider detector through the output of the level 2 trigger processor, assuming for now that there is no level 0 trigger processor.



There are then four characteristic times (or frequencies) which determine the data flow

- The bunch crossing interval
- Level 1 processor decision time (constant)
- The mean interval between the events passed by level 1
- Level 2 decision time (for now assumed to be constant)

The bunch crossing time for the SSC is 16 ns, and for the upgraded (1995) Tevatron might be as short as 132 ns. Level 1 and 2 decision times are typically expected to be of the order of 1 and 10 μs respectively.

Level 1 processor (in the absence of a level 0) has to contend with a queue of the order of 10 (FNAL) or 100 (SSC) events waiting to be processed. A pipelined synchronous processor clocked by the bunch crossing frequency of sufficient depth is then needed.

The level 1 rejection ratio must be such that that the mean interval between two events passed by it be at most equal to the level 2 decision time, if we want to avoid building another pipelined processor. However, even if such a rejection factor is achieved, a buffer is still needed to avoid loss due to the statistical distribution of the inter-event times around the mean.

The present simulation is devoted to the study of such a stochastic buffer.

Modeling the system

MODSIM II is used, which keeps track of the various events and processes in "simulation time".

There are two independent clocks:

- Bunch crossings: every 132 ns
- Level 2 checks the buffer for events every 50 ns (unless busy processing an earlier event)

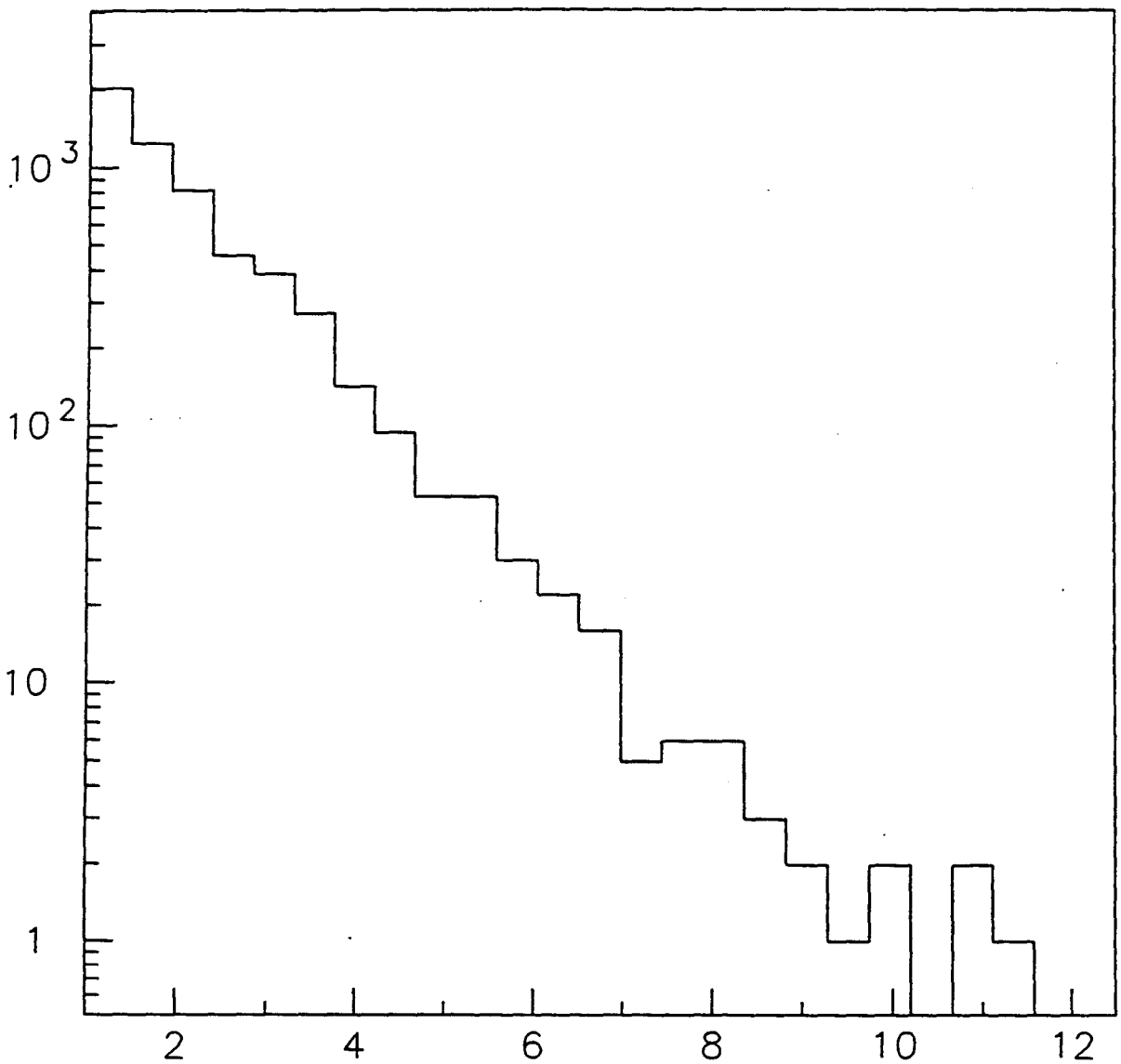
The level 1 and level 2 decision times were set to 1.2 μs and 10 μs, respectively.

The level 1 rejection ratio and the buffer depth are adjustable input parameter. K is the ratio of the level 2 decision time to the mean inter-event interval passed by level1. Histograms 1 and 2 show the inter-event interval distribution for $K = 1$ and $K = 2.2$.

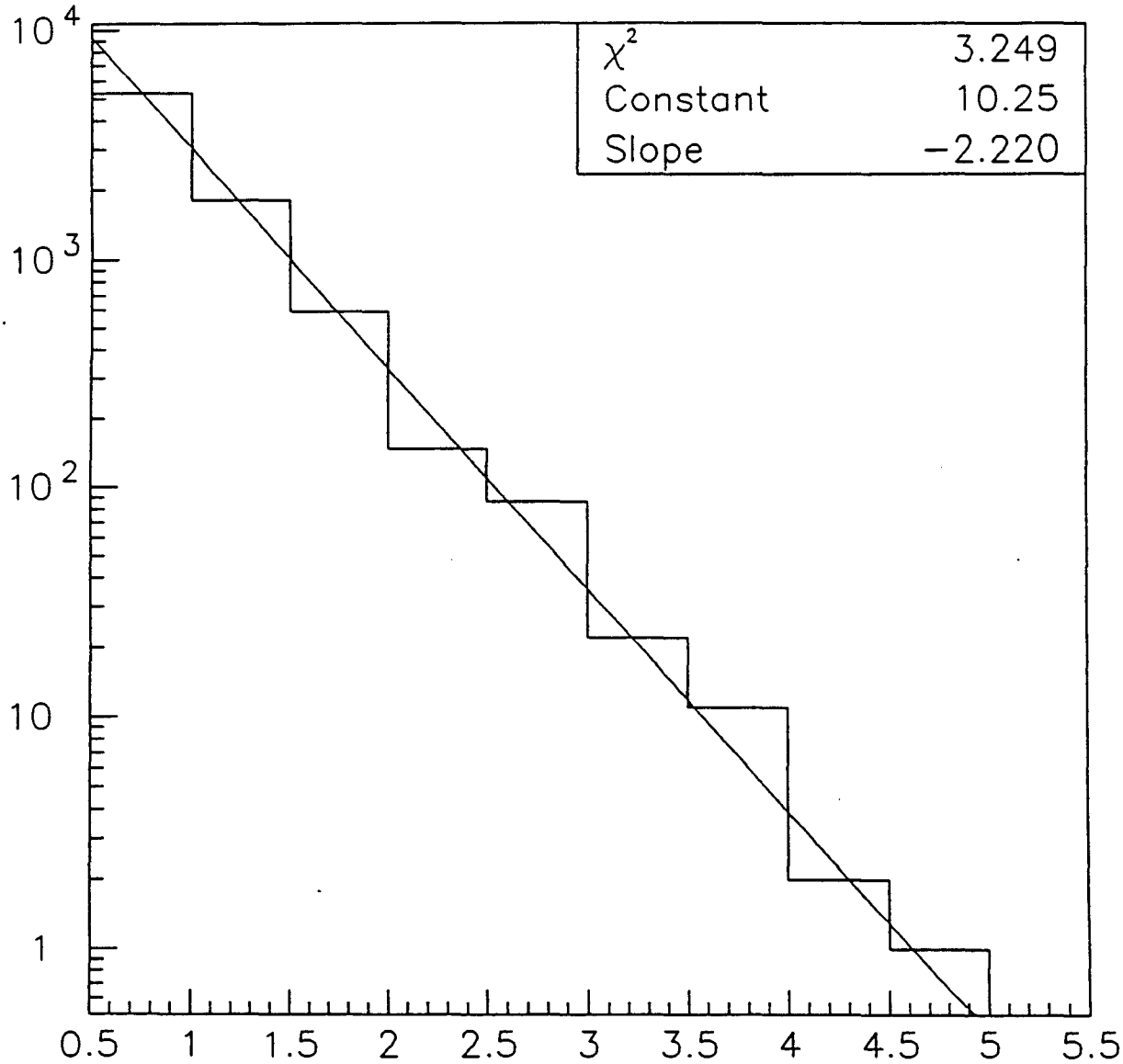
We will see that the parameter K, which is a function of the level 1 rejection ratio, determines the dead time and the buffer occupancy.

Histograms 3 and 4 represent the occupancy of a 32 event deep buffer for situations with $K = 1$ and $K = 1.1$.

The next transparency represents the dead time of the system as a function of the buffer depth and the parameter K.

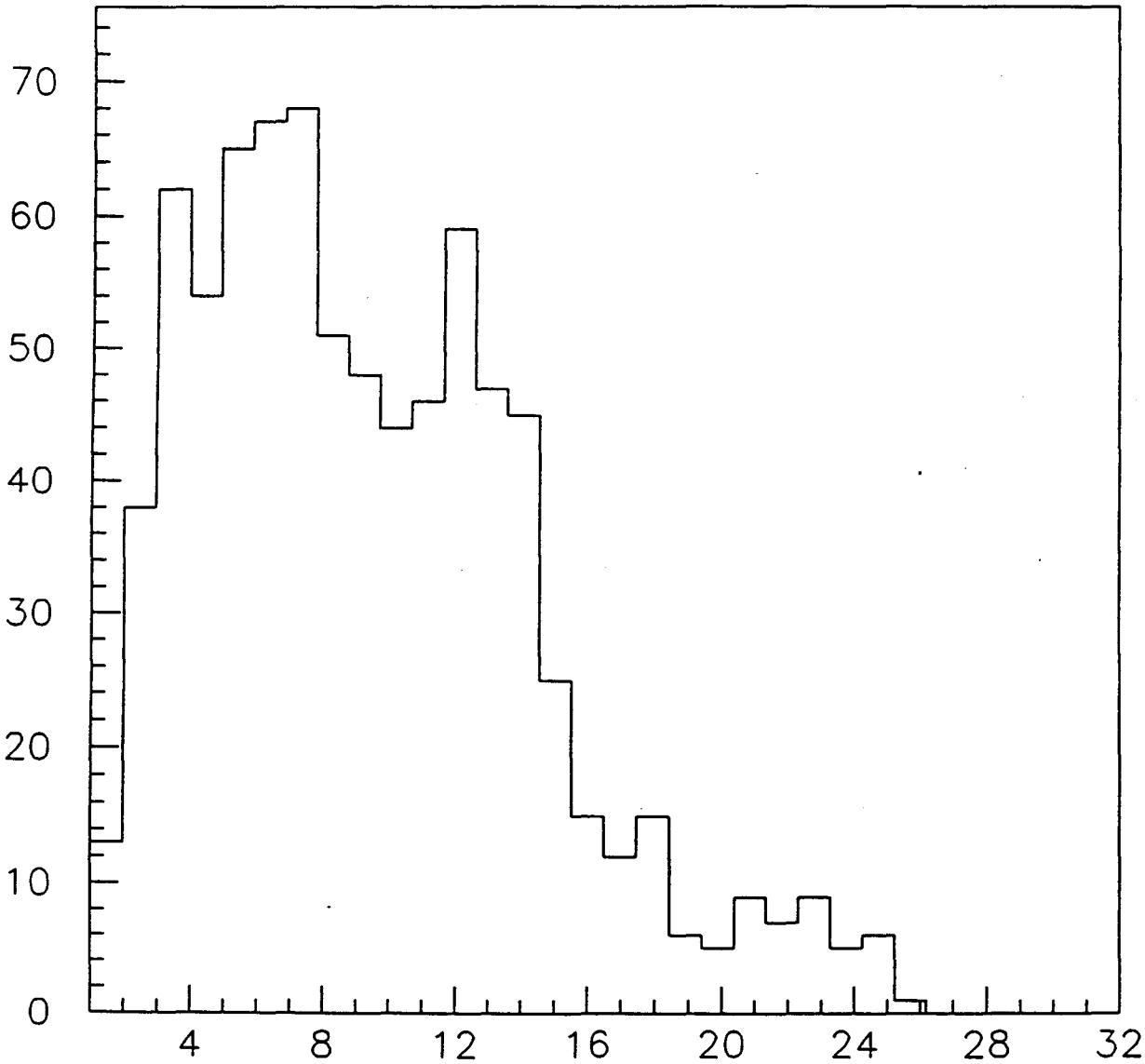


Interval between events, K=1.0 H/



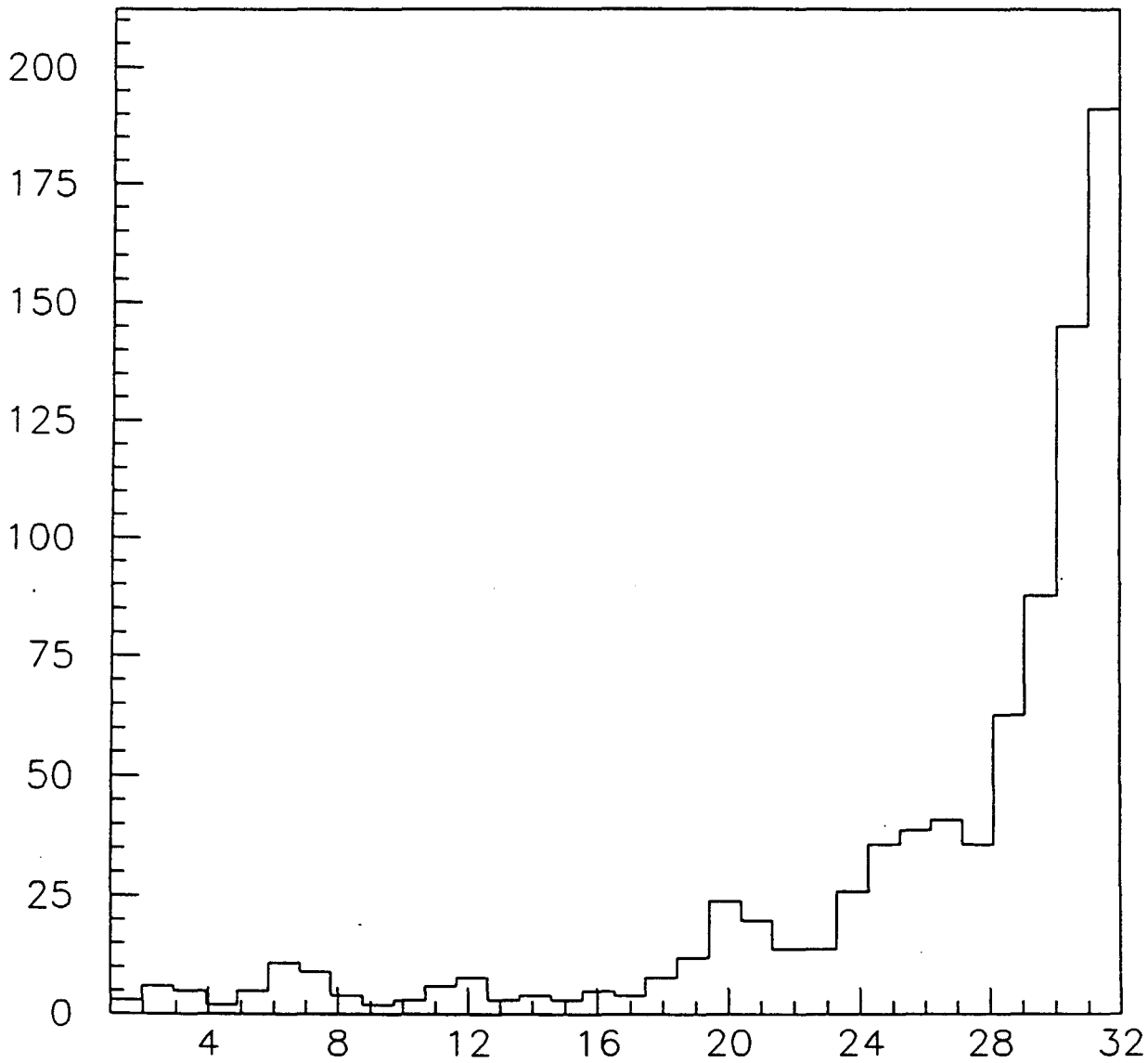
Interval between events, K=2.2

H2



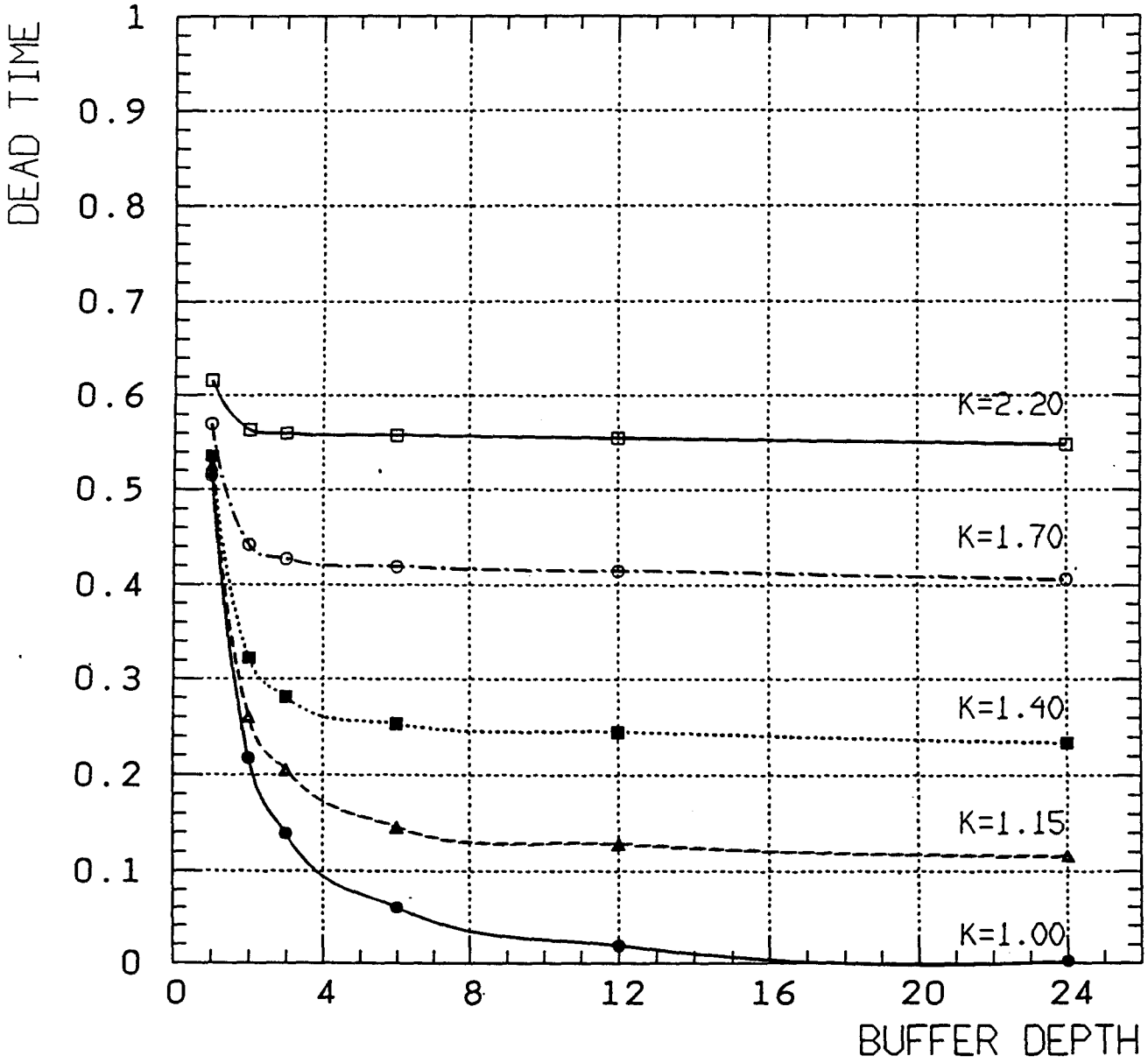
Buffer occupancy, K=1.0

H3



Buffer occupancy, K=1.1

H4


DEAD TIME
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
4
8
12
16
20
24
BUFFER DEPTH
K=2.20
K=1.70
K=1.40
K=1.15
K=1.00

Conclusions

- Trigger processors must not be oversaturated
- Relatively shallow buffers can be very helpful in preventing or reducing the dead time due to statistical fluctuations in the the event interarrival interval at the output of level1 (or level 0) trigger processor.
- MODSIM can be quite useful for behavioral and stochastic simulations

SDC Trigger Simulations

W. Temple, W. Smith, J. Lackey,
T. Gorskii, S. Dasu

University of Wisconsin-Madison

Topics:

- o SDC Level 1 Trigger concepts
- o Jet Trigger Studies
- o Electron Trigger Studies

SDC TRIGGER SYSTEM

Requirements:

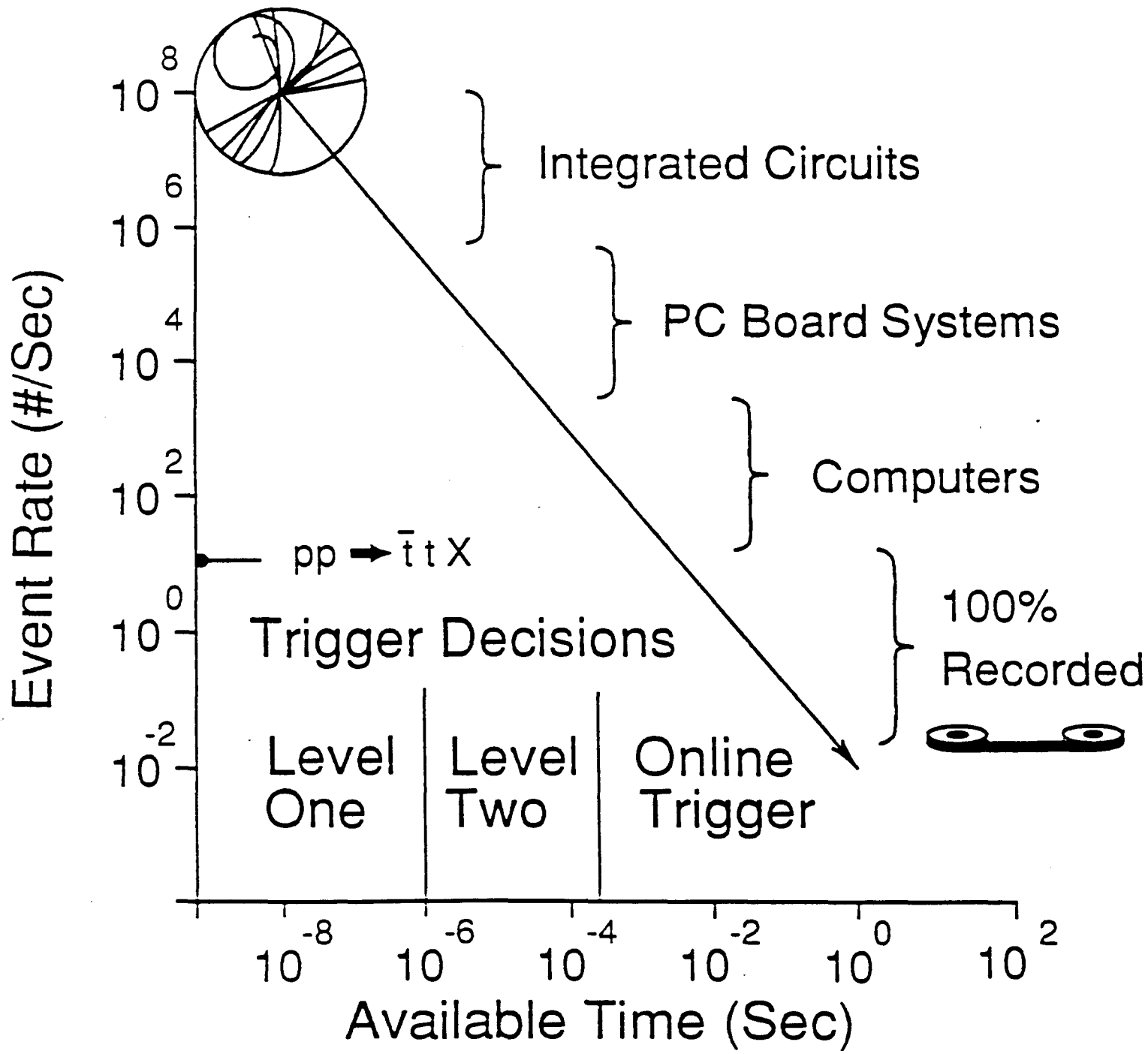
Input Rate: 10^8 Interactions per second (average 1.6 interactions per 16 nsec bunch crossing)

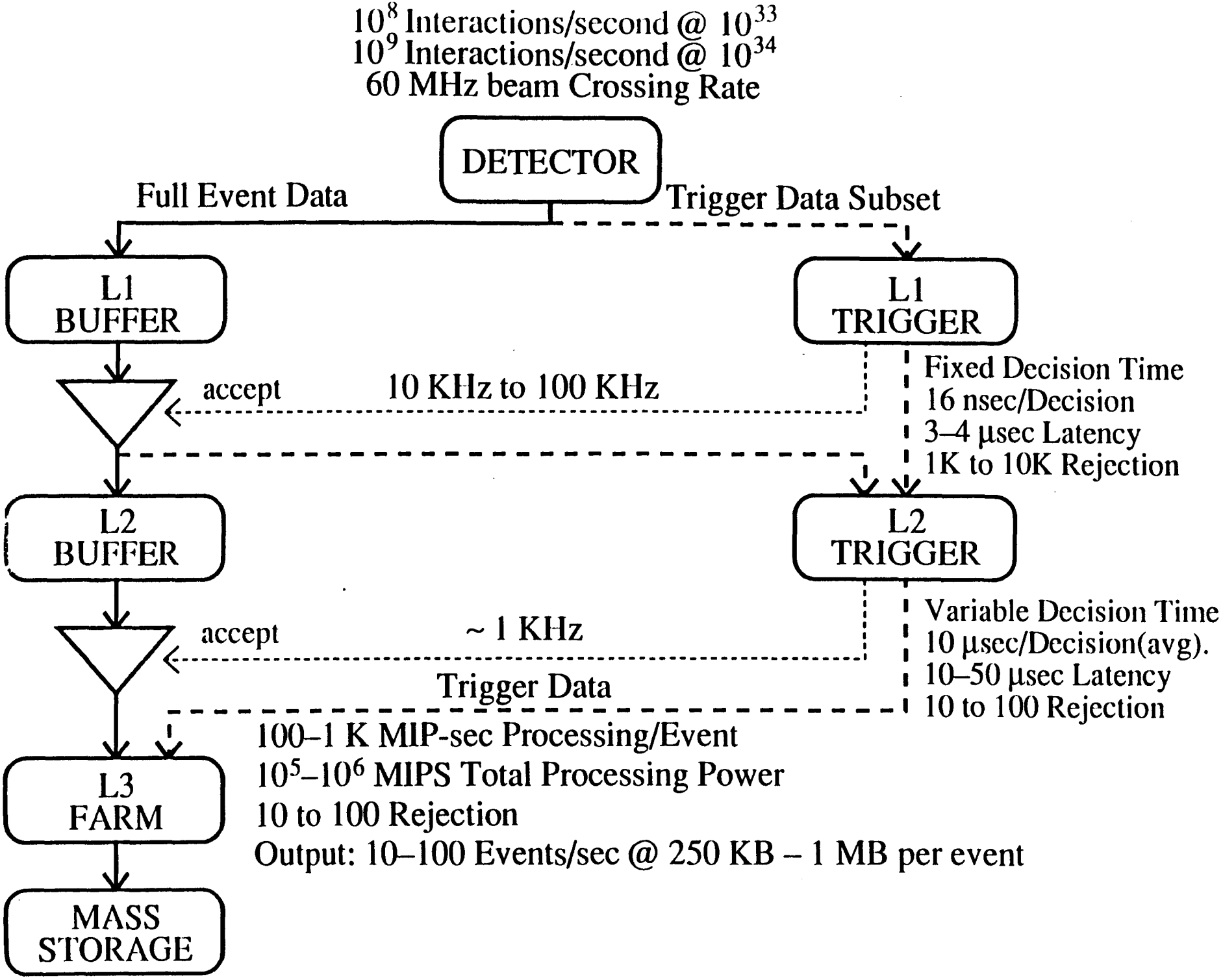
Output Rate: 50 - 100 Hz can be written to tape

Physics Rate: 50 Hz of Top, W's, Z's *(Little Spare Bandwidth)*

☞ 10^6 Rejection needed

SSC TRIGGERING: EVENT RATES & TECHNOLOGIES




10^8 Interactions/second @ 10^{33}
10^9 Interactions/second @ 10^{34}
60 MHz beam Crossing Rate
DETECTOR
Full Event Data
Trigger Data Subset
L1 BUFFER
L1 TRIGGER
accept
10 KHz to 100 KHz
Fixed Decision Time
16 nsec/Decision
3–4 μsec Latency
1K to 10K Rejection
L2 BUFFER
L2 TRIGGER
accept
~ 1 KHz
Variable Decision Time
10 μsec/Decision(avg).
10–50 μsec Latency
10 to 100 Rejection
Trigger Data
100–1 K MIP-sec Processing/Event
10^5–10^6 MIPS Total Processing Power
10 to 100 Rejection
Output: 10–100 Events/sec @ 250 KB – 1 MB per event
L3 FARM
MASS STORAGE

SDC TRIGGER PRINCIPLES

Philosophy:

Local:

Signatures of e's, γ's, μ's, jets

in $\eta \times \phi$ regions

(ex: ν's)

Measurably Efficient:

Overlapping Programmable Triggers

Prescaled Lower Thresholds

Prescaled Triggers w/ less conditions

Efficient Use of DAQ Bandwidth:

Efficient Lepton and Jet Identification

Consistency with Offline Cuts

Benchmarks for Trigger Performance:

e's, μ's from inclusive W's, Z's:

50% efficiency (cut on lepton P_T)

Jets, γ's at high P_T:

1-2 decade overlap with lower $\sqrt{s}$ data

Missing E_T:

Good efficiency for channels such as:

$H \rightarrow 2\ell 2\nu$

SUSY particles

Low P_T multileptons:

B Physics

SDC TRIGGER LEVELS

Level 1:

Identify Physics Objects:

- Electrons
- Photons
- Muons
- Taus
- Jets
- Neutrinos
- *Combinations of Above*

Level 2:

Refine Identification of Physics Objects:

- Sharper P_T Cuts
- Electrons From Conversions
- Muons from Decay/Punchthrough
- Refine Energy Sums/Clusters
- Displaced Vertices

Level 3:

Full Physics Analysis/Decisions:

- Specialized Algorithms
- Heirarchy of Decisions
- DST-type cuts on Physics

LEVEL 1

Electrons & Photons:

Find (.1 x .1) Cal Towers with E_{em} > Thr

Require E_{hac}/E_{em} < .04 - .10

Option: Pattern of surrounding quiet towers (isolation)

γ's: Match w/Shower Max in .2η x .2ϕ

e's: Find Outer Track Segments w/P_T > 10 GeV/c

Match Track Segments w/Shower Max in ϕ in 1/64

Assign $\Delta\eta$ = 0.2 from Shower Max to Track

Match Track w/Cal on $\Delta\eta$=0.2, $\Delta\phi$=0.2

Muons:

Muon Tracks from Scint + θ-layers w/P_T > 10 GeV/c

*Option:*Find Central Tracker Sgmts w/P_T > 10 GeV/c

Match w/Muon ϕ-layer Track Sgmts in 1/64

Use Momentum cut on Central Tracker P_T cut

(Use Scint to associate ϕ and θ tracks)

Match Track w/Quiet Cal on $\Delta\eta$=0.2, $\Delta\phi$=0.2

Jets:

1.6 x 1.6 grids of overlapping towers > E_{thr}

Neutrinos:

Sum Missing E_T over .1 x .1 Towers > E_{thr}

Level 1 system segmentation and trigger information.

Subsystem	Total channels	L1 segmentation	L1 data
Barrel Track	$64\phi \times 2\eta$	0.1ϕ	2 p_t bits
Int. Track	$64\phi \times 2$ ends	0.1ϕ	1 p_t bit
Barrel Cal.	$64\phi \times 28\eta$	$0.1\ \phi \times 0.1\eta$	8 bits EM & HAC Energy
Endcap Cal.	$64\phi \times 14\eta \times 2$ ends	$0.1\phi \times 0.1\eta^{\dagger}$	8 bits EM & HAC Energy
Forward Cal.	$8\phi \times 4\eta \times 2$ ends	$0.8\phi \times 0.8\eta$	8 bits EM & HAC Energy
Shower Max.	$64\phi \times 30\eta$	$0.1\phi \times 0.2\eta$	Hit Flags
Barrel Muon	$32\phi \times 10\eta$	$0.2\phi \times 0.2\eta$	2 p_t bits $\times$ 2 tracks
Int. & Fwd. Muon	$32\phi \times 8\eta \times 2$ ends	$0.2\phi \times 0.2\eta$	2 p_t bits $\times$ 2 tracks

† For $|\eta| > 2.2$, the η and ϕ trigger tower dimensions switch to 0.2 and then to 0.4.

LEVEL 1 TRIGGER STUDIES

Rate:

Level 1 Rate Target = 30 KHz

15 KHz each for Cal. and Muon Triggers

For Calorimeter, total rate is sum of:

- single e's
- single γ's
- pairs: ee, eγ, $\gamma\gamma$
- jets
- $\not{E}_T$

☞ Target Rate of 3 KHz for single object triggers

Study Calorimeter Triggers :

Find threshold for e, γ, jet, $\not{E}_T \Rightarrow$ 3 KHz

Examine threshold vs. trigger conditions

Examine efficiency vs. E_T

Event Generation

- ISAJET to generate events

  120000 QCD 2-jet events from 20-200 GeV for e-study

  20000 Drell-Yan W events

  20000 Drell-Yan Z events

  19000 QCD 2-jet events from 20-400 GeV for jet-study - has better distribution of events over energy

  Minimum bias events corresponding to Poisson distribution with mean 1.6 events per crossing; 16 events per crossing for higher luminosity

Detector Simulation

Greg Sullivan, U. Chicago

- Include γ conversions, decays
  No Bremsstrahlung
  Newer version has Brems.

- Simple calorimeter parameterization

- Tracking thresholds only
  Newer version has more realistic tracking efficiencies

- η-coverage $\eta < 3.0$ for calorimeter; and $\eta < 1.6$ for tracking

L1 Trigger simulation

- Complete baseline design electronics simulation

- Includes:

  Threshold cutoffs

  Local calorimeter energy sums

  Isolated electron search

Jet Trigger Study

- What size grid of energy sums is optimal for finding jets?
- Should overlapping grids of jet towers be used or will non-overlapping towers suffice?
- How will increased luminosity of 10^{34} change the performance of trigger?
- Is the type of integer scale used for energy summing affect the results?

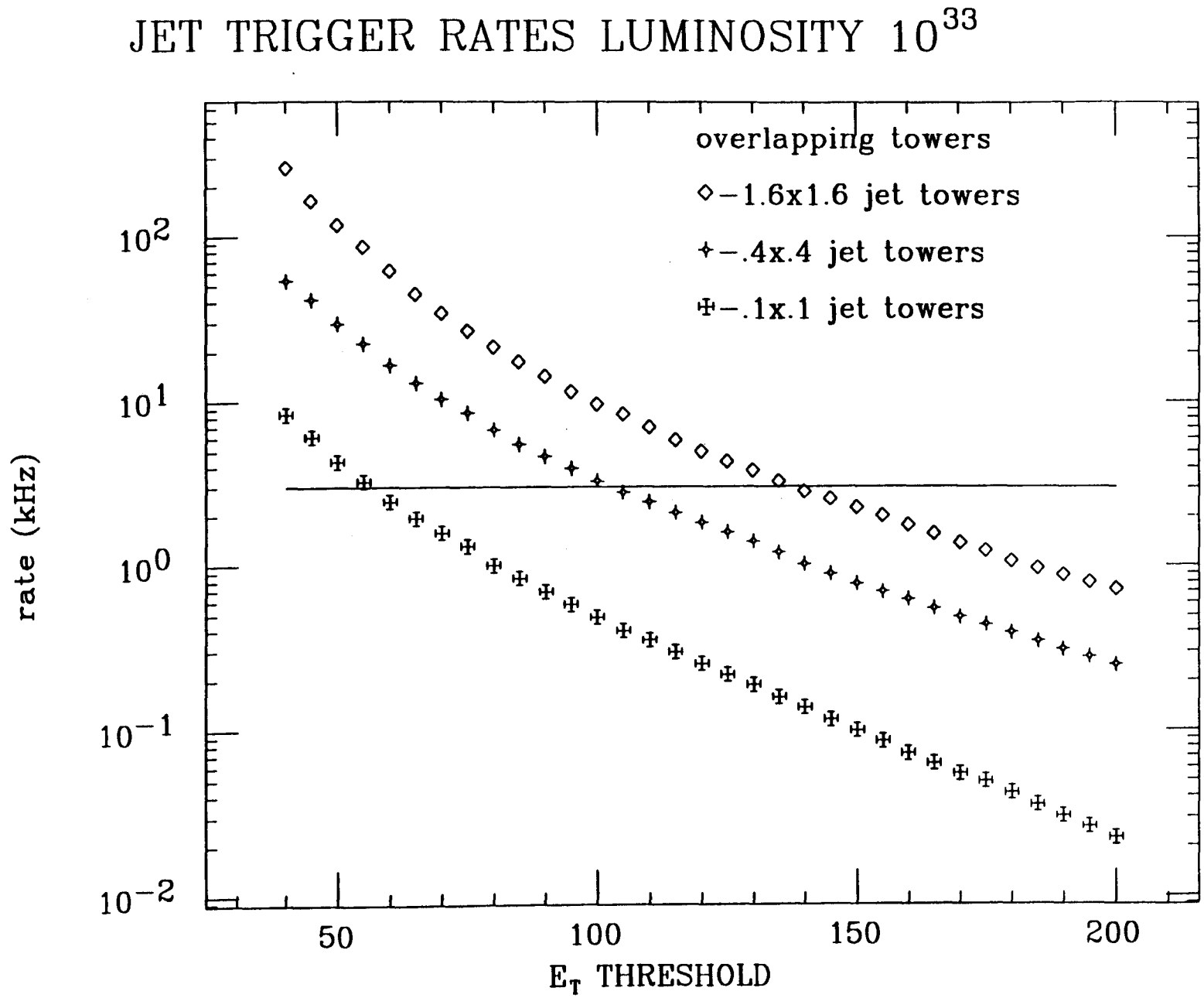

JET TRIGGER RATES LUMINOSITY 10^{33}
overlapping towers
◇–1.6x1.6 jet towers
✧–.4x.4 jet towers
✠–.1x.1 jet towers
rate (kHz)
10^2
10^1
10^0
10^{-1}
10^{-2}
50
100
150
200
E_T THRESHOLD

Figure 1

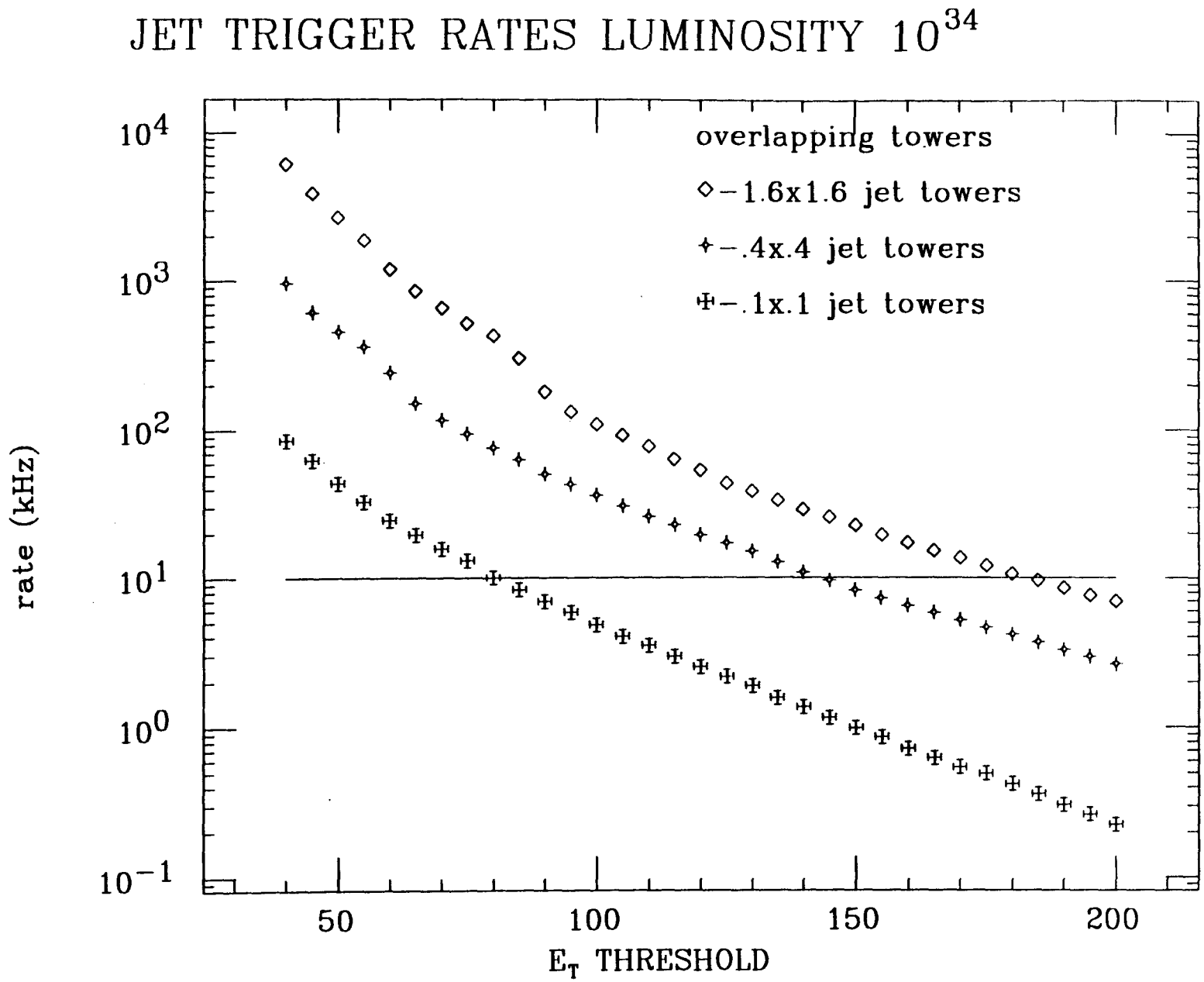

JET TRIGGER RATES LUMINOSITY 10^{34}
overlapping towers
◇–1.6x1.6 jet towers
✧–.4x.4 jet towers
✠–.1x.1 jet towers
rate (kHz)
10^4
10^3
10^2
10^1
10^0
10^{-1}
50
100
150
200
E_T THRESHOLD

Figure 2

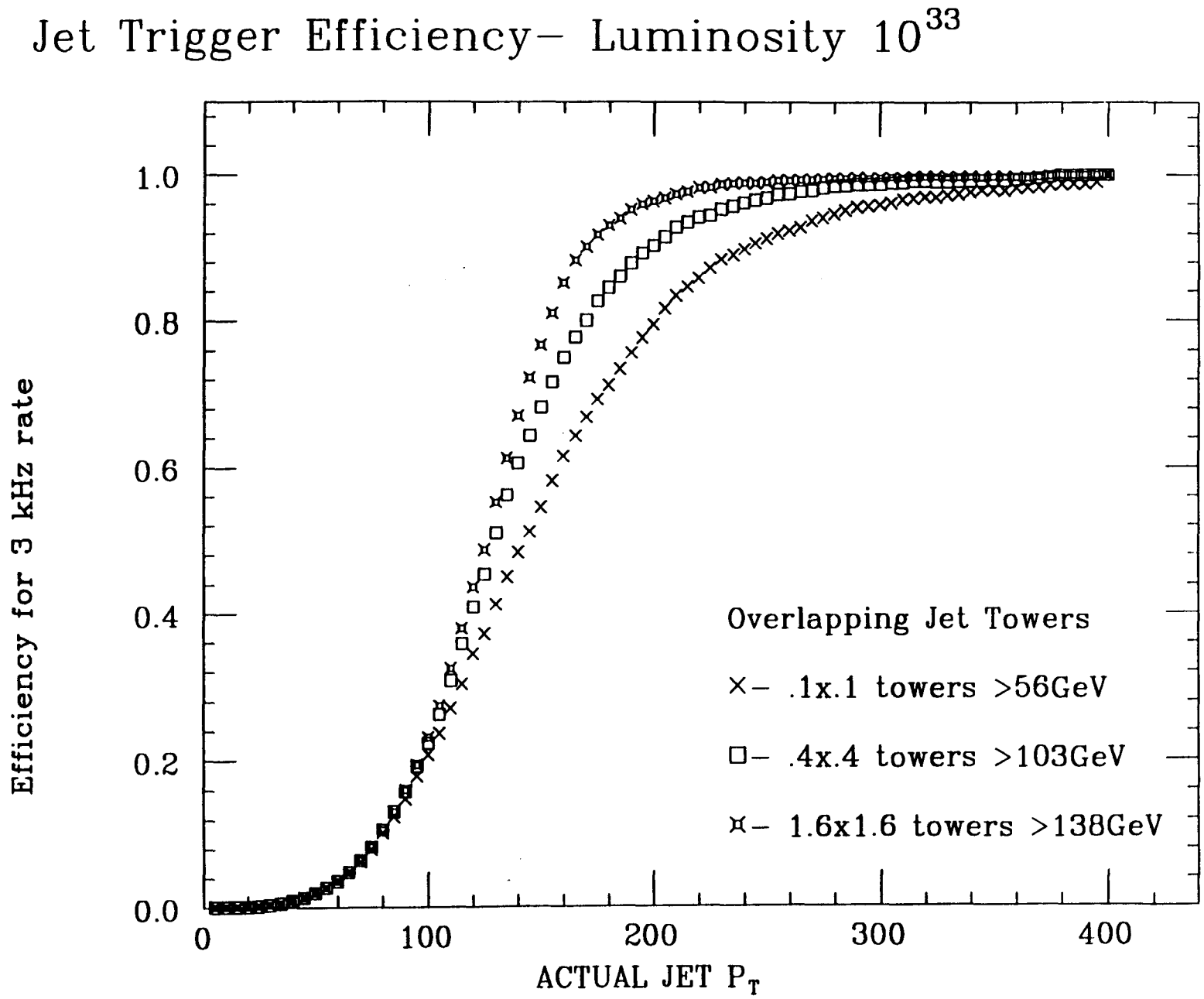

Jet Trigger Efficiency– Luminosity 10^{33}
Efficiency for 3 kHz rate
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
ACTUAL JET P_T
Overlapping Jet Towers
×– .1x.1 towers >56GeV
□– .4x.4 towers >103GeV
¤– 1.6x1.6 towers >138GeV

Figure 3

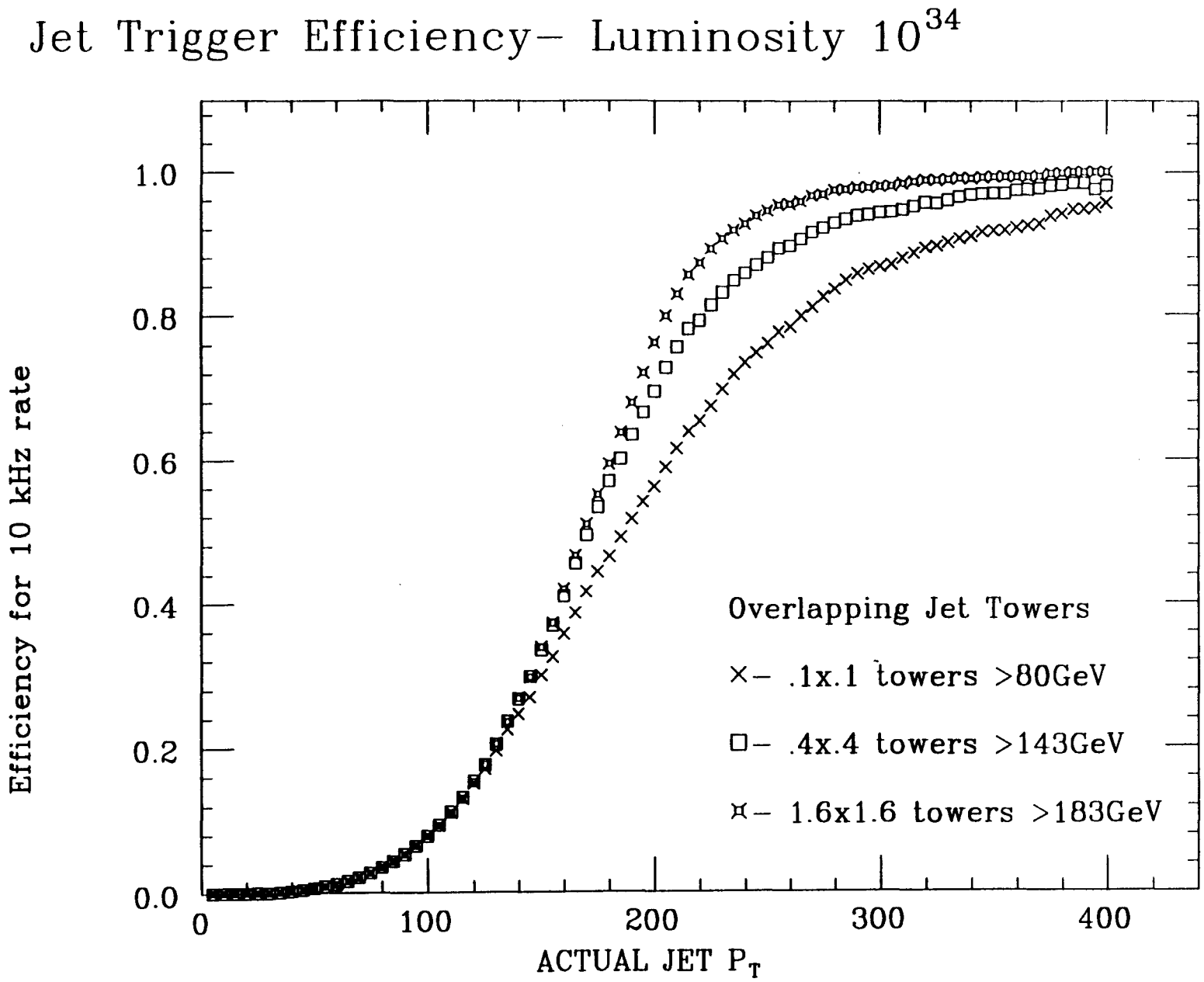

Jet Trigger Efficiency– Luminosity 10^{34}
Efficiency for 10 kHz rate
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
ACTUAL JET P_T
Overlapping Jet Towers
×– .1x.1 towers >80GeV
□– .4x.4 towers >143GeV
¤– 1.6x1.6 towers >183GeV

Figure 4

Jet Trigger Efficiency– Luminosity 10^{33}

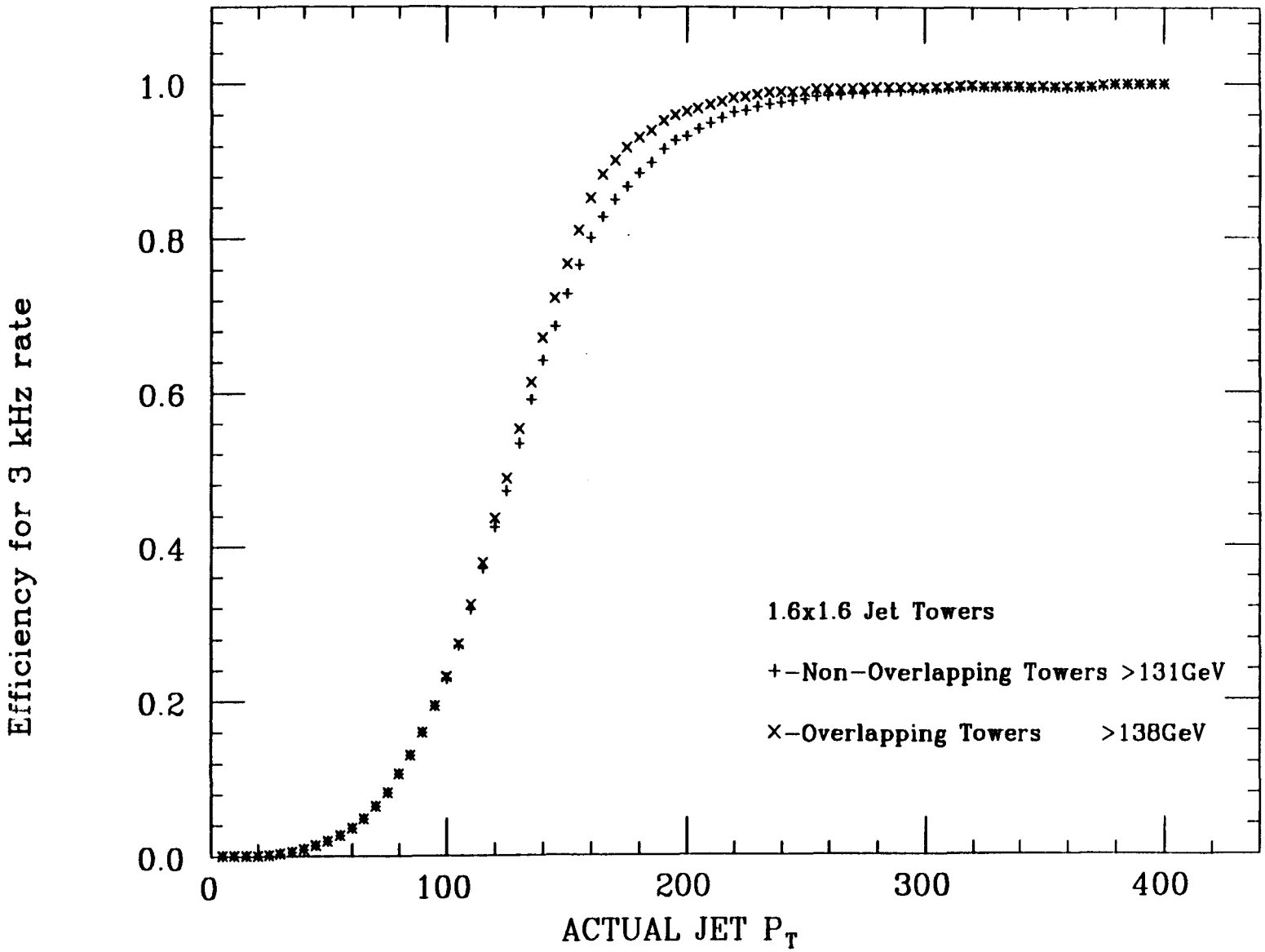


Figure 5

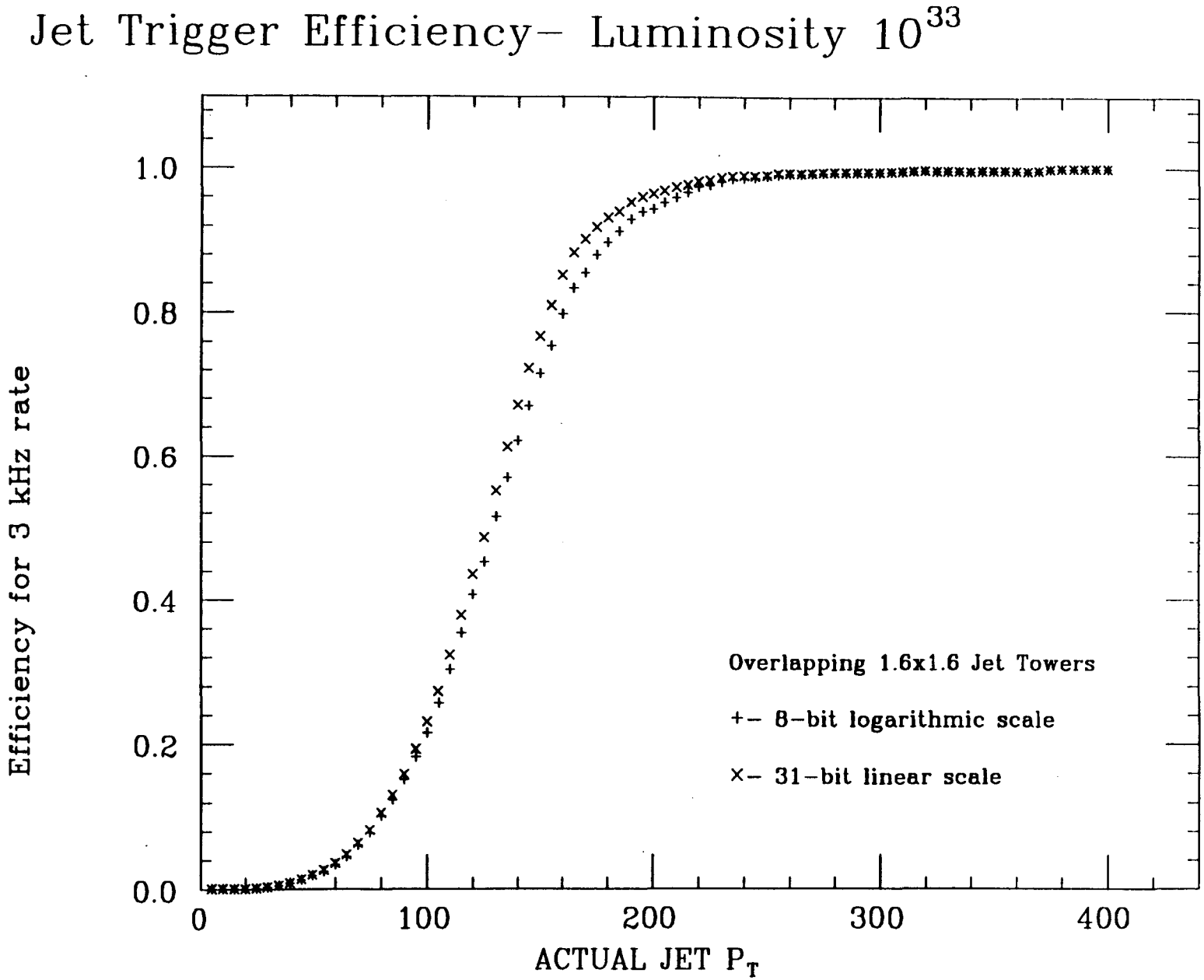

Jet Trigger Efficiency– Luminosity 10^{33}
Efficiency for 3 kHz rate
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
ACTUAL JET P_T
Overlapping 1.6x1.6 Jet Towers
+– 8–bit logarithmic scale
×– 31–bit linear scale

Figure 6

Type of jet trigger	3 kHz at 10^{33}	10 kHz at 10^{34}
$.1 \times .1$	287	387
$.4 \times .4$ overlapping	229	313[2]
1.6×1.6 overlapping	190	253[3]
$.4 \times .4$ non-overlapping	243[3]	336[2]
1.6×1.6 non-overlapping	211	281[3]
all have low energy cutoff at .1 GeV except as noted [2] – low energy cutoff at .2 GeV [3] – low energy cutoff at .5 GeV		

Table 1: Jet p_T at which 95% efficiency is achieved (GeV)

Electron Trigger Study

- Effect of Hadronic/Electromagnetic energy deposit ratio cut
- Effect of transverse isolation
- Effect of track requirement or photon trigger.
- How will increased luminosity of 10^{34} change the performance of trigger?

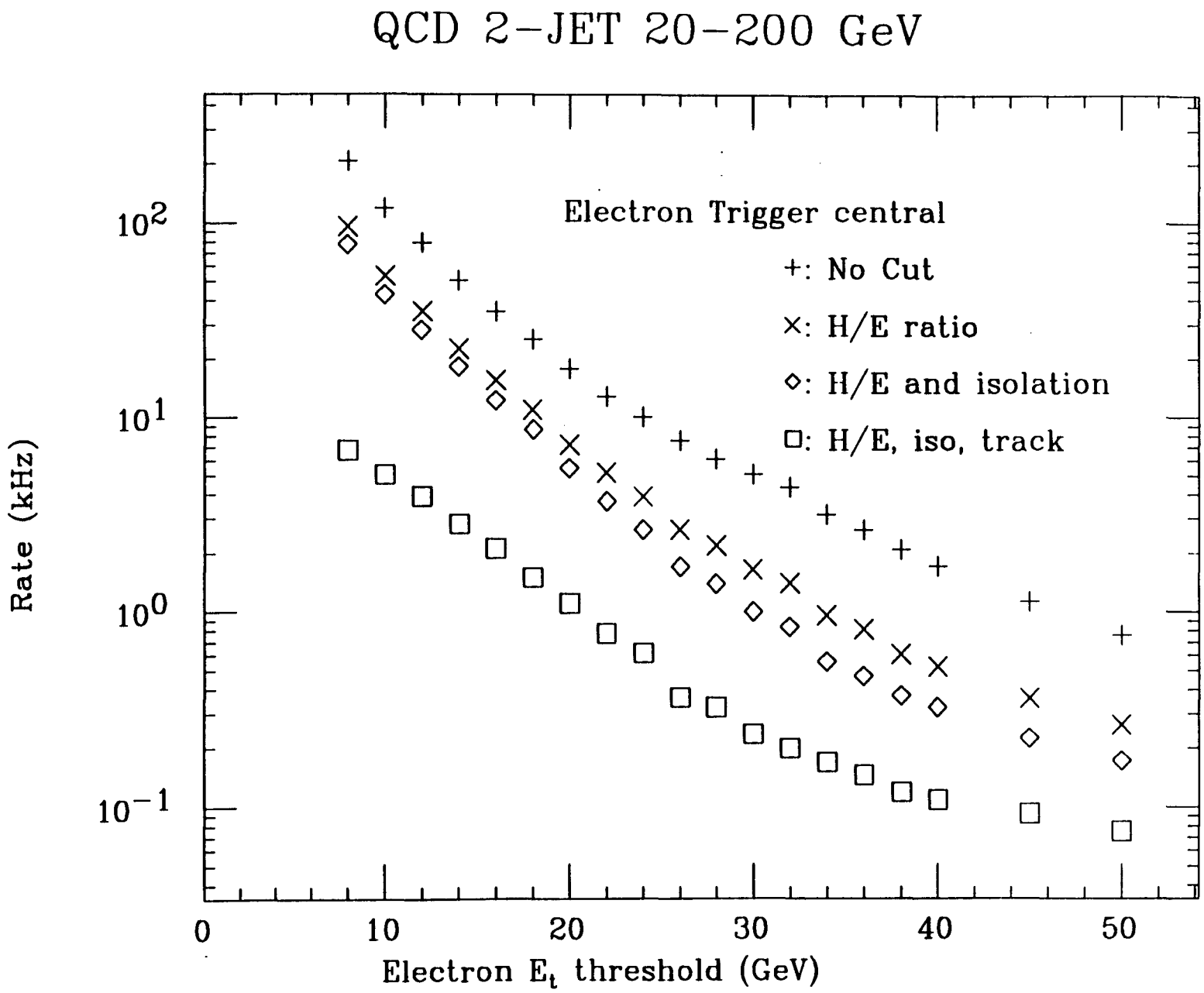
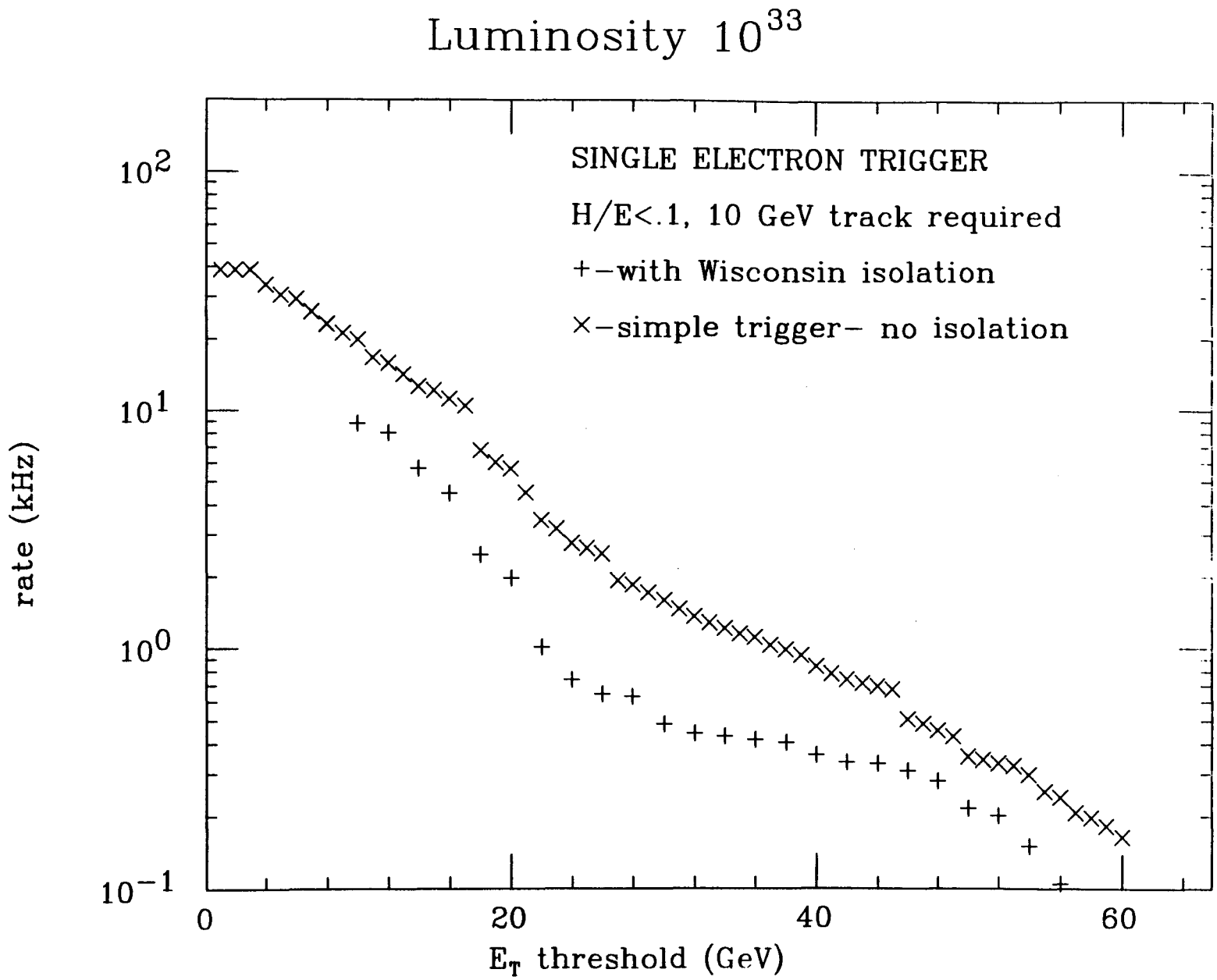
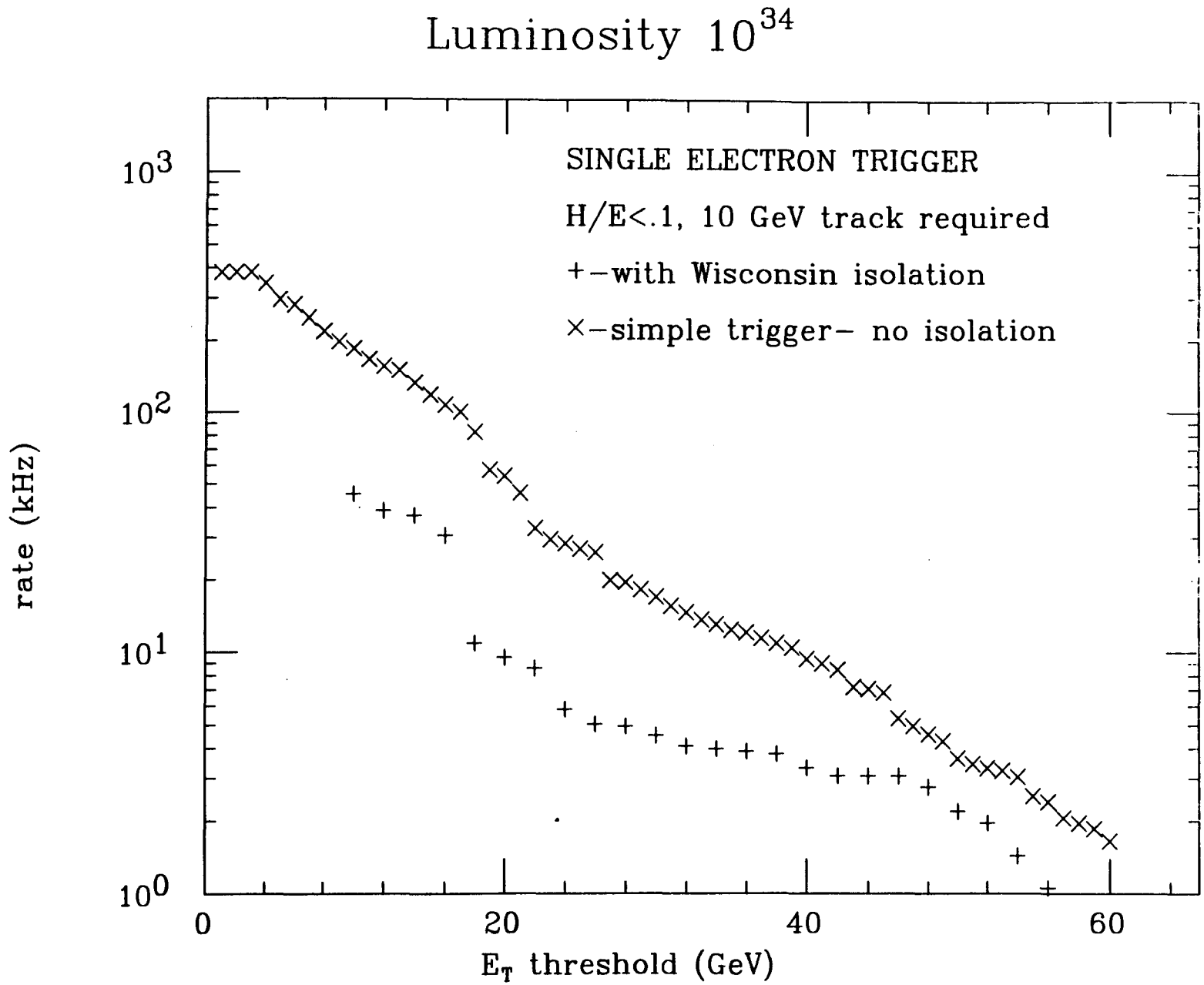

QCD 2-JET 20-200 GeV
Electron Trigger central
+: No Cut
×: H/E ratio
◇: H/E and isolation
□: H/E, iso, track
Rate (kHz)
10^2
10^1
10^0
10^{-1}
0
10
20
30
40
50
Electron E_t threshold (GeV)

Figure 7


Luminosity 10^{33}
SINGLE ELECTRON TRIGGER
H/E<.1, 10 GeV track required
+−with Wisconsin isolation
×−simple trigger− no isolation
rate (kHz)
10^2
10^1
10^0
10^{-1}
0
20
40
60
E_T threshold (GeV)


Luminosity 10^{34}
SINGLE ELECTRON TRIGGER
H/E<.1, 10 GeV track required
+−with Wisconsin isolation
×−simple trigger− no isolation
rate (kHz)
10^3
10^2
10^1
10^0
0
20
40
60
E_T threshold (GeV)

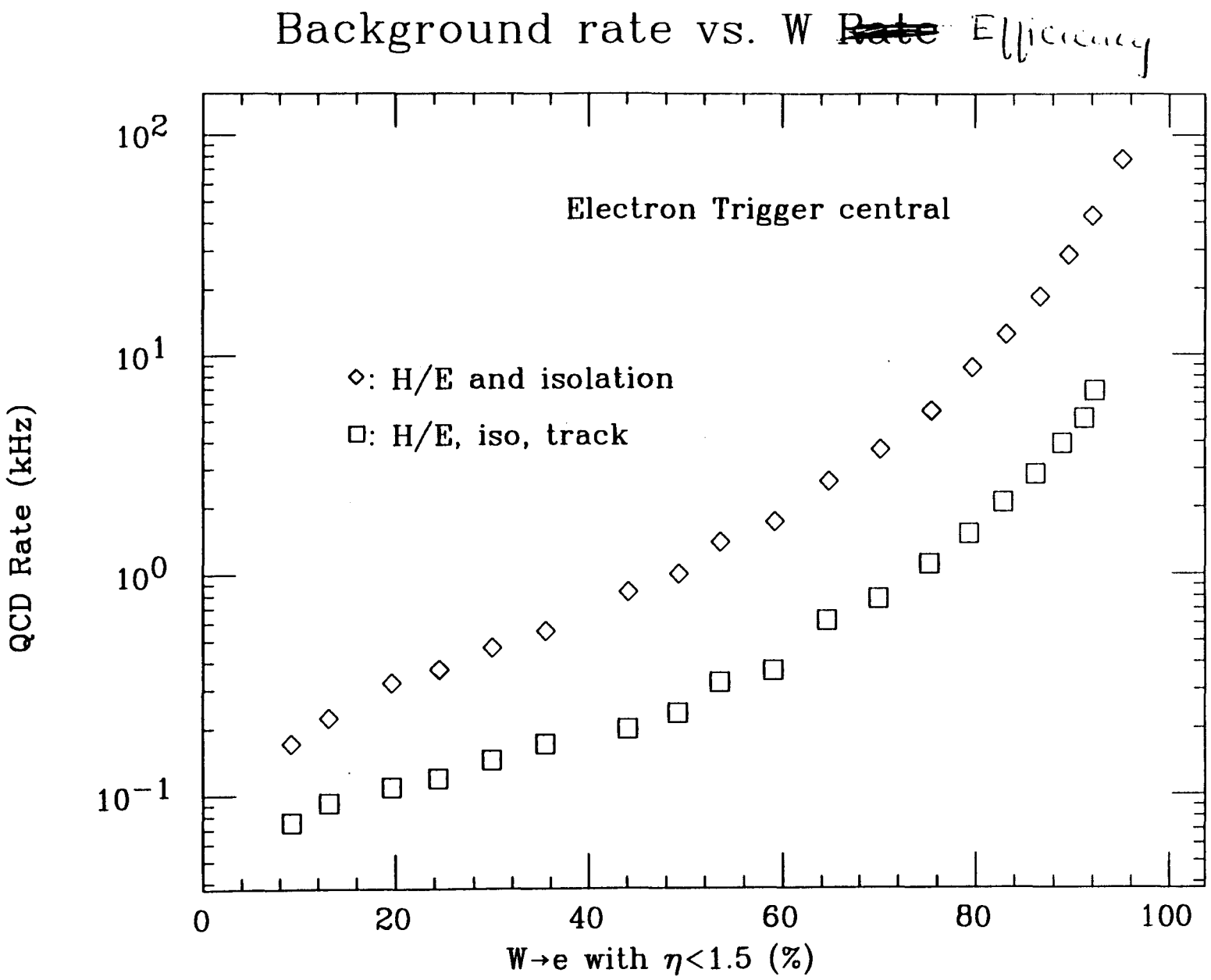

Background rate vs. W ~~Rate~~ Efficiency
Electron Trigger central
◇: H/E and isolation
□: H/E, iso, track
QCD Rate (kHz)
10^2
10^1
10^0
10^{-1}
0
20
40
60
80
100
W→e with η<1.5 (%)

Figure 8

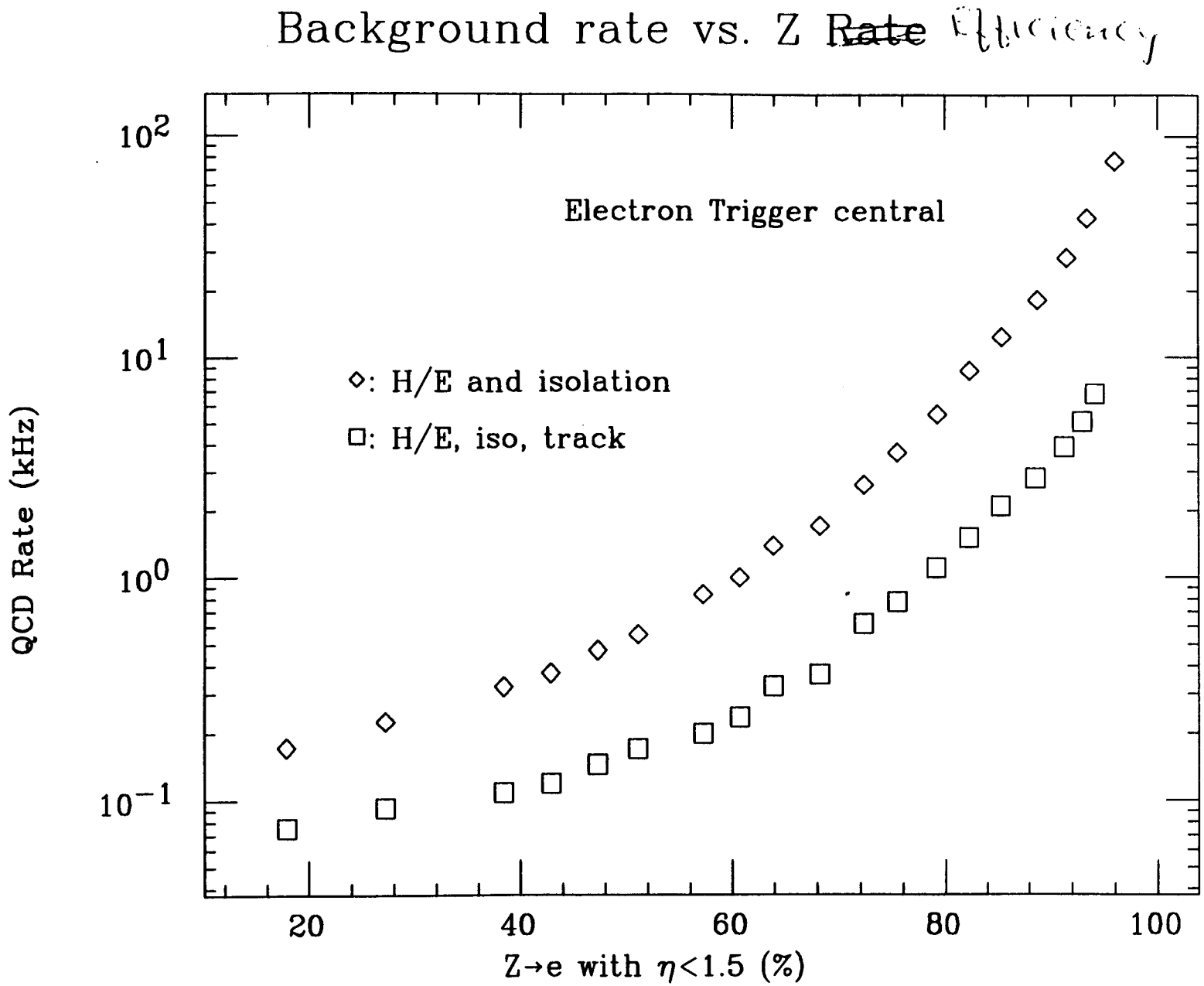

Background rate vs. Z Rate Efficiency
Electron Trigger central
◇: H/E and isolation
□: H/E, iso, track
QCD Rate (kHz)
10^2
10^1
10^0
10^{-1}
20
40
60
80
100
Z→e with η<1.5 (%)

Figure 9

Type of trigger		1 kHz		3 kHz		10 kHz	
		E/H only	E/H, ISO	E/H only	E/H, ISO	E/H only	E/H, ISO
10^{33}	1γ	47	40	34	30	25	22
	$1e$	38	24	24	18	18	< 10
	2γ	17	< 10	14	< 10	11	< 10
	$2e$	< 10	< 10	< 10	< 10	< 10	< 10
10^{34}	1γ	82	> 58	62	54	48	38
	$1e$	77	58	55	48	40	20
	2γ	31	14	23	12	18	< 10
	$2e$	20	< 10	14	< 10	< 10	< 10

Table 1: E_T thresholds required to achieve given rates.

Representative level 1 trigger E_t thresholds.

Trigger	Threshold
Jet (1.6×1.6 sum)	140 GeV
Electron	20 GeV
Photon	30 GeV
Muon	20 GeV
Missing E_t	80 GeV
Two electrons	10 GeV
Two photons	20 GeV

Conclusions

- SDC L1 trigger simulations indicate that the goals of reducing the interaction rate from 10^8 down to more manageable 10^4 - 10^5 can be met in the current baseline design, with good efficiency for interesting events.

- SDC L1 trigger simulations indicate that may be gains to be made, using wider grids for tower energy sums in jet trigger; and using transverse isolation criteria for electron trigger.

- These gains are better realized at higher luminosity.

- More studies, including better simulation of detector response, in particular for tracking, are needed.

G. Sullivan
U. Chicago

Trigger Rates, DAQ, & Online Processing at the SSC.

or

How to get the Physics to Tape

Simulation Workshop
SSCL
May 22-25, 1992

Boundry Conditions

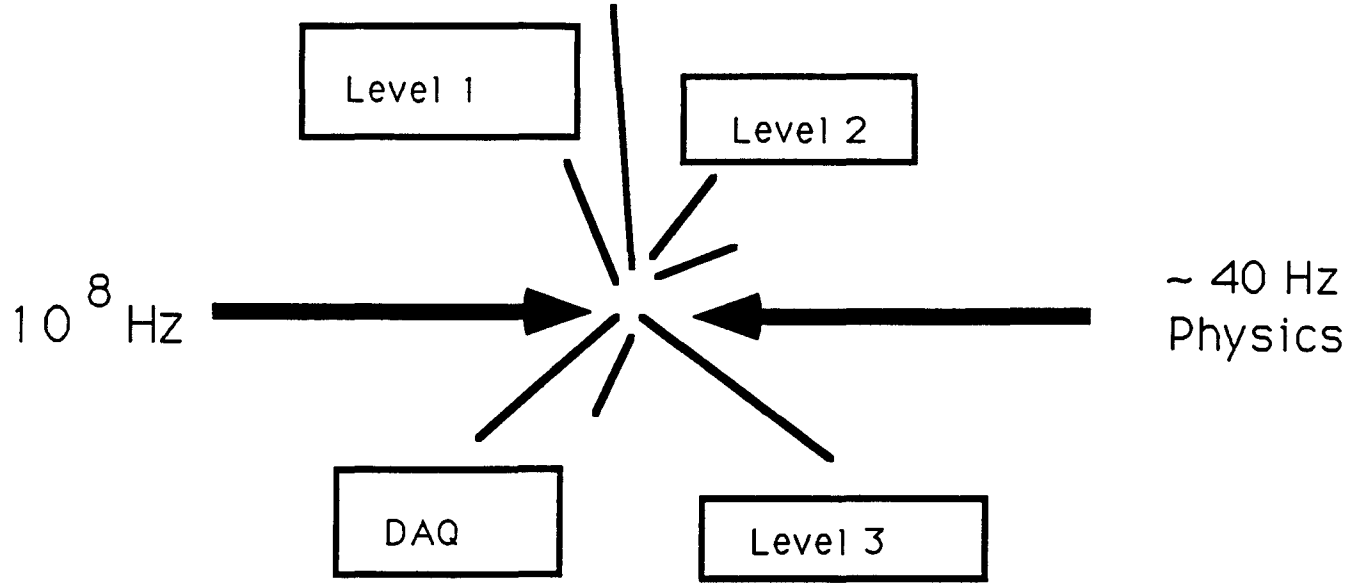
- 100 million interactions/second
  16 nS bunch spacing

- <u>Physics Rate</u> @ 10^{33}

$W \longrightarrow \mu\nu$	10 Hz	
$W \longrightarrow e\nu$	10 Hz	
$t\bar{t}$	20 Hz	(M_{top} = 140 GeV)
	~40 Hz	

- <u>Rate to Tape</u>

  10's - 100 Hz

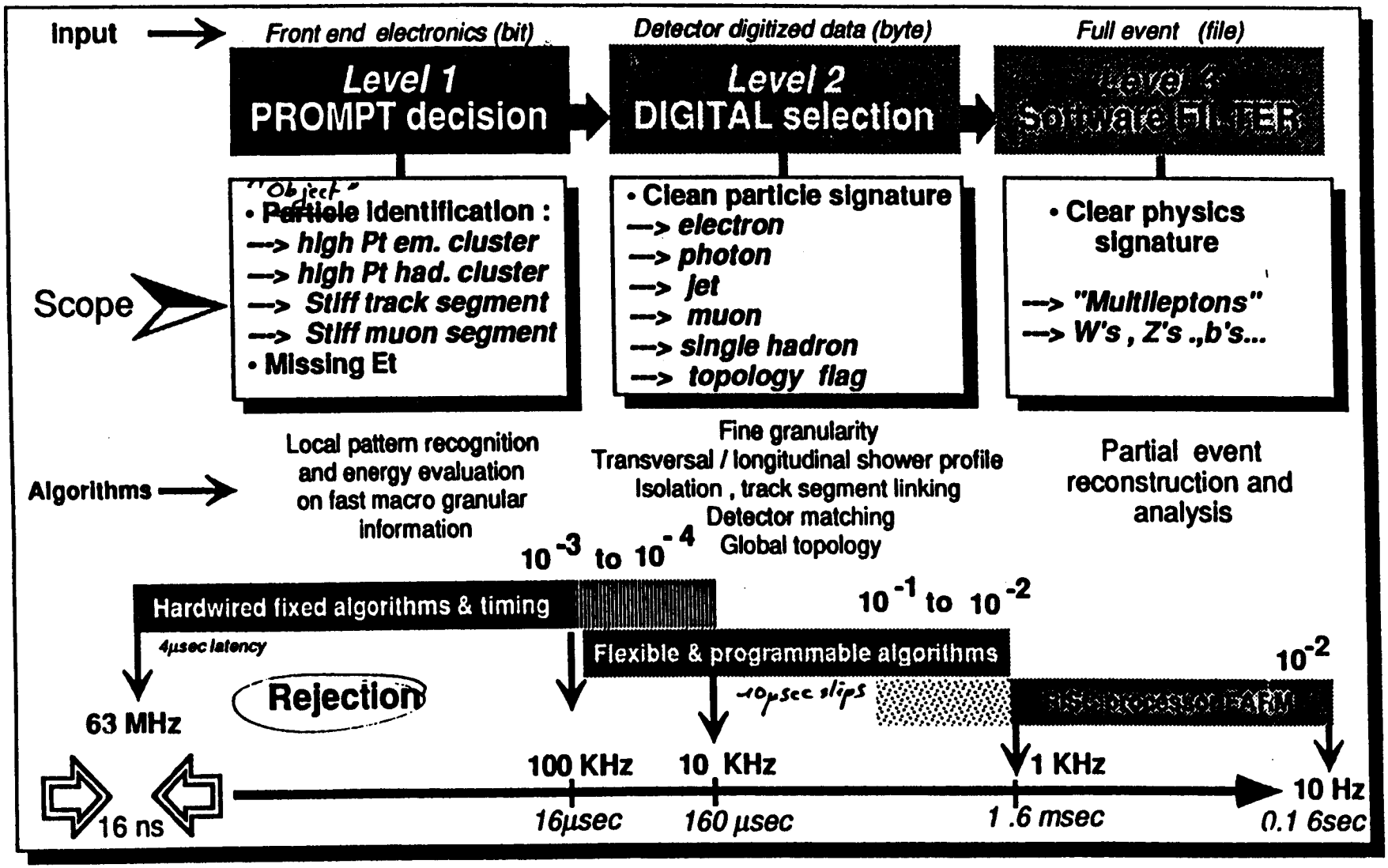


- Tape speed ~ Physics rate
  - Final Trigger level must be close to offline cuts
  - Full or nearly full reconstruction
  - Lots of MIPs - must minimize rate into final level

- 16 nS Beam spacing
  - Fast Level 1, fixed algorithms
  - Not enough reduction in 16 nS decision for readout

- 3 Level Trigger System
  - Balance between Level 1, Level 2, Level 3
  - Simplify Level 1 - reduce rate for Level 2 (cost eff)
  - Level 2 is programmable - reduce rate for DAQ/L3

Event selection scheme



SDC level 2 trigger -

3/3/92

Patrick LeDu

Trigger Rate Simulation

Isajet plus fast detector simulation. Uses parameterizations of detector response.
(SDC note SDC-91-00099)

Very fast Monte carlo (20 S/event on 3100/76)

Includes:

- Multiple interactions
- Vertex position smearing
- Calorimeter response shaping function
- π and K Deacys
- Photon conversions
- Tracking resolution and efficiency from full simulations of tracking trigger
- Calorimeter longitudinal & lateral shower fluctuations
- Electron Bremsstrahlung
- Shower maximum detector parameterized from full simulation

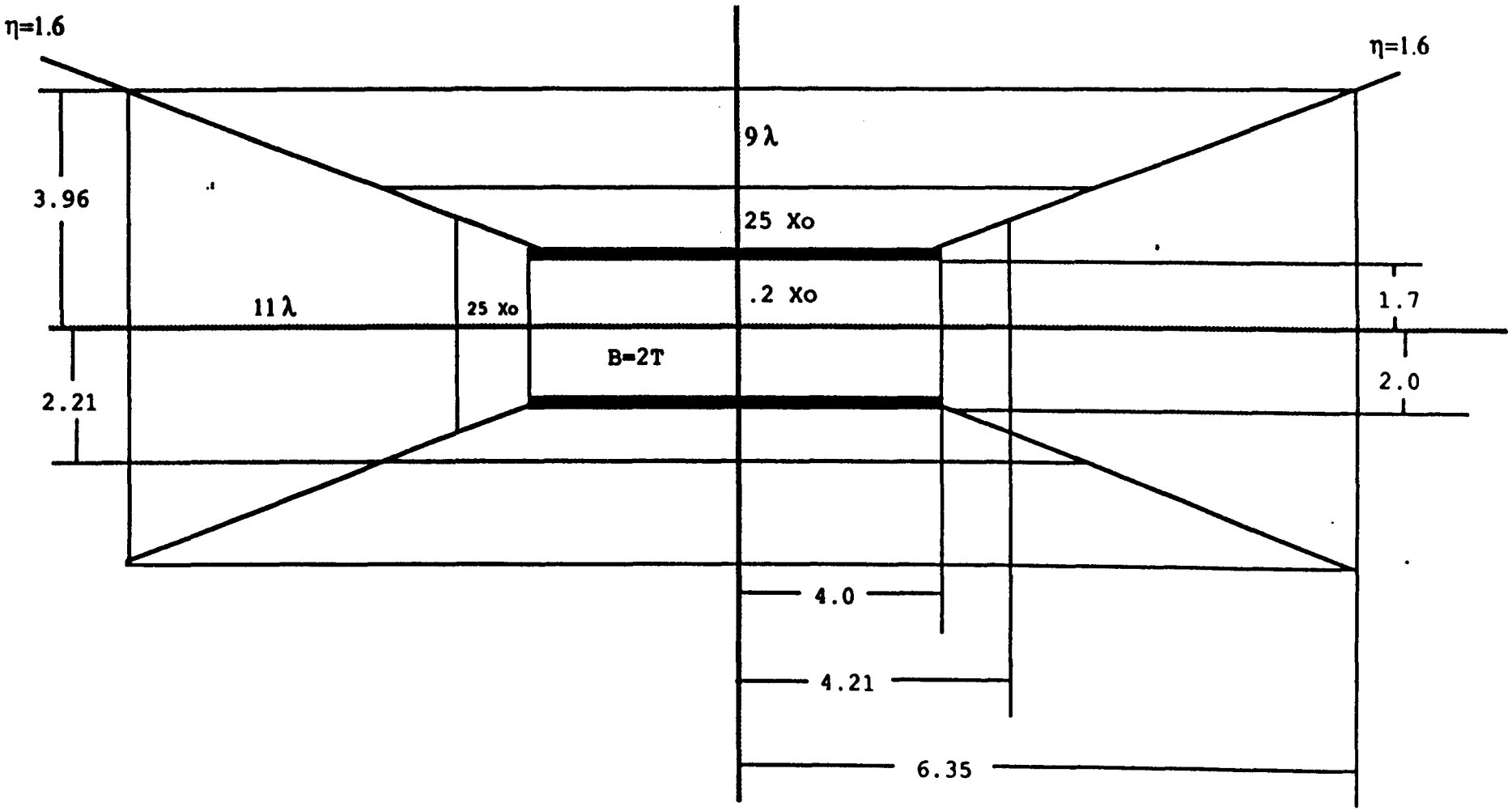
Simulation Parameters

$\eta < 1.6$

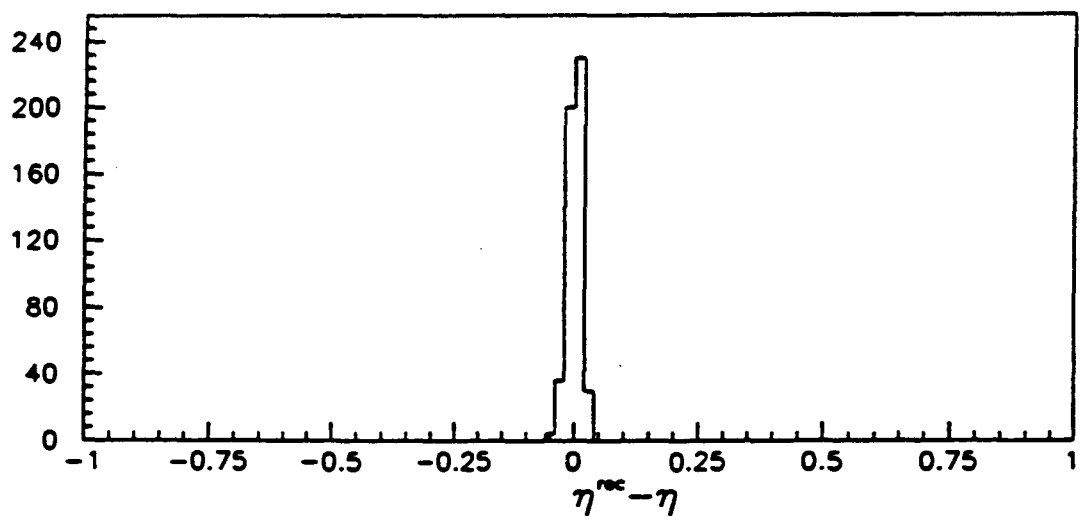
		trigger
128 ϕ towers	→	64ϕ
$\Delta\eta = .05$	→	$\Delta\eta = 0.1$

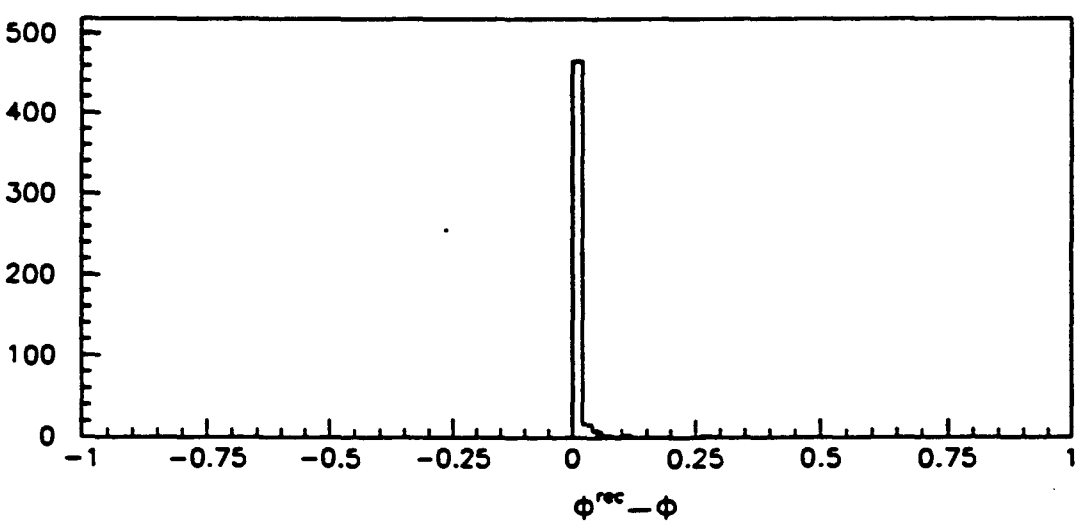
$1.6 < \eta < 3.0$

		trigger
64 ϕ towers	→	64 ϕ towers
$\Delta\eta =$ 4 x .05	→	2 x 0.1
4 x .075		2 x 0.15
4 x .125		2 x 0.25
2 x .2		1 x 0.4

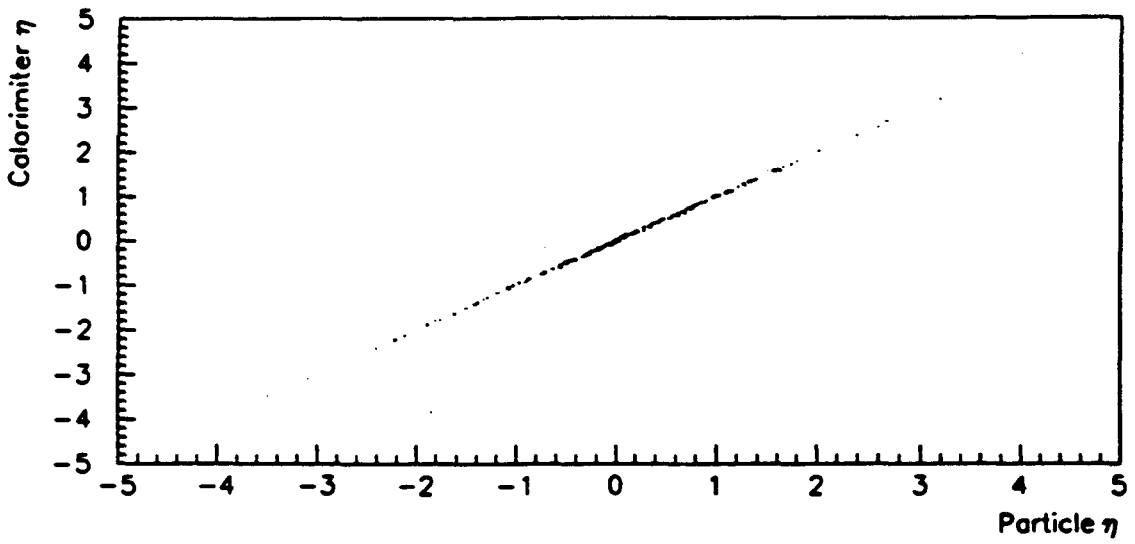


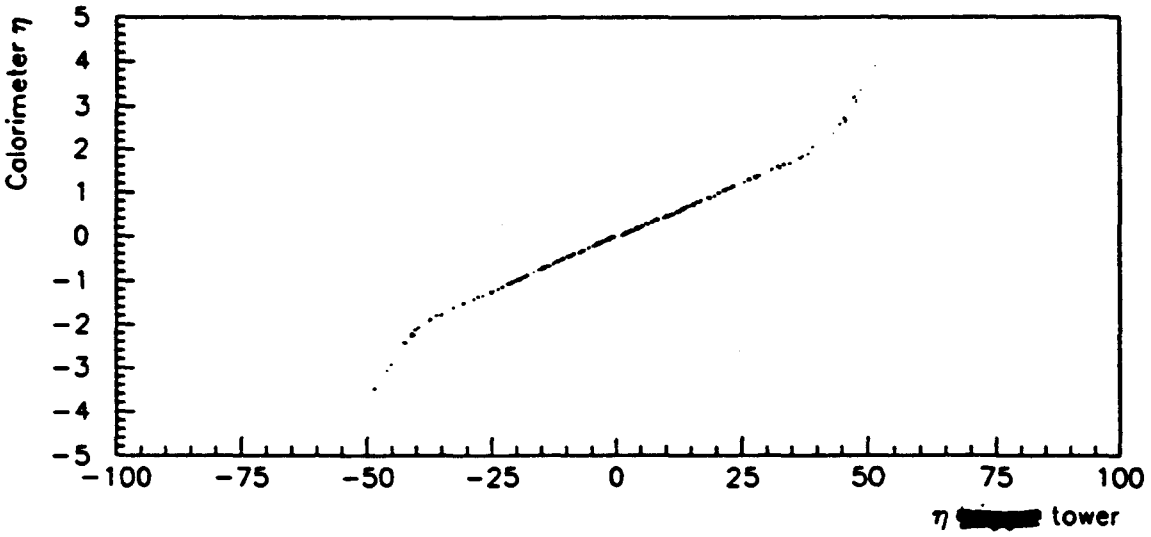
CHECK OF SIMULATION USING SINGLE PARTICLES

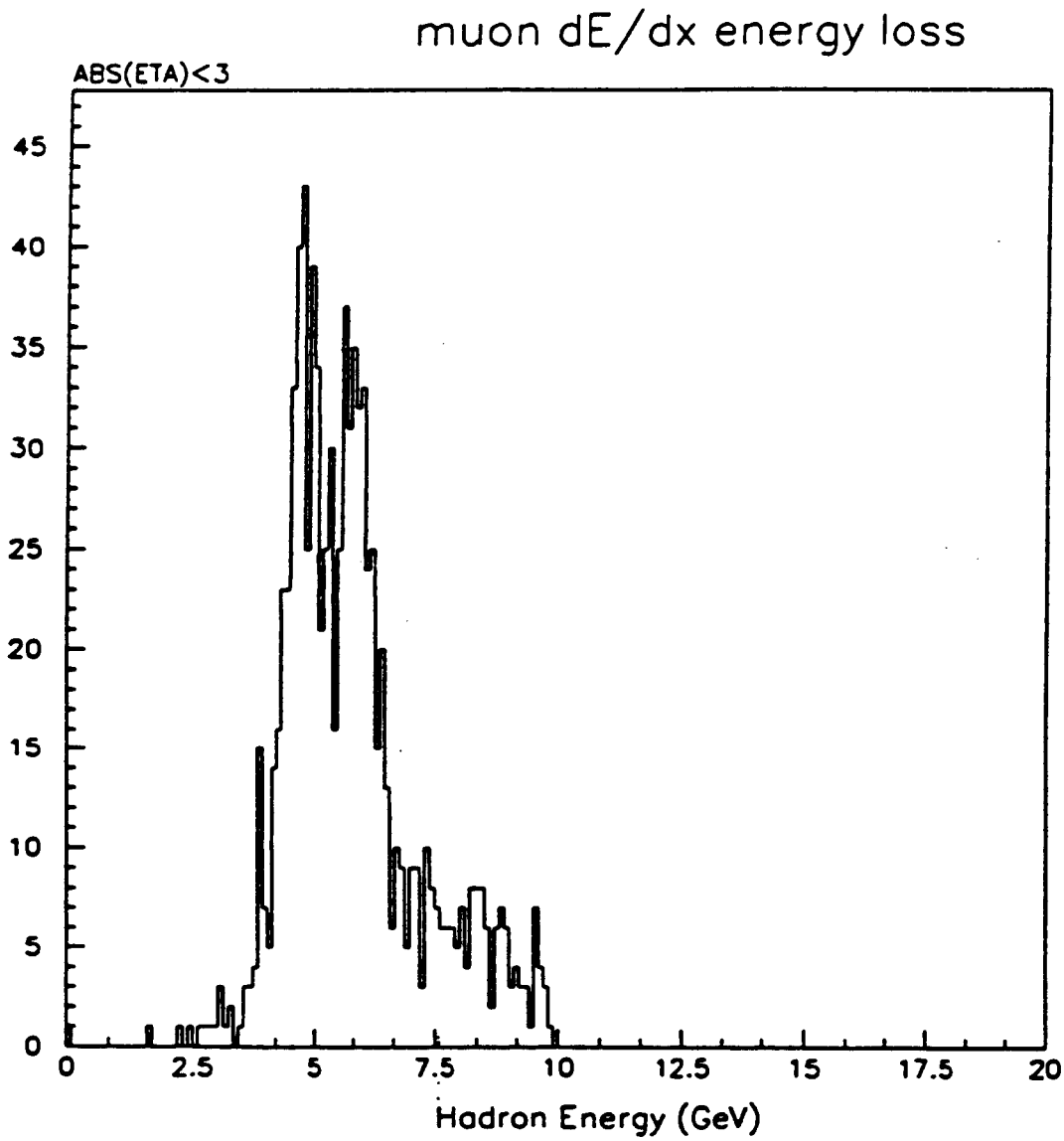

240
200
160
120
80
40
0
-1
-0.75
-0.5
-0.25
0
0.25
0.5
0.75
1
$\eta^{rec} - \eta$


500
400
300
200
100
0
-1
-0.75
-0.5
-0.25
0
0.25
0.5
0.75
1
$\phi^{rec} - \phi$

η check

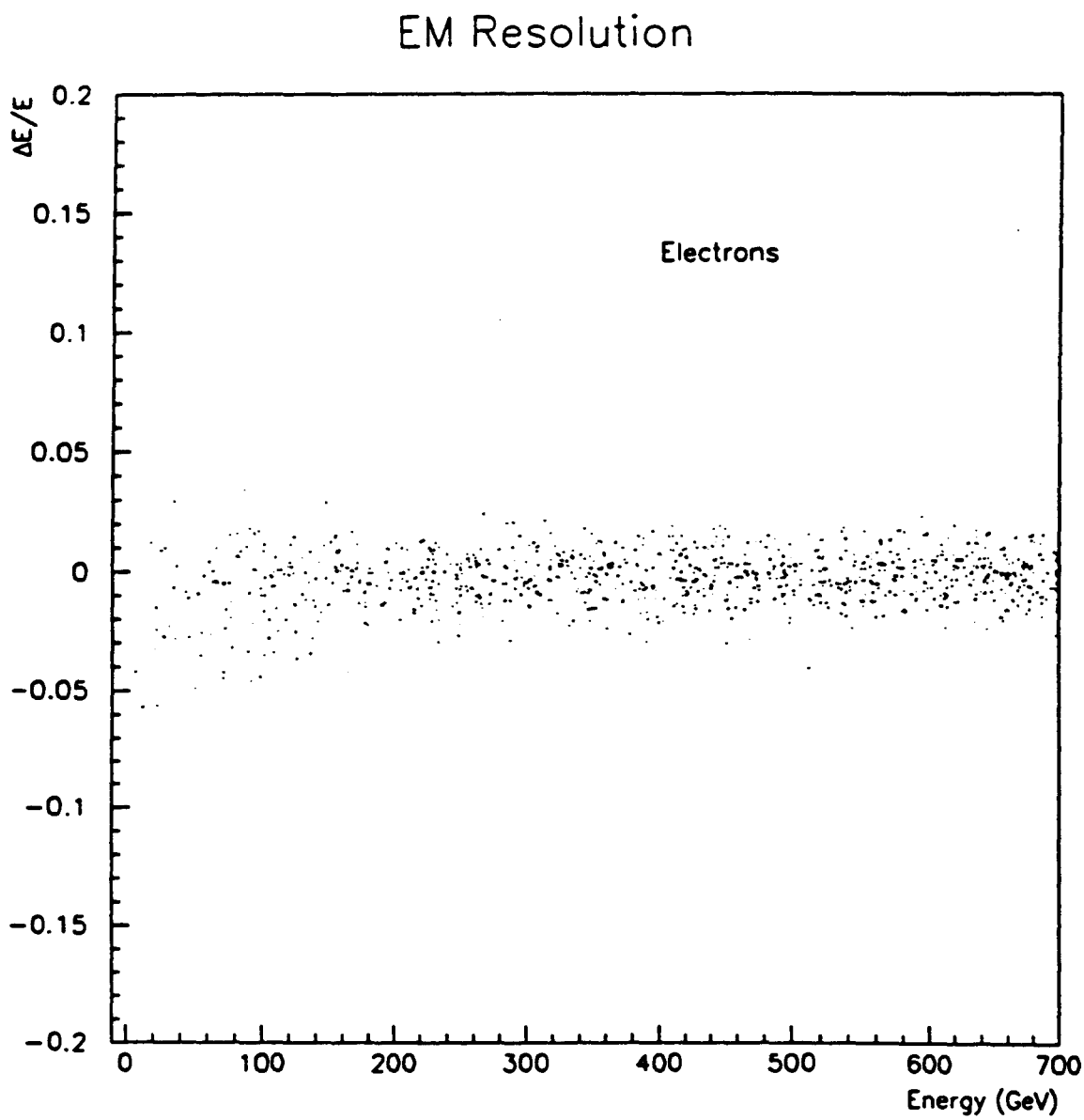

Colorimiter η
5
4
3
2
1
0
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
0
1
2
3
4
5
Particle η


Colorimeter η
5
4
3
2
1
0
−1
−2
−3
−4
−5
−100
−75
−50
−25
0
25
50
75
100
η [redacted] tower


muon dE/dx energy loss
ABS(ETA)<3
45
40
35
30
25
20
15
10
5
0
0
2.5
5
7.5
10
12.5
15
17.5
20
Hadron Energy (GeV)

EM: $\frac{\Delta E}{E} = \frac{.15}{\sqrt{E}} \oplus .01$

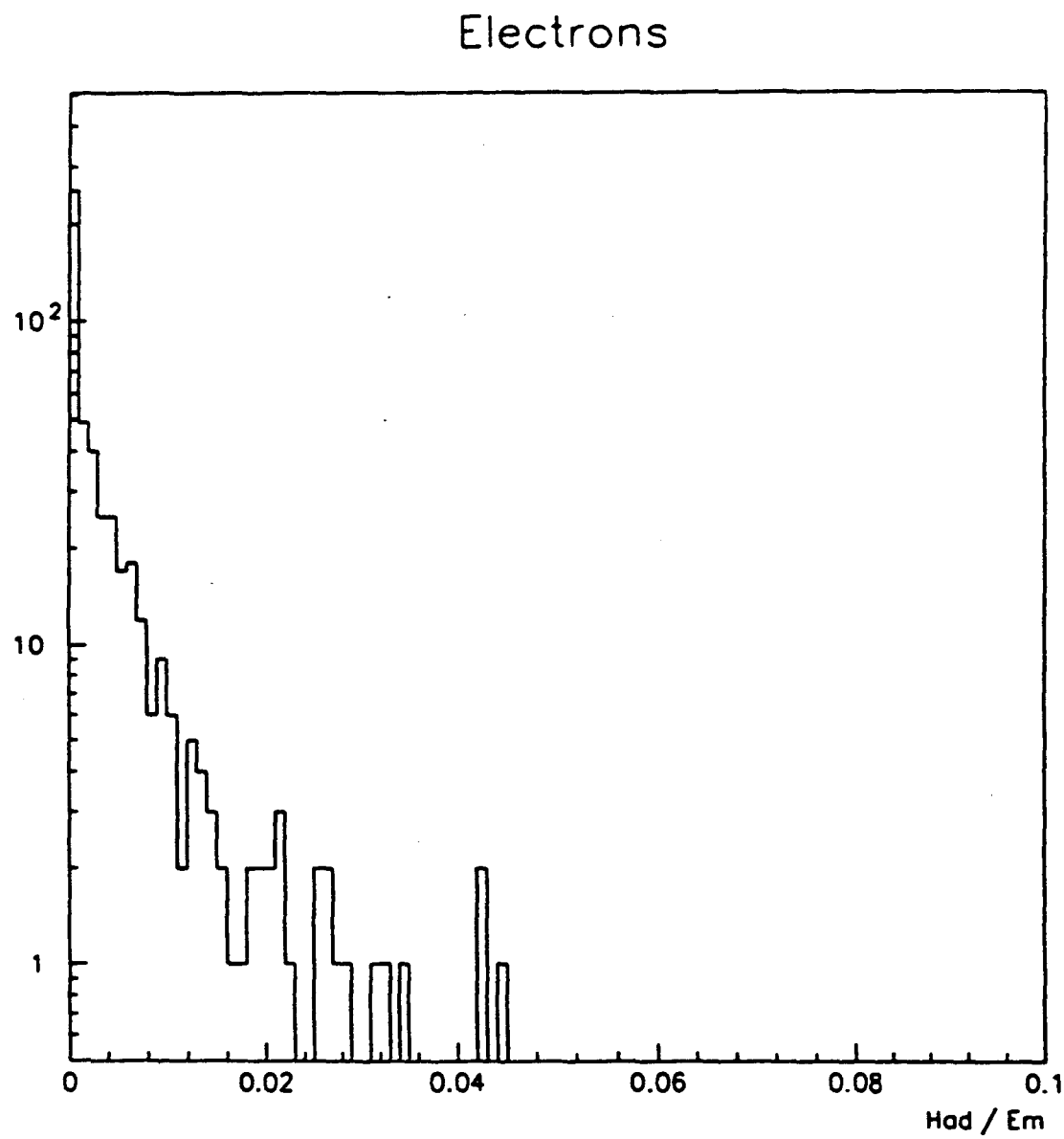
HAD: $\frac{\Delta E}{E} = \frac{.40}{\sqrt{E}} \oplus .02$



1) .1 × .1 tower (in Central)

2) 1 GeV < E_{Elec} < 700 GeV

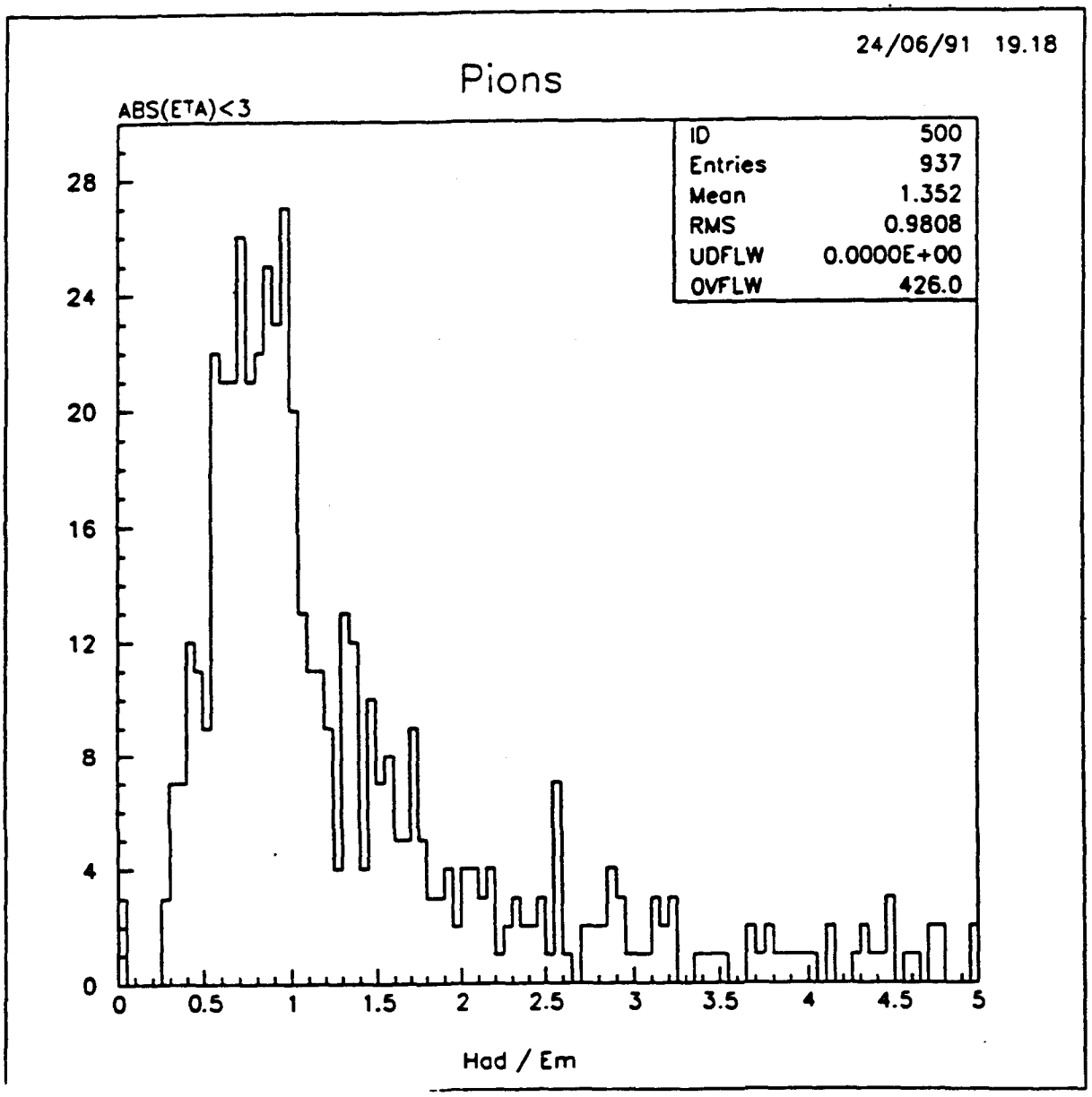
3) $|\eta| < 3.0$



1) .1 × .1 tower (in Central)

2) 1 GeV < E_{pion} < 700 GeV

3) $|\eta| < 3.0$



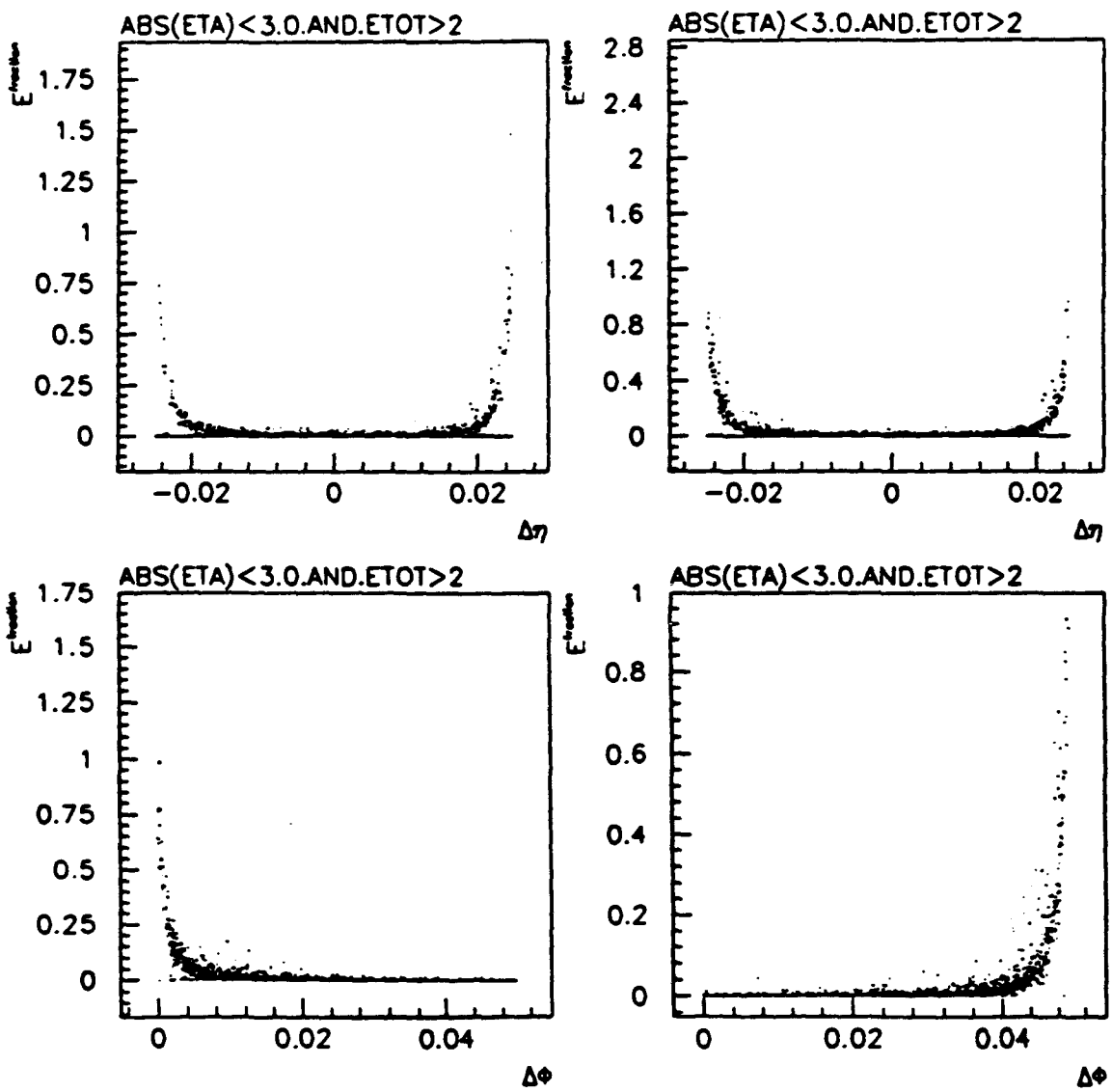
~~Fract~~ Ratio of Energy in adjacent tower

1) Single physical tower

2) $2\ GeV < E_{elec} < 700\ GeV$

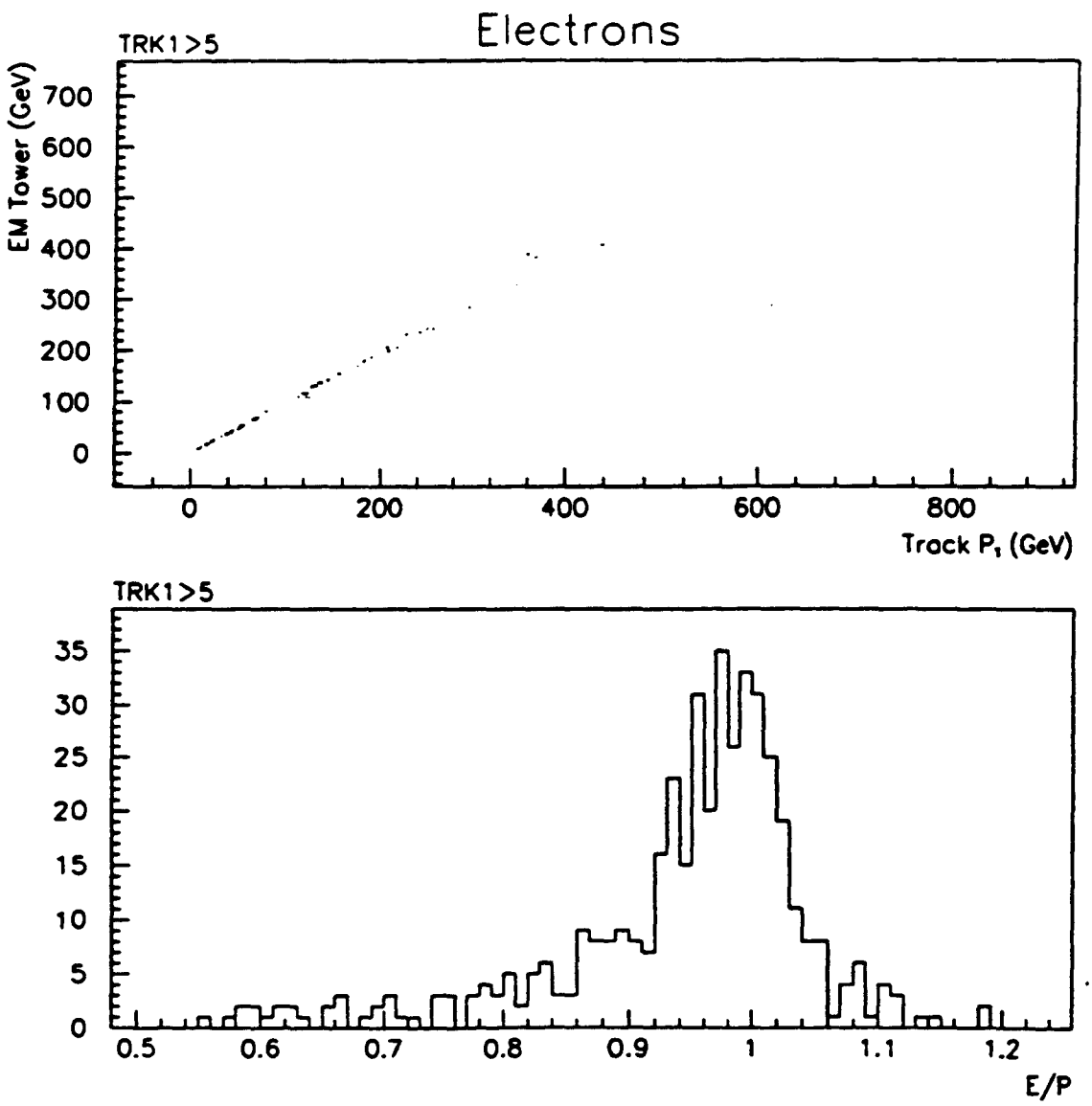
3) $|\eta| < 3.0$

Border Towers - Electrons

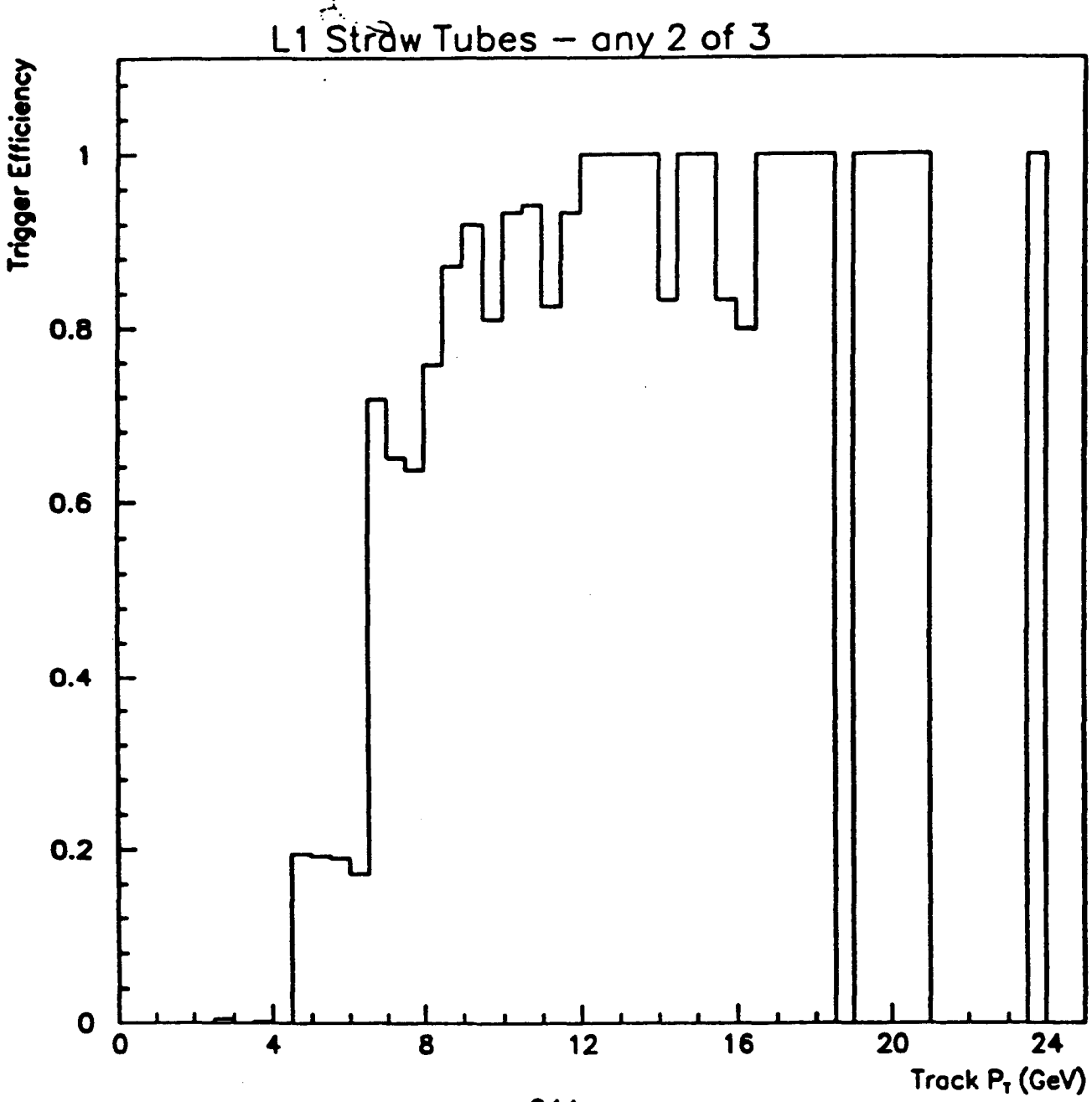


1) $E \equiv$ Single tower em energy

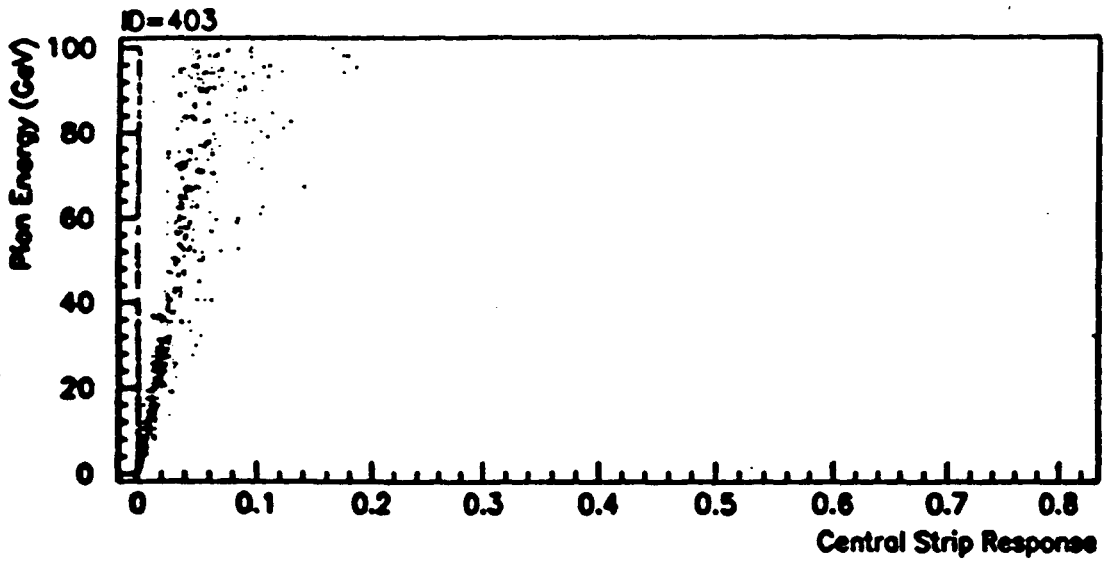
2) $P_T > 5.0$ GeV



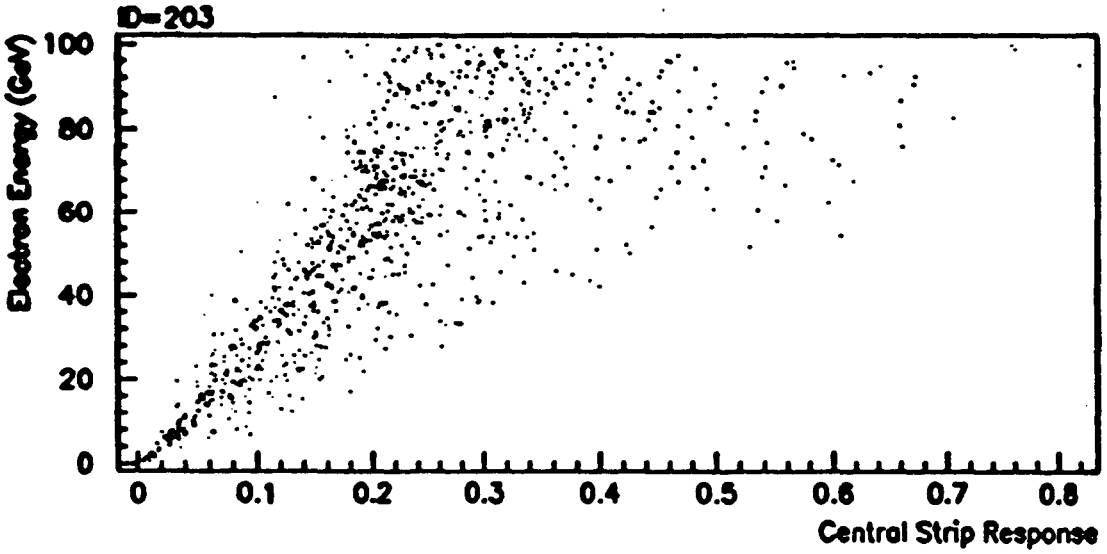
L1 Tracking efficiency vs. Track P_T



Shower-Max Detector Central strip response vs. particle energy for pions and electrons

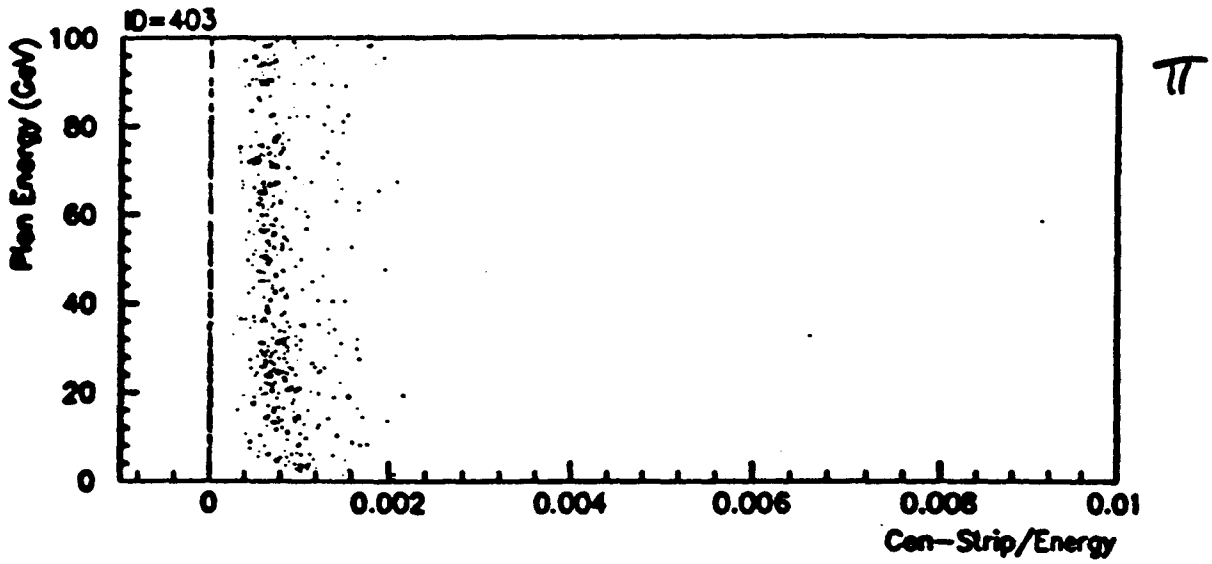


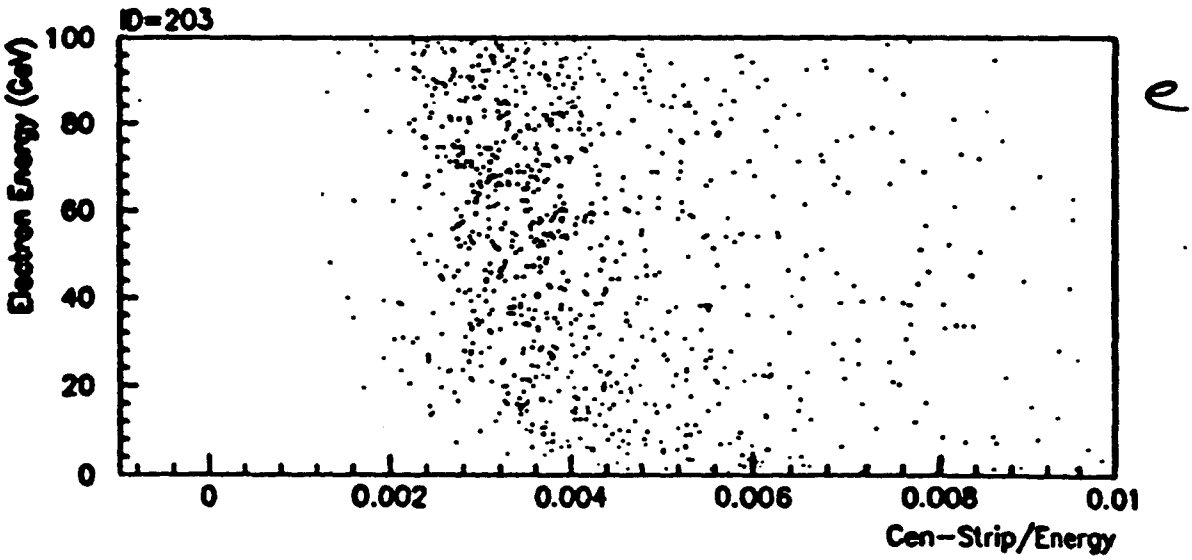
π



e

Shower Max Detector central strip response normalized by particle energy vs. particle energy for pions and electrons



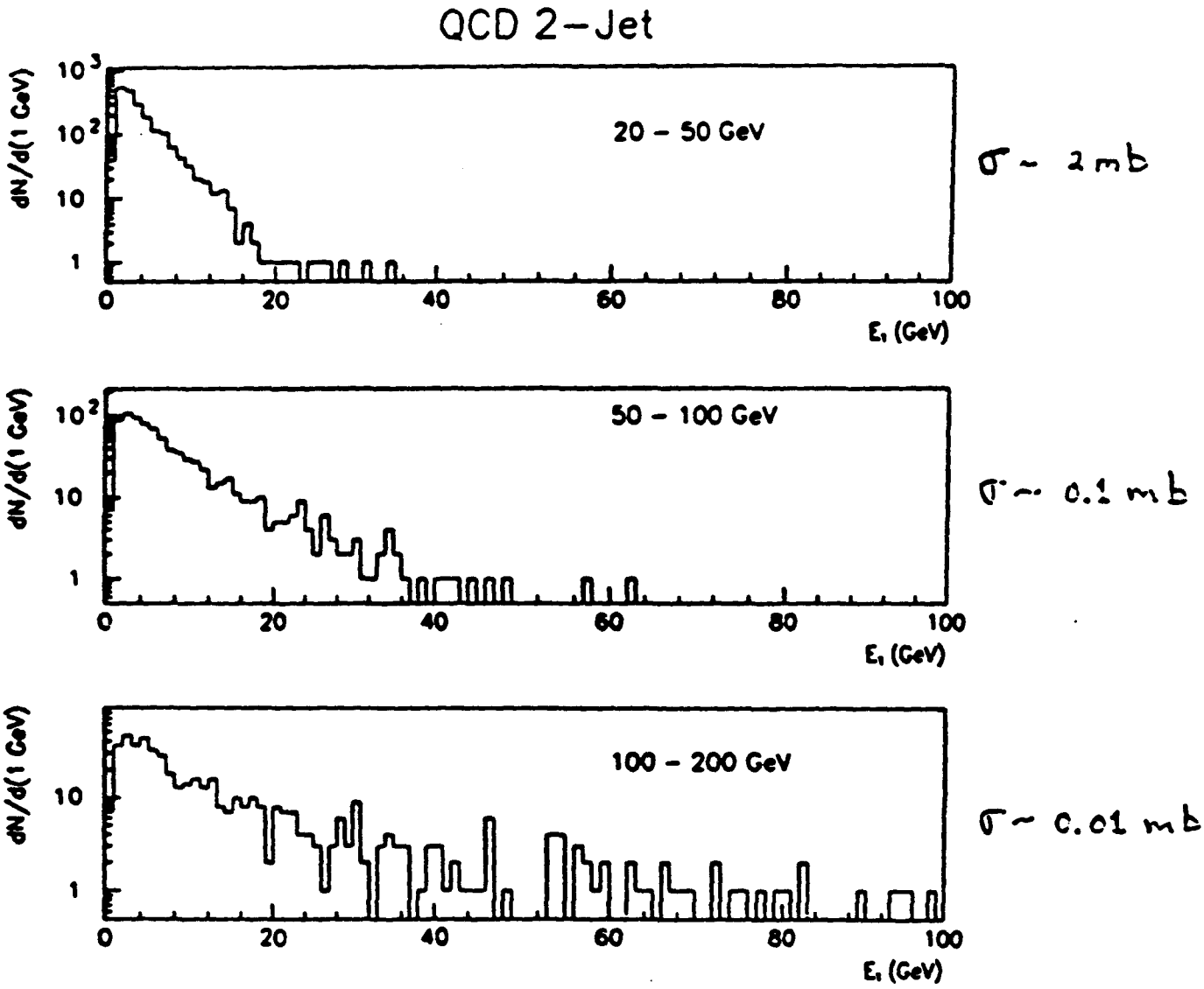


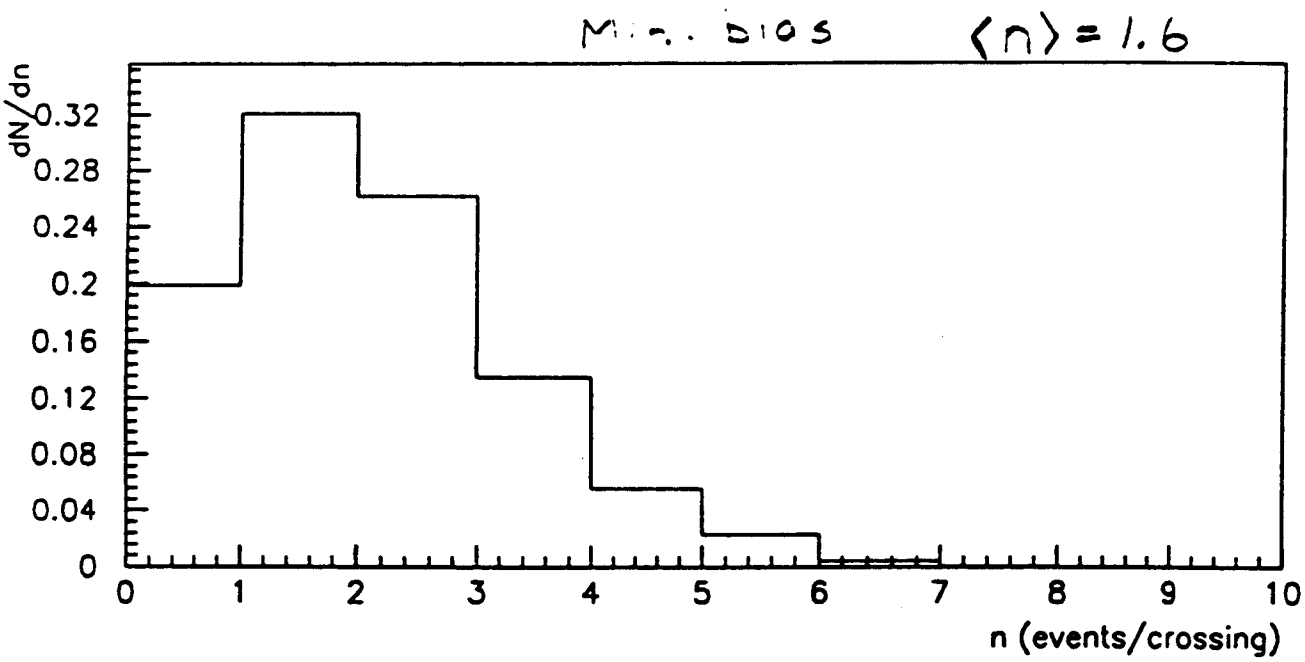
SDC Level 1
Trigger Rtaes

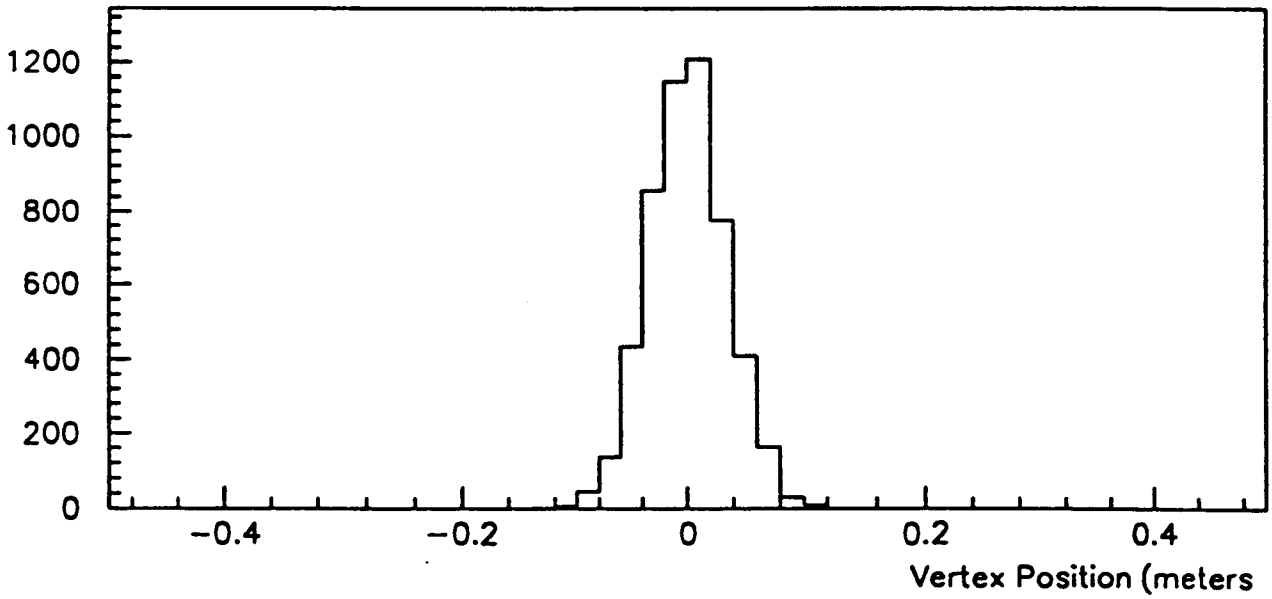
$|\eta| < 3.0$

1 mb = 1 MHz @ 10^{33}

Background Rates from QCD 2-jet events
with 20 GeV < Pt < 200 GeV
mixed with minimum bias events

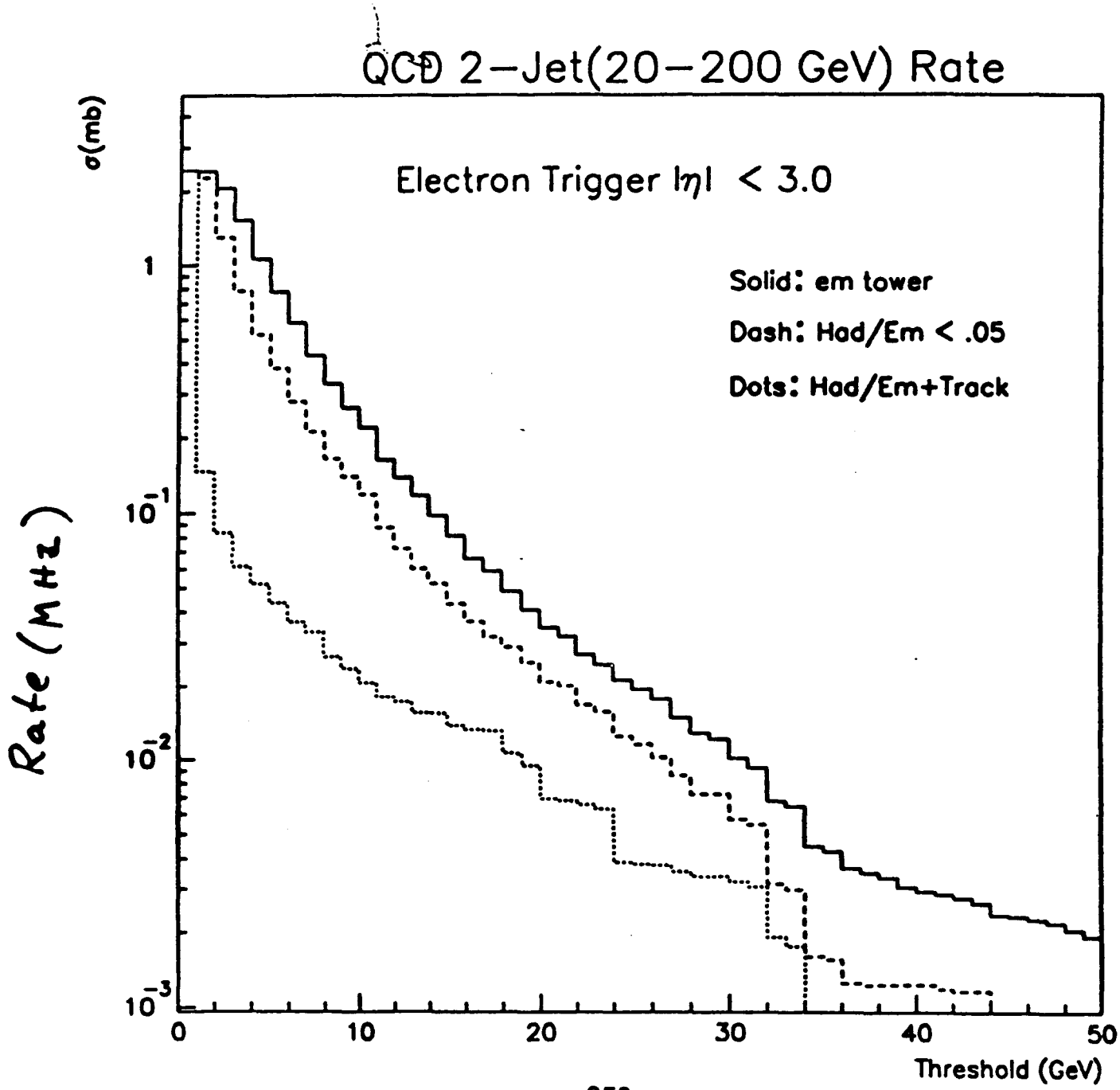

QCD 2-Jet
dN/d(1 GeV)
20 – 50 GeV
σ ~ 2 mb
50 – 100 GeV
σ ~ 0.1 mb
100 – 200 GeV
σ ~ 0.01 mb
E_t (GeV)
highest em tower (h/em < 0.2)


Min. bias
⟨n⟩ = 1.6
dN/dn
0.32
0.28
0.24
0.2
0.16
0.12
0.08
0.04
0
0
1
2
3
4
5
6
7
8
9
10
n (events/crossing)

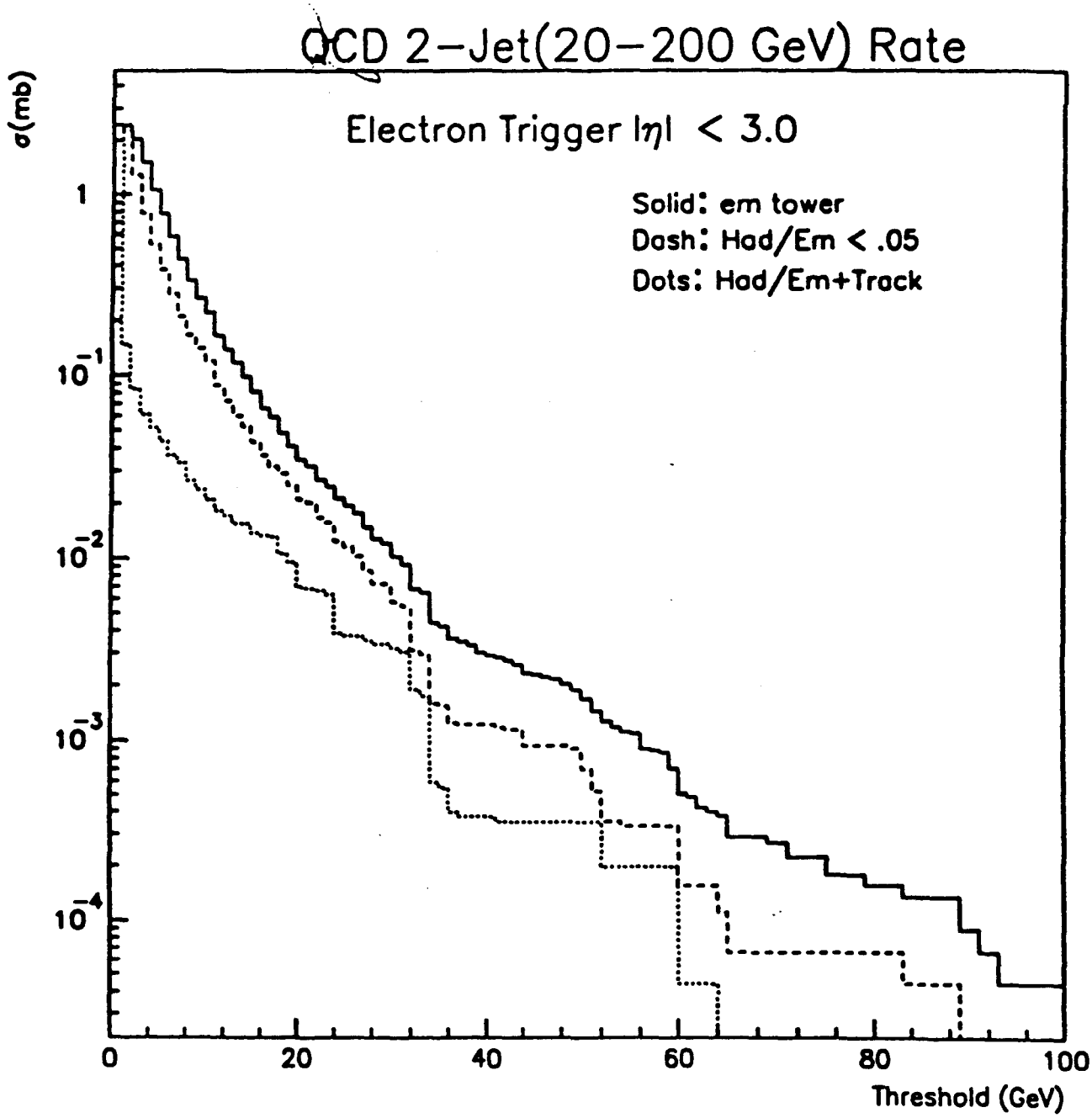

1200
1000
800
600
400
200
0
−0.4
−0.2
0
0.2
0.4
Vertex Position (meters

L1 Single electron trigger rate for $|\eta| < 3.0$

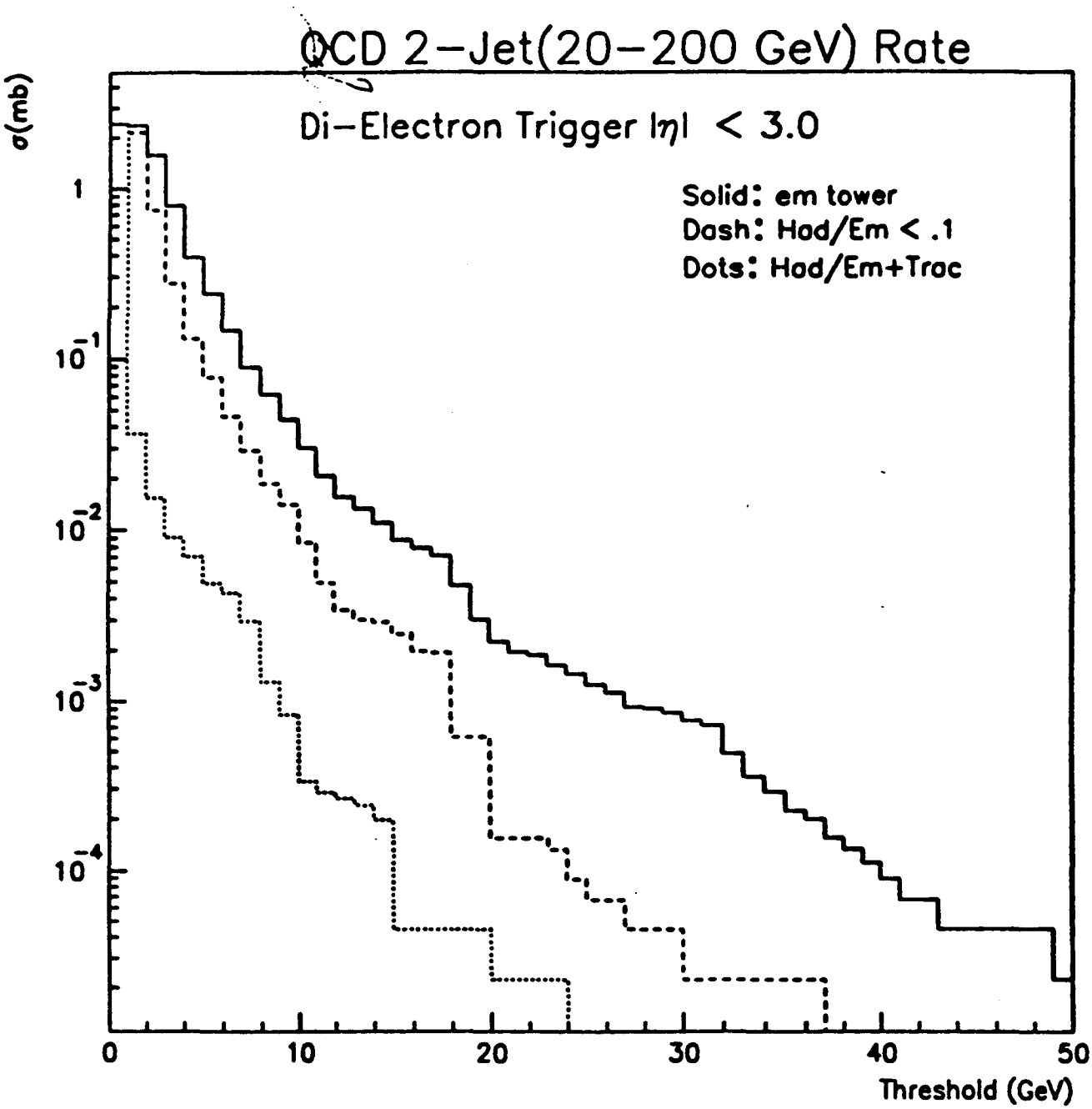
Track ≡ Track $P_T > 10.0$ matched in ϕ to tower.



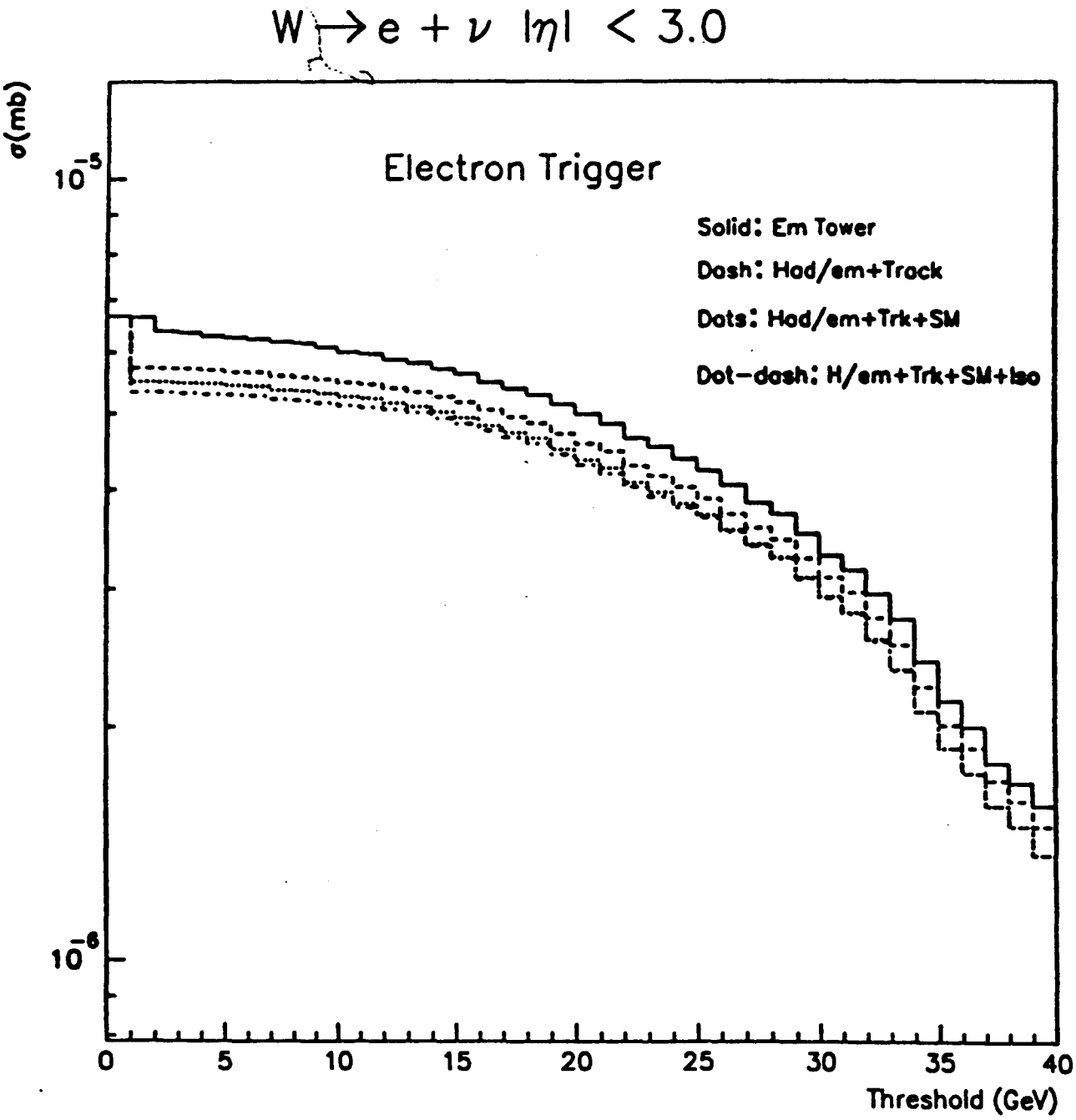
Same as before Different
Energy Axis scale



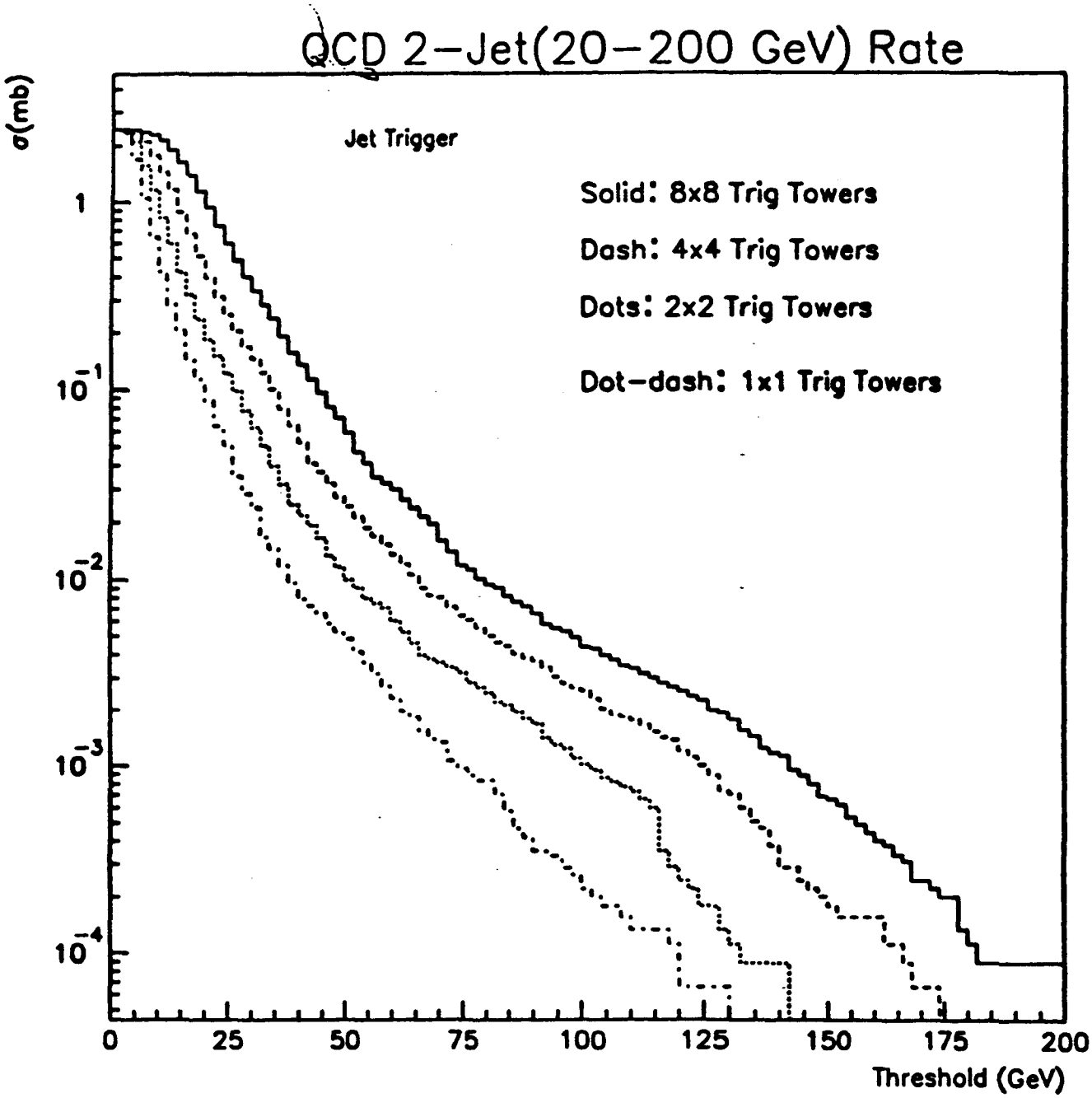
L1 di-photon and di-electron rates
Rate (MHz) for
2 towers over threshold.



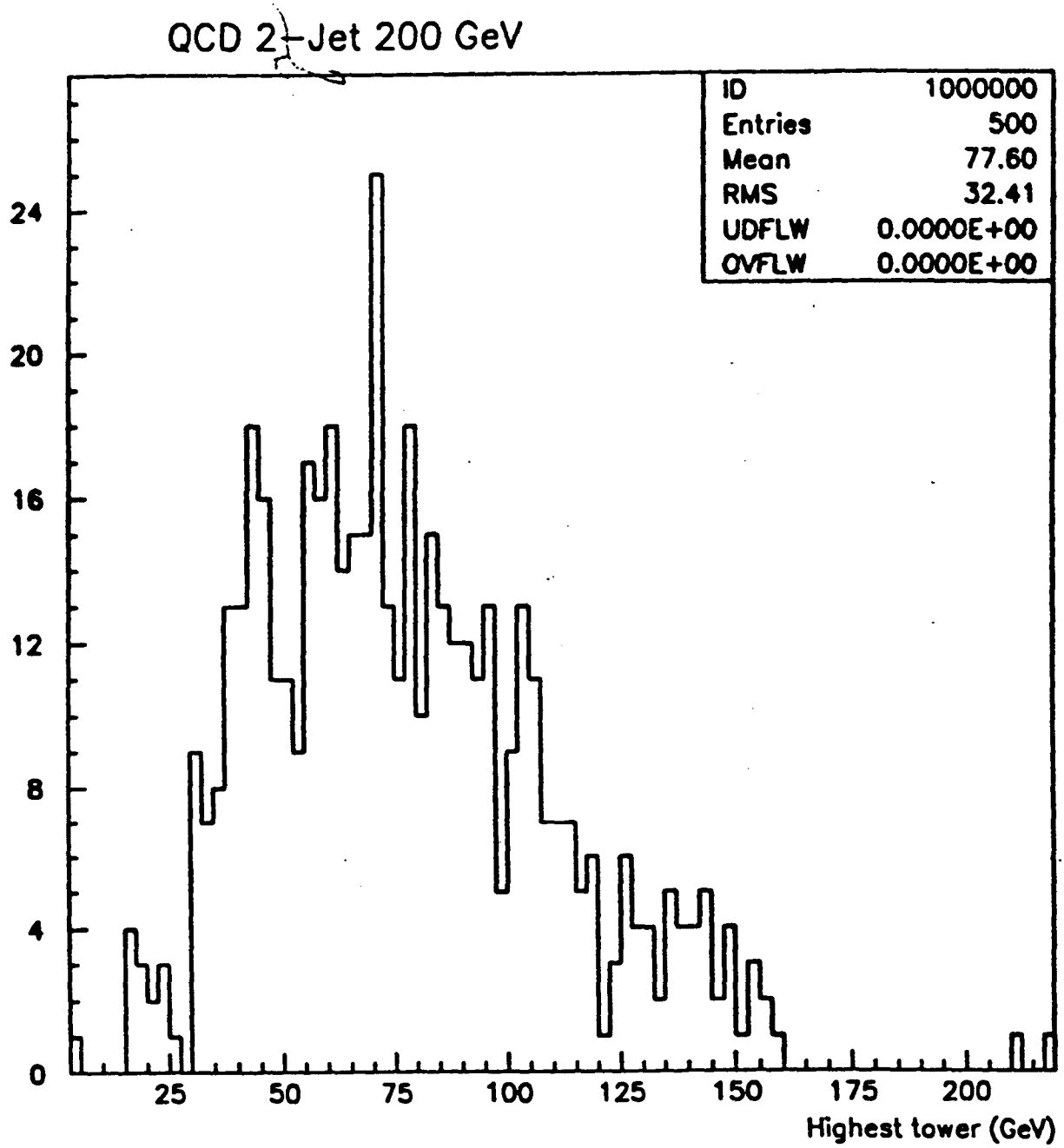
$W \to e + \nu$ with $|\eta_{ela}| < 3.0$



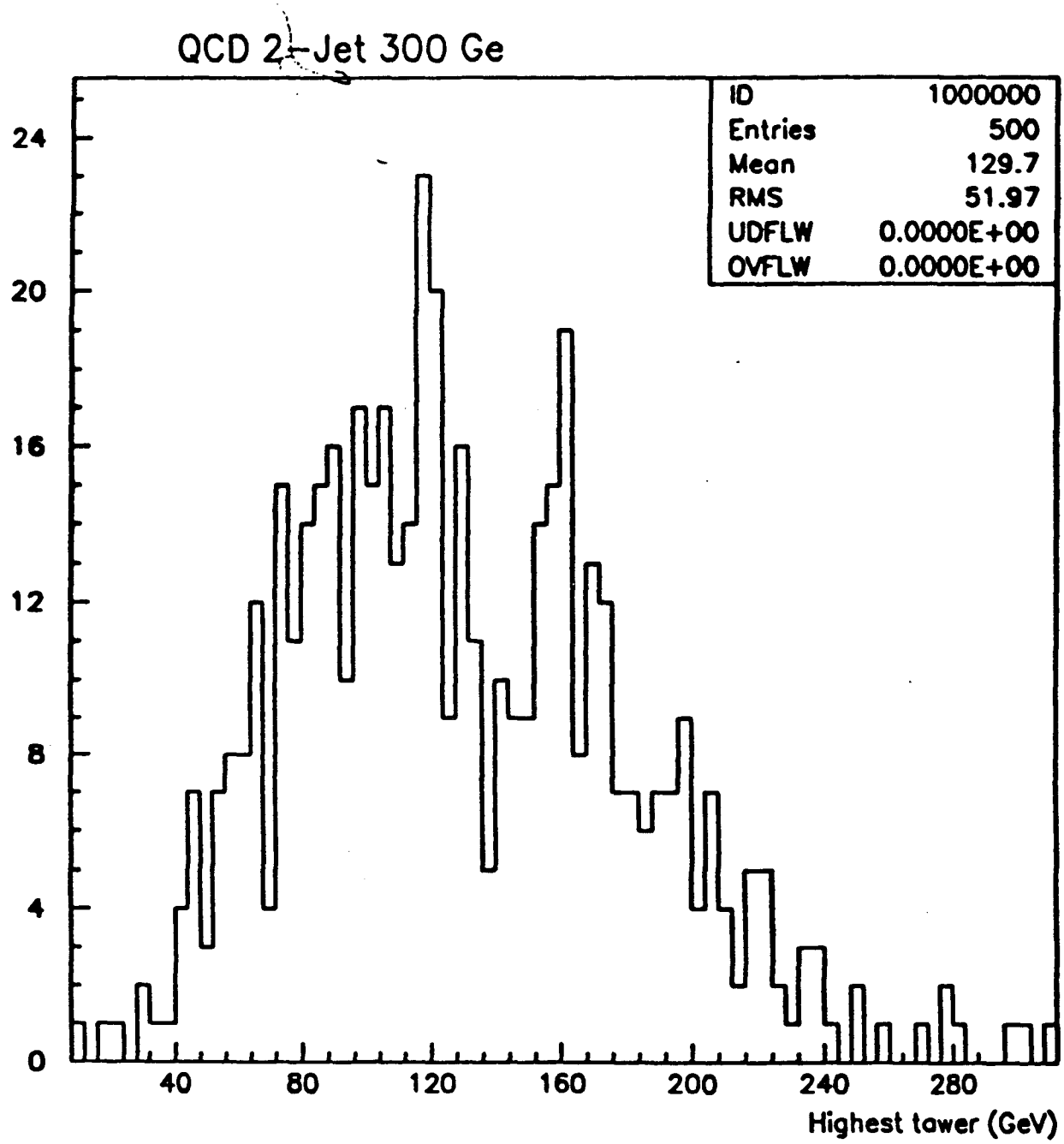
Rate vs. Highest Energy tower.



Distribution of Energy for highest energy tower from QCD 2-Jets at 200 GeV.



Same for 300 GeV 2-Jet.



Combined EM Trigger Rates

e = electron ; Had/Em < 0.05 ,
Track Pt > 10 GeV matched
in ϕ with tower

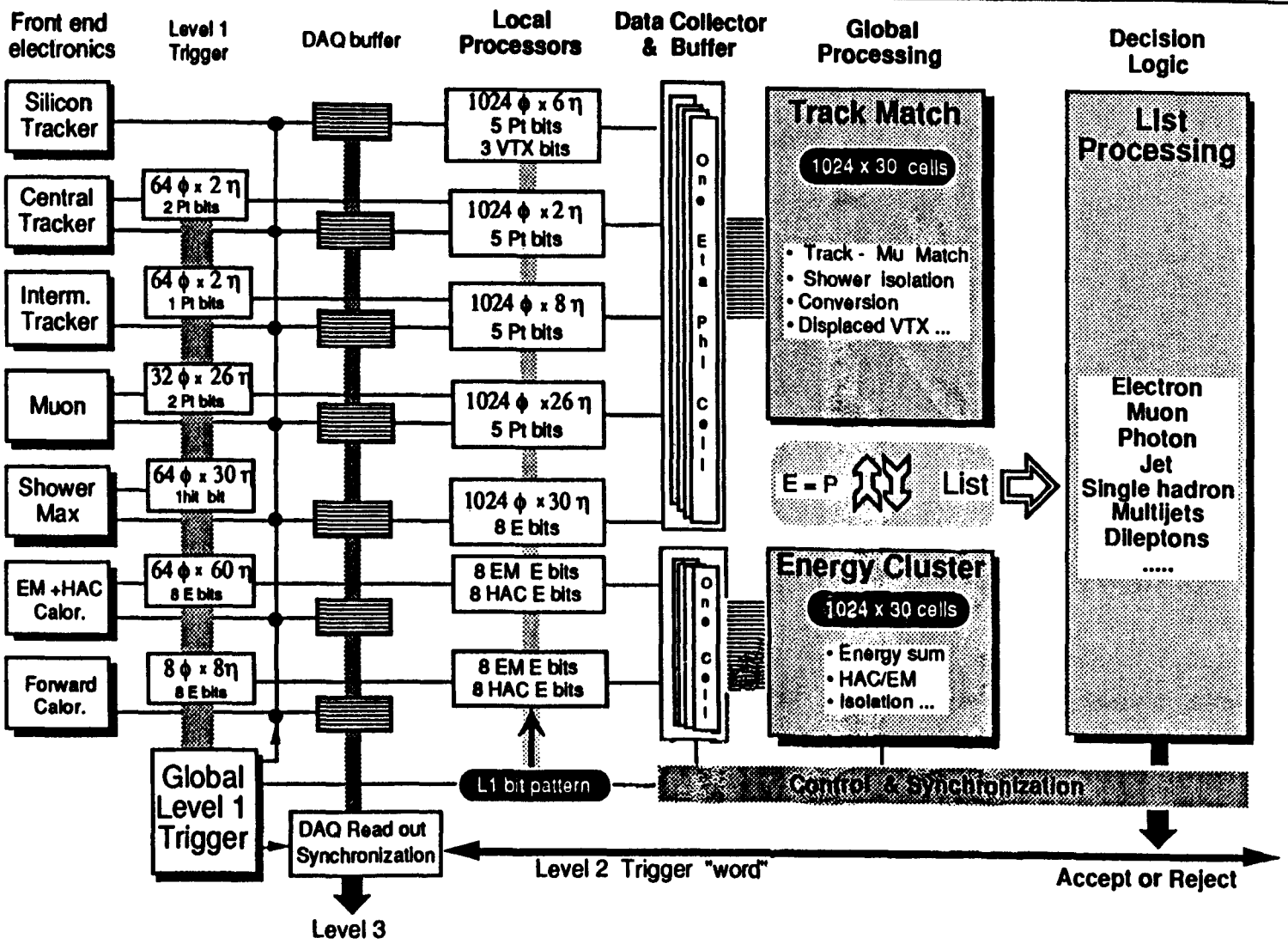
2e = dielectron ; Had/Em < 0.1 ,
Track Pt > 10 GeV

γ = photon; Had/Em < 0.05

2γ = diphoton; Had/Em < 0.05

Trigger Threshold (GeV)				Rate (kHz)
e	$2e$	γ	2γ	$@10^{33} cm^{-2} S^{-1}$
20	10	30	20	9.8
20	10	40	20	8.1
25	10	40	20	5.0
25	15	40	30	4.7
30	20	45	30	3.8
20	–	–	–	7.0
–	10	–	–	0.3

Data Flow (TDR)



SDC level 2 trigger -

3/25/92

Preliminary
SDC Level 2
Trigger Rtaes

$|\eta| < 3.0$

1 mb = 1 MHz @ 10^{33}

Background Rates from QCD 2-jet events
with 20 GeV < Pt < 200 GeV
mixed with minimum bias events

<u>NOTE:</u>

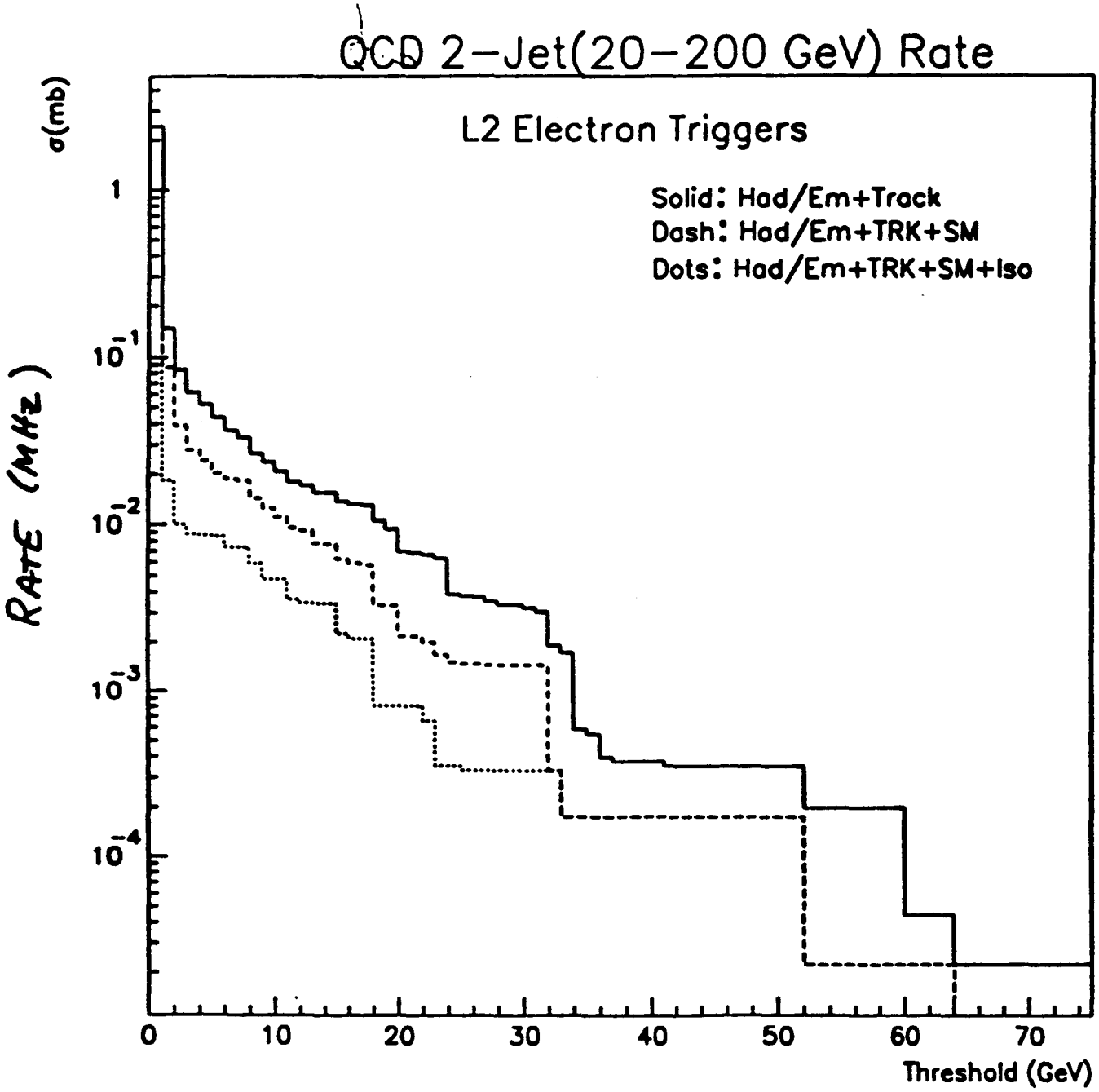
No Level 2 clustering Done - single towers only

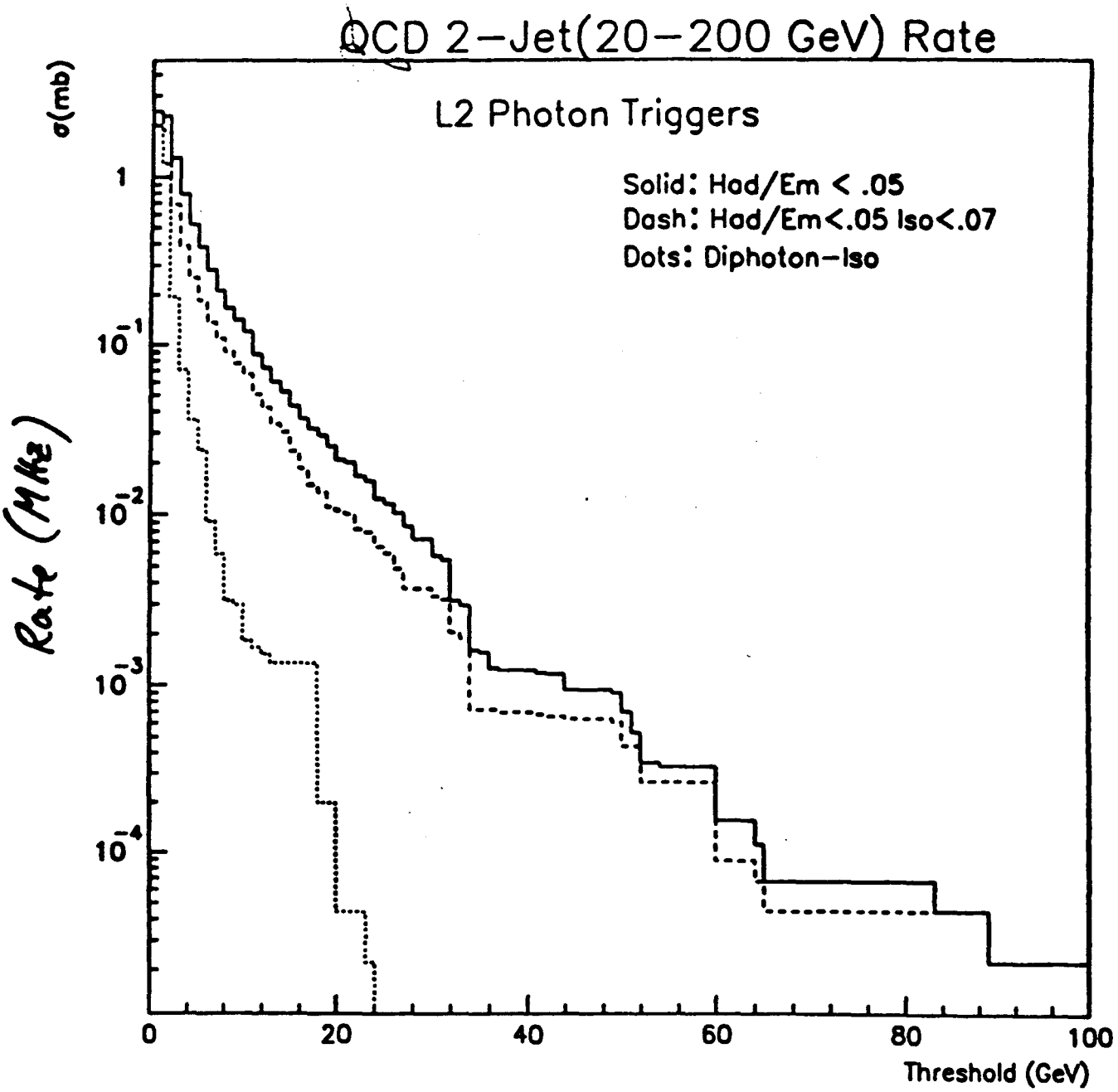
No Conversions removed - The level 1 rate is dominated by charged pion-neutral pion overlap and conversion electrons in approx. equal proportions.

TRK ≡ Track P_T >10.0 GeV matched in ϕ with tower

SM ≡ Energy Normalized Strip over threshold matches in ϕ with track and η with tower

ISO ≡ 8 Surrounding HAC towers/Energy < .07

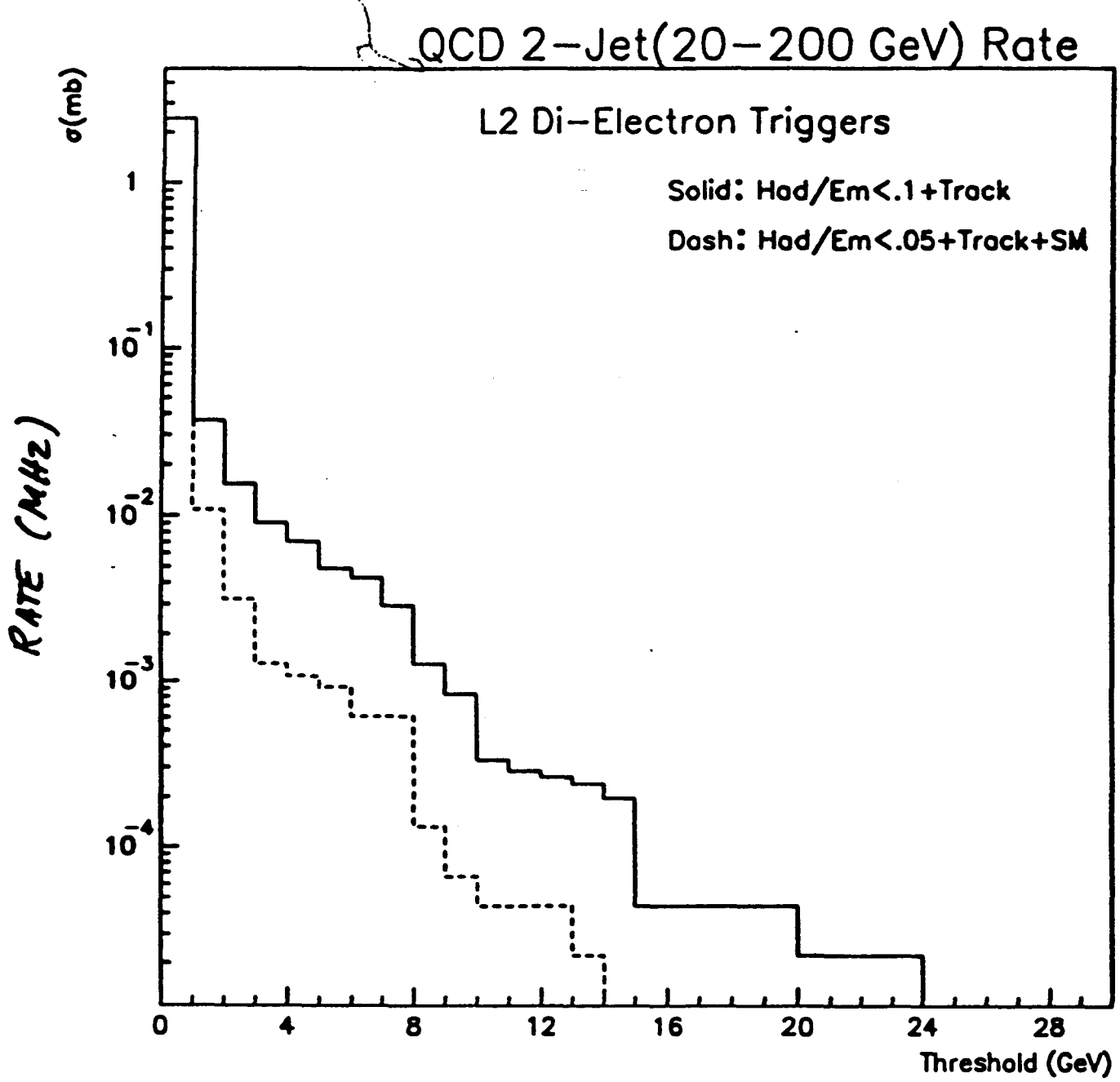



QCD 2-Jet(20-200 GeV) Rate
L2 Photon Triggers
Solid: Had/Em < .05
Dash: Had/Em<.05 Iso<.07
Dots: Diphoton-Iso
σ(mb)
Rate (MHz)
1
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
20
40
60
80
100
Threshold (GeV)

Rate for two towers over threshold

TRACK ≡ Track P_T > 10.0 GeV matched in ϕ with tower

SM ≡ normalized response above threshold matched in ϕ to track and η with tower.



Conclusions:

LEVEL 1

1) Level 1 calorimter trigger rate on the order of 15 kHz looks alright.
2) Level 1 Muon Rate of 15 kHz needs more work!

LEVEL 2

1) Preliminary rates look ok for dielectron and diphoton rates.
2) Shower max -track match removes overlaps.
3) Need work on reduction of rates using silicon to tag conversions.
4) Clustering algorithms - sharpen thresholds.

5) Need to demonstrate a reasonable L2 rate. Set specs for DAQ bandwidth.

LEVEL 3 / DAQ

1) DAQ bandwidths necessary/achievable?

2) Can we get within a factor of 2 of the physics rates?

CACI Products Company

Planned Enhancements to MODSIM II

&

SIMOBJECT - an Overview

Ron Belanger
ErgoSoft
7122 Cather Court
San Diego, CA 92122
(619) 450-1108 fax (619) 450-1677

MODSIM II

- Modular, Object-Oriented language for writing discrete event simulation models
- Models written in traditional way - Pascal-like code

SIMOBJECT

- Graphical, object-oriented programming environment for simulation
- More than a CASE tool
- More than a code generator

MODSIM II - Future Enhancements

- Persistent Objects
- Improved SIMLAB
- More Statistical Distributions
- Data Allocation Monitor
- Performance improvements
- User choice of discipline for Pending List
- Improved Documentation

MODSIM II - Future Enhancements (cont)

Persistent Objects

- Provide ability to preserve instances of objects, records and arrays across runs of a program or for use by other programs at a later time.
- Persistent objects are preserved in a data base.
- Nearly "transparent" to the user.

MODSIM II - Future Enhancements (cont)

Improved SIMLAB

- SIMLAB is a Graphical User Interface to tools comprising MODSIM II
- Compilation manager for MODSIM II
- MODSIM Module Browser
- Portable across GUIs on Unix workstations, Windows, OS/2
- Takes on look and feel of local GUI environment

MODSIM II - Future Enhancements (cont)

Data Allocation Monitor

- Audits NEW and DISPOSE of any data type
- Helps to detect and track memory leaks
- Optional compilation switch - no overhead unless needed for debugging

MODSIM II - Future Enhancements (cont)

Pending List Discipline

- Pending List (event list) now linear linked list
- User will soon have choice of linear or b-tree
- User can optionally leave choice of discipline to system which will adjust dynamically within high and low threshold parameters

SIMOBJECT

- Graphical, object-oriented programming environment
- Currently under development
- Structure of program maintained graphically using graphic meta-editor
- Program exists in a database, which can be queried about program
- Code is encapsulated in graphical objects

SIMOBJECT (cont)

- System generates code for standard constructs and simulation objects which are inserted or inherited from other objects.
- Generated and user-written code can be accessed, added to or embellished.
- Program can be viewed at several levels of abstraction.
- Highest level of abstraction can map to standard formal design methodologies.

SIMOBJECT (cont)

- Programming and simulation capabilities similar to MODSIM II
- Changes to inheritance structure made graphically
- Need to learn & remember syntax reduced. Programmer can concentrate on programming
- Inheritance specified by dropping one object type into another
- NETOBJECT - simulation tool kit and environment built on top of SIMOBJECT

MODSIM II Enhancements Under Consideration

- Optional garbage collection
- Two pass compiler. Would eliminate need for EXPORTTYPE and FORWARD statements. Further code optimization would be possible.
- Interpreted development environment. Would provide further debugging capabilities.

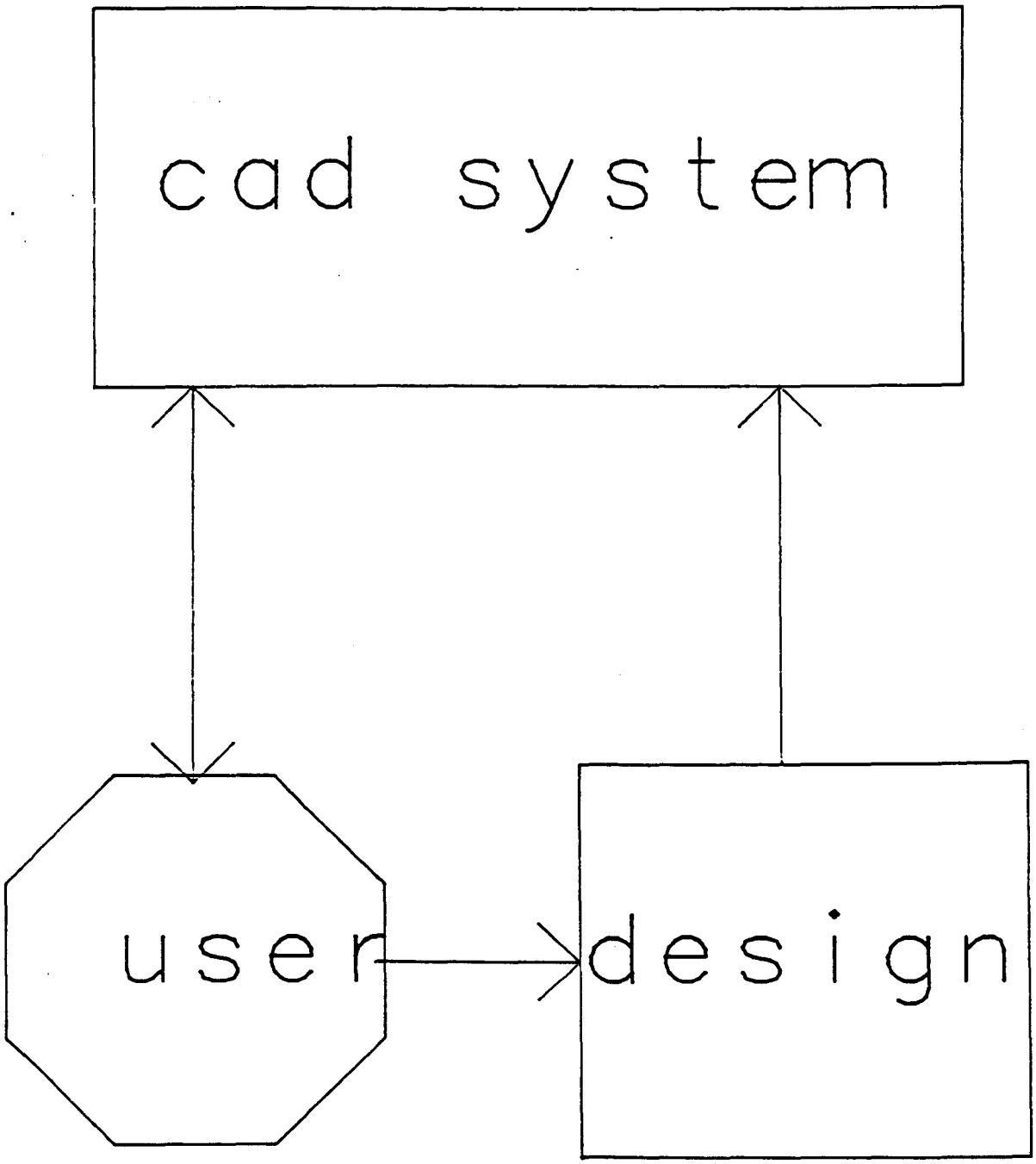
DAGAR

- A Synthesis System

by

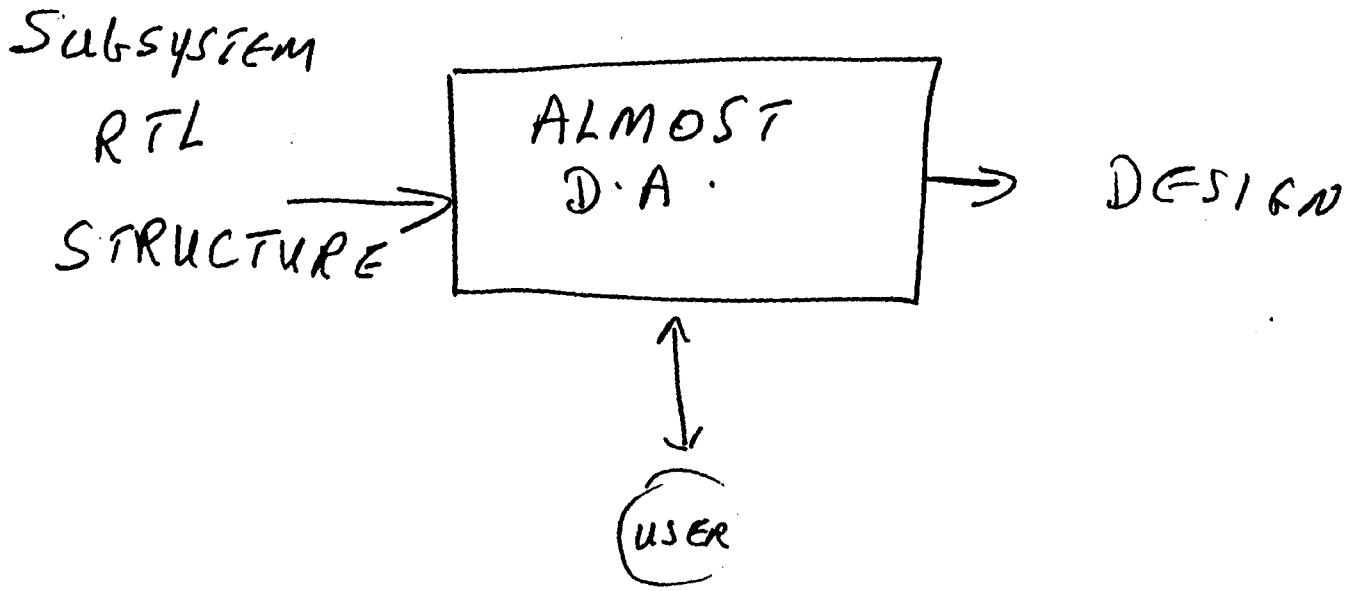
V. K. RAJ
U. T. Arlington.
raj@cse.uta.edu

c a d

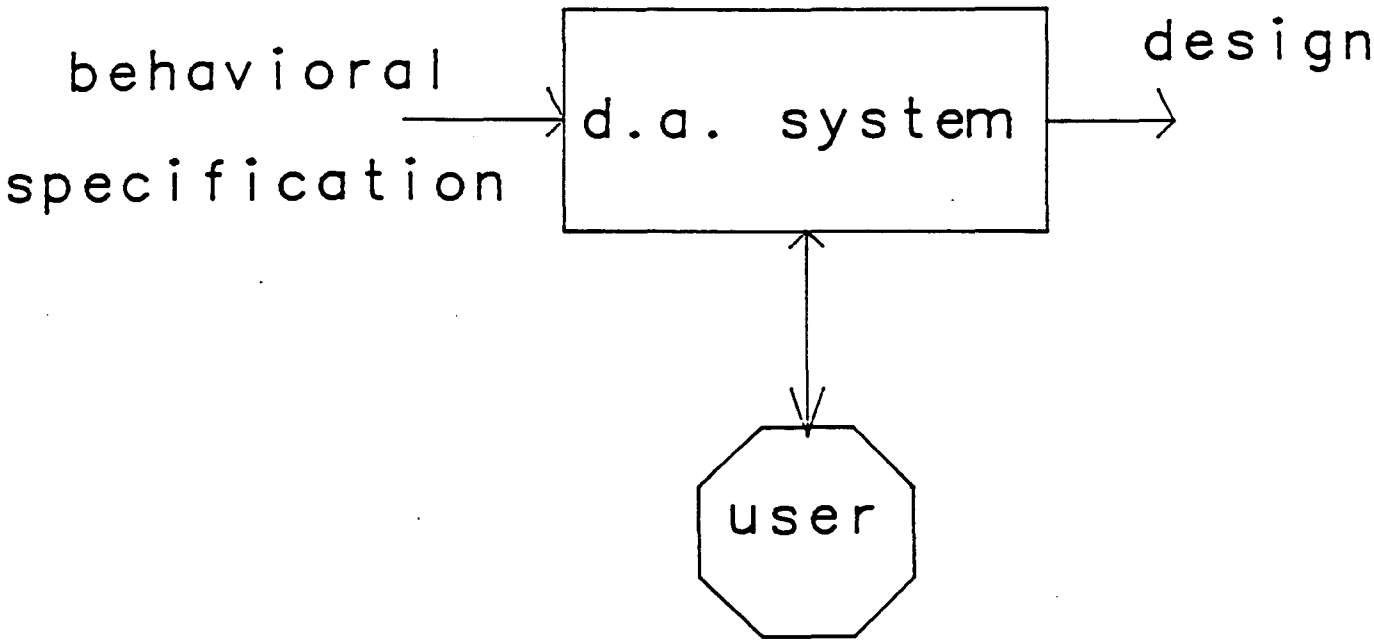


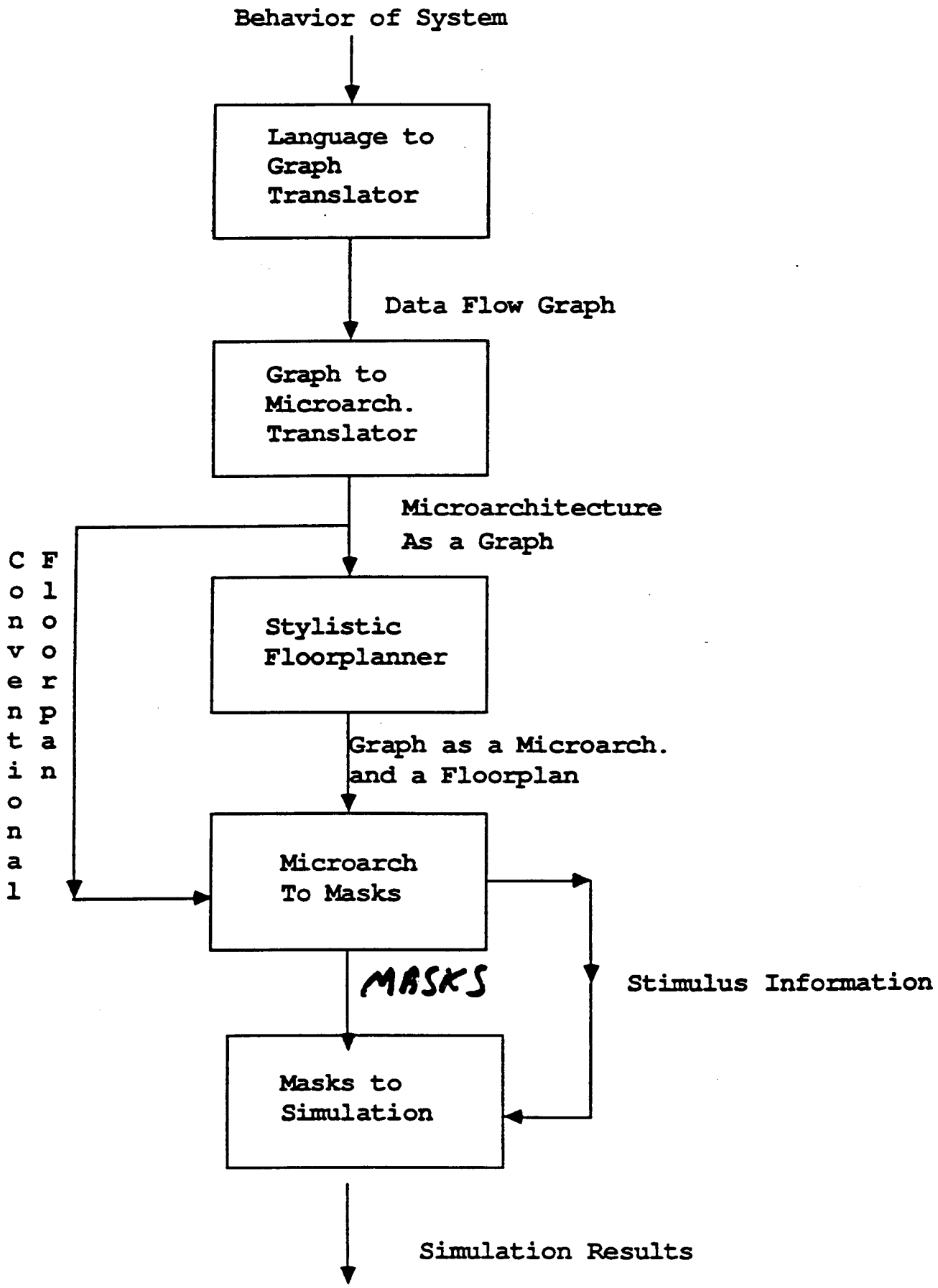
user does design
cad system checks it

Synthesis Today



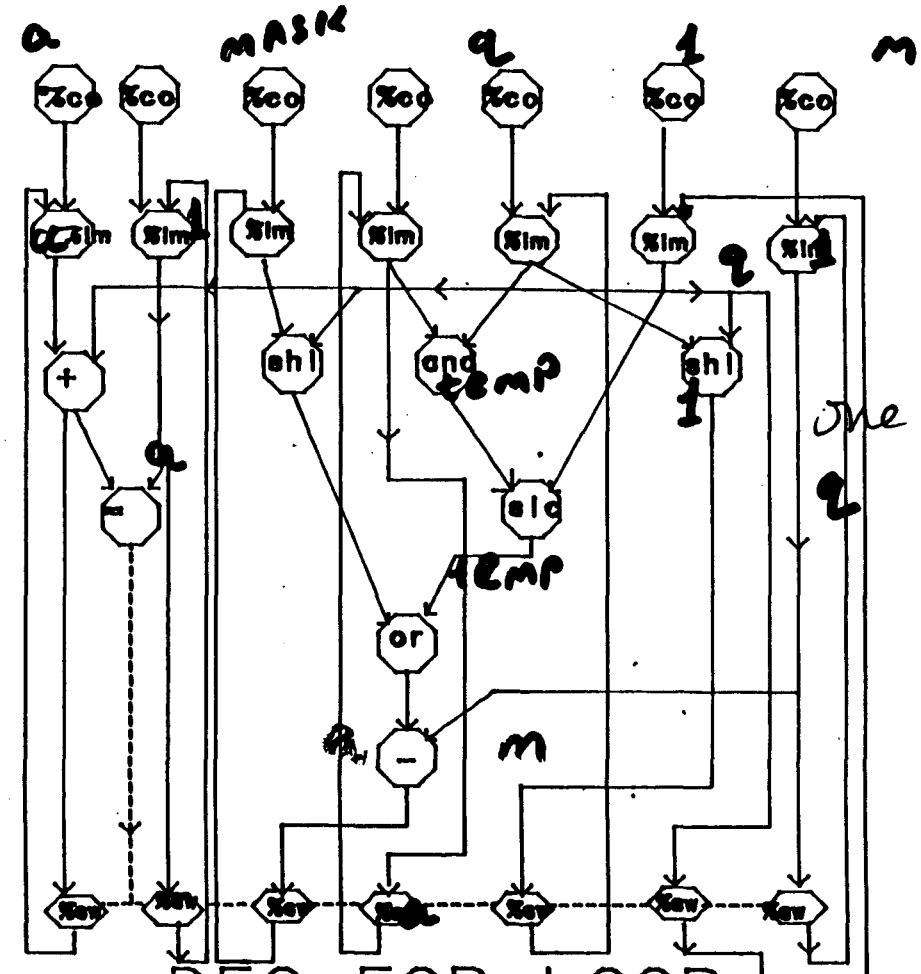
DESIGN AUTOMATION




Behavior of System
Language to Graph Translator
Data Flow Graph
Graph to Microarch. Translator
Microarchitecture As a Graph
Conventional Floorplan
Stylistic Floorplanner
Graph as a Microarch. and a Floorplan
Microarch To Masks
MASKS
Stimulus Information
Masks to Simulation
Simulation Results

```
const n = 16;
      mask = '100000'b;

var       q,
      a,
      m,
      count,
      temp : bit (n);

begin
      in      q, m;
      a     = 0;
      count = 0;

      loop
            a     = a shl 1;
            temp  = mask and q;
            temp  = temp slc 1;
            a     = a or temp;
            a     = a - m;
            q     = q shl 1;
            count = count + 1;
            until count = n;
      endloop;
end.
```

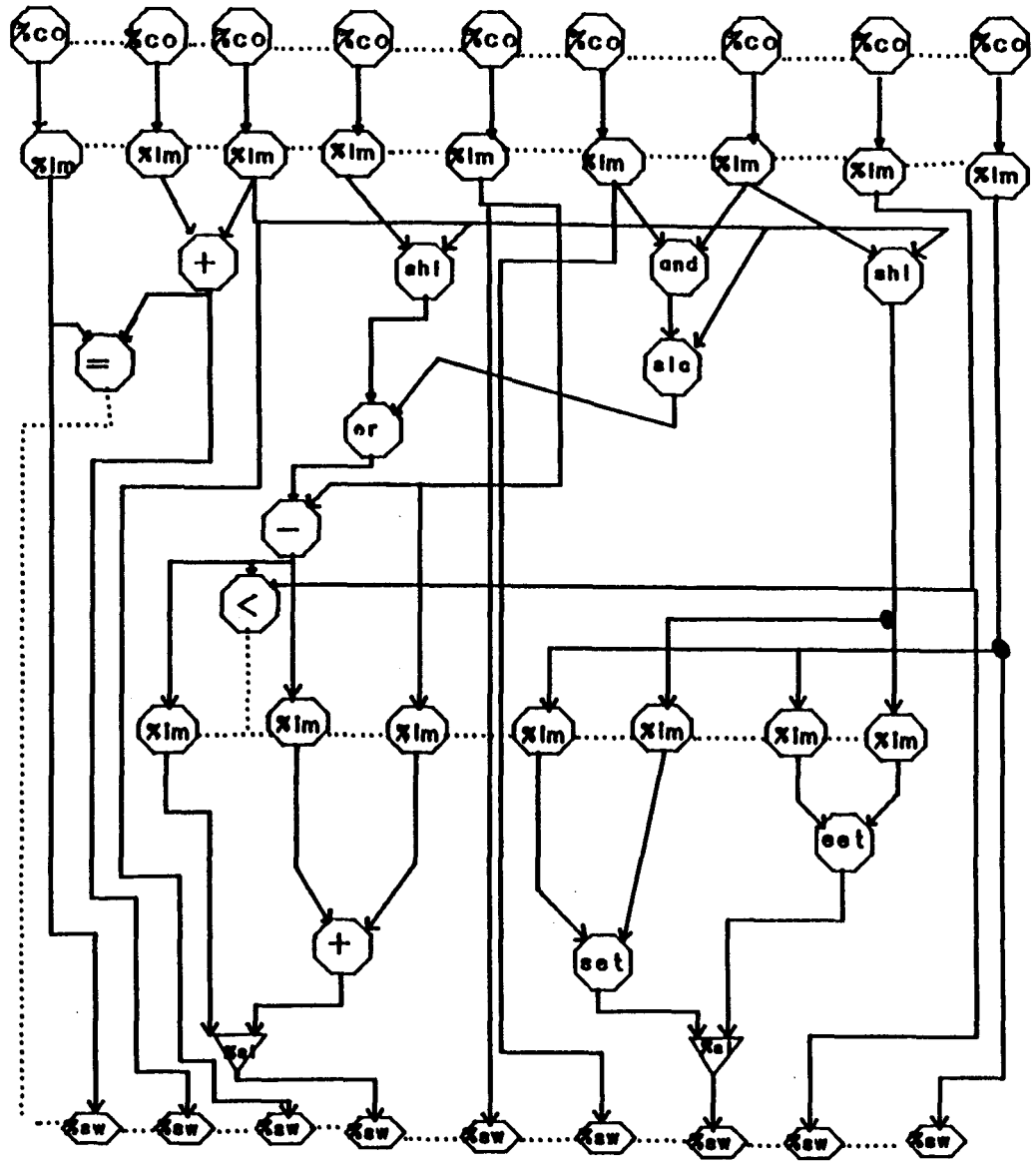
behaviour specification for loop.



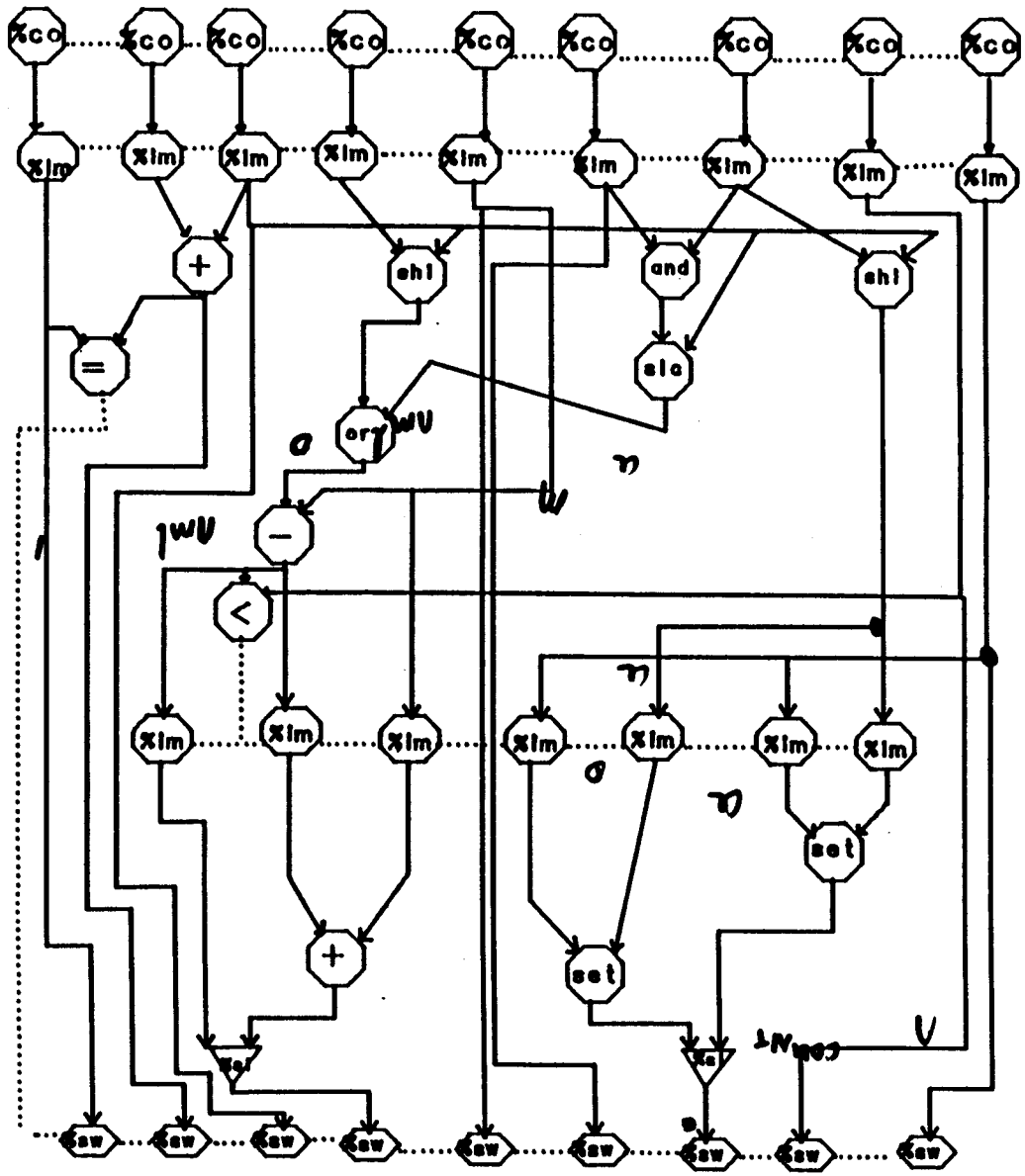
DFG FOR LOOP

one slide

```
procedure int rest_div

const   n = 16;                 /* define constants for program */
        nml = n-1;
        mask = '100000'b;

type    int  = bit(n);          /* constants for arsenic system */

int     q,                      /* quotient */
        a,                      /* accumulator */
        m,                      /* mantissa */
        temp;

begin
        q= 100;                 /* assign random values to q and m */
        m = 2 ;
        a  = 0;

        repeat
          temp = mask and q;
          temp = temp shc 1     /* rotate by 1 using operation shc */
          a    = a shl 1;       /* shift variable a left by 1 */
          a    = a or temp;
          a    = a - m;
          q    = q shl 1;
          if ( a < 0 )
               {
               q(nml) set 0;  /* set the bit nml of q to zero */
               a = a + m;
               }
          else   q(nml) set  1; /* set the bit nml of q to one */
          count = count + 1;    /* repeat n times */
       until  count = n;
```

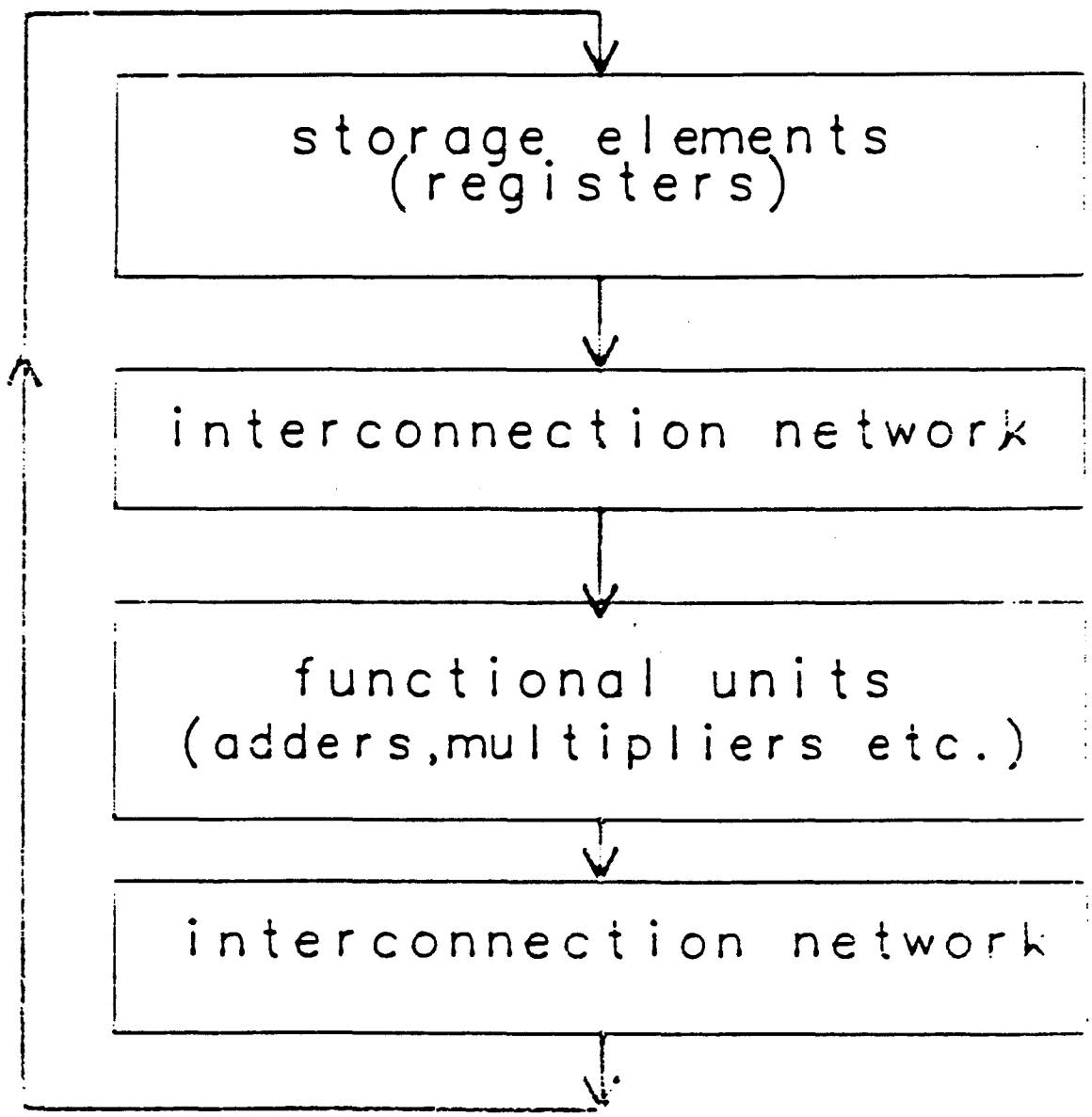


dfg for integer restoring division

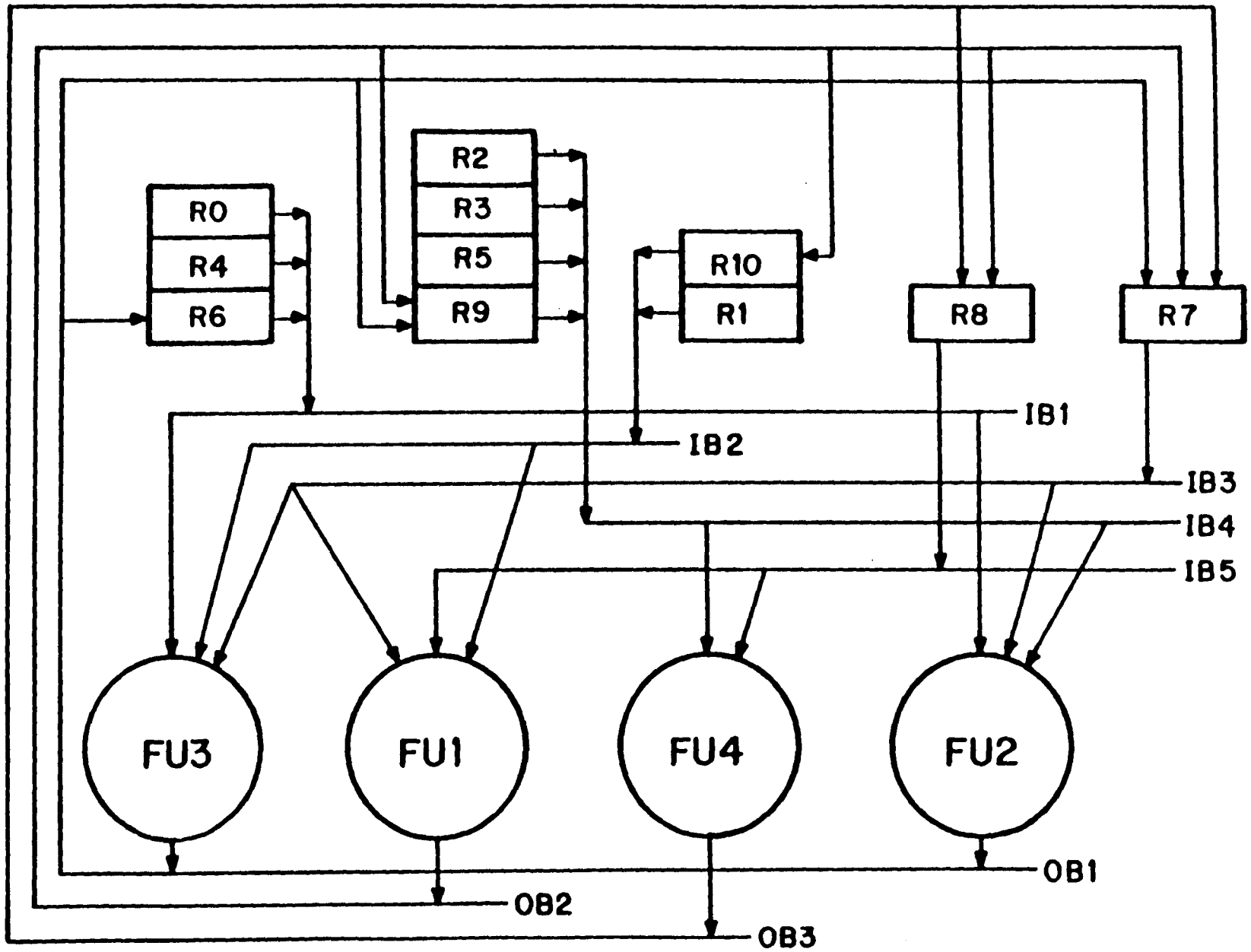


dfg for integer restoring division

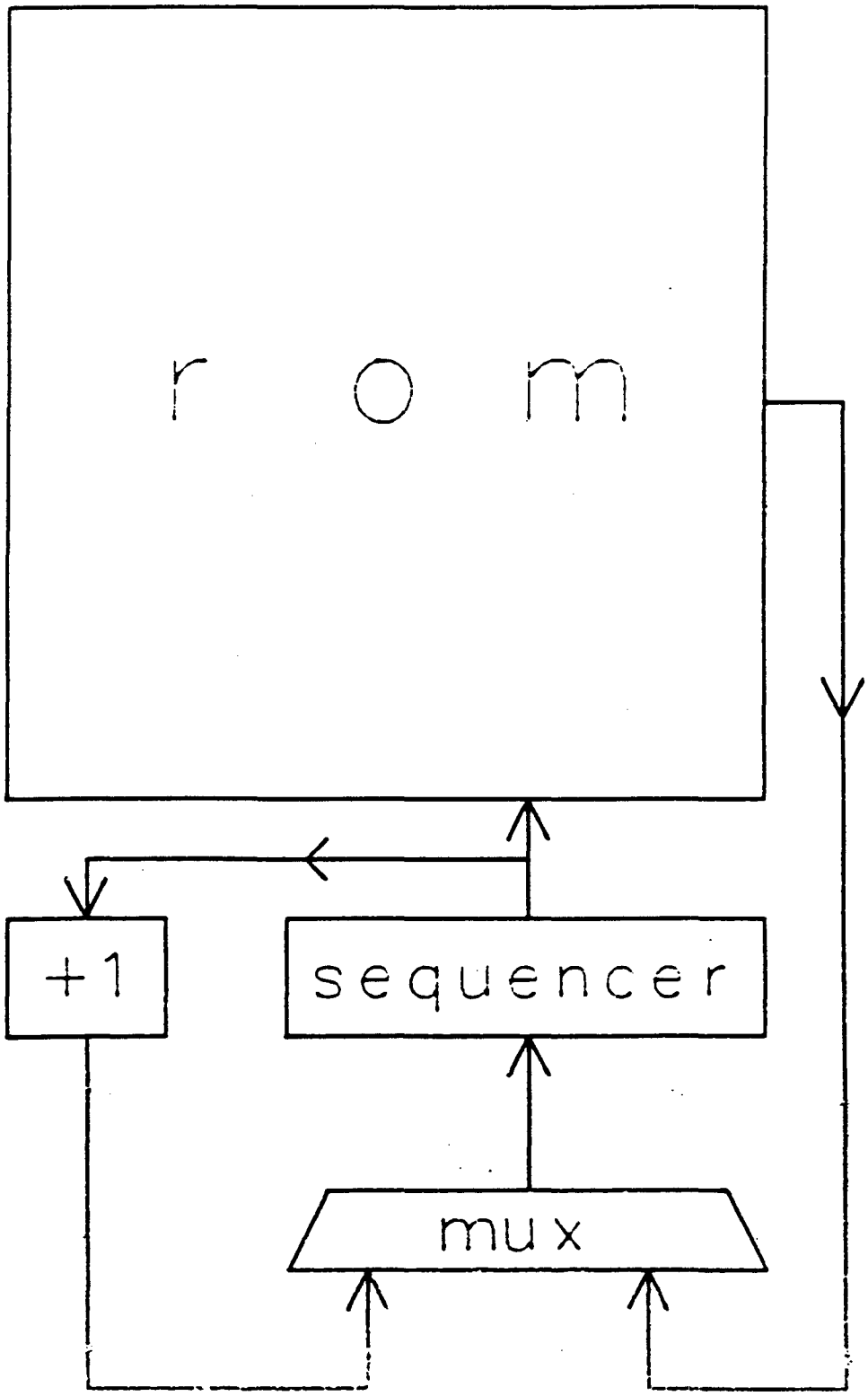
MODEL OF DATA PATH



DATA PATH FOR MI TABLE



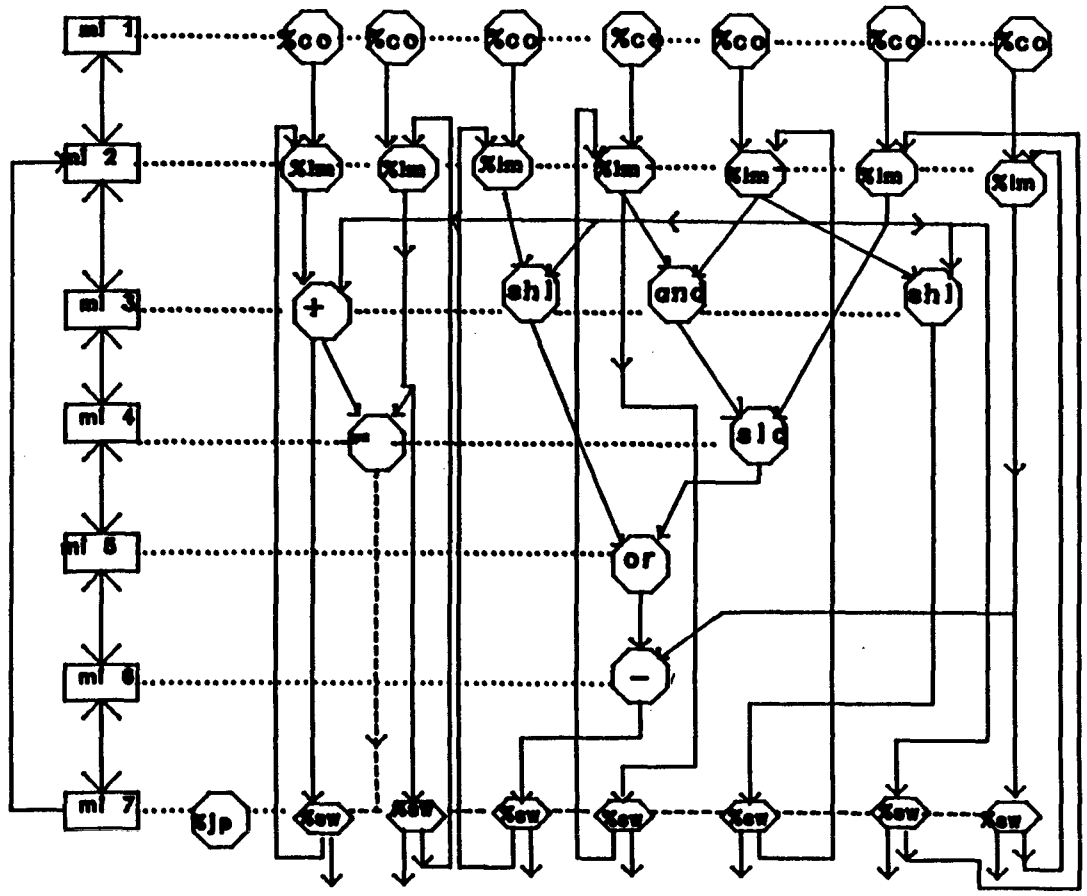
MODEL OF CONTROL UNIT



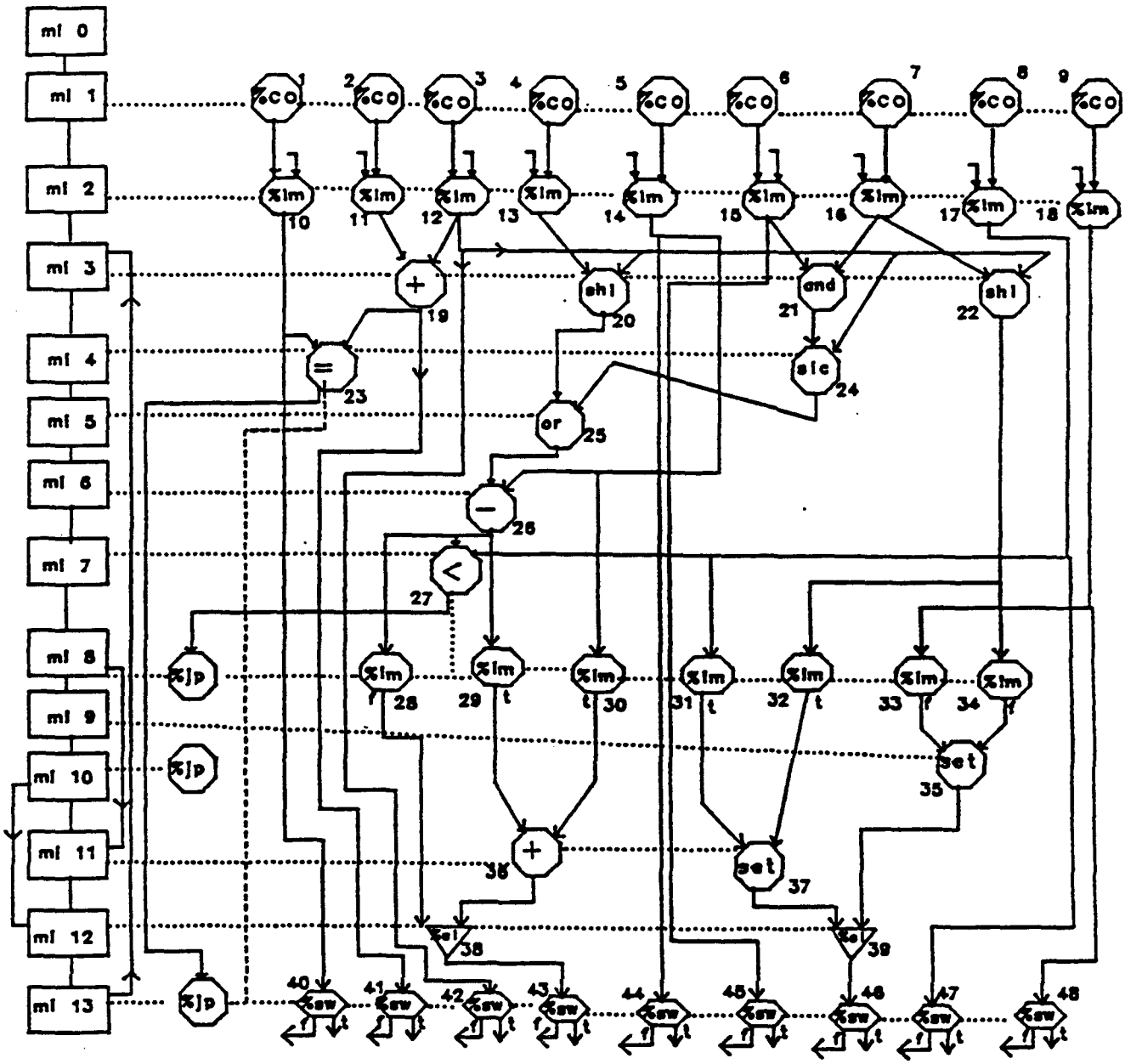
MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(and:r002,b00 r003,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
3		r012,B00 = FU01(=:r007,b00 r008,b01) r009,B02 = FU02(slc:r009,b02 r006,b03)
4		r009,B00 = FU01(or:r009,b00 r011,b01)
5		r005,B00 = FU01(-:r009,b00 r004,b01)
6		r009,B00 = FU01(<:r005,b00 r001,b01)
7	10	FU99($jp:r009,b07)
8		r002,B00 = FU01(set:r000,b00 r010,b01)
9	11	FU99($jp:)
10		r005,B07 = FU04(+:r004,b06 r005,b07) r002,B00 = FU01(set:r000,b00 r010,b01)
11	2	FU99($jp:r012,b07)
12		

CS of Division Algorithm

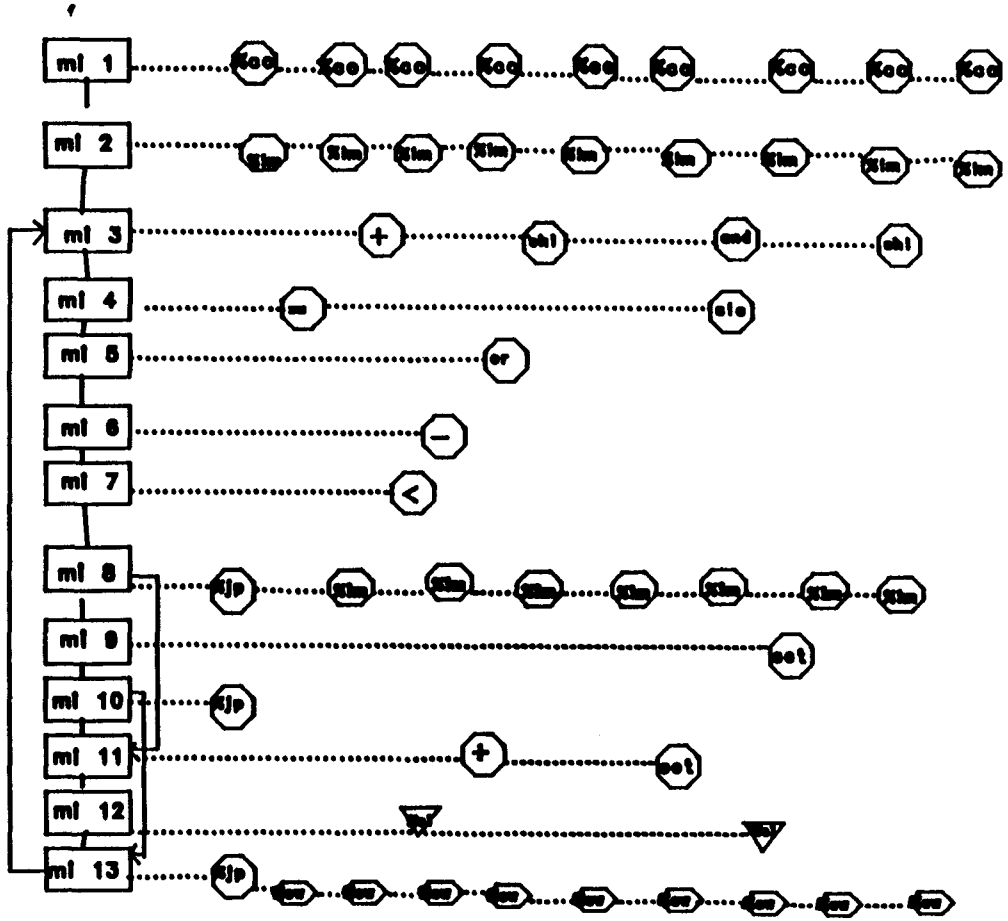
System : Single Clocked FUs
Number of FUs : 4
Execution Time : 125 ns * 12 clocks = 1500 ns



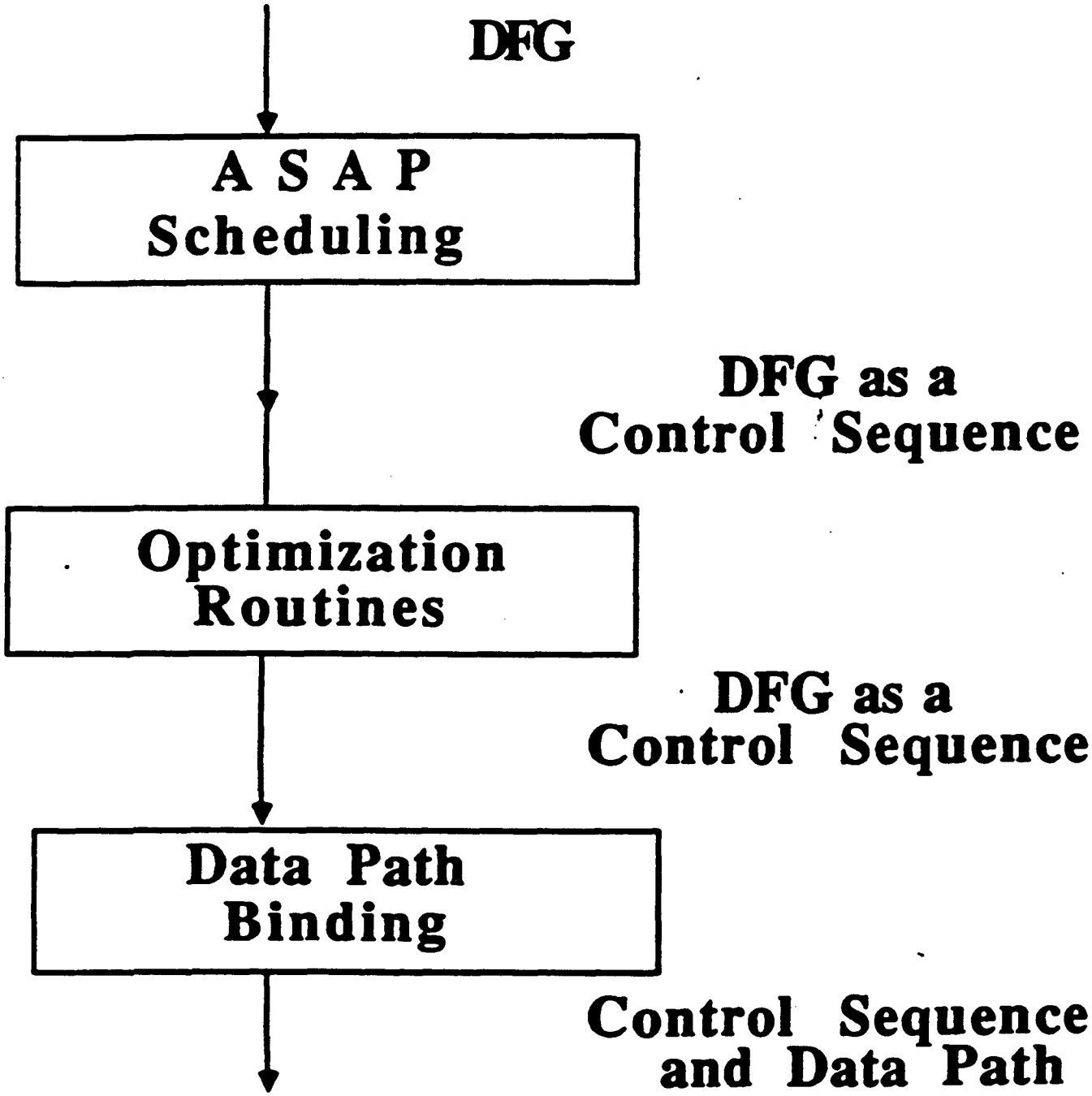
abstract machine for loop



Abstract Machine for Integer Restoring Division



abstract machine for division

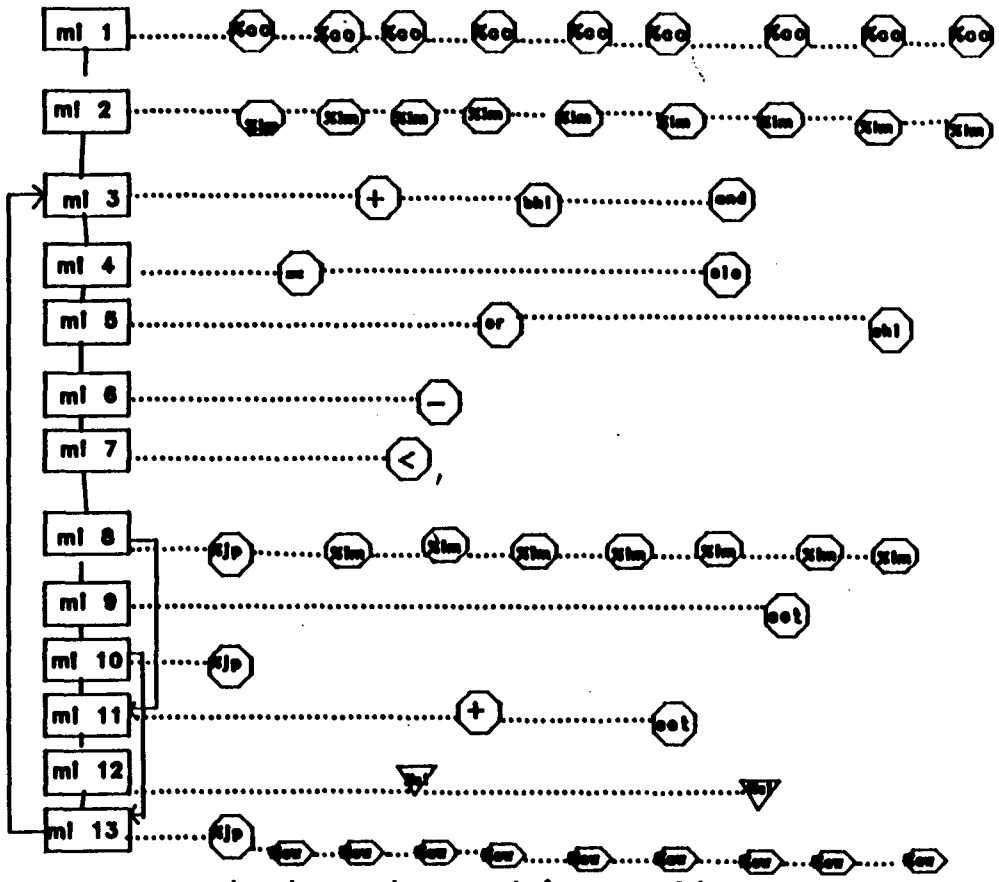

DFG
A S A P
Scheduling
DFG as a
Control Sequence
Optimization
Routines
DFG as a
Control Sequence
Data Path
Binding
Control Sequence
and Data Path

+	SHL	AND	=	SLC	OR	-	<	SET
1	2	1	1	1	1	1	1	1

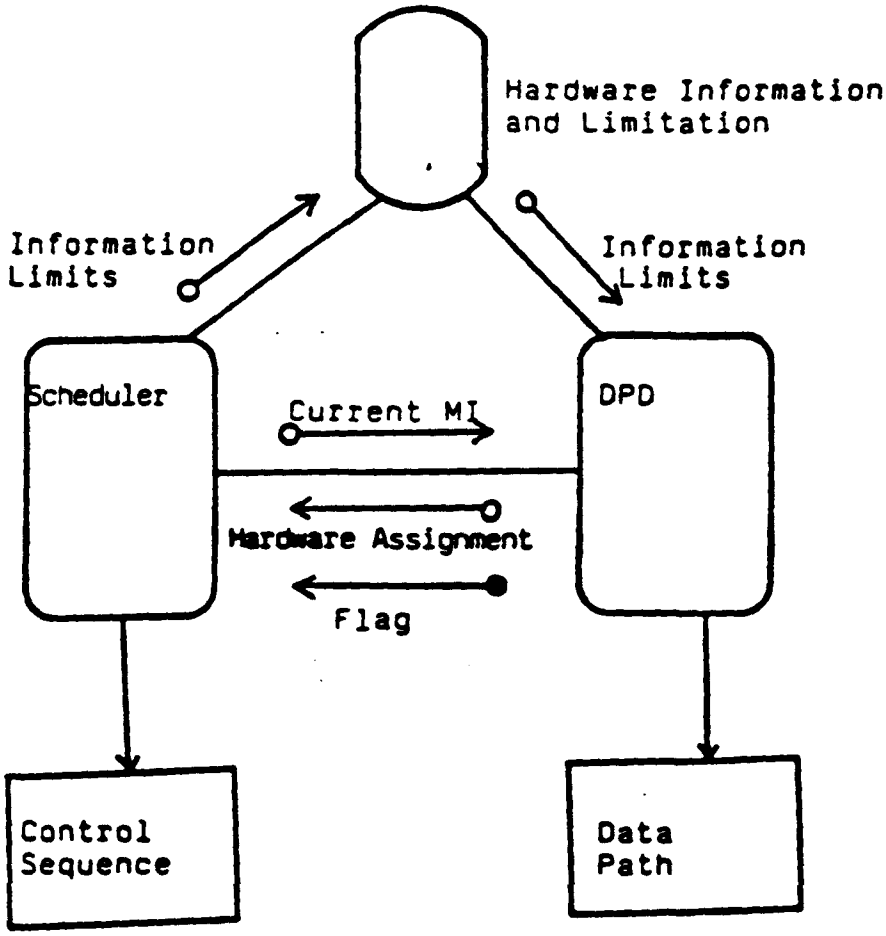
+
SHL
AND
=
SLC
OR
-
<
SET
abstract machine after
op-unit elimination

	+	SHL	AND	=	SLC	OR	-	<	SET
+		#	#						
SHL	#		#					#	
AND	#	#							
=		#			#				
SLC				#					
OR									
-									
<		#							
SET									

	SHL SLC	+ -	ALL OTHERS
SHL SLC		#	###
+ -	#		##
ALL OTHERS	###	##	

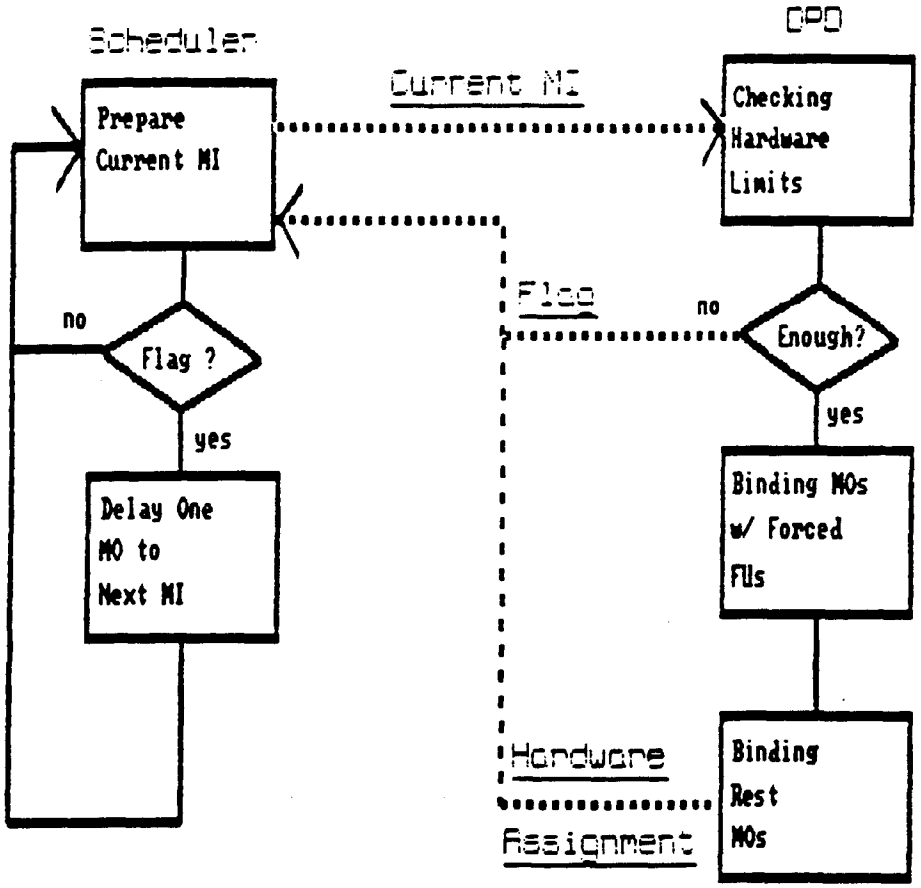


abstract machine after
op-unit merging

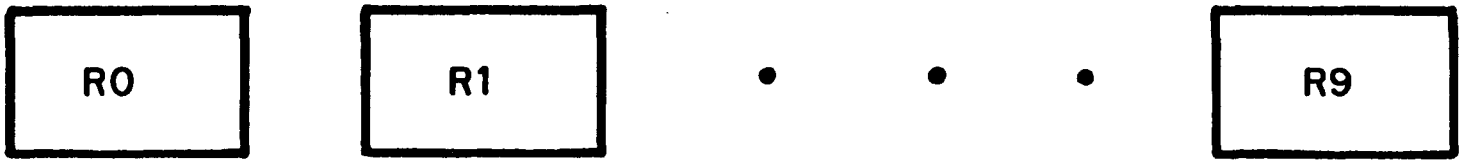


Model of DAGAR

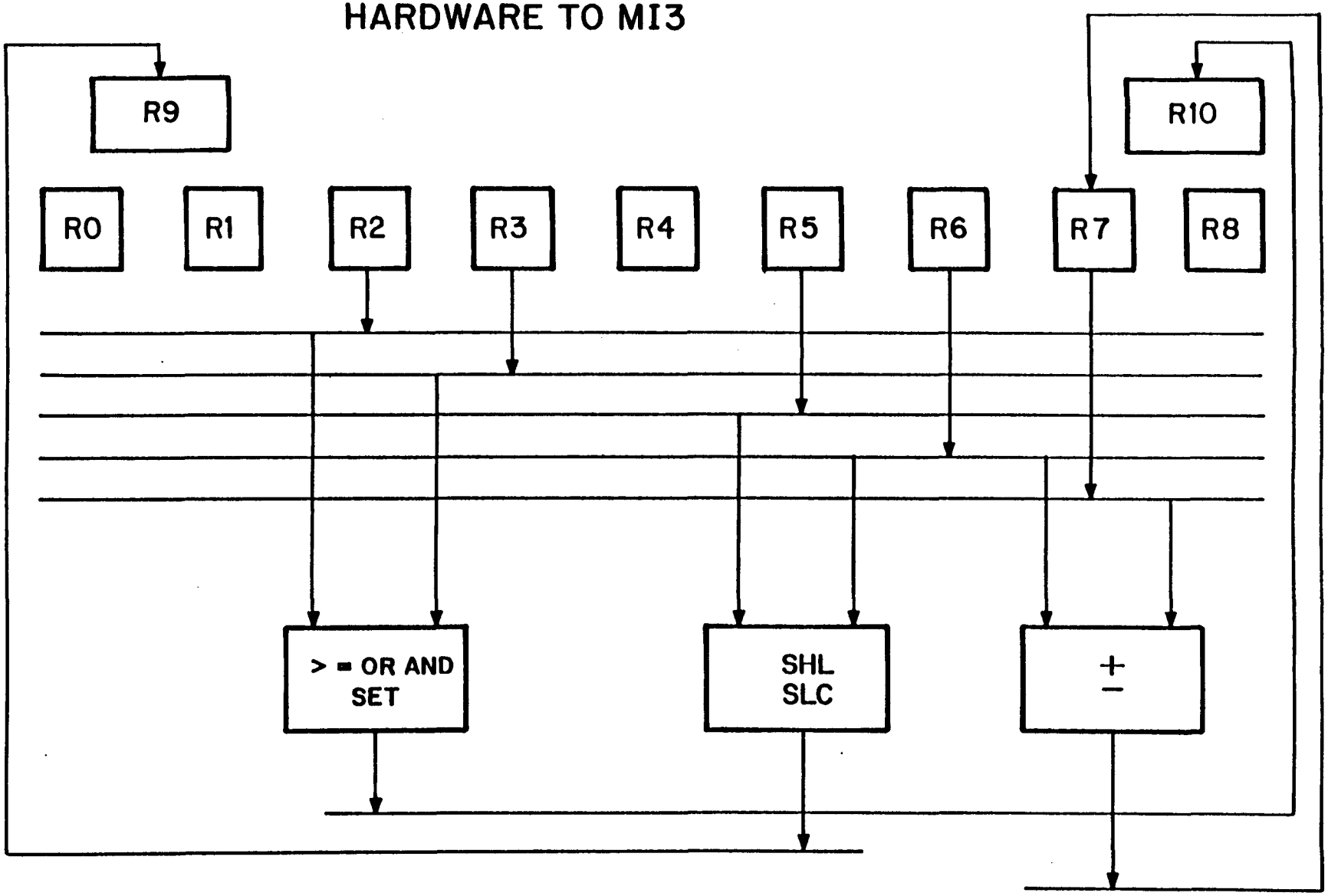
Model of Computation



DATA PATH AFTER ASSIGNING REGISTERS TO CONSTANTS



DATA PATH AFTER ASSIGNING HARDWARE TO MI3



8 MHz clock.

MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(and:r002,b00 r003,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
3		r012,B00 = FU01(=:r007,b00 r008,b01) r009,B02 = FU02(slc:r009,b02 r006,b03)
4		r009,B00 = FU01(or:r009,b00 r011,b01)
5		r005,B00 = FU01(-:r009,b00 r004,b01)
6		r009,B00 = FU01(<:r005,b00 r001,b01)
7	10	FU99($jp:r009,b07)
8		r002,B00 = FU01(set:r000,b00 r010,b01)
9	11	FU99($jp:)
10		r005,B07 = FU04(+:r004,b06 r005,b07) r002,B00 = FU01(set:r000,b00 r010,b01)
11	2	FU99($jp:r012,b07)
12		

CS of Division Algorithm

System : Single Clocked FUs
Number of FUs : 4
Execution Time : 125 ns * 12 clocks = 1500 ns

8 MHz clock.

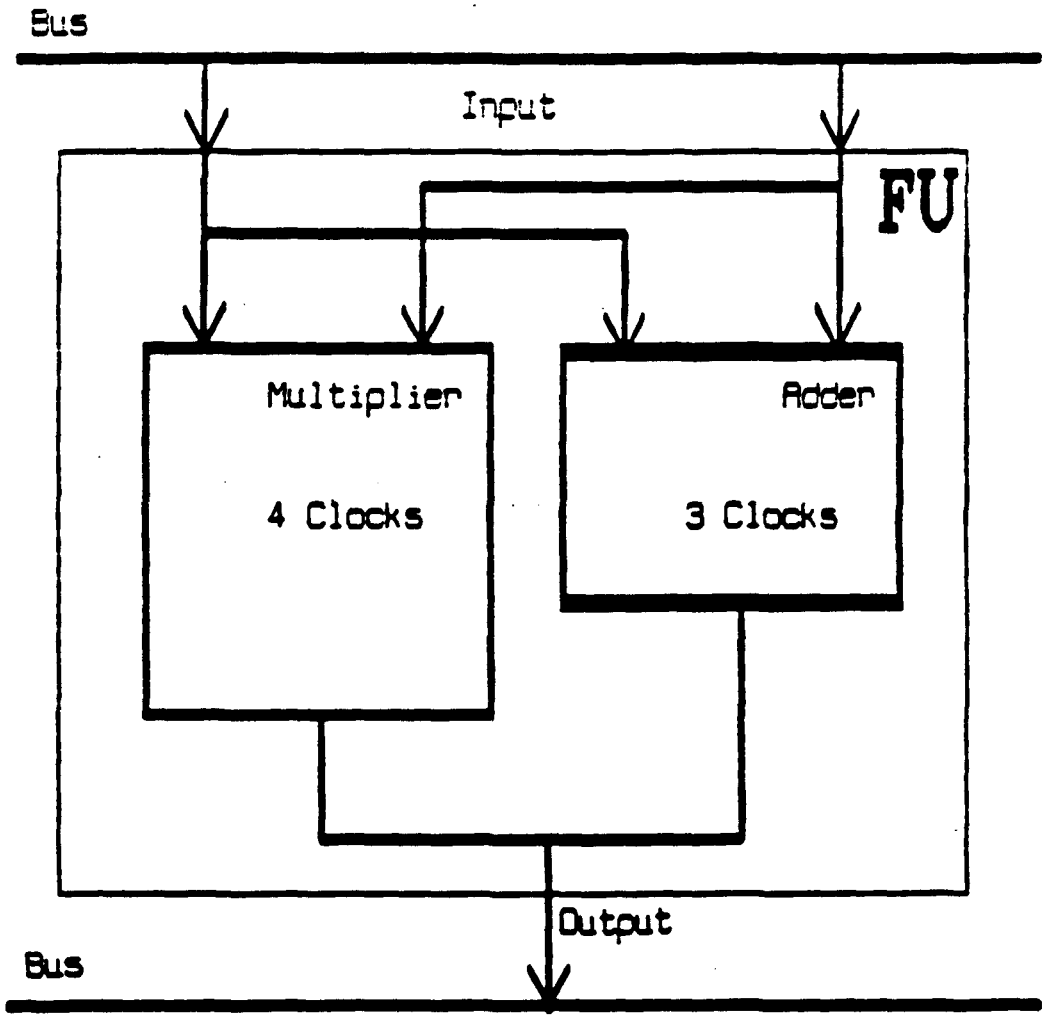
MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(shl:r005,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
3		r011,B02 = FU02(and:r002,b02 r003,b03) r010,B00 = FU01(shl:r006,b00 r002,b01)
4		r012,B00 = FU01(=:r007,b00 r008,b01) r011,B02 = FU02(slc:r011,b02 r006,b03)
5		r009,B00 = FU01(or:r011,b00 r009,b01)
6		r005,B00 = FU01(-:r009,b00 r004,b01)
7		r009,B00 = FU01(<:r005,b00 r001,b01)
8	11	FU99($jp:r009,b03)
9		r002,B01 = FU01(set:r000,b00 r010,b01)
10	12	FU99($jp:)
11		r005,B03 = FU02(+:r004,b02 r005,b03) r002,B00 = FU01(set:r000,b00 r010,b01)
12	2	FU99($jp:r012,b03)
13		

CS - with Restriction

System : Single Clocked FUs
Number of FUs : 2
Execution Time : 125 ns * 13 clocks = 1625 ns

Operator Set	Delay Time (ns)	Number of Clocks to Complete Operation			
		5 MHz	8 MHz	10 MHz	20 MHz
Cycle Time (ns)		200	125	100	50
+ -	150	1	1	2	3
* /	200	1	2	2	4
< > <= >= !=	150	1	1	2	3
shl shr	100	1	1	1	2
slc src	100	1	1	1	2
and or	50	1	1	1	1
set ext	50	1	1	1	1
abs	150	1	2	2	3

Figure 12 : Delay time vs. Number of clocks



Multiclocked FU

20 MHz CLOCK

MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(and:r002,b00 r003,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
3		r009,B00 = FU01(slc:r009,b00 r006,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
4		r007,B07 = FU04(+:r006,b06 r007,b07) r009,B00 = FU01(slc:r009,b00 r006,b01)
5		r009,B02 = FU02(or:r009,b02 r011,b03) r009,B00 = FU01(=:r007,b00 r008,b01)
6		r005,B02 = FU02(-:r009,b02 r004,b03) r009,B00 = FU01(=:r007,b00 r008,b01)
7		r009,B00 = FU01(=:r007,b00 r008,b01) r005,B02 = FU02(-:r009,b02 r004,b03)
8		r005,B02 = FU02(-:r009,b02 r004,b03)
9		r011,B00 = FU01(<:r005,b00 r001,b01)
10		r011,B00 = FU01(<:r005,b00 r001,b01)
11		r011,B00 = FU01(<:r005,b00 r001,b01)
12	15	FU99($jp:r011,b07)
13		r002,B00 = FU01(set:r000,b00 r010,b01)
14	18	FU99($jp:)
15		r005,B07 = FU04(+:r004,b06 r005,b07) r002,B00 = FU01(set:r000,b00 r010,b01)
16		r005,B07 = FU04(+:r004,b06 r005,b07)
17		r005,B07 = FU04(+:r004,b06 r005,b07)
18	2	FU99($jp:r009,b07)
19		

CS of Division Algorithm - Multiclocked FU

System : Multiclocked FUs
Number of FUs : 4
Execution Time : 50 ns * 19 clocks = 950 ns

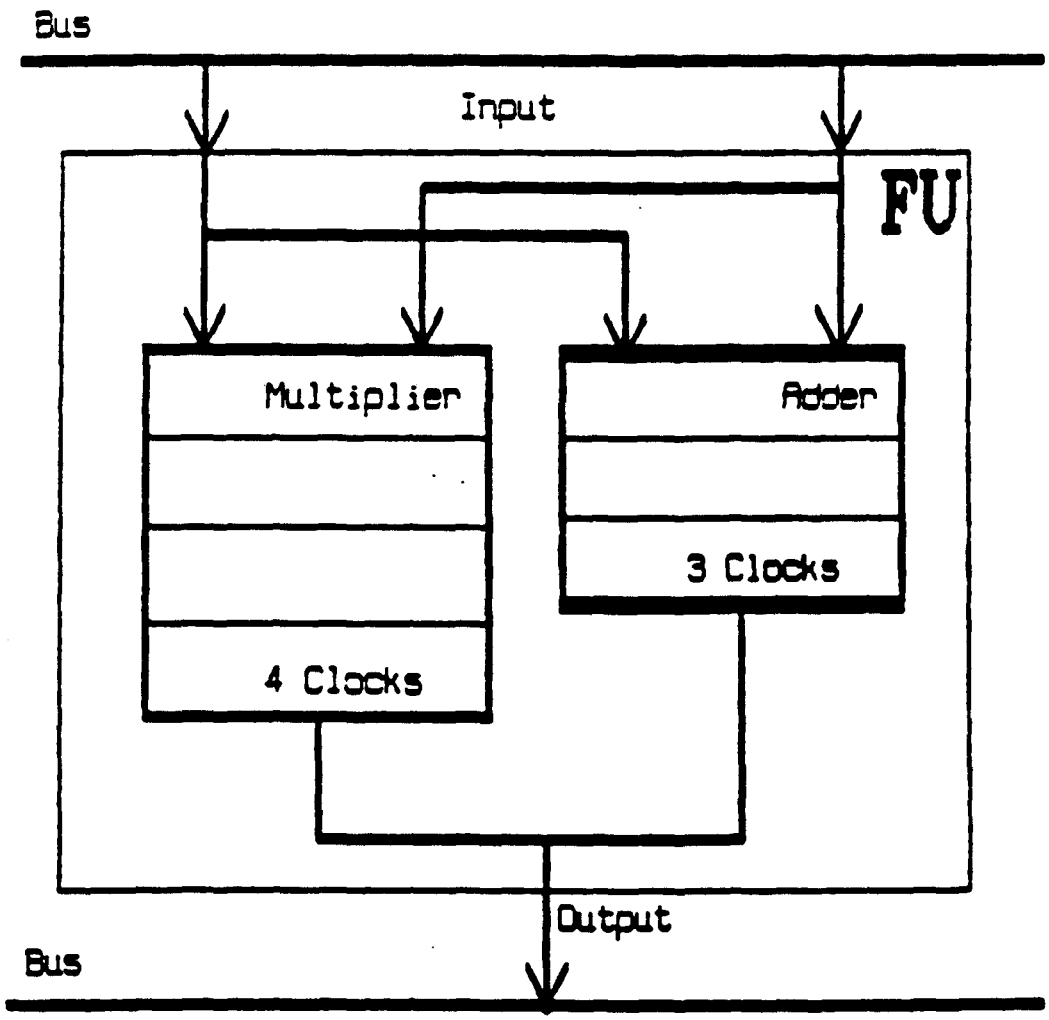
MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(shl:r005,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
3		r009,B00 = FU01(shl:r005,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
4		r010,B00 = FU01(and:r002,b00 r003,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
5		r010,B00 = FU01(shl:r006,b00 r002,b01) r011,B02 = FU02(slc:r010,b02 r006,b03)
6		r010,B00 = FU01(shl:r006,b00 r002,b01) r011,B02 = FU02(slc:r010,b02 r006,b03)
7		r009,B00 = FU01(=:r007,b00 r008,b01) r011,B02 = FU02(or:r011,b02 r009,b03)
8		r005,B02 = FU02(-:r011,b02 r004,b03) r009,B00 = FU01(=:r007,b00 r008,b01)
9		r009,B00 = FU01(=:r007,b00 r008,b01) r005,B02 = FU02(-:r011,b02 r004,b03)
10		r005,B02 = FU02(-:r011,b02 r004,b03)
11		r011,B00 = FU01(<:r005,b00 r001,b01)
12		r011,B00 = FU01(<:r005,b00 r001,b01)
13		r011,B00 = FU01(<:r005,b00 r001,b01)
14	17	FU99($jp:r011,b03)
15		r002,B00 = FU01(set:r000,b00 r010,b01)
16	20	FU99($jp:)
17		r005,B02 = FU02(+:r004,b02 r005,b03) r002,B00 = FU01(set:r000,b00 r010,b01)
18		r005,B02 = FU02(+:r004,b02 r005,b03)
19		r005,B02 = FU02(+:r004,b02 r005,b03)
20	2	FU99($jp:r009,b03)
21		

CS - Multiclocked FU with Restriction

System : Multiclocked FUs
Number of FUs : 2
Execution Time : 50 ns * 21 clocks = 1050 ns

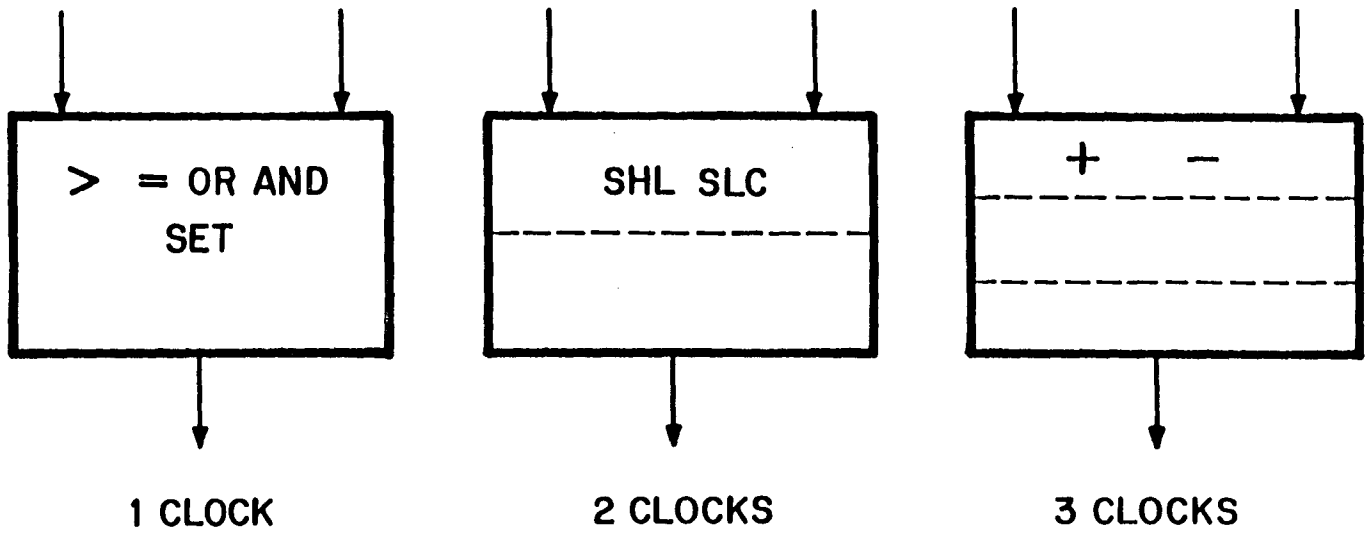
Model of Pipelined FUs

- Free to Accept Input in Each Clock
- Output Should not Conflict
- Assign Reg. and Bus in Final Clock



Pipelined FU

MULTI CLOCK OR PIPELINED UNITS



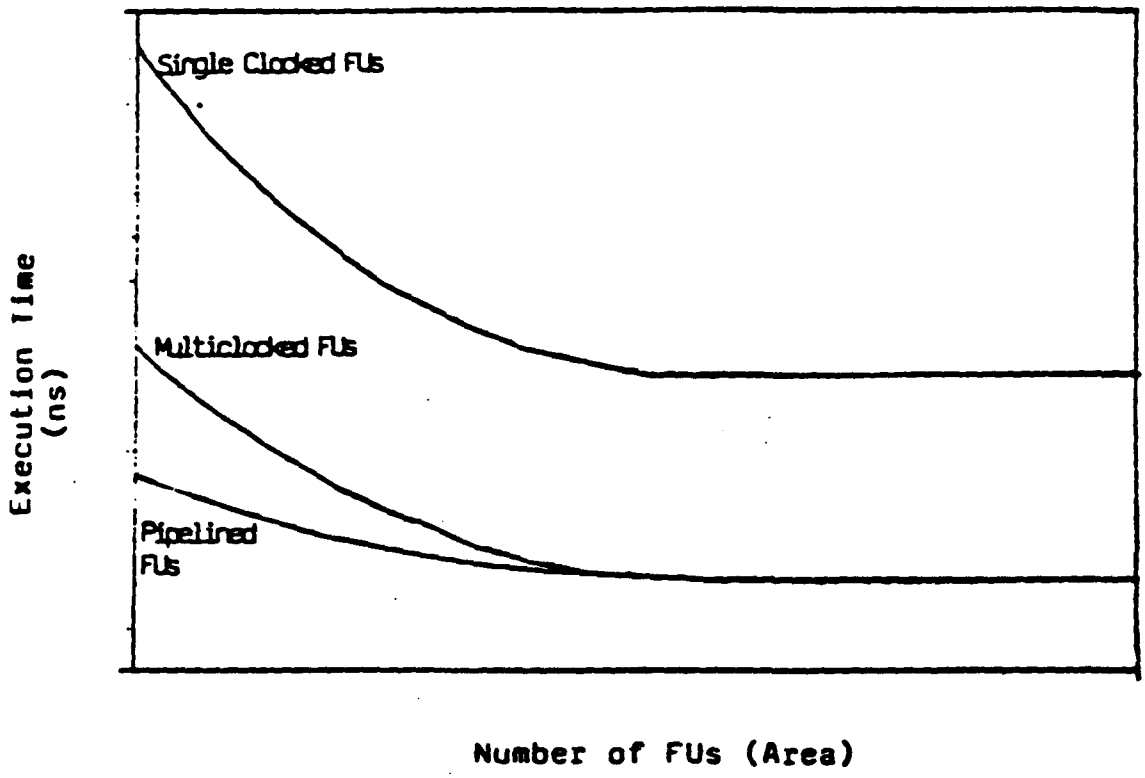
20 MHz clock.

MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(shl:r005,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
3		r010,B02 = FU02(and:r002,b02 r003,b03) r011,B00 = FU01(shl:r006,b00 r002,b01) r009,B00 = FU01(shl:r005,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03)
4		r010,B00 = FU01(slc:r010,b00 r006,b01) r007,B03 = FU02(+:r006,b02 r007,b03) r011,B00 = FU01(shl:r006,b00 r002,b01)
5		r010,B00 = FU01(=:r007,b00 r008,b01) r010,B00 = FU01(slc:r010,b00 r006,b01)
6		r009,B00 = FU01(or:r010,b00 r009,b01) r010,B00 = FU01(=:r007,b00 r008,b01)
7		r005,B00 = FU01(-:r009,b00 r004,b01) r010,B00 = FU01(=:r007,b00 r008,b01)
8		r005,B00 = FU01(-:r009,b00 r004,b01)
9		r005,B00 = FU01(-:r009,b00 r004,b01)
10		r009,B00 = FU01(<:r005,b00 r001,b01)
11		r009,B00 = FU01(<:r005,b00 r001,b01)
12		r009,B00 = FU01(<:r005,b00 r001,b01)
13	16	FU99($jp:r009,b03)
14		r002,B01 = FU01(set:r000,b00 r011,b01)
15	19	FU99($jp:)
16		r005,B03 = FU02(+:r004,b02 r005,b03) r002,B00 = FU01(set:r000,b00 r011,b01)
17		r005,B03 = FU02(+:r004,b02 r005,b03)
18		r005,B03 = FU02(+:r004,b02 r005,b03)
19	2	FU99($jp:r010,b03)
20		

CS - Pipelined FU with Restriction

System : Pipelined FUs
Number of FUs : 2
Execution Time : 50 ns * 20 clocks = 1000 ns

Design Space
Timer/Area Tradeoff

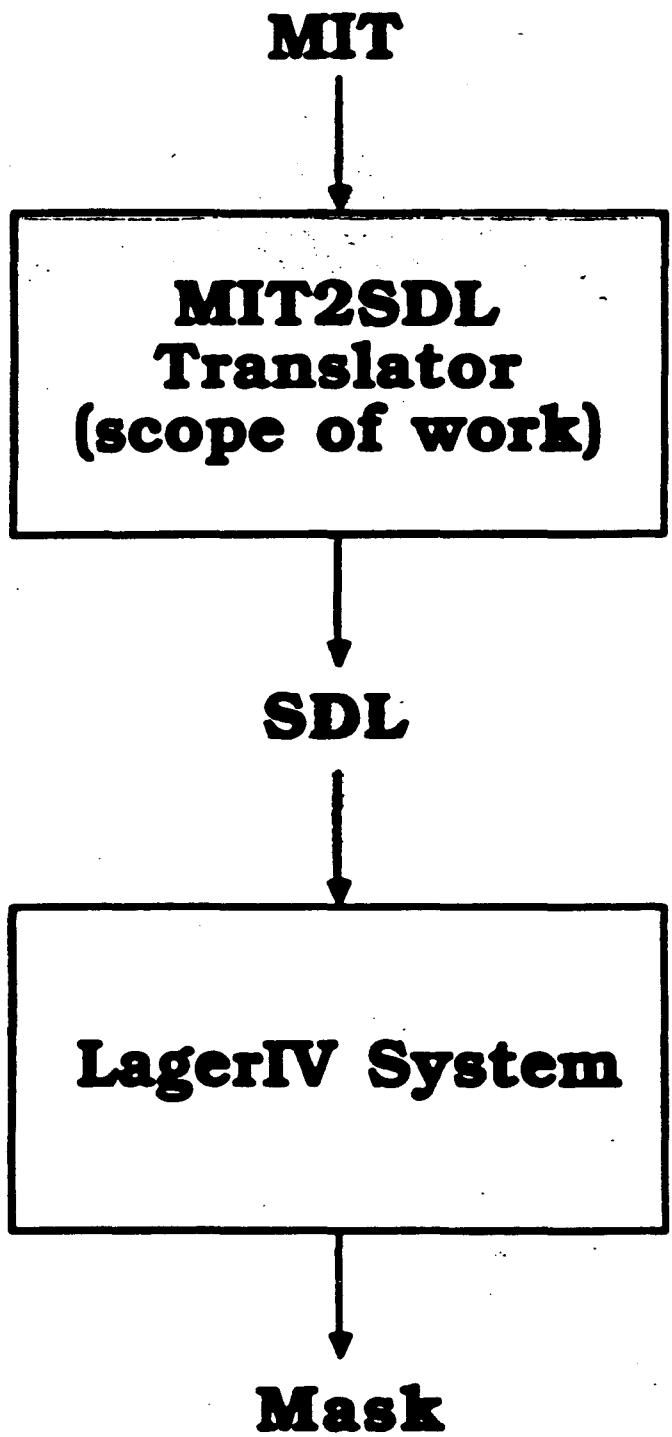


Time/Area Tradeoff

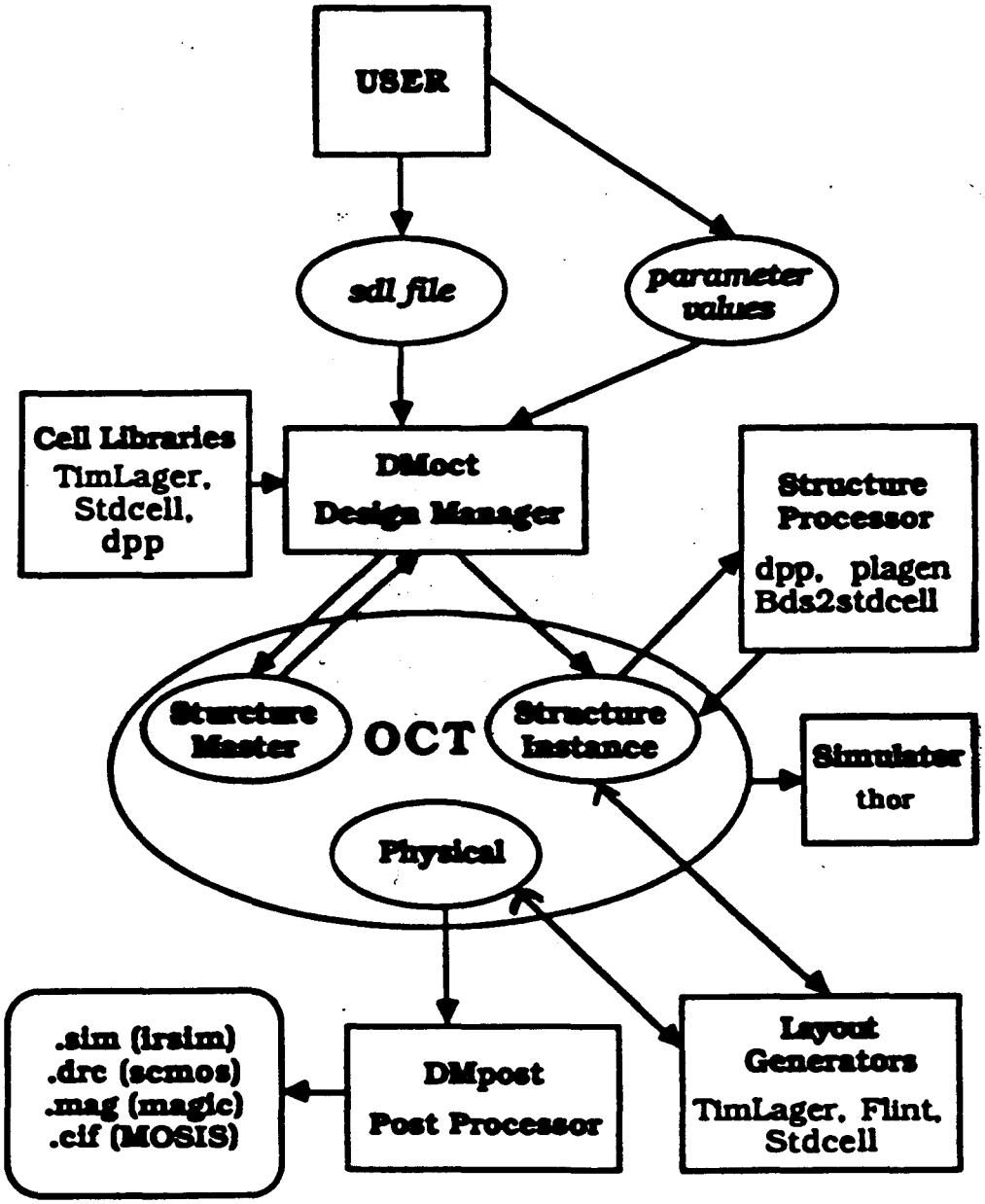
NUMBER OF MIs BY THEMSELVES MAY NOT HAVE MUCH MEANING AS A LONG WIRE MAY MAKE CLOCK CYCLES UNACCEPTABLY LONG.

HENCE A NEED TO LINK FLOORPLAN TO HIGH LEVEL SYNTHESIS

ONE OF THE PRIMARY AREAS OF ACTIVE RESEARCH.



Scope of Research



LagerIV system layout

2. INPUT

Micro-Instruction Table (MIT).

There are four types of MO's in the MIT:

1. rxxx = value

2. rxxx,BXX = FUXX(op-type:rxxx,bxx rxxx,bxx)

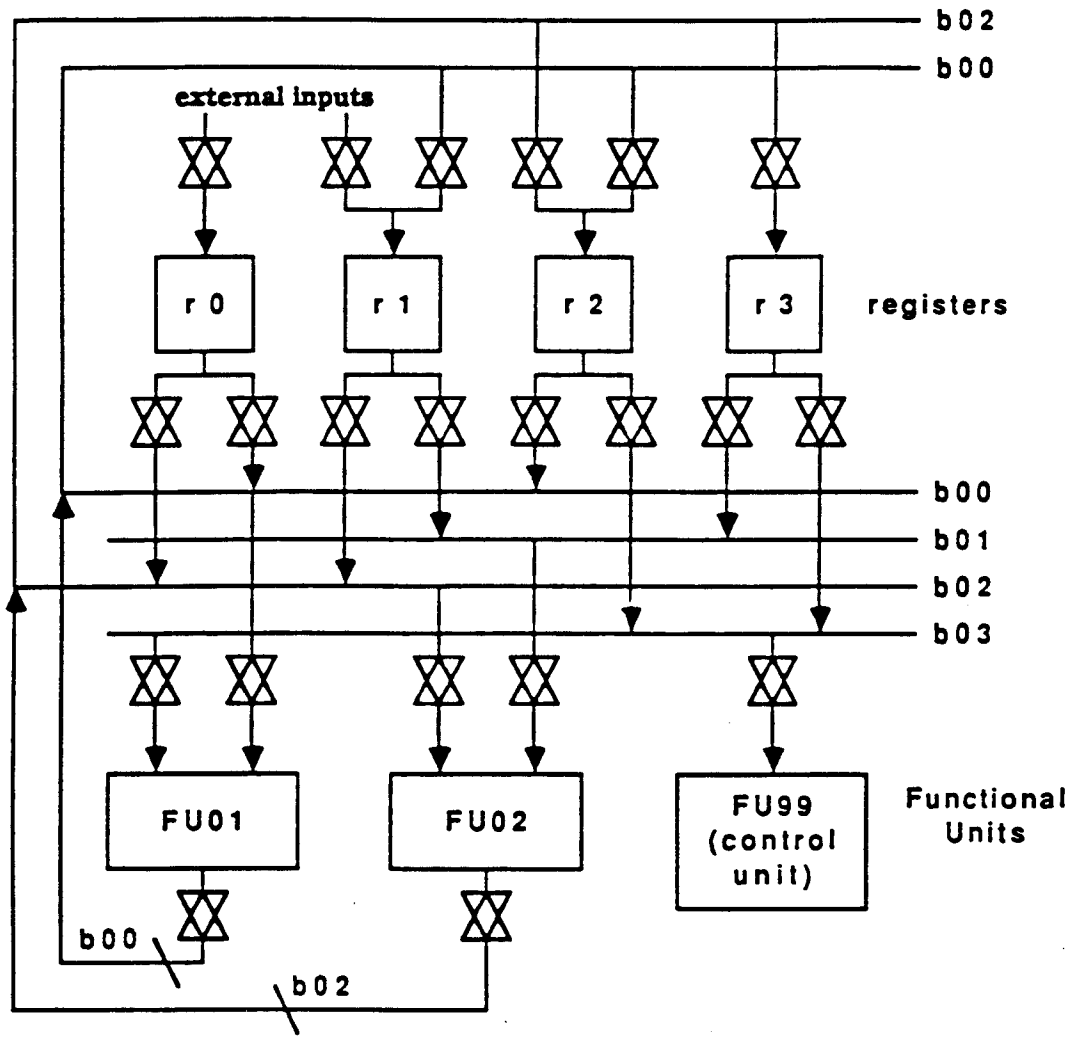
3. FU99($jp:rxxx,bxx)

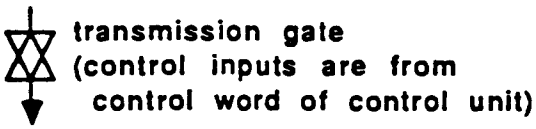
4. FU99($jp)

MI #	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(and:r002,b00 r003,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
3		r012,B00 = FU01(=:r007,b00 r008,b01) r009,B02 = FU02(slc:r009,b02 r006,b03)
4		r009,B00 = FU01(or:r009,b00 r011,b01)
5		r005,B00 = FU01(:r009,b00 r004,b01)
6		r009,B00 = FU01(<:r005,b00 r001,b01)
7	10	FU99($jp:r009,b07)
8		r002,B00 = FU01(set:r000,b00 r010,b01)
9	11	FU99($jp:)
10		r005,B07 = FU04(+:r004,b06 r005,b07) r002,B00 = FU01(set:r000,b00 r010,b01)
11	2	FU99($jp:r012,b07)

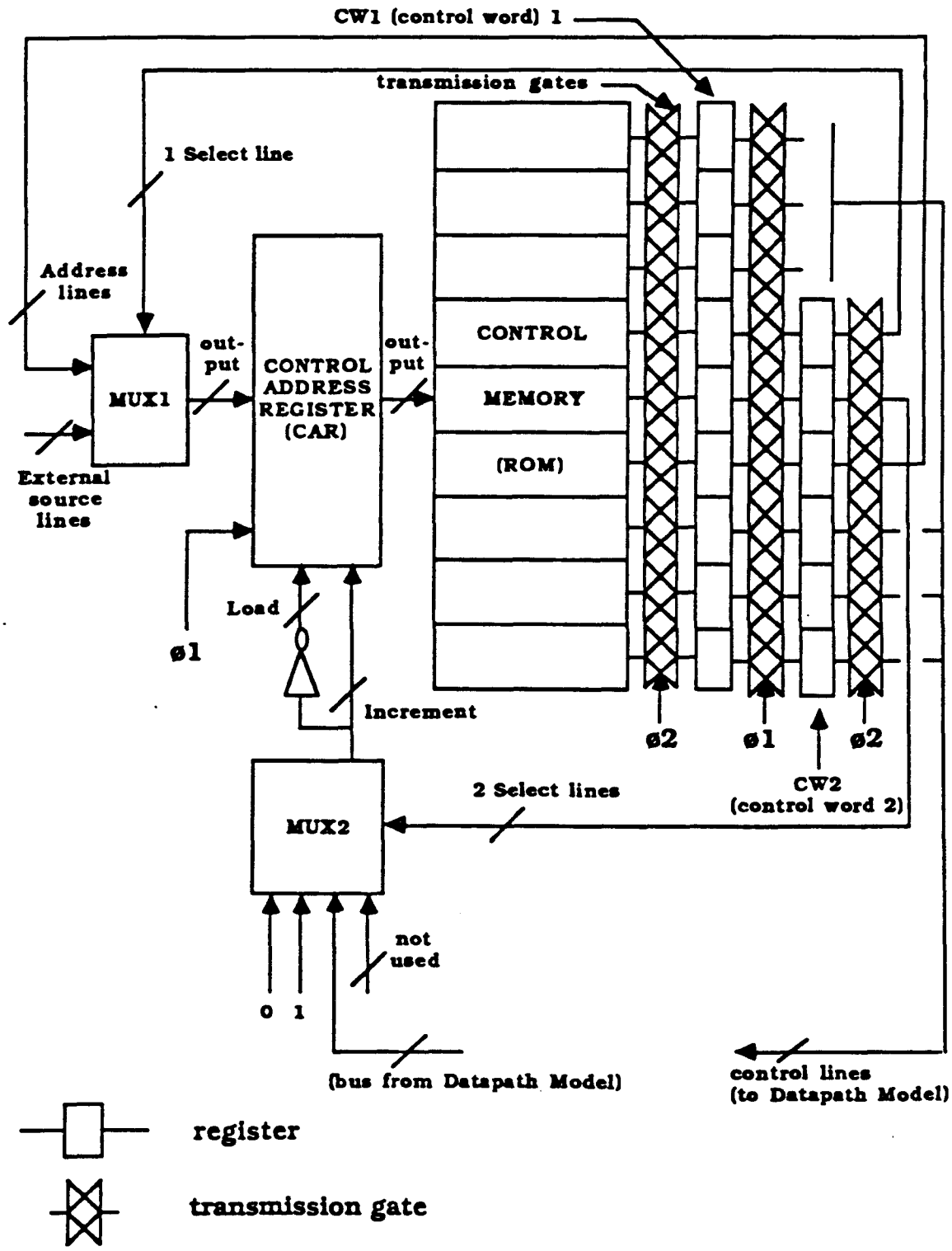
3. OUTPUT

1. Model of Datapath
2. Model of Control Unit





Model Datapath



Model Control Unit

MI#	JUMP	MICRO INSTRUCTIONS
1		r000 = value r001 = value r002 = value r003 = value r004 = value r005 = value r006 = value r007 = value r008 = value
2		r009,B00 = FU01(and:r002,b00 r003,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
3		r009,B00 = FU01(slc:r009,b00 r006,b01) r010,B02 = FU02(shl:r006,b02 r002,b03) r011,B04 = FU03(shl:r005,b04 r006,b05) r007,B07 = FU04(+:r006,b06 r007,b07)
4		r007,B07 = FU04(+:r006,b06 r007,b07) r009,B00 = FU01(slc:r009,b00 r006,b01)
5		r009,B02 = FU02(or:r009,b02 r011,b03) r009,B00 = FU01(=:r007,b00 r008,b01)
6		r005,B00 = FU01(-:r009,b00 r004,b01) r009,B00 = FU01(=:r007,b00 r008,b01)
7		r009,B00 = FU01(=:r007,b00 r008,b01) r005,B00 = FU01(-:r009,b00 r004,b01)
8		r005,B00 = FU01(-:r009,b00 r004,b01)
9		r011,B00 = FU01(<:r005,b00 r001,b01)
10		r011,B00 = FU01(<:r005,b00 r001,b01)
11		r011,B00 = FU01(<:r005,b00 r001,b01)
12	15	FU99($jp:r011,b07)
13		r002,B00 = FU01(set:r000,b00 r010,b01)
14	18	FU99($jp:)
15		r005,B07 = FU04(+:r004,b06 r005,b07) r002,B00 = FU01(set:r000,b00 r010,b01)
16		r005,B07 = FU04(+:r004,b06 r005,b07)
17		r005,B07 = FU04(+:r004,b06 r005,b07)
18	2	FU99($jp:r009,b07)
19		

CS of Division Algorithm - Pipelined FU

System : Pipelined FUs
Number of FUs : 4
Execution Time : 50 ns * 19 clocks = 950 ns

SDL - Structure Description Language

The sdl file for the parent-cell at any level of the hierarchy gives 5 essential pieces of information:

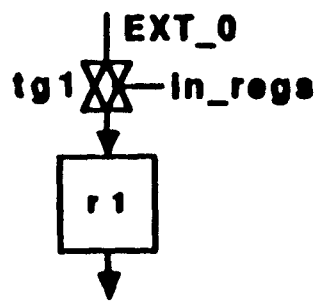
1. The **parent-cell** *master_name*
2. The cell **parameters**
3. The **subcells**
4. The cell **layout-generator** program
5. The **net** list

SDL is used to instantiate and connect these registers, transmission gates and functional units to create a datapath.

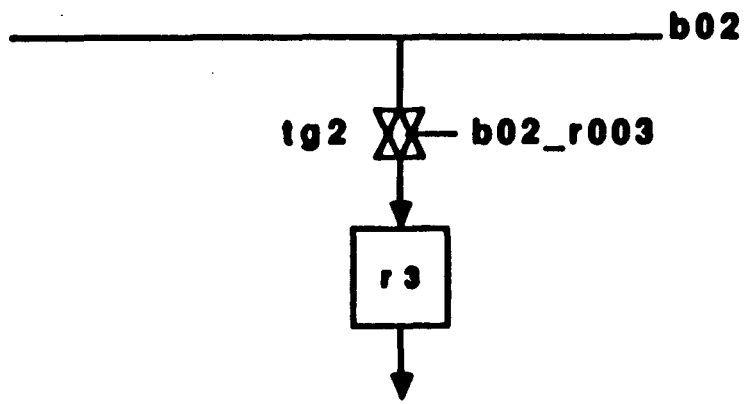
ALGORITHM 1 - Register Instantiation

1. (subcells (register (r001 r003))
   (trans_gate (tg1 tg2 tg4)))

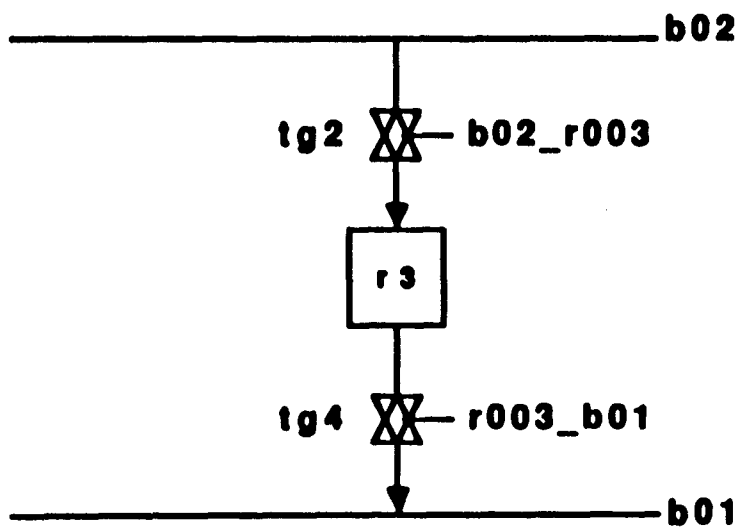
2. (net ext_1 (NETTYPE SIGNAL) ((parent EXT_0)
   (tg1 IN)))
   (net tg1_r001 (NETTYPE SIGNAL) ((tg1 OUT)
   (r001 IN)))
   (net in_regs (NETTYPE SIGNAL) ((parent in_regs)
   (tg1 CNTL)))
   (terminal in_regs (TERMTYPE SIGNAL))



3. (net b02 (NETTYPE SIGNAL) ((parent b02) (tg2 IN)))
(net tg2_r003 (NETTYPE SIGNAL) ((tg2 OUT)
(r003 IN)))
(net b02_r003 (NETTYPE SIGNAL)
((parent b02_r003) (tg2 CNTL)))
(terminal b02_r003 (TERMTYPE SIGNAL))



4. (net r003_tg4 (NETTYPE SIGNAL) ((r003 OUT)
(tg4 IN)))
(net b01 (NETTYPE SIGNAL) ((parent b01)
(tg4 OUT)))
(net r003_b01 (NETTYPE SIGNAL)
((parent r003_b01) (tg4 CNTL)))
(terminal r003_b01 (TERMTYPE SIGNAL))

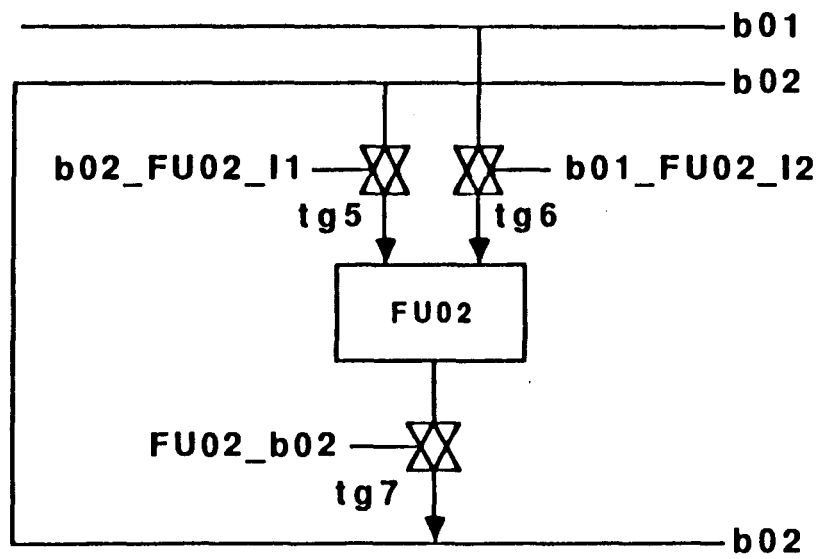


ALGORITHM 2 - Functional Unit Instantiation

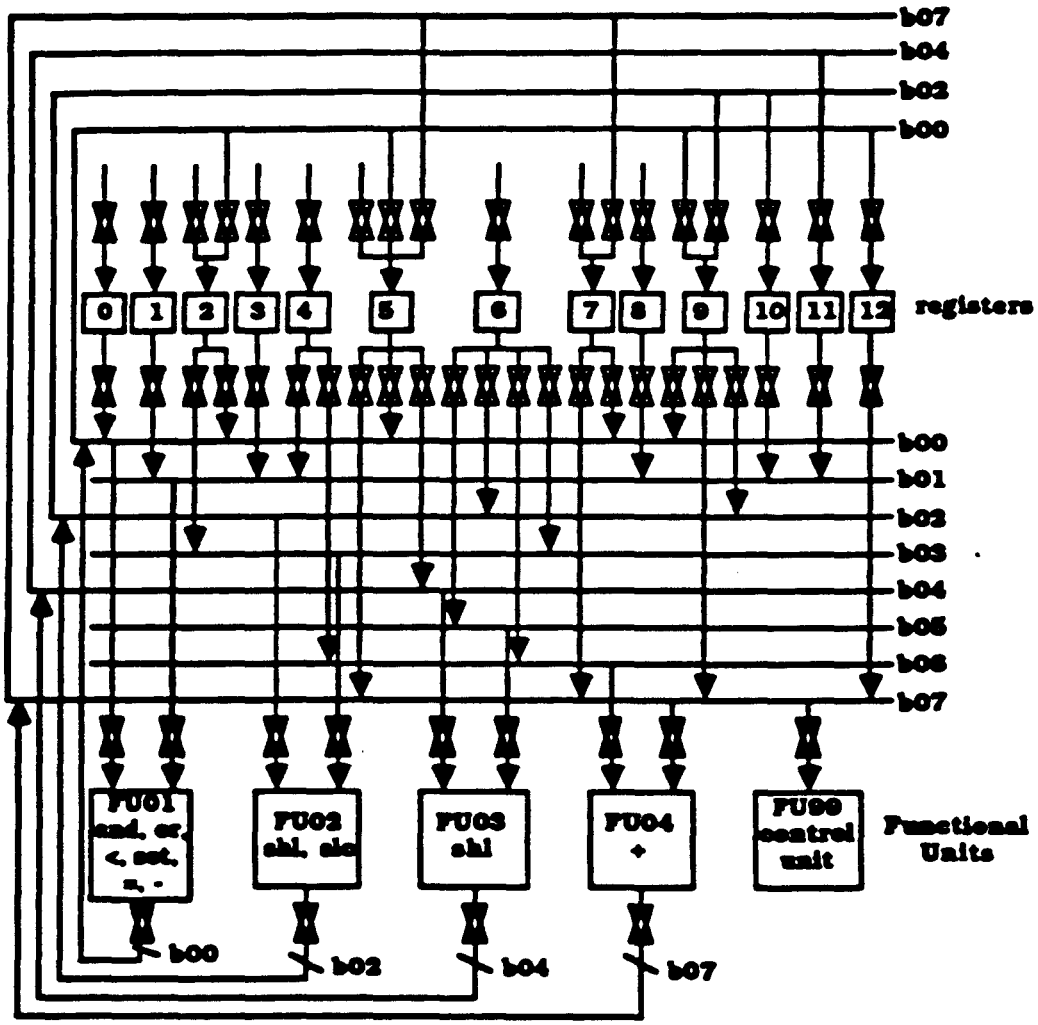
1. (subcells (FFU02 FU02))
   (trans_gate (tg6 tg7 tg8)))

2. (net b02 (NETTYPE SIGNAL) ((parent b02)
   (tg5 IN)))
   (net tg5_FU02_I1 (NETTYPE SIGNAL) ((FU02 IN1)
   (tg5 OUT)))
   (net (b02_FU02_I1 (NETTYPE SIGNAL)
   ((parent b02_FU02_I1) (tg5 CNTL)))
   (terminal b02_FU02_I1 (TERMTYPE SIGNAL))

3. (net FU02_tg7 (NETTYPE SIGNAL) ((FU02 OUT)
(tg7 IN)))
(net b02 (NETTYPE SIGNAL) ((parent b02)
(tg7 OUT)))
(net FU02_b02 (NETTYPE SIGNAL)
((parent FU02_b02) (tg7 CNTL)))
(terminal FU02_b02 (TERMTYPE SIGNAL))



4. (net FU02_and (NETTYPE SIGNAL)
((parent FU02_and) (FU02 and)))
(terminal FU02_and (TERMTYPE SIGNAL))



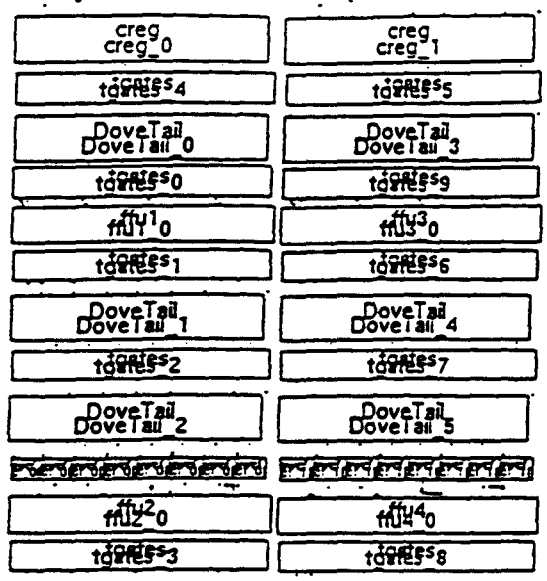
transmission gate
(control inputs are from
control word of control unit)

Datapath

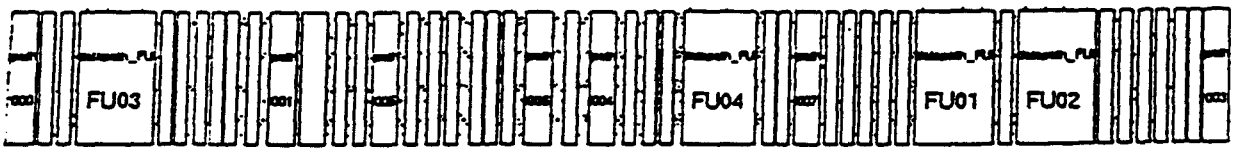
ALGORITHM 3 - Control Memory (ROM)

The different fields in the control memory are listed below :

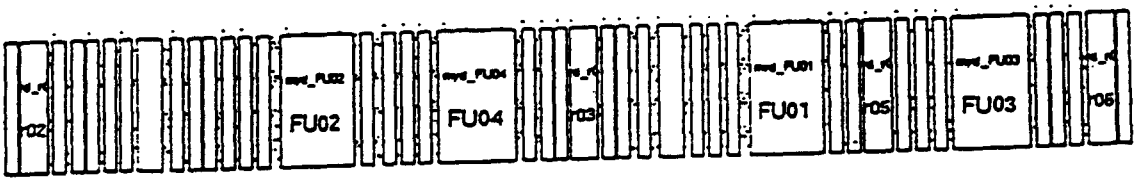
	FIELD	SYMBOL	FUNCTION
1)	RTB	rxxx_bxx	register output to I/O buses
2)	BTF	bxx_FUXX_IX	inputs to functional units from I/O buses
3)	OPS	FUXX_op	operation select of functional units
4)	SEL1	int, ext	MUX1 select (int = 0, ext = 1)
5)	SEL2	0, 1, FU99	MUX2 select (0 = 00, 1 = 01,FU99 = 10)
6)	ADRS	address #	address field
7)	INR	in_regs	initialize registers
8)	FTB	FUXX_bxx	output of functional units to I/O buses
9)	BTR	bxx_rxxx	inputs to registers from I/O buses



The Stylistic Data Path



The Conventioanl Data Path With 5 Micro Instructions



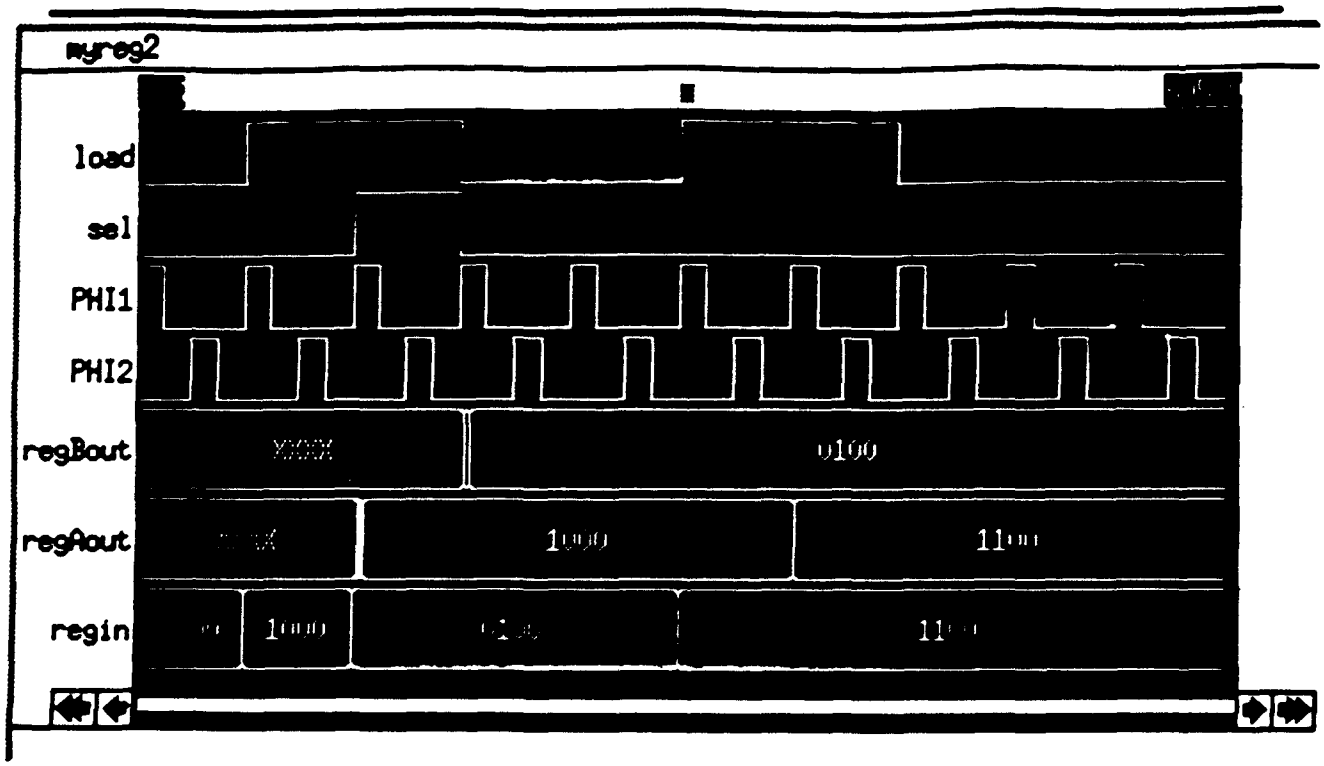
The Conventioanl Data Path With 4 Micro Instructions

This slide can also be used at end of next section.

SWITCH LEVEL SIMULATION

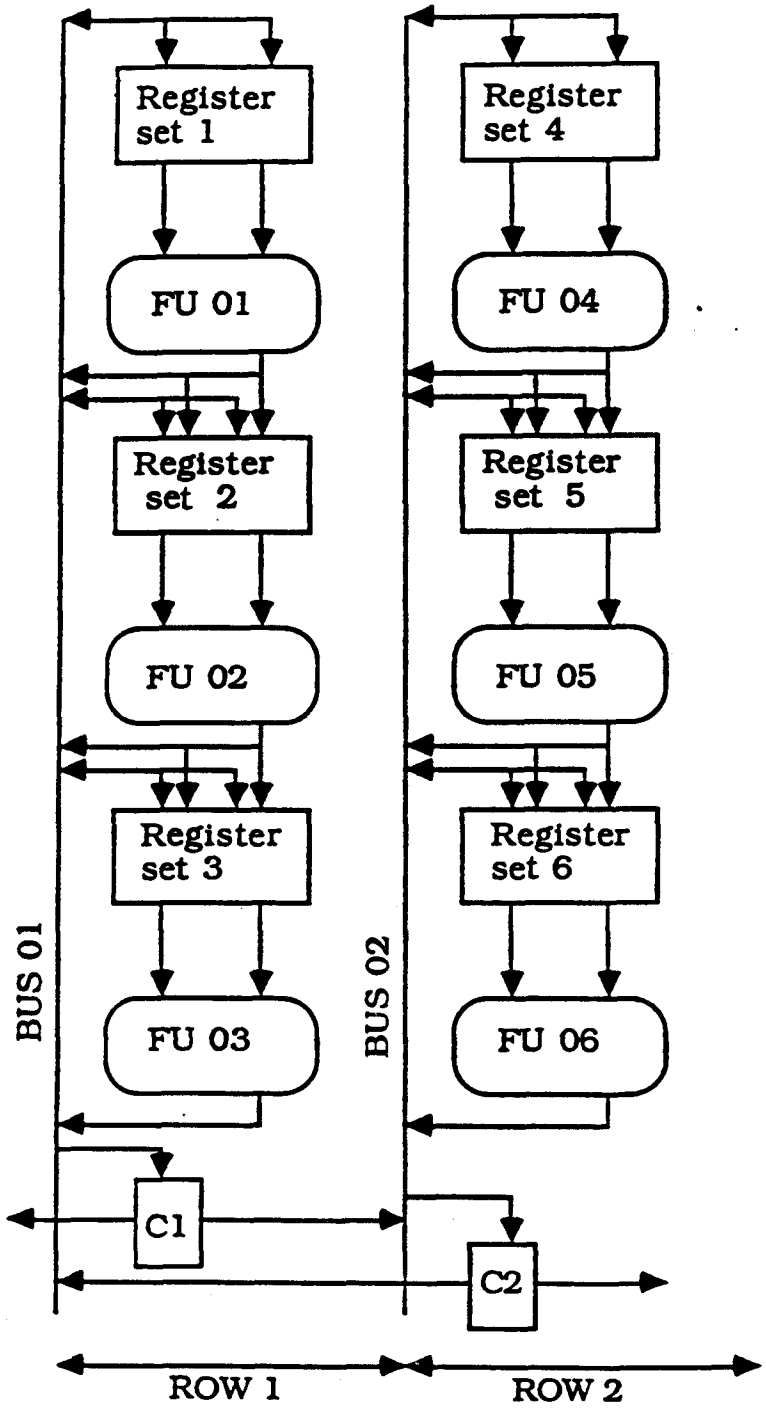
- o **Extract Circuit from Masks (standard tool)**

- o **Stimulus is Known "r1,b1 <- FU1:(+: r1,b1 r5,b2)"**
  Simulate each microinstruction
  Use the worst case clock cycle

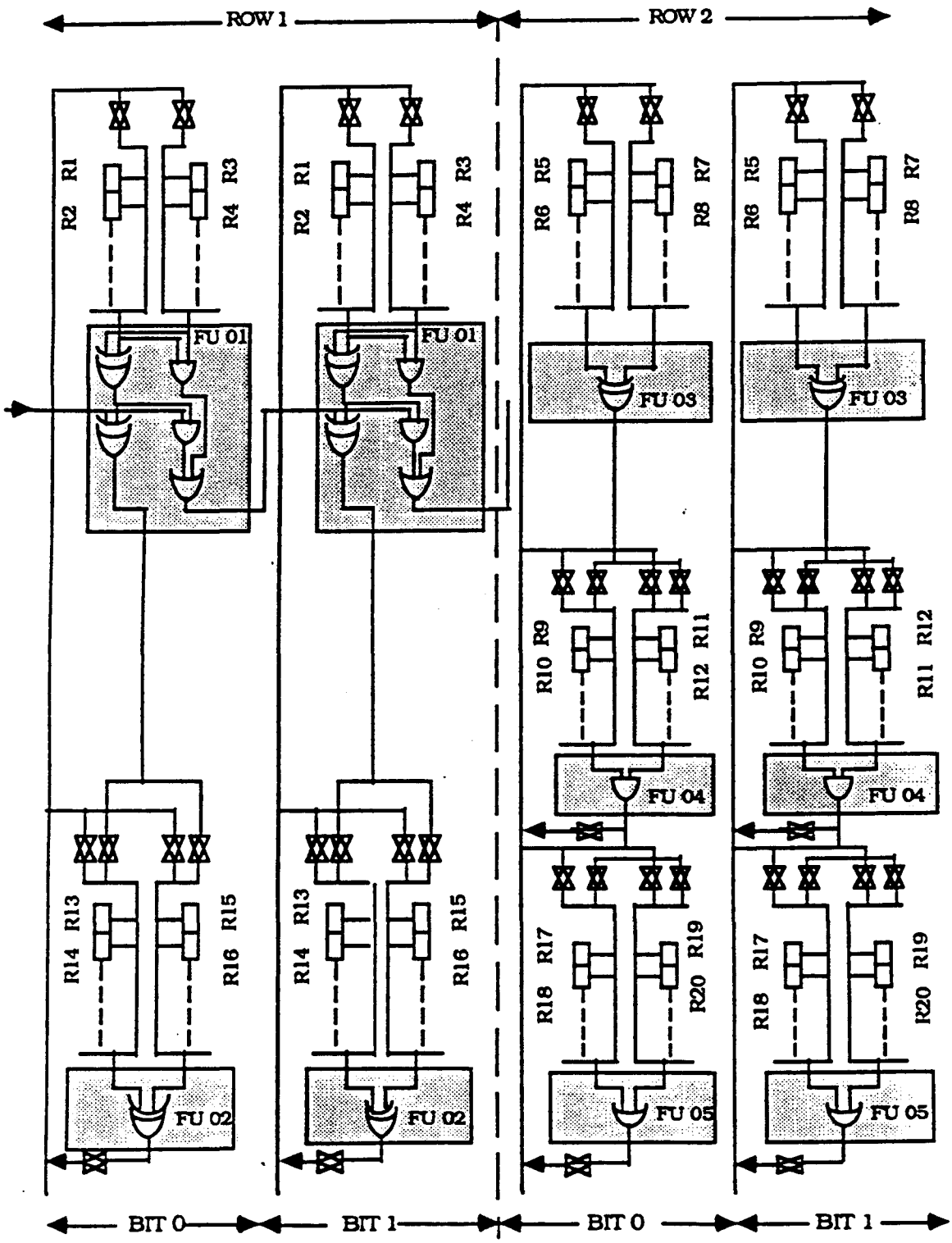


Invoke IRSIM with **irsim scmos100.prm myreg2.sim –reg2.irsim.cmds** where the file *reg2.irsim.cmds* contains the following statements:

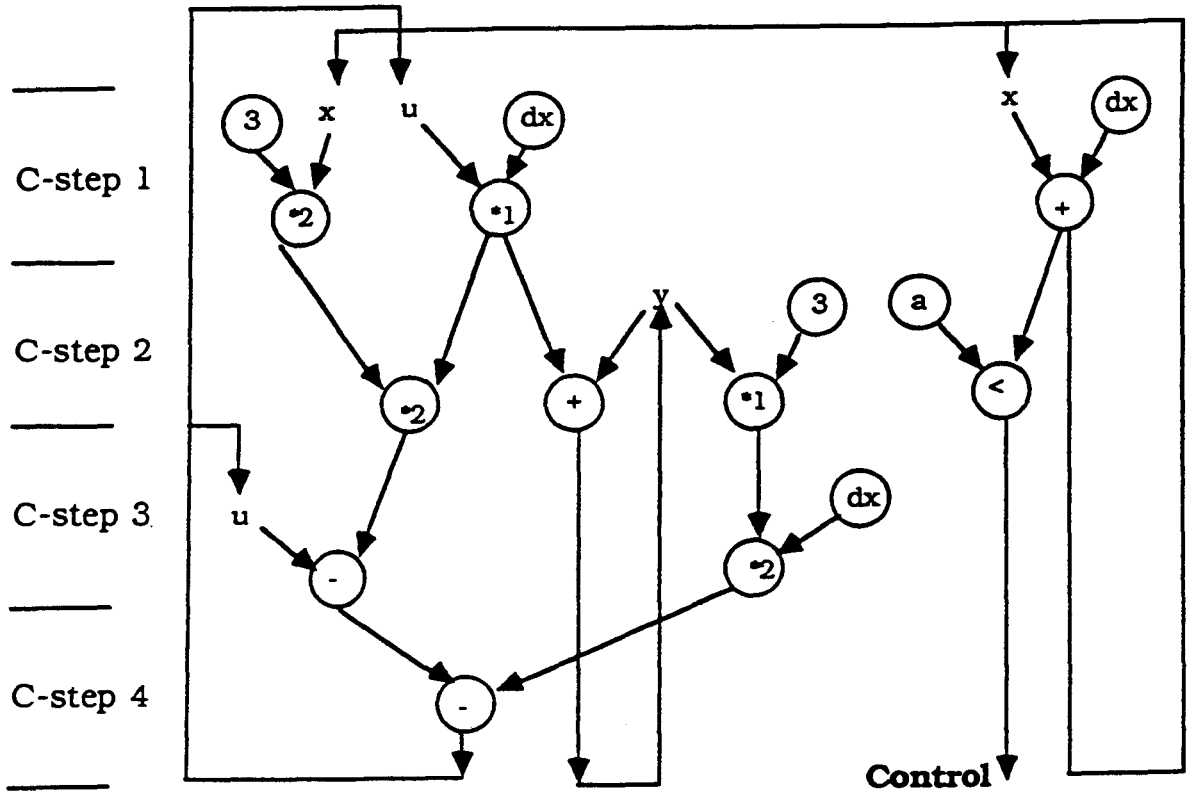
```
clock PHI1 1 0 0 0
clock PHI2 0 0 1 0
vector regin regin[3] regin[2] regin[1] regin[0]
vector regAout regAout[3] regAout[2] regAout[1] regAout[0]
vector regBout regBout[3] regBout[2] regBout[1] regBout[0]
analyzer load sel PHI1 PHI2 regBout regAout  regin
stepsize 15
V load   0    1    1    0    0    1    1    0    0    0
V sel    0    0    1    0    0    0    0    0    0    0
V regin 0000 1000 0100 0100 0100 1100 1100 1100 1100 1100

R                    (the R command runs the simulator with the defined vectors)
```

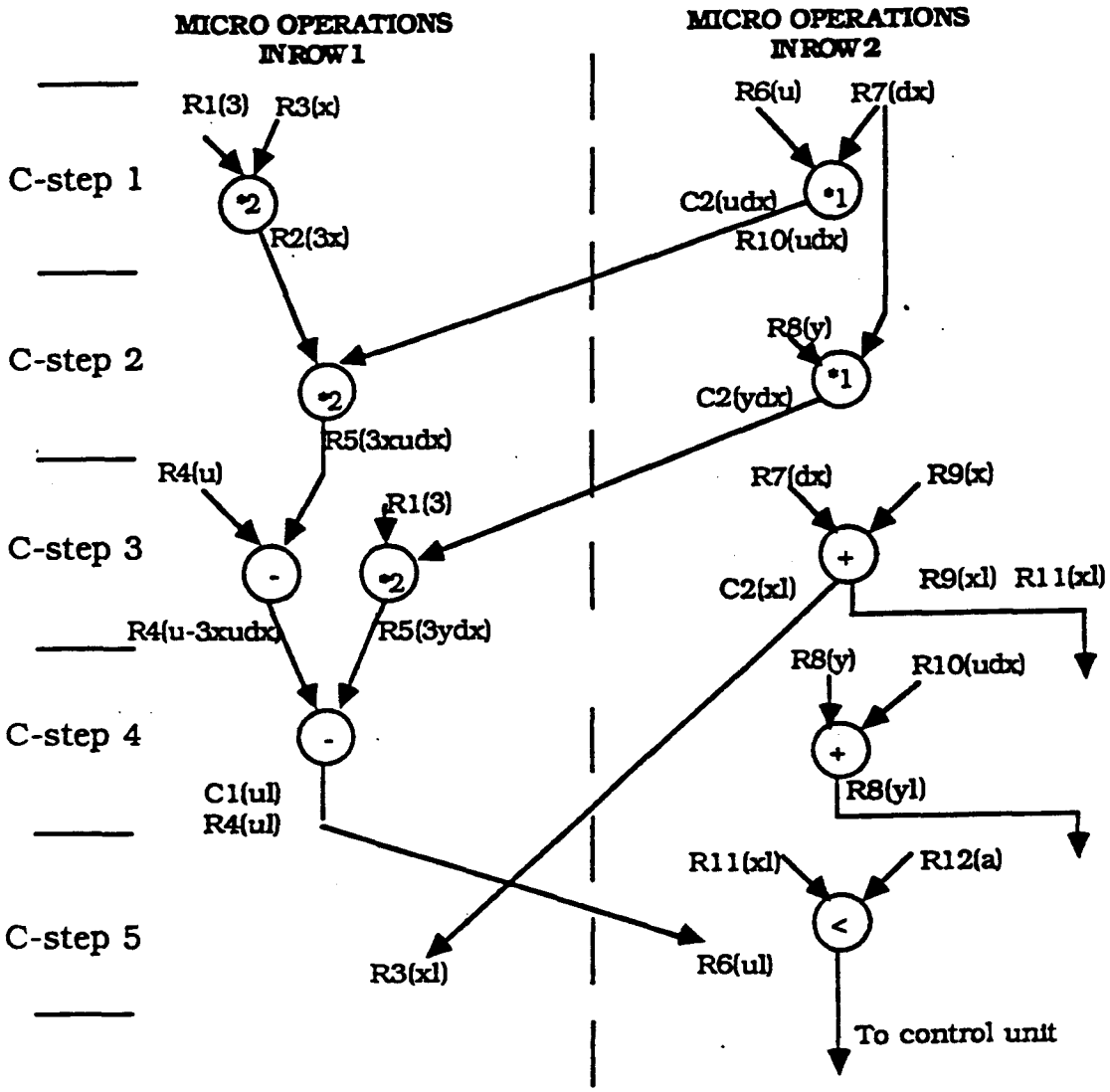


MODEL OF THE DATA PATH AT BLOCK LEVEL.



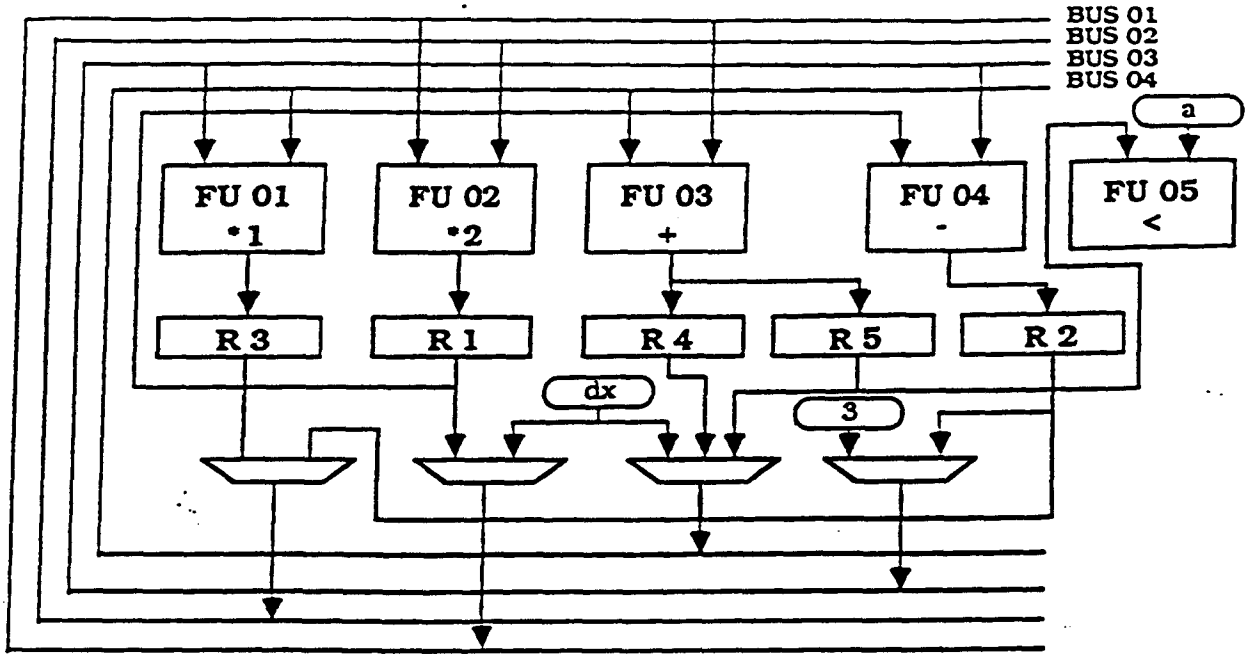
Gate level diagrams of a data path model.


C-step 1
C-step 2
C-step 3
C-step 4
3
x
u
dx
*2
*1
x
dx
+
y
3
a
*2
+
*1
<
u
dx
-
*2
-
Control

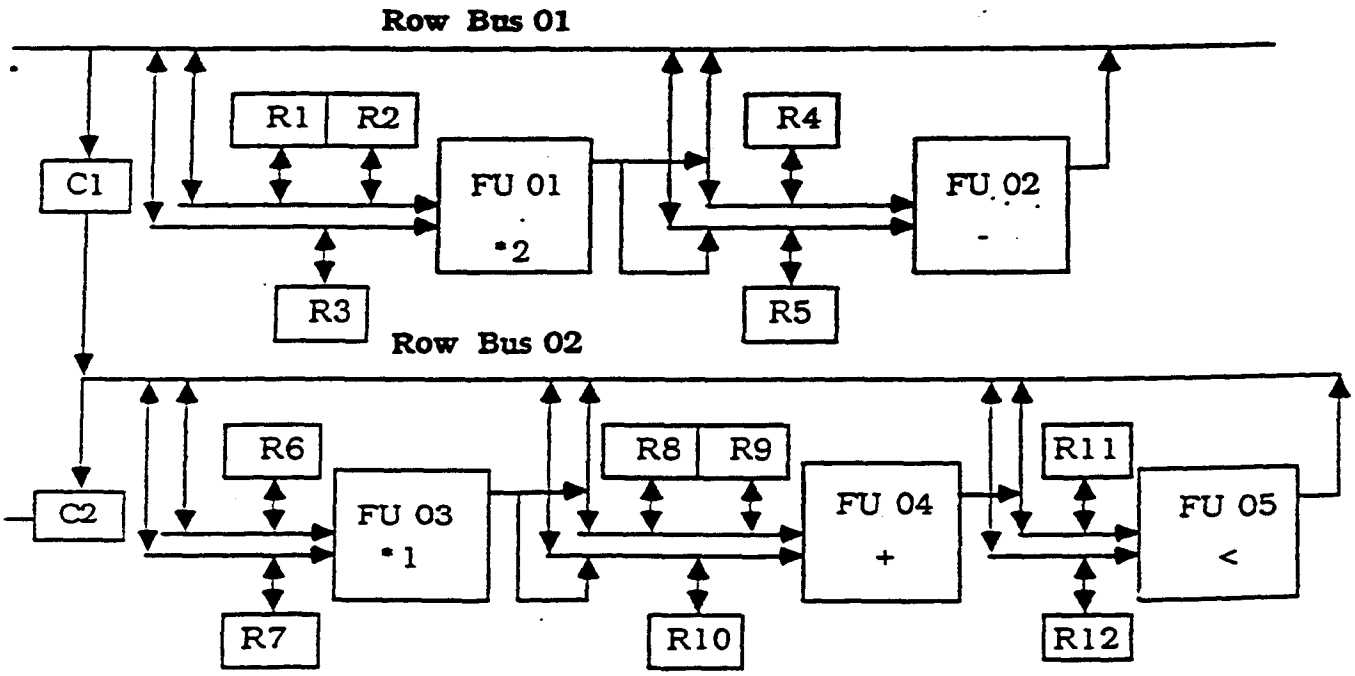


Note: Rx(y) means Register # x contains y.

MICRO-INSTRUCTION TABLE.



Non-Stylistic



Stylistic

Interconnectivity of Data Path

See layout slide preceding this section.

ASIC
(Application Specific IC)

- Traditional Method of Design Using CAD (Computer Aided Design System)

- Soon to Become Popular Design Automation (High Level Synthesis or Digital System Design by "Others")

SPEED OF MICROPROCESSOR BASED DESIGNS
(Rough Approximation)

Assume 30 MHz Clock

- "Worst Case" Instruction ADD (x), (Y)

Fetch Inst.	=	3 memory Accesses
Indirect on X	= fetch X then operand =	2 memory Accesses
Indirect on Y	= fetch Y then operand =	2 memory Accesses
Store Result	= (Optimistically)	1 memory Access

  At 4 cycles per access = 8 * 4 = 32 cycles
  Decode and execute = 2 cycles
  Total 34 cycles or

  1 Inst per Micro-second.

- "Best Case" Instruction ADD R0, R1

One memory Access	=	4 cycles
Decode and Execute	=	2 cycles

  Total 6 cycles or

  5 inst per Micro-second

- Average
  (excluding co-processor overhead;
  numerous assembler insts per task etc.)

  3 insts per Micro-second

300 nano-seconds per instruction

ASIC DESIGN

2 clocks per task

- 60 MHz clock (CMOS)

  30 Nano-second per inst (Approx)

- 200 MHz Clock (GaAs)

  10 Nano Seconds per Instruction (Approx)

DISADVANTAGES AND ADVANTAGES OF ASIC DESIGNS

DISADVANTAGES

- DIFFICULT TO CHANGE
- COST MAY BE HIGHER

ADVANTAGES

- SPEED OF CIRCUIT
- EASY TO MAINTAIN
- EASY TO IMPLEMENT
- NEW TECHNOLOGY
- ENHANCE UNIVERSITY RESEARCH

ISSUES TO BE ADDRESSED

- Develop a Front End Interface
- Integrate into an "Enviornment"

SSCL

Proposed Silicon Compiler for Physics Applications

* * *

Guy Vanstraelen

SSC Laboratory

Outline

SSCL

1. Silicon compiler: What?
2. Classes of Synthesizable Problems
   - → High Throughput systems
   - → Synthesis
3. "Meet in the Middle" Concept
4. Conclusions

1. Silicon Compiler: What?

SSCL

Definition:

= software system supporting

behavioral description at the algorithmic level

↓ **input: throughput, area**
expert design rules

chip layout synthesis

Procedure:

i) Behavioral description at system level

ii) Verification of behavioral correctness (Verilog, VHDL)

iii)Optimization

iv)Compilation into structure

Important: **"THE" silicon compiler does NOT exist.**

2. Synthesizable Problems

SSCL

Architectural style	Target application	Building blocks
Highly multiplexed	Low sample rate, complex decision making algorithms	Predefined highly programmable data-paths (ALUs)
Pipelined (*)	High throughput	Predefined funtional units (FUs)
Lowly multiplexed cooperating data-path architectural style (*)	High throughput	Abstract building blocks (ABBs)

Composition of the data-paths based on

- repetitiousness (FOR, WHILE)
- signal flow dependencies (recursion) of the signal flow graph representing the DSP algorithm

e.g. Cathedral III

High Throughput Systems

SSCL

Characteristics:

- **<u>Dedicated data-paths</u> build with ABBs to cope with the high throughput requirements.**

  ABB = abstract type of primitive operator at the RT-level, for which one or more hardware representations exist in a hardware library

- **<u>Internal composition of data-paths not predefined</u>, defined on the fly during synthesis**
- **<u>Main high-level synthesis requirement</u>: *definition* and *optimization* of the *data-paths* and *memory management***

Synthesis

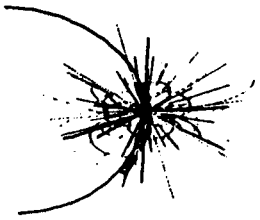
= two step process

SSCL

1. Behavioral synthesis

2. Structural synthesis

Synthesis: Behavioral



SSCL

<u>Input:</u> behavioral description of the algorithm

<u>Output:</u> architecture netlist, built up while iterating over:

- ***high-level memory management*** **(memory allocation, address generation)**
- ***high-level data-path mapping tasks*** **(optimization of transformations, partioning and allocation, data-path definition, assignment and optimization, initial scheduling)**
- ***low-level memory management*** **(memory allocation for temporarily storage)**
- ***low-level data-path mapping tasks*** **(detailed scheduling and detailed interconnect generation)**

Synthesis: Structural

SSCL

Input: architecture netlist

Output: detailed netlist, to be used by the layout generation

3. "Meet in the Middle" Concept

SSCL

Dedicated data-paths

→ ***system*** *design* **separated from** ***silicon*** *design*

Interface: located at the level of

- **arithmetic and logic operator blocks**
- **data storage**
- **controllers and I/O units**

→ **available under the form of** ***modules***

Result:

Design at system level

= ***translation of system specs → netlist of module instances***

Modules

SSCL

Module design ...

- **requires a powerful design environment**
- **should be done by a team of silicon designers**

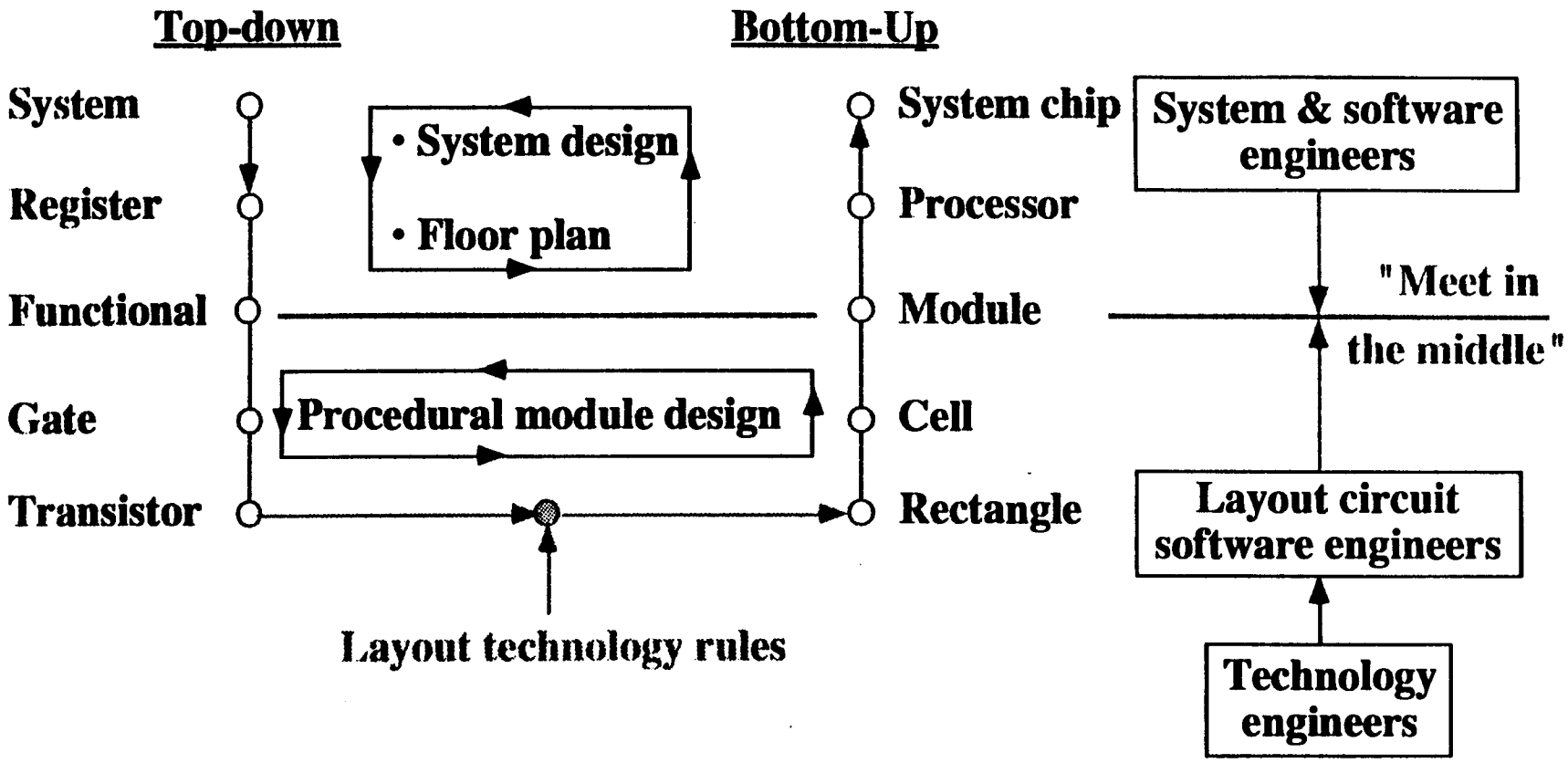
→ *expensive*

<u>Essential:</u>

- **re-usibility (library)**
- **technology adaptability (symbolic layout)**

"Meet in the middle" strategy

SSCL



4. Conclusions

SSCL

1. **Physics applications: *high throughput* requirements, *interdependent data-streams.***
2. ***Dedicated data-paths*: defined on the fly during the synthesis process.**
3. ***"Meet in the middle"* concept: the *high-level algorithmic description* is *synthesized using technology independent modules*, available in a library.**
4. **The *modules* are realized in a *bottom-up fashion*. The *data-paths* are obtained in a *top-down* fashion.**

Timed-LOTOS in a PROLOG Environment: an Algebraic Language for Simulation

M.L. Ferrer, INFN-LNF, Italy
G. Mirabelli, INFN-Roma, Italy ← Major ideas and major work

- Introduction
- LOTOS (short tutorial)
- LOTOS tools (without time)
    - BELASI
    - HIPPO
- Timed-LOTOS
    - Timed-action
    - Timed-interaction
    - Single-Timed (this presentation)

- Single-Timed Lotos:
    - Semantic (examples) ← PROLOG tutorial?
    - Lotos and Prolog tools: scanner, syntax Checker, Static semantic Analyzer, Prolog code generator, Lotos Simulator

- Example: Describing and Simulating Futurebus+ system

Estelle Lotos SDL are formal specification languages to specify OSI protocol and services, assuring:
- correct specification
- and implementation

Describe the "logical" behaviour of system:
- order of interactions
- allowed parameter values

Properties relating to performance aspects are not addressed:
- maximum throughput of a link
- end-to-end transmission delay
- system degradation due to error-rate

To do this, the specification language must include the description of such things as
- time delays
- resource usage
- stochastic behaviour

This is the object of the work presented in the following.
Other approaches are to create a different formalism to deal with performance issues, loosing some "logical" properties of the system.

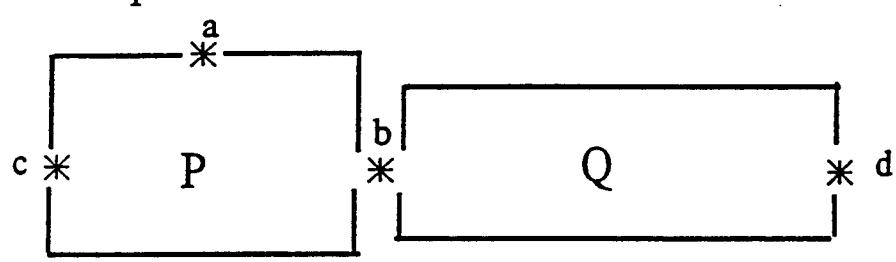
VERILOG, MODSIM, ...

LOTOS (Language of Temporal Ordering Specification) '81-'86
based on CCS (Milner's work)

for formal specification of open distributed systems,
in particular for OSI computer network architectures
→ DAQ HEP exp. ?

Systems are described in terms of *processes* that may consist of several interacting subprocesses

A *processor* is like a *black box* that is capable of communication with its environment by means of interactions.
An *event* (atomic form of interaction) is a unit of synchronized communication that may exist between two processes that can both perform that event.



Two interacting processes

process <process-identifier><parameter-list> :=
<behaviour-expression>
endproc

behaviour-expresion is the LOTOS expression defining the observable behaviour of the process. The contents of the parameter-list qualifies the type of the process.

process P[a,b,c] := ...**endproc**

Basic Lotos: only describes a process synchronization
Full Lotos: also describes interprocess value communication

Behaviour expressions in **Basic Lotos** formally relate the order in which events may occur

Constructor name	SYNTAX
inaction	stop
action prefix	
- unobservable (internal)	i;B
- observable	g;B
choice	B1 [] B2
parallel composition	
-general case	B1 |[g1,...,gn]| B2
-pure interliving	B1 ||| B2
-full syncronization	B1 || B2
hiding	hide g1,...,gn in B
process instantiation	p [g1,...,gn]
successful termination	exit
sequential composition	B1 >> B2
disabling	B1 [> B2

Table 1. Syntax of behaviour expression in basic LOTOS

A basic process: **inaction** stop

this process cannot perform any event

Two basic operators:

Action prefix: this construct produces a new behaviour expression out of an existing one by prefixing it with an event name. If B is the existing expression and a is the event name the result is written as a;B

i ; B

a ; B

Interpretation: the process offers participation in event a; if a takes place the resulting behaviour is given by B. The operator ';' implies sequentiality

unobservable

observable

Choice: if B1 and B2 are existing behaviour expression then B1 [] B2 denotes a process that behaves either like B1 or B2. If another process in the environment offers an initial event of B1 (or B2) the B1 (or B2) may be selected. If both the outcome is not determined

B1 [] B2

Recursion: to define the infinite behaviour.

Example, a buffer definition:

```
processbuffer[in_data,out_data]:=
                in_data
            ;   out_data
            ;   buffer[in_data,out_data]
endproc
```

Parallelism:

→ of independent processes (pure interleaving)

the constructor B1 ||| B2 means that the processor can partecipate in an event if B1 or B2 can partecipate. B1 ||| B2 refuses partecipation in any event that is refused by both B1 and B2.

→ of dependent processes

B1 || B2 can partecipate in an event only if both B1 and B2 can partecipate in it. The general case of the constructor is B1 | [g1, ... ,gn] | B2 were g1, ... ,gn are the common events that B1 and B2 must synchronize

→ the hiding operator

to define subprocessors synchronization without interference with the environment

hide $g_1, \dots, g_n$ in B

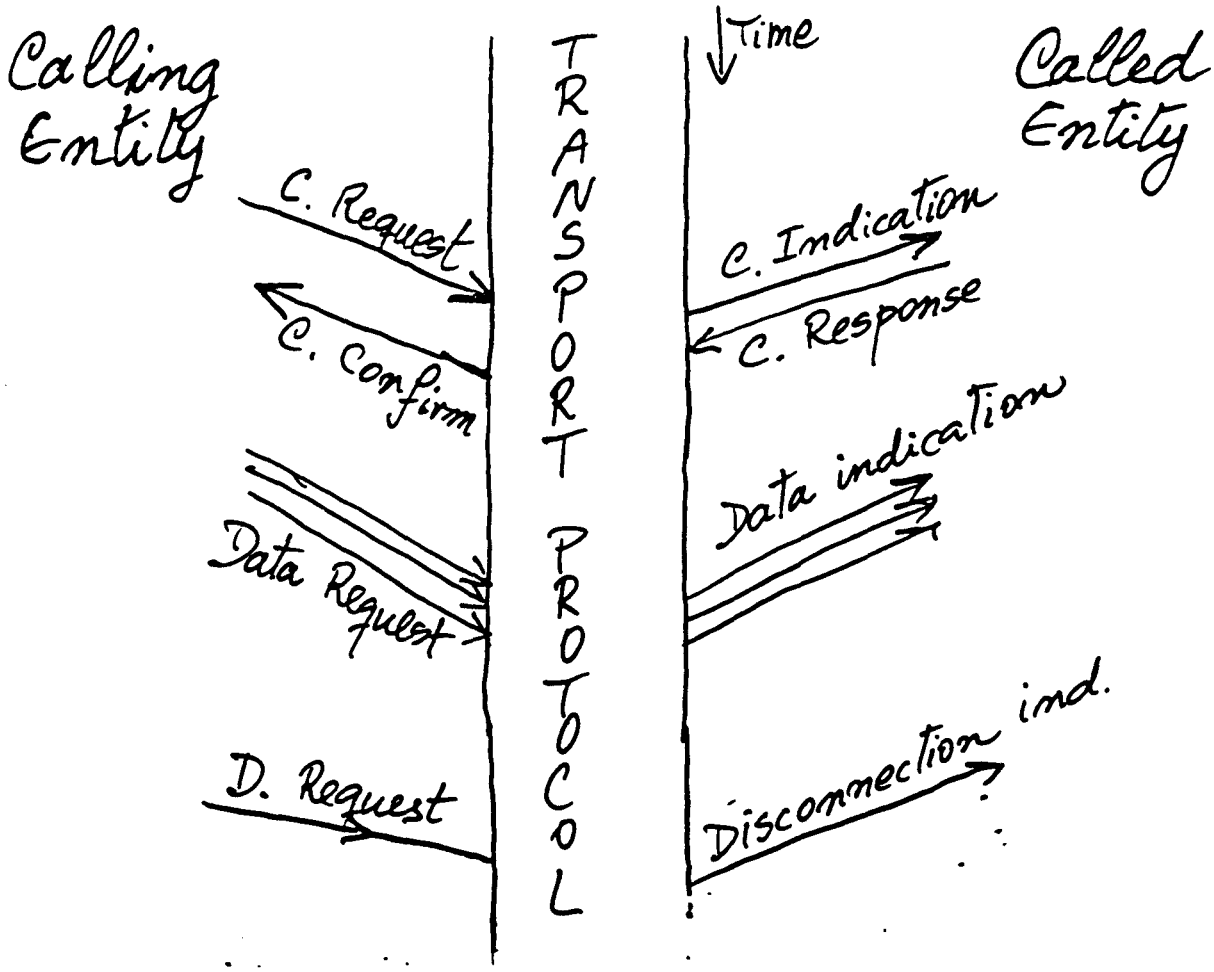
Sequential composition of processes:

B1 >>B2 If B1 terminates successfully, the execution of B2 is enabled

Disruption of processes:

B1 [> B2 defines a processor where at each point in the execution of B1, an initial event of B2 can occurs. If such an event occurs control is transferred to B2, leaving B1.

Manager of the Transport Service in the Session Protocol



Connection
Data transfer
Disconnection

Example on Basic Lotos ⟹

```
process Handler [CReq, CInd, CRes, CCnf, DReq, DInd, DisReq, DisInd]
  Connection_Phase [CReq, CInd, CRes, CCnf, DisReq, DisInd]
   >> ( Data_Phase [DReq, DInd]
        [> Termination_Phase [DisReq, DisInd] )
   >> Handler [CReq, CInd, CRes, CCnf, DReq, DInd, DisReq, DisI
where
    process Connection_Phase [CRq, CI, CR, CC, DR, DI] :=
      (i; Calling [CRq, CI, CR, CC, DR, DI]
      [] Called [CRq, CI, CR, CC, DR, DI] )
      where
        process Calling [CRq, CI, CR, CC, DR, DI] :=
              CRq; (CC; exit
                    [] DI; Connection_Phase [CRq, CI, CR, CC, DR, DI]
                    )
        endprocess
        process Called [CRq, CI, CR, CC, DR, DI] :=
              CI; (i; CR; exit
                    [] i; DR; Connection_Phase [CRq, CI, CR, C
                                                DR, DI
        endproc
    endproc (*Connection_Phase*)
    process Data_Phase [DReq, DInd] :=
          i; DReq; Data_Phase [DReq, DInd]
          [] DInd; Data_Phase [DReq, DInd]
    end process
    process Termination_Phase [DisR, DisI] :=
          i; DisR; exit
          [] DisI; exit
    endproc
endproc (*Handler*)
```

LOTOS data types, dealing with the description of data structures and value expressions, inspired by the abstract data type technique ACT ONE, with some differences

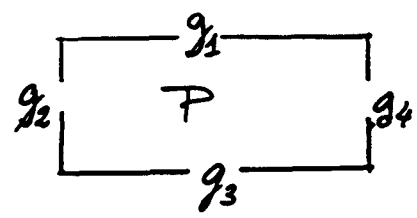
Signature: the names of data carriers (*sorts*) and operations. Defines the syntax of a data type. A grammar can be provided to produce signatures with a finite number of *sorts* and *operations*

```
type Boolean is

sorts Bool

opns true,false              :          ->Bool
   not                      :Bool      ->Bool
   _and_,_or_,_xor_,_implies_,
   _iff_,_eq_,_ne_          :Bool,Bool->Bool

eqns forall x,y :Bool
   ofsort Bool
   not(true)     = false ;
   not(false)    = true  ;
   x and true    = x     ;
   x and false   = false ;
   x or true     = true  ;
   x xor false   = x     ;
   x xor y       = (x and not(y)) or (y and not(x)) ;
   x implies y   = y or not(x) ;
   x iff y       = (x implies y) and (y implies x)  ;
   x eq y        = x iff y  ;
   x ne y        = x xor y  ;
endtype
```

Full LOTOS

Provides interprocess communication

While in Basic LOTOS an observable action coincides with a gate name, in Full LOTOS (or LOTOS) it is formed by a gate name followed by a list of zero or more values offered at that gate



Example:

g_1<TRUE, "tree", 3>

is the observable action offering the boolean value TRUE, character string "tree", and natural number 3 at gate g_1

Since the offered values may range over infinite sets, an infinite number of observable actions is expressible in full LOTOS

To work with most complex data type definitions,

The parametrization: using a general structure of a data type to define other *sorts*. For instance **arrays, stacks** and **queues** can be defining using the **Nat_numbers** definition

The renaming: where an already defined abstract data type is needed in a specific environment, but without any changes in the intended semantics.

F u l l L O T O S

Structured event offers: consist of a *label* or (event) *gate name* identifying the point of interaction or (event) gate and a finite list of attributes:

→ value declaration: !(3+5), !min(x,y)

Let **B1** and **B2** be behaviour expressions and **e** an event then by B1 - e -> B2 we mean taht B1 can transform into B2 performing e.

g!E;B means g<value(E)> ->B

→ variable declarations: *?x:t* where *x* is the name of the variable and *t* is the sort identifier

?x:integer

g?x:t describes a set of events in the value domain *t*

The scope of a variable *x* is the behaviour expression following that event offer in an action prefix construction

Types of interaction

Interaction between processes can take place if both processes have enabled one or more identical events

process A	process B	sync. condition	interaction sort	effect
g!E1	*g!E2*	*value(E1) = value(E2)*	*value matching*	*synchronizati*
g!E	*g?x:t*	*value(E) ∈ domain(t)*	*value passing*	*after synchronization x=value(E)*
g?x:t	*g?y:u*	*t=u*	*value generation*	*after synchronization x=y=v for v ∈ domain(t)*

? input
! output

Conditional constructs

→ Selection predicates. An event offer may be followed by a predicate that can impose additional restrictions of the values

g?x:nat[x<max] ; B1(x)

→ Guarded expressions. Any behaviour expression may be preceded by a predicate; if the predicate holds the behaviour is possible

[x>0] ->g!x;some-process(...)(x,..)

LOTOS tools

ESPRIT Software Thecnology project
SEDOS (Software Environment for the Design of Open Distributed Systems) to develop formal descriptions techniques and related tools. Syntax + Semantic

Regarding LOTOS,

-BELASI (Behavioural Language Simulator) Univ. of Twente
* Simulation tool implementing the LOTOS semantic
* only rudimentary support of the Abstract Data type part
* written in C-Prolog, a LOTOS specification must be translated to a C-Prolog
* they found limits due to the implementation

it was superseded by HIPPO

- HIPPO (Lotos Simulator) in C-language
* generation of an Abstract Data Type term rewrite system
* evaluation of an Abstract Data Type term rewrite system
* communication tree walker
* menu computation and selection
* term analysis and display
* state information display at each step in a communication tree

future work on: coverage and performance analysis, human interface. property checking (deadlock, livelock, reachibility, ...)

Distribution to non-profit organ. by University of Twente

The introduction of timing in LOTOS

to achieve realistic description of systems which exhibit time dependent behaviour

- Timed-Action proposed by Quemada and Fernandez
- Timed-Interaction proposed by Bolognesi, Lucidio and Triglia
- Simple-Timed proposed in this presentation

Grammar of Timed-Action LOTOS behaviour expression

```
behaviour-expression ::=
    stop
|   gate-id [time-interval] ; behaviour-expression
|   behaviour-expression [] behaviour-expression
|   behaviour-expression | gate-id-list | behaviour-expression
|   [old(time,behaviour-expression)]
|   process-id gate-id-list.

process-id      ::=  id.
gate-id         ::=  id.
gate-id-list    ::=  [ ]
                 |   [gate-id {,gate-id} ].
time-interval   ::   <time,time>
time            ::   natural.
```

Behaviour expression g;B is now defined as g<time1,time2>;B

Basic LOTOS

Bolognesi et al. ...

Grammar of Timed-Interaction LOTOS behaviour expression

```
behaviour-expression    ::=
      stop
|     gate-id ; behaviour-expression
|     behaviour-expression   [] behaviour-expression
|     behaviour-expression   | gate-id-list   | behaviour-expression
|     old(time,behaviour-expression)
|     process-id  gate-id-list
|     timer gate-id  time-interval  in behaviour-expression
```

No time interval is explicity associated with action prefix. Such intervals are instead associated with the 'timer' operator, that is applied to any behaviour expression.
The model follows the Fernandez and Quemada work . The intent was the modelling of the "wait-until-timeout" scenario in the basic-LOTOS environment

We think that if we consider the global LOTOS not only the modelling of this scenario is possible without the introduction of the 'timer' operator, but the syntax is also more simple

Grammar of Simple-Timed LOTOS behaviour expression

```
behaviour-expression    ::=
    stop
|   gate-id [time] ; behaviour-expression
|   [guard-mode -> behaviour-expression]
|   behaviour-expression   [] behaviour-expression
|   behaviour-expression   | gate-id-list   | behaviour-expression
|   process-id  gate-id-list.

time            ::   <natural>.
```

The time (not a time interval) is introduced in the action prefix. This time is the instant in which the action is hold. The introduction of the logical function 'delta' in the guard-mode is able to model the "wait-until-timeot" scenario.

In LOTOS syntax, an example (from the Futurebus+ simulation)

```
power_system1   [gv] (timeP:time)[redacted] :=
        [delta(timeP,tdel)] --> i<Tim_i>;
                power_system2 [gv] (Tim_i)
```

Tim_i is the timeout function with an argument Tim_i

Simple-Timed LOTOS implement the standard LOTOS, extending with time the LOTOS syntax and semantic, and giving to the guard-mo the ability to call PROLOG clauses.

PROLOG ⟺ LOGIC PROGRAMMING

LOGICAL INFERENCE:

given a fact (or premise) A and the rule of inference if A then B we deduce the fact (or conclusion) B

Facts and rules are referred to as clauses

I should drive to work if — head=conclusion
it is cold and
it is raining — body=premises

Prolog programs allow answer such questions as "Should I drive to work on tuesday?"

Proposition or predicate, arguments determine if the proposition is true or false

```
drive_to_work(tuesday) :-
    it_is_cold(tuesday),
    it_is_raining(tuesday).
```

The proposition

```
drive_to_work(tuesday).
```

represents in English language the query "Should I drive..."

Unification: similar Prolog terms are matched with one another. This allows to write programs that include variables.

```
driver_to_work(Day) :-
    it_is_cold(Day),
    it_is_raining(Day).
```

Variables: Uppercase to be instantiated

If the database contains the facts

```
it_is_cold(tuesday).        equivalent to
                            it_is_cold(tuesday):=true

it_is_raining(tuesday).
```

the system will answer "yes" to the query

```
|?- drive_to_work(tuesday).
yes
|?- it_is_cold(Day).
Day = tuesday;
- - - - - - - -
```

semicolon means request of Backtracking. The system will look for another fact matching the query and will give different answers until

```
no
|?-
```

A fundamental unit of a Logic Program is the procedure call, that can consist of a sequence of <u>sentences</u> containing a <u>head</u> and a <u>body</u>

```
employed(X) :- employs(Y,X).
```

<u>sentence</u>

"To find whether a person X is employed, find whether any Y employs X"

"Any X is employed if any Y employs X"

list can be expressed by [a|b] indicating with a the <u>head</u> and with [b] the <u>tail</u> of the list.

procedure

```
concatenate ([], L, L).
concatenate ([X|L1], L2, [X|L3]) :-
        concatenate (L1, L2, L3).
```

"the list L1 concatenated with the list L2 is the list L3"

Disjunction |

```
grandfather (X, Z) :-
        (mother (X, Y)
        | father (X, Y)),
        father (Y, Z).
```

when Y is instantiated its value remains for all the proposition body.

Cut !

when first encountered, cut succeeds immediately. If backtracking should later return to the cut the head of the proposition will fail.

```
member (X, [X|L]).
member (X, [Y|L]) :- member (X, L).
```

can be used to test if a given term is in a list

```
|?- member (b, [a, b, c]).
yes
|?- member (X, [d, e, f]).
| X = d ;
| X = e ;
| X = f ;
no
```

extract elements from a list

```
member (X, [X|L]) :- !
|?- member (X, [d, e, f]).
| X = d ;        no
```

Conditional goals:

P -> Q ; R If P then Q else R
P -> Q | R If P then Q or R

```
repeat,
  action1,
  action2,
  :
  actionn,
  test,
!
```

if a failure of the test, the backtracking will be caught by repeat and re-execute the actions.

Comparison of terms:

T1==T2
T1\==T2
T1@<T2
T1@>T2
T1@<=T2
T1@>=T2

compare (op, T1, T2)
sort (L1, L2)

To modify the database
assert (clause).
retract (clause).

...
...

allows new definitions of operators.

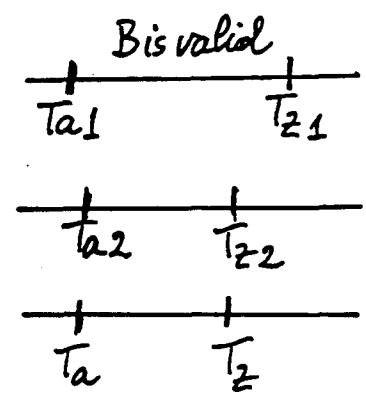
The Simple-Timed LOTOS tools are written in Quintus-Prolog* (we started using C-Prolog).

Implement the standard LOTOS, extending with time the LOTOS syntax and semantic, and giving to the guard-mode the ability to call Prolog clauses.

Semantic of Simple-Timed LOTOS in Prolog, one example:

Guard Mode

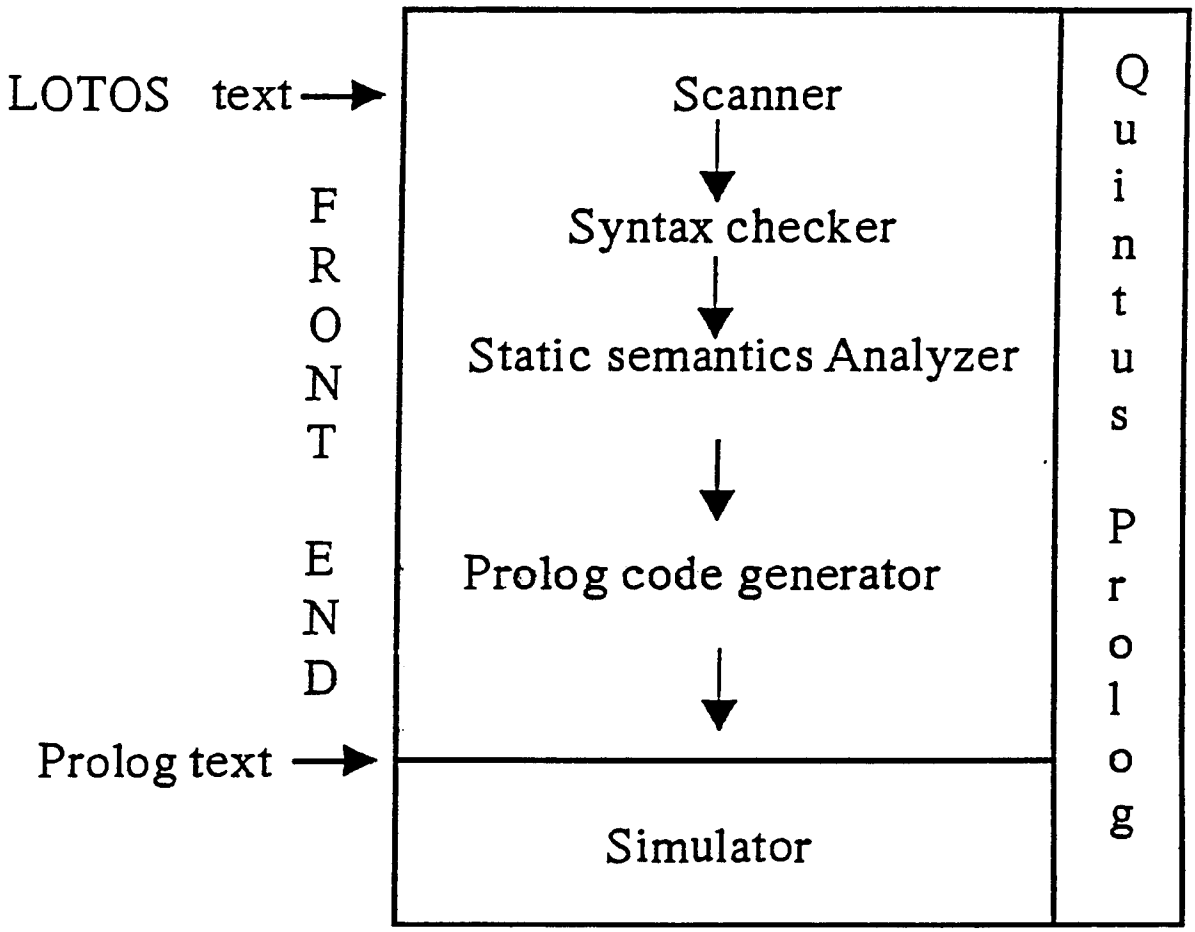
```
der(B-->P,A,Q,[Ta,Tz]):-!,
    valid(B,[Ta1,Tz1]),
    der(P,A,Q,[Ta2,Tz2]),
    maxtim(Ta1,Ta2,Ta),
    mintim(Tz1,Tz2,Tz),
    ge(Tz,Ta).
```



B is the guard-mode
P is the behaviour if B is true
A is the action
Q is the new behaviour

*Quintus-Prolog: interpreter (with debug)
compile
interface to C-language

Major toolset componets



Scanner accept an ASCII text file containing ordinary LOTOS text and output a list of Tokens that is the input to the **Syntax Checker**

The Syntax Checker

Prolog's grammar rule are used.
A grammar rule in Prolog takes the general form

head --> body

meaning "a possible form for head is body". Both body and head are sequences of one or more items linked by the standard Prolog conjunctor operator ',' and disjunctor operator '|'.

Extra conditions, in form of Prolog procedure calls, may be included in the righ end side of a grammar rule. Such procedure calls are written enclosed in curly brackets ('{' and '}').

```
data_type_definition(Type)      -->
    type_symbol,
    type_identifier(TypId),
    is_symbol,
    p_expression(Pexpr),
    end_type_symbol,
        {getsym(type,Type,[type,TypId,is,Pexpr,endtype])}
  | library_declaration(LibDecl),
        {getsym(type,Type,LibDecl)}.

p_expression(Pexpr)    -->
.....
```

getsym is used to produce the output of the Abstract Syntax tree, adding a new clause into the Database and giving a pointer type

The Static Semantic Analyzer

This tool reads a coloured Abstract Syntax Tree as produced by the Syntax Checker, and performs the combined functions of static semantic checking, Abstract Data Type library insertion, and Abstract Data Type flattening.

solve the internal references between data types.

The Prolog Code Generator

Some examples:

```
sorts(true,bool,boolean).
sorts(false,bool,boolean).

sorts(not(X),bool,boolean):-
   sorts(X,bool,boolean).

sorts(X and Y,bool,boolean):-
   sorts(X,bool,boolean),
   sorts(Y,bool,boolean).

....
```

The signature of Boolean data type in Prolog

```
rewrite_rule(boolean,bool,not(true),    false).
rewrite_rule(boolean,bool,not(false),   true).

rewrite_rule(boolean,bool,X  and true,  X):-
    sorts(X,bool,boolean).

rewrite_rule(boolean,bool,X  implies  Y, Y or not(X)):-
    sorts(X,bool,boolean),
    sorts(Y,bool,boolean).
```

The equation of Boolean data type in Prolog

Simulator

The Simulator is a single tool which perform the following functions:

* evaluation of an Abstract Data Type term rewrite system,
* communication tree walker,
* menu computation and selection,
* term analysis and display,
* state information display at each step in a communication tree.

as HIPPO ?

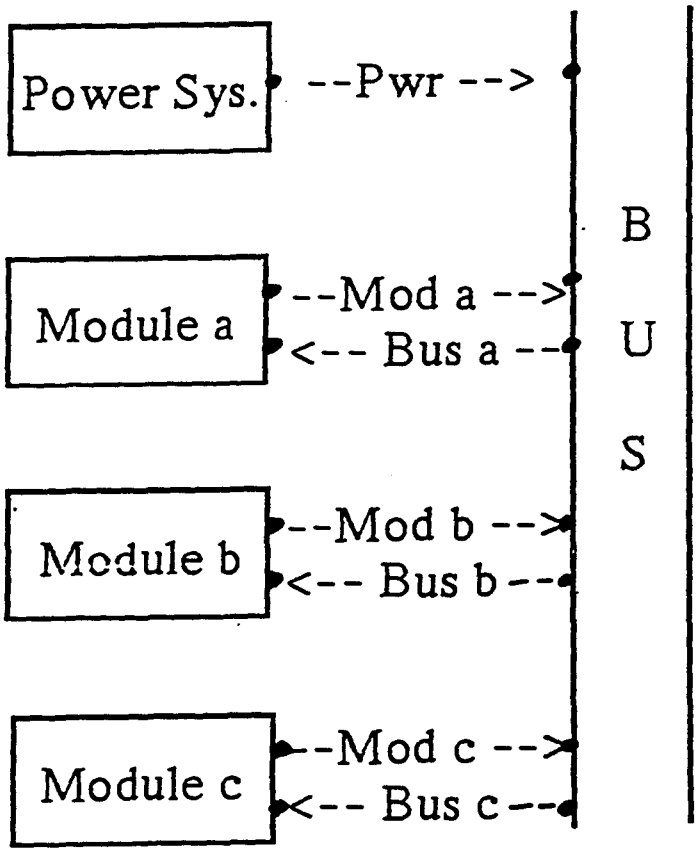
It accepts to its input the Prolog code of LOTOS text and Prolog procedure called in guard mode from LOTOS text. This feature i an useful and powerful extension of LOTOS.

Simulation based on Logical Layer Specification of Futurebust. Arbitration Competion Logic and the Bus Control: Powerup, System Reset, Bus Initialise, Live Insertion and Live Withdrawal.

Bus	Signal	Module		32-bit version
			INFORMATION LINES	
	<--AD->		AD Address/Data	32
	<--BP->		BP Byte Parity	4
	<--TG->		TG Tag	8
	<--TP->		TP Tag Parity	1
	<--CM->		CM Command Field	8
	<--CP->		CP Command Parity	1
	<--ST->		ST Status	8
	<--CA->		Ca Capability	3
			SYNCHRONIZATION LINE	
	<--AS->		AS Address Sync	1
	<--AK->		AK Address Acknowledge	1
	<--AI->		AI Address Acknowledge Inverse	1
	<--DS->		DS Data Sync	1
	<--DK->		DK Data Acknowledge	1
	<--DI->		DI Data Acknowledge Inverse	1
			CONTROL ACQUISITION LINE	
	<--AB->		AB Arbitration Bus	8
	<-ABP->		ABP Arbitration Parity	1
	<--AP->		AP Arbitration Synchronization	1
	<--AQ->		AQ Arbitration Synchronization	1
	<--AR->		AR Arbitration Synchronization	1
	<--AC->		AC Arbitration Condition	2
			RESET/BUS INITIALIZE LINE	
	<--RE->		RE Reset/Bus	1

Simulation takes into account the Time parameter

The Futurebus+ model with three Modules



Mod a
Bus a
⋮
are gates

Elementary functions of the Futurebus+ example

```
modW is   Mod <Time> ~Sg
modR  is  Bus <Time> ?Sg:sgn [get_bus(Mod,Bus,Sg,Time)]

busW is   Bus <Time> ~Sg
busR  is  Mod <Time> ?Sg:sgn [wor(Sg,Time,CH)]
```

get_bus and wor are Prolog Clauses defined in the following

```
get_bus(Mod,Bus,Sg,Time):-
  lines(bus,Sg,Sgv,_),
  is_line(Bus,Sg,Sgv,Time),
  bus_control_attributes(Mod,Bus,Sg).


wor(Signal,Time,CH):-
  lines(modA,Signal,SGa,_),
  lines(modB,Signal,SGb,_),
  lines(modC,Signal,SGc,_),
  worb(SGa,SGb,SGc,SGbus),
  is_line_CH(bus,Signal,SGbus,CH,Time),!.
```

~ offer event value (like ! in LOTOS)

CH yes no indicating if the signal changes

THE BUS Description in LOTOS

```
bus([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[]):=
  busR(Pwr,_,SigRE,TimeRE,CH);
     bus1([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[SigRE,TimeRE,CH])
'[]'
  busR(Moda,Busa,Siga,Timea,CH);
     bus1([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[Siga,Timea,CH])
'[]'
  busR(Modb,Busb,Sigb,Timeb,CH);
     bus1([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[Sigb,Timeb,CH])
'[]'
  busR(Modc,Busc,Sigc,Timec,CH);
     bus1([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[Sigc,Timec,CH]).


bus1([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],
   [Sig:sgn,Time:time,CH:yesno]):=
    ([CH=yes(_Sg1),!]-->delay([Busa,Busb,Busc],[Sig,Time])
     '[]'
     [CH=no(_Sg2),!]-->stop)
       intl
     bus([Pwr,Moda,Modb,Modc,Busa,Busb,Busc],[]).

delay([Busa,Busb,Busc],[Sig:sgn,Time:time]):=
   [delta(Time,time5)]-->busW(_Moda,Busa,Sig,_Timea);stop
      intl
   [delta(Time,time6)]-->busW(_Modb,Busb,Sig,_Timeb);stop
      intl
   [delta(Time,time7)]-->busW(_Modc,Busc,Sig,_Timec);stop .
```

dependent from the module distances

The LOTOS Program of the Futurbus+ model

```
start([],[]):=
   bus([pwr,modA,modB,modC,busA,busB,busC],[])
    lpar [pwr,modA,modB,modC,busA,busB,busC] rpar
   power_start([pwr,modA,modB,modC,busA,busB,busC],
          [[0,0,0,0],[0,0,0,0],[0,0,0,0],[1,0,0,0]]).
          sec,msec,μsec,ns

power_start([Pwr,ModA,ModB,ModC,BusA,BusB,BusC],
        [TimeP:time,TimeA:time,TimeB:time,TimeC:time]):=
   power_system1([Pwr],[TimeP])
    intl
   module([ModA,BusA],[TimeA])
    intl
   module([ModB,BusB],[TimeB])
    intl
   module([ModC,BusC],[TimeC]).
```

Powerup

When a system is first powered, the power system assert the reset signal re*. The power system must assert re* prior to any of the bus power voltages reachingh 40% of their nominal value (this physical condition is not rapresented in the simulation of the logical layer). The power system mantains re* asserted for 100 to 200 msec after a the bus power voltages are within regulation. This allows all modules on the bus to detect that a system reset is occurrring.

The power system shall:

1. Prior to any bus voltage reaching 40% of nominal voltage assert re*

2. Maintain re* asserted until all bus voltages are within regulation specification

3. Then continue to assert re* for an additional 100-200 ms

4. Then release re*

Powerup
=======

The Powerup System in LOTOS

```
power_system1([Pwr],[TimeP:time]):=
  [delta(TimeP,time0),set_line(Pwr,'RE',TimeP),!]-->
      modW(Pwr,_,'RE',Time);
  power_system2([Pwr],[Time]).
```

offers the 'RE' signal

```
power_system2([Pwr],[Time:time]):=
  power_system3([Pwr],[Time]).

power_system3([Pwr],[Time:time]):=
  [delta(Time,time100)]-->power_system4([Pwr],[]).

power_system4([Pwr],[]):=
  i(clearRE,Time,[clear_line(Pwr,'RE',Time),!]);
  modW(Pwr,_,'RE',Time);
  stop.
```

clear RE and offers the new value

Live insertion

The logical protocols of Futurbus+ provide facilities for modules which are inserted into a powered operating system to align their logic state machines with the other modules on the bus and begin operation. When a module is inserted into the bus and detects that it has just powered up it checks to see if RE* is asserted. If it is not asserted the module assumes it has been inserted into a live system. If the reset line is asserted it waits to see if is a full system reset and not a bus initialize or live insertion. Eventually the module knows if it has been inserted into a live system.

A module inserted into a live system assert re* to notify all the other modules that it needs to align. All other modules upon detecting RE* asserted should be set their transaction timeout register to limit the remaining lenght of the current parallel bus transaction to 128 usec, then finish the current parallel bus transaction and the current arbitration sequence and then wait. This causes the bus to go into an idle state. The inserted module waits after it has asserted reset until is has detected the bus in a continuus idle state for at least 1 usec and then it asserts ai* and ar* to complete the alignement. It then release re* and all modules resume operation when they detect REf at logic zero.

Live insertion

The modules shall:

1. Perform all internal reset operations necessary to allow partecipation in bus transaction

2. If RE* wait until SYS_RESET / rEf

3. If SYS_RESET, proceed to system reset

4. If rEf wait for a minimum of 1us and then assert re*

5. ...

The Live Insertion System in LOTOS

```
live_insertion1([Mod,Bus],[Time:time]):=
   i(reset,Time1,[bus_control_attributes(Mod,Bus)]);
    live_insertion1a([Mod,Bus],[Time1]).

live_insertion1a([Mod,Bus],[Time:time]):=
  [delta(Time,time45)]-->
 i(wait_reset_complete,_Time1,[chk_RESET_COMPLETE(Mod,Bus)
       (live_insertion2([Mod,Bus],[])
        '[]'
        live_insertion4([Mod,Bus],[])).
```

- - - - -

The bus control procedure in Prolog

```
bus_control_attributes(Mod,Bus,Sig):-
   (Sig='RE',!,bus_control_attributes(Mod,Bus),!);
   (Sig='AS',!,bus_control_attributes(Mod,Bus),!);
   (Sig='AI',!,bus_control_attributes(Mod,Bus),!);
   (Sig='AK',!,bus_control_attributes(Mod,Bus),!).
bus_control_attributes(_Mod,_Bus,_Sig):-!.

bus_control_attributes(Mod,Bus):-
   chk0_SYS_RESET(Mod,Bus),
   chk_BUS_IDLE(Mod,Bus),
   chk_BUS_IDLE_1us(Mod,Bus),
   chk_POWER_UP(Mod,Bus),
   chk_BUS_INIT(Mod,Bus),
   chk_ENTRANT(Mod,Bus),
   chk_LIVE_INSERTION(Mod,Bus),
   chk_BUS_HOLD(Mod,Bus),
   chk_INIT(Mod,Bus),
   chk_LIVE_WITHDRAWAL_COMMAND(Mod,Bus),
   chk_READY_FOR_WITHDRAWAL(Mod,Bus).
```

written in Prolog.

Total simulation uses { 80% PROLOG text
20% LOTOS text

The Trace of Events during a simulation

```
Gate              Time        Event

pwr               <0>         ?RE:sgn [wor(RE,[0,0,0,0],yes([0]))]
busA              <5>         ~RE    [get_bus(modA,busA,RE,[0,0,0,5])]
busB              <6>         ~RE    [get_bus(modB,busB,RE,[0,0,0,6])]
busC              <7>         ~RE    [get_bus(modC,busC,RE,[0,0,0,7])]
i(power_up)       <10>            [set_Attr(power_up,modA),!]
i(power_up)       <10>            [set_Attr(power_up,modB),!]
i(wait_SYS_RESET) <10>
i(wait_SYS_RESET) <10>
i(chk1)           <30:0:5>        [chk1_SYS_RESET(modA,busA)]
modA              <30:0:5>    ?RE:sgn [wor(RE,[0,30,0,5],no([0]))]
i(chk1)           <30:0:6>        [chk1_SYS_RESET(modB,busB)]
modB              <30:0:6>    ?RE:sgn [wor(RE,[0,30,0,6],no([0]))]
i(chk1)           <30:0:7>        [chk1_SYS_RESET(modC,busC)]
i(wait_reset_compl) <30:45:5>     [chk_RESET_COMPLETE(modA,busA)]
modA              <30:45:5>   ?RE:sgn [wor(RE,[0,30,45,5],no([0]))]
i(wait_reset_compl) <30:45:6>     [chk_RESET_COMPLETE(modB,busB)]
modB              <30:45:6>   ?RE:sgn [wor(RE,[0,30,45,6],no([0]))]
i(clearRE)        <100:0:0>       [clear_line(pwr,RE,[0,100,0,0]),!]
pwr               <100:0:0>   ?RE:sgn [wor(RE,[0,100,0,0],yes([1]))]
busA              <100:0:5>   ~RE    [get_bus(modA,busA,RE,[0,100,0,5])]
.......................................

i(chk1)           <1:30:46:17>    [chk1_SYS_RESET(modC,busC)]
i(wait_reset_compl) <1:30:91:15>  [chk_RESET_COMPLETE(modA,busA)]
modA              <1:30:91:15> ?RE:sgn [wor(RE,[1,30,91,15],no([0]))]
i(wait_reset_compl) <1:30:91:16>  [chk_RESET_COMPLETE(modB,busB)]
modB              <1:30:91:16> ?RE:sgn [wor(RE,[1,30,91,16],no([0]))]
i(wait_power_up)  <1:530:91:15>
i(wait_power_up)  <1:530:91:16>
                  sec:msec:usec:ns
```

system reset completed at modules A and B

power_system4

all data required for good presentation

Conclusions:

- Artificial intelligence languages offer new ways to simulation
- LOTOS definition offer the ability to test and validate implementations: deadlock, livelock, ... analysis easier!
- Comparison with VERILOG tool will be provided (*)
- One year/man is still required to complete the whole tool.
- Prolog procedures called in guard-mode from LOTOS text is an useful and powerful extension of LOTOS.

* simulating DAQ system for KLOE experiment at the new ϕ factory in Frascati, that in principle will use Futurebus at the event processor farm level.

Modeling and Simulation of an Event Builder for High Energy Physics Data Acquisition Systems

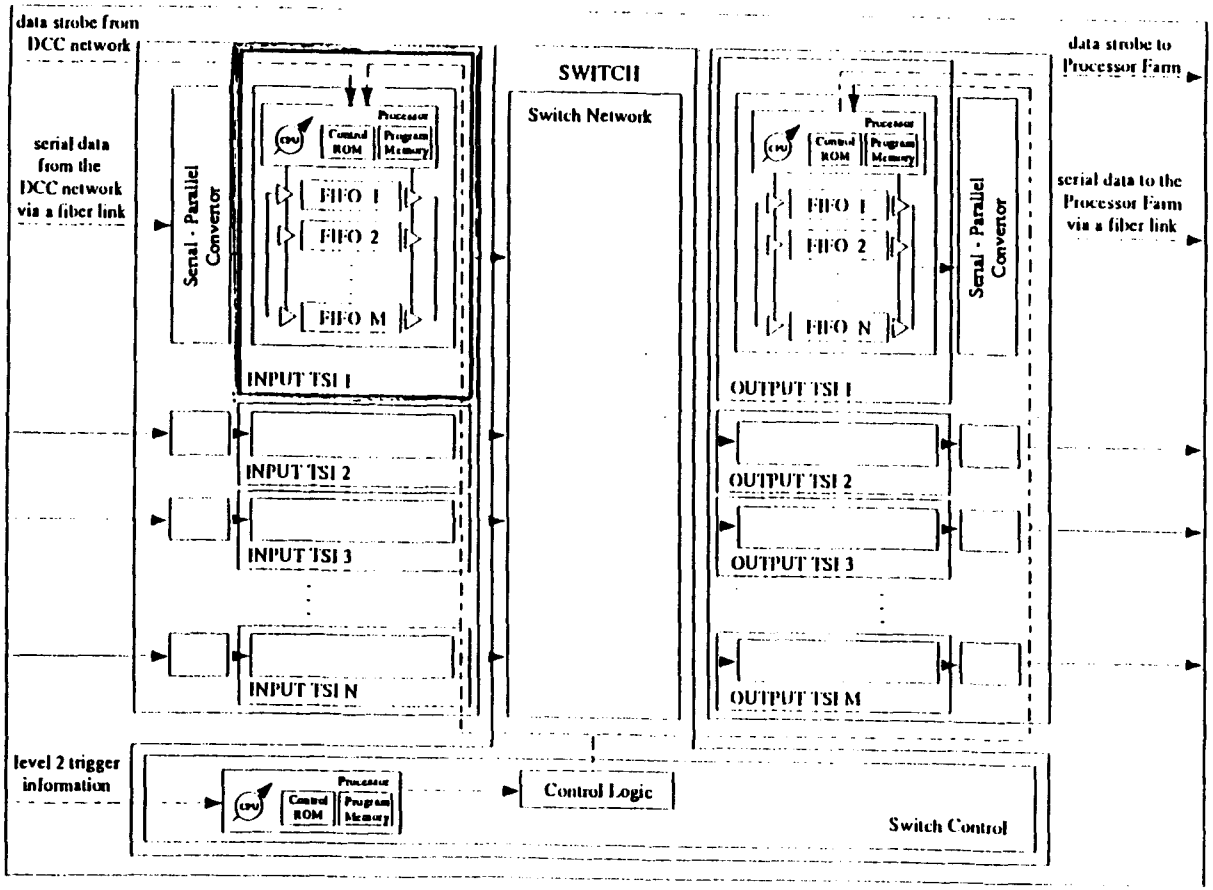
Vishal S. Kapoor

University of Texas at Arlington
Department Of Computer Science Engineering

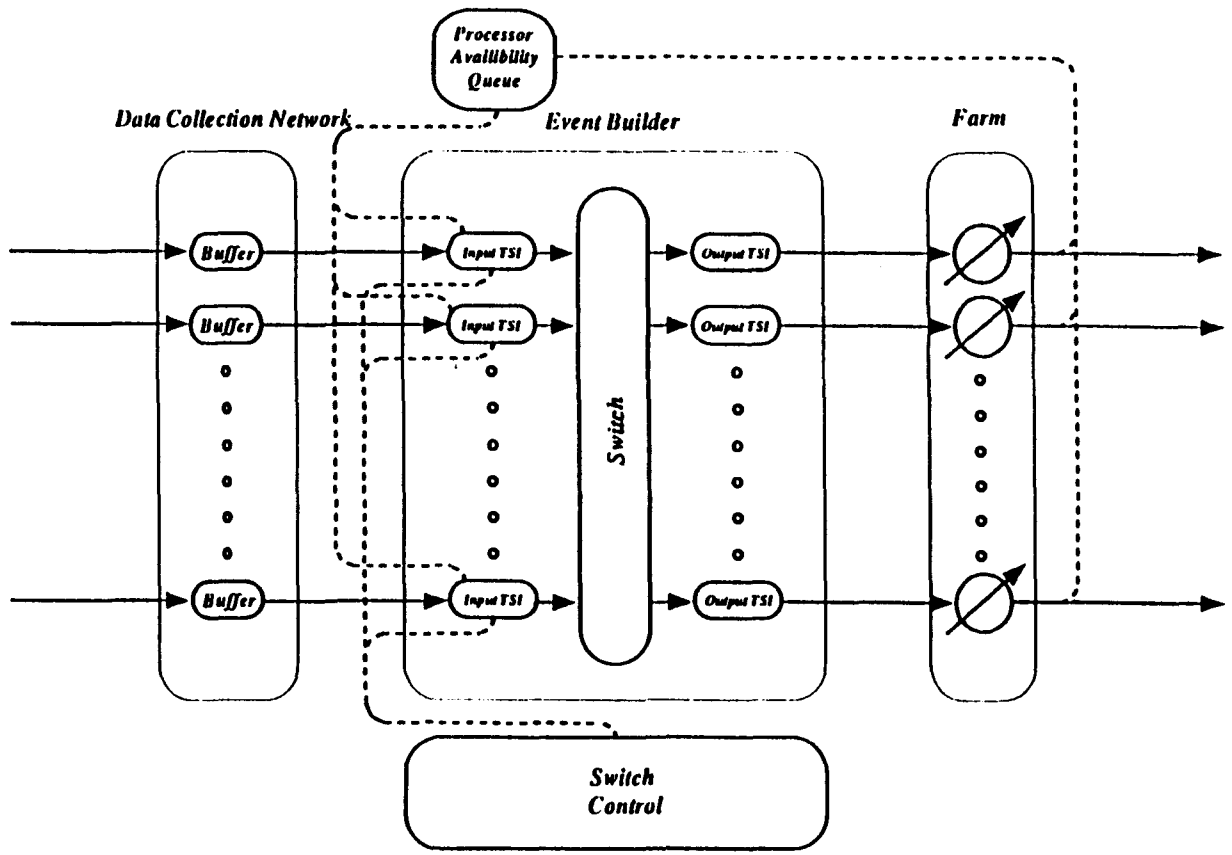
Introduction

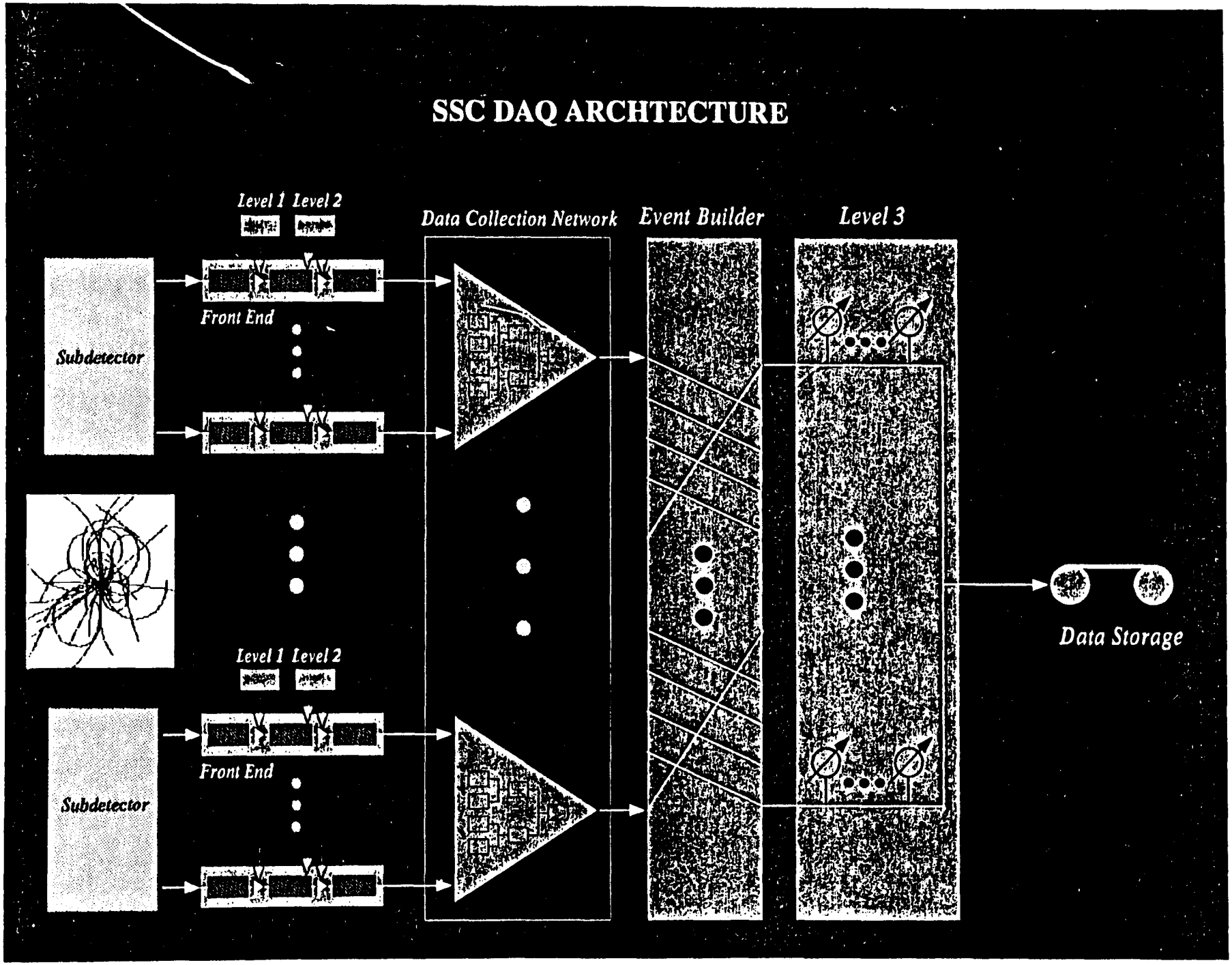
Problem	To specify an event builder for HEP DAQ systems
Challenge	To handle high data rates and volumes
Approach	System Level Modeling and Simulation Gate Level Modeling and Simulation Top Down Design Methodology
Solution	Design of a parallel switching architecture

HARDWARE DESIGN OF THE EVENT BUILDER



PROCESSOR AVAILABILITY FEEDBACK

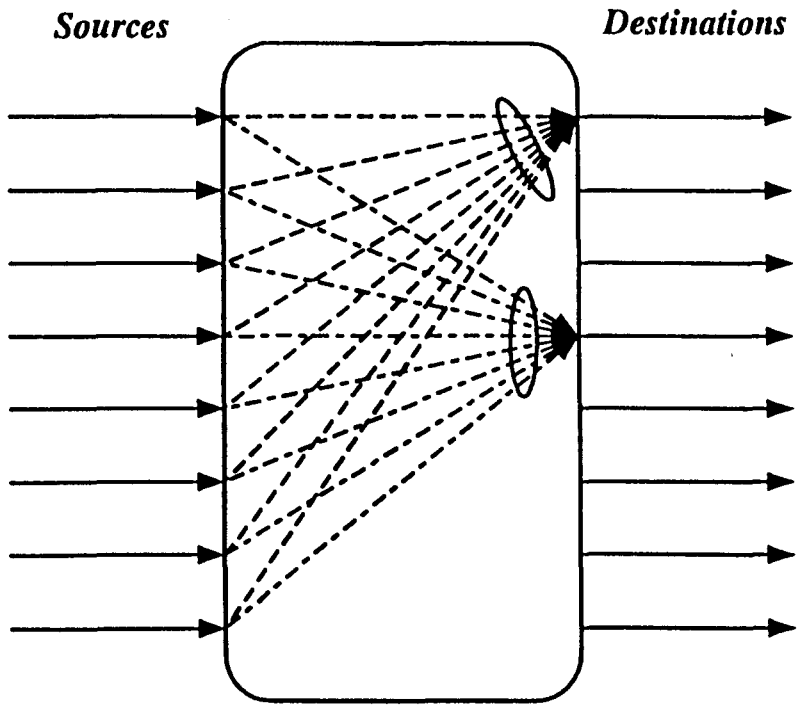



SSC DAQ ARCHTECTURE
Level 1
Level 2
Front End
Subdetector
Data Collection Network
Event Builder
Level 3
Data Storage

HISTORY OF EVENT BUILDING
(70s until the present)

- minicomputer sequential readout
- very little buffering
- CAMAC packaging
- tree-structured data acquisition systems
- FASTBUS packaging
- parallel sub-event building
- experiment triggers disabled while reading
- > 10% deadtimes
- upto 10s of MB/s event building

Event Building

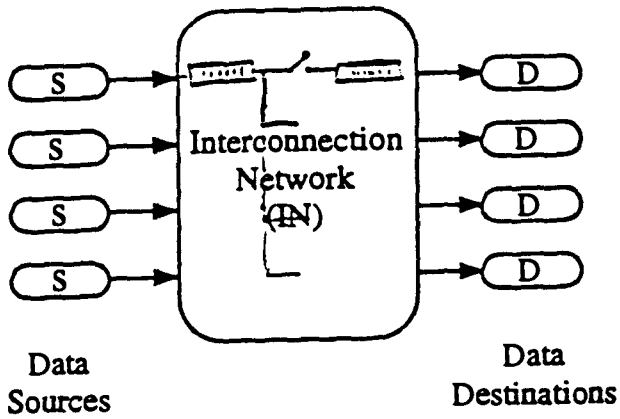


Interconnect Architectures

HEP

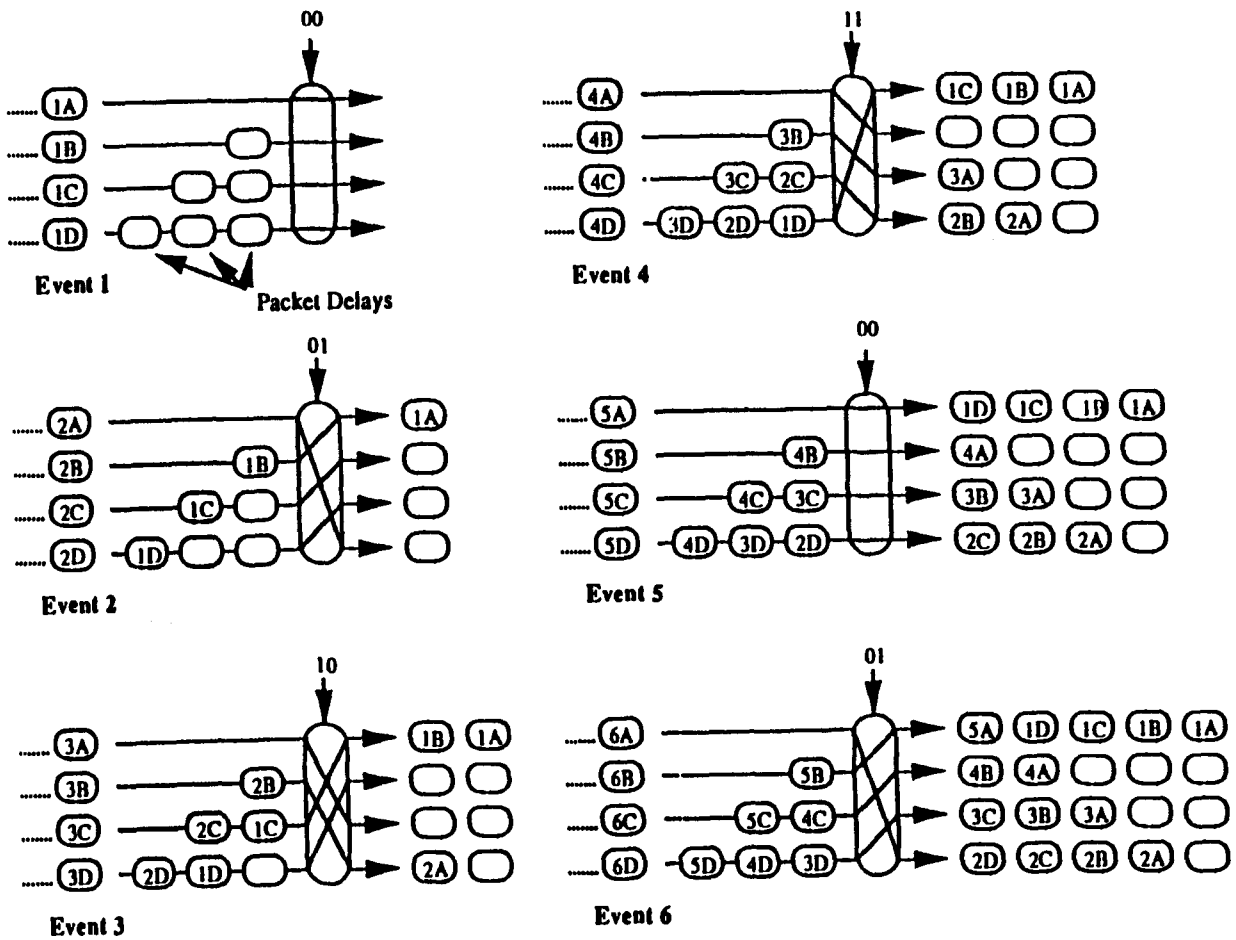
- shared bus
- multiple bus
- multiport memory
- dual port memory
- crossbar interconnect
  (barrel shifter)
- mesh interconnect

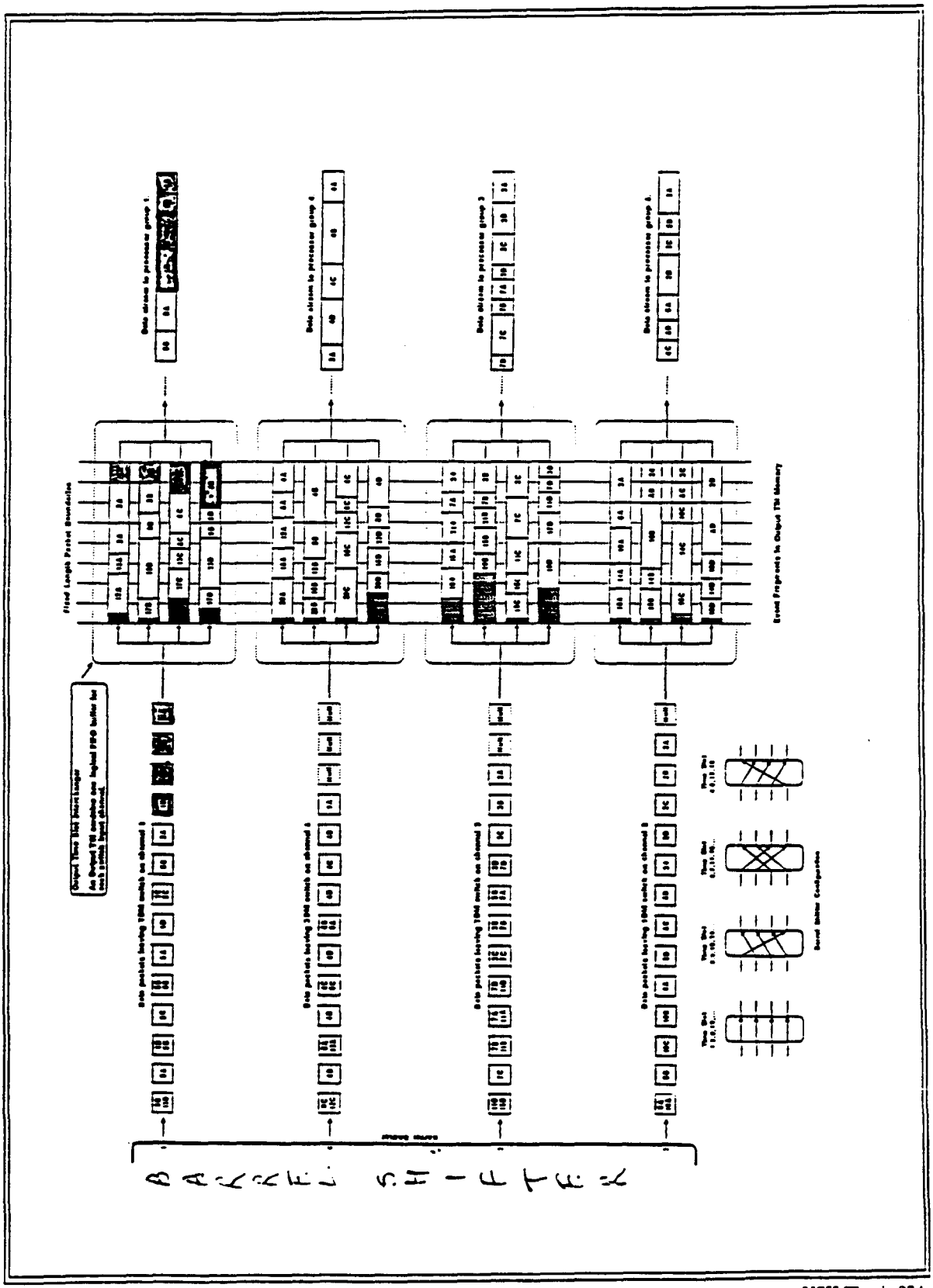
Interconnection Network Terminology

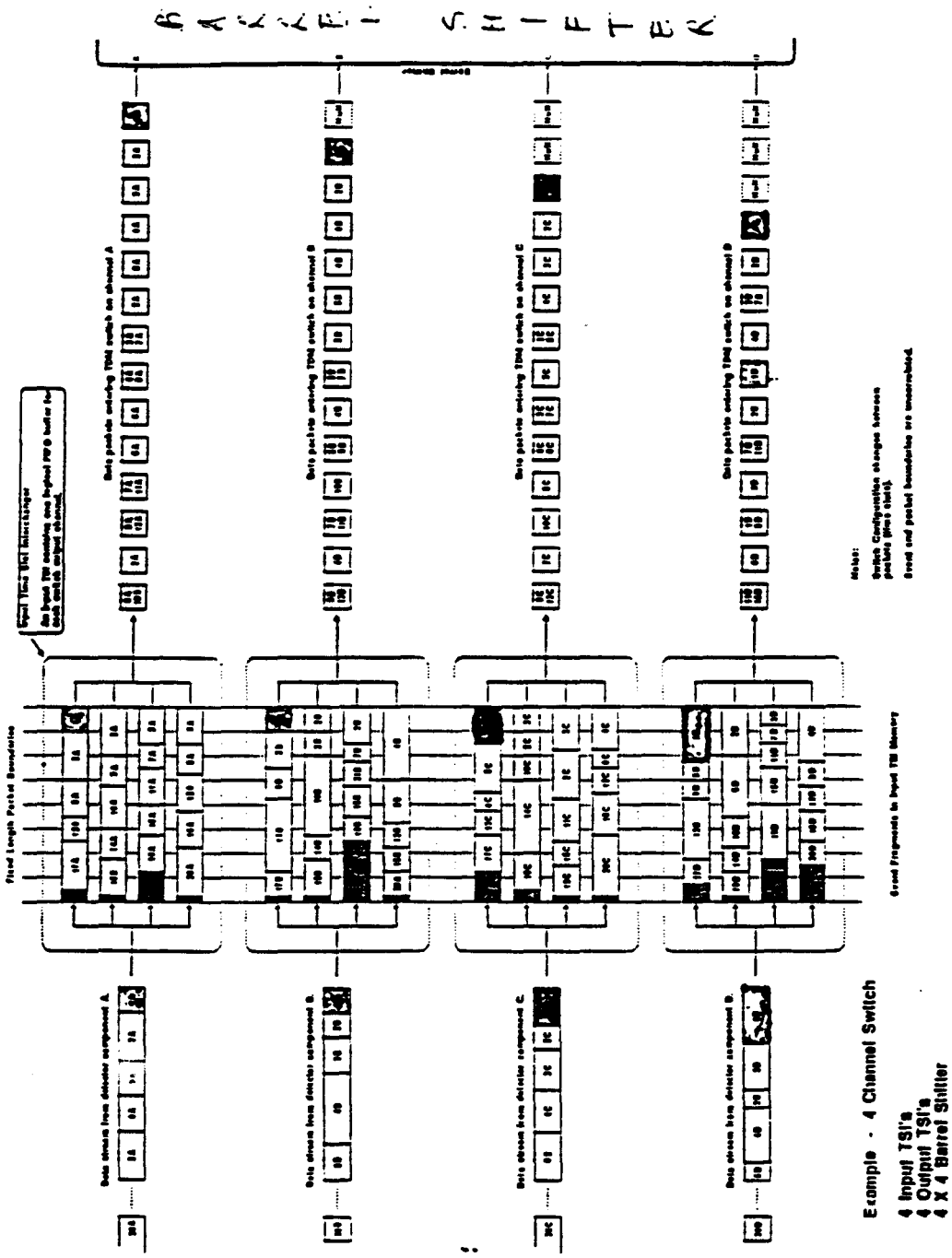


- ✓ Packet Switching
  - Circuit Switching
- ✓ Synchronous
  - Asynchronous
- ✓ Centralized
  - Decentralized
- ✓ Non-blocking
  - Blocking

BARREL SHIFT OPERATION

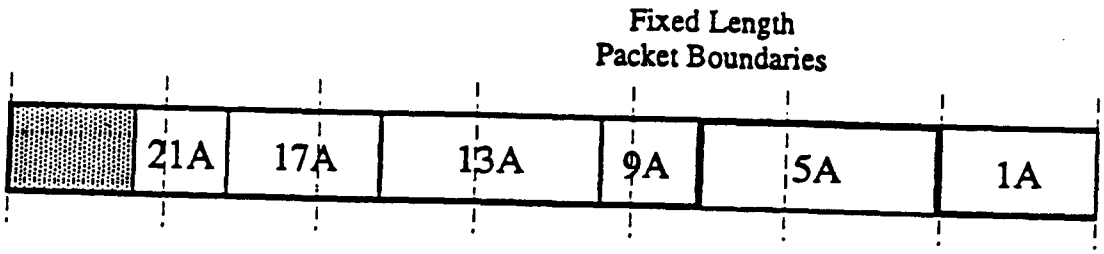




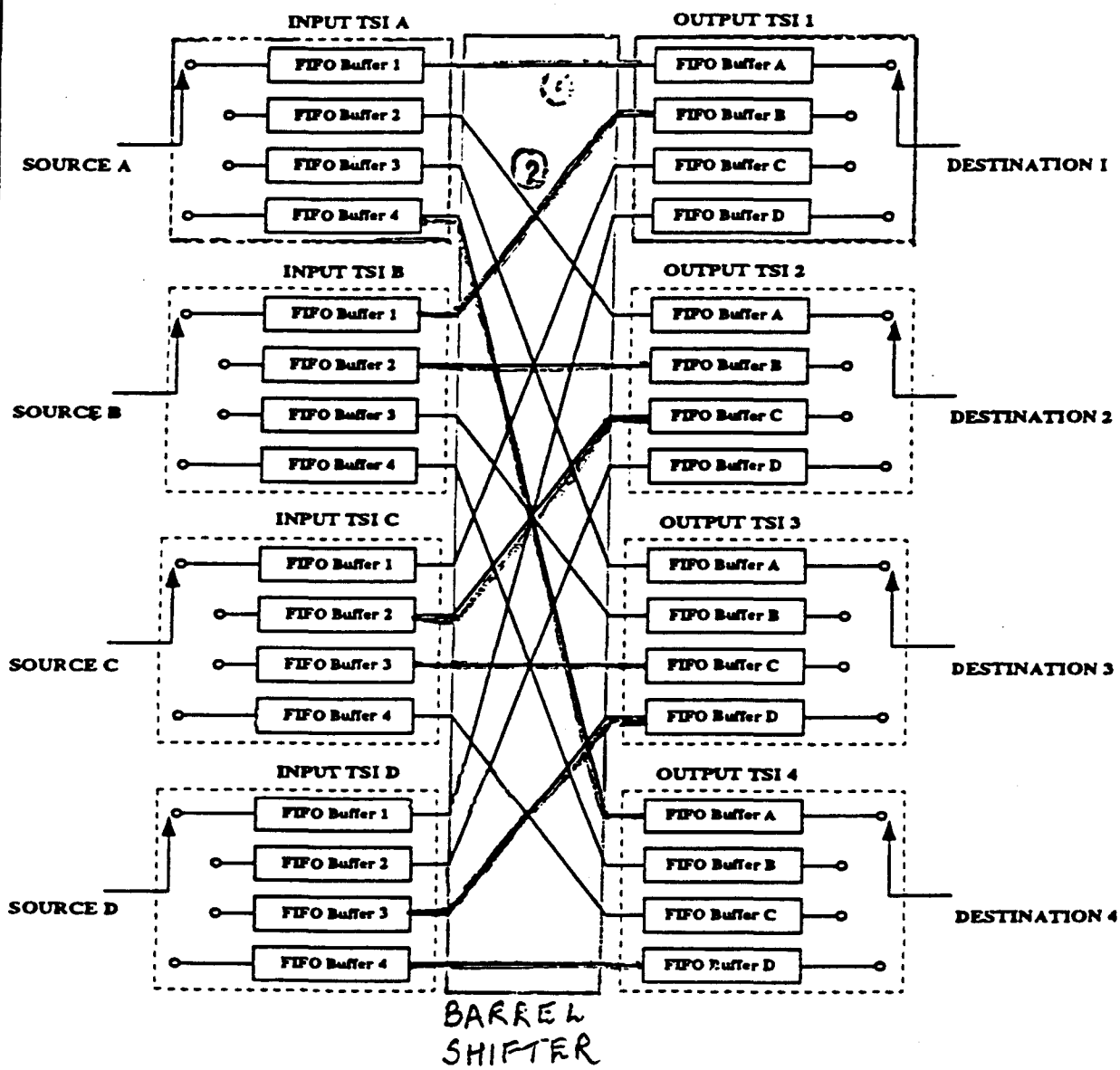

BARREL SHIFTER
Example - 4 Channel Switch
4 Input TSI's
4 Output TSI's
4 X 4 Barrel Shifter

EVENT BUILDER
irregular subevent sizes

- only nulls
- single subevent data
- multiple subevent data
- single subevent data padded with nulls
- multiple subevent data padded with nulls



PARALLEL EVENT BUILDER WITH TIME SLOT INTERCHANGERS



HARDWIRED

SYSTEM-LEVEL SIMULATION

- descriptive modeling
- behavioral simulation
- system-level variables such as throughput and deadtime

SYSTEM PARAMETERS

- the system is tuned such that it saturates at a event rate of 160 Hz

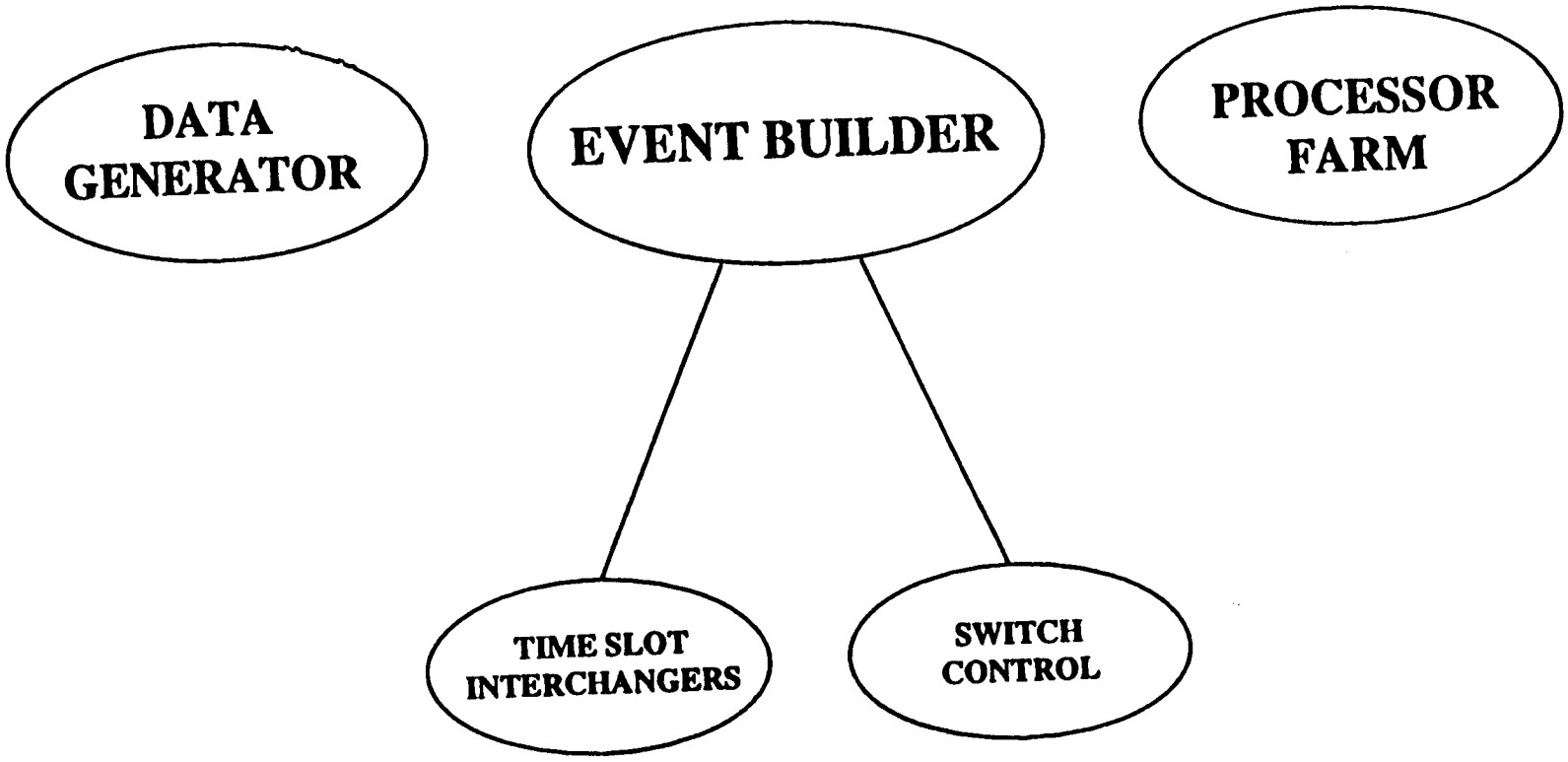
  1 MB <event size> & 8 channels
  125 KB / channel
  20 MB / second links on each channel
  20 MB s^{-1} / 125 KB = 160 Hz

- 130 Hz - 240 Hz - event input rate (fixed rate)

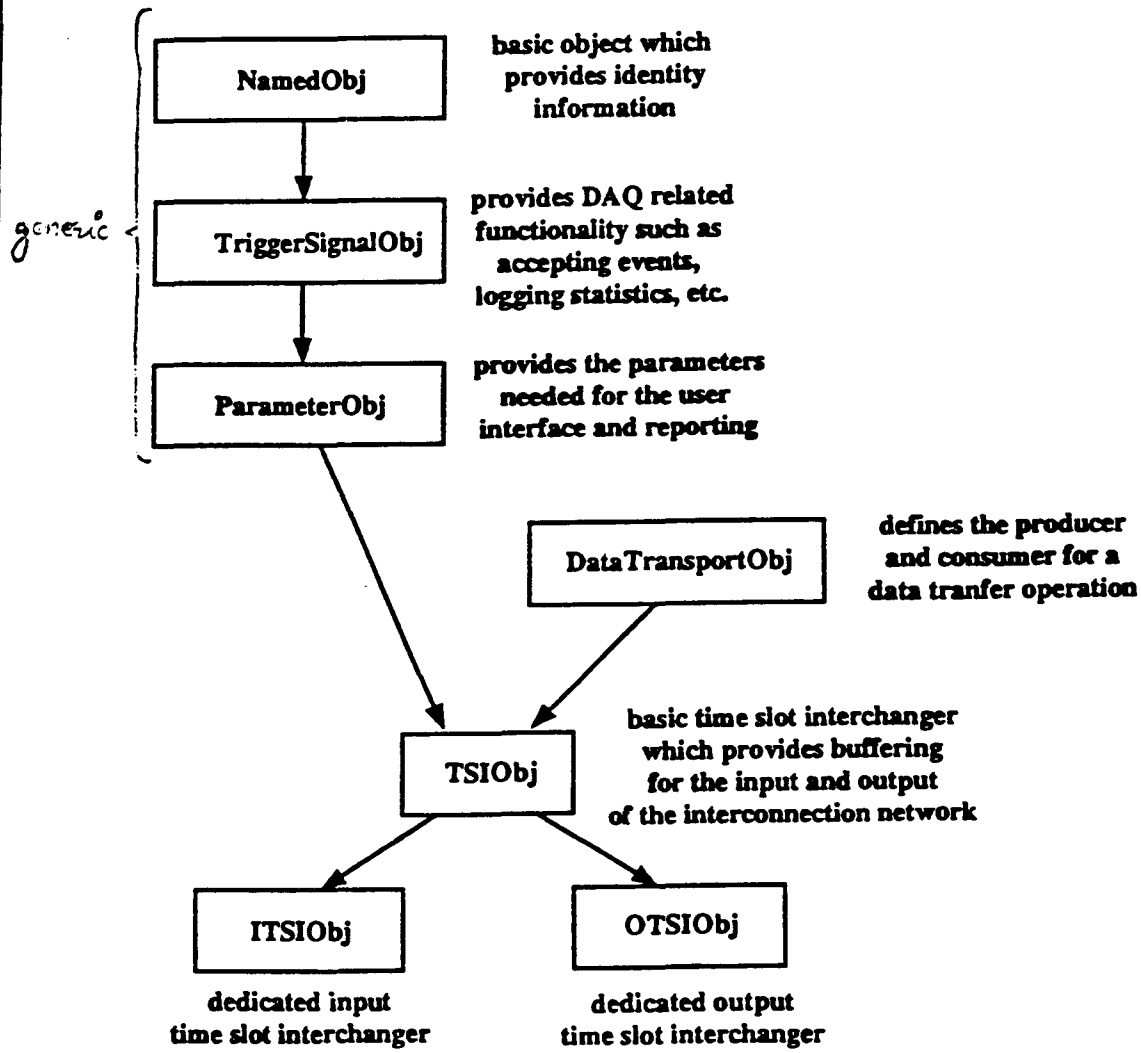
- 1MB mean event size around various distributions

- two architectures
  - base
  - availability feedback loop

SYSTEM MODEL



Inheritance tree for the Input Time Slot Interchanger



MODSIM II

- object-oriented
- modular
- block structured
- strongly-typed
- process-oriented
- discrete-event simulation

Program Information

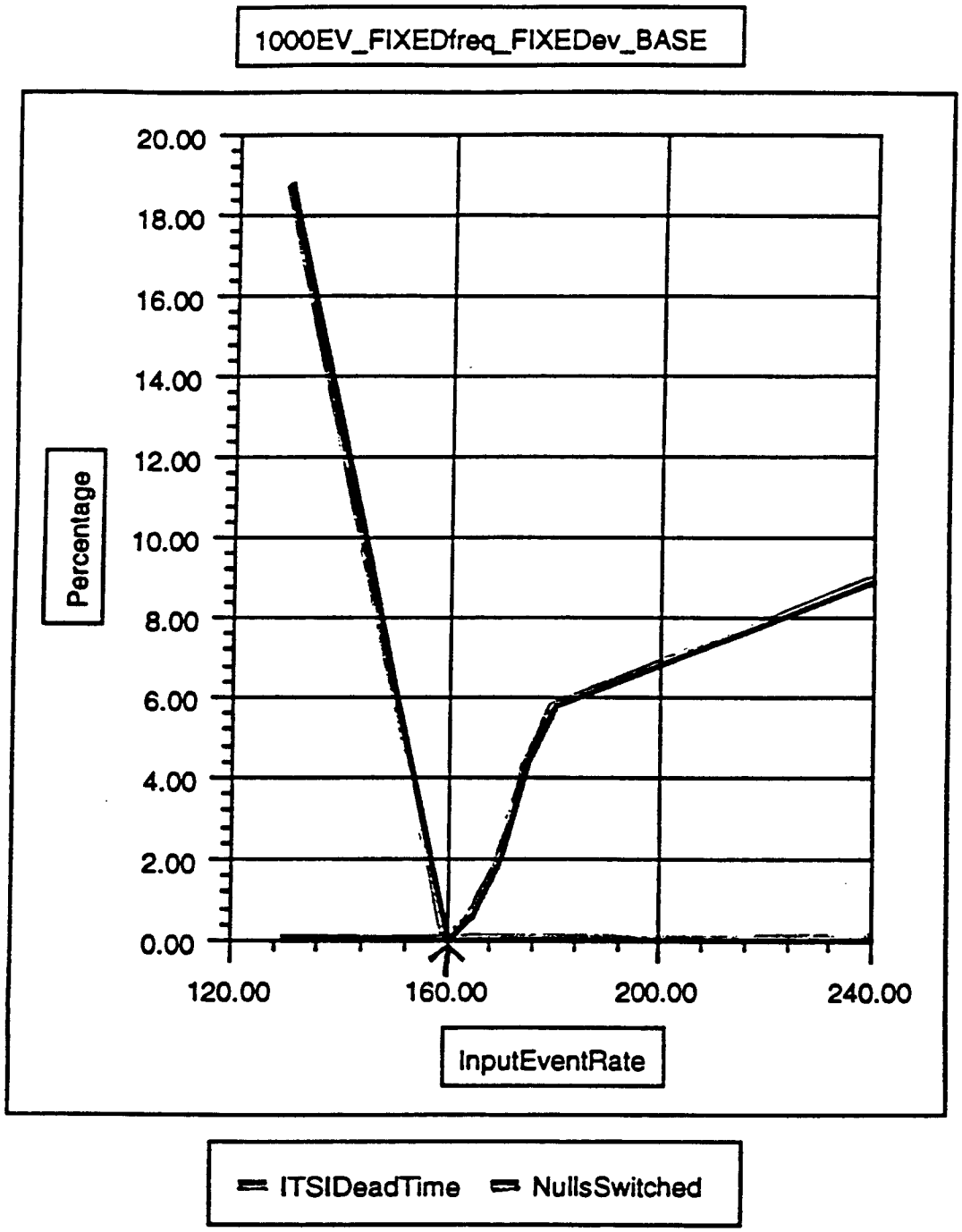
- 4500 lines of MODSIM code
- 20 objects (DAQ and utility)
- Average simulation time 1-2 hours (64 X 64)
- 500 runs - various topologies

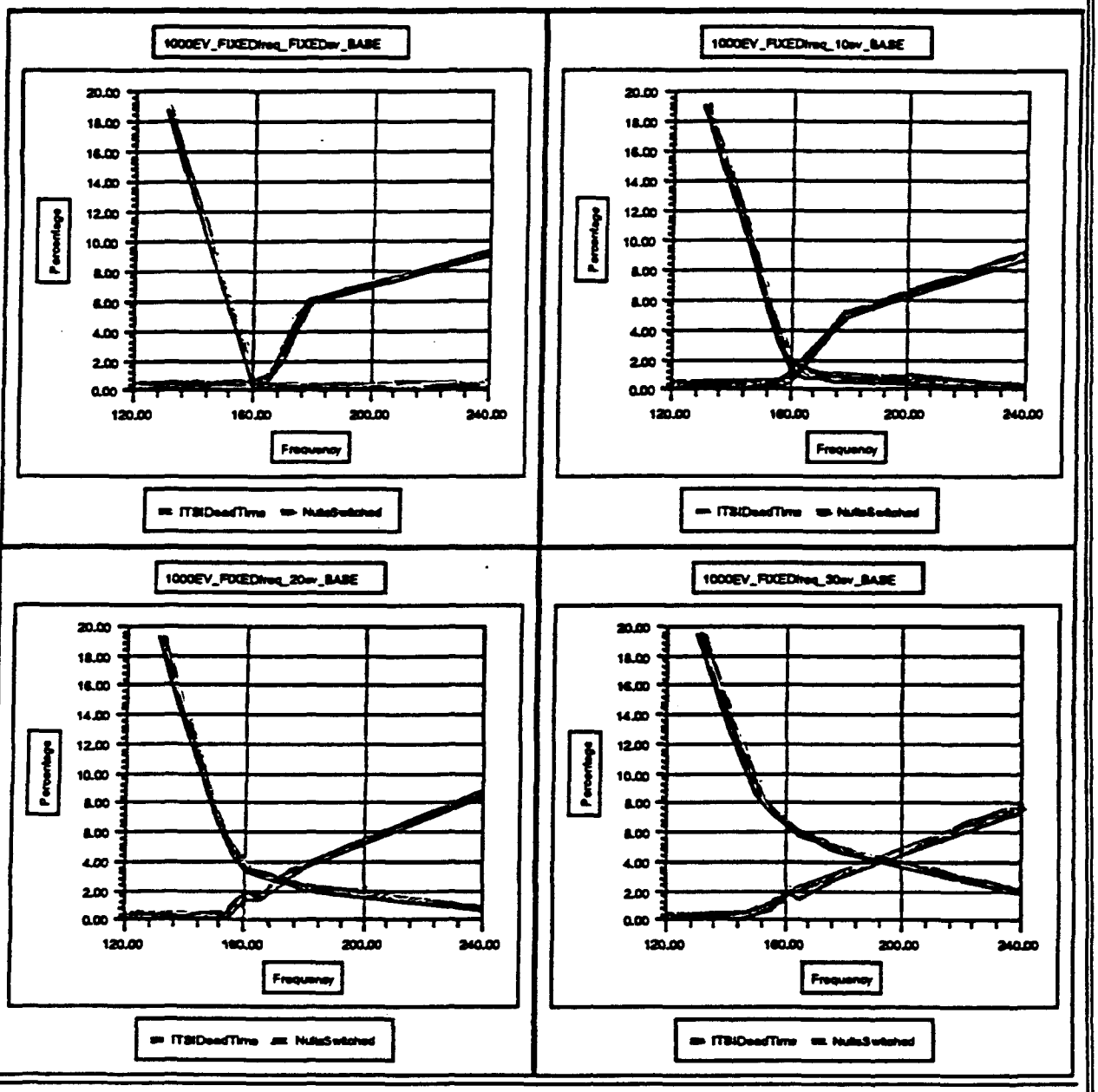
SYSTEM VARIABLES MEASURED

- dead time in the Input Time Slot Interchanger (ITSI) buffers
- percentage of nulls being switched through the interconnection network
- percentage of complete events built by the system

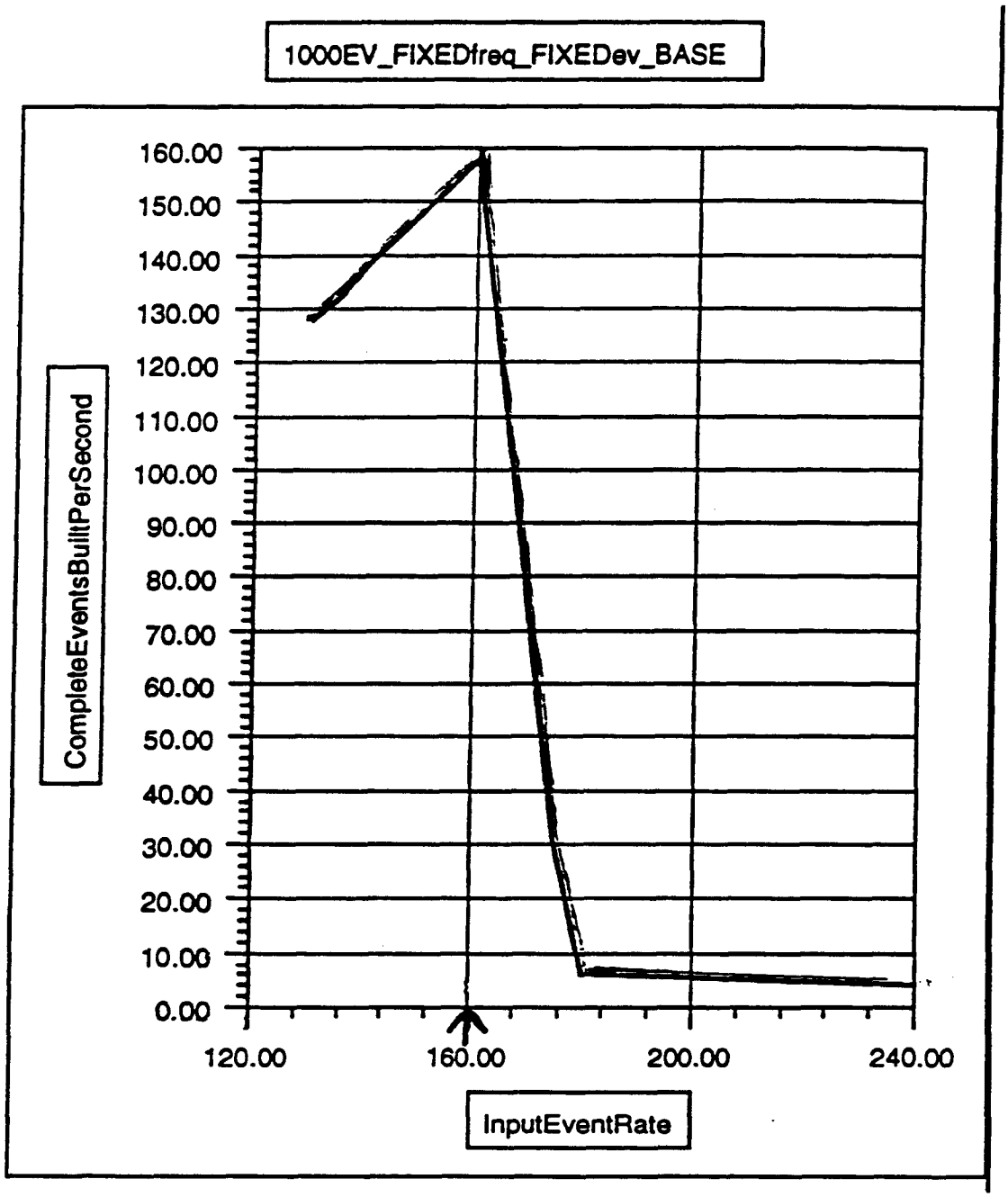
SYSTEM CONSTANTS

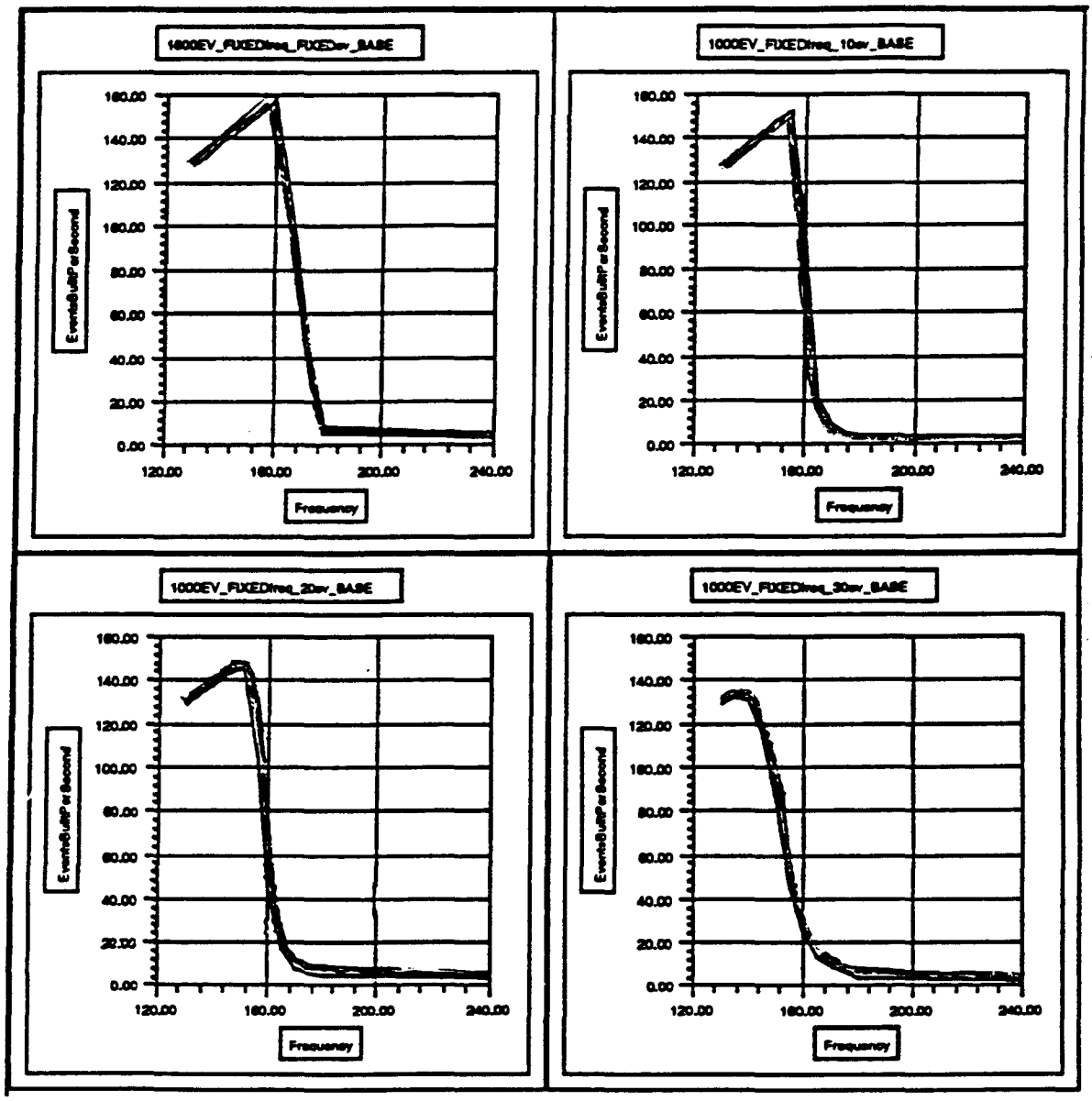
- 8 X 8 switch
- 20 MBytes/second on each link
- 1 MB memory for each ITSI
- 3 MB memory for each OTSI
- each TSI divided into 8 logical buffers
- packet size equal to 10% of the <subevent>size


1000EV_FIXEDfreq_FIXEDev_BASE
Percentage
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
120.00
160.00
200.00
240.00
InputEventRate
ITSIDeadTime
NullsSwitched

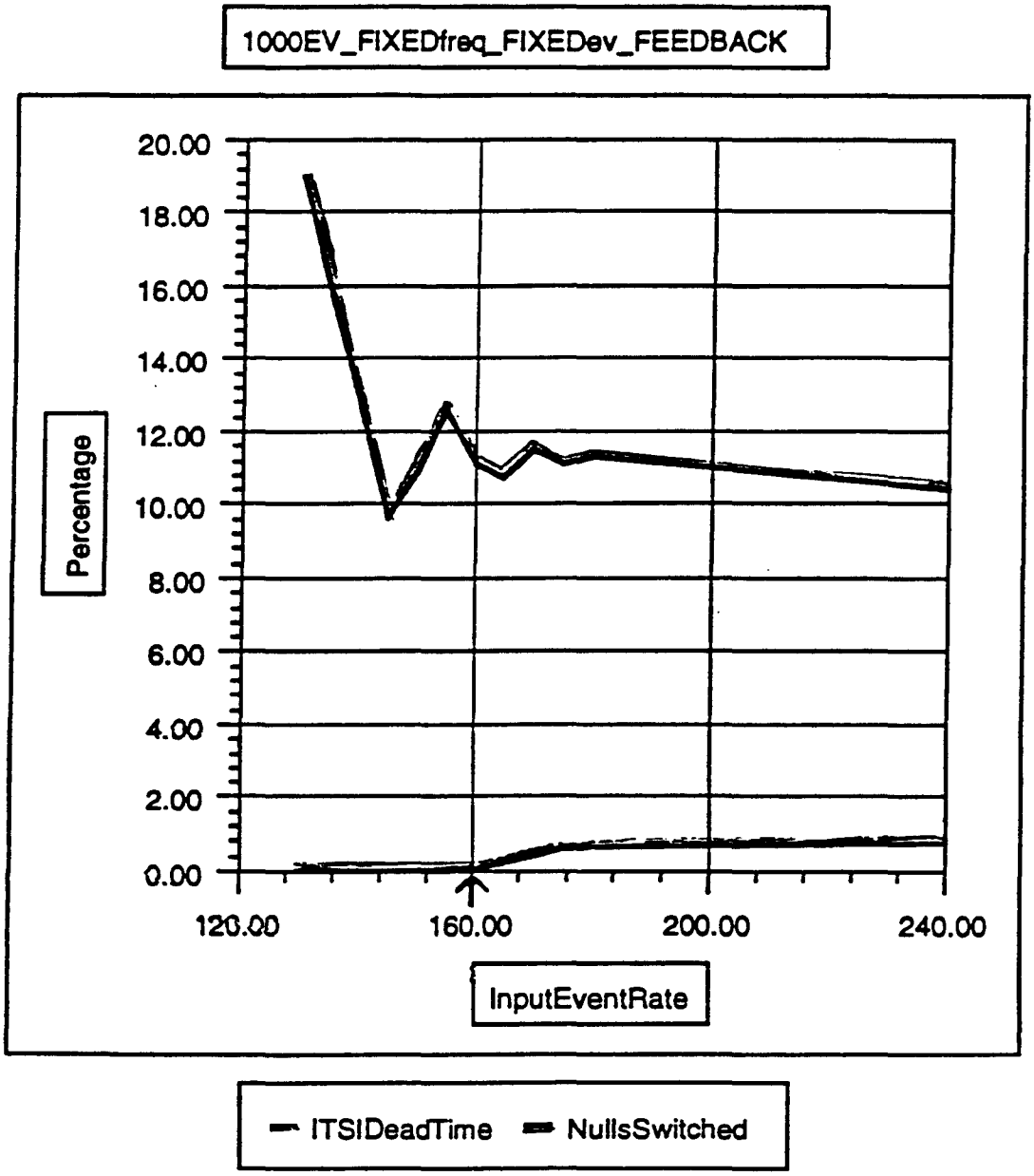


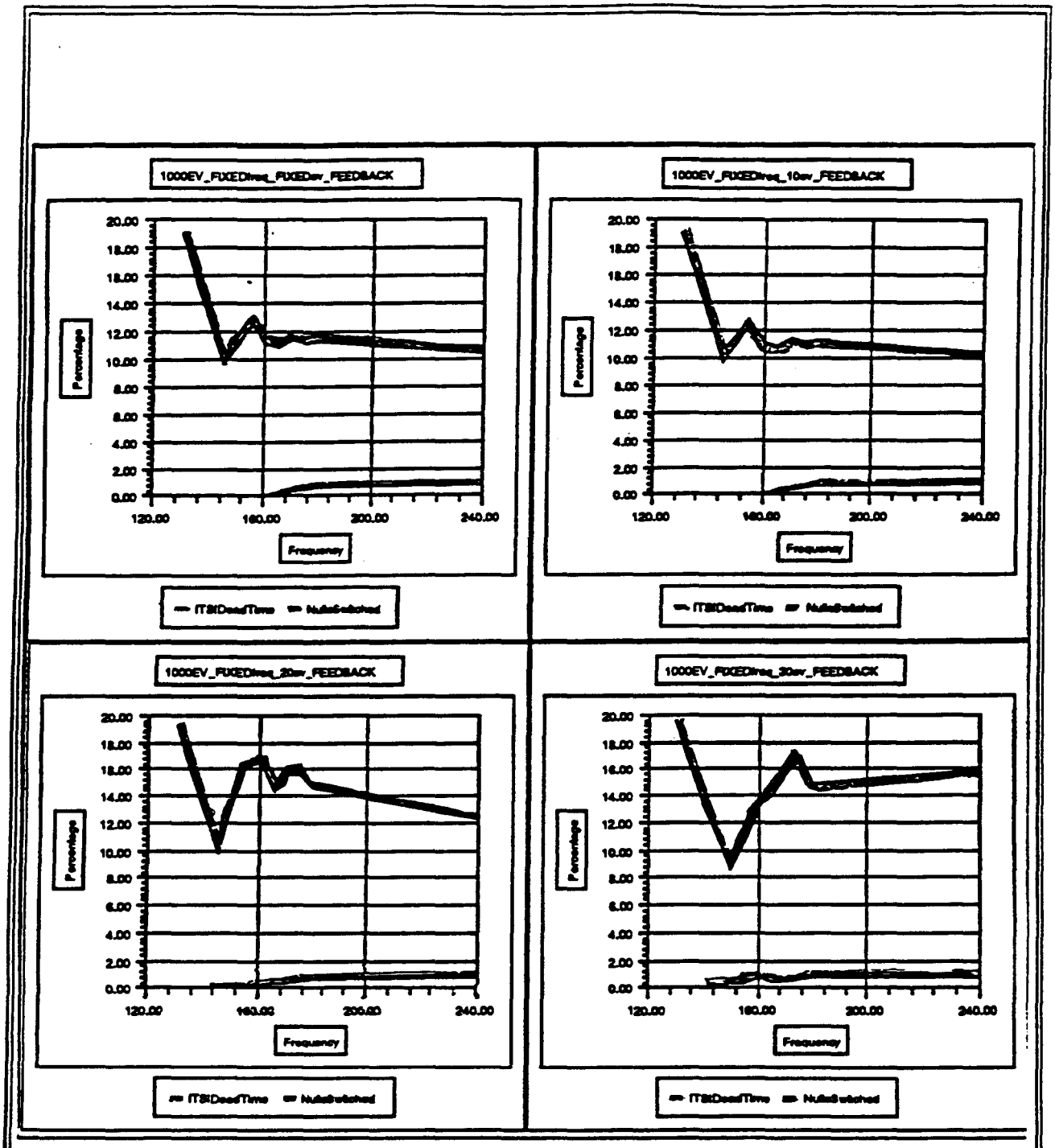
—— Percentage dead time in the ITSI buffers
—·—·— Percentage of nulls switched
v/s
Event Input Rate


1000EV_FIXEDfreq_FIXEDev_BASE
CompleteEventsBuiltPerSecond
160.00
150.00
140.00
130.00
120.00
110.00
100.00
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
120.00
160.00
200.00
240.00
InputEventRate

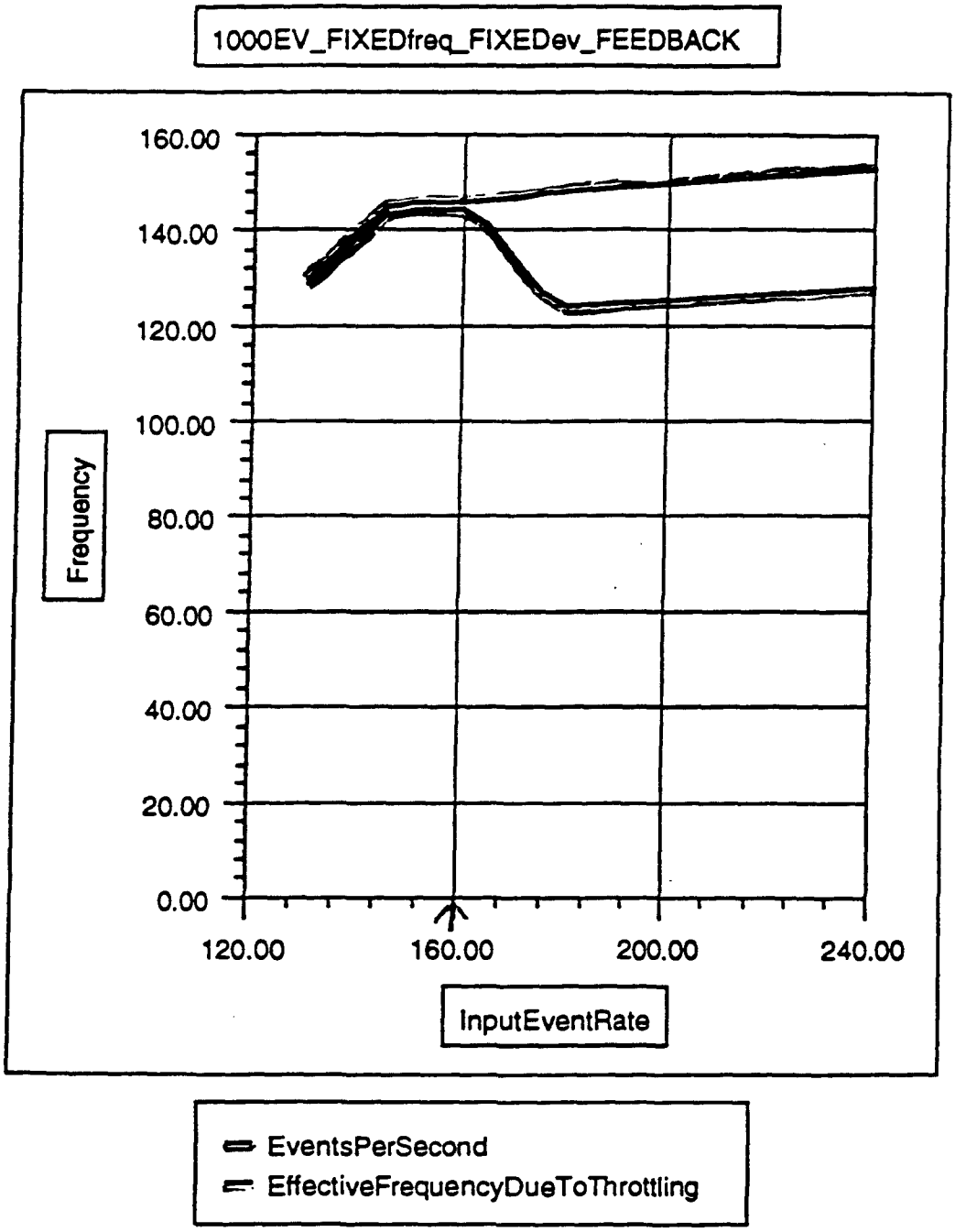


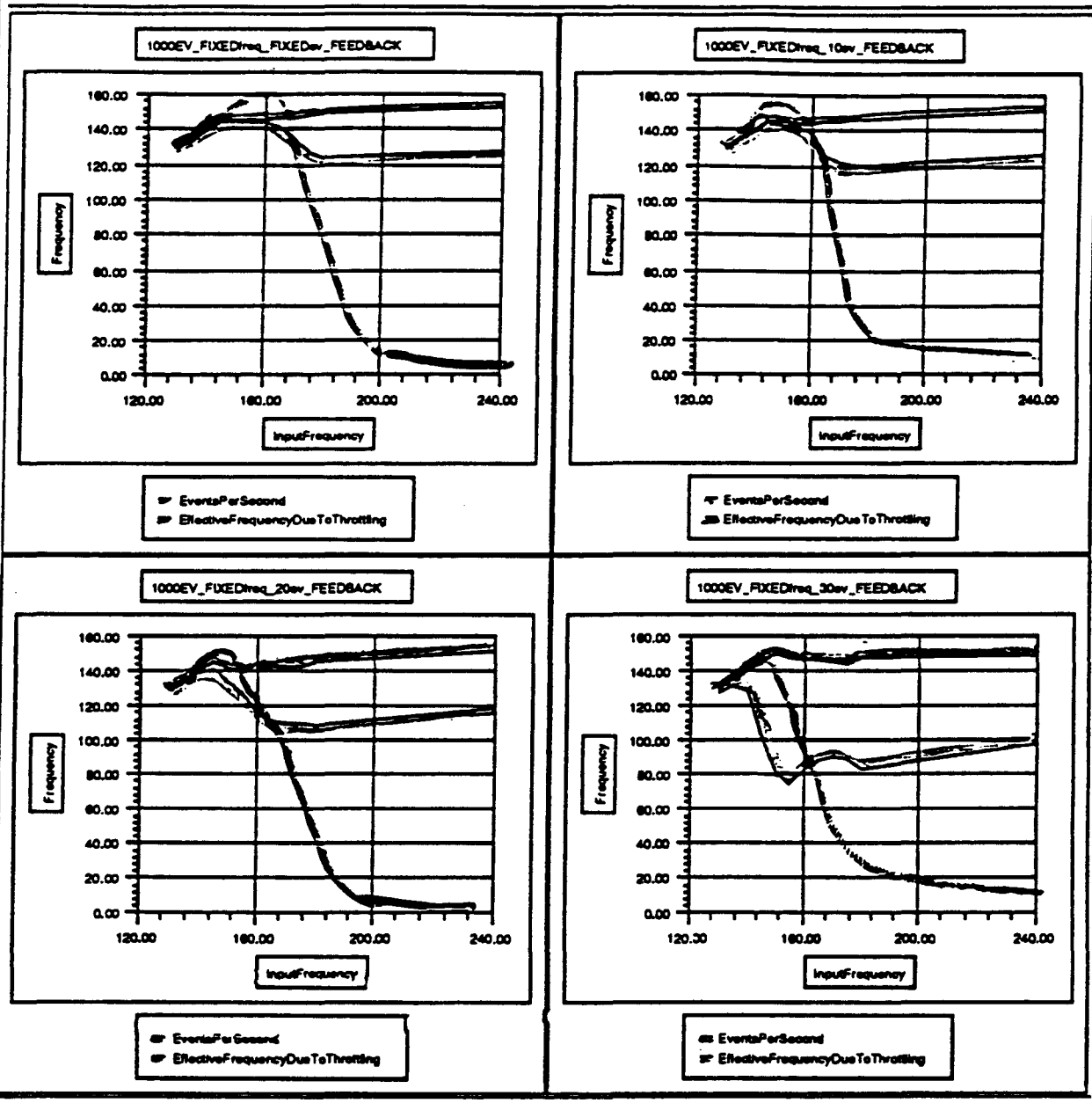
Number of complete events built per second
v/s
Event Input Rate


1000EV_FIXEDfreq_FIXEDev_FEEDBACK
Percentage
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
120.00
160.00
200.00
240.00
InputEventRate
ITSIDeadTime
NullsSwitched



Percentage of dead time in the ITSI buffers
Percentage of nulls switched through the interconnection network
v/s
Event Input Rate


1000EV_FIXEDfreq_FIXEDev_FEEDBACK
160.00
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
Frequency
120.00
160.00
200.00
240.00
InputEventRate
EventsPerSecond
EffectiveFrequencyDueToThrottling



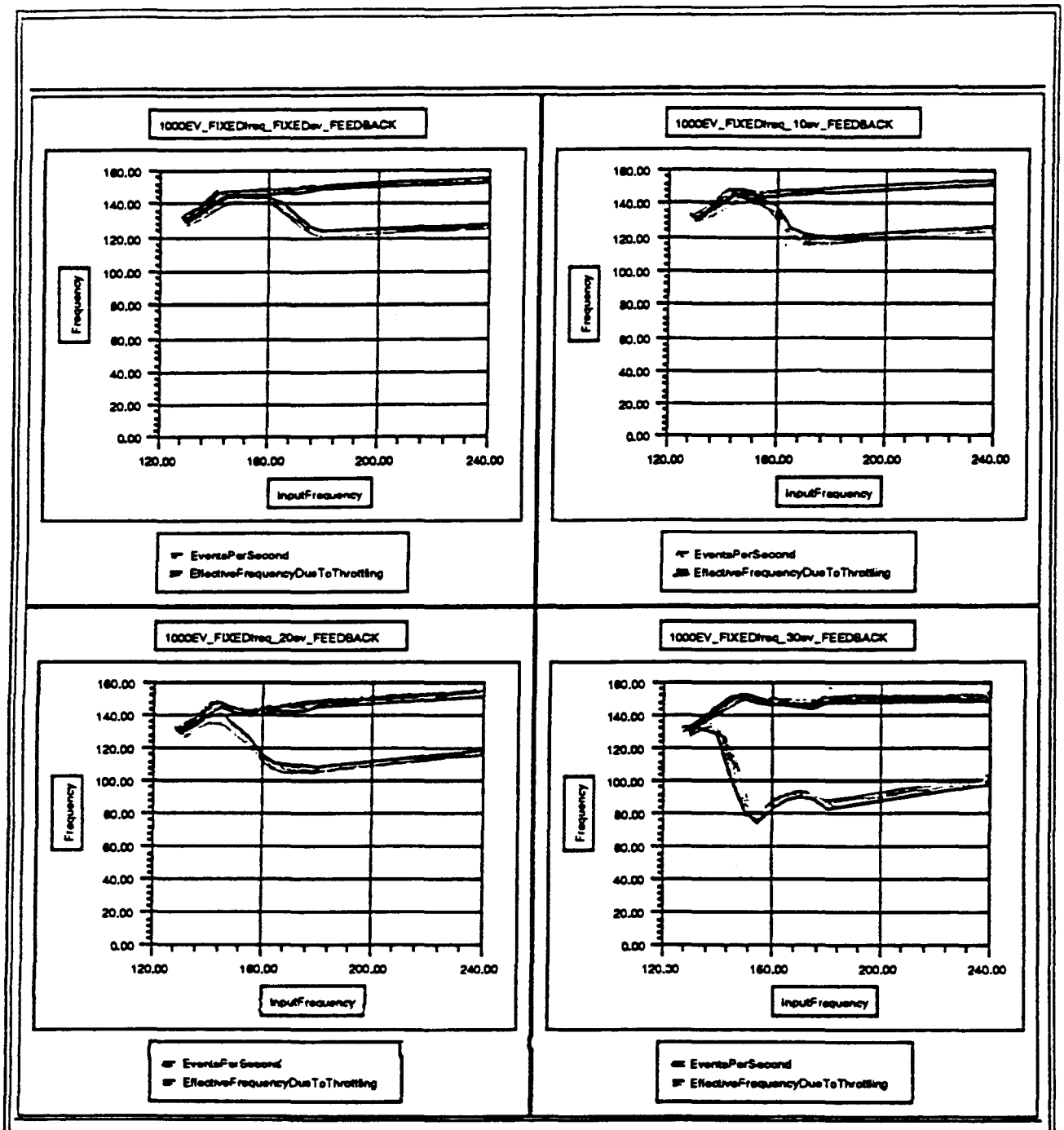
Effective event input rate

Number of complete events built per second

v/s

Event Input Rate

— Number of complete events (no feedback)



Effective event input rate

Number of complete events built per second

v/s

Event Input Rate

CONCLUSIONS
(behavioral)

- ITSI buffer size insufficient (one event)
- packet size feasible
- feedback improves the efficiency
- such an architecture requires a high degree of synchronization and regularity

GATE-LEVEL SIMULATION

- prescriptive modeling
- hardware simulation
- design functionality

ITSI REQUIREMENTS

- memory organized as FIFO buffers
- header and data separation
- header decoding
- sequencing packets for transmission

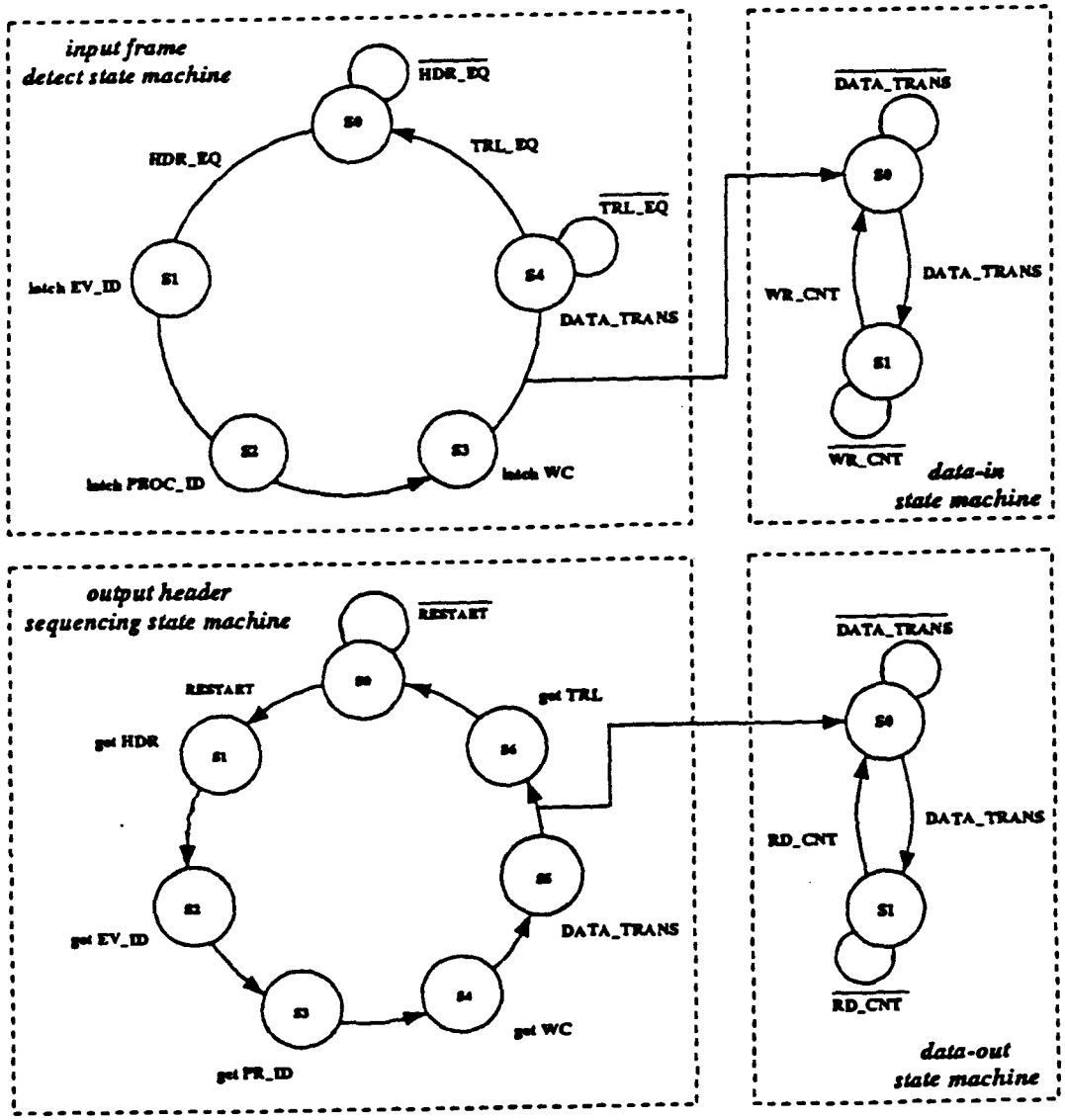
8-bits	1 to 8 8-bit words	8-bits	8-bits	8-bits	8-bits
trl ptrn	data	wc	pr id	ev id	hdr ptrn

data format

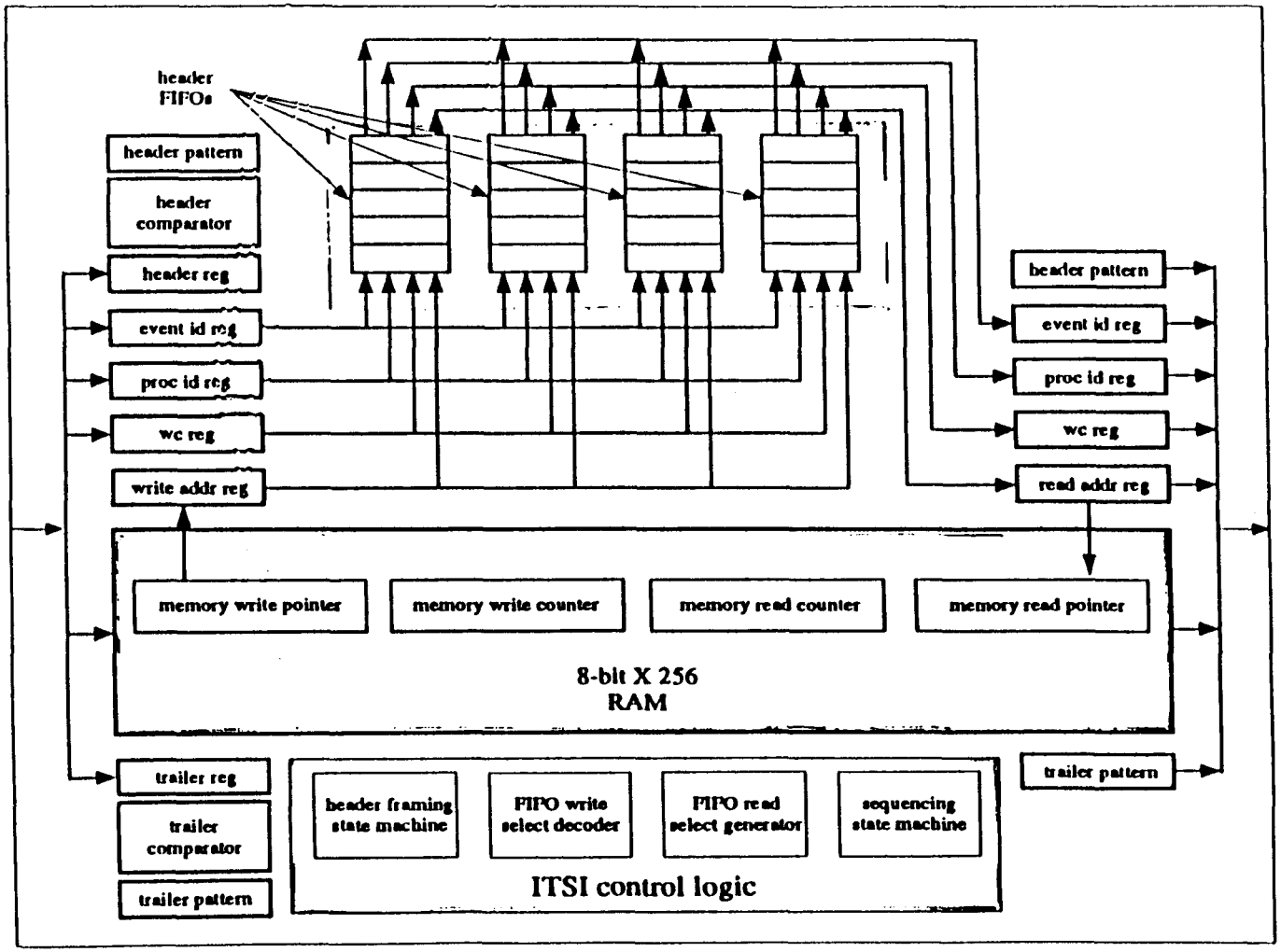
VERILOG

- different levels of simulation
  - system/architectural
  - behavioral/register transfer
  - gate level
- allows mixed level simulation
- stochastic analysis
- hierarchical specification
- libraries for logic synthesis
- interactive debugging

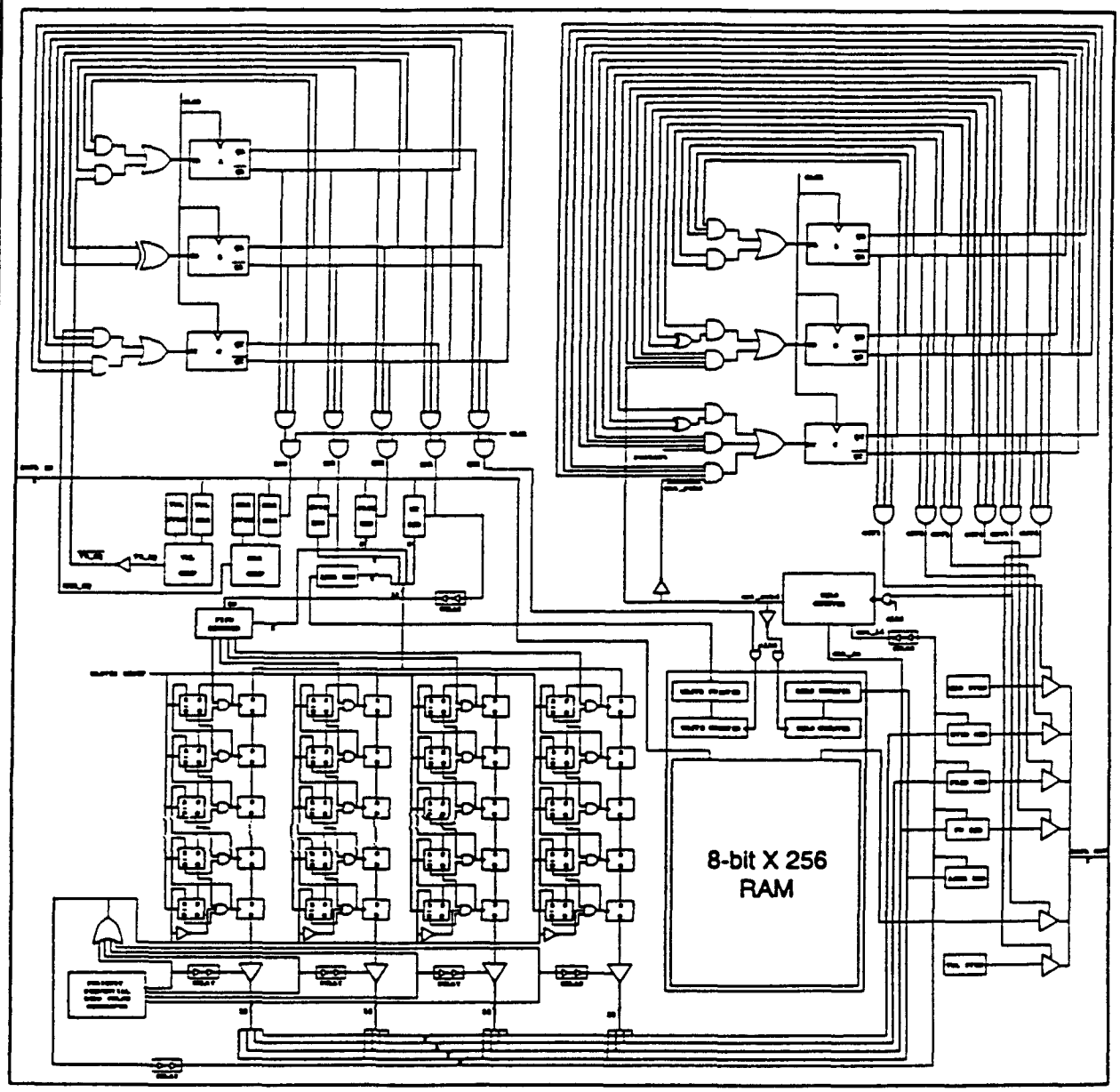
STATE DIAGRAMS FOR THE ITSI OPERATION



Internal Structure of the ITSI



Logic-level Schematic diagram of the ITSI

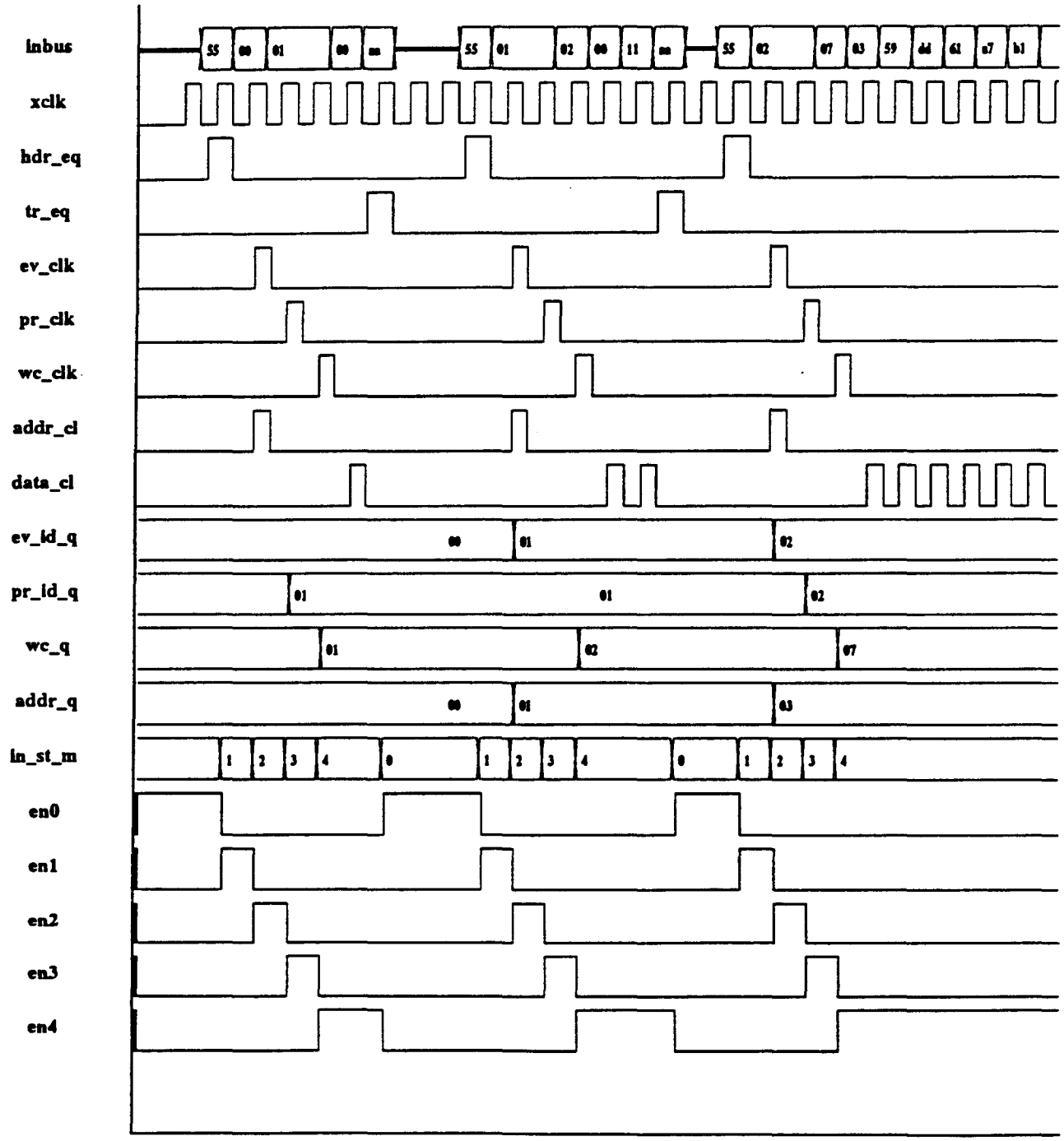


Gateway

Header: INPUT STATE MACHINE AND FRAMING LOGIC

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:25:33 **Time Scale From:** 0 **To:** 1425 **Page:** 1 of 1

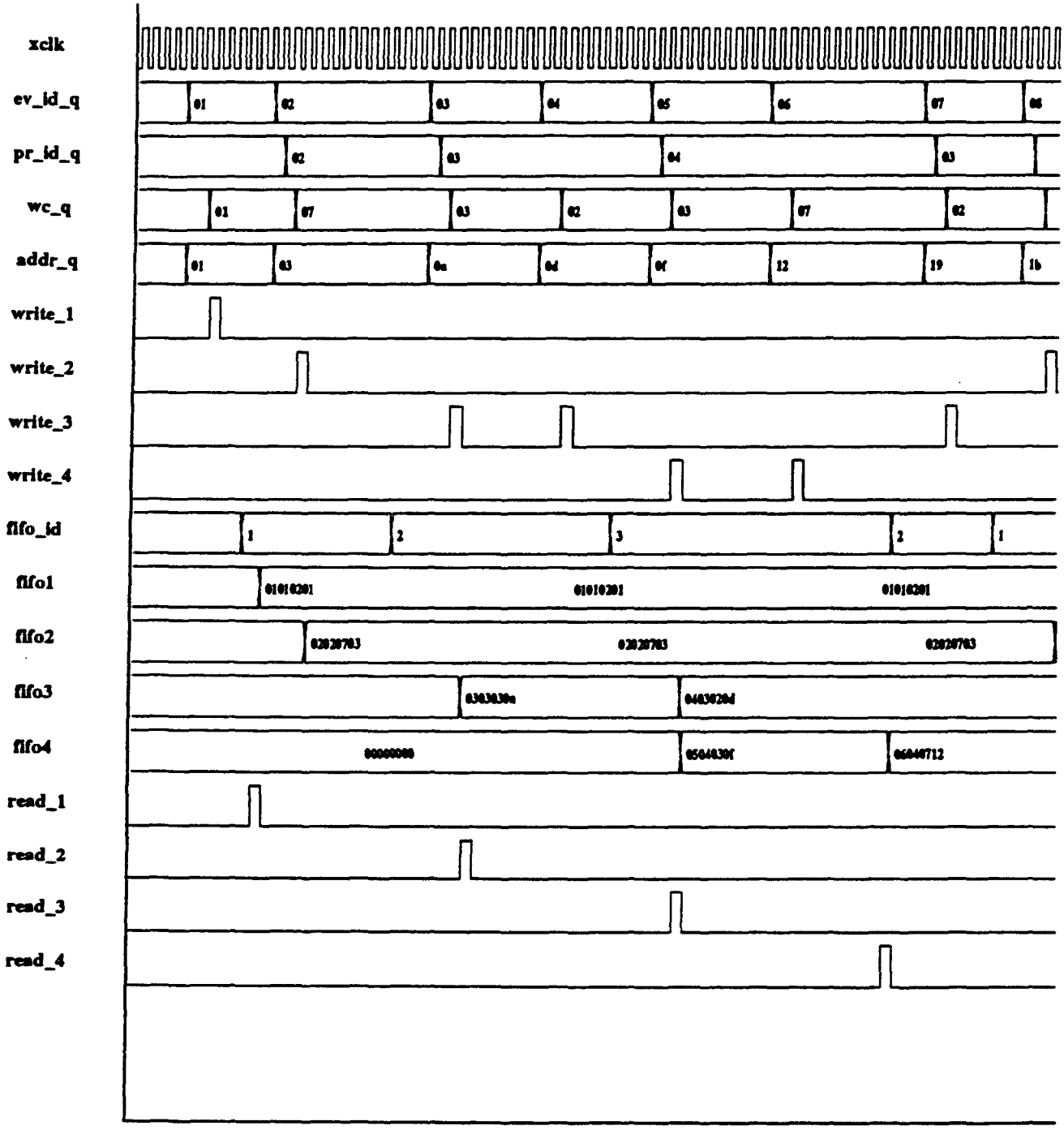


Gateway

Header: FIFO WRITE AND READ OPERATIONS

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:30:21 **Time Scale From:** 355 **To:** 4550 **Page:** 1 of 1

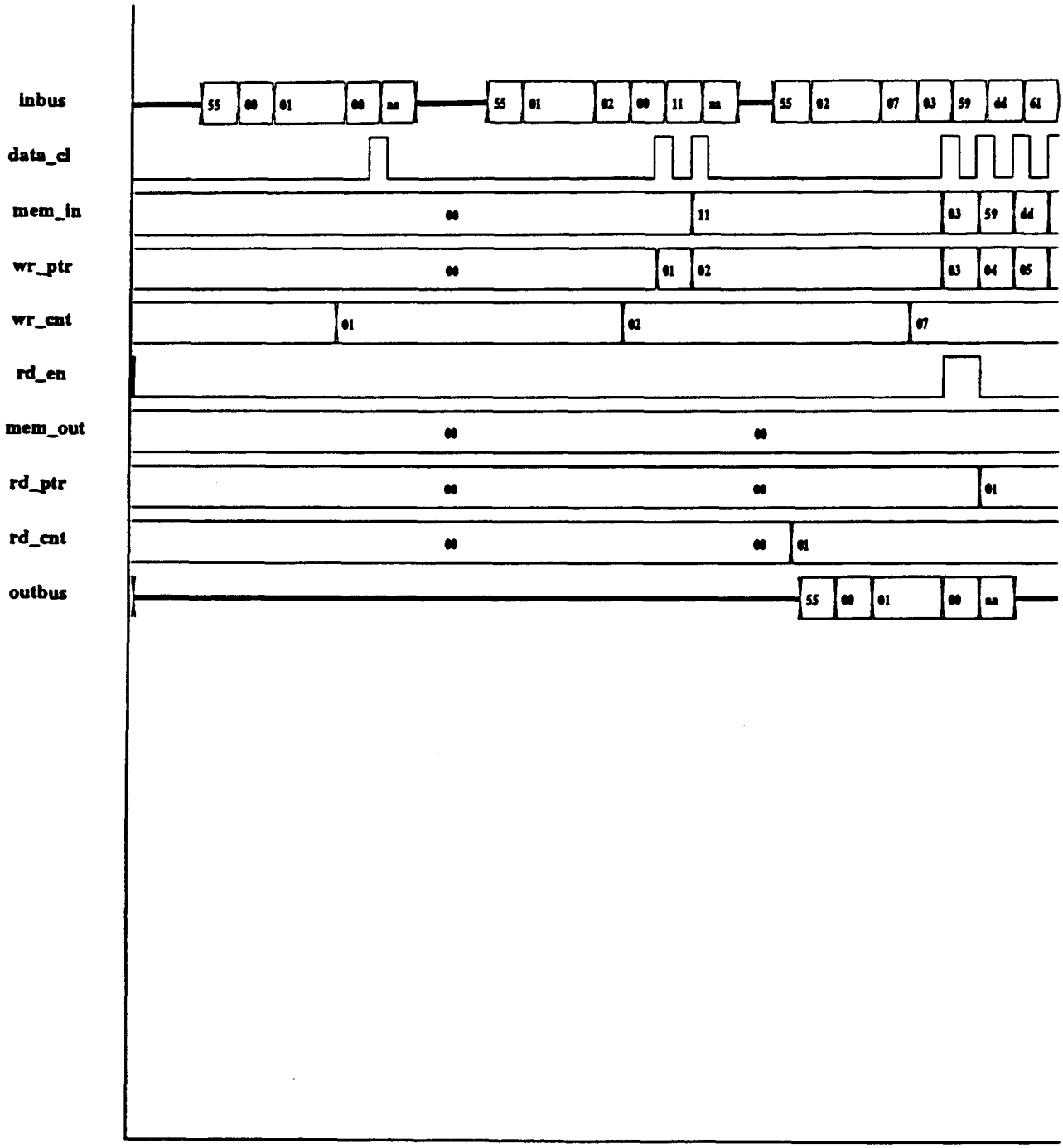


Gateway

Header: MEMORY WRITE AND READ OPERATIONS - 1

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:38:30 | **Time Scale From:** 0 **To:** 1295 | **Page:** 1 of 1

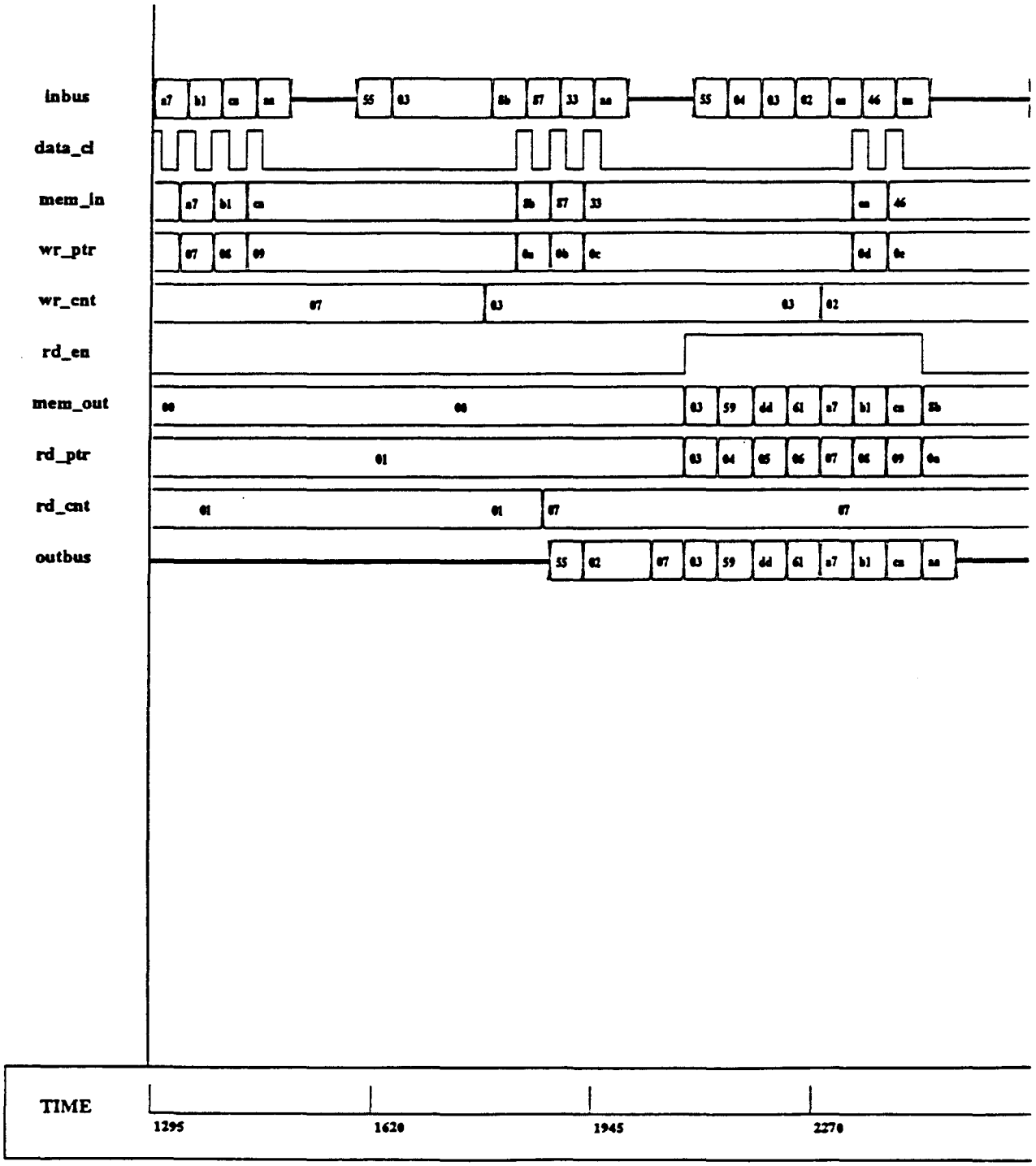


Gateway

Header: MEMORY WRITE AND READ OPERATIONS - 2

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:39:10 **Time Scale From:** 1295 **To:** 2595 **Page:** 1 of 1

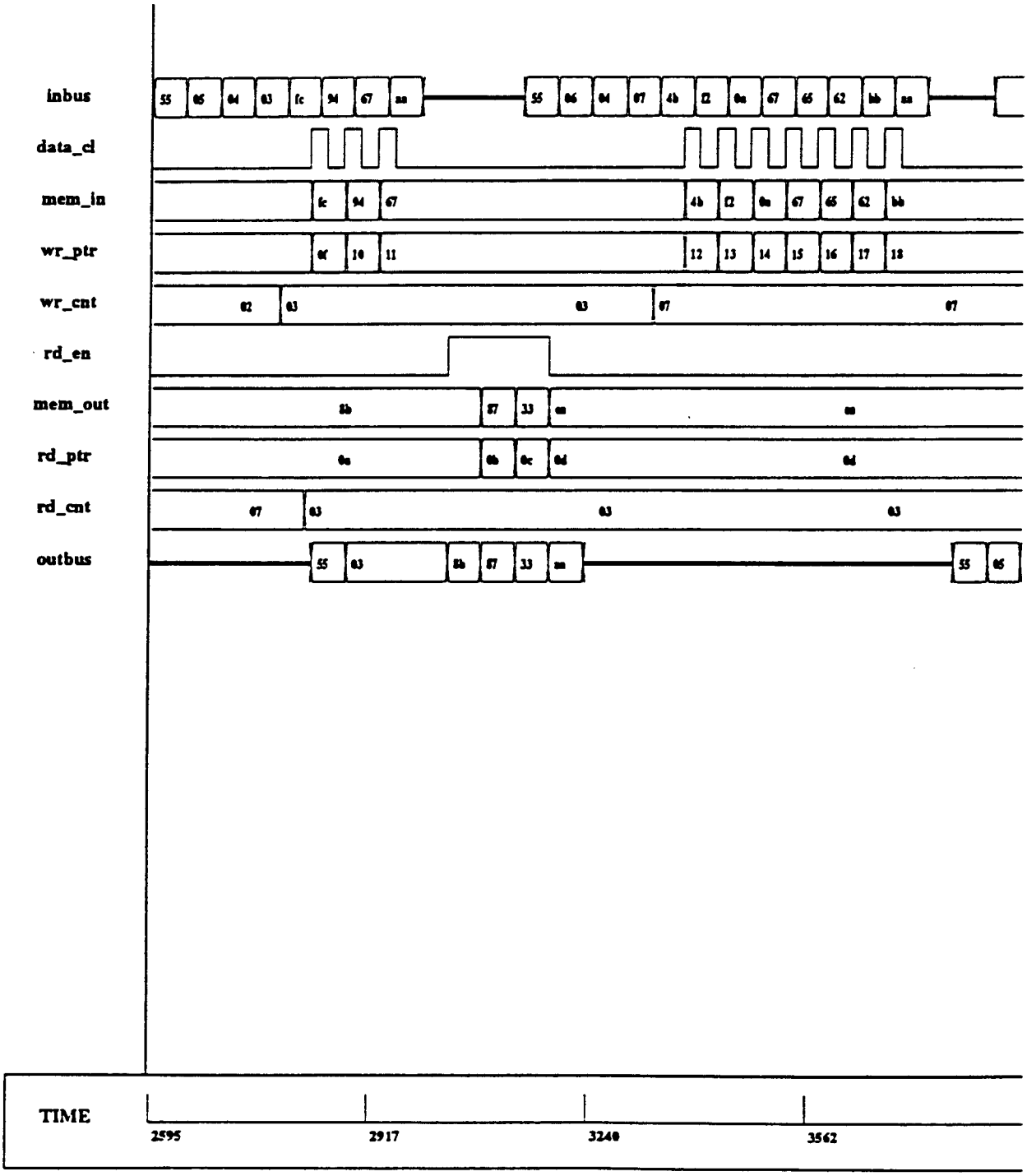


Gateway

Header: MEMORY WRITE AND READ OPERATIONS - 3

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:39:58 | Time Scale From: 2595 To: 3885 | Page: 1 of 1

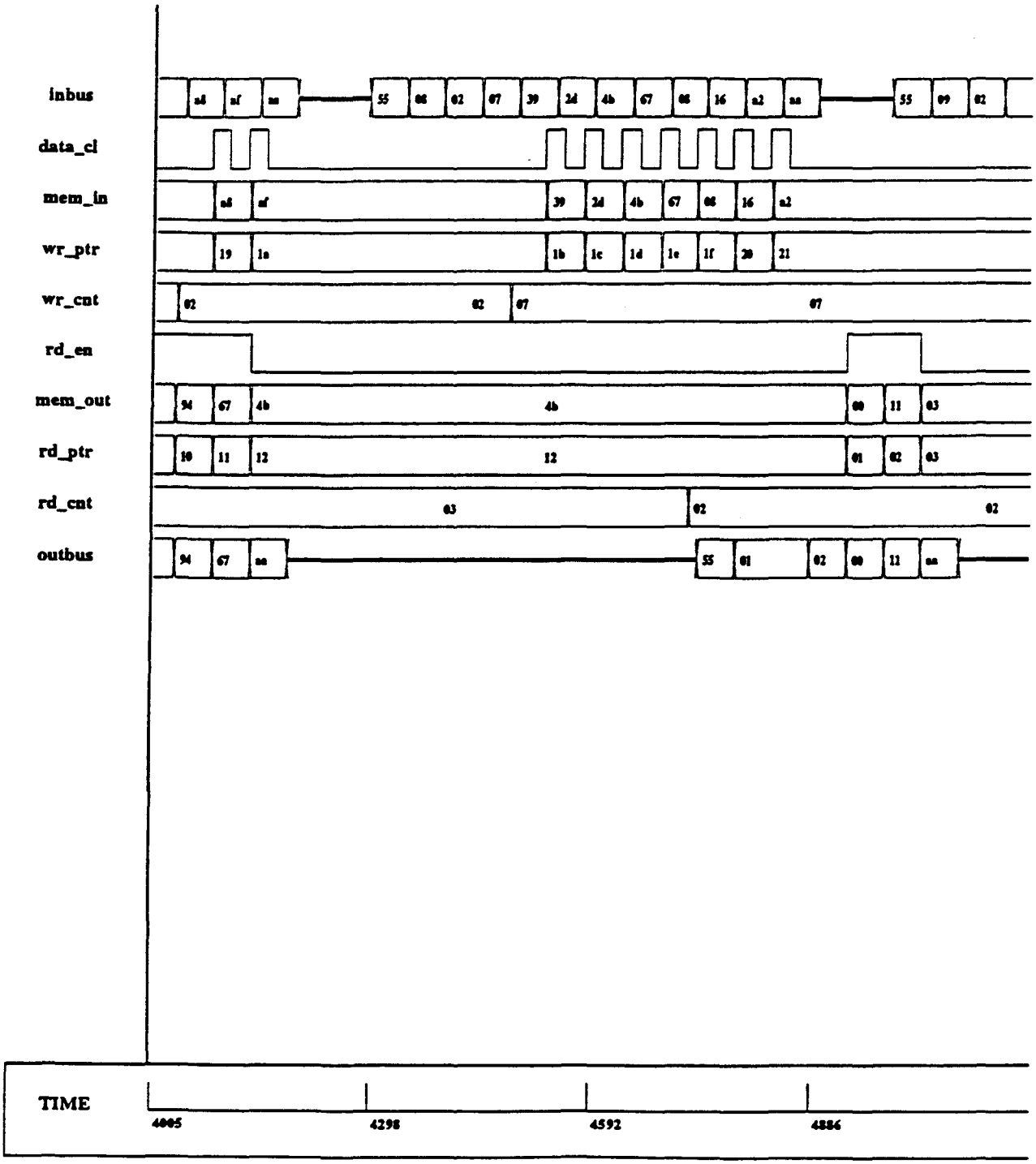


Gateway

Header: MEMORY WRITE AND READ OPERATIONS - 4

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:41:27 **Time Scale From:** 4005 **To:** 5180 **Page:** 1 of 1

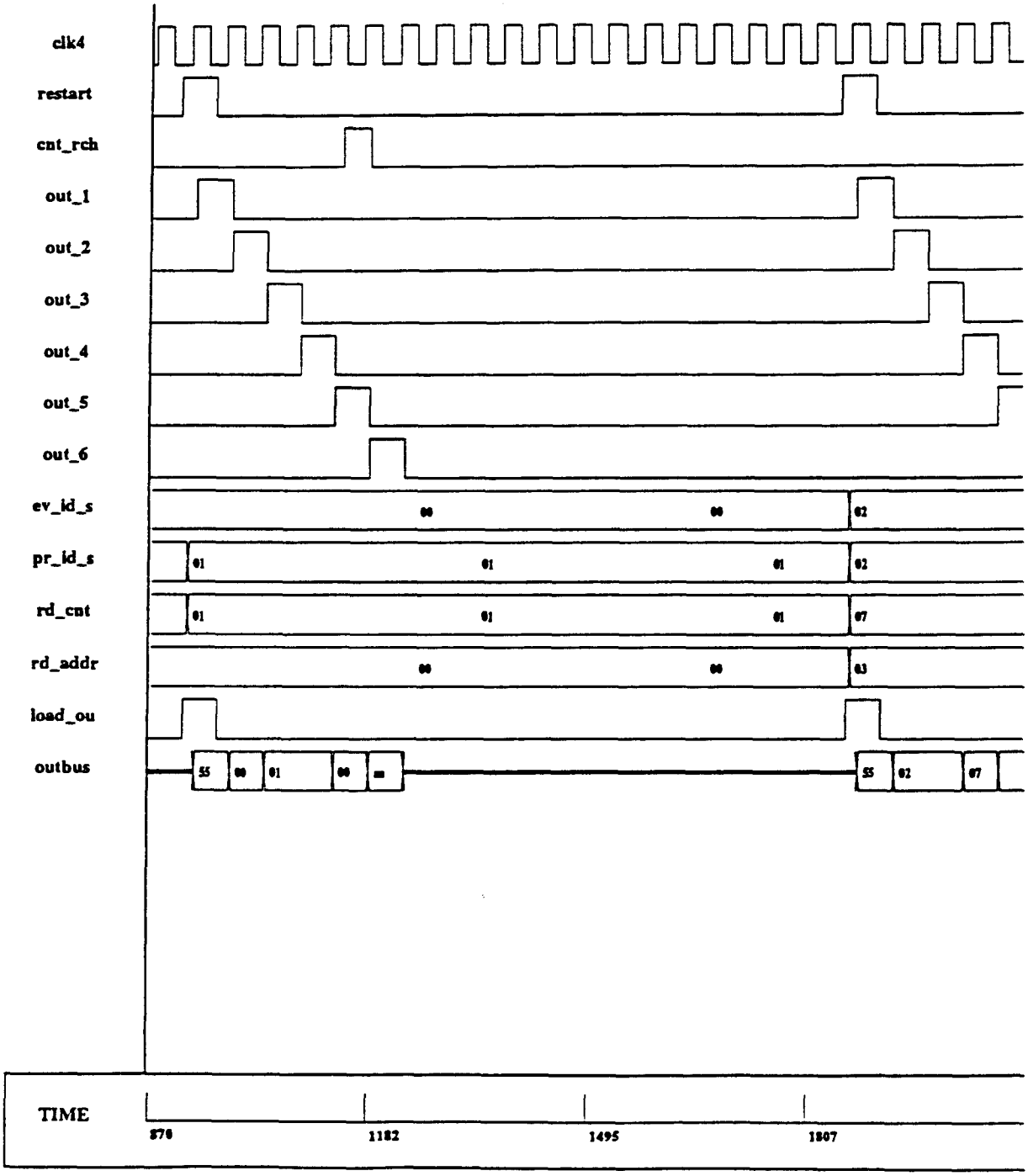


Gateway

Header: OUTPUT STATE MACINE AND SEQUENCING LOGIC

User: Vishal S. Kapoor

Date: Jan 9, 1992 11:42:46 | Time Scale From: 870 To: 2120 | Page: 1 of 1



CONCLUSIONS
(gate - level)

- idea of load balancing the buffers in the ITSI, learnt from the MODSIM simulation, was implemented

- functionality of the design was verified

SUMMARY

- studied the event builder problem at the SSC

- behavioral study done in MODSIM II
  - deadtime
  - switch efficiency
  - gross system behavior

- gate-level study done in VERILOG
  - focussed on packet structure
  - memory (ITSI) implementation

- results are a useful basis for a final ASIC design

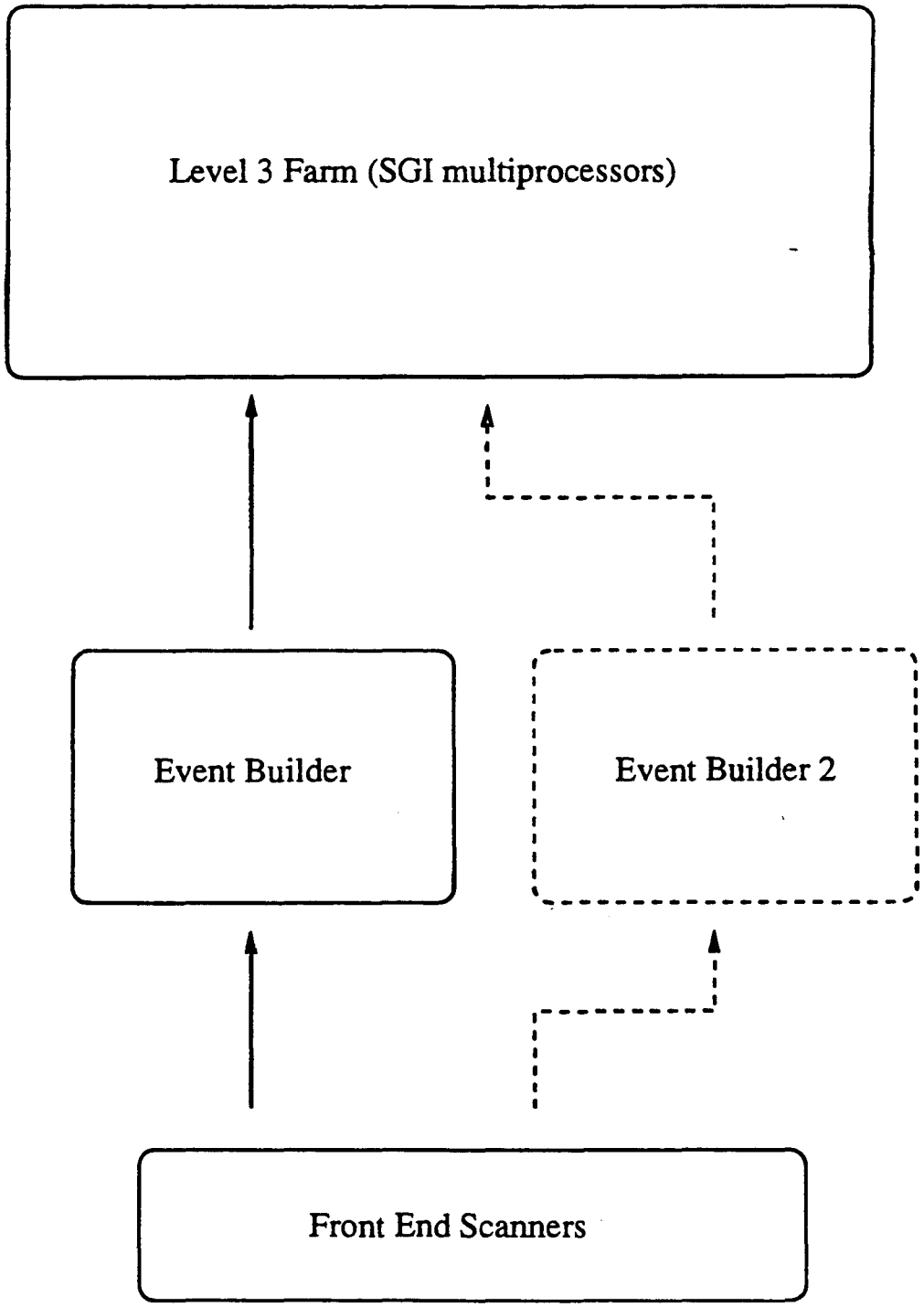
A Verilog Simulation for the ?1991? CDF DAQ

1. Design goal for the hardware upgrade.
2. Why a simulation?
3. Design of the simulation.
4. Results.
5. A look forward.

R. Grindley & P.K. Sinervo

Proposed Modification to CDF DAQ System

(Solid lines represent previously existing hardware)

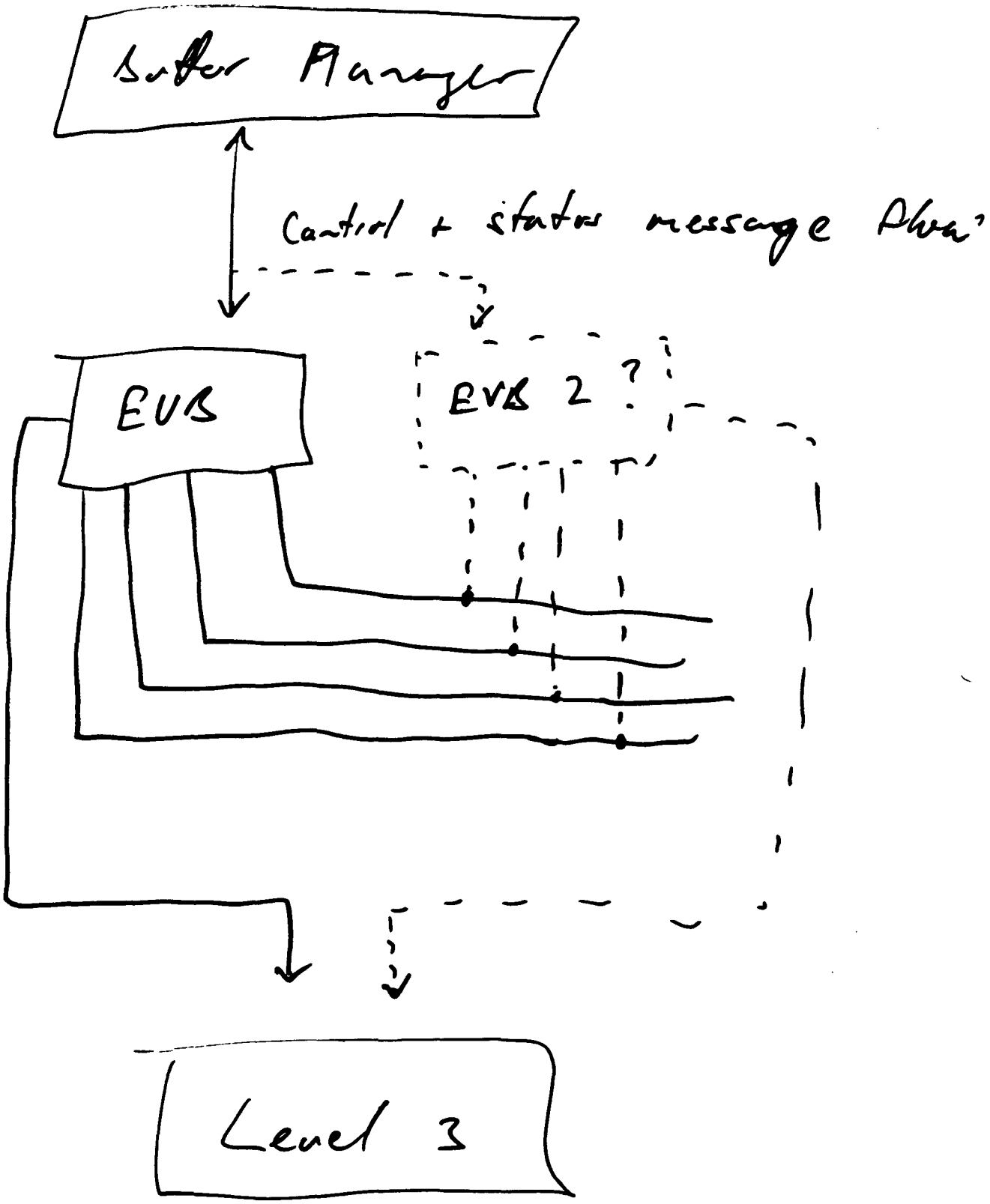


- Most DAQ systems are non-linear.

- In the very linear region (far away from any saturations) we know how it scales.

- In the very non-linear region, we also know that it does not scale at all. (Hitting a brick wall, as all or at least one component in a pipeline saturates)

- As the number of components gets high and complexity (interdependence of components) increases, our ability to "intuit" the system decreases.


Buffer Manager
Control + status message flow
EVS
EVS 2 ?
Level 3

Front End scan time ~ 2 ms.

If L2 trigger rate is 40 Hz, scanner livetime

$$\epsilon = 1 - \frac{0.002}{0.025} = 92\%$$

Design goal 40-50 Hz allows $\lesssim 10\%$ deadtime.

Existing hardware has 4 FastBus cable segments, each connected to one reformatter board in the Event Builder (EVB).

$\Rightarrow$ <u>Basic Question</u>

Increase # of EVB's or # of reformatter boards / EVB?

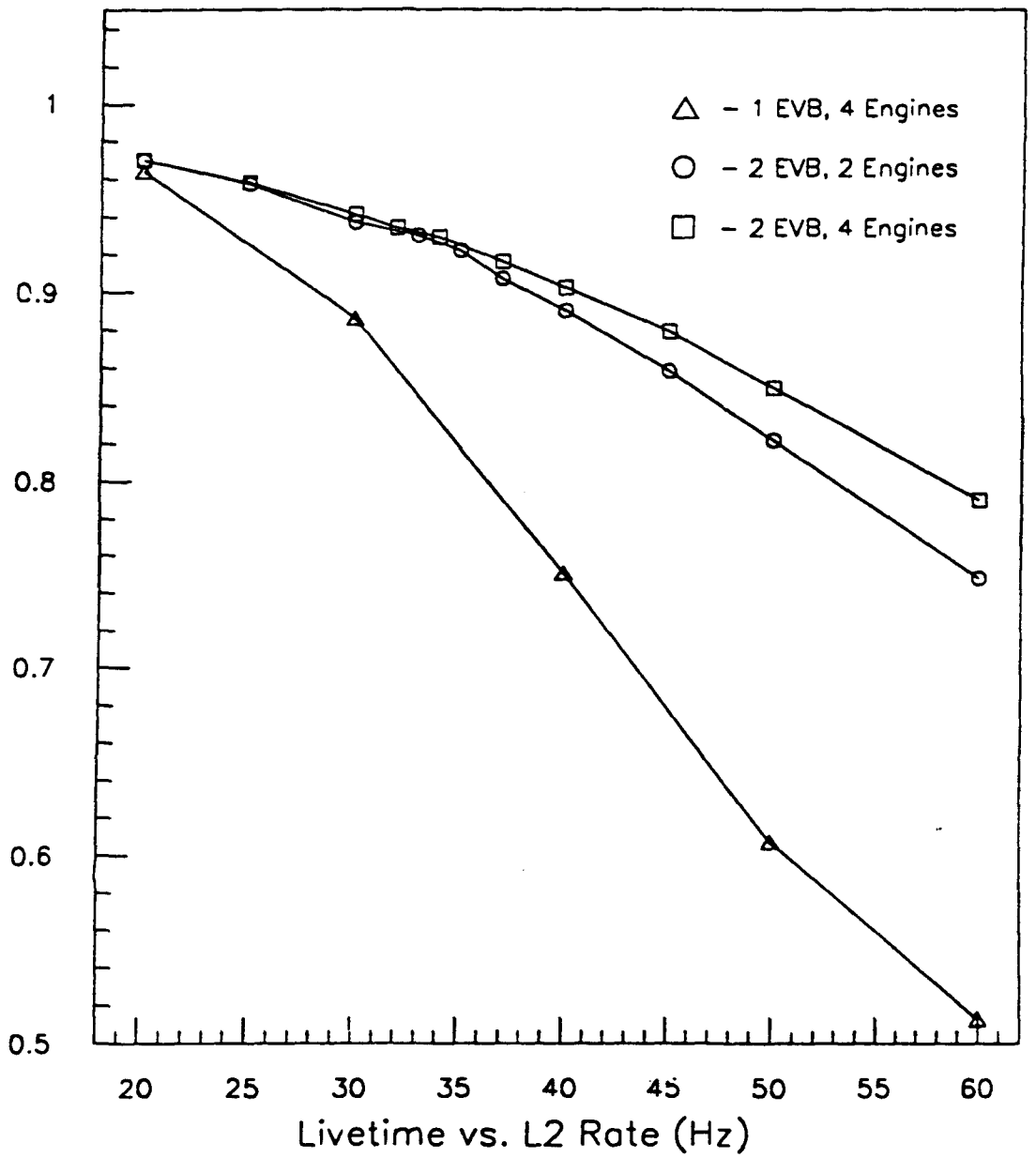


Figure 2: The combined scanner and buffering livetime of the CDF DAQ system operating with two Event Builders. This livetime only includes the loss of L2 triggers due to a buffer-full condition in the scanners and the loss of triggers when the scanners are digitizing. It does not include L1 or L2 trigger deadtime. We plot the rates separately for the three configurations discussed in the text, a single EVB with 4 engines, and two EVB's with both 2 engines and 4 engines.

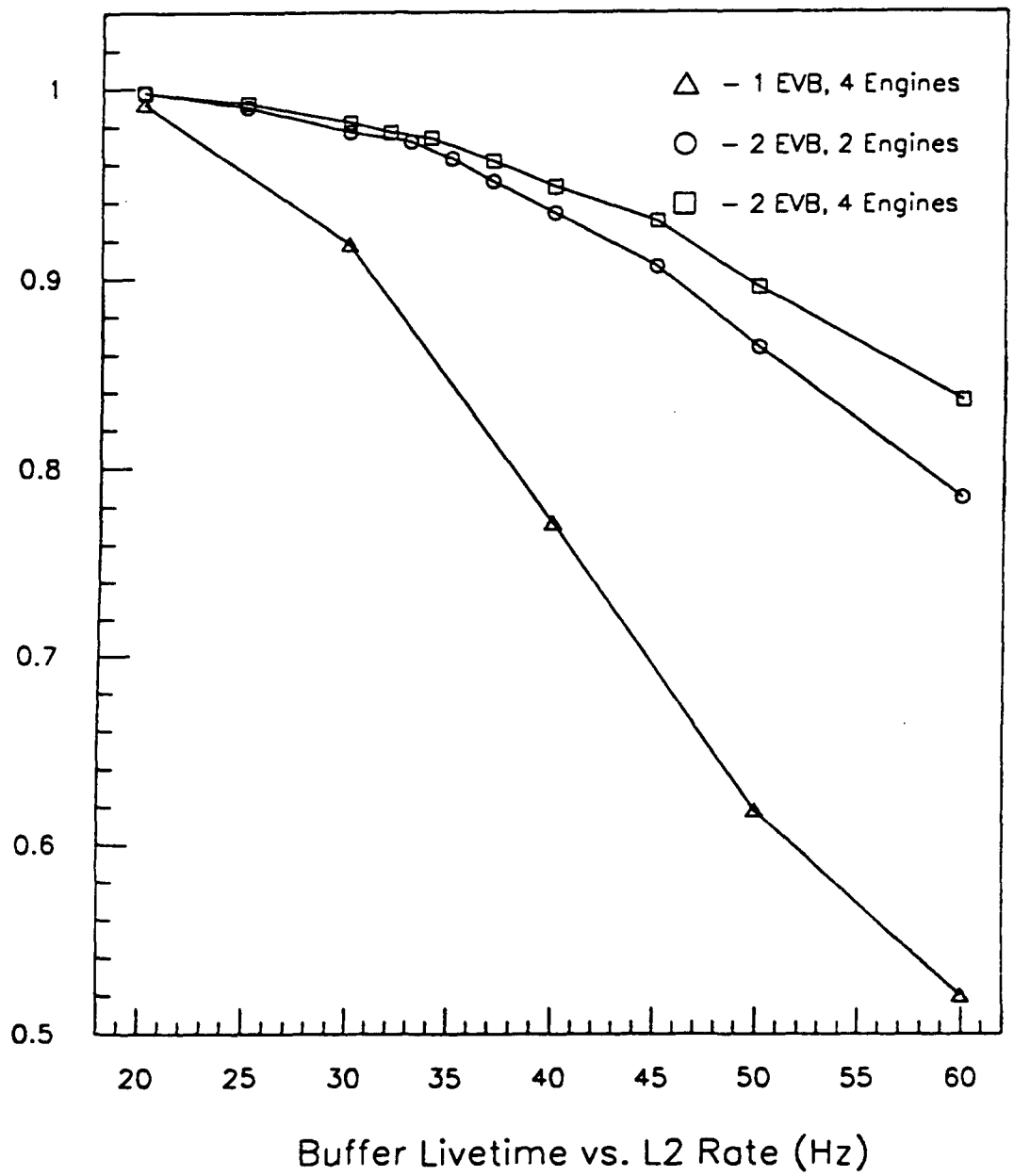
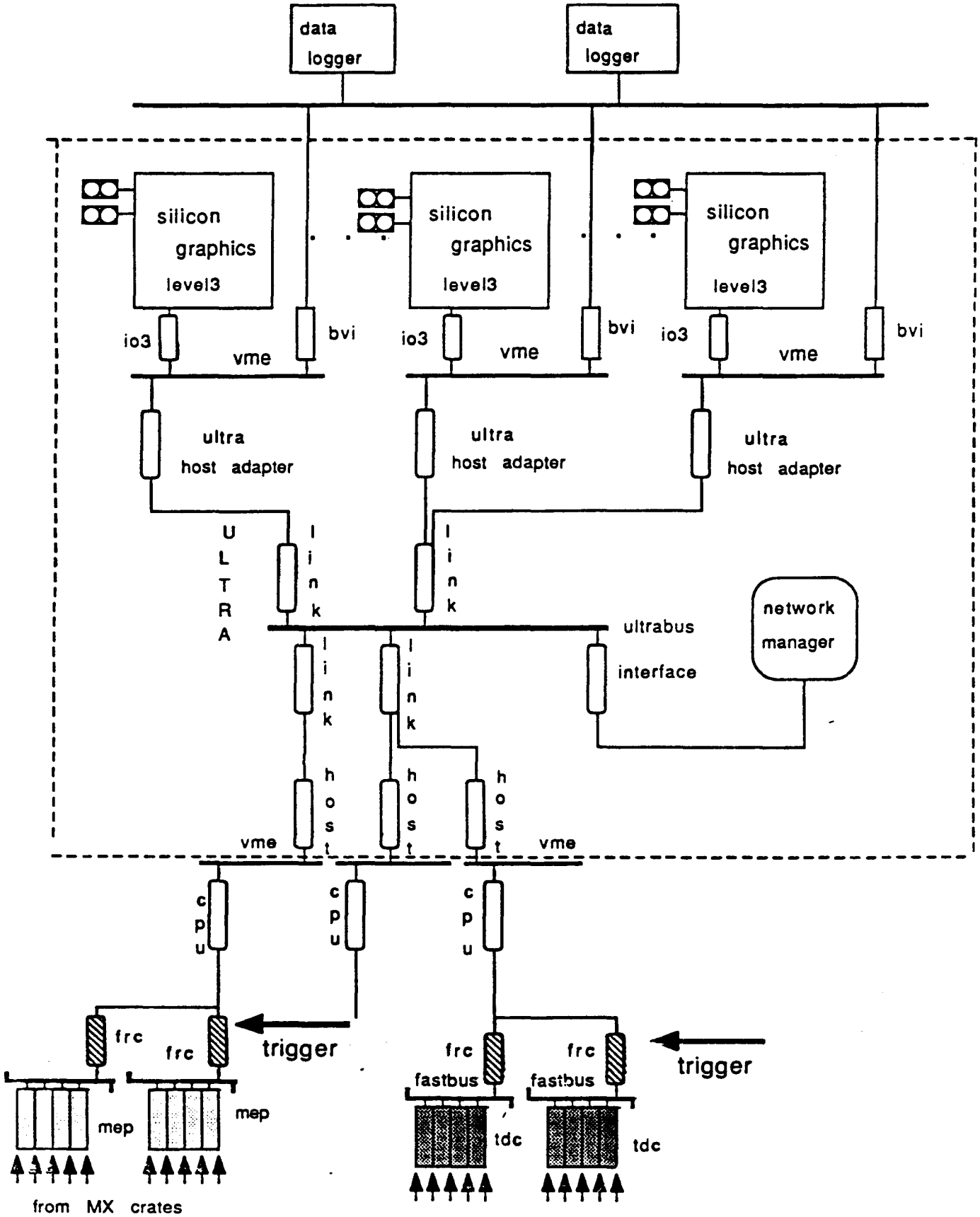


Figure 1: The buffering livetime of the CDF DAQ system operating with two Event Builders. This livetime only includes the loss of L2 triggers due to a buffer-full condition in the scanners. Thus, it does not include deadtime associated with scanner and trigger processing. We plot the rates separately for the three configurations discussed in the text, a single EVB with 4 engines, and two EVB's with both 2 engines and 4 engines.



Upgraded CDF DAQ system layout
figure 1

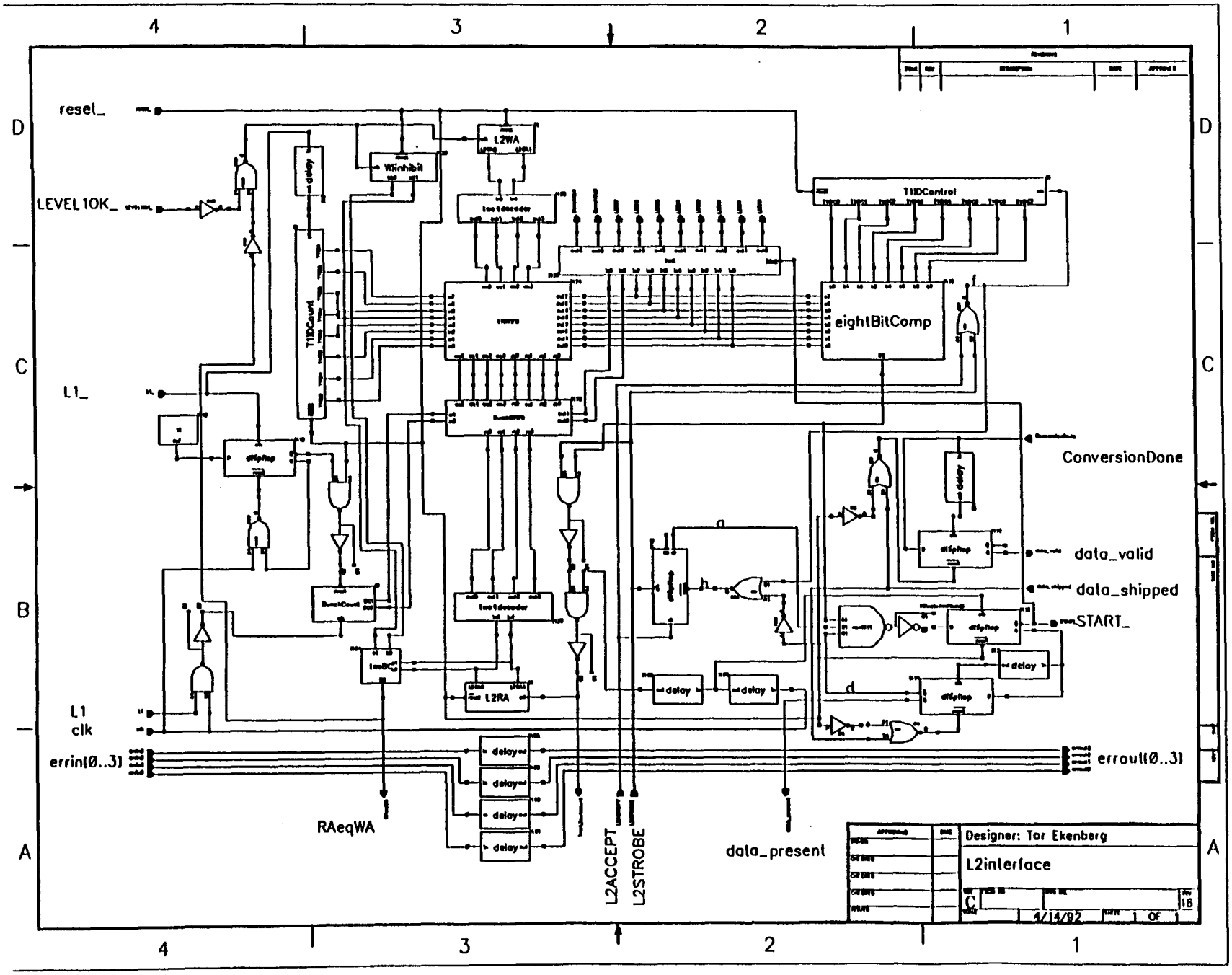
SIMULATION TO DESIGN WITH VERILOG

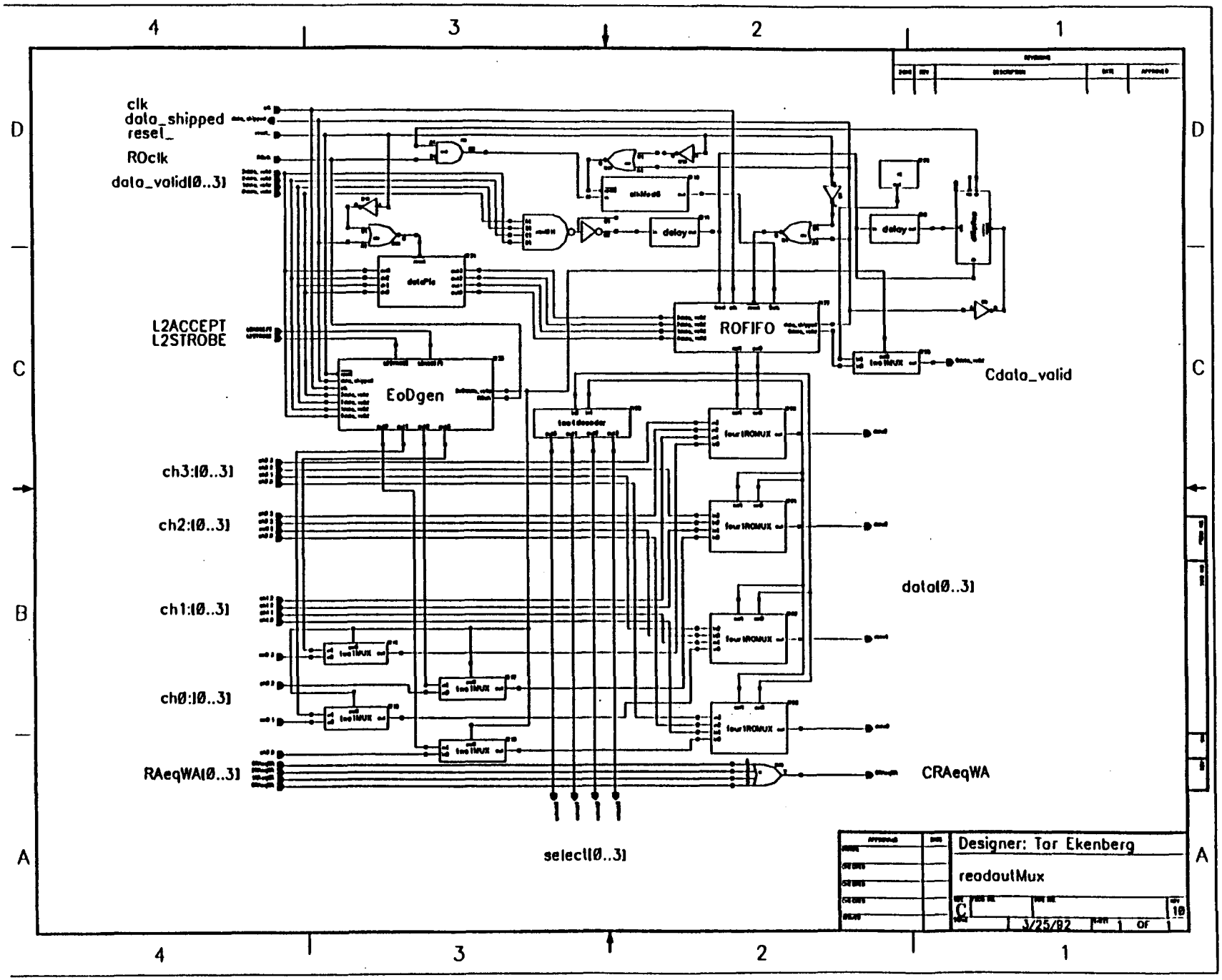
TOR EKENBERG
University of Pennsylvania

SSCL, 4/24/92

OUR PROBLEM

- DESIGN AN INTERFACE BETWEEN FRONT-END CHIP AND DCC/DAQ FOR "FULL DENSITY" TEST
- INCORPORATE A DETAILED MODEL OF THIS INTERFACE INTO MORE ABSTRACT DAQ/TRIGGERING SIMULATIONS
- HAVE THE ABILITY TO USE THE SAME "DESIGN" TO GENERATE NETLISTS FOR BOTH SIMULATIONS AND SILICON


resel_
LEVEL10K_
L1_
L1
clk
errin(0..3)
erroul(0..3)
L2WA
Winhibit
T1IDControl
eightBitComp
ConversionDone
data_valid
data_shipped
START_
L2RA
RAeqWA
L2ACCEPT
L2STROBE
data_present
Designer: Tor Ekenberg
L2interface
4/14/92


clk
data_shipped
resel_
ROclk
data_valid[0..3]
L2ACCEPT
L2STROBE
EoDgen
ROFIFO
Cdata_valid
ch3:[0..3]
ch2:[0..3]
ch1:[0..3]
ch0:[0..3]
RAeqWA[0..3]
data[0..3]
CRAeqWA
select[0..3]
Designer: Tor Ekenberg
readoutMux
3/25/92

VERILOG

- DIFFERENT LEVELS OF ABSTRACTION
  1) BEHAVIOR
  2) GATE
  3) TRANSISTOR

  ALL 3 LEVELS OF ABSTRACTION CAN BE INCLUDED IN THE SAME NETLIST

- EFFICENT SIMULATOR
- PREVIOUS WORK / EXPERIENCE
- CADENCE

CURRENT USE

- SIMULATION OF BLOCK DIAGRAM WITH BEHAVIORAL LEVEL CODE ASSOCIATED WITH EACH BLOCK VERIFIES LOGICAL OPERATION

- COMPARING BEHAVIOR SIMULATION WITH GATE-LEVEL SIMULATION VERIFIES DESIGN OF EACH BLOCK

STANDARD CELL LIBRARY

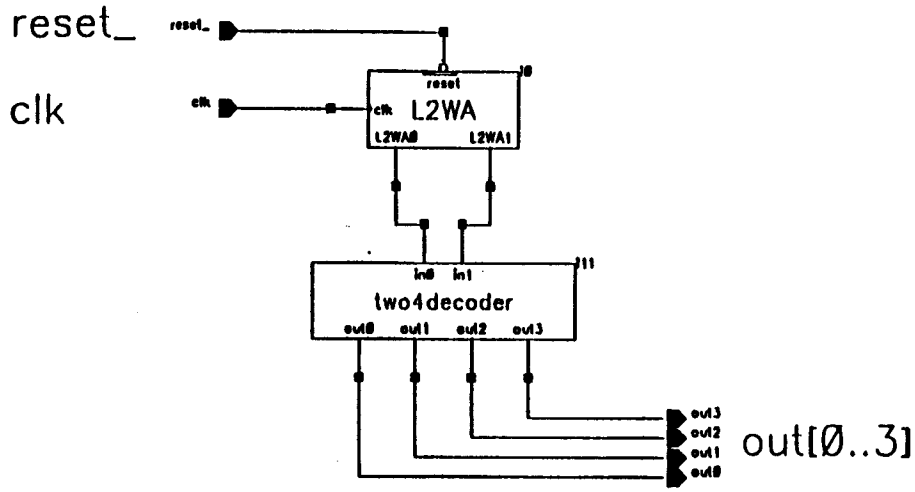
- dlmV3.0 : U OF MISS
  - LEAF CELLS (PRIMITIVES LIKE AND, NAND, NOR, ...)
  - COMPILED MODULES (COUNTERS, COMPARATORS, ...) USING LAGER IV

- WE HANDBUILT THE MODULES FROM LEAFCELLS TO GET BETTER CONTROL OF ASPECT RATIO

FSCM NO.	DWG NO.	SH	REV

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



APPROVALS	DATE
DRAWN	
CHECKED	
CHECKED	
CHECKED	
ISSUED	

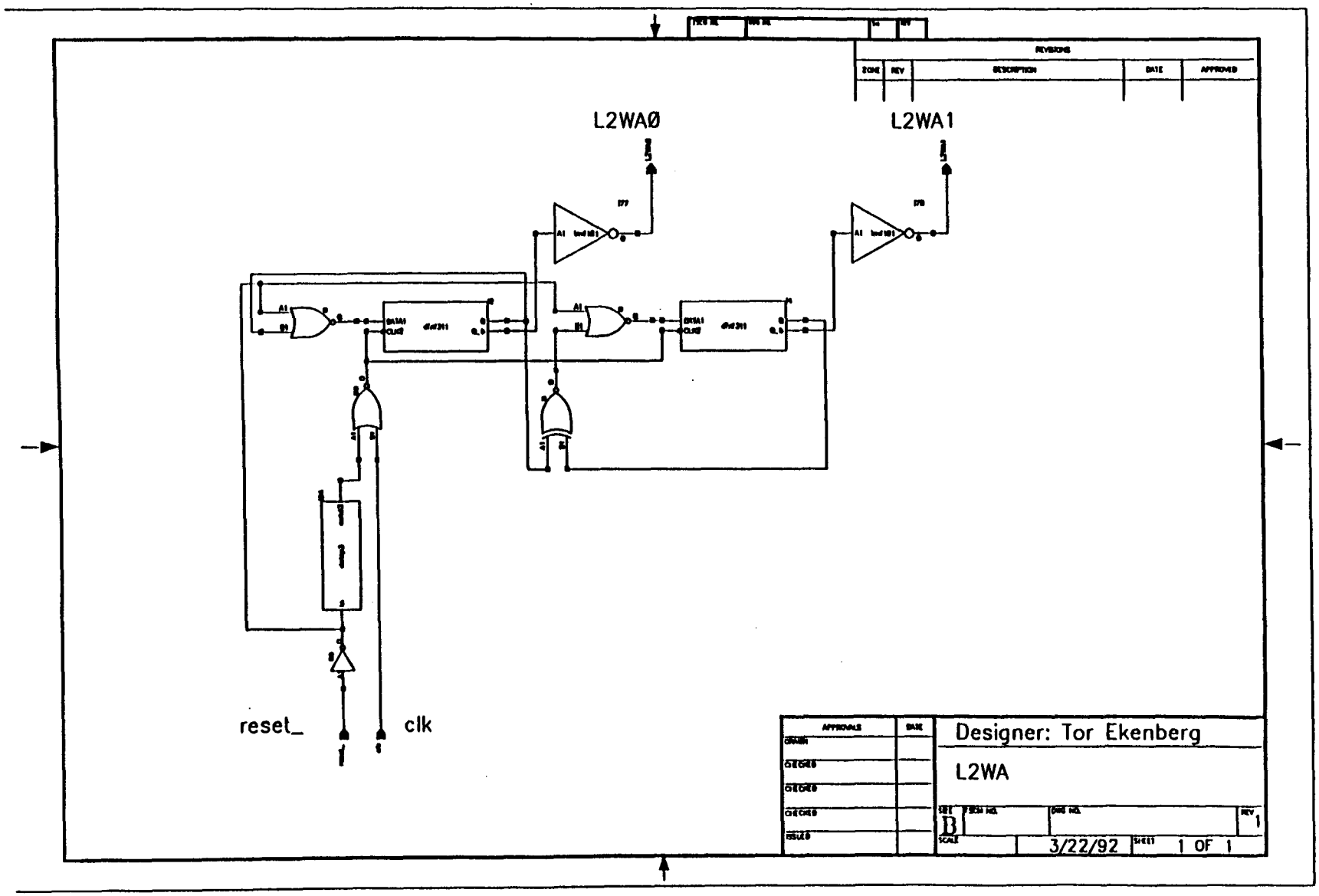
Designer: Tor Ekenberg

DaqEx

SIZE	FSCM NO.	DWG NO.	REV
A			

SCALE | SHEET OF

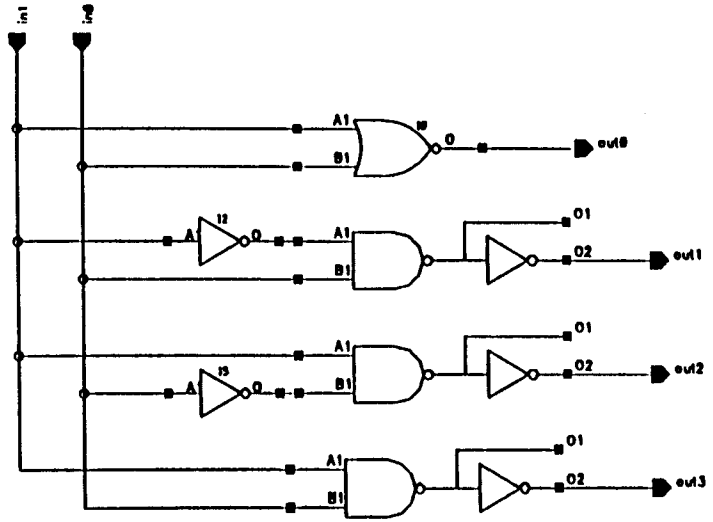
```
//
//      Behavioral level code for a two bit counter
//      that triggers on posistive going edge and has
//      asyncronous reset. This the L2WA.
//
//      file: ~ekenberg/vlsi/tvcamu/L2WA/L2WABverilog
//
//      library file: ~ekenberg/lib/verilog/modules/L2WA
//
module L2WA(L2WA0,L2WA1,clk,reset_);
        output L2WA0;
        output L2WA1;
        input clk;
        input reset_;

        wire clk;
        wire reset_;

        reg L2WA0;
        reg L2WA1;

        reg [0:1] twobitc;

        always @(posedge clk)
          begin
                if (reset_)
                  begin
                        twobitc = twobitc + 1;
                        L2WA0 = twobitc[1];
                        L2WA1 = twobitc[0];
                  end
                else
                  begin
                        twobitc = 0;
                        L2WA0 = twobitc[1];
                        L2WA1 = twobitc[0];
                  end
          end

        always @(negedge reset_)
          begin
                twobitc = 0;
                L2WA0 = twobitc[1];
                L2WA1 = twobitc[0];
          end

endmodule
```


L2WAØ
L2WA1
reset_
clk
Designer: Tor Ekenberg
L2WA
3/22/92
1 OF 1

FSCM NO.	DWG NO.	SH	REV

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED

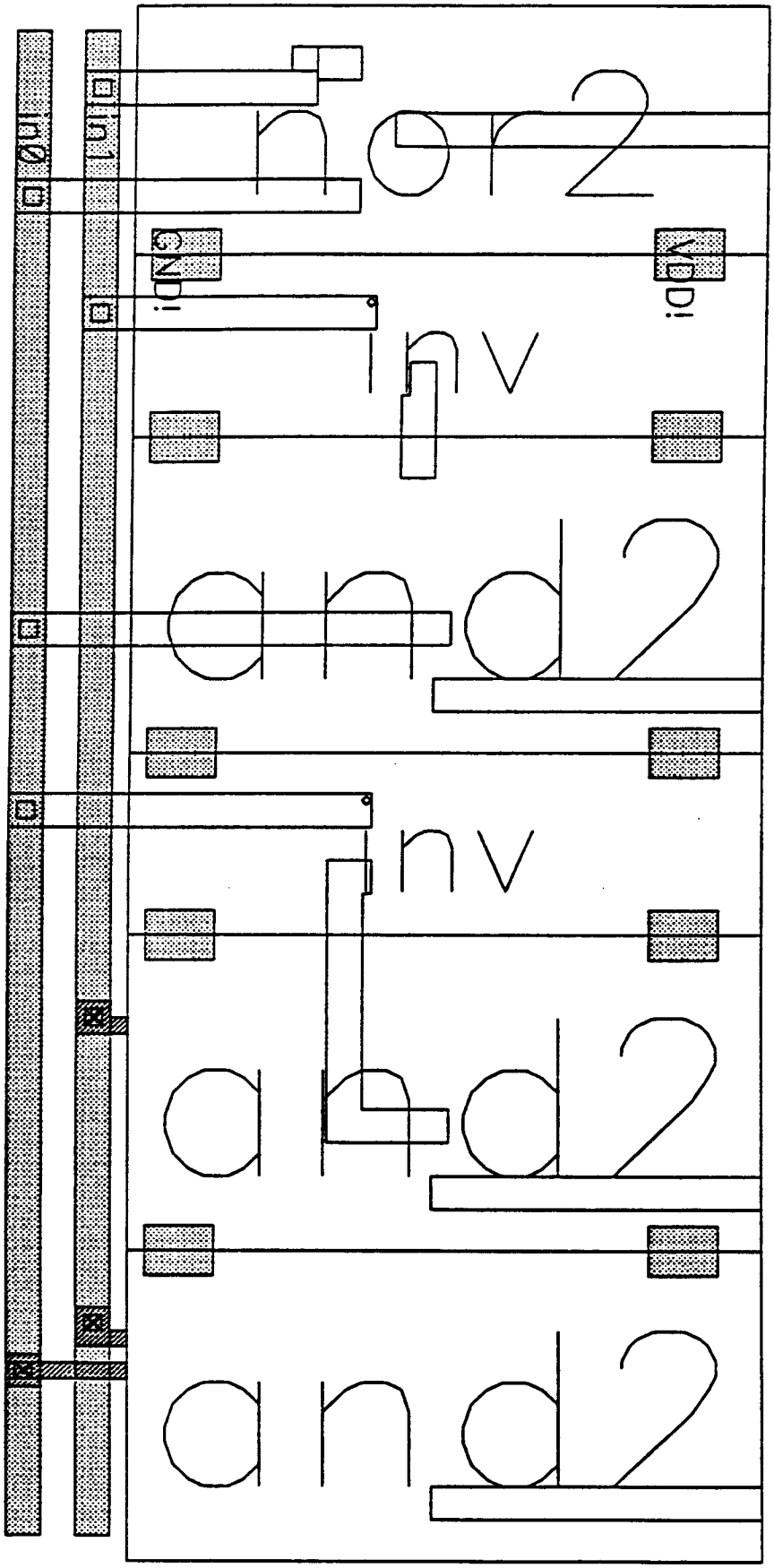


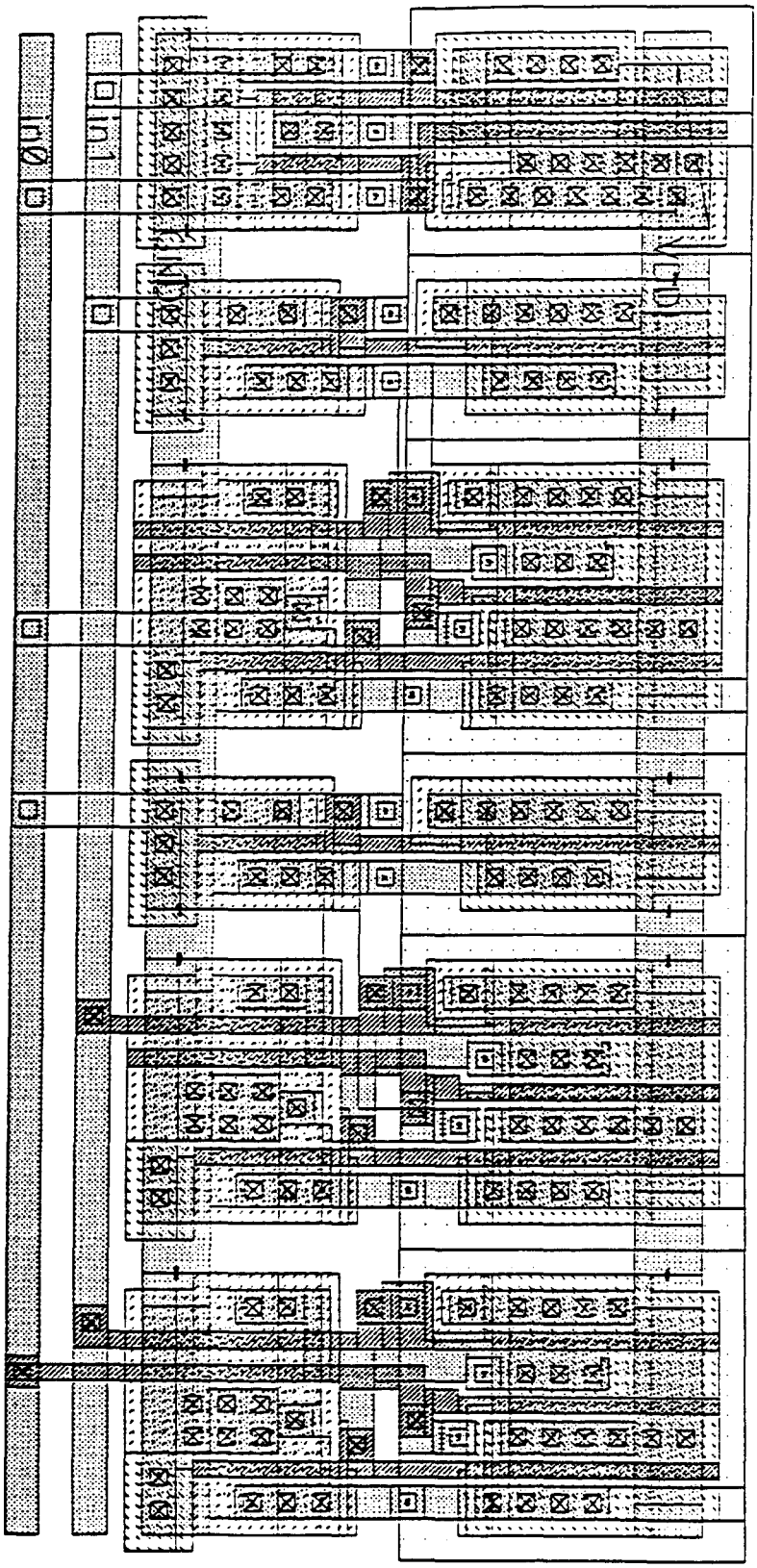
APPROVALS	DATE
DRAWN	
CHECKED	
CHECKED	
CHECKED	
ISSUED	

Designer: Tor Ekenberg

two4decoder

SIZE	FSCM NO.	DWG NO.	REV
A			1
SCALE	3/2/92	SHEET 1 OF 1	


in0
in1
GND
VDD
nor2
inv
and2
inv
and2
and2


OUT
IN1
IN2
VDD

```
// Verilog netlist of
//"/home/u2/ekenberg/vlsi/tvcamu/DaqEx"

// HDL models


// End HDL models

module DaqEx(out0, out1, out2, out3, clk, hnl_14);

        output out0, out1, out2, out3;

        input clk, hnl_14;

        two4decoder I11(out0, out1, out2, out3, hnl_15, hnl_16);
        L2WA I0(hnl_15, hnl_16, clk, hnl_14);

endmodule
```

```
// Verilog netlist of
//"/home/u2/ekenberg/vlsi/tvcamu/DaqEx"

// HDL models


// End HDL models


module nanf211 (O1, O2, A1, B1);
output O1, O2;
input A1, B1;
supply0 hnl_0;
supply1 hnl_1;
tranif1 hnl_2(O2, hnl_0, O1);
tranif1 hnl_3(hnl_4, hnl_0, B1);
tranif1 hnl_5(O1, hnl_4, A1);
tranif0 hnl_6(O2, hnl_1, O1);
tranif0 hnl_7(O1, hnl_1, B1);
tranif0 hnl_8(O1, hnl_1, A1);
endmodule

module two4decoder (out0, out1, out2, out3, in0, in1);
output out0, out1, out2, out3;
input in0, in1;
supply0 hnl_0;
supply1 hnl_1;
nanf211 I17(hnl_9, out1, hnl_10, in0);
nanf211 I16(hnl_11, out2, in1, hnl_12);
nanf211 I15(hnl_13, out3, in1, in0);
not I5(hnl_12, in0);
not I2(hnl_10, in1);
nor I0(out0, in1, in0);
endmodule

module DaqEx(out0, out1, out2, out3, clk, hnl_14);
output out0, out1, out2, out3;
input clk, hnl_14;
supply0 hnl_0;
supply1 hnl_1;
two4decoder I11(out0, out1, out2, out3, hnl_15, hnl_16);
L2WA I0(hnl_15, hnl_16, clk, hnl_14);
endmodule
```

- EXAMPLE OF DIFFERENT LEVELS OF ABSTRACTION IN SAME NETLIST

- BEHAVIORAL LEVEL CODE FOR SIMPLE BLOCKS EASY TO WRITE

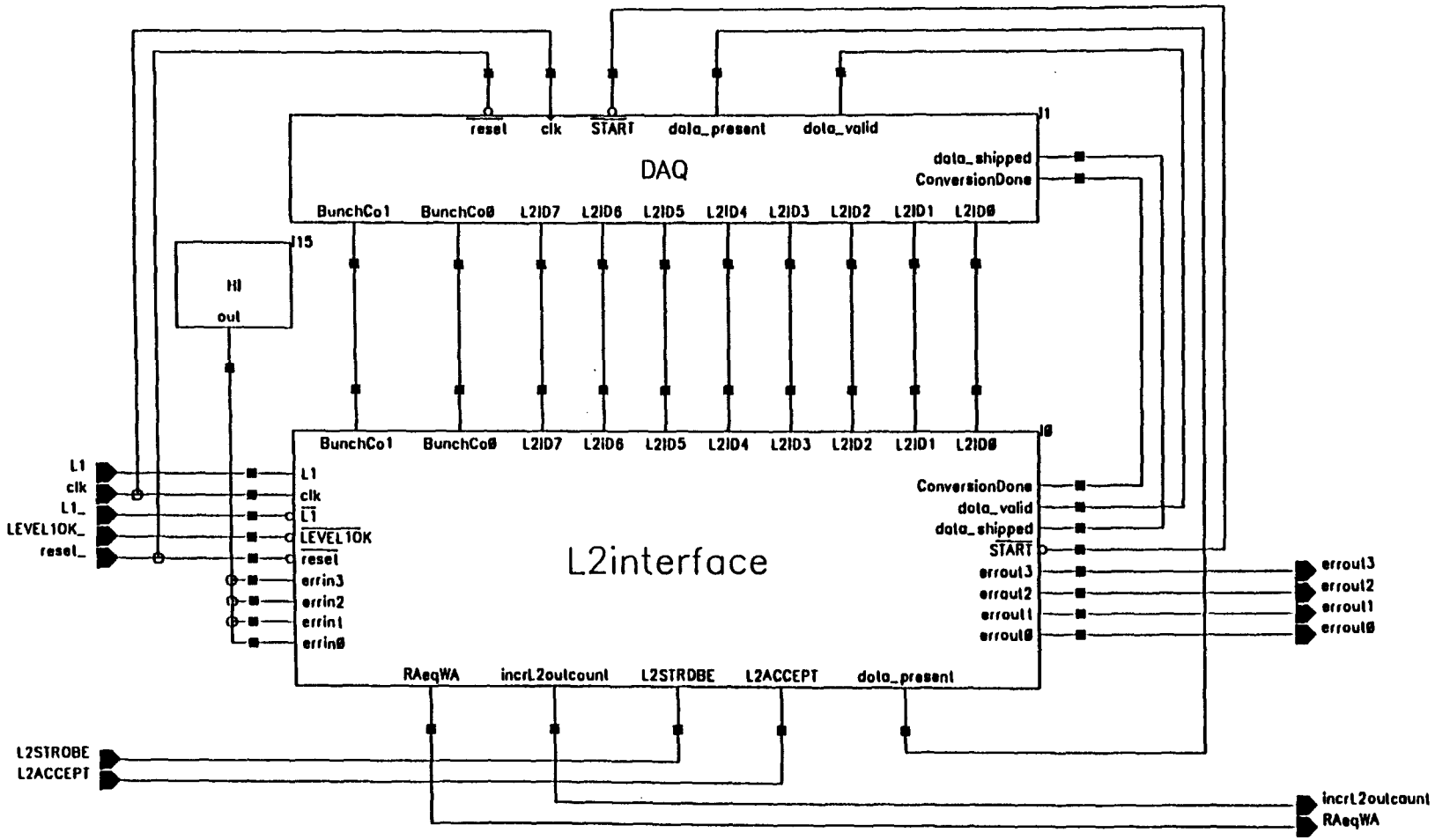
- SIMULATION EFFICENCY:

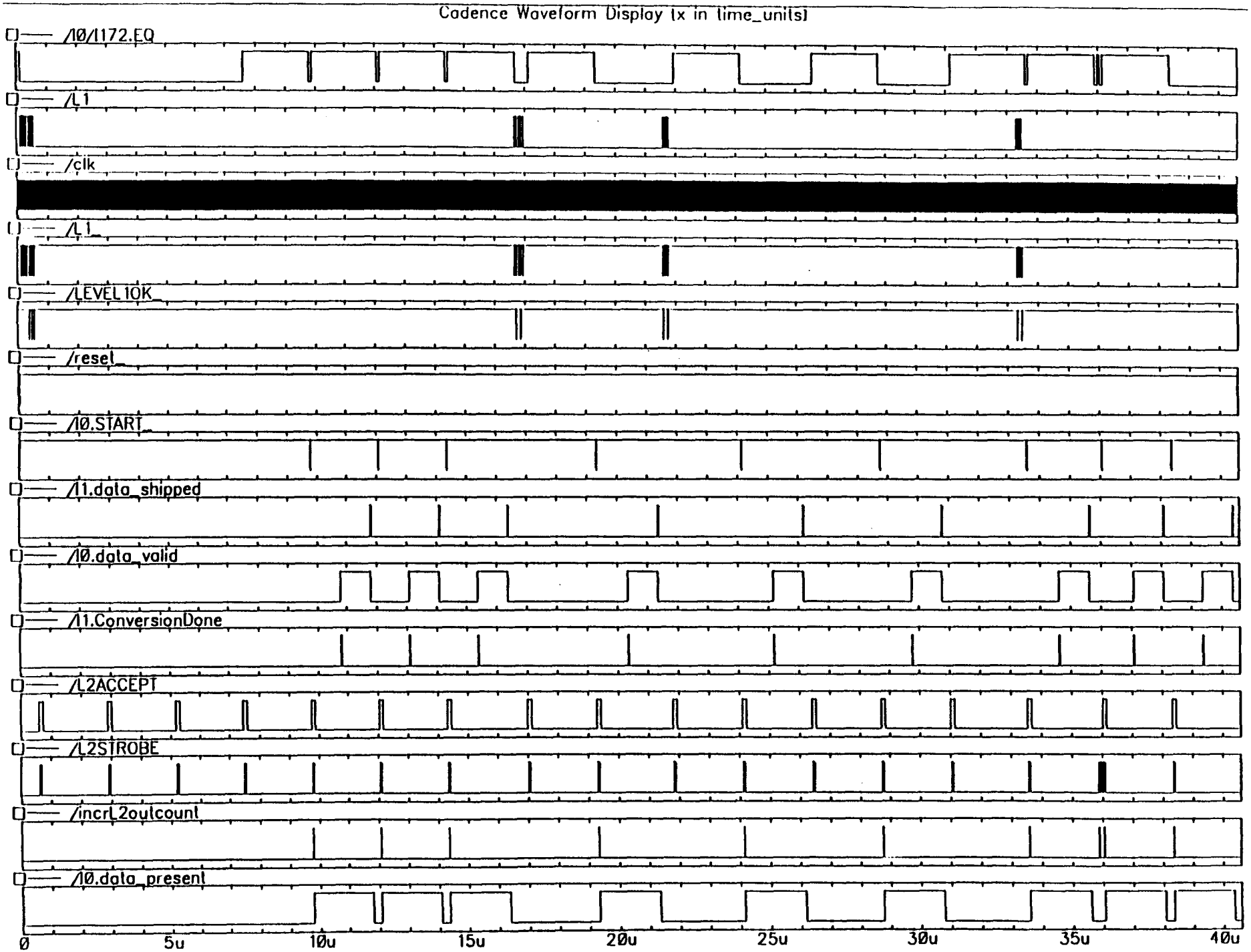
  300 μs of counting at 60MHz

  transistor level: 166 s

  gate level: 63 s

  behavioral level: 62 s


DAQ
reset
clk
START
data_present
data_valid
data_shipped
ConversionDone
BunchCo1
BunchCo0
L2ID7
L2ID6
L2ID5
L2ID4
L2ID3
L2ID2
L2ID1
L2ID0
J1
J15
HI
out
J0
L2interface
L1
clk
L1_
LEVEL1OK_
reset_
L1
clk
L1
LEVEL1OK
reset
errin3
errin2
errin1
errin0
ConversionDone
data_valid
data_shipped
START
errout3
errout2
errout1
errout0
RAeqWA
incrL2outcount
L2STROBE
L2ACCEPT
data_present
L2STROBE
L2ACCEPT
incrL2outcount
RAeqWA


Cadence Waveform Display (x in time_units)
/10/I172.EQ
/L1
/clk
/L1_
/LEVEL10K_
/resel_
/10.START_
/11.data_shipped
/10.data_valid
/11.ConversionDone
/L2ACCEPT
/L2STROBE
/incrL2outcount
/10.data_present
0
5u
10u
15u
20u
25u
30u
35u
40u

```
VERILOG-XL 1.5c log file created Apr 22, 1992  15:17:54
  * Copyright Cadence Design Systems, Inc. 1985, 1988.    *
  *     All Rights Reserved.       Licensed Software.     *
  * Confidential and proprietary information which is the *
  *      property of Cadence Design Systems, Inc.          *
Verilog 1.5c with Edge additionsCompiling source file "si.inp"
Compiling source file "verisave.cmdfile"
Scanning library directory "/home/u2/ekenberg/lib/verilog/modules"
Highest level modules:
test
save_mod

START_            0 =    4 (0000000100)
START_            1 =    5 (0000000101)
START_            2 =    6 (0000000110)
START_            3 =    8 (0000001000)
START_            4 =   10 (0000001010)
START_            5 =   12 (0000001100)
START_            6 =   14 (0000001110)
START_            7 =   17 (0000010001)
START_            8 =   18 (0000010010)
L5100 "si.inp": $finish at simulation time 40608
2370404 simulation events
CPU time: 2 secs to compile + 0 secs to link + 108 secs in simulation
End of VERILOG-XL 1.5c   Apr 22, 1992  15:19:47
```

EXAMPLE OF INCORPORATING DETAILED DESIGNS INTO MORE ABSTRACT SIMULATIONS.

A SIMPLE WAY TO TEST NEW CONCEPTS TOGETHER WITH EXISTING DESIGNS

TIMING SIMULATIONS

- GATE LEVEL SIMULATIONS SUPLEMENTED BY SPICE-LIKE SIMULATIONS.

- CHARACTERIZE EACH BLOCK WITH DETAILED CIRCUIT SIMULATION ⟹ USE THE RESULT IN VERILOG

- FIRST ORDER RESULTS

SUMMARY & CONCLUSIONS

- BEHAVIOR LEVEL VERILOG
  1) EASY AND EFFICENT TO SIMULATE CIRCUIT ON BLOCK DIAGRAM LEVEL FOR LOGICAL FUNCTIONALITY
  2) CAN INTERFACE TO MORE ABSTRACT SIMULATIONS OF DAQ / TRIGGERING
- GATE LEVEL VERILOG
  1) EFFICENT WAY OF SIMULATING TIMING OF LARGE CIRCUITS

- MIXED BEHAVIORAL / GATE LEVEL SIMULATIONS

  1) SIMPLE WAY TO TEST NEW CIRCUIT CONCEPTS (REPRESENTED BY BLOCKS OF BEHAVIORAL LEVEL CODE) TOGETHER WITH EXISTING DESIGNS

The DZero Data Acquisition System: Model and Measurements

J.A. Wightman
Texas A&M University

R. Angstadt, M.E. Johnson, and I.L. Manning
Fermi National Accelerator Laboratory

Workshop on Data Acquisition and Trigger System Simulations for High Energy Physics
April 24, 1992

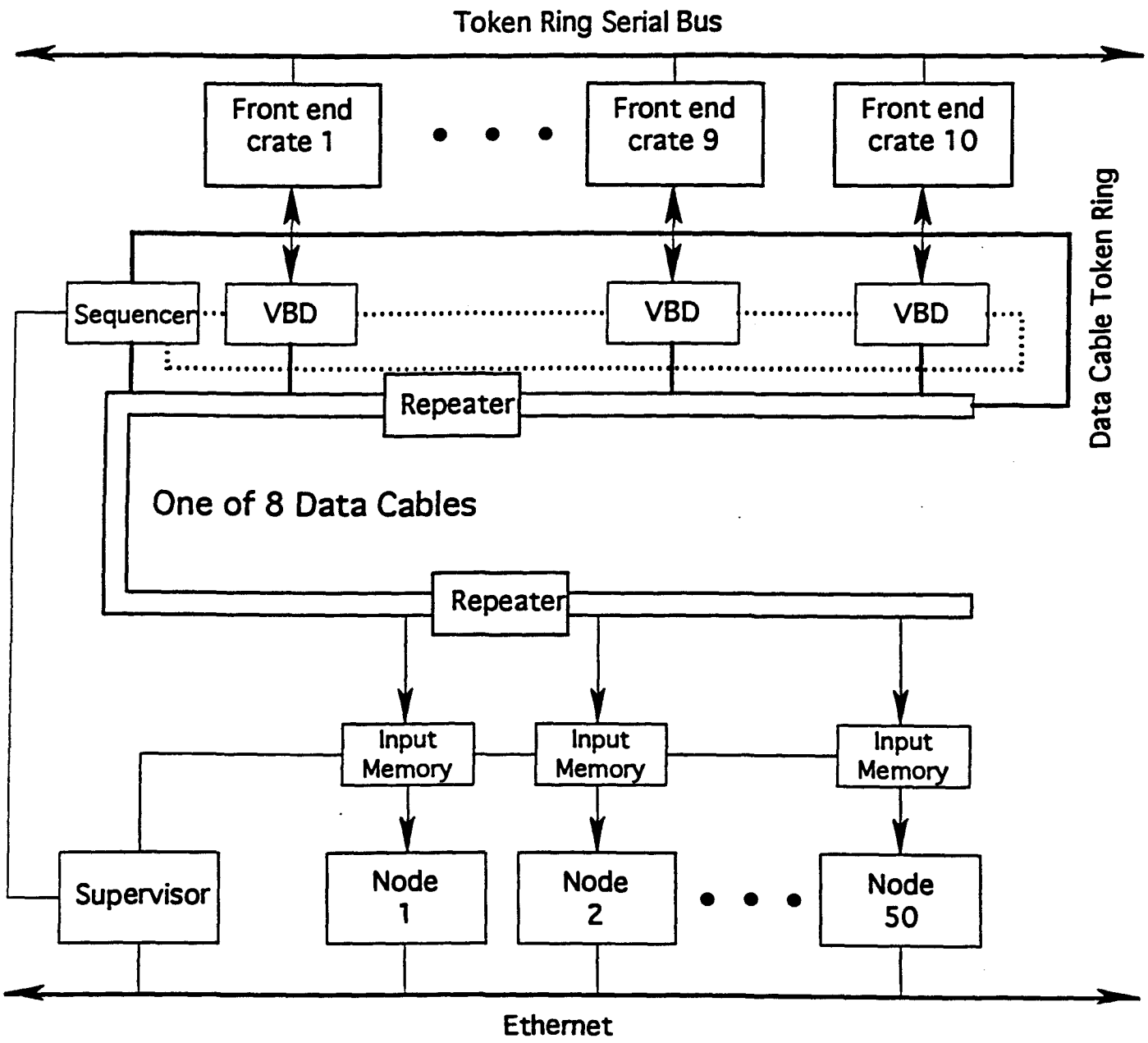
1. The DZero Data Acquisition Hardware
2. Model of the DZero DAQ System
3. Understanding the RESQ model
4. Prospects for the Future
5. Conclusions

The DZero Data Acquisition Hardware

Before describing the DAQ System, we need to discuss the ground work on which the rest of this talk is based:

① While the DAQ System, in general, is a very comprehensive term, encompassing all the hardware from the point at which the data is generated by the detector through the readout electronics to the Level 2 software trigger (residing in a μVAX farm) up to the Host computer and finally on to 8 mm tape, we will confine our attention to that portion of the hardware from the readout electronics to the Level 2 μVAX farm.

② All of our modeling efforts to date have been centered on the Central Detector (CD) electronics in part because this was the first system to be designed but predominantly because Monte Carlo studies showed the CD would produce, by far, the most data for any given event with the Vertex Detector (VTX) leading the way. Therefore, the VTX data cable is the best place to study bottlenecks in this part of the DAQ system.


Token Ring Serial Bus
Front end crate 1
Front end crate 9
Front end crate 10
Data Cable Token Ring
Sequencer
VBD
VBD
VBD
Repeater
One of 8 Data Cables
Repeater
Input Memory
Input Memory
Input Memory
Supervisor
Node 1
Node 2
Node 50
Ethernet

Front-End Crates – VME (Flash ADC or FADC)
13 MBytes/sec Bandwidth
2 Back-end buffers
Token Ring serial bus Control Path

Readout from VME to the Data Cable – VME Buffer, Driver (VBD)
2 Buffers

Data Cable is a 32-bit data path
40 MBytes/sec Bandwidth
Crate read out onto Data Cable is serial
Controlled by the Sequencer

Level 1 sends trigger information to the Supervisor
who determines if there is a free Level 2 node
Supervisor communicates trigger information to the
Sequencer μVAX
Sequencer μVAX formulates tokens and signals
each Sequencer board to circulate its token
and thus read out its crates on the Data Cable
Supervisor enables the input memory of the chosen
Level 2 node to strobe the data off the
Data Cable

The DZero DAQ System is a hybrid architecture, that is, it is push-pull

The Front-end crates are push
the FADC's raise SLAVE READY when ready to transfer the data

The VBD read out on the Data Cable is pull
this results from the token control of the read out

Model of the DZero DAQ System

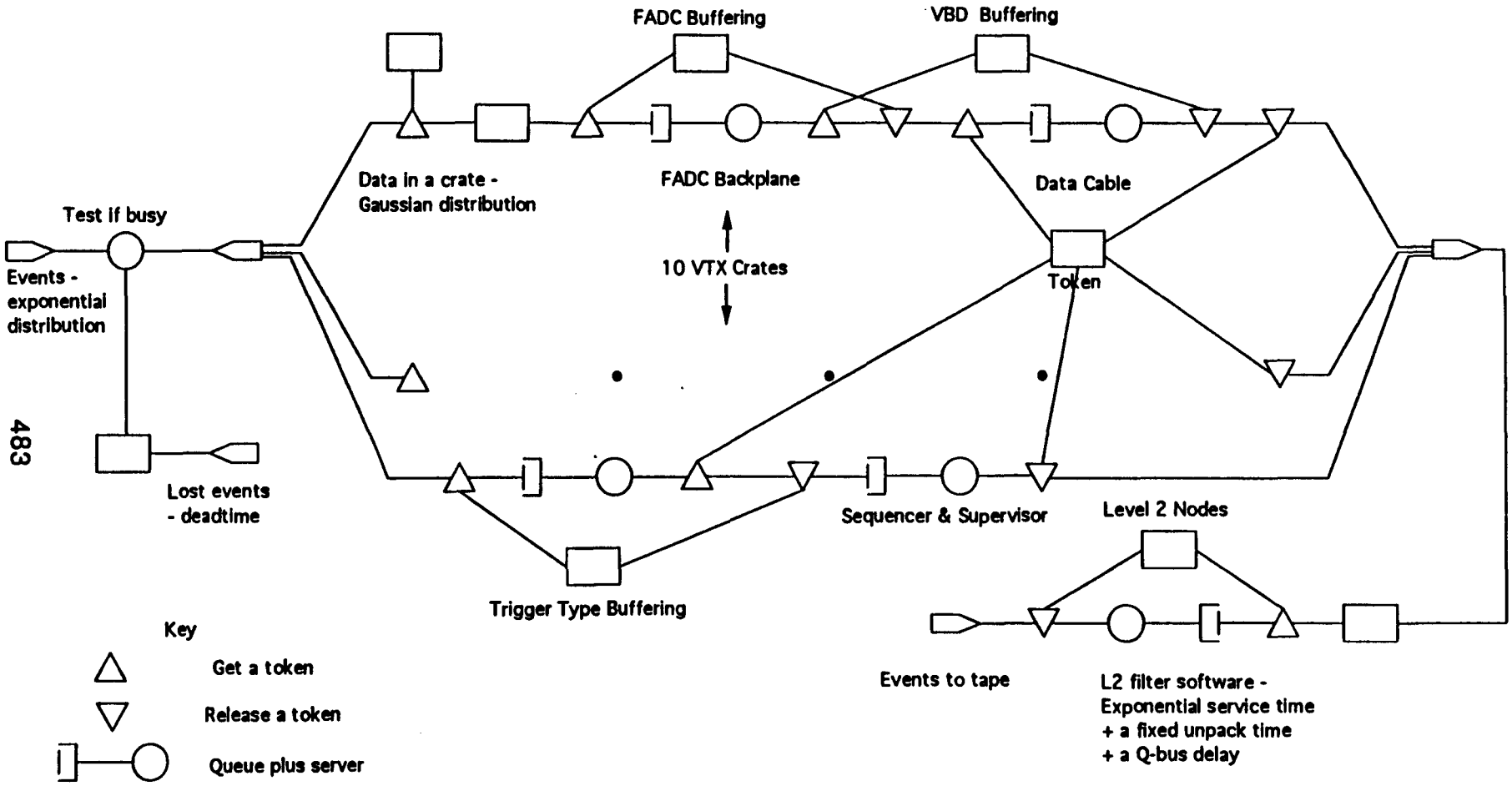
We have used IBM's Research Queuing Package Version 2 (RESQ)
- this is a Monte Carlo simulation based on queuing theory
- this package is used to model computer networks
- it uses Poisson arrival times
- we use Gaussian event sizes based on GEANT simulations of the DZero data

Each crate is modeled as 2 queues in series
- one for the buffers of the FADC's
- one for the buffers of the VBD's

The crates are all handled in parallel

We have modeled the Supervisor and Sequencer as a queue in parallel with the crates because this framework keeps track of the 4 events we can have buffered in the FADC + VBD

Level 2, while having 50 nodes, is modeled as a single queue because all nodes are equally likely to receive an event



RESQ Model For The Simulation Of The VTX Readout

Understanding the RESQ Model
— The Analytical Solution —

One major problem with using a modeling package is that it is a "black box"

Our analytical solution assumes a Markov Process
we write down all possible states of the system

P_{ffwn}

where ff is the occupancy of the double buffer in the FADC
w is the occupancy of the double buffer in the VBD
n is the occupancy of the Level 2 nodes

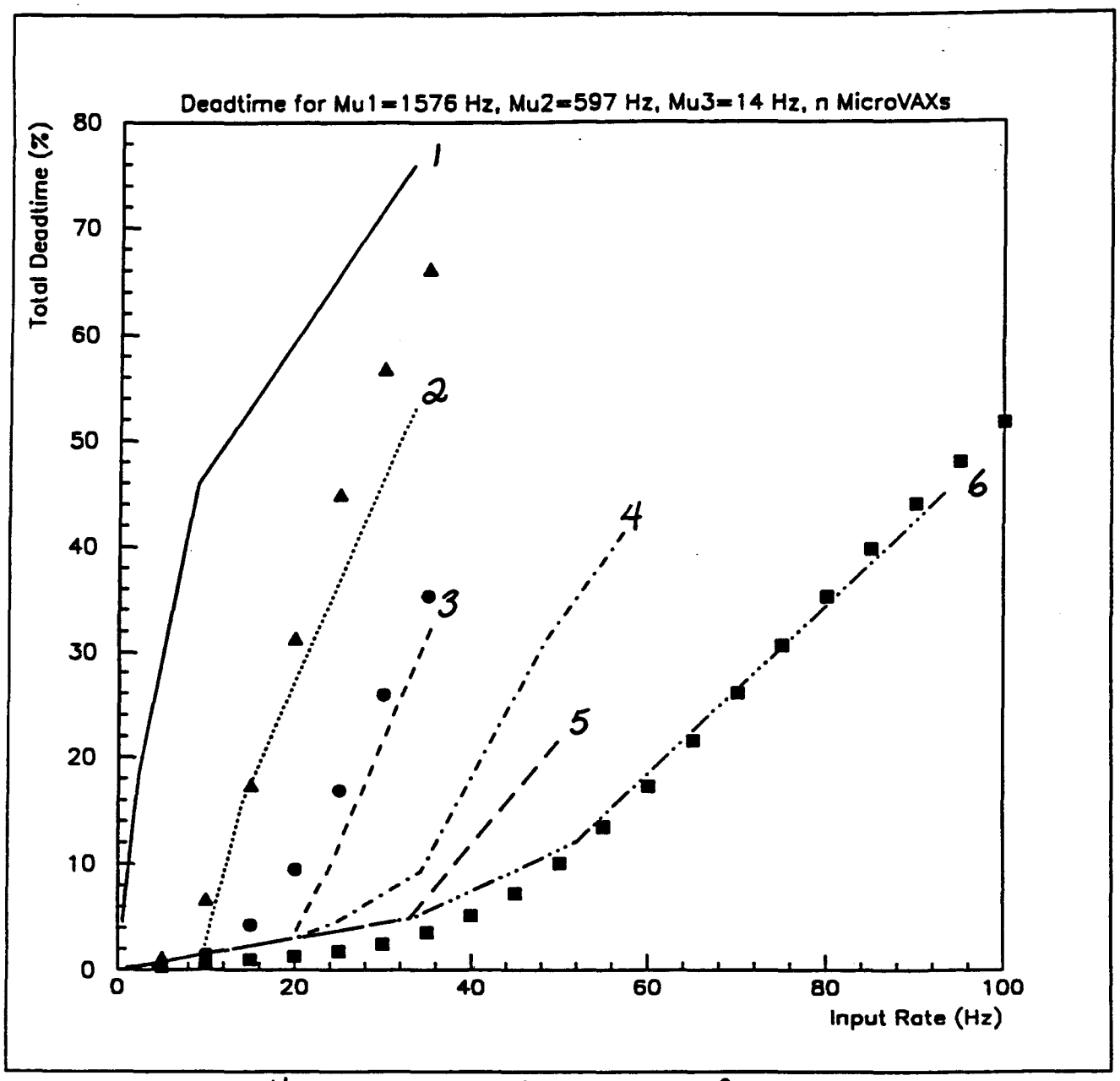
we input
λ the input event rate
μ_1 the transfer rate from FADC to VBD
μ_2 the transfer rate from VBD to Level 2
μ_3 the processing rate in Level 2

Balance equations are written and linearized and the steady-state solution is found

For 50 Level 2 nodes, we have 569 states
+1 eqn : $\Sigma P = 1$

Direct solution (invert) $\longrightarrow$ $1\frac{1}{2}$ hours 3100/76

Iterative Solution (sparse) $\longrightarrow$ $\lesssim$ 2 minutes 3100/76



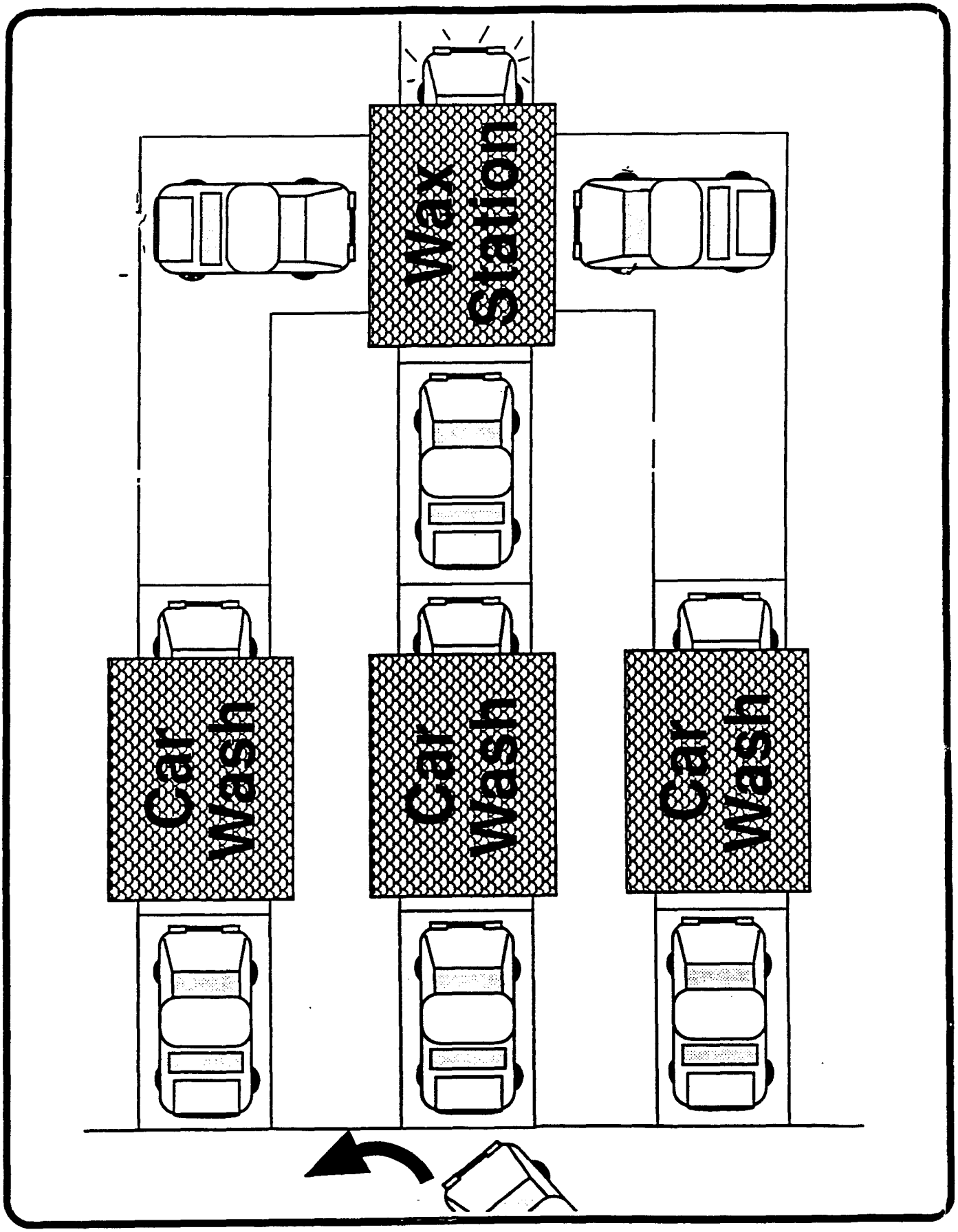
Lines are the measured dead times for 1-6 Level 2 Nodes in the system

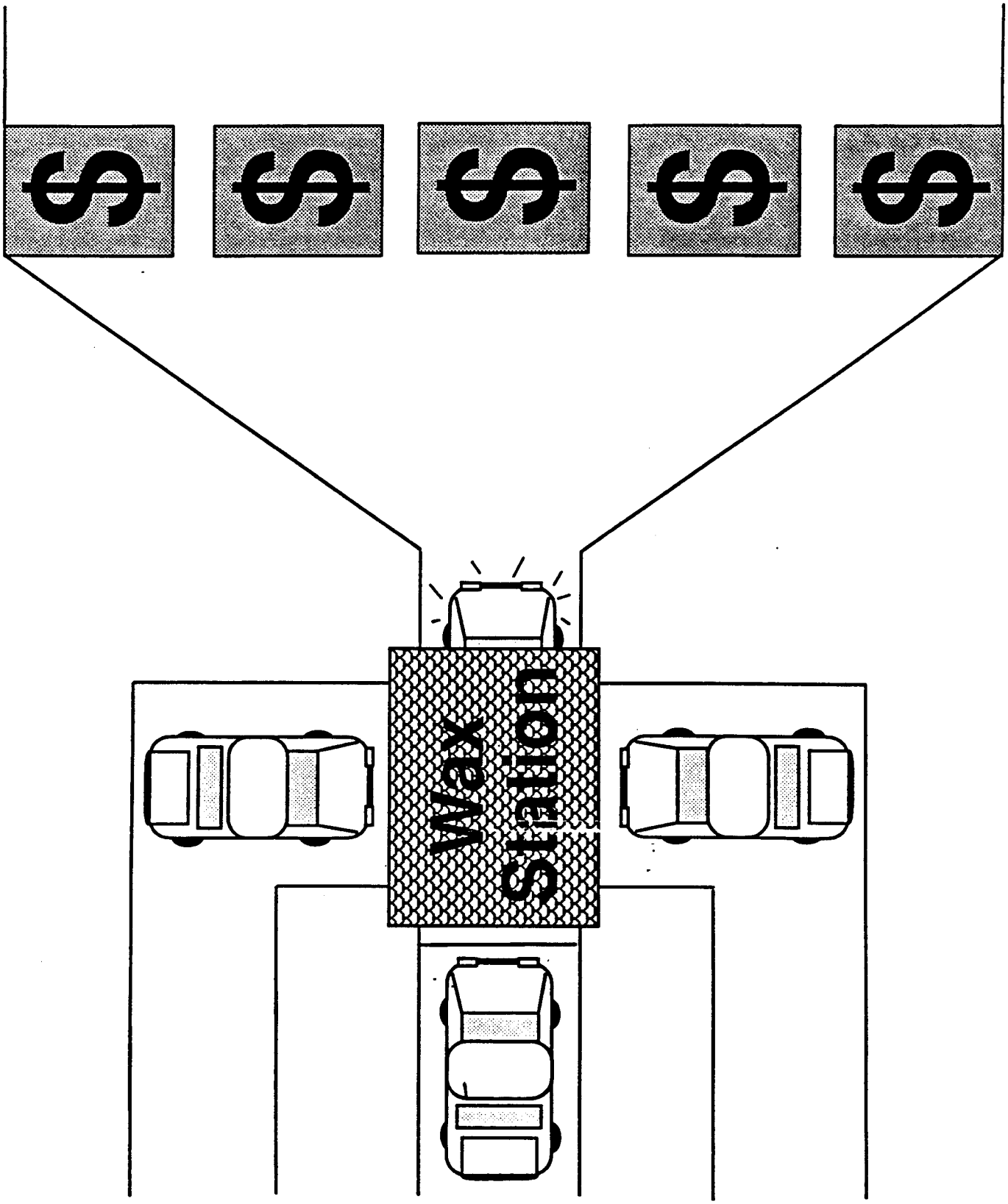
Points are the analytical solution for

▲ 2 Level 2 Nodes

● 3 Level 2 Nodes

■ 6 Level 2 Nodes


Liquid Wax
Car Wash
Car Wash
Car Wash


MAXIMUM
SECURITY

The DZero "Car Wash"

RESQ	"Car Wash"
Front-end (FADC) 2 Buffers	1 wash station with 1 parking space in front
VBD 2 Buffers	2 parking spaces in front of 1 wax station
Level 2 Nodes 50	50 cashiers

Prospects for the Future

The DZero DAQ System is undergoing upgrades as much of the software control in the Supervisor and Sequencer is replaced with hardware

The Level 2 trigger is a software trigger with ~200 Hz into Level 2 and ~1 Hz out so with 50 nodes → ~4 Hz processing in each Level 2 node

<u>Question</u>: Is this good enough?

We use the analytical solution to study Level 2 processing rates (μ_3)

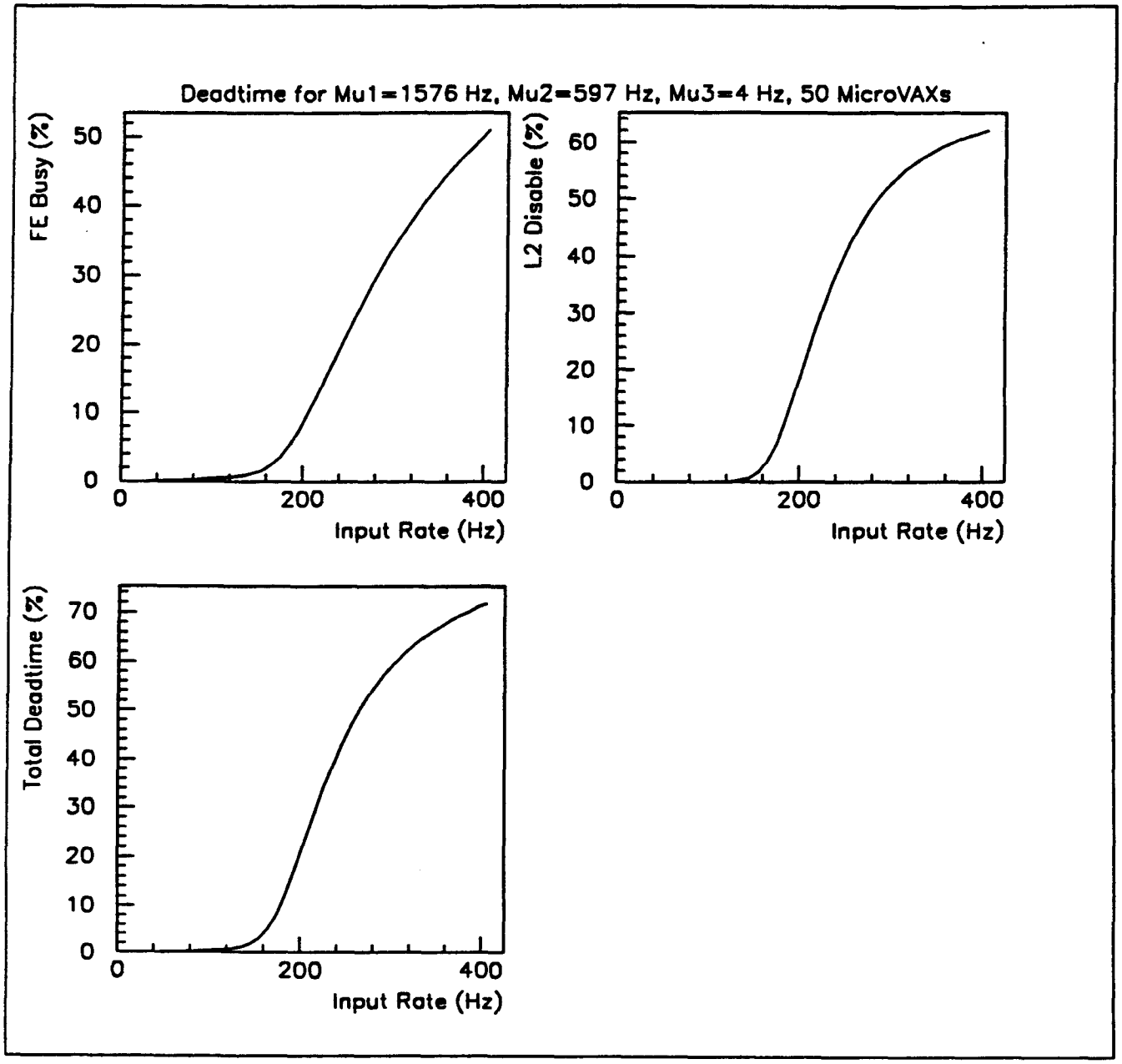
We have 2 sources for inhibiting triggers

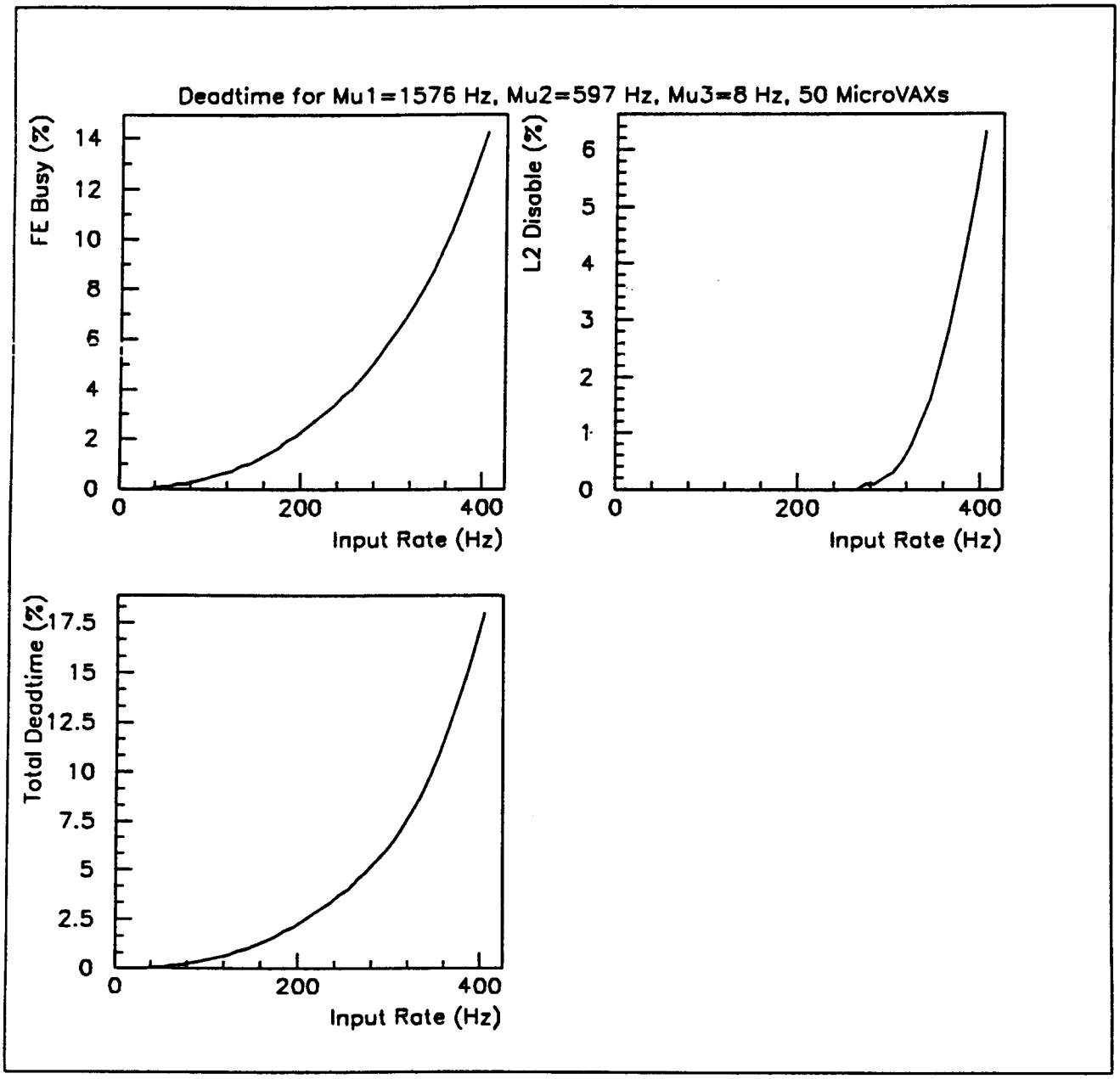
1.) Front-End Busy

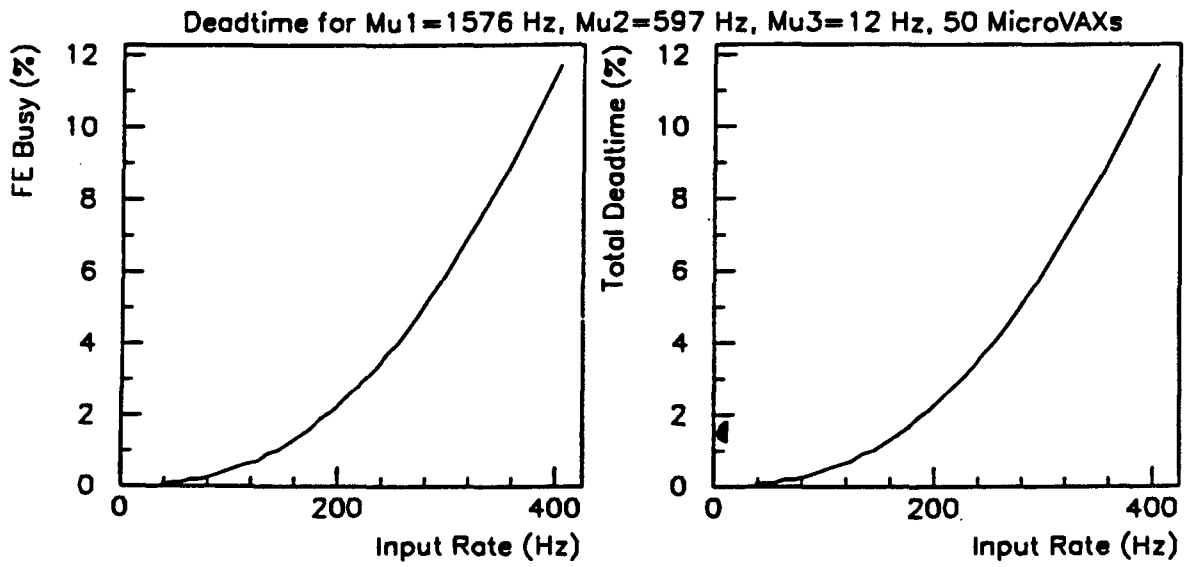
Hardware driven by not having a free buffer in the FADC's

2.) L2 Disable

Software driven by the Supervisor when there is no free Level 2 node.


Deadtime for Mu1=1576 Hz, Mu2=597 Hz, Mu3=4 Hz, 50 MicroVAXs
FE Busy (%)
0
10
20
30
40
50
0
200
400
Input Rate (Hz)
L2 Disable (%)
0
10
20
30
40
50
60
0
200
400
Input Rate (Hz)
Total Deadtime (%)
0
10
20
30
40
50
60
70
0
200
400
Input Rate (Hz)


Deadtime for Mu1=1576 Hz, Mu2=597 Hz, Mu3=8 Hz, 50 MicroVAXs
FE Busy (%)
14
12
10
8
6
4
2
0
0
200
400
Input Rate (Hz)
L2 Disable (%)
6
5
4
3
2
1
0
0
200
400
Input Rate (Hz)
Total Deadtime (%)
17.5
15
12.5
10
7.5
5
2.5
0
0
200
400
Input Rate (Hz)


Deadtime for Mu1=1576 Hz, Mu2=597 Hz, Mu3=12 Hz, 50 MicroVAXs
FE Busy (%)
12
10
8
6
4
2
0
0
200
400
Input Rate (Hz)
Total Deadtime (%)
12
10
8
6
4
2
0
0
200
400
Input Rate (Hz)

Conclusions

① Modeling results are meaningless if you don't understand what it is that you are trying to simulate

② The RESQ model provides us with a good understanding of the DZero DAQ System from a global point of view.

LEVEL 1.5 MODELING

M. JOHNSON, I. MANNING, R. ANGSTADT.
FERMILAB

J. WIGHTMAN TEXAS A&M.

INITIAL DØ DESIGN WAS A FAST (ONE CROSSING, 3.5μS) HARDWARE TRIGGER (LEVEL I) AND A MICROPROCESSOR FARM (LEVEL II).

IT BECAME OBVIOUS THAT SOME LEVEL OF HARDWARE TRIGGER WITH A TIME SCALE A FEW CROSSINGS (10-30μS) WOULD IMPROVE THE DETECTOR.

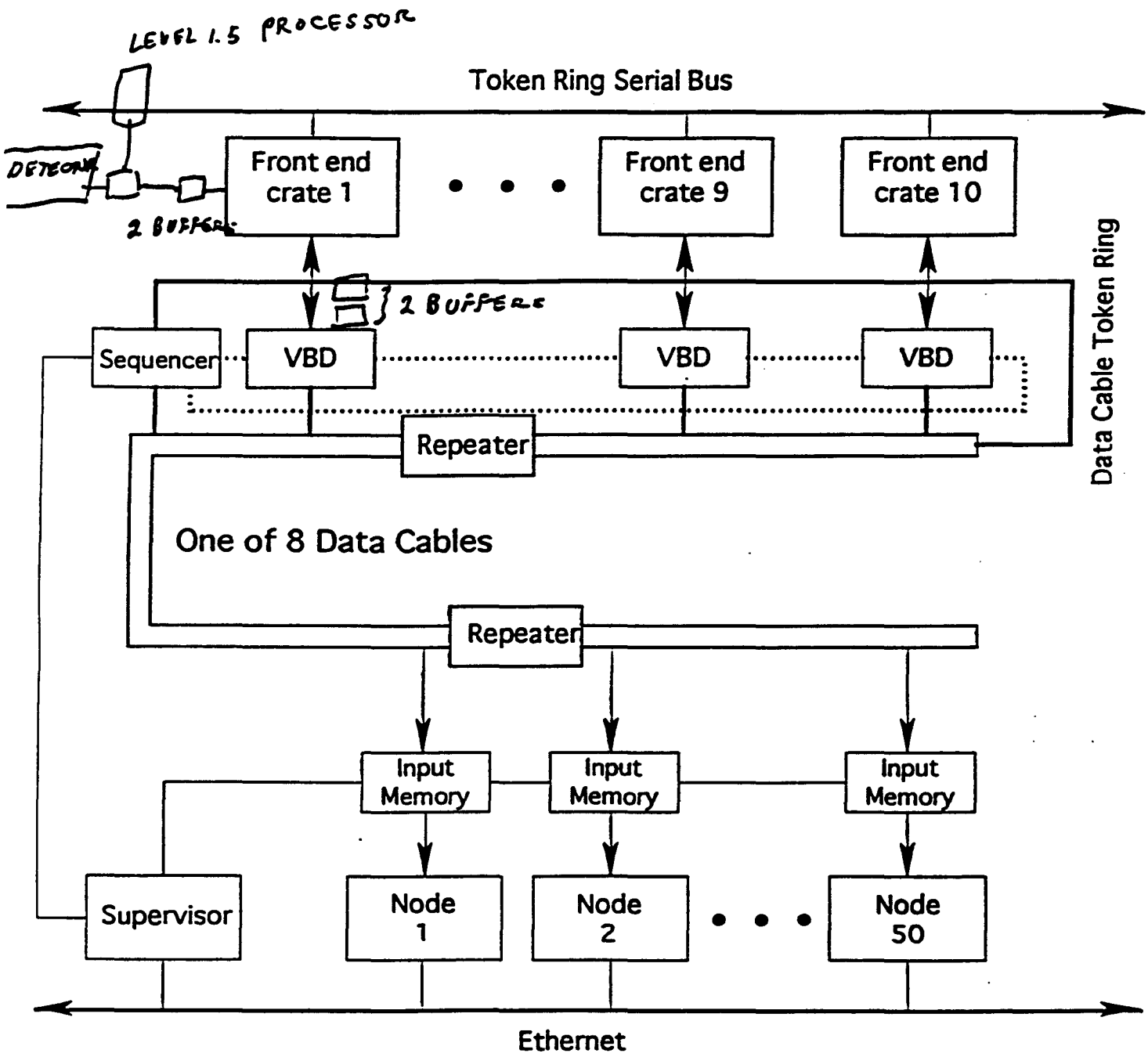
TWO TRIGGERS BEING DEVELOPED

μ SYSTEM. REJECTION ~10
AVERAGE TIME ~ 10μS.

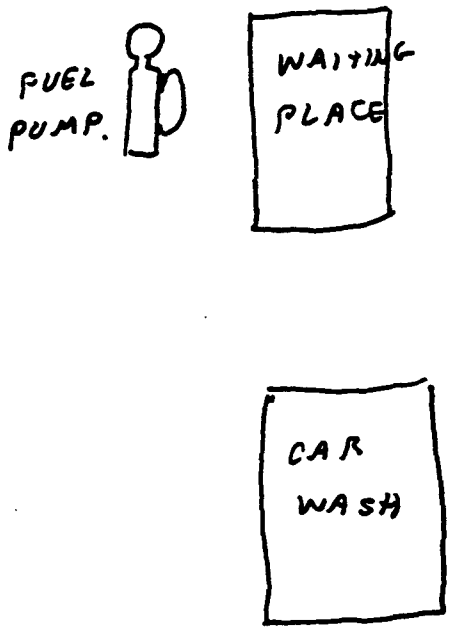
ELECTRON TRIGGER REJECTION ~ 2
TIME ~ 30μS.

FIGURE OF MERIT ≈ REJECTION/TIME.

⇒ μ IS MUCH BETTER.


LEVEL 1.5 PROCESSOR
Token Ring Serial Bus
DETECTOR
2 BUFFERS
Front end crate 1
Front end crate 9
Front end crate 10
2 BUFFERS
Sequencer
VBD
VBD
VBD
Repeater
Data Cable Token Ring
One of 8 Data Cables
Repeater
Input Memory
Input Memory
Input Memory
Supervisor
Node 1
Node 2
Node 50
Ethernet

CAR WASH ANALOGY



CAR WASH & GASOLINE SERVICE STATION.

NO LEVEL 1.5 CORRESPONDS TO PEOPLE WASHING CAR AND NOT BUYING GAS

LEVEL 1.5 REJECT CORRESPONDS TO PEOPLE BUYING GAS BUT NOT WASHING CAR

LEVEL 1.5 ACCEPT CORRESPONDS TO PEOPLE BUYING GAS & WASHING CAR.

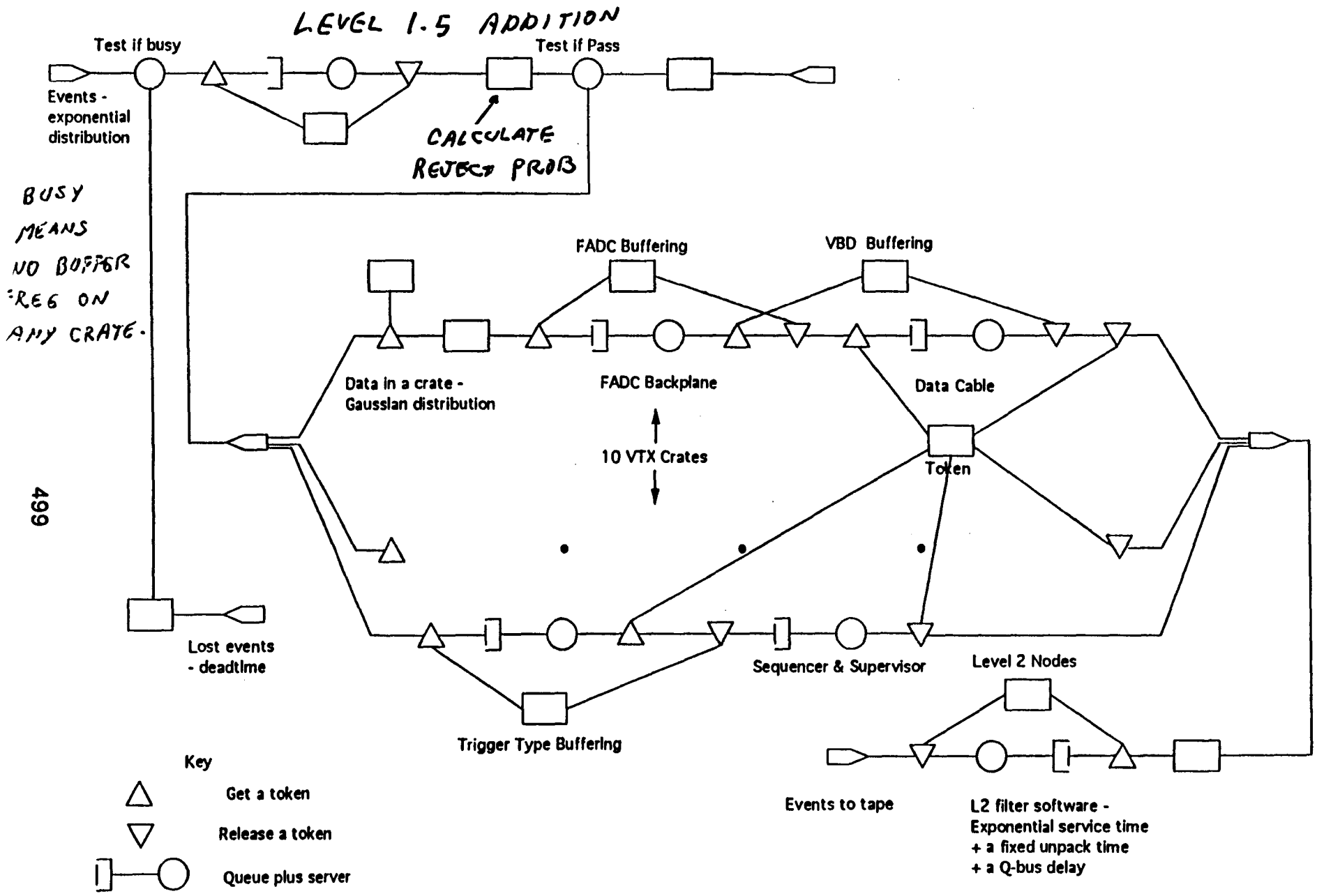
GAS PUMPING TIME CORRESPONDS TO LEVEL 1.5 PROCESSING TIME.

THE LEVEL 1.5 PROCESSING IS DEPENDENT ON # OF CANDIDATES SO IS ~ POISSON (EXPONENTIAL PROCESSING TIME).

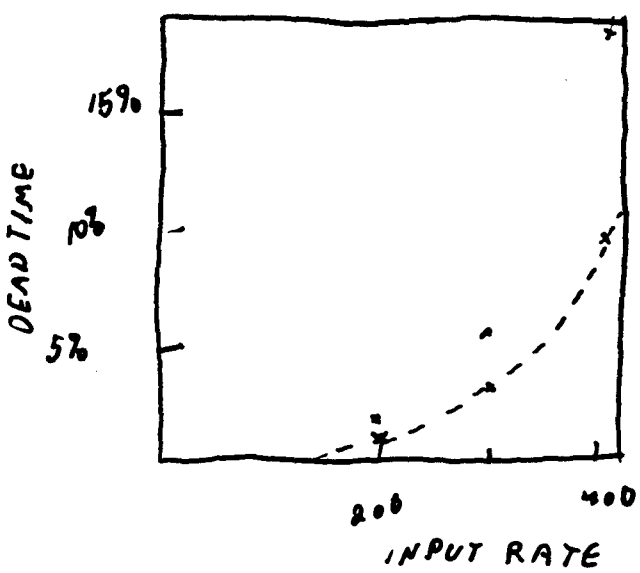
NOTE THAT CARS HAVE DIFFERENT SIZE FUEL TANKS SOO FUELING TIME IS ALSO ~ EXPONENTIAL SO THE ANALOGY IS VERY CLOSE.

NOTE THAT FOR THE CASE WHERE ALL EVENTS GO TO LEVEL 1.5 (ALL CARS BUY FUEL) THAT WE HAVE LOST DOUBLE BUFFERING ON THE FRONT END. IT IS JUST 3 QUEUES IN SERIES.

→ MUST KEEP LEVEL 1.5 TIME SMALL ←
MUCH LESS THAN BACKPLANE TRANSFER TIME.



RESQ Model For The Simulation Of The VTX Readout

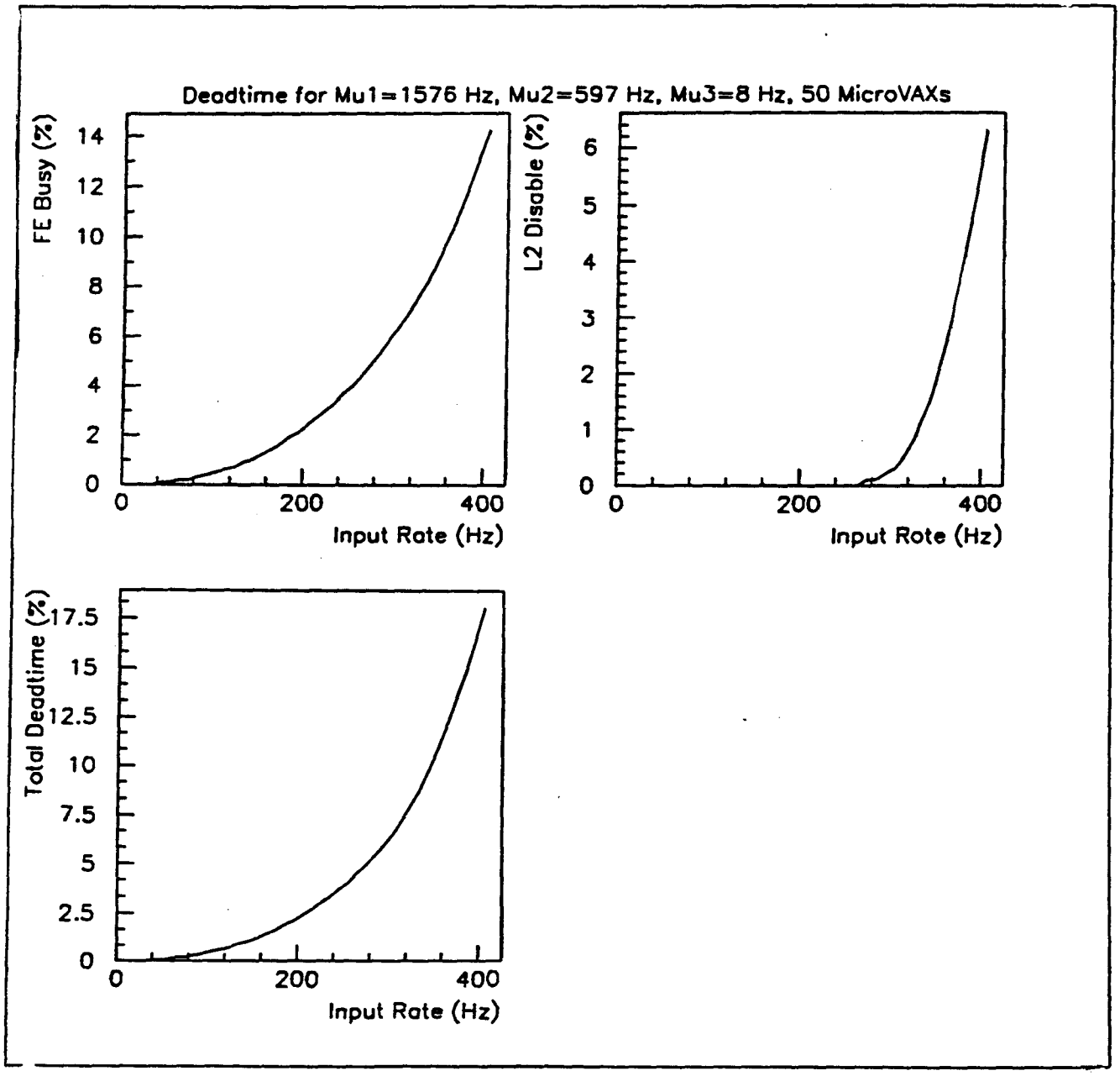


DEAD TIME LEVEL 1.5

FACTOR OF 2 REJECTION

10μS AVERAGE PROCESSING TIME

SAME AS RED BUT MULTIPLIED BY 2 TO SHOW AGREEMENT WITH ANALYTIC CALCULATION


Deadtime for Mu1=1576 Hz, Mu2=597 Hz, Mu3=8 Hz, 50 MicroVAXs
FE Busy (%)
14
12
10
8
6
4
2
0
0
200
400
Input Rate (Hz)
L2 Disable (%)
6
5
4
3
2
1
0
0
200
400
Input Rate (Hz)
Total Deadtime (%)
17.5
15
12.5
10
7.5
5
2.5
0
0
200
400
Input Rate (Hz)

1 MICROVAX CASE HAS 30 EQUATIONS. (NO 1.5)

SOLVED ON A MACINTOSH SPREAD SHEET.
(SPARSELY POPULATED)

50 MVA SYSTEM REQUIRES < 25x30 EQNS
(~ 570 EQNS)

SIMPLE ITERATIVE SOLVER.

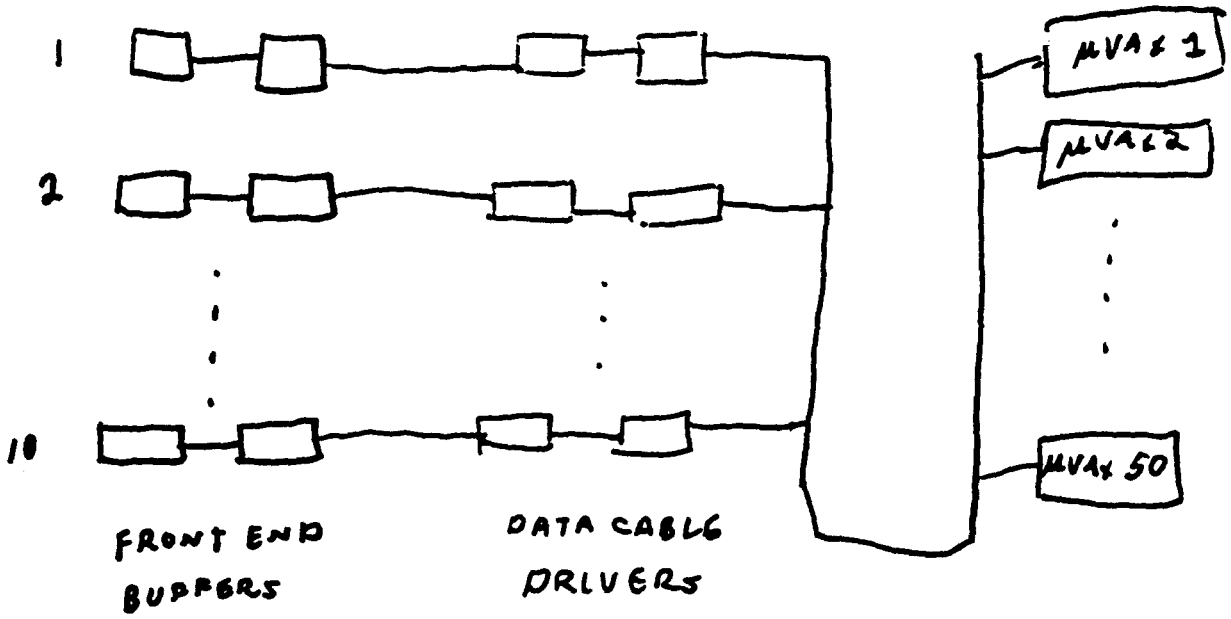
FEW HUNDRED LINES OF PASCAL.

~2 MINUTES TO SOLVE ON MACINTOSH II.
(1 SET OF VALUES)

1 MICROVAX CASE WITH LEVEL 1.5 HAS
38 EQUATIONS.

1 MICROVAX CASE WITH SOME LEVEL 1.5 AND
SOME NO LEVEL 1.5 HAS 57 EQUATIONS
~ FACTOR OF 2 MORE COMPLEX.

EXTRACTION OF ANALYTIC MODEL.

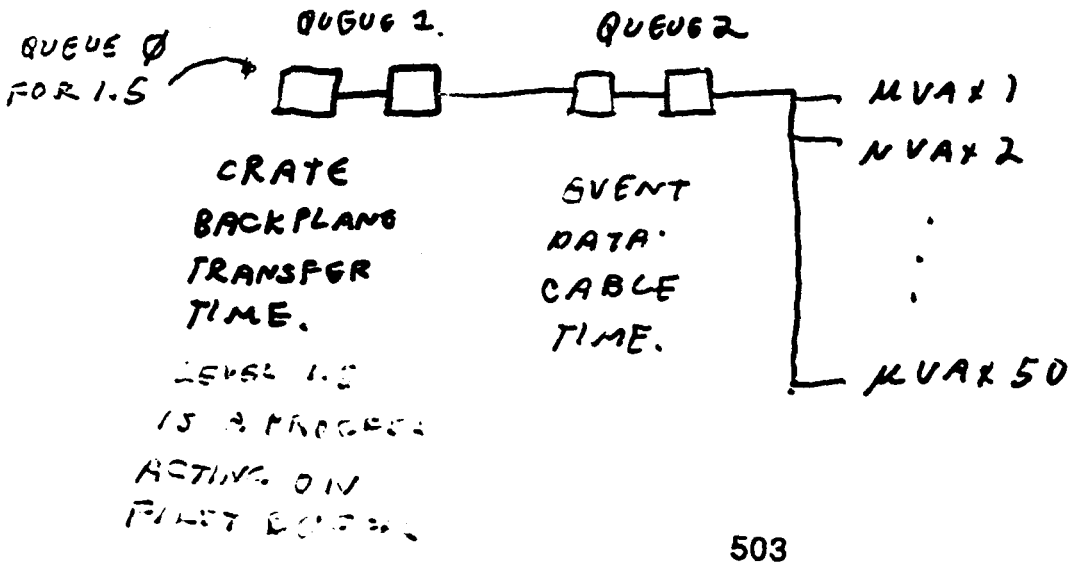


TOO COMPLEX TO CALCULATE ANALYTICALLY.

HOWEVER

1. DATA IS TRANSFERED IN PARALLEL TO DATA CABLE BUFFERS

2. DATA IS TRANSFERRED ACROSS DATA CABLE IN ORDER THAT IT BECOMES AVAILABLE ⇒ TRANSFER NEARLY CONTINUOUS



ANALYTIC VS COMPUTER (RESQ) MODEL.

RESQ

PLUS

- FAIRLY EASY TO DEVELOP MODELS
- EASY TO ADD SMALL FEATURES

MINUS

EASY TO MAKE ERRORS

NOT EASY TO UNDERSTAND STATISTICAL ERRORS

FAIRLY LONG COMPUTATIONAL RUNS ON AMDAHL.

ANALYTIC.

PLUS

- OVER CONSTRAINED SOLUTION SO DIFFICULT TO GET A CONSISTENT YET INCORRECT SOLUTION.
- STATISTICAL ERRORS ARE WELL DEFINED
- GET OCCUPANCY OF ALL STATES
- STANDARD ALGEBRA SO MANY PEOPLE CAN CHECK.
- SENSITIVITY TO DEAD TIME (OR OTHER PARAMETERS) IS EASILY AVAILABLE THROUGH CALCULUS. I.E. THE DERIVATIVE.

MINUS

- MOST MODELS NEED CAREFUL SIMPLIFICATION BEFORE THEY CAN BE MODELED
- SOME SYSTEMS CANNOT BE SOLVED (LOOPS ARE POSSIBLE)
- REQUIRES CONSIDERABLE ANALYTIC SKILLS (EXPERT)
- CANNOT STUDY "WARM UP"
- RESTRICTED TO POISSON (EXPONENTIAL) DISTRIBUTIONS.
- LARGE SYSTEMS REQUIRE MACHINE GENERATION OF EQUATIONS.

DØ EXPERIENCE WITH DAQ MODELS.

STARTED WITH RESQ IN EARLY 1986 TO ANSWER 2 QUESTIONS.

1. NUMBER OF FRONT END BUFFERS
2. SPEED OF VME BACKPLANE. "DO WE NEED 20 MB/S?"

} 3 MAN MONTHS

MODELED ONE CRATE + THEN ONE DATA CABLE.
DEVELOPED ANALYTIC MODEL TO CHECK RESQ MODEL

IN 1990 J. WIGHTMAN DEVELOPED A SERIES OF TEST PROGRAMS TO CHECK DATA ACQUISITION SYSTEM FROM FRONT END TO LEVEL II.

REALIZED THAT THIS COULD BE USED TO TEST DAQ SYSTEM MODEL.

1991 BROUGHT MODEL UP TO DATE WITH AS-BUILT HARDWARE. EXTENDED ANALYTICAL MODEL TO ENTIRE DATA CABLE 3-4 MAN MONTHS

1992- DEVELOPING MODEL FOR LEVEL 1.5 TRIGGER
RESQ MODEL RUNNING. ANALYTIC MODEL WRITTEN DOWN BUT NOT YET SOLVED 1-MAN MONTH
ANTICIPATE LOT'S OF USE FOR DØ UPGRADE.

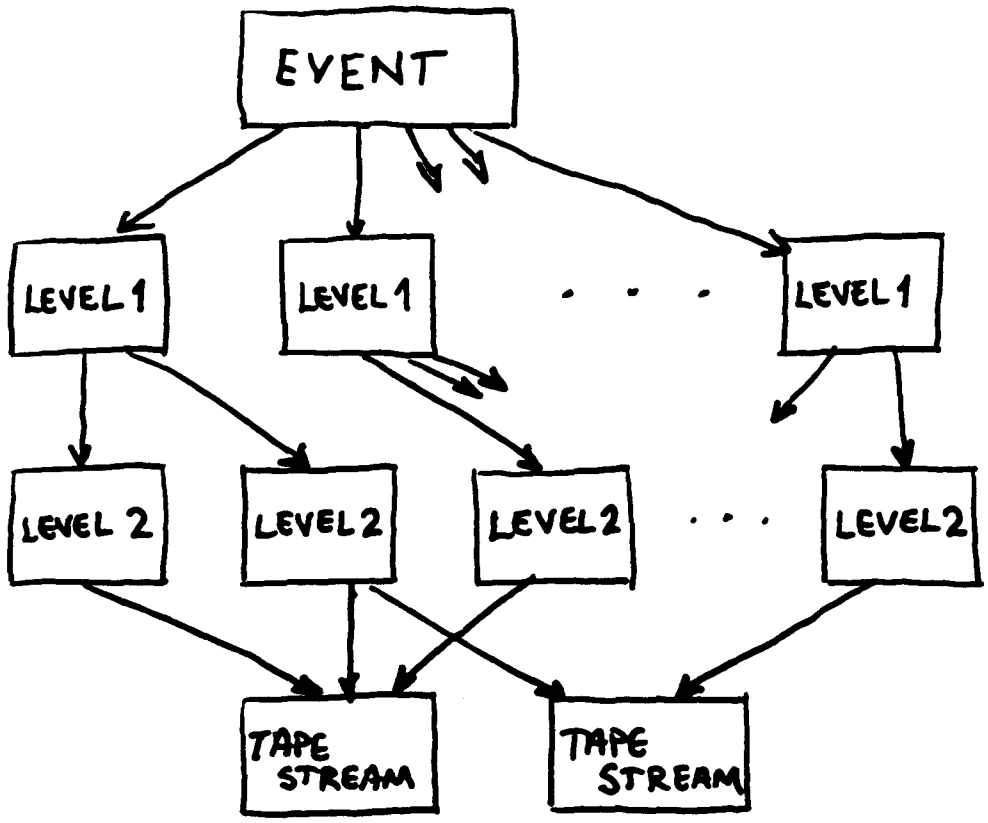
STRATEGIES OPTIMIZING DATA LOAD IN THE DZERO TRIGGERS

M. FORTNER
NORTHERN ILLINOIS UNIVERSITY

24 - APRIL - 1992

* OR, WHAT DO YOU DO WITH ALL THE SIMULATION OUTPUT?

LOGICAL STRUCTURE OF THE DØ TRIGGER



LEVEL 1 IS THE HARDWARE TRIGGER INCLUDING BOTH SYNCHRONOUS AND ASYNCHRONOUS TERMS.

LEVEL 2 IS THE SOFTWARE TRIGGER IN A FARM OF PROCESSORS.

CONSTRAINTS ON THE DØ TRIGGER

PARAMETERS	
LUMINOSITY	PEAK EXPECTED FROM THE ACCELERATOR - $10^{31}\ cm^{-2} sec^{-1}$
DEAD TIME	FROM TRIGGER SIMULATION - A FEW % FOR SOME
# OF TRIGGERS	HARDWARE LIMITATION - 32 WITH "AND" ONLY
CROSS SECTION	FROM MONTE CARLO WITH BACKGROUND EVENTS
1→2 BANDWIDTH	FROM DAQ SIMULATION - 200 Hz
# OF TRIGGERS	SOFTWARE LIMITATION - 128 FILTER SCRIPTS
CROSS SECTION	FROM MONTE CARLO WITH BACKGROUND EVENTS
2→TAPE BANDWIDTH	FROM DAQ SIMULATION - 2 Hz

THE VERTICAL PROBLEM

EXAMPLE 1 : LEVEL 1 ↔ LEVEL 2

INCLUSIVE JET PHYSICS

STUDY 3 JET / 2 JET
DIJET MASS
CONE SIZE
$d^2\sigma/dp_t dy$

REQUIRES LARGE STATISTICS $\sim 3 \times 10^5$ /BIN

CONTROLS AT LEVEL 1:

JET TOWER ENERGY THRESHOLD

JET TOWER MULTIPLICITY

TRIGGER PRESCALE

VERTEX CONSTRAINT (A SMART PRESCALE)

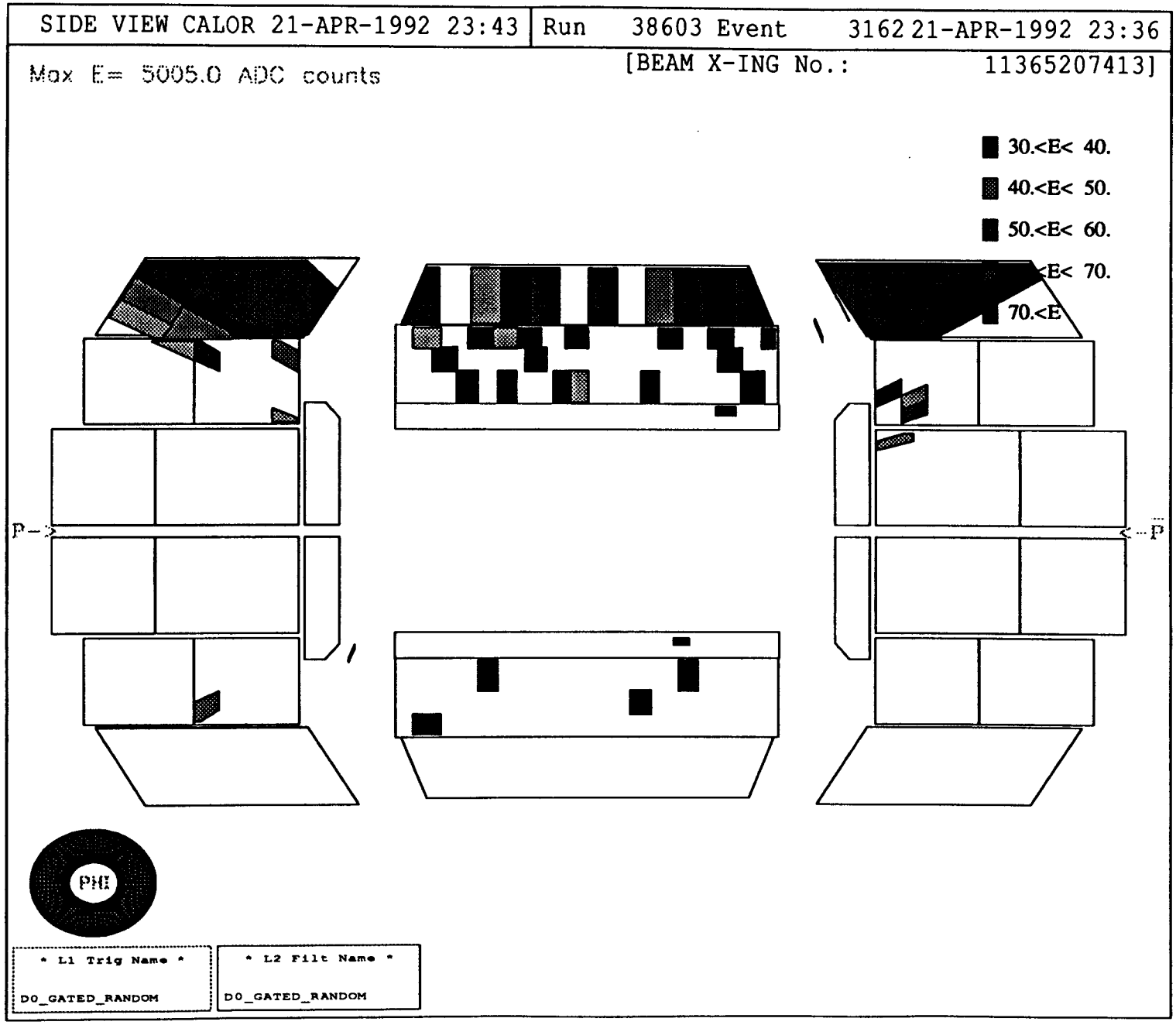
CONTROLS AT LEVEL 2:

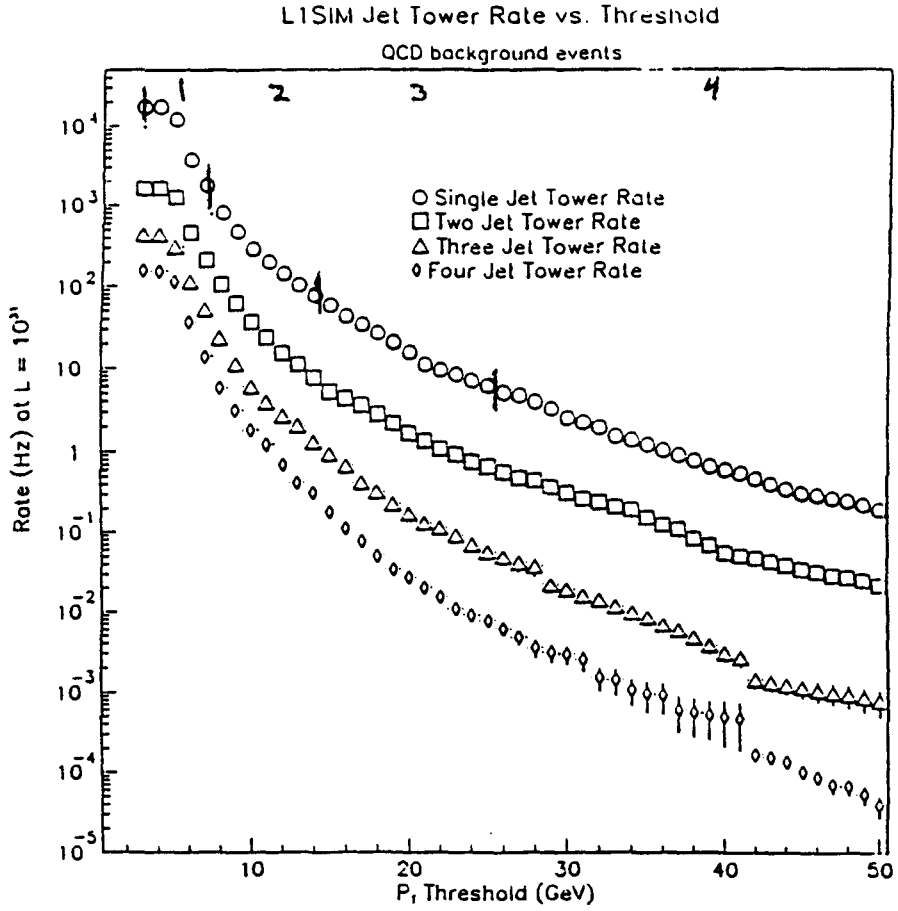
JET ENERGY THRESHOLD

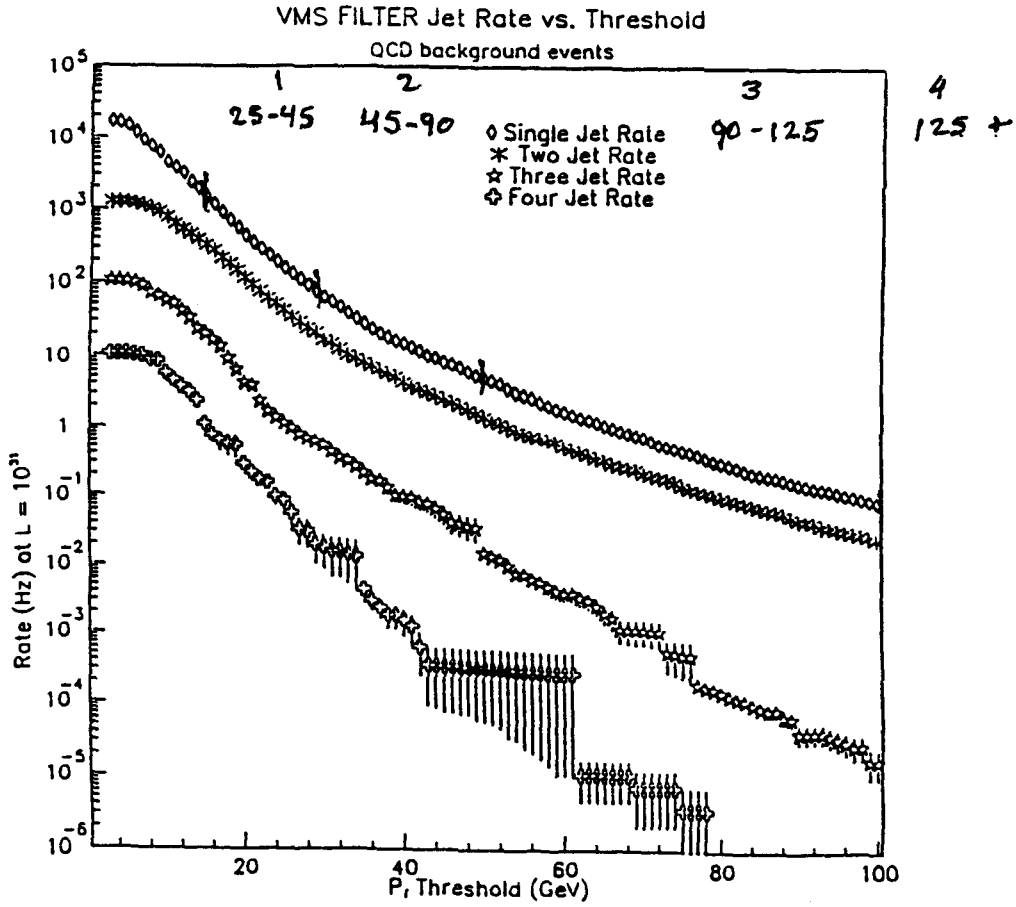
JET CLUSTERING

SOLUTION REQUIRES MULTIPLE BINS EACH OPTIMIZED TO GIVE HIGH EFFICIENCY BETWEEN LEVEL 1 AND LEVEL 2.

MAIN RING BACKGROUND




L1SIM Jet Tower Rate vs. Threshold
QCD background events
1
2
3
4
O Single Jet Tower Rate
□ Two Jet Tower Rate
△ Three Jet Tower Rate
◊ Four Jet Tower Rate
Rate (Hz) at L = 10^31
P_t Threshold (GeV)


VMS FILTER Jet Rate vs. Threshold
QCD background events
1
2
3
4
25-45
45-90
90-125
125 +
◊ Single Jet Rate
✳ Two Jet Rate
☆ Three Jet Rate
✧ Four Jet Rate
Rate (Hz) at L = 10^31
P_t Threshold (GeV)

THE VERTICAL PROBLEM

EXAMPLE 2: LEVEL 1 ↔ LEVEL 1.5 ↔ LEVEL 2

$W \to \mu\nu$

REQUIRES NO PRESCALE, HIGH EFFICIENCY

CONTROLS AT LEVEL 1:

MUON RAPIDITY CUT

CONTROLS AT LEVEL 1.5

MUON MOMENTUM CUT (COARSE)

CONTROLS AT LEVEL 2

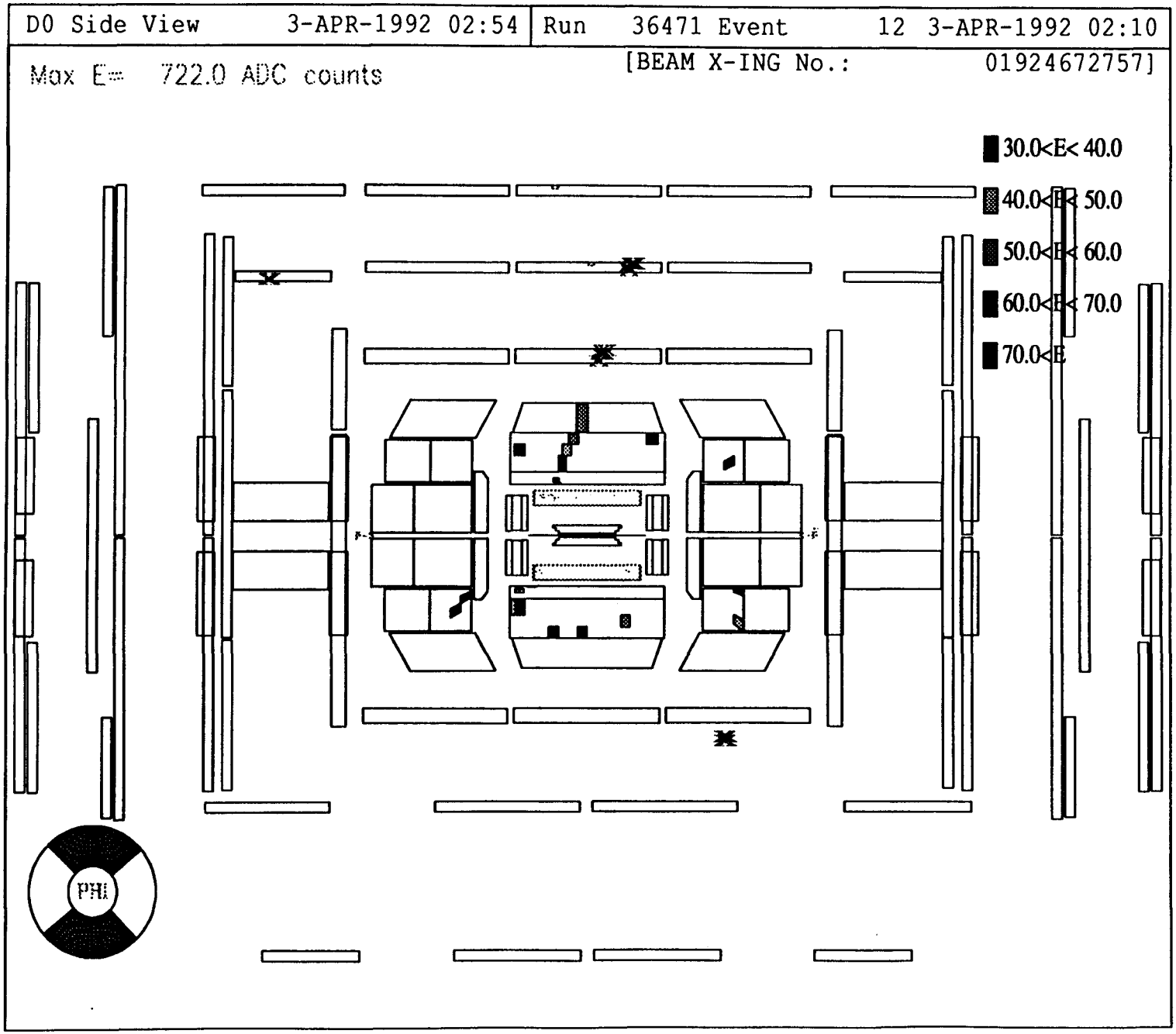
MUON MOMENTUM CUT

WITH FULL MUON COVERAGE BACKGROUND IS DOMINATED BY π/K DECAYS AND BEAM SCRAPING ~ 5-30 KHz.

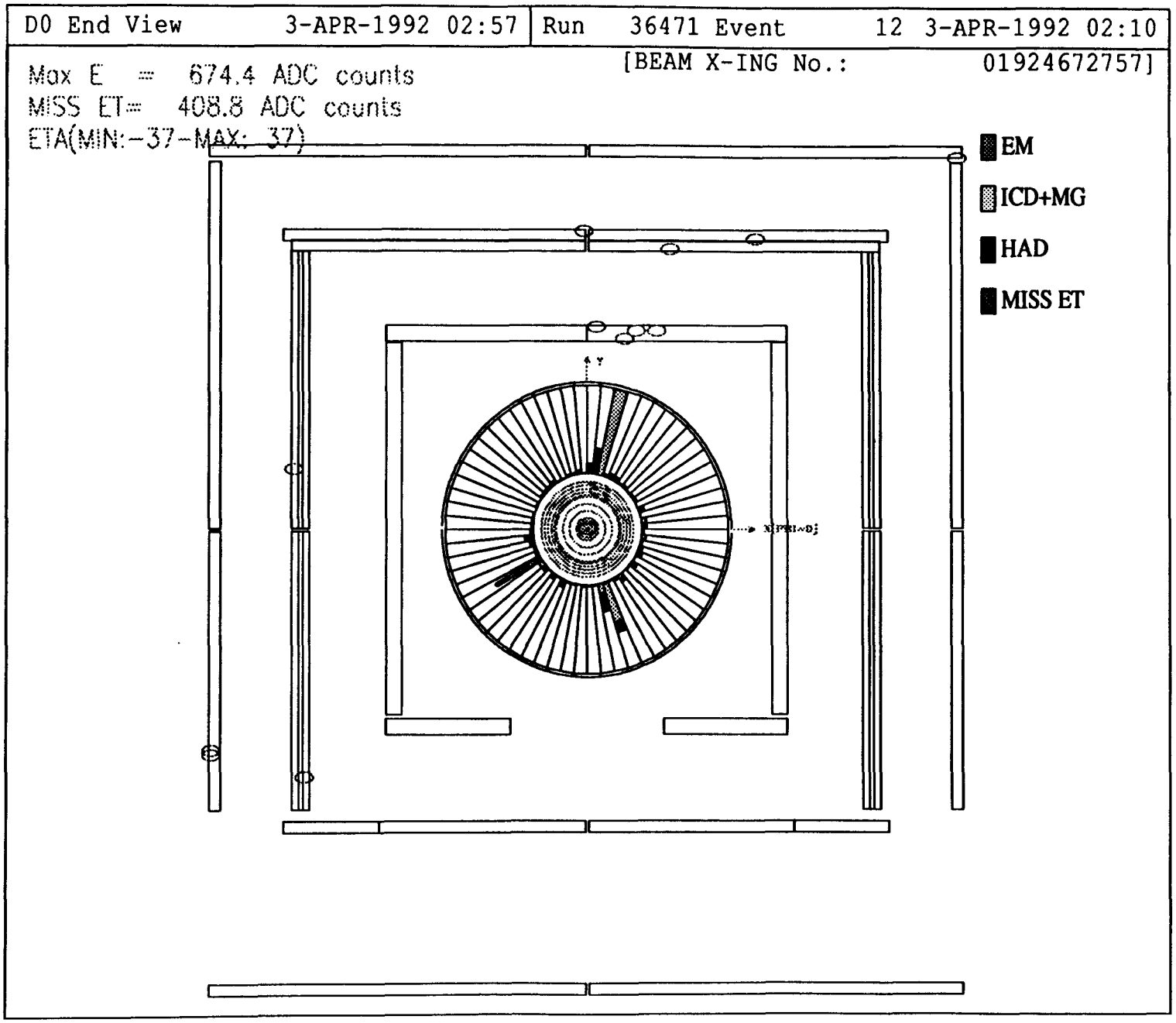
IN CENTRAL REGION BACKGROUND IS DOMINATED BY COSMIC RAYS ~ 200 Hz @ LEVEL 1.

SOLUTION REQUIRES HARD CUTS AT LEVEL 1 AND LEVEL 1.5 TO REDUCE RATE BELOW BANDWIDTH CONSTRAINTS.

COSMIC RAY BACKGROUND



COSMIC RAY BACKGROUND



THE HORIZONTAL PROBLEM

EXAMPLE 3: SPLITTING BANDWIDTH FROM LEVEL 1 TO LEVEL 2

REQUESTS ARE GENERATED BY PHYSICS ANALYSIS NEEDS.

PHYSICS GROUPS:
- QCD
- DIRECT γ
- "NEW"/SUSY
- W/Z
- TOP
- BOTTOM

EACH HAVE UNIQUE SETS OF TRIGGERS, AND WOULD LIKE ALL THE RATE POSSIBLE.

USEFUL CONTROLS:

BIT SHARING - MULTIPLE PHYSICS FROM ONE THRESHOLD

MULTIELEMENT TRIGGERS
- SINGLE ELEMENT TRIGGERS USUALLY TOO LOOSE FOR BANDWIDTH CONSTRAINTS

THE FULL LIST OF PHYSICS IS TOO LARGE FOR THE TRIGGER GROUP TO HANDLE ALL SIMULATION. FEEDBACK AND ITERATION ARE REQUIRED BETWEEN PHYSICS ANALYSIS AND TRIGGER IMPLEMENTATION.

The Level 1 & 2 Trigger Menu (version 4.1)
1/25/92

The first list is the tentative level 1 (and associated level 1.5) trigger list for D0. Triggers are numbered from 0 to 31. Triggers using level 1.5 are indicated with a second line to distinguish level 1 dead time. A luminosity of up to 1 x 10^31 and a maximum rate of 200 Hz to level 2 form constraints to the settings.

The second list indicates the level 2 filters associated with the level 1 triggers. There are 128 possible level 2 triggers each associated with one level 1 trigger. The luminosity of 1 x 10^31 was also used here, along with a maximum rate of 2 Hz to the tape streams and 0.2 Hz to the express line.

For Level 1:

```
SC(e)      :  Total scalar transverse energy in calorimeter greater than e GeV.
MS(e)      :  Missing transverse energy in calorimeter greater then e GeV.
JT(n,e)    :  n or more hadronic plus EM trigger towers with Et greater than
              e GeV; the reference sets used are 3,5,9,20 GeV.
EM(n,e/h)  :  n or more electromagnetic towers greater then e GeV, with an
              hadronic veto at h; the reference sets used are 2,6,12 GeV
              with no hadronic veto.
MU(n,g)    :  n or more muons confined to a section eta<g (eta<1, eta<2, eta<3).
L0(g)      :  Level 0 interaction measure, g=0 for possible interaction,
              g=1 for single good interaction, g=2 for one or more interactions
```

For Level 1.5:

```
MX(n,g,e)  :  n or more muons with pt greater than e GeV in a section eta<g;
              reference sets used are 3,5,7 GeV/c, where 3 is a preset base value
```

For Level 2: (and assumed rejections with no raising of threshold)

```
L2SC(e)     :  Total scalar transverse energy in calorimeter greater than e GeV.
               rej guessed at 2??????
L2MS(e)     :  Missing transverse energy in calorimeter greater then e GeV.
               rejection guessed at 10 (worse for higher threshold)
L2JT(n,e)   :  n or more jets (usually in a .7 X .7 square) with Et greater than
               e GeV   rejection 10 if 11 chosen for 100% effic of offline
               threshold mentioned, and 12 chosen also for 100%
L2EM(n,e/c)
            :  n or more electromagnetic clusters greater than e GeV, passing
               shape cuts c and track requirements c :
               c = ELE    electron long and transverse shape cuts, require track
                   GAM    shape cuts tuned for photons, no track requirement
                   GIS    GAM + isolation in some cone
                   ESC    no shape cuts or track requirement
               rejection 40 for ELE
               rejection 10 for GAM (a GUESS????????)
               rejection 50 for GIS     "
               rejection  1 for ESC a little pessimistic

L2MU(n,g,e):  n or more muons confined to a section eta<g (eta<1, eta<2,
               eta<3), with pt greater than e GeV
               rejection 20 (a GUESS?????, pessimistic
```

L1 Bit	Name	Terms	Cross Section (ub)	Pre-scale	Rate (Hz)	Physics Interest
0	min_bias	L0(0)	5.0E+4	(3E+7)	0.01	min bias
1	scalar_et	SC(150)	8.0E-2		0.8	zoo
2	missing_et	MS(23)	1.4E+0		14.0	gluino
3						
4						
5	jet_low	JT(1,3) & L0(1)	1.6E+3	(2E+4)	0.4	QCD
6	jet_medium	JT(1,7) & L0(1)	8.0E+1	(1E+3)	0.4	QCD
7	jet_high	JT(1,14) & L0(1)	4.0E+0	(5E+1)	0.4	QCD
8	jet_max	JT(1,25)	2.0E-1		2.0	QCD
9	jet_calib	JT(1,7) & SC(100)	8.0E-1	(2E+1)	0.4	Calibration
10						
11	four_jets	JT(4,7)	1.2E+0		12.0	t -> jets
12						
13						
14	em_low	EM(1,2)	2.0E+3	(4E+4)	0.5	gam
15	em_medium	EM(1,6)	8.0E+1	(4E+2)	2.0	e,gam
16	em_high	EM(1,12)	4.0E+0		40.0	W,t -> e; gam
17	two_em_low	EM(2,2)	2.0E+2	(2E+3)	1.0	psi, gam
18	two_em_med	EM(2,6)	3.8E+0		38.0	Z, ups, gam
19						
20						
21	mu_two_em	MU(1,3) & EM(2,2)	1.0E+0		10.0	b -> mu psi
22	mu_low	MU(1,3) & L0(1)	5.0E+3	(2E+2)		
		MX(1,3,3)	5.0E+1	(2E+2)	1.0	b -> mu
23	two_mu_low	MU(2,3)	5.0E+2			
		MX(1,3,3)	1.0E+0		10.0	b,Z,t -> mu mu
24	mu_and_em	MU(1,3) & EM(1,6)	<8.0E+1			
		MX(1,3,5)	1.0E-1		1.0	t -> mu e
25	mu_and_jets	MU(1,3) & JT(2,7)	<1.0E+0		10.0	t -> mu jet
26	mu_high_cent	MU(1,2)	2.0E+2			
		MX(1,2,7)	1.0E+0		10.0	W,Z,t -> mu
27	two_mu_cent	MU(2,2)	2.0E+0		20.0	Z,t -> mu mu
28	mu_cent	MU(1,2)	2.0E+2	(4E+3)	0.5	Calibration
29						
30						
31	none					Level 1 Monito
	Total				184.5	

L1 bit	L2 bit	name	L2 terms	Cross Section (ub)	L1 Pre-scale	L2 Rate (Hz)	Physics Interest (efficiency)
0	0	min_bias	none	5.0E+4	(3E+7)	0.01	min bias
	1	good_lvl0	L2L0				
1	10	scalar_et	L2SC(300)	6.0E-3		0.06	zoo
2	11	missing_et	L2MS(30)	1.0E-2		0.10	gluino
	101	miss_et_x	L2MS(50)	1.0E-3		0.010X	susy
3							
4							
5	20	jet_low	L2JT(1,15)	1.6E+2	(2E+4)	0.04	QCD
6	21	jet_medium	L2JT(1,30)	8.0E+0	(1E+3)	0.04	QCD
7	22	jet_high	L2JT(1,50)	6.0E-1	(5E+1)	0.06	QCD
8	23	jet_max	L2JT(1,110)	8.0E-3		0.08	QCD
9	24	jet_calib	L2JT(1,7) &L2SC(100)	1.0E+0	(2E+1)	0.05	Calibration
10							
11	60	top_jets	L2JT(4,20)	2.0E-2		0.20	t -> jets
12							
13							
14	30	em_low_esc	L2EM(1,6,ESC)	8.0E+1	(4E+4)	0.02	em background
	33	gam_low	L2EM(1,3,GAM)	4.0E+1	(4E+4)	0.01	gamma
15	31	em_med_esc	L2EM(1,20,ESC)	8.0E-2	(4E+2)	0.02	em background
	34	gam_medium	L2EM(1,10,GAM)	6.0E-1	(4E+2)	0.02	gamma
	37	gam_med_iso	L2EM(1,6,GIS)	8.0E-1	(4E+2)	0.02	gamma
	51	ele_medium	L2EM(1,8,ELE)	2.0E+0	(4E+2)	0.05	electron
16	32	em_high_esc	L2EM(1,50,ESC)	6.0E-3		0.06	em background
	35	gam_high	L2EM(1,40,GAM)	2.0E-3		0.02	gamma
	38	gam_high_iso	L2EM(1,25,GIS)	8.0E-3		0.08	gamma
	52	ele_high	L2EM(1,20,ELE)	4.0E-2		0.40	electron
	111	ele_high_x	L2EM(1,35,ELE)	4.0E-3		0.040X	W,t -> e
17	40	two_em_esc	L2EM(2,6,ESC)	3.8E+0	(2E+3)	0.02	2 gammas
	80	two_ele_low	L2EM(2,2,ELE)	2.0E+0	(2E+3)	0.01	psi,ups
18	43	two_gam_med	L2EM(2,8,GAM)	5.0E-3		0.05	2 gammas
	81	two_ele_med	L2EM(2,8,ELE)	8.0E-3		0.08	ups,susy
	112	two_ele_x	L2EM(2,10,ELE)	2.0E-3		0.020X	Z --> ee
19							
20							
21	121	mu_psi_e	L2EM(2,2,ELE) &L2MU(1,3,0)	6.0E-4		0.006X	b -> mu psi
22	90	mu_low	L2MU(1,3,3)	2.5E+0	(2E+2)	0.05	b -> mu
23	122	mu_psi_mu	L2MU(3,3,0)	1.0E-3		0.010X	b -> mu psi
	123	two_mu_low_x	L2MU(2,3,0) &L2MU(1,3,3)	3.0E-3		0.030X	b,Z,t ->mu mu
24	113	top_mu_e_x	L2EM(1,6,ELE) &L2MU(1,3,5)	1.0E-4		0.001X	t -> e mu
25	61	top_to_mu_j	L2MU(1,3,0) &L2JT(2,25)	1.0E-2		0.10	t -> mu jet
26	60	mu_high	L2MU(1,2,7)	6.0E-3		0.06	W,Z,t -> mu
	114	mu_high_x	L2MU(1,3,10)	2.0E-3		0.020X	W,Z,t -> mu
27	62	two_mu_cen	L2MU(2,2,0)	5.0E-3		0.05	Z,t -> mu mu
	115	two_mu_cen_x	L2MU(2,2,5)	2.0E-3		0.020X	Z,t -> mu mu
28	91	mu_central	L2MU(1,2,0)	2.0E+1	(4E+3)	0.05	Mu Calibration
29							
30							
31	none						Level 1 Monito
	Total					2.03	
	Express					0.177X	

Workshop on DAQ and Trigger System Simulation for HEP

SSCL, 23-25 Apr. 92

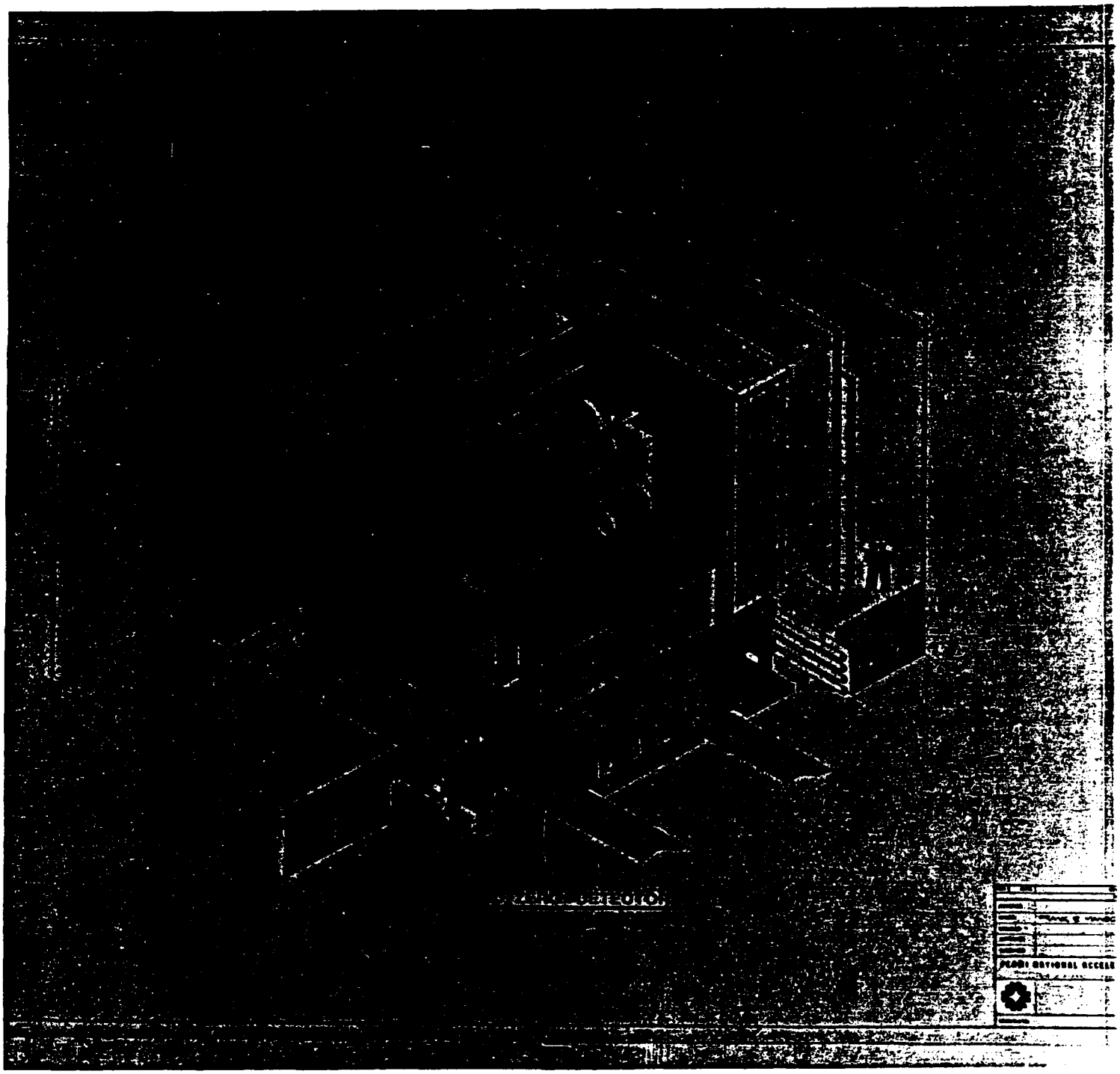
Simulation of the DØ/Level-2 Data Acquisition System

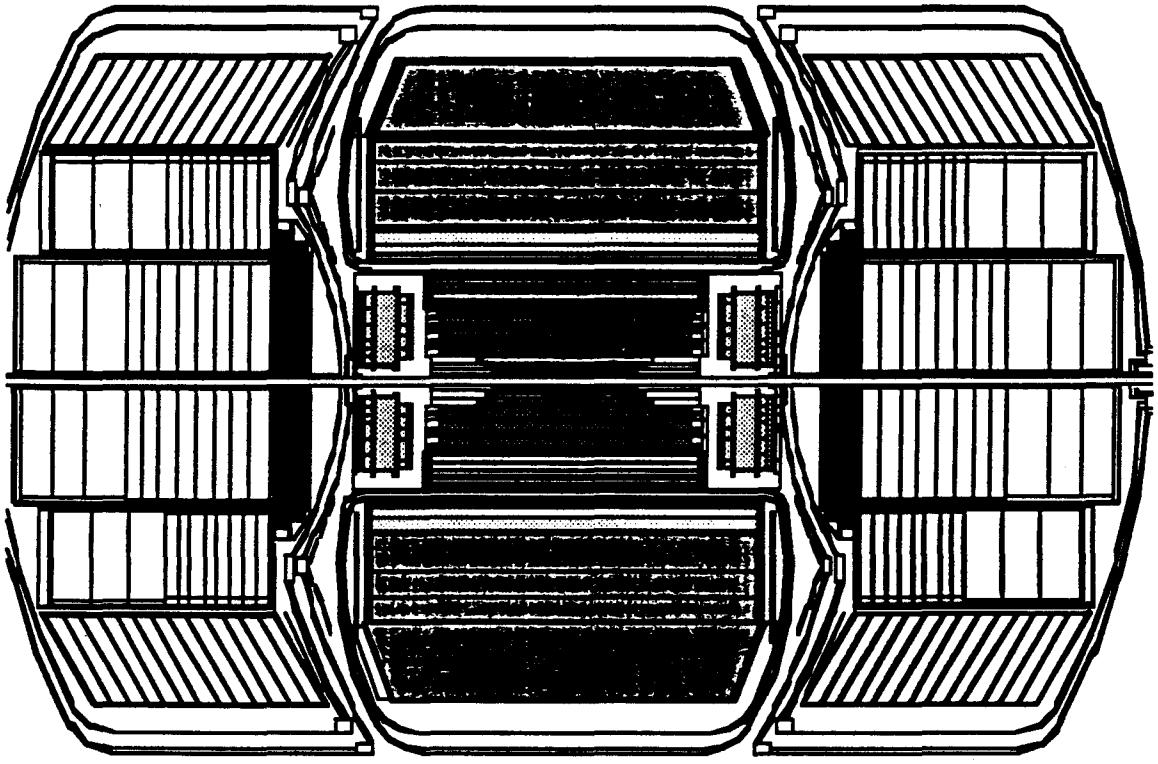
D. Cutts, J. Hoftun, H. Hattson, D. Nešić
Brown University

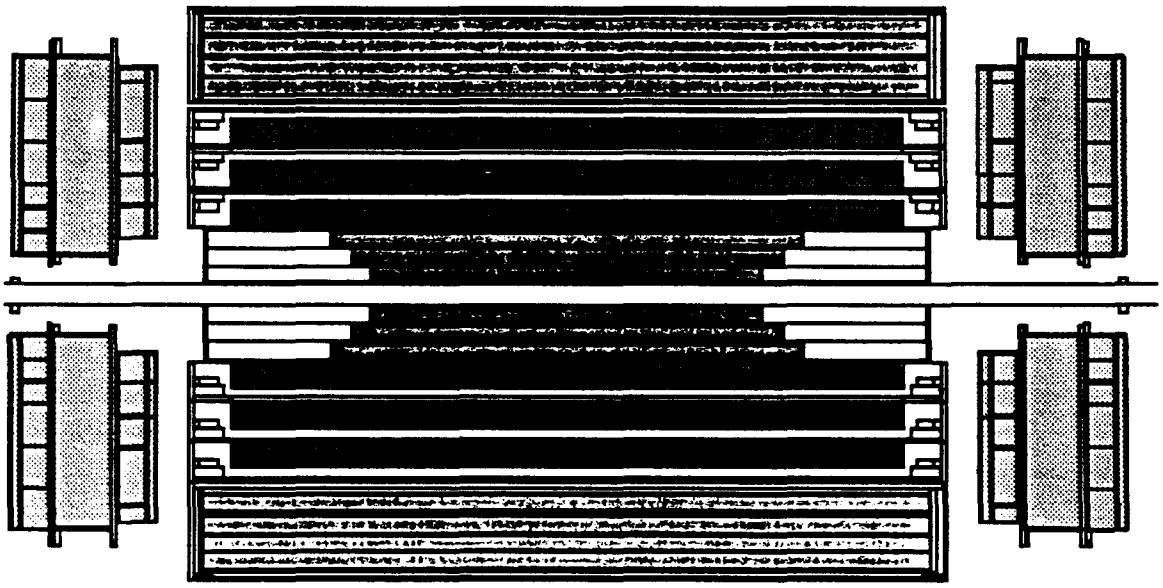
- Pictorial overview of the DØ/Level-2 DAQ
- Simulation goals
- Simulation Model
- Comparison with the 15 node installation/commissioning system
- Future plans

Credits: R. Partridge
T. Fahland

86-474CN







DØ / Level-2

Provides high level software trigger for DØ detector

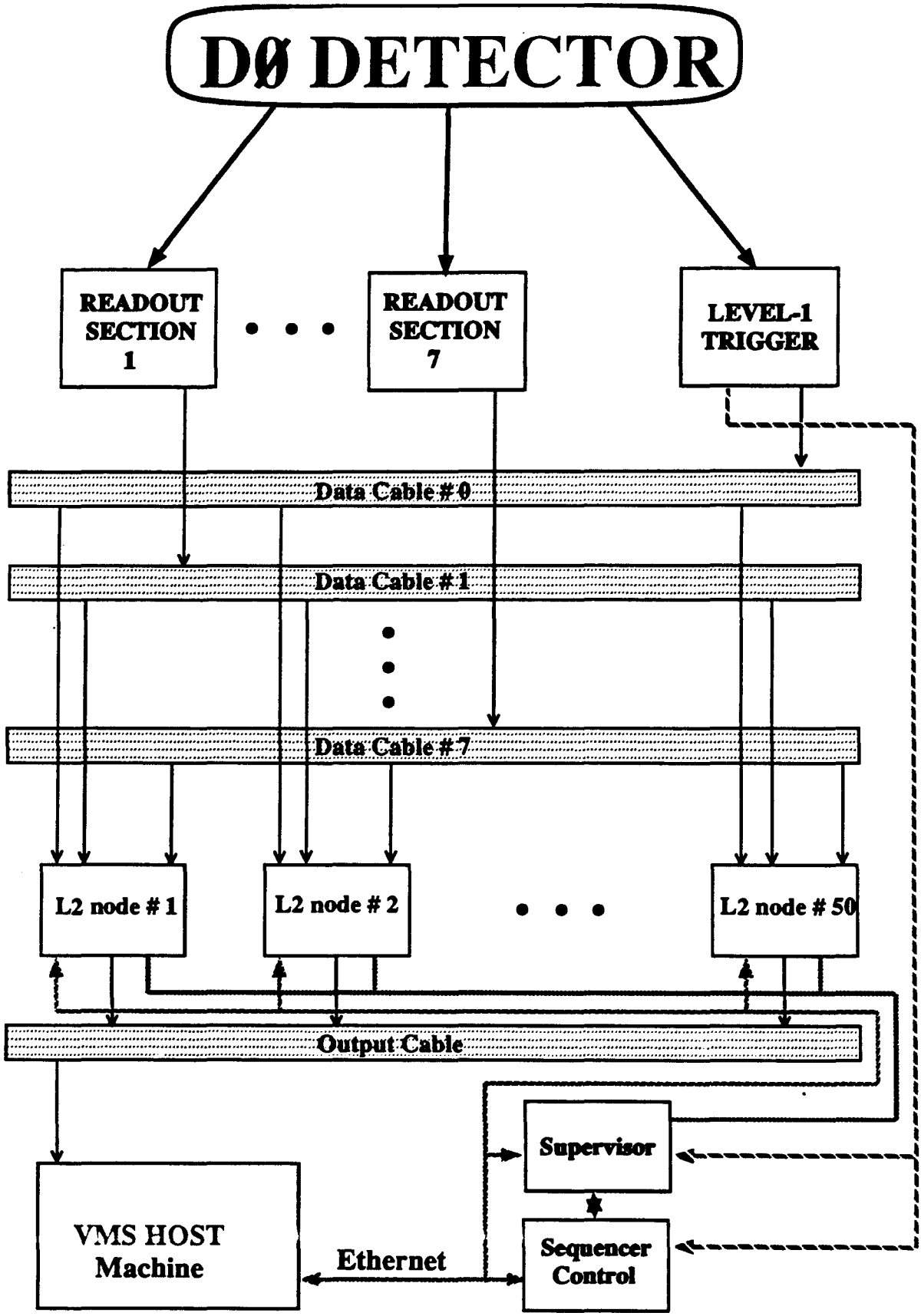
Hardware

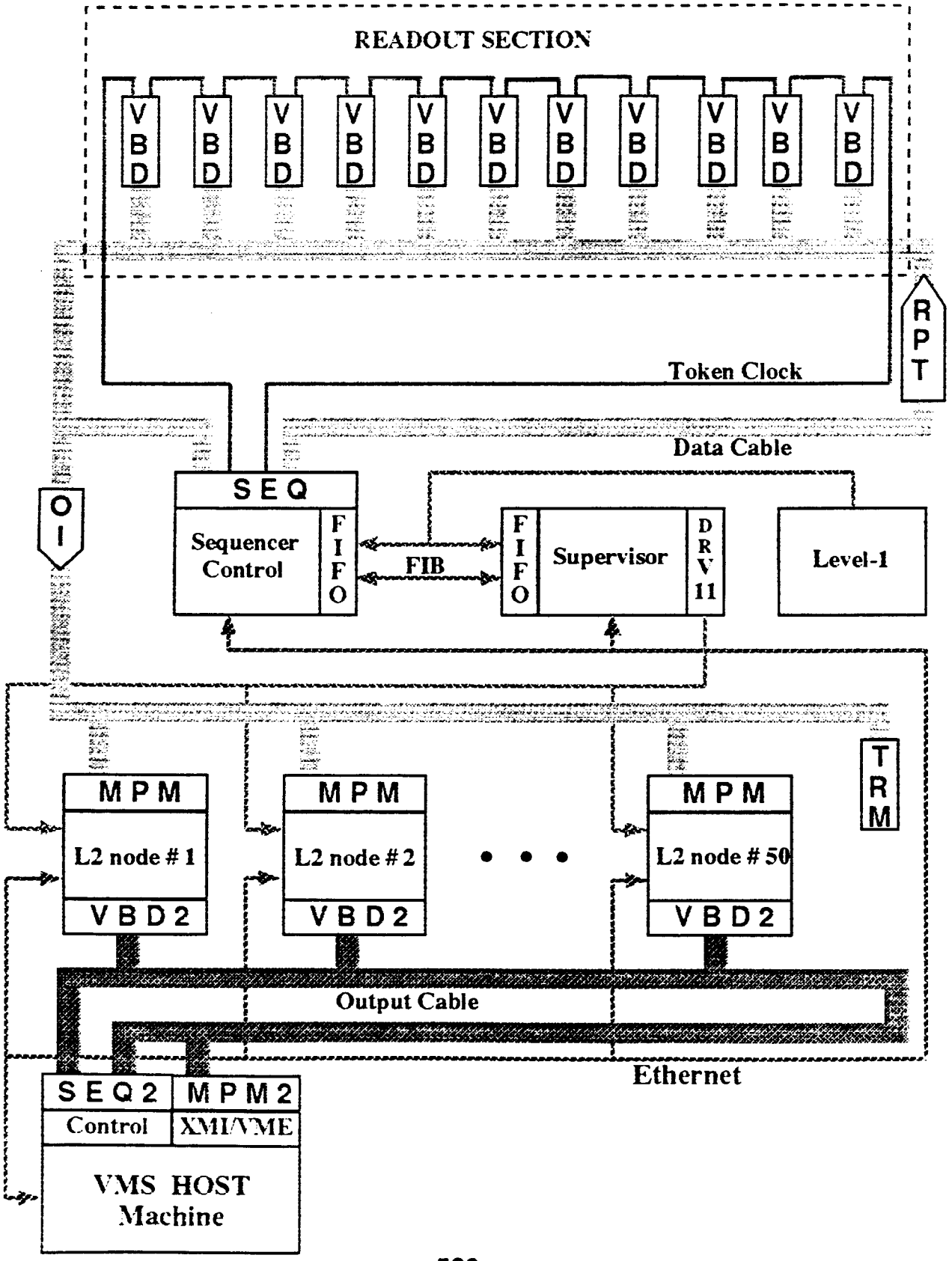
- VBD (VME Buffer / Driver)
- High speed data paths
- Sequencers
- Supervisor
- Level-2 nodes
- MPM (Multiport memories)

Software

- ELN operating system (real time)
- Filter framework - rigid
- Filtering tools - user defined, flexible

Note: No need for copying data
There is no real event building


DØ DETECTOR
READOUT SECTION 1
READOUT SECTION 7
LEVEL-1 TRIGGER
Data Cable # 0
Data Cable # 1
Data Cable # 7
L2 node # 1
L2 node # 2
L2 node # 50
Output Cable
Supervisor
VMS HOST Machine
Ethernet
Sequencer Control


READOUT SECTION
VBD
VBD
VBD
VBD
VBD
VBD
VBD
VBD
VBD
VBD
VBD
RPT
Token Clock
Data Cable
OI
SEQ
Sequencer Control
FIFO
FIB
FIFO
Supervisor
DRV11
Level-1
TRM
MPM
L2 node # 1
VBD2
MPM
L2 node # 2
VBD2
MPM
L2 node # 50
VBD2
Output Cable
Ethernet
SEQ2
Control
MPM2
XMI/VME
VMS HOST Machine

92-557-8

Simulation Goals

Design a model of the DØ / Level-2 to study present system, identify bottlenecks and explore strategies for upgrade.

Simulation model

Consists of five blocks

- Event Generation
- Digitization
- Supervisor
- Sequencer
- Level-2 node

Event Generation

Generates events that pass level-1 trigger:
exponential distribution for "real" runs
constant rate for comparison with pulser runs

Events are accepted if all VME crates have at least one buffer empty
or rejected and added to a deadtime list.

If accepted
- Event ID added to the supervisor event list
- Size of the event is generated
- Passed to the digitization block

Digitization block

- Adds constant digitization time
- If there are free VBD buffers adds transfer time along backplane and fills the buffer, otherwise waits for one to open

Supervisor block

- Checks is there an event in the event queue, and an available Level-2 node.

- If yes
  - adds startup time
  - selects free Level-2 node and assigns the event ID to it
  - signals the sequencers and Level-2 node to start processing

  otherwise waits

- Waits for all sequencers to signal they are done
- While waiting updates the list of available Level-2 nodes.

Sequencer block

Waits for the signal from the supervisor

- Delays startup time
- All sequencers start token passing in parallel to all VBD's associated with their data cable
- Data from VBD's sent to MPM's of the chosen Level-2 node
- Signals the supervisor when finished

Level-2 Node Block

Simulates software filter in Level-2 node

- Waits for signal from the supervisor block
- Generates event analysis time
- Starts analysis time as soon as at least one data cable has transfered data to HPM but at least half of the analysis time has to be spent after all data cables are done
- When done signals the supervisor.

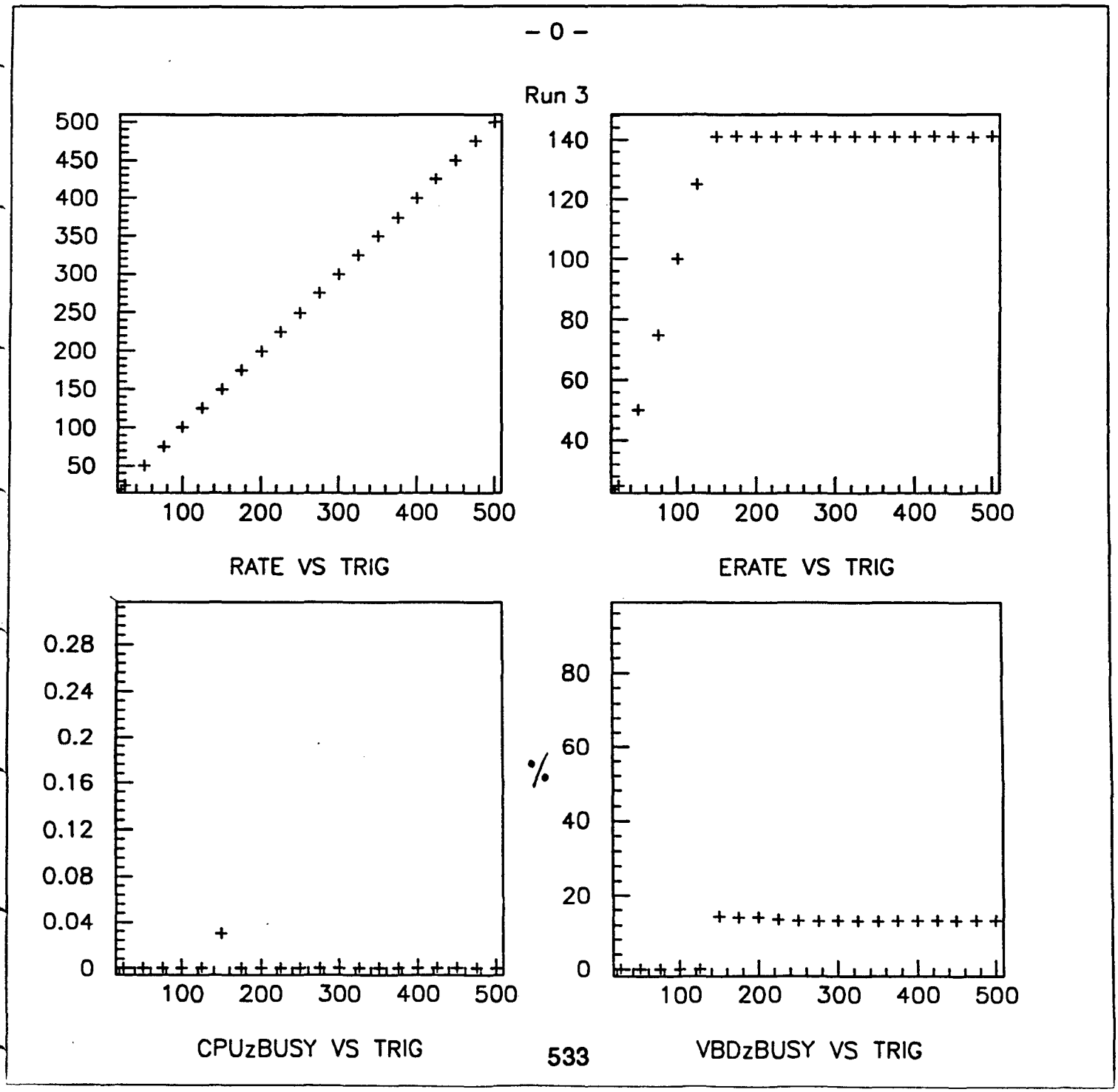
Comparison with 15 Level-2 Node System

7 crates

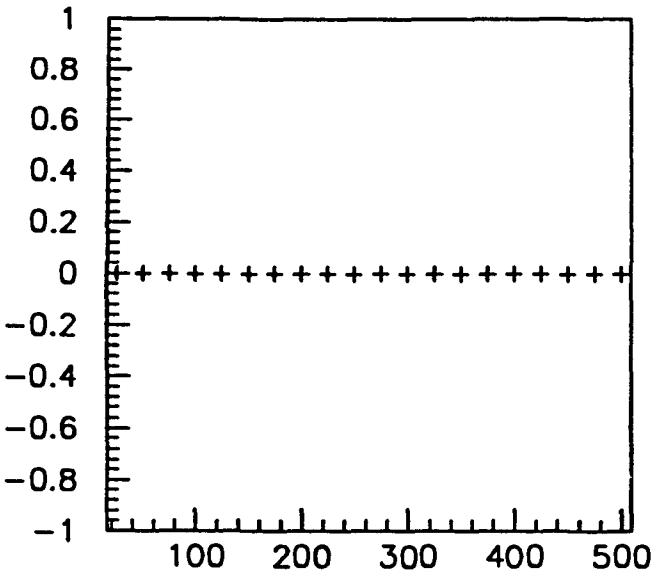
1 Data Cable

Model	Reality
Constant event size 10K Byte	Constant event size 10,704 Byte
1 ms supervisor setup time	(μVAX III) .91 ms
5 ms sequencer setup time	6.19 ms
100 ms Level-2 node analysis time	100 ms delay
141 Hz	139 - 141 Hz

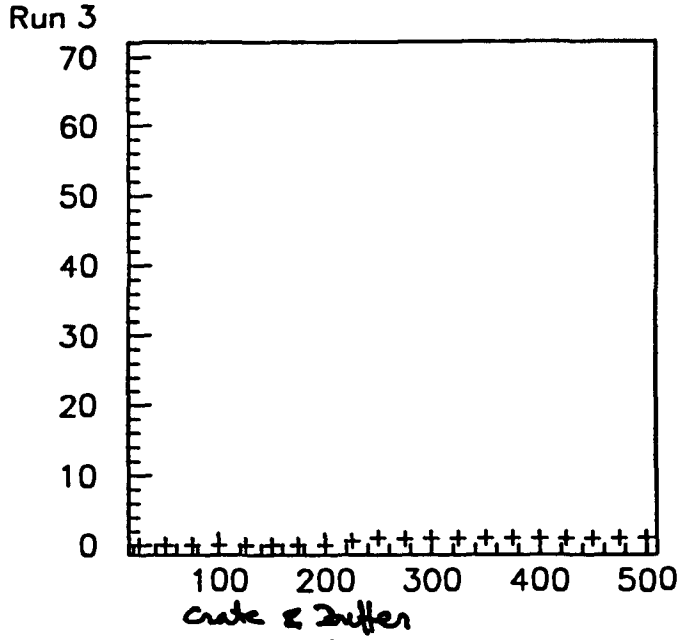
15 Level-2 nodes
7 crates
1 data cable
10KByte event size
1 ms supervisor setup time
5 ms sequencer setup time



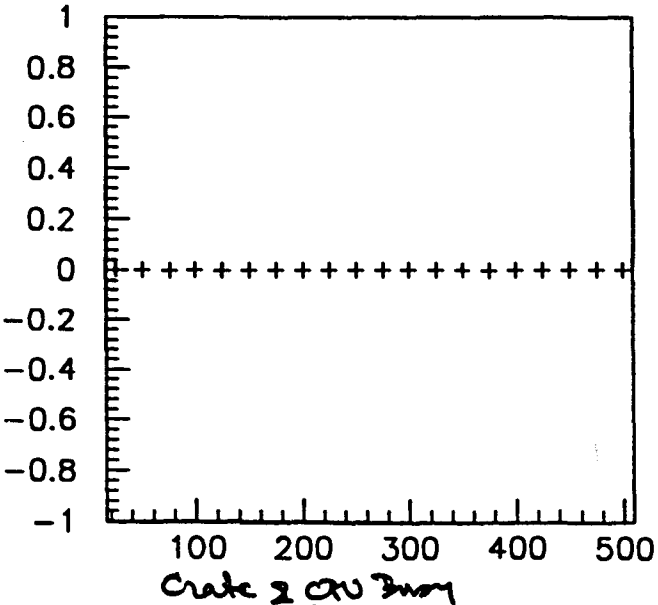
Run 3



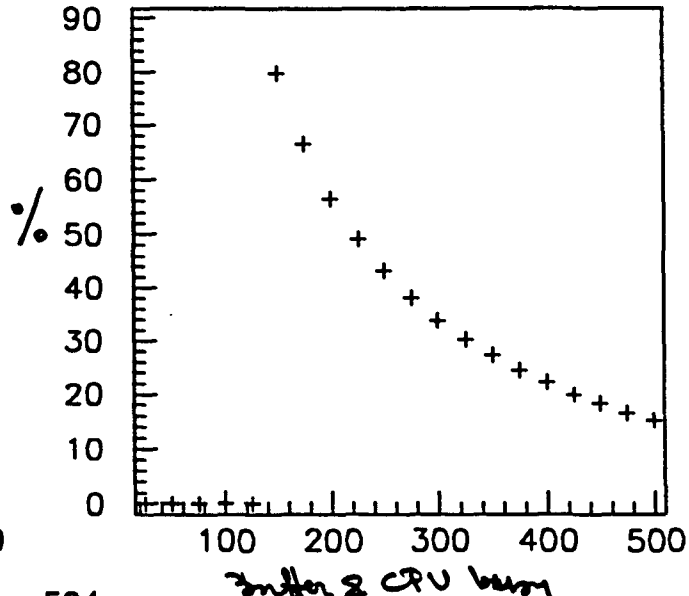
VMEzBUSY VS TRIG



~~26~~ VS TRIG



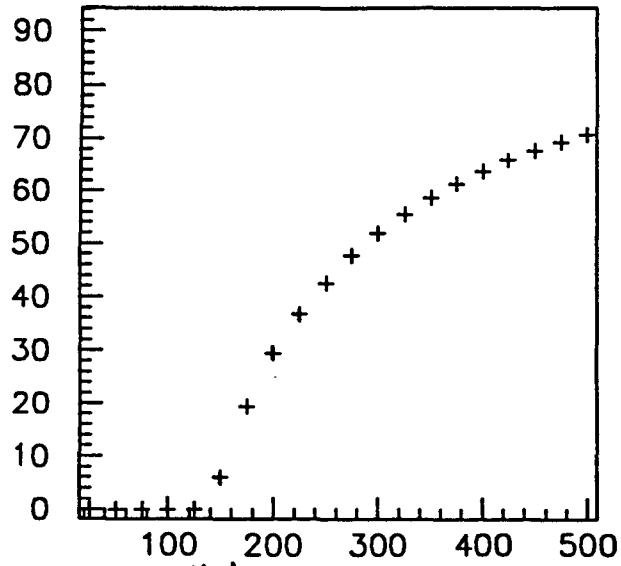
~~27~~ VS TRIG



~~28~~ VS TRIG

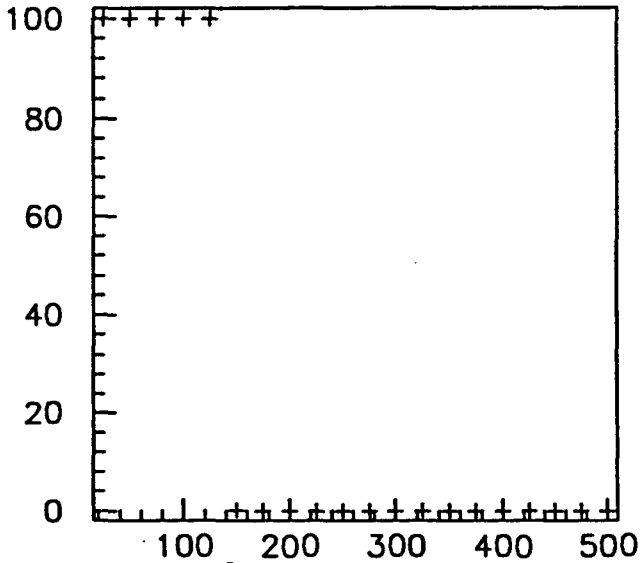
– 0 –

Run 3



all busy

29 VS TRIG



Nothing busy

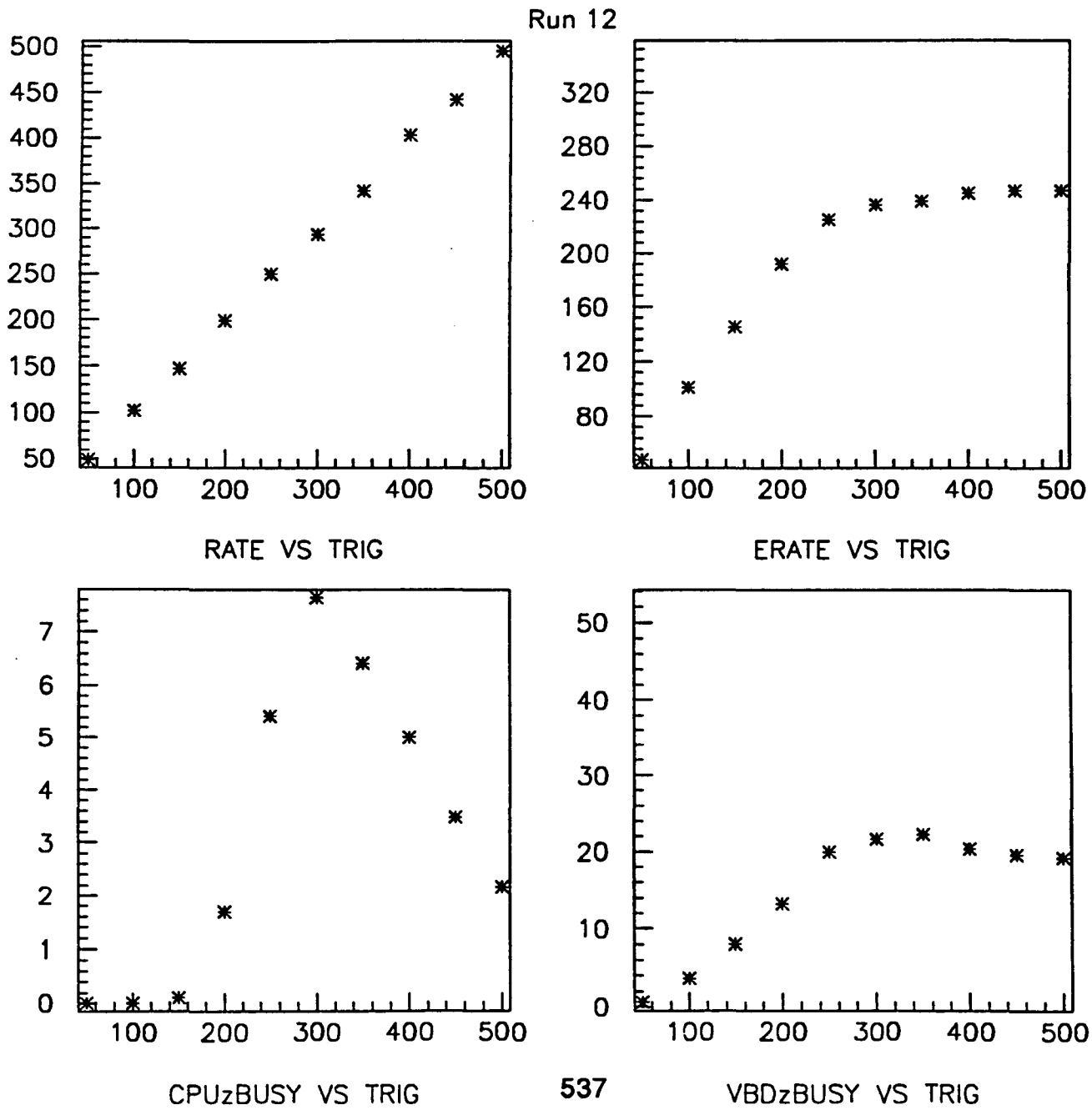
30 VS TRIG

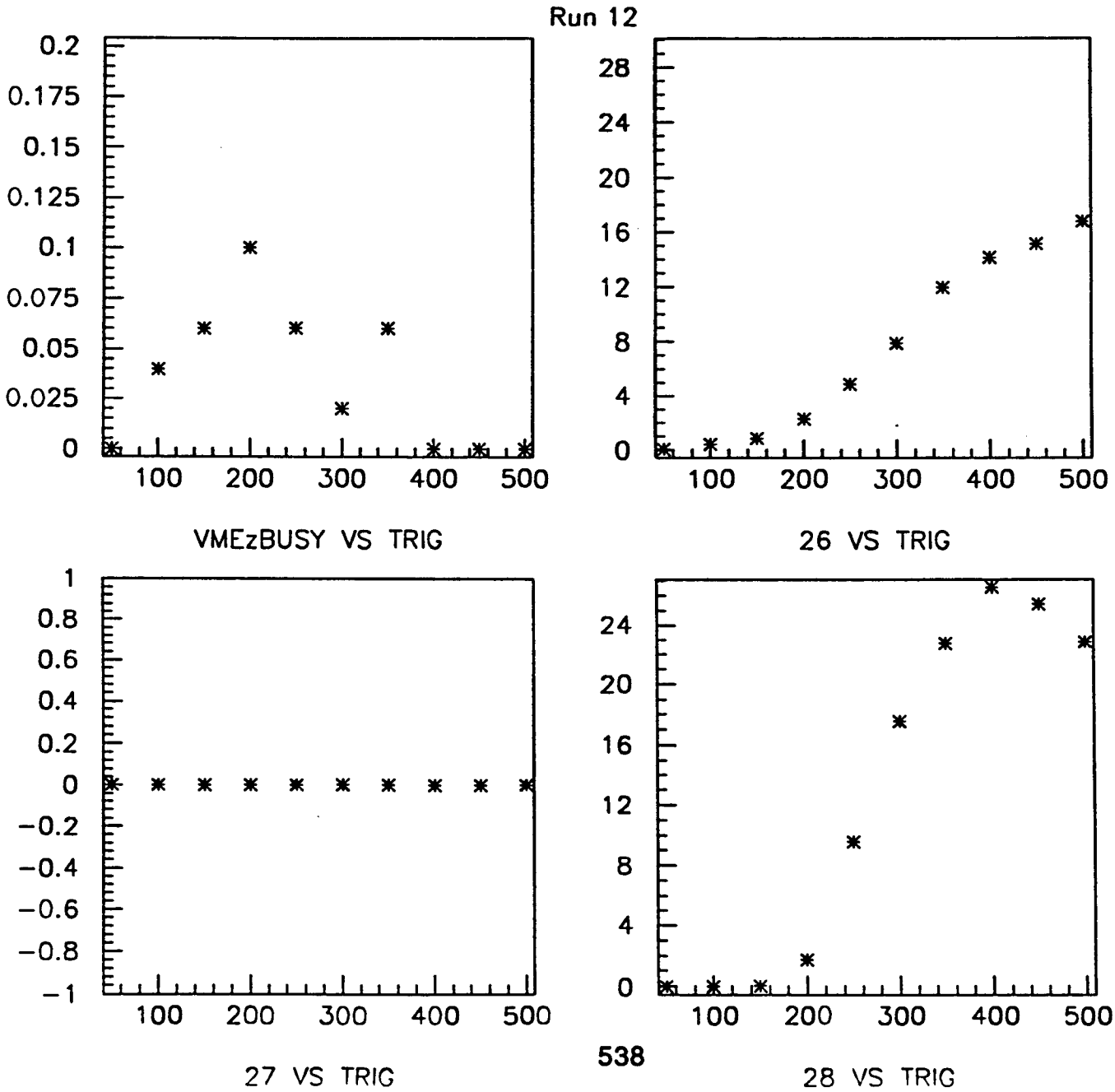
Future plans

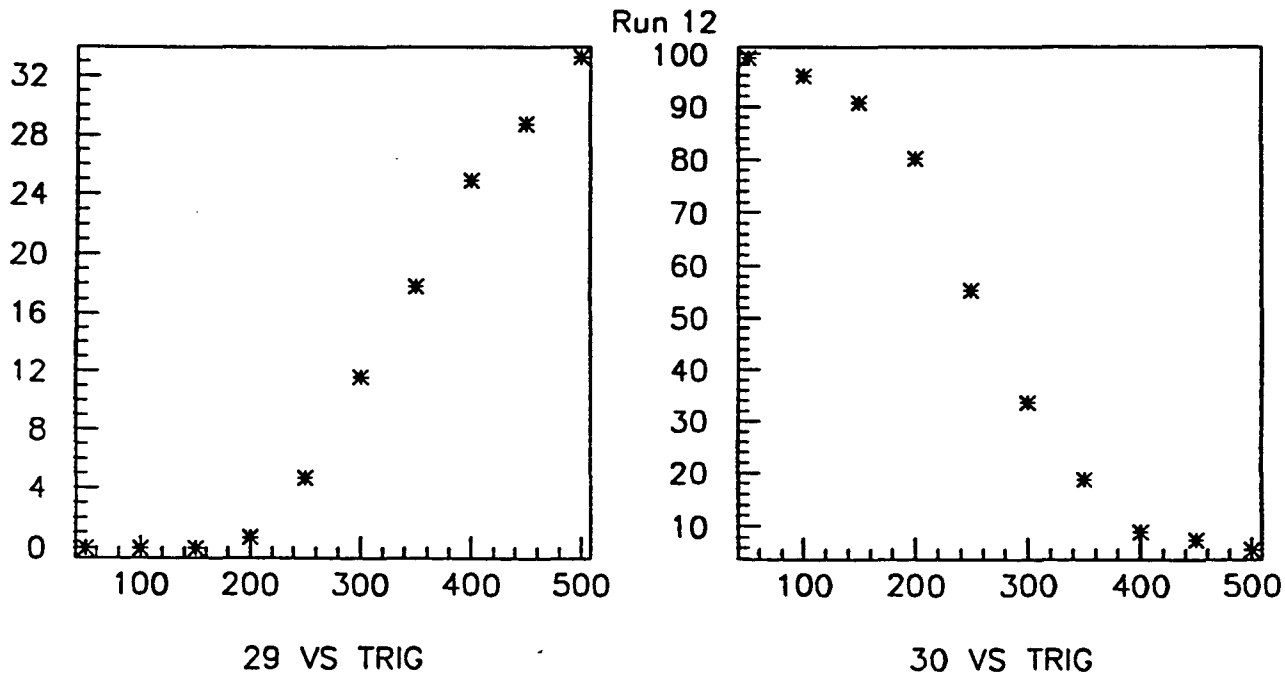
Fine tuning of the full system (50 nodes)

Further exploration of the upgrade options.

50 Level-2 nodes
8 Data Cables
79 Crates
.3 ms Supervisor time
.3 ms Sequencer time




Run 12
VMEzBUSY VS TRIG
26 VS TRIG
27 VS TRIG
28 VS TRIG


Run 12
32
28
24
20
16
12
8
4
0
100
200
300
400
500
29 VS TRIG
100
90
80
70
60
50
40
30
20
10
100
200
300
400
500
30 VS TRIG

A Fast Method for Calculating D-Zero Level 1 Jet Trigger Properties

Andrew Milder

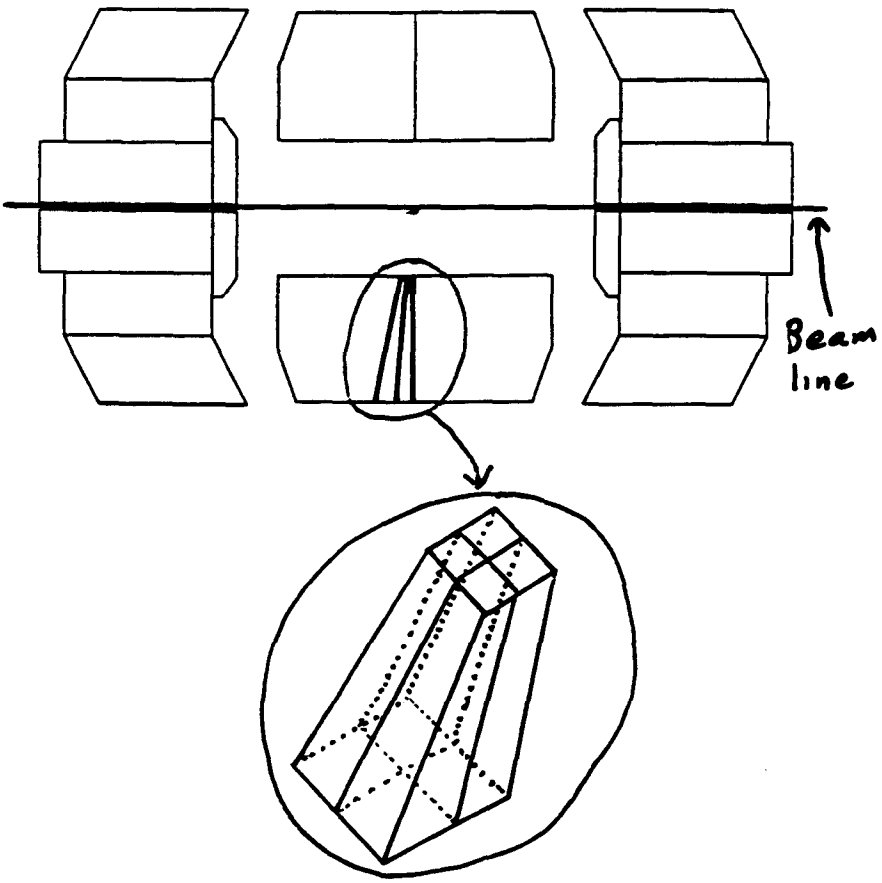
Geoffrey Forden

University of Arizona

Overview: By combining single-jet efficiencies into an overall event efficiency using a partonic Monte Carlo we can try to answer some questions about the trigger :

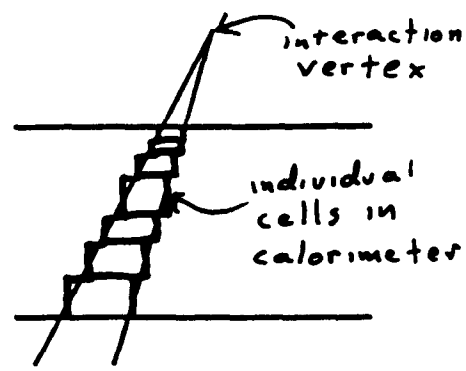
- What is the rate for a specific trigger?
- What is the efficiency versus eta, Pt
- Are there any systematic geometrical biases?

D-Zero Level 1 Jet Trigger



- Hardware trigger
- 4 adjacent calorimeter towers are summed together
- Calorimeter tower = sum of all cells with same eta, phi

Calorimeter towers are actually "psuedo-projective" :



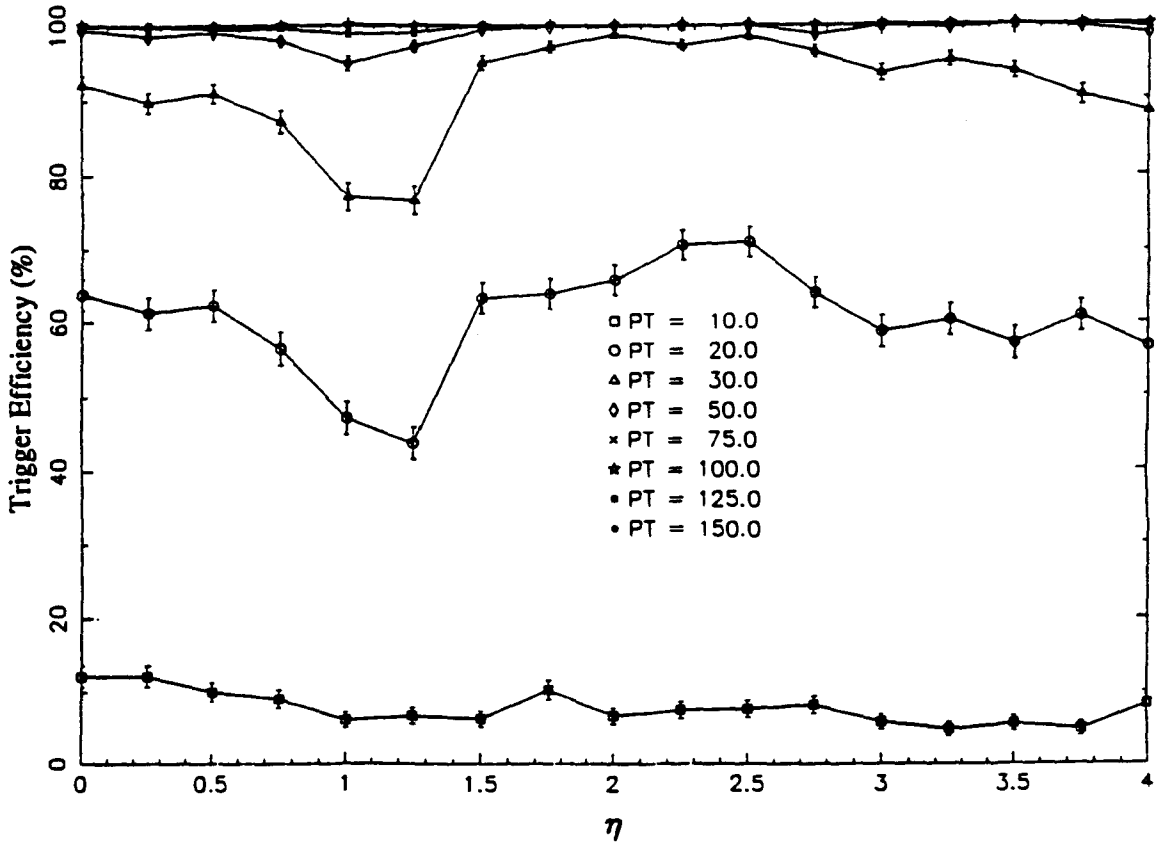
- Trigger tower Et compared to threshold
- A trigger is defined by specifying the number of towers you require to be above a given threshold

Examples:

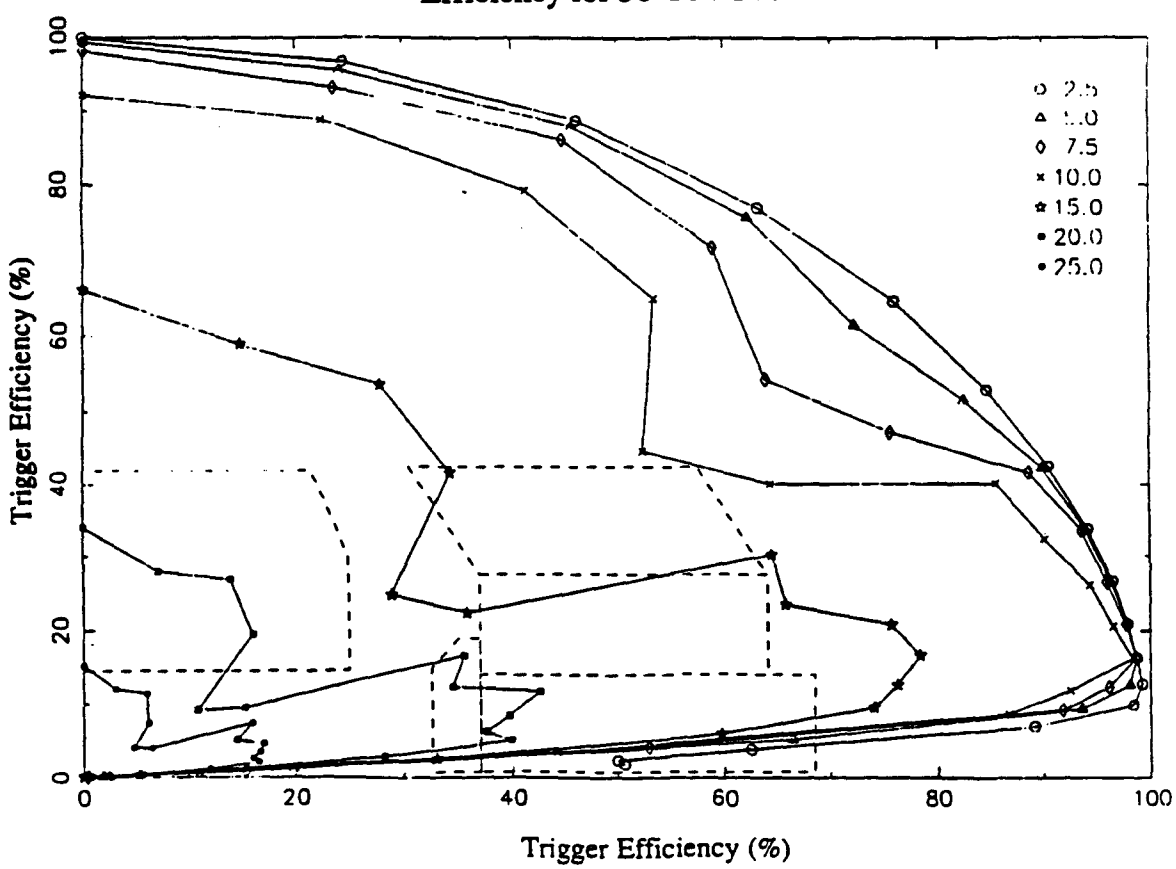
JT(1,5) - require at least one tower with 5 GeV Et

JT(3,10) - require at least three towers with 10 GeV Et each

JT(1,5)



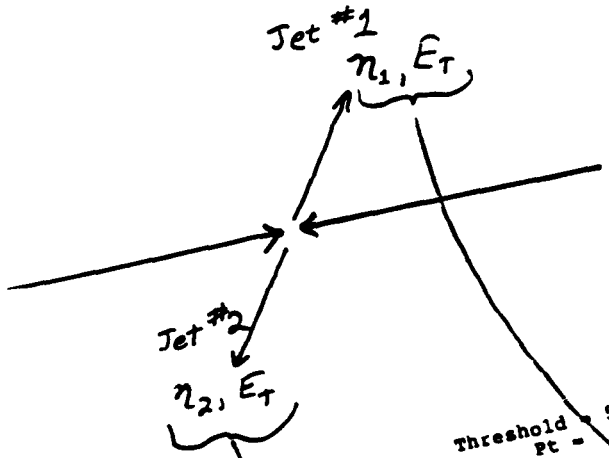
Efficiency for 50 GeV Pt Jets



Combining Single-Jet Efficiencies

Generate di-jet events using the partonic Monte Carlo Papageno which gives the eta and Pt of each parton

Simplest case: 1 trigger tower



Use eta, Pt given by Papageno to interpolate an efficiency from the table

Threshold = 5.0, 1 tower

Pt =	2.	5.	10.	20.	30.	50.	75.	100.	125
Eta= 0.00	0.00	0.60	11.60	71.80	98.40	99.60	100.00	100.00	100.0
Eta= 0.25	0.00	0.60	8.80	73.00	97.20	99.80	100.00	100.00	100.
Eta= 0.50	0.00	1.40	11.00	69.00	95.00	99.80	100.00	100.00	100
Eta= 0.75	0.00	1.00	9.40	67.00	93.40	99.00	99.40	100.00	9
Eta= 1.00	0.00	0.60	7.00	51.20	81.20	97.80	99.40	100.00	10
Eta= 1.25	0.00	0.60	4.80	48.40	82.60	98.60	99.60	99.80	1
Eta= 1.50	0.00	0.60	8.60	70.00	95.00	100.00	99.80	100.00	
Eta= 1.75	0.00	1.00	9.80	74.60	97.80	99.40	100.00	100.00	
Eta= 2.00	0.00	0.00	8.60	74.20	99.00	99.80	100.00	100.00	
Eta= 2.25	0.00	0.00	8.40	68.60	98.80	100.00	99.80	100.00	
Eta= 2.50	0.00	0.00	7.60	69.40	99.60	100.00	100.00	100.0	
Eta= 2.75	0.00	0.00	6.80	70.20	97.60	99.60	100.00	100.0	
Eta= 3.00	0.00	0.00	5.80	63.60	94.00	99.60	100.00	100.	
Eta= 3.25	0.00	0.20	4.40	61.80	96.80	99.80	100.00	100	
Eta= 3.50	0.00	0.20	4.80	60.60	96.60	99.80	100.00	10	
Eta= 3.75	0.00	0.00	8.00	58.80	91.20	100.00	100.00	10	
Eta= 4.00	0.00	0.20	8.20	60.60	94.40	99.20	99.40		

ϵ_1

ϵ_2

$1 - \epsilon_1$ = Probability that jet #1 did not cause a trigger

$$\epsilon = 1 - (1 - \epsilon_1)(1 - \epsilon_2)$$

= overall event efficiency

Rates and Efficiencies for Two-Jet Events

Rates are calculated by summing event weight times efficiency

Number of generated Papageno events

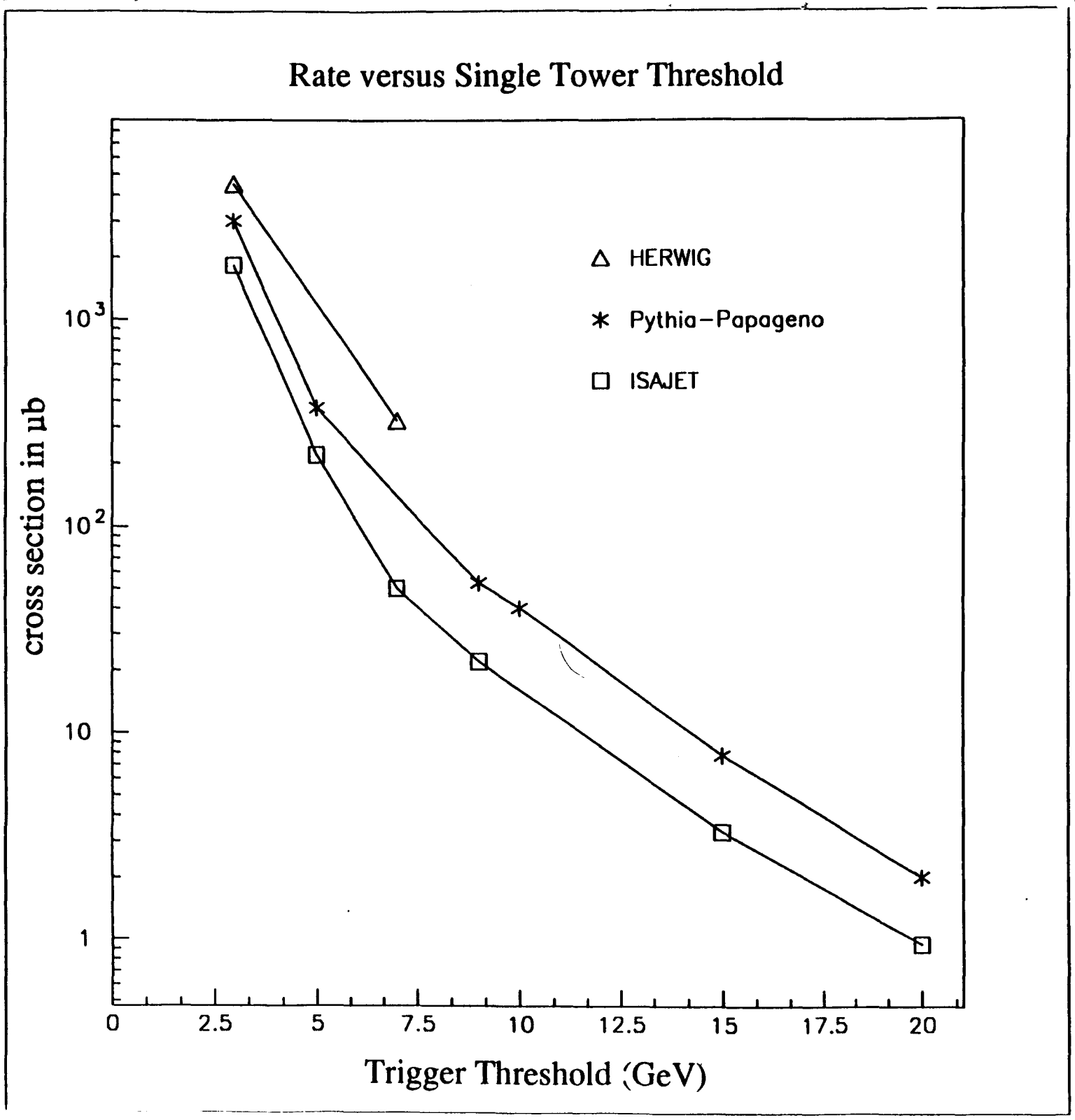
overall event efficiency

$$\sigma_{\text{passed}} = \sum_{i=1}^{N} w_i \epsilon_i$$

Event weight from Papageno

Efficiencies are obtained by dividing: $\dfrac{\sigma_{\text{passed}}}{\sigma_{\text{total}}}$

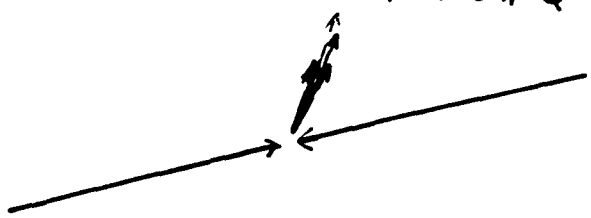
sum of weights only


Rate versus Single Tower Threshold
HERWIG
Pythia–Papageno
ISAJET
cross section in μb
10^3
10^2
10
1
0
2.5
5
7.5
10
12.5
15
17.5
20
Trigger Threshold (GeV)

Results from Single-Jet Events

- Generated single-jet events using Pythia
- Events run through Geant with D-Zero geometry

Jet at specified η, P_T
random φ (assume azimuthal symmetry)

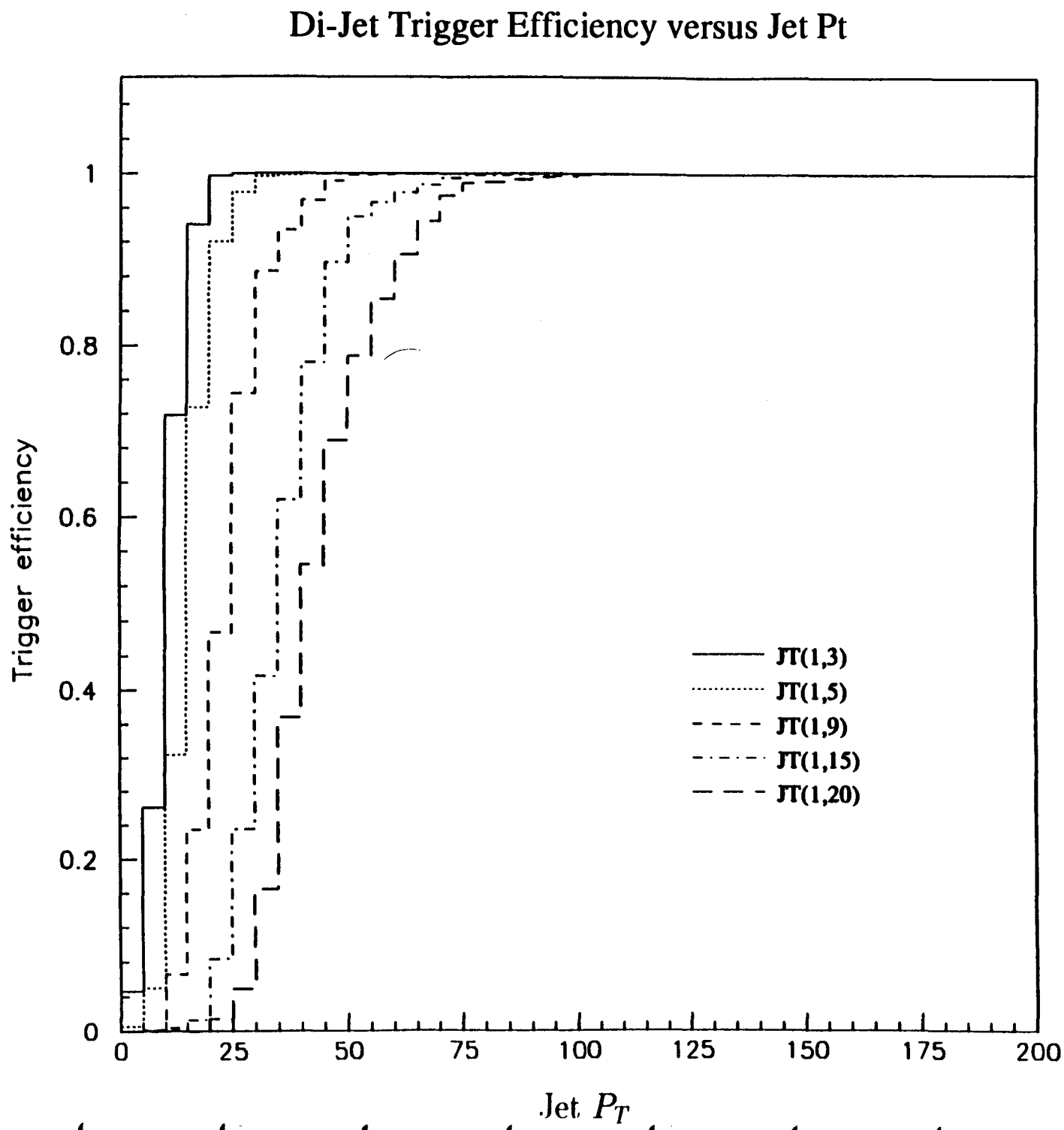


- Use level 1 trigger simulator to get efficiencies versus eta, Pt for given trigger
- Put efficiencies in a table

500 events for each eta, Pt

Threshold = 5.0, 1 tower

Pt =	2.	5.	10.	20.	30.	50.	75.	100.	125.	150.
Eta= 0.00	0.00	0.60	11.60	71.80	98.40	99.60	100.00	100.00	100.00	100.00
Eta= 0.25	0.00	0.60	8.80	73.00	97.20	99.80	100.00	100.00	100.00	100.00
Eta= 0.50	0.00	1.40	11.00	69.00	95.00	99.80	100.00	100.00	100.00	100.00
Eta= 0.75	0.00	1.00	9.40	67.00	93.40	99.00	99.40	100.00	99.80	100.00
Eta= 1.00	0.00	0.60	7.00	51.20	81.20	97.80	99.40	100.00	100.00	100.00
Eta= 1.25	0.00	0.60	4.80	48.40	82.60	98.60	99.60	99.80	100.00	100.00
Eta= 1.50	0.00	0.60	8.60	70.[illegible]	95.00	100.00	99.80	100.00	100.00	100.00
Eta= 1.75	0.00	1.00	9.80	74.60	97.80	99.40	100.00	100.00	100.00	100.00
Eta= 2.00	0.00	0.00	8.60	74.20	99.00	99.80	100.00	100.00	100.00	100.00
Eta= 2.25	0.00	0.60	8.40	68.60	98.80	100.00	99.80	100.00	100.00	100.00
Eta= 2.50	0.00	0.00	7.60	69.40	99.60	100.00	100.00	100.00	100.00	100.00
Eta= 2.75	0.00	0.00	6.80	70.20	97.60	99.60	100.00	100.00	100.00	100.00
Eta= 3.00	0.00	0.00	5.80	63.60	94.00	99.80	100.00	100.00	100.00	100.00
Eta= 3.25	0.00	0.20	4.40	61.80	96.80	99.80	100.00	100.00	100.00	100.00
Eta= 3.50	0.00	0.20	4.80	60.60	96.60	100.00	100.00	100.00	100.00	100.00
Eta= 3.75	0.00	0.00	8.00	58.80	91.20	100.00	100.00	100.00	99.80	100.00
Eta= 4.00	0.00	0.20	8.20	60.60	94.40	99.20	99.40	99.40	99.40	99.80


Di-Jet Trigger Efficiency versus Jet Pt
Trigger efficiency
Jet P_T
0
0.2
0.4
0.6
0.8
1
0
25
50
75
100
125
150
175
200
JT(1,3)
JT(1,5)
JT(1,9)
JT(1,15)
JT(1,20)

Summary

By using this method and applying efficiencies from the tables to QCD di-jet events we are able to generate rates and efficiencies quickly. Systematic eta effects which have been missed in other studies become apparent and represent a potentially important angular correlation efficiency problem.

E. Wang
SSCL
25-Apr-1992

Physics Input to DAQ Studies

Outline

1. Detector Simulation

2. Trigger Algorithms/ Rates

3. Features of the "data"

Triggering at the SSC

(only one of a number of possibilities)

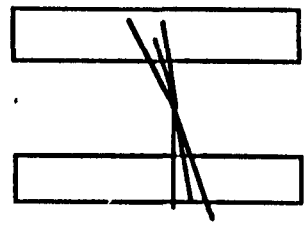
in		out	algorithm
60Mhz	Level 0	60Mhz	load FE buffers
60Mhz	Level 1	100 Khz	Et clusters, muon "roads"
100 Khz	Level 2	1-10 Khz	Isolation, shower-tk, mass
1-10Khz	Level 3	1-10 hz (to storage)	Initial physics analysis using full event info

We wish to generate representative samples of these events

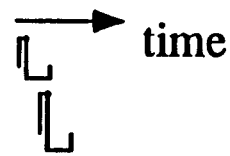
Procedure

1. Generate event
   (Isajet, Pythia ...)

2. Detector simulation
   (Geant , other fast sim)



   include pileup (previous pulse history)



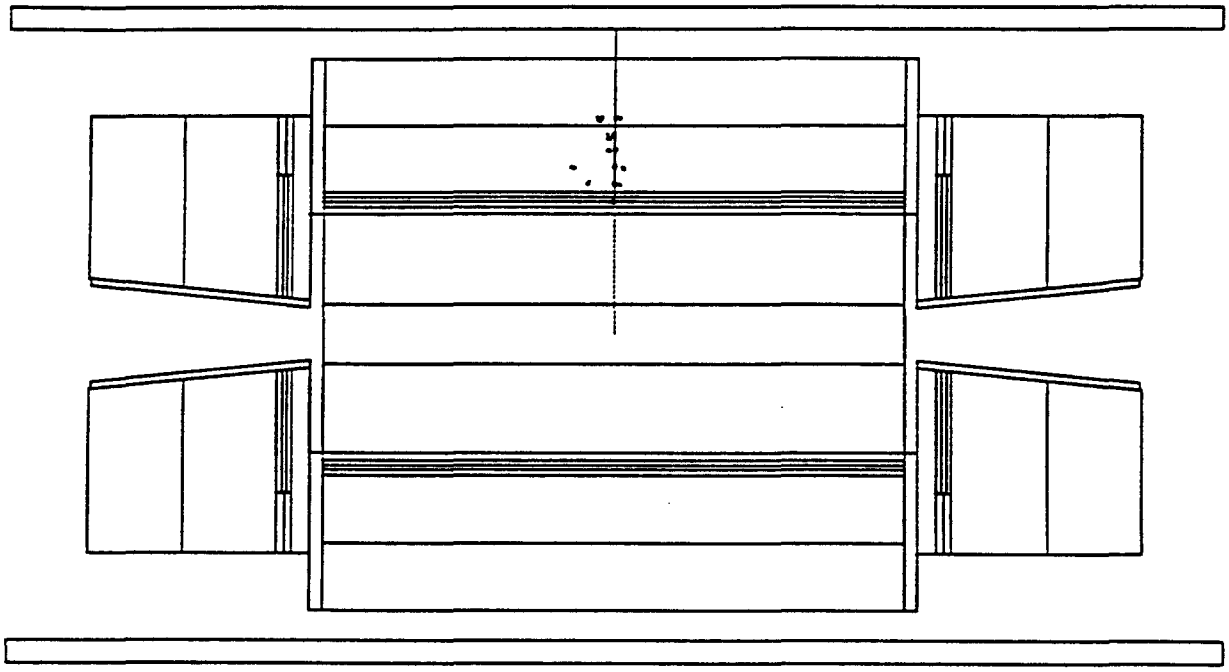
3. Impose Trigger Algorithm

4. Store events that pass on disk

---

5. Modsim reads trigger file and does it's thing...

Geometry



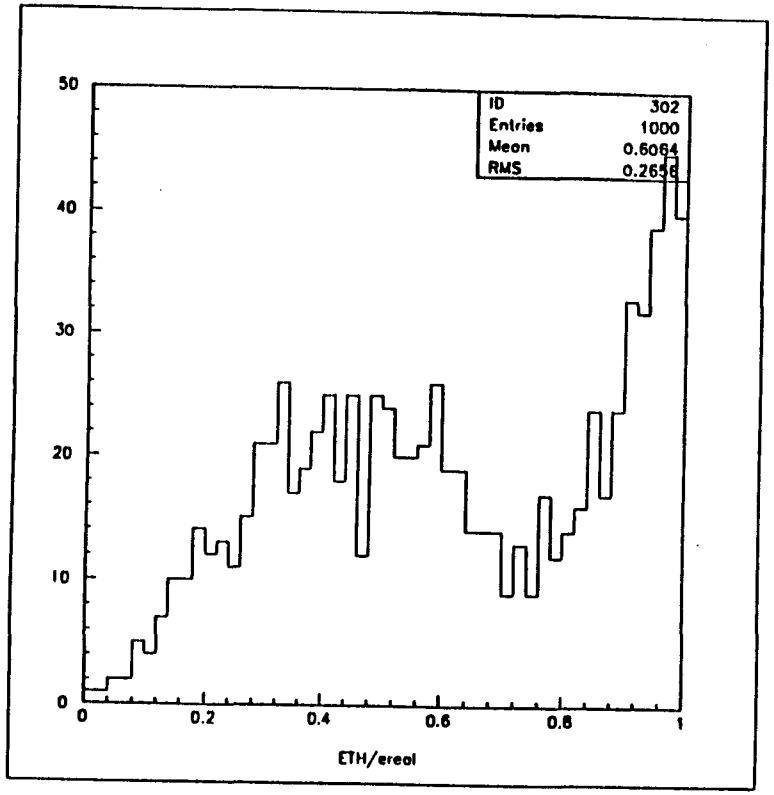
Calorimeter Simulation

1. Full Simulation too slow (few minutes per event).

2. Parametrized Showers

deposit energy by throwing
spots weighted by transverse and longitudinal shapes
determined by CDF test data and Fluka calculations (~10 s/ev)

CDF #753 Note



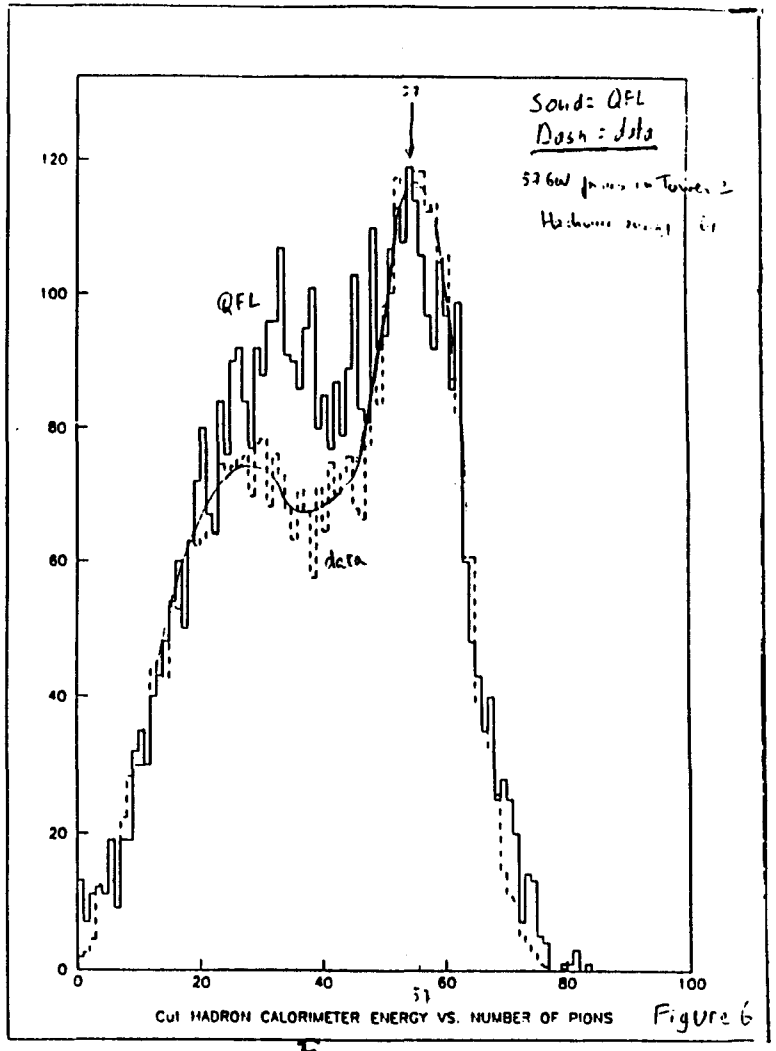


Figure 6

CDF Test module

Trigger Types

Find thresholds/ algorithms that give rough L2 rates

QCD jets (worst case?)

single EM (i.e. $W+ X_{\text{interesting but rare}} + X_{\text{uninteresting}}$)

$W \rightarrow e$

$X_{\text{interesting but rare}} \rightarrow H \rightarrow bb$

Missing Jet ET (we wont discuss this today)

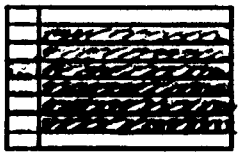
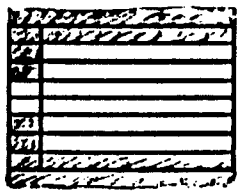
QCD events

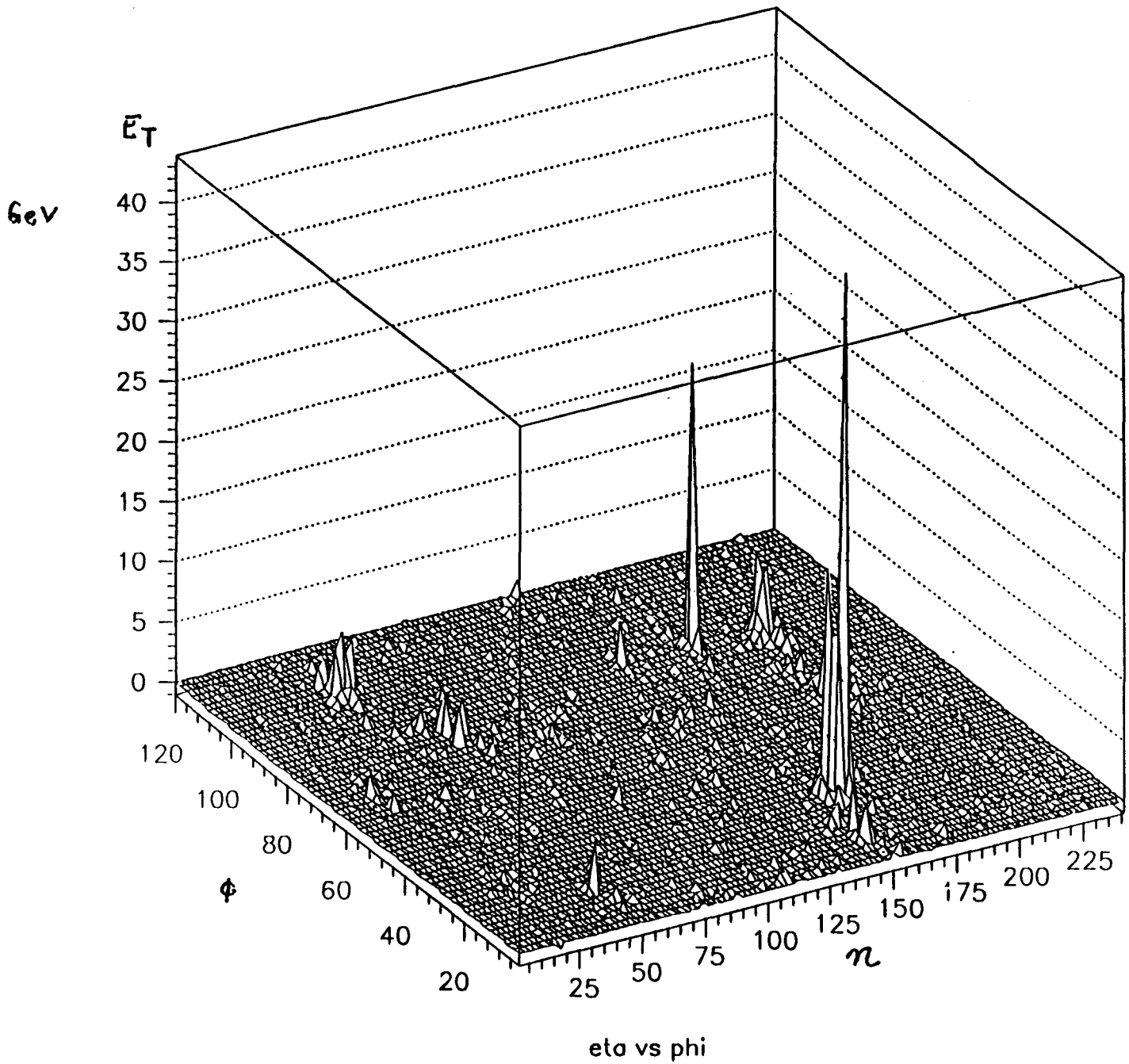
Standard "two jet" events are the primary source of triggers.

We generated a sample of QCD 2-jet events with jet E_t >100 GeV

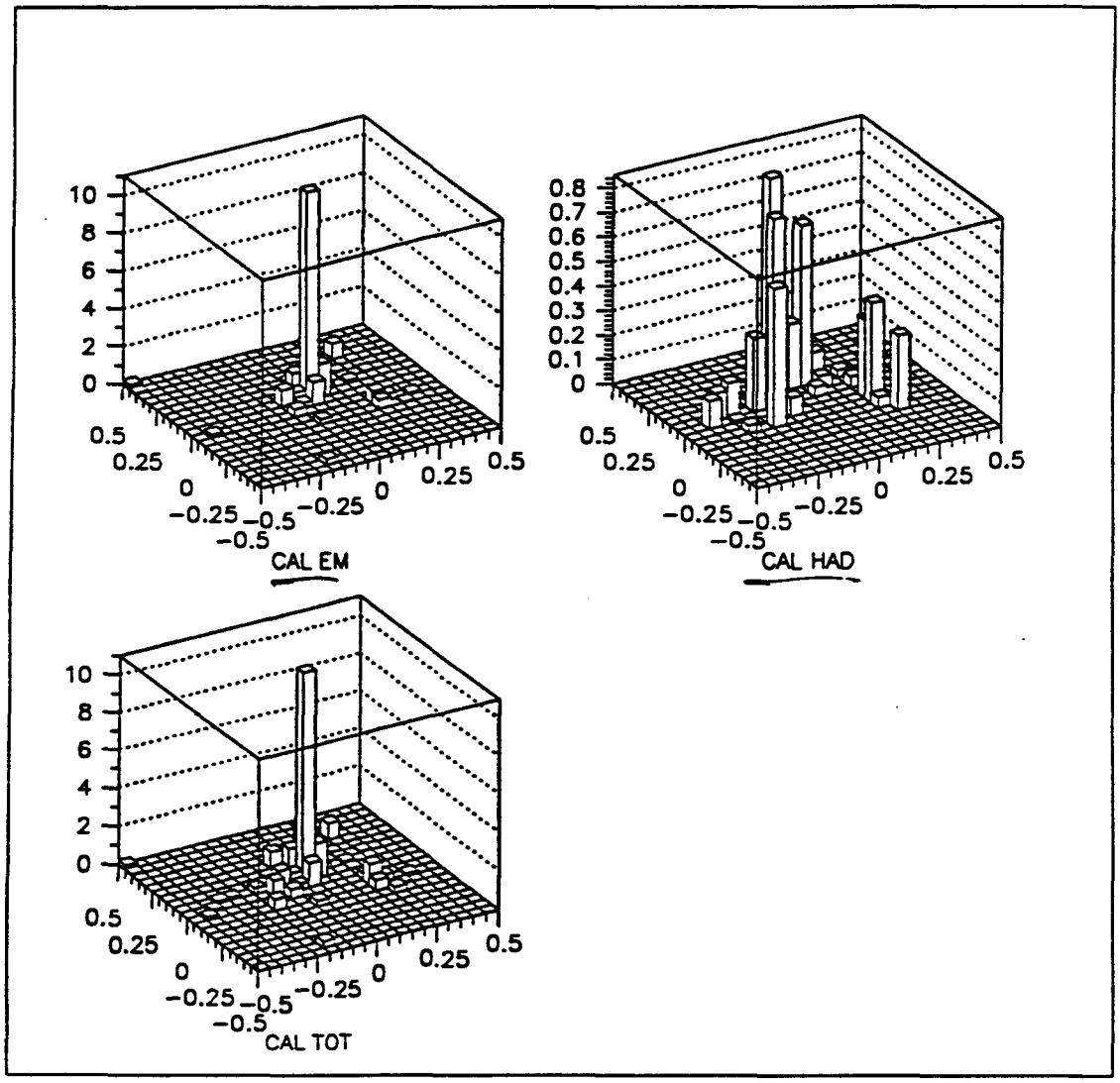
Such events occur at 10Khz.

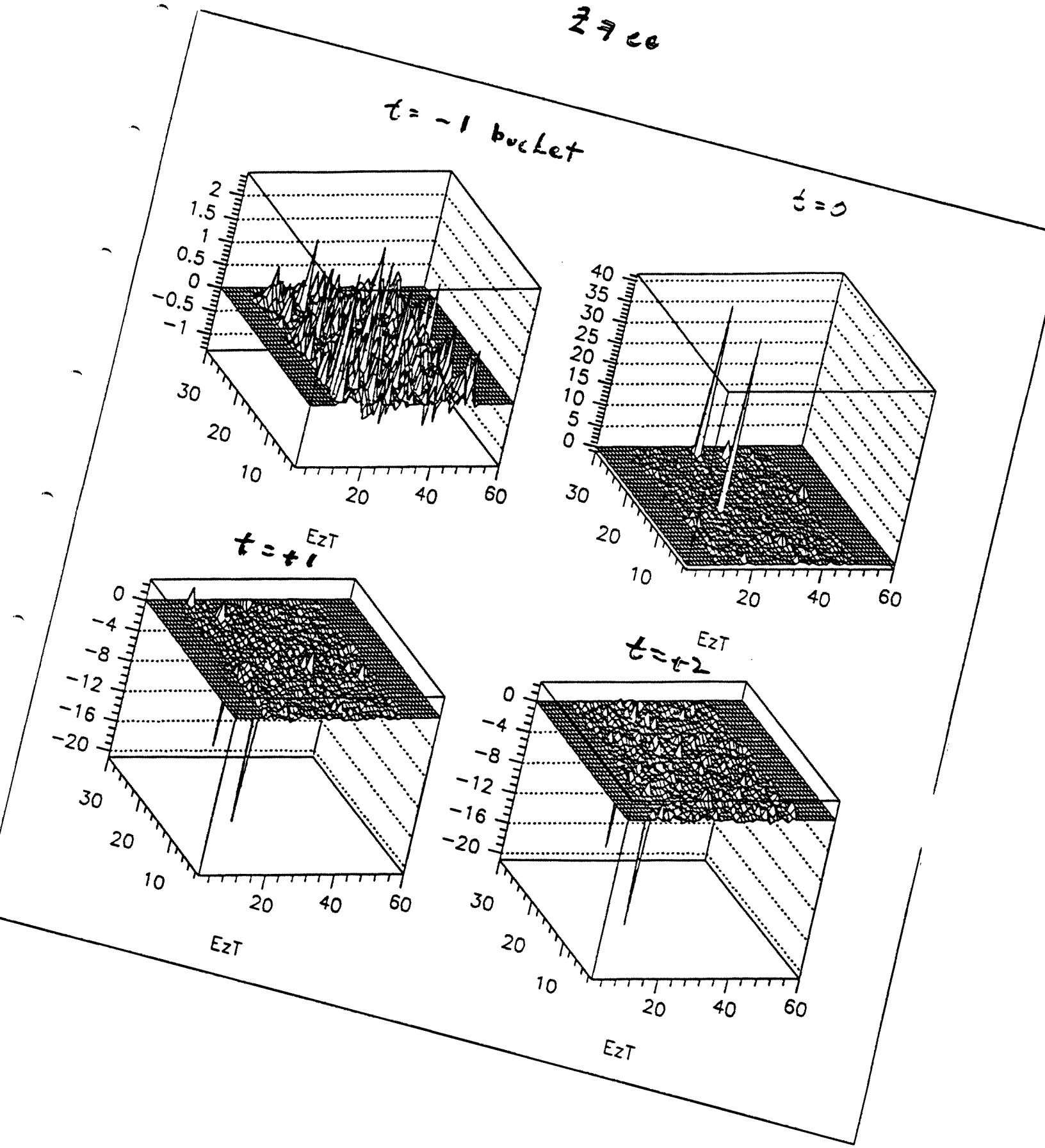
1- Electron Trigger Algorithm

		trigger rate
$E_{t\ em} > 15$ GeV in 2x2 cells		100Khz
$E_{t\ had}/E_{t\ em} < 0.2$		
$E_{t\ border}/E_{t\ em} < 0.2$		~10 Khz

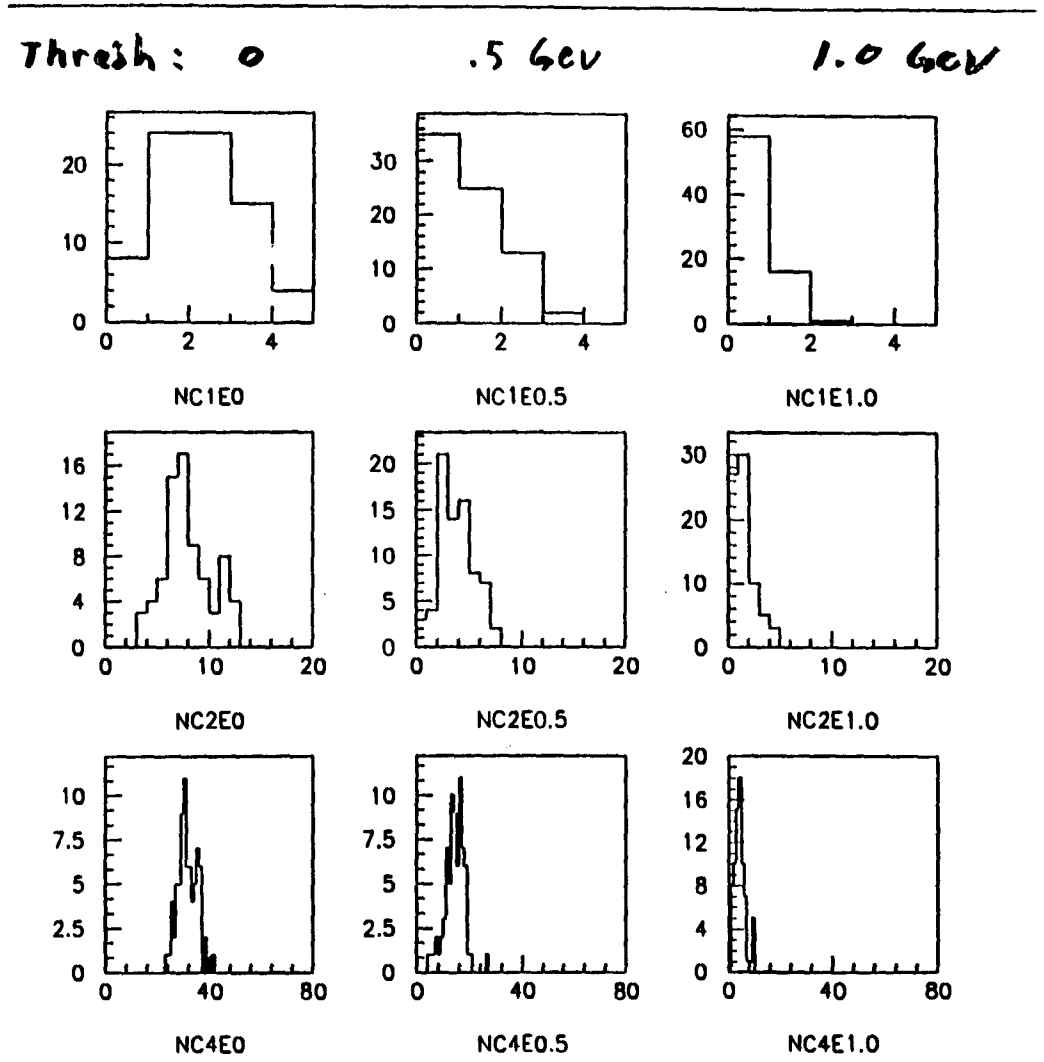

E_T
GeV
40
35
30
25
20
15
10
5
0
120
100
80
60
40
20
ϕ
25
50
75
100
125
150
175
200
225
η
eta vs phi

QCD → "electron" trigger




t = -1 bucket
t = 0
t = t1
t = t2
EzT
EzT
EzT
EzT

1-Electron triggers



Cells Above Threshold Distributions
(noise- 60 MeV em, 630 MeV hadronic)

Speed

physics generation (isajet)	1 s /evnt
detector simulation	5-10s/evnt

(read in pileup that was previously simulated)

depending on trigger, have to generate >> 10K events

Summary

- We have cleared a path that allows one to link physics first principles to detector DAQ/trigger architecture design.

DAQ & TRIGGERING SYSTEM SIMULATION WORKSHOP

SOME SUMMARIZING REMARKS (FROM A TOOLS PERSPECTIVE)

ALEX BOOTH

* SYSTEM & GATE LEVEL SIMULATION

  MODSIM, VERILOG, VHDL, C++, RESQ, SIMULA, DAGGER, LOTUS-PROLOG,

* VARIOUS APPROACHES TO UNDERSTANDING THROUGHPUT & DEADTIME

* DAQ SYSTEM DESIGN IS COMPLEX REQUIRING A VARIETY OF EXPERTISE

* WHATEVER TOOL WE USE TO DESIGN DAQ SYSTEMS, THE IMPORTANCE OF THE PHYSICS ISSUES AND TRIGGERING CAN NOT BE OVERSTATED

TRIGGERING

SDC LEVEL 1 TRIGGER	S. DASU UNIV. OF WISCONSIN
SDC RATE SIMULATIONS	GREG SULLIVAN UNIV. OF CHICAGO
OPTIMIZING DATA LOAD IN THE DZERO TRIGGER	MIKE FORTNER NORTHERN ILL. UNIV.
DZERO JET PROPERTIES	ANDY MILDER UNIV. OF ARIZONA
PHYSICS SIMULATION FOR INPUT TO BEHAVIORAL SIMULATIONS	ED WANG SSC

C ++

EFFICIENT DATA TRANSMISSION — KLAUS LACKNER, LOS ALAMOS

SCI — OSLO UNIV.

SIMULA

SCI — OSLO UNIV.

DAGGER

DATA FLOW SYNTHESIS — VIJAY RAJ, UTA

LOTOS-PROLOG

ALGEBRAIC LANGUAGE FOR SIMULATION — M. L. FERRER, INFN

"SYTHESIS TOOLS"

SILICON COMPILER "MEET IN THE MIDDLE" — GUY VANSTRAELEN, SSC

VHDL

DCC STUDIES

GEORGE THAKAKAN
& ERIC HUGHES
UNIV. OF ILLINOIS

VISUAL VHDL

DIANA MILLER-KARLOW
UNIV. OF ILLINOIS

RESQ

DZERO DAQ SYSTEM

JAY WIGHTMAN
FNAL

DZERO TRIGGER

MARVIN JOHNSON
FNAL

VERILOG

SDC STRAW TUBE TRACKING SYSTEM	ANDREAS HOLSCHER UNIV. OF TORONTO
HEAVILY BUFFERED NON-BLOCKING ARCHITECTURES	RICH PARTRIDGE BROWN UNIV.
DEADTIME STUDIES IN THE DCC	ERIC HUGHES UNIV. OF ILLINOIS
SCI - "node chip". INTERFACE	DOLPHIN SERVER TECHNOLOGY
CDF DAQ SYSTEM	ROBIN GRINDLEY UNIV. OF TORONTO
SDC STRAW TUBE FRONT END	TOR EKENBERG UNIV. OF PENN.
DZERO LEVEL 2	D. NESIC BROWN UNIV.

MODSIM II

* ENHANCEMENTS TO MODSIM & SIMOBJECT GRAPHICAL TOOL — LON BELANGER, ERGOSOFT

* DAQSIM – A TOOL FOR UNDERSTANDING DAQ SYSTEMS — ALEX BECH ET AL, SSC

* SCI — CERN, ANDRE BOGAERTS

* ASYNCHRONOUS BUFFERS — HENRY KASHA, YALE UNIV.

* BARREL SWITCH EVENT BUILDER — VISHAL KAPOOR, UNIV. OF TEXAS AT ARLINGTON