Evaluation and Simulation of Event Building Techniques for a Detector at the LHC

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Abteilung Physik
an der Universität Dortmund

vorgelegt von Diplomphysiker
Ralf Spiwoks
aus Duisburg

- Dortmund, Oktober 1995 -

Gutachter:

Prof. Dr. C. Gößling
Universität Dortmund

Dr. L. Mapelli
CERN

Abstract

The main objectives of future experiments at the Large Hadron Collider are the search for the Higgs boson (or bosons), the verification of the Standard Model and the search beyond the Standard Model in a new energy range up to a few TeV. These experiments will have to cope with unprecedented high data rates and will need event building systems which can offer a bandwidth of 1 to 100 GB/s and which can assemble events from 100 to 1000 readout memories at rates of 1 to 100 kHz. This work investigates the feasibility of parallel event building systems using commercially available high speed interconnects and switches. Studies are performed by building a small-scale prototype and by modelling this prototype and realistic architectures with discrete-event simulations.

The prototype is based on the HiPPI standard and uses commercially available VME-HiPPI interfaces and a HiPPI switch together with modular and scalable software. The setup operates successfully as a parallel event building system of limited size in different configurations, with different input data and different data flow management schemes. Realistic parameters of 40 MB/s for the link speed and of 100 µs for the overhead have been measured and the total throughput is scalable with the number of destinations. The prototype measurements lead to a parametrized model of a parallel event building system which is implemented in a simulation program. This is used to simulate large-scale systems including a realistic model of the ATLAS event building system with realistic event size distributions from off-line simulations. The influence of different parameters and the scaling behaviour are investigated. Different data flow management schemes for destination assignment and traffic shaping are studied as well as a two-stage event building system.

Acknowledgements

I would like to thank my supervisor Prof. Dr. C. Gößling for his support and guidance over the last few years. I am also very grateful to Dr. L. Mapelli for his support and supervision of this work which was started at his suggestion. I am indebted to both for their help to obtain a CERN doctoral fellowship without which this work would not have been possible.

My stay at CERN has been of immense worth to me and I am in particular grateful to all members of the RD13 collaboration for the friendly atmosphere and the many fruitful discussions. I would especially like to thank Dr. G. Mornacchi for his competent help and advice in all questions of data acquisition and event building systems, Dr. S. Buono for helping to begin with the HiPPI standard and modules, Dr. R. Jones for his help in questions of computing and I. Gaponenko for the many discussions on event building and modelling.

I would like to thank Drs. E. v.d. Bij and R. McLaren from CERN/ECP for their advice with the HiPPI hardware and software and for the loan of some of the modules. I am also indebted to Dr. W. Greiman from LBL for the original version of the simulation program and to K. Djidi from Saclay for the collaboration on DSL and its GUI. I am very grateful to Dr. R. Hawkings from CERN/PPE for his help with the ATRIG program and for providing the trigger events. I would also like to thank Dr. T. Hansl from Saclay for making the ATRIG program run.

I would like to thank all of the people in RD11, RD24, RD31 and the ATLAS DAQ groups for the friendly atmosphere and the fruitful discussions. In particular, am I very grateful to Dr. A. Bogaerts, D. Calvet, Dr. F. Harris, S. Hunt, W. Iwanski, Dr. K. Korcyl, Dr. M. Letheren, Dr. I. Mandjavidze and Dr. J. Vermeulen.

I am also indebted to D. Klein for her support with many organizational aspects of the work, in particular in helping to print it. I would like to thank Dr. E. Tsesmelis for proofreading this work and for the good company over the last few years. I am also indebted to D. Hearty for looking through this work and for being a good friend.

I would like to warmly thank Keiwan Fakhr-Soltani for his understanding and patience without which this work could not have been finished.

Table of Contents

1.0 Introduction

The Standard Model (SM) is a description of all elementary particles and their interactions known today. Its biggest successes are the predictions of phenomena which could subsequently be discovered and measured in experiments, like the intermediate vector bosons [Arn83] and the top-quark [Abe95].

But there are also still open questions: the Higgs boson which explains the spontaneous breaking of symmetry in the electroweak sector of the SM has not yet been discovered. Its existence is closely related to a very fundamental question, namely: what is the origin of the different particle masses? And furthermore, when searching at ever higher energies there is always the possibility to discover new physics which could change the understanding of particle physics.

In order to research in this direction the Large Hadron Collider (LHC) [Eva95] was proposed as a hadron collider with unprecedented high energies and high luminosity. General-purpose experiments at this collider will be able to explore a new field of particle physics with high precision measurements. They will have to cope with high data rates which require sophisticated trigger and data acquisition systems to observe and select the rare and interesting phenomena.

An event building system is a part of the trigger and data acquisition system where data from events belonging to the same interaction are assembled. It has to match with the trigger rates up-stream and down-stream, must not introduce deadtime and not lose event data. Its performance is determined by the input rate and the event fragment size distributions and their correlations. Future experiments at the LHC [ATL94][CMS94] will need event building systems with a bandwidth of 1 to 100 GB/s and which will be able to assemble events from 100 to 1000 data sources at rates of 1 to 100 kHz. Bus based systems are not appropriate to fulfill this task and parallel event building based on high speed interconnects with switching elements has to be envisaged.

This work presents studies carried out to prepare a future experiment at the LHC. The feasibility of event building systems using commercial high speed interconnect standards and commercially available communication switches is investigated in two complementary ways: by building a prototype and by simulations. A small-scale prototype has been built to adapt a given technology to the task of event building and to obtain a realistic model of the switch. Simulations, on the other hand, use the model and the parameters measured in the prototype and extrapolate to full-scale systems as needed in a future experiment. Realistic event size distributions can be studied as well as different data flow managment schemes.

In chapter 2 this work presents an overview of the LHC, its physics issues and one of the general-purpose experiments with its detector lay-out and its trigger and data acquisition system. Chapter 3 defines the components needed in an event building system and discusses different architectures and high speed interconnect standards available. The prototype and its measurements are presented in chapter 4. Chapter 5 presents the simulation program used. And simulations of realistic event building systems are discussed in chapter 6. Different control schemes of event building systems are compared in chapter 7.

2.0 ATLAS Experiment

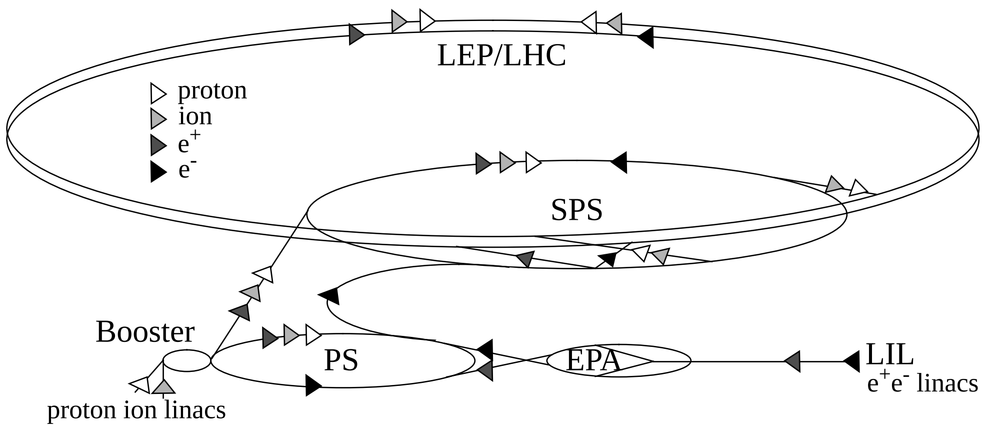
2.1 Large Hadron Collider

The LHC [Eva95] is a future proton-proton collider at CERN operating at a centre-of-mass energy of $\sqrt{s} = 14$ TeV and at a nominal luminosity of $L = 10^{34}$ cm^{-2}s^{-1}. The LHC will reach higher energies than ever achieved before and thus open a new field of research in particle physics, in particular for the search for the Higgs boson.

2.1.1 Machine

The LHC, approved in 1994, will be operational in 2004. It will be accommodated in the LEP tunnel. The existing accelerator complex at CERN consisting of the 50 MeV linac, the 1 GeV booster, the 26 GeV Proton Synchrotron (PS) and the 450 GeV Super Proton Synchrotron (SPS) constitute an excellent injection complex for the LHC as shown in figure 2.1.

FIGURE 2.1. LHC and Injection Complex at CERN



Furthermore, constructing the LHC in the LEP tunnel opens up the possibility of having electron-proton collisions of a centre-of-mass energy up to $\sqrt{s} = 1.7$ TeV. The LHC can further be used for heavy ion collisions and reaches for lead ions a centre-of-mass energy of up to $\sqrt{s} = 1312$ TeV. Two general-purpose pp experiments (ATLAS [ATL94] and CMS [CMS94]) and one heavy ion experiment (ALICE [ALI93]) are proposed.

TABLE 2.1. Parameters of the LHC

Circumference	27 km
Proton Energy	7.0 TeV
Luminosity	10^{34} cm^{-2}s^{-1}
Bunch Spacing	25 ns

2.1.2 Magnets

A technical challenge which must be overcome at the LHC is the realization of the superconducting dipole magnets [Eva95] providing a field of 8.4 T. Figure 2.2 shows a cross-section of the magnets with two counter-rotating proton beams in one device. This design uses copper-clad niobium-titanium windings and operates at a temperature of 1.9 K. Some other parameters are listed in table 2.2. A total of 1,296 dipole magnets are needed, plus 2,500 other magnets ranging from normally conducting bending magnets to large superconducting focusing quadrupoles. The cryogenics system for the magnets will contain some 700,000 l of liquid helium and have a power consumption of about 140 kW.

FIGURE 2.2. LHC Magnets

Sc. Bus-Bars
Iron Yoke
Shrinking Cylinder
Thermal Shield
2.2 K He Pipe
Radiative Insulation
Vacuum Vessel
1.8 K He Pipe
50+75 K He Pipe
Heat Exchanger Pipe
Superconducting Coils
Beam Screen
4.5 K He Pipe
Non-Magnetic Collars
Beam Pipe
20 K He Pipe
Support Post
50+75 K He Pipe
Alignment Target

TABLE 2.2. LHC Dipole Magnet Parameters

Operational field	8.4 T
Coil Aperture	56 mm
Distance between Aperture Axes	180 mm
Outer Diameter of Cryostat	980 mm
Magnetic Length	14.2 m
Operating Current	12 kA
Operating Temperature	1.9 K

2.2 Physics Issues at the LHC

The LHC will open a new field of research in particle physics which cannot be accessed with today’s experiments. The SM will be tested in an energy range where it has not been challenged before.

2.2.1 Standard Model

Today’s understanding of particle physics is summarized in the SM which describes all known phenomena in the world of particles and their interactions. In the SM there is a set of fundamental particles and their interactions based on fundamental forces. All known matter is made up from these particles which are spin-1/2 particles (fermions). There are two groups: the leptons and the quarks. The leptons interact electromagnetically and weakly and fall into three families, also called generations:

$$\begin{pmatrix} e \\ \nu_e \end{pmatrix} \qquad \begin{pmatrix} \mu \\ \nu_\mu \end{pmatrix} \qquad \begin{pmatrix} \tau \\ \nu_\tau \end{pmatrix}$$

The quarks interact electromagnetically, weakly and strongly. All mesons and baryons are composed of the quarks which also fall into three families:

$$\begin{pmatrix} u \\ d \end{pmatrix} \qquad \begin{pmatrix} c \\ s \end{pmatrix} \qquad \begin{pmatrix} t \\ b \end{pmatrix}$$

There are three fundamental forces which are the electromagnetic, the weak and the strong force. They are described by means of gauge theories and are transmitted by one or more boson(s) which are summarized in table 2.3.

TABLE 2.3. Fundamental Forces and their Bosons

Force	Boson	Symbol	Relative Strength
weak	intermediate vector bosons	$W^{\pm},Z^0$	$\alpha_{weak} = 1.02\ 10^{-5}$
electromagnetic	photon	γ	$\alpha_{em} = 1/137$
strong	gluons	g	$\alpha_{strong} \approx 0.1$

In order to explain the spontaneous symmetry breaking in the electroweak sector a mechanism has been introduced which gives masses to the W and Z bosons. This requires in its minimal formulation another spin-less particle: the Higgs boson. This particle can also explain the masses of all the fermions of the SM. In a minimal supersymmetric extension of the SM the single Higgs boson is replaced by a set of 5 bosons: $H^{\pm}$, h, H^0, A.

2.2.2 Physics Potential

For an estimated non-diffractive cross-section of ~70 mb [ATL94], an average of 18 minimum-bias pile-up events per bunch crossing of the LHC are expected at peak luminosity. These will mainly be QCD events and can be used to study event shape and energy flow and to test QCD and jet cross-sections over several orders of magnitude. The more interesting physics like the events where a Higgs boson is produced are several orders of magnitude ($\sim 10^{-13}$) less frequent. The following fields of interesting physics [ATL94] can be exploited at the LHC:

- **Higgs boson:**

  The most prominent issue for the LHC is the quest for the origin of the spontaneous symmetry breaking in the electroweak sector of the SM and the quest for the origin of particle masses. At the LHC the search for the Higgs boson can be conducted over the wide range of masses from $m_H \approx 80$ GeV up to $m_H \approx 1$ TeV.

- **Top quark:**

  Even at the moderate luminosity expected during the first years of LHC operation (10^{33} cm^{-2} s^{-1}), the LHC will operate as a top quark factory delivering roughly 10^7 $t\bar{t}$ pairs per year. The mass of the top quark can be measured with an accuracy of about ±2 GeV for a mass of $m_t \approx 170$ GeV.

- **B-physics:**

  A particularly rich field will be available in B-physics, the main emphasis being on the precise measurements of CP violation in the B_d^0 system and the determination of the angles in the Cabibbo-Kobayashi-Maskawa unitarity triangle. In addition, $B\overline{B}$ mixing in the B_s^0 system and rare B decays can be studied.

- **Supersymmetric particles:**

  Supersymmetric extensions of the SM predict a wide spectrum of new particles with masses and production rates such that at the LHC they could be discovered over a large fraction of the parameter space. Events with a high jet multiplicity and large missing energy make a search possible in the range of 1 to 4 TeV.

- **Physics beyond the SM:**

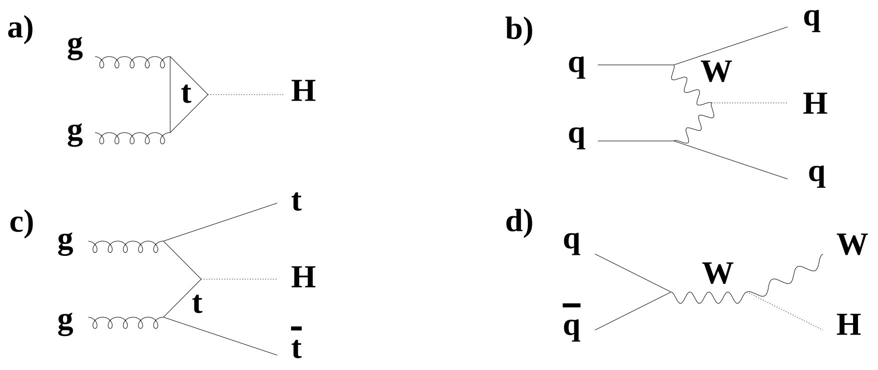
  - New **heavy gauge bosons** W’ and Z’ can be searched at the LHC for masses up to 5..6 TeV.
  - **Leptoquarks** carry both lepton and baryon quantum numbers and also couple to both. They can be produced via $qg \rightarrow lLQ$ and the final state consists in 25% of the cases in two electrons and one jet. At the LHC the sensitive area is around $m_{LQ} \approx 1$ TeV.
  - Deviations from QCD for jet cross-sections and energy spectra can be used to check the **compositeness** of quarks.
  - Possible **anomalous couplings** of gauge bosons can be studied.

2.2.3 Higgs Boson

The search for the Higgs boson is the most prominent issue for the LHC. It is used to optimize the ATLAS detector geometry and is given here as an example of the physics potential. Unitary reasons require the Higgs boson to have a mass $m_H < 1$ TeV [Vel77]. For small masses ($m_H < 1$ GeV) its existence has been excluded in several experiments and looking at different decay channels [Her90]. LEP delivers a lower limit of $m_H > 63$ GeV [Blon94] and LEP2 will increase this limit up to $m_H > 80$ GeV if it does not discover the Higgs boson [Wu87]. Principally the Higgs boson can be found at the LHC covering the mass region of 80 GeV $< m_H <$ 1 TeV.

Cross-sections for production of the Higgs boson can be calculated in the SM. These calculations use as input parameters the centre-of-mass energy $\sqrt{s}$ = 14 TeV, the top quark mass of around 170 GeV and the structure functions. The Feynman-diagrams for Higgs boson production are shown in figure 2.3.

FIGURE 2.3. Feynman-Diagrams for Higgs Boson Production



Over the mass range of 80 GeV < m_H < 800 GeV the gluon fusion (a) is dominant. qq→qqH (b) becomes dominant for bigger masses up to 1 TeV. The production cross-sections times the branching ratios for some prominent processes are shown in table 2.4.

TABLE 2.4. Cross-Section Times Branching Ratio for Some Higgs Boson Processes

Process	m_H [GeV]	$\sigma \cdot BR$ [fb]
$pp \rightarrow H \rightarrow \gamma\gamma$	100	44
$pp \rightarrow H \rightarrow ZZ^* \rightarrow 4l^\pm$	130	3
$pp \rightarrow H \rightarrow WW \rightarrow l\nu jj$	1000	17

Search strategies for the Higgs boson depend on its mass and several have to be combined to cover the full mass range:

- **$H \rightarrow b\bar{b}$:**

  With a Higgs boson mass below the threshold for decays into a pair of vector bosons this decay mode is essentially 100%. The signature used will be a lepton from one b-quark and a b-quark jet from the other, possibly used with the associated production (figure 2.3 (c) and (d)). This channel is sensitive at 80 GeV < m_H < 100 GeV.

- **$H \rightarrow \gamma\gamma$:**

  This is a sensitive channel for 90 GeV < m_H < 150 GeV and requires an excellent electromagnetic calorimeter and identification of photons against a huge background from jets misidentified as photons.

- **$H \rightarrow ZZ^{(*)} \rightarrow 4l^\pm$:**

  For masses between 130 GeV < m_H < $2m_Z$ one of the Z bosons is virtual and the Higgs boson is rather narrow with a big background from boson pair production. For masses

$m_H > 2m_Z$ both bosons are real and the signal is rather clean.

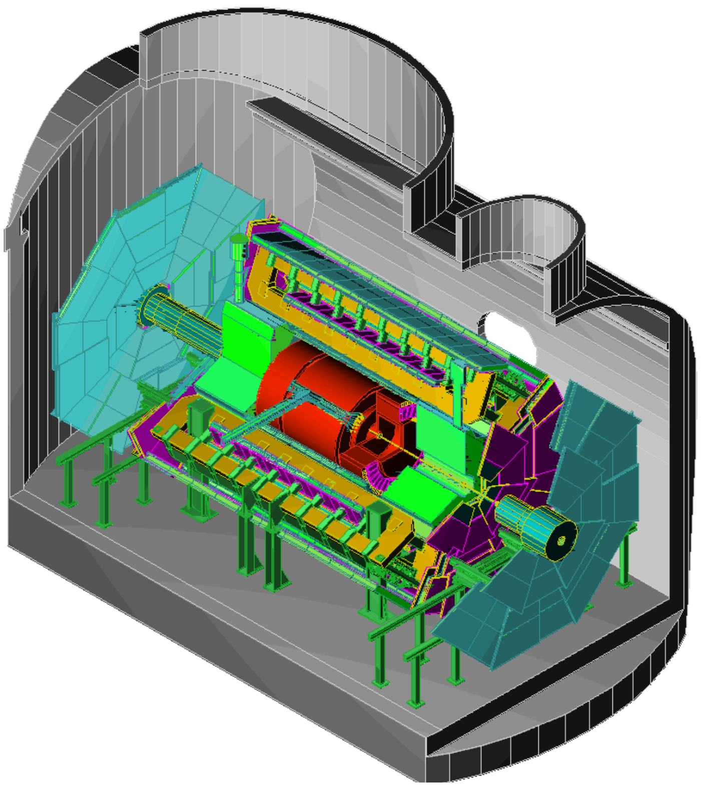
- **$H \rightarrow WW, ZZ \rightarrow l^{\pm}\nu jj, 2l^{\pm}jj$:**

  This signature is important in the mass range up to $m_H \approx 1$ TeV and uses two jets for identification.

2.3 ATLAS Detector

ATLAS [ATL94] is a proposed pp experiment at the LHC. It wants to exploit the full discovery potential of the LHC. It will operate at high luminosity but also at the lower luminosity in the initial phase of the collider. It will try to use as many physics signatures as possible, like electron, photon, muon and jet detection but also missing transverse energy and jets from b quarks. At lower luminosity it will in addition try to identify τ leptons and heavy quark flavors and resolve their secondary vertices.

FIGURE 2.4. Three-Dimensional View of the ATLAS Detector



2.3.1 Overview

The basic considerations for the design of the ATLAS detector are the following:

- a very good electromagnetic calorimetry that will be able to identify and measure electrons and photons. Hermetic jet measurement and missing transverse energy are also required;
- efficient tracking at high luminosity that will provide lepton momenta and b-tagging. The inner detector will also contribute to the electron and photon identification and

allow detection of secondary vertices at lower luminosity;

- a stand-alone muon spectrometer that will identify and measure muons.

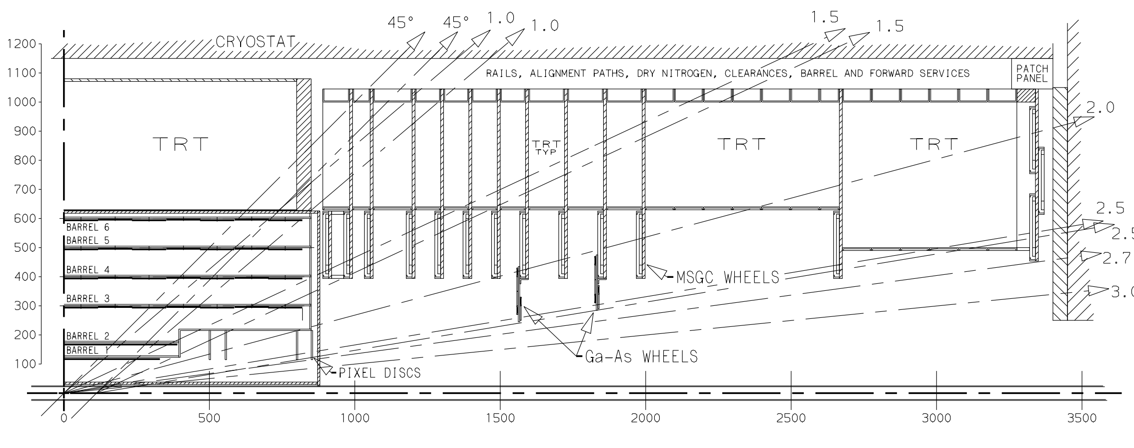
In addition, a large acceptance of the detector over the η range and a triggering capability at low transverse momenta are required.

The inner detector of ATLAS is in a solenoid magnetic field of 2 T and performs pattern recognition and particle identification. It has a cylindrical structure and covers the area between the beam pipe and a radius of 1.15 m and has a length of 6.8 m. The calorimetry is based on liquid argon calorimeters and a novel type scintillator tile calorimeter which cover the radial space up to a radius of 2.25 m for electromagnetic and up to a radius of 4.25 m for hadronic calorimetry. The muon spectrometer is based on a superconducting air-core toroidal magnet system and uses different types of chambers up to radii of 19.5 m to detect and trigger on muon tracks. The ATLAS detector has a multi-level trigger system to accommodate the high data rates and to select interesting physics.

2.3.2 Inner Detector

The ATLAS inner detector occupies the cylindrical cavity defined by the boundaries of the cryostats for the electromagnetic calorimeters at a radius of 1.15 m and on the inner radius by the boundary of the beam pipe with a diameter of about 5 cm. The inner detector is in an axial central field of 2 T provided by a superconducting solenoid coil which is integrated in the cryostat for the electromagnetic calorimeter. The inner detector combines high-granularity detectors at inner radii and continuous tracking elements at larger radii.

FIGURE 2.5. Layout of the ATLAS Inner Detector



Mechanically, the inner detector is divided into three units: a barrel unit extending over 80 cm along the beam axis and two identical forward units covering the rest of the cylindrical space. In the barrel region the detectors are arranged on concentric cylinders around the beam axis. All the forward tracking elements are located in planes perpendicular to the beam axis. Their sensitive parts are oriented in radial or almost radial direction. The different detector technologies are listed briefly below:

- The **pixel detector** is based on silicon detectors which contain an array of pixel diodes and also the bus lines for control and readout. The readout electronics chip is directly bump-bonded onto the silicon detector. Two layers of the pixel detector at small radii provide two-dimensional spatial information. An additional layer might be installed at the initial phase of lower luminosity running as close as practical around the beam pipe. This will enhance the secondary vertex measurements.

- The **semiconductor tracker** (SCT) uses strip detectors with fine granularity in the φ direction. Silicon is foreseen in the barrel region and GaAs substrates in the forward region where the radiation doses are higher. Each layer of the SCT consists of two detectors glued back-to-back to measure alternating combinations of φ and u or φ and v in the barrel region and u and v in the forward region.
- The **micro strip gas counters** (MSGC)[1] have been developed as strip detectors for large areas. The detectors operate as drift chambers with finely segmented metal anodes etched on a glass substrate. Small drift times and high-precision position measurement can be achieved. Each disc of the MSGC in the forward region measures the three coordinates φ, u and v.
- The **transition radiation tracker** (TRT) is based on straw tubes of 4 mm diameter. These tubes are made from 60 μm thin kapton walls with a 50 μm gold-plated beryllium wire along the straw axis. The straws are interleaved with polyethylene radiators to produce and detect X-ray emission from very relativistic particles. A high p_T charged particle will transverse 64 layers of the TRT and give at least 36 tracking points providing a good pattern recognition. The TRT will also enhance the electron identification.

TABLE 2.5. Parameters of the Inner Detector

Detector	Radius [cm]	Length [cm]	Elements	η Coverage
Pixel	11.5	±35	50μm×300μm	2.5
	16.5	±45		
	11.5..21.3	50,55,80,85		
SCT (silicon)	30,40,50,60	82	75μm×12cm	±1.4
SCT (GaAs)	20..35	155.7	50μm×7.6cm	2.0..2.5
	29..44	182.5		
MSGC	44..60	90,..,265.6	200μm×16cm	1.4..2.5
	50..96	336.0		
TRT	63..107	±80	4mm diameter×80cm	±2.5
	64..103	80..265	39cm	
	50..103	267..327	53cm	

The combination of high-precision tracking detectors and continuous tracking elements provides a very robust pattern recognition and full tracking coverage over the rapidity

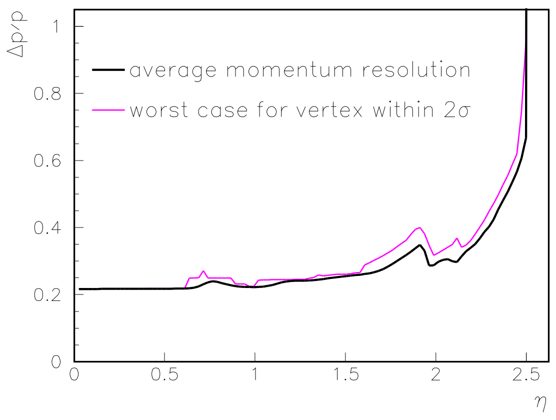
1. In September 1995 the ATLAS collaboration decided not to build this subdetector.

range of $|\eta| < 2.5$. The resulting momentum resolution of the combined detectors is shown in figure 2.6. The vertex resolution can be parametrized (σ in μm and p_T in GeV):

$$\sigma_{R\varphi} = 13 \oplus \frac{62}{p_T \cdot \sqrt{\sin\vartheta}}$$

$$\sigma_z = 39 \oplus \frac{90}{p_T \cdot \sqrt{\sin\vartheta}^3}$$

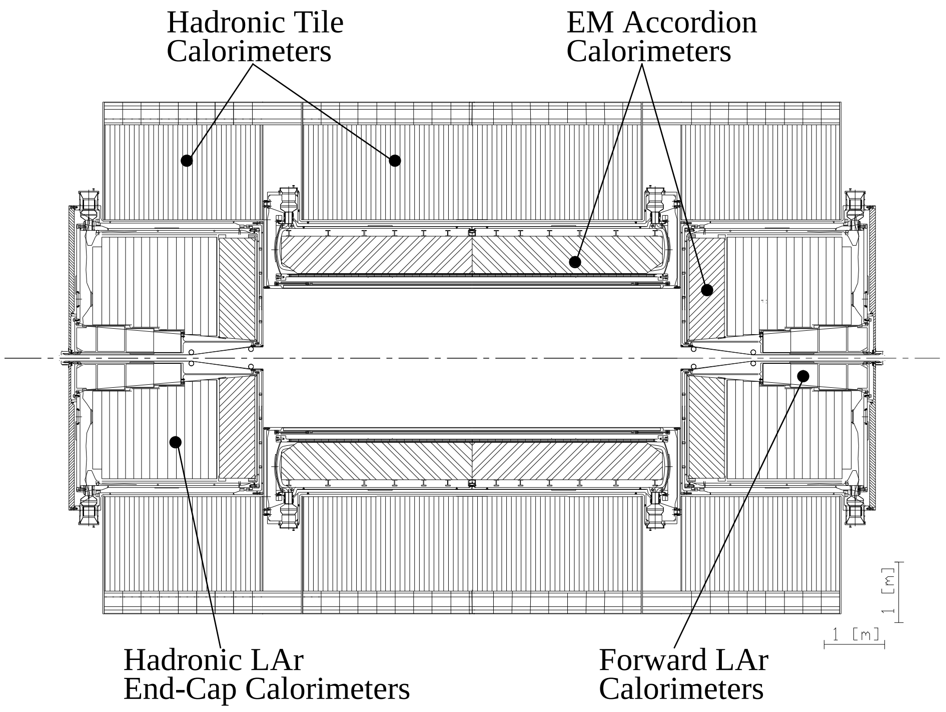
FIGURE 2.6. Momentum Resolution in the Inner Detector



2.3.3 Calorimetry

The ATLAS calorimetry consists of an inner barrel cylinder and end-caps using liquid argon (LAr) technology and an outer barrel cylinder and two extended barrel sections using a novel type scintillator tile technology.

FIGURE 2.7. Layout of the ATLAS Calorimeter System



Hadronic Tile Calorimeters
EM Accordion Calorimeters
Hadronic LAr End-Cap Calorimeters
Forward LAr Calorimeters
1 [m]
1 [m]

The barrel electromagnetic calorimeter is made of two half-barrels and uses lead absorbers in LAr implemented in an “accordion” geometry. There are 1024 absorber plates arranged around the beam axis in the φ direction. Between two absorbers there are two gaps with LAr of 1.94 mm and with a readout electrode of 300 μm thickness. The electrode, held in place by a light honeycomb structure, acts as a blocking capacitor for the high voltage and contains segmented copper strips for the charge collection. The first compartment can be read out with very fine granularity in η to act as an integrated preshower for photon identification. In addition, there will be a separate presampler device in front of the barrel calorimeter to preserve the energy and direction resolution of particles passing the cryostat and coil material. The minimal total thickness of the electromagnetic barrel calorimeter is 26 X_0 (at $\eta = 0$).

The LAr end-cap calorimeter consists of two identical end-caps divided into eight wedge-shaped modules on two coaxial wheels. The “accordion” shaped absorber plates are arranged in a way that cracks in φ are completely avoided. The total thickness of this calorimeter is 28 X_0 for all rapidity values. The first compartment is again finely segmented in η to act as a preshower. No external presampler is needed.

The end-cap hadronic calorimeter is made of two wheels per end-cap which are assembled from 32 sector modules. LAr technology is used with copper absorber plates and electrodes. The forward calorimetry which is integrated with the end-cap electromagnetic and hadronic calorimeters in the same cryostat uses LAr technology in a tube-like structure to operate in the high radiation region. A very narrow LAr gap between an absorber rod and a tube will be used, the absorber being copper for the electromagnetic compartment and tungsten alloy for the two hadronic compartments.

The hadronic scintillator tile calorimeter is based on a sampling technology with iron absorbers and plastic scintillator plates (tiles). The tiles are placed perpendicular to the beam axis and read out by wavelength shifting fibres. The tile calorimeter consists of three cylindrical structures with each of them sub-divided into 64 independent azimuthal modules. Readout cells are defined by grouping together a set of fibres onto one photomultiplier.

The parameters of segmentation and rapidity coverage can be found in table 2.6. The performance of the ATLAS calorimetry has been extensively simulated and confirmed by test-beam results. The energy resolution of the electromagnetic calorimeters can be described by

$$\frac{\sigma}{E} = \frac{0.1}{\sqrt{E\,[\mathrm{GeV}]}} + \frac{400\mathrm{MeV}}{E} + 0.003$$

and the clusters of electromagnetic showers are to 95% contained in windows of 3×7 cells. The energy resolution of the barrel hadronic calorimeter for charged pions is

$$\frac{\sigma}{E} = \frac{0.45}{\sqrt{E\,[\mathrm{GeV}]}} + 0.015$$

TABLE 2.6. Parameters of the Calorimeter System

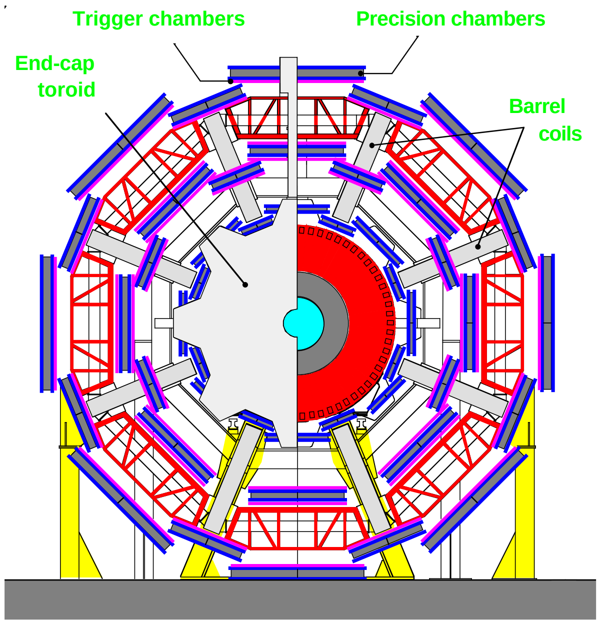
Type	Region	Technology (Absorber)	Elements		η Coverage
			Depth	Segment[a]	
electro-magnetic	barrel	LAr accordion (lead)	4	0.025×0.100 (presampler) 0.025×0.025	±1.4
	end-cap	LAr accordion (lead)	3	0.003×0.100 (presampler) 0.025×0.025	1.4..3.2
	forward	LAr tube (copper)	3	~0.15×0.15	3.1..4.9
hadronic	barrel	Scintillator tile (iron)	3	0.1×0.1	±1.6
	end-cap	LAr accordion (iron)	4	0.1×0.1	1.5..3.2
	forward	LAr tube (tungsten)	3	~0.15×0.15	1.5..3.2

a. The segmentation varies in the longitudinal compartments, only some values are shown.

2.3.4 Muon Spectrometer

The ATLAS muon spectrometer is based on a superconducting air-core toroid magnet system which consists of a 26 m long barrel part with an inner bore of 9.4 m and an outer radius of 19.5 m and two end-caps with lengths of 5.6 m and inner bores of 1.26 m. Each toroid consists of eight flat coils symmetrically arranged around the beam axis with the end-caps rotated with respect to the barrel so that the coils interleave.

FIGURE 2.8. Transverse View of the ATLAS Muon Spetcrometer



Muon chamber planes are attached to the toroids to measure the trajectories of muons. In the barrel the layout consists of three layers of chambers and in the end-caps the chambers are placed on the front and back faces of the cryostats. A third layer is fixed on the cavern wall. Two types of chambers are used for the high-precision measurements:

- The **monitored drift tube** chambers (MDT) consist of two multi-layers of four planes of pressurized thin-wall aluminium tubes with a diameter of 30 mm. They are arranged at distances of 150 to 350 mm and are used for a very large part of the rapidity range.
- The **cathode strip chambers** (CSC) are multi-wire proportional chambers with a symmetric cell in which the anode-cathode distance equals the anode wire spacing, both typically 2.5 mm. The high precision position is measured by determining the centre-of-gravity of the induced charge on the finely segmented cathode strips made from etched copper on glass fibre laminates. These chambers can be operated in highest rate environments at large values of η using an appropriate segmentation.

The high-precision measurements are complemented with chambers for triggering. There are also two types used for this:

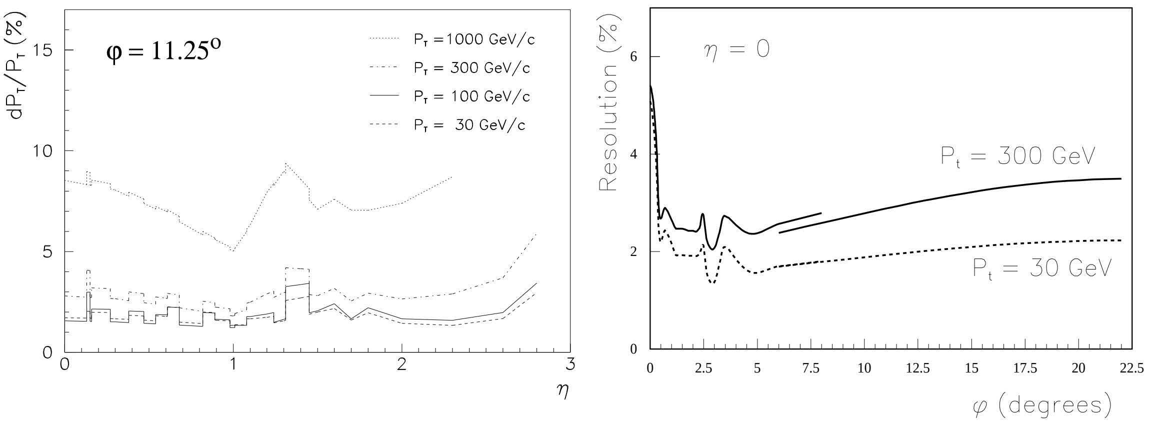
- The **resistive plate chambers** (RPC) are gaseous parallel plate detectors with two bakelite plates coated with layers of graphite paint providing the electric field and external pick-up strips on plastic material for the signal. A set of two orthogonal strips is used to provide two-dimensional information with good spatial resolution.
- The **thin gap chambers** (TGC) are used in the forward region. They are wire chambers operated in saturated mode. Two graphite cathodes with a distance of 3.2 mm are sandwiched with 50 μm diameter anode wires with a pitch of 2 mm. Capacitive readout on pads and strips gives the spatial resolution required and the time resolution is less than 5 ns.

TABLE 2.7. Muon Chambers used in the Muon Spectrometer

Region		Inner Layer	Middle Layer	Outer Layer	η Coverage
Barrel	Radius [m]	~4.5	~7	~10	±1.05
	Precision	MDT	MDT	MDT	
	Trigger	-	RPC	RPC	
Transition	z [m]	~7	~10	~13	1.05..1.4
	Precision	MDT	MDT	MDT	
	Trigger	TGC	TGC	TGC	
End-Cap	z [m]	~7	~14	~21	1.4..3.0
	Precision	MDT/CSC	MDT/CSC	MDT/CSC	
	Trigger	TGC	TGC	TGC	

The momentum resolution of the combined muon spectrometer relies heavily on the ability to align the muon chambers precisely over big distances. The resolution has been calculated taking into account the three-dimensional magnetic field, multiple scattering in the various materials and errors in the alignment. The resolution as a function of η and φ (w.r.t to the coil plane) is shown in figure 2.9.

FIGURE 2.9. Momentum Resolution of the ATLAS Muon Spectrometer



2.4 ATLAS Trigger and Data Acquisition System

At the LHC every 25 ns an average of 18 minimum bias events will overlap. With some tens of 10^6 channels of the tracking detectors, some 10^5 channels of the calorimeters and some 10^6 channels of the muon spectrometer the ATLAS detector will produce an immense data flow. This flow needs to be reduced so that it finally can be read out without creating unreasonable deadtime and can be written to a mass storage system at a reasonable rate. A highly sophisticated system of data reduction and selection of the interesting events without losing new and unforeseen physics is required for this task. The ATLAS trigger and data acquisition system is based on a three-level trigger system with a high degree of parallelism at every stage.

2.4.1 Trigger Strategy

The level-1 (LVL1) trigger accepts data at the full LHC rate of 40 MHz and uses reduced-granularity data from the calorimeters and muon spectrometer to apply an event selection based on the highly significant signatures of electron, photon, muon, jet and missing transverse energy. Simulations [ATL94] show that inclusive signatures lead to prompt trigger rates of a few tens of kHz (table 2.8). Including a safety margin it can be expected that the LVL1 trigger accepts events at a maximum rate of 100 kHz.

The LVL1 trigger not only reduces the incoming rate by a factor of about 400 but also identifies for each accepted event regions of interest (RoI). These are areas of the acceptance space where something interesting has been detected. Lower thresholds than for the LVL1 trigger are used to identify these areas which are then passed to the level-2 (LVL2) trigger to guide further event selection.

The LVL2 trigger uses full-granularity and full-precision data but acts only on parts of the data identified by the RoIs. Higher precision of the calorimeter and muon data allows to refine the decision taken at LVL1. Also data from the tracking detectors will be included in order to provide together with the calorimeter and muon information a reduction factor of 100. Studies [ATL94] have shown that this mechanism will only need a few percent of the data to deliver an output rate of accepted events of 1 kHz.

TABLE 2.8. Some Simulated Trigger Rates at LVL1 ($L = 10^{34}$ cm^{-2}s^{-1})

Trigger Requirement	LVL1 Rate [kHz]
1 isolated em. cluster, $E_T > 30$ GeV	20
1μ, $p_T > 20$ GeV	4
2 isolated em. clusters, $E_T > 20$ GeV	4
2μ, $p_T > 6$ GeV	1
1 jet, $p_T > 150$ GeV	3

The level-3 (LVL3) trigger will act on the full event data and reduce the data by another factor 10. While LVL1 and LVL2 have only been acting on inclusive signatures, at LVL3 detailed reconstruction can be performed though it is desirable to keep the thresholds lower than the ones used in the analysis to avoid trigger bias and to facilitate background studies.

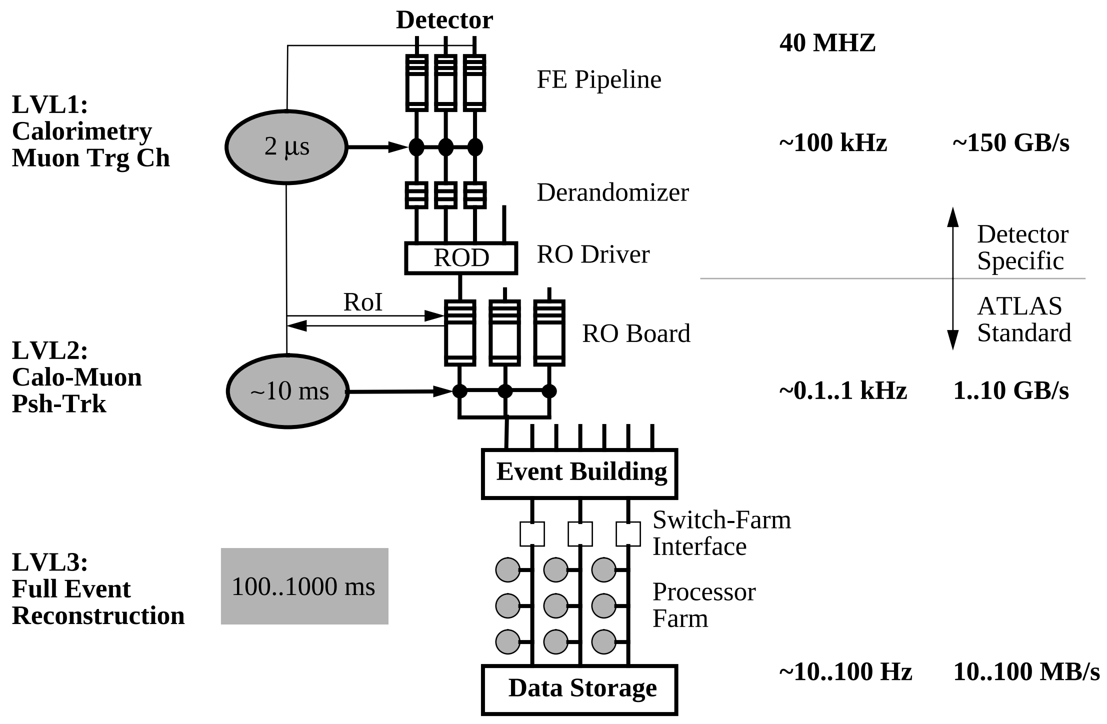
At the initial lower luminosities of the LHC B-physics will be of much interest and a different trigger strategy will have to be considered. A low-p_T muon trigger will be used to give prompt trigger rates of up to 10 kHz. On the LVL2 information of the precise tracking detectors, the TRT and calorimeter will be investigated sequentially to give the required rate of about 1 kHz. At LVL3 full event reconstruction will be used to select interesting B-physics.

2.4.2 Trigger and Data Acquisition Architecture

During the LVL1 processing which is done in a completely independent unit all data of the whole detector are stored in pipeline memories. These can be analogue or digital if the signals are digitized at bunch-crossing frequency. The pipelines must be long enough to keep all the signals for the LVL1 latency which will be fixed at about 2 μs corresponding to about 80 bunch-crossings.

When an event is accepted by the LVL1 trigger the data from all the pipeline memories are read out to off-detector readout boards [Far95]. Since the LVL1 trigger decision is synchronized and arrives a fixed number of bunch-crossings after the interaction the correct data in the pipeline can be identified and sent to the derandomizer. This is an intermittent storage which accepts the digitized and possibly zero-suppressed data at the random LVL1-accept frequency and adopts the output rate to the front-end link. Over this link which will be based on optical fibres, the data are sent to the readout driver (ROD) which receives them and checks their consistency with the timing information it gets from the timing, trigger and control system (TTC). Sparsification and data compression are possible. It multiplexes several data flows from different pipeline memories onto one readout link.

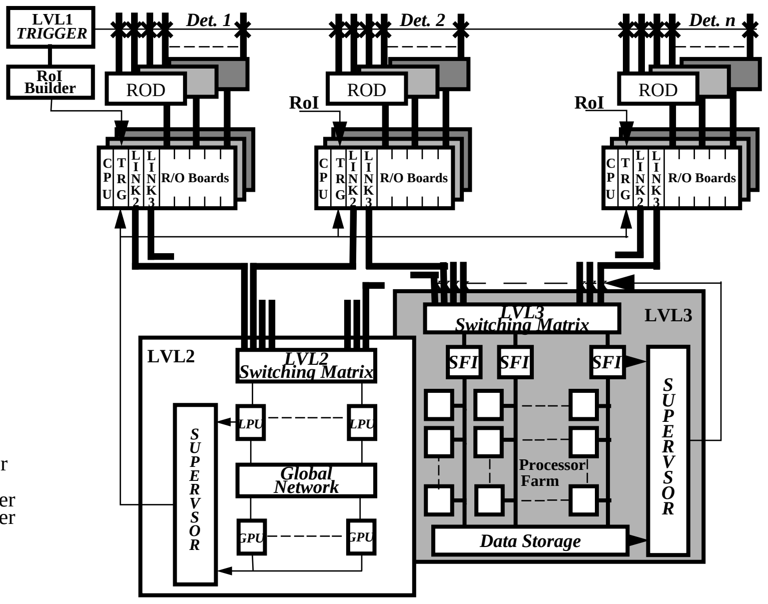
FIGURE 2.10. Three-Level Trigger Architecture of ATLAS



While up to this point the technology of the front-end electronics can be detector specific, from the readout link onwards a uniform technology will be used to facilitate construction and control of the ATLAS data acquisition system. The readout link will be based on optical fibres with a relatively uniform data flow of about 1 Gb/s. The data flow will also be arranged in a way that it matches the RoI segmentation and one RoI will be spread over a minimal number of readout boards/buffers (ROB). These receive the data at the frequency of the LVL1-accept and store them during LVL2 processing and until the readout to the LVL3 trigger is completed. The ROB will contain a multi-port memory with at least two fast access ports for input and output. For convenience several ROBs will be grouped together using shared links to the LVL2 and LVL3 trigger. These groups would in today's technology be based on (VME) crates. The readout crates would, in addition to the ROBs, house a trigger interface for receiving the RoI information from LVL1 trigger and the LVL2 decision. The RoI information will start transmission of the corresponding data from the ROB to the LVL2 link. The LVL2 decision would either release the corresponding buffer space or start transmission of the full event data to the LVL3 link.

The LVL2 trigger can be regarded as an independent unit which receives the data of the RoIs from the ROBs and sends back a decision. The LVL3 trigger includes an event building stage which assembles full events in parallel and sends them to farms of processors running the LVL3 algorithm. The selected and compressed events will then be written to a mass storage system for off-line analysis.

FIGURE 2.11. Trigger/DAQ Architecture of ATLAS



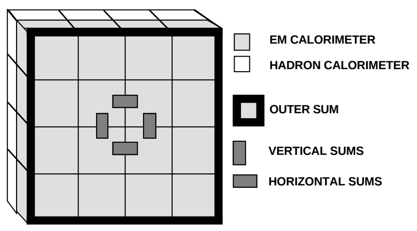
CPU = DAQ Supervisor
TRG = Trigger Interface
RoI = Region of Interest Builder
ROD = Readout Driver
LINK2 = Link to Level 2 Trigger
LINK3 = Link to Level 3 Trigger
LPU = Local Processing Unit
GPU = Global Processing Unit
SFI = Switch Farm Interface

2.4.3 Level-1 Trigger

The LVL1 calorimeter trigger uses reduced-granularity cells of $\Delta\eta\times\Delta\varphi = 0.1\times0.1$ with single depth compartments in electromagnetic and hadronic calorimeters. Purpose-built processors receive the signals from the trigger cells at full LHC frequency and process them in a pipelined fashion. They will digitize the signals with a smaller dynamic range, calibrate them with the help of look-up tables and calculate sums for the electromagnetic cluster finding, the jet finding and missing transverse energy measurement. Since these algorithms are based on overlapping windows each processing element also requires information from its neighbours which is achieved by fanning out the input data.

The electromagnetic cluster finding is based on a 4×4 trigger cell window and applies thresholds on the sum of E_T in pairs of adjacent electromagnetic cells in the 2×2 area in the centre, the sum of E_T around this area for isolation and the sum of E_T in the 16 hadronic cells at the back of the window. This is schematically shown in figure 2.12.

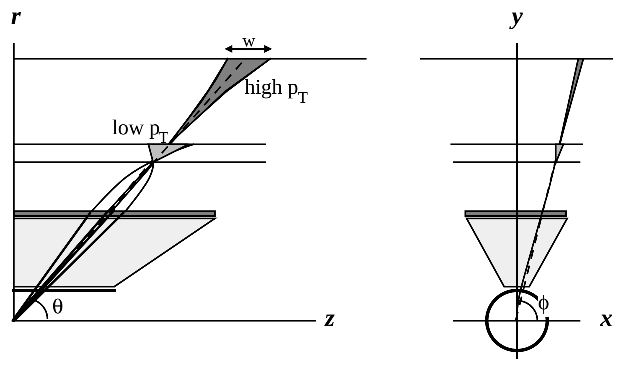
FIGURE 2.12. LVL1 Electromagnetic Cluster Finding in the Calorimeters



The jet finding is based on windows of 8×8 trigger cells above a certain E_T threshold. The missing transverse energy trigger uses the sum over all cells above an E_T threshold. The processing elements do, in addition to the trigger processing, define the RoIs using algorithms similar to the electromagnetic cluster and jet finding but with lower thresholds.

The muon trigger chambers are used for the LVL1 trigger processing. Four layers of chambers are located at the middle of the barrel toroid, arranged in two groups of two chambers separated by about 40 cm. For the low p_T trigger (~6 GeV) a coincidence between these groups is made in a road defining the p_T sensitivity (see figure 2.13). Additional chambers at larger radii are used for high p_T triggering (~20 GeV). In the end-cap a similar algorithm is used. In both cases the muon trigger tracks will be identified by a programmable coincidence matrix ASIC. The same information is also used to generate the RoI information.

FIGURE 2.13. LVL1 Muon Trigger Algorithm



The central trigger logic combines the information for the calorimeter and muon trigger processors and makes an overall decision. The latency of this decision will be constant and including all cable delays of about 2 µs. It will be distributed, together with the LHC clock and other control signals to the front-end electronics of all detectors. A timing, trigger and control system (TTC) based on opto-electrical fan-outs will take care of this task.

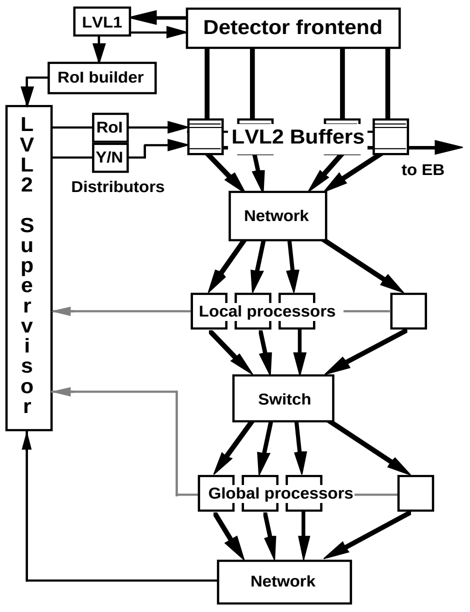
2.4.4 Level-2 Trigger

The LVL2 trigger is based on the use of RoI information delivered by the LVL1 trigger. It only has to access and process a small fraction of the total data and proceeds in three steps:

1. The feature extraction reduces the data from one subdetector in one RoI into a more physical quantity like track or cluster parameters.
2. The features from several subdetectors in one RoI are combined to form an object, which can be a particle candidate.
3. All objects of all RoIs are combined to a global decision.

A possible architecture is shown in figure 2.14. Data from each RoI are extracted from the ROB (which here acts as a LVL2 buffer) and routed via a network to a set of local processors where the feature extraction is done. The features are then sent through a network to a set of global processors where the object forming and the global decision taking is done. The final decision is sent to a supervisor which keeps track of the status of all the processors and assigns them to the different tasks. All processors are assumed to be fully programmable high-level processors. For initial low luminosity running the same LVL2 trigger system will be used to run a set of selections in different subdetectors in a sequential fashion. The overall event selection is made in the global processors using information from all detectors.

FIGURE 2.14. A possible LVL2 Trigger Architecture



2.4.5 Event Building and Level-3 Trigger

When an event is accepted by the LVL2 trigger the decision is distributed to the readout crates which will read the corresponding data from all its ROBs and send them to the LVL3 link. The data will be kept buffered in the ROBs as long as the event building system is busy. The data will pass through an event building system based on a switching element and full events will be assembled in parallel at the switch-farm-interfaces on the receiving side. These will buffer the full events during the LVL3 processing and provide feedback to the data flow management system controlling the data flow through the event building system.

LVL3 processing units are running the LVL3 trigger based on full event reconstruction. They will most likely be organized in the form of farms, each farm having one or more processing units, an input and an output unit. The input unit (switch-farm-interface) receives the full events from the event building system and buffers them in memory. The output unit receives the LVL3-accepted data and sends them to the permanent storage system for off-line analysis.

Estimations of the data flow in the individual subdetectors of ATLAS and the number of LVL3 links needed is summarized in table 2.9 [Map95a]. All values are based on average calculations and are still subject to frequent changes.

TABLE 2.9. Data Flow of the ATLAS Detector

Subdetector	# Channels [10^6]	Occupancy [%]	Event Size [kByte]	Throughput [MB/s]	# LVL3 Links
Pixel	140	0.01	50	50	5
SCT	2.9	1	100	100	10
GaAs	0.8	1	30	30	3
MSGC	1.5	2	100	100	10
TRT	0.42	12..33	500	500	50
Calo & PS	0.23	5..15	350	350	35
Muon	1.3	2..8	200	200	20
total	147.15	-	1330	1330	133

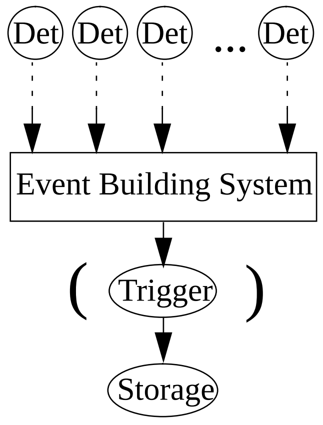
From the estimated data flow it can be seen that the event building system in the ATLAS detector will have about 100 data sources which provide a full event with a total amount of data of about 1 MByte, leading to an average event fragment size of about 10 kByte per source. Assuming a LVL2-accept rate of 1 kHz the total throughput of the event building system can be estimated to be about 1 GB/s, requiring a minimal bandwidth of 10 MB/s per source. About 100 destinations will receive the full events at a rate of about 10 Hz which leaves them 100 ms per event. With processing times of up to 1,000 ms per event this could be handled with a reasonable number of processing units in the LVL3 farm and a single high-speed bus system for data transfer as shown in [Mor94].

3.0 Event Building Systems

Experiments in high energy physics produce data in different places over the whole detector. Every subdetector which might be divided into several modules, has several buffers into which data is written. To store the data and to make them available for off-line analysis, the data belonging to the same physical event has to be assembled in one place. Also, some initial trigger processes are based on only part of the data or on data coming from a coarser granularity of the subdetectors. More sophisticated algorithms based on the complete data can provide a better selection and reduce the amount of data to be stored for off-line analysis.

Event building is the part of a data acquisition system where the different data scattered over the buffers of the subdetectors but belonging to the same physical event are assembled and the complete data is transferred to the next level of trigger processing and/or data storage (figure 3.1). The scattered data is often referred to as "event fragment data" while the complete data is called "event data".

FIGURE 3.1. Event Building in a Data Acquistion System



Over the last 15 years the use of VLSI front-end circuitry has increased the allowable trigger rate by at least three orders of magnitude and the performance of trigger processors and the density of data storage have improved at an equivalent rate. On the contrary, the speed of standard bus systems used for event building has only improved by a factor ten in the same time period [Bar90]. Event building has become a bottleneck in data acquisition systems for modern experiments in high energy physics.

Future experiments at the LHC will need event building systems with an unprecedented throughput and high level of interconnectivity (see section 2.4.5). Standard bus systems do not provide adequate performance and parallel event building based on high speed interconnect and switching elements has to be envisaged. Event building thus will evolve from single-bus data collectors to multiple parallel channels feeding farms of processors. This will allow a much higher total throughput of data if the system is free of inherent bottlenecks. Such a system might have internal buffering and a sophisticated data flow management to make sure that no data gets lost, that all event data are assembled correctly and the capacity of the system is optimally used.

Commercial systems should play an important role in event building which is similar to some problems in telecommunications and high performance computing. Event building, however, is in principle different from requirements in these fields as the data flow is uni-directional, apart from control flow, and the data pattern is relatively uniform compared to the burstiness of telephone calls but it might be possible to use these technologies cost-effectively.

3.1 Definition

An event building system is an element in the data flow of a physics experiment with different input and output data streams, combining them so that it fulfills the two following requirements [Spi94b]:

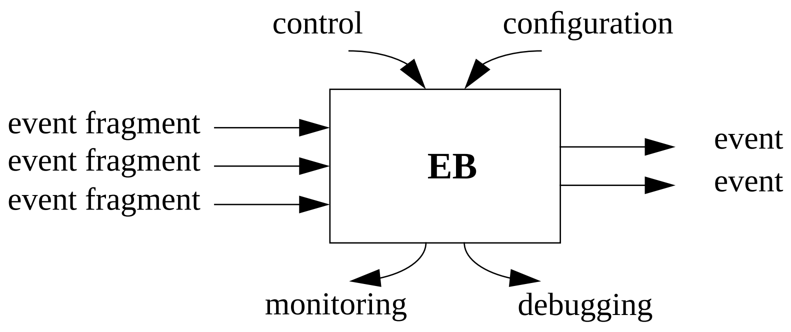
- The event building system is a connection between sources and destinations and **transfers** data from the first to the latter.
- The data from the different sources are **merged** in a way that all fragments belonging to the same physical event are sent to the same data destination.

Furthermore, an event building system has to be an integral part of the overall data acquisition system and has to provide the following functionalities:

- control: start and stop the event building system, report error messages,
- configuration: configure the event building system in terms of number of sources and destinations and their various parameters,
- monitoring: report performance statistics;
- debugging: trace individual data packets through the event building system.

The event building system can be described as a black box model as shown in figure 3.2

FIGURE 3.2. Functionality of an Event Building System



3.1.1 Data model

As the event building system is acting on data it is important to give a model of these data. Data are logically grouped in forms of event fragments and have three layers:

- **Actual Data:**

  This is the accumulation of bits and bytes in memory. They have a well defined format and their size is known.

- **Description:**

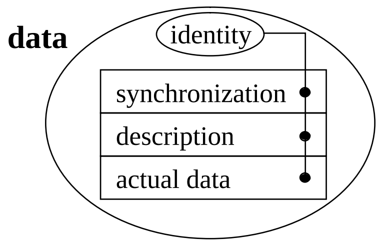
  This is a description of the data in terms of memory location, size, format and quality. The description can be part of the actual data as well.

- **Synchronization and Timing:**

  This is the timing aspect of data: they can be available, being used by a process or be discarded. Usually this aspect is not an integral part of the data but logically belongs to them. Often signals are used to represent a state change in the “life” of data.

Furthermore, every piece of data has an identity which might consist of several fields and usually is part of the actual data and the description but it might also be in the synchronization. This data model is shown in figure 3.3.

FIGURE 3.3. Data Model



Internal to the event building system all three layers of data have to be handled. This might be done in different ways with different hardware for each of the layers, e.g. synchronization via interrupts on a bus system, description via Ethernet and actual data via high speed data link.

3.1.2 Elements

Inside the black box the event building system must have elements to input data, to transfer data, to output data and to control the data flow. It has sources, an interconnecting network, destinations and a data flow management:

- **Source:**

  The source is the interface to the input side of the event building system. It receives the data (event fragments) from the up-stream data flow, processes them and sends them to the interconnecting network.

- **Interconnecting Network:**

  The interconnecting network connects all sources and all destinations and transfers data and control information, if necessary, between these.

- **Destination:**

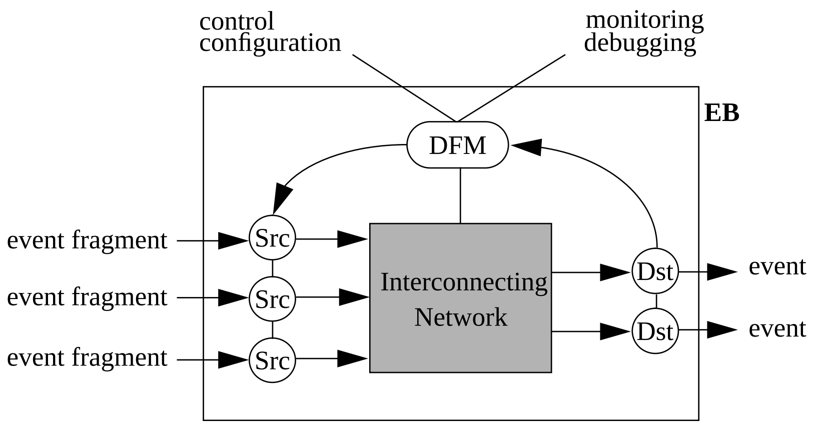
  The destination is the interface to the output side of the event building system. It receives the data from the interconnecting network, merges them and outputs full events to the down-stream data flow.

- **Data Flow Management (DFM):**

  The data flow management represents the control aspect of the event building system. It controls the transfer of data, the handling of events and the configuration of the event building system. It detects and handles errors and communicates with the control system of the overall data acquisition system.

Looking inside the black box model one thus gets the picture shown in figure 3.4.

FIGURE 3.4. Components of an Event Building System

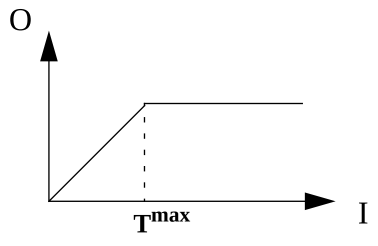


3.1.3 Performance Indices

The most important performance index of an event building system is the maximum throughput T^{max} of data that can be digested by it. The input data rate I is defined as the accumulated data flow of event fragments into the sources, and similarly the output data rate O as the accumulated data flow of events from the data destinations. If no data get lost and the re-formatting of the events does not change the total size of it, input and output data rate will be equal (conservation of data flow):

$$O = I \qquad \text{(Eq. 3.1)}$$

FIGURE 3.5. Maximum Throughput of an Event Building System



Nevertheless, due to the limited capacities of an event building system, increasing the input rate I will lead to an increased output rate only until the maximum throughput T^{max} is reached, after which the output rate cannot further be increased. Ideally this looks like in figure 3.5. In reality this curve might be smoother due to the statistical fluctuations of the data flow when approaching the maximum. Related to the input rate is the input frequency which is denoted as f and f^{max} is the maximum input frequency, equal to T^{max} divided by the event size.

The maximum throughput T^{max} can be used to define the efficiency ε of the event building system as the ratio of the achieved maximum throughput to the ideally possible maximum throughput which equals the aggregate bandwidth:

$$T^{max} = \varepsilon \cdot T^{max}_{ideal} = \varepsilon \cdot N_{Dst} \cdot speed \qquad \text{(Eq. 3.2)}$$

where N_{Dst} is the number of destinations and *speed* the link speed.

Other important performance indices are the latency L and the buffer occupancy B. The latency L is the time between the first event fragment arrives in the source and the full event is available in the destination. The fragment latency L_{Frg} is the time between an event fragment arrives at the source and it is released from it. The following relation exists between the two:

$$L(evt) = \max_{src} L_{Frg}(evt, src) \qquad \text{(Eq. 3.3)}$$

The buffer occupancy B is the amount of data that needs to be stored in the source due to the limited capacities of the event building system. When the input data flow underlies statistical fluctuations buffer occupancy is not equal to zero even if the input data rate is less than the maximum throughput. If the input data rate is greater than the maximum throughput the buffer occupancy will increase with time until it reaches the physical limits of the buffer. Buffer occupancy can be measured in number of event fragments as well as in the buffer space occupied.

Other important issues are also the fault tolerance and reliability of the event building system, which is a way to characterize how easily it can handle failures in hardware and software and how long it takes to recover from such a failure. Cost is important and the cost of an event building system should be seen in relation to the overall cost of the whole data acquisition system.

3.2 Interconnecting Networks and Architectures

An interconnecting network is a set of busses, switches and/or data links that permit data transfer between two or more devices. In an event building system this will mainly be used to transfer data from sources to destinations but it could also be used for the control flow if necessary. Looking at interconnecting networks used in existing event building systems in high energy physics experiments and interconnecting networks used in computer and telecommunication networks several architectures can be foreseen [Bar90].

3.2.1 Shared Bus Architectures

In a shared bus architecture all sources and destinations are connected to the same time-shared single bus system (figure 3.6). This is the most common architecture for event building and several standard bus systems are available and actually used in existing event building systems. A single shared bus architecture allows simple control algorithms and the same bus can be used for data and control traffic. It usually can be extended to different numbers of sources and destinations but does not increase in total throughput beyond the maximum throughput of the medium. It might even decrease with an increasing number of sources and/or destinations as the control traffic and the bus arbitration take time. Failure of the bus system will disable the whole event building system while on the contrary, failure of single sources or destinations is usually not critical.

FIGURE 3.6. Shared Bus Architecture

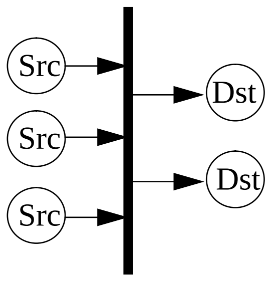


Table 3.1 shows typical maximum bandwidth values for some standard bus systems used in high energy physics [Par90]. Usually the theoretical bandwidth is not achieved due to the various control messages being sent and the actually achieved bandwidth is much lower.

TABLE 3.1. Maximum Throughput for Some Standard Bus Systems

Bus System	Standard	Theoretical Throughput	Practical Throughput
CAMAC	IEEE 583 (1975)	20 MB/s	< 1 MB/s
VMEbus	IEEE 1014 (1987)	40 MB/s	10..20 MB/s
VICbus	IEC VICbus	30 MB/s	~3 MB/s
Fastbus	IEEE 960 (1989)	200 MB/s	40..60 MB/s
Futurebus+	IEEE P896	3.2 GB/s	not yet available

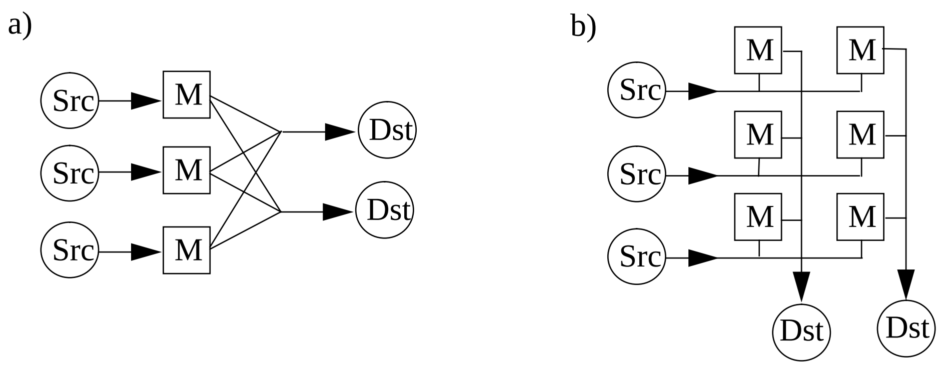
Some architectures try to overcome the limitations by using several independent single bus systems in a tree-like structure. This reduces the control flow in any of the single branches and failure of a single bus will only disable a certain branch rather than the whole system. Nevertheless the total throughput of such an hierarchial structure is always equal to that of the single bus.

In order to overcome the performance limit, another alternative is to use more than one single bus system in parallel. The sources and destinations are connected to several bus systems and if they do not contend for the same internal resources (e.g. by using multiple port memory) the total throughput can be increased and ideally reach the sum of throughputs of the single busses used. This is achieved by the fact that several events can be built in parallel if the sources can hold more than one event fragment at a time. Only a relatively small amount of control is needed and failure of one of the bus systems is not critical as the others can take over its data flow. Nevertheless, in practice, expanding an event building system using parallel busses is usually unreasonable for more than three busses.

3.2.2 Multiple Port Memories

Another architecture of event building systems can be based on multiple port memories. One example can be a system where the sources provide their data in multiple port memories. Each of the output ports of these memories can be connected to a destination. This architecture (figure 3.7a) provides parallel event building. It has similar features to the multiple parallel bus architecture and the same performance characteristics. Nevertheless, it is less reliable because there is only one path from each source to each destination. Expendability of such a system is limited by the number of physical ports the memory can have which usually is no more than three.

FIGURE 3.7. Interconnecting Networks with Multiple Port Memories

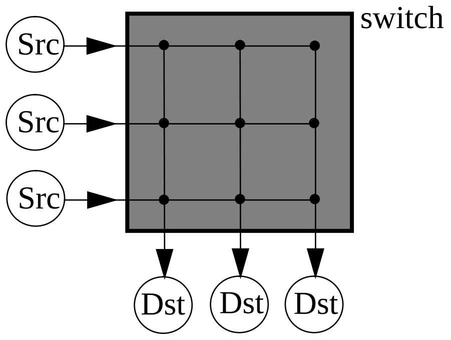


Dual port memories can alternatively be used in an array architecture where one port is connected to the sources and the other to the destinations (figure 3.7b). Event fragments are sent in parallel to the memories in a given column and sent sequentially to the destination while the next event can already be transmitted to another column. This architecture needs a relatively simple control and can tolerate failure of the destination busses. The system which is similar to a full crossbar switch can be extended easily but for a large number of sources and destinations it becomes expansive as the number of multiple port memories needed increases proportional to the product of the number of sources and the number of destinations.

3.2.3 Switches

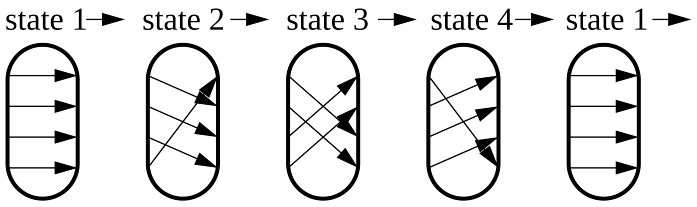
The connections between sources and destinations can also be established using a crossbar switch (figure 3.8). This switch which usually, though not necessarily, has the same number of inputs as outputs allows to connect any source with any destination and establish several connection at the same time, providing they do not involve the same sources and destinations. Event building using such switches can be done fully in parallel and the throughput is optimally that of the sum of links available. Buffering in the source and destination will be necessary but the number of memories needed is only proportional to the sum of the number of sources and the number of destinations. The full crossbar switch requires some control function to arbitrate the requests. It can be made tolerant to failures in the sources and destinations but failure of the switch will disable the whole system. However, reliable switches are available for several evolving data link technologies (see section 3.3).

FIGURE 3.8. Crossbar Switch



A special kind of switch is a "barrel shifter" with a simple rotating interconnection pattern (figure 3.9). Its control is much easier than for a full crossbar switch and it only needs a single control input defining the interconnection pattern. The time slot for each pattern should be appropriate to transmit the maximum sized event fragment. As they usually vary in size the switch might not be used efficiently. Alternatively the event fragment could be cut into packets queued individually for each destination.

FIGURE 3.9. Barrel Shifter

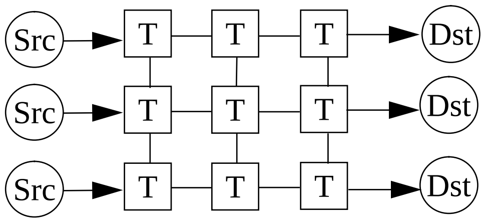


As the number of crosspoints in any crossbar switch increases with the product of the number of sources and the number of destinations, a multistage interconnecting network might be more efficient. In such architectures the full switch is replaced by a network of interconnected switches of smaller size. This leads to less crosspoints in total and is more tolerant to faults if redundancy of data paths is considered. Usually data in such networks is cut into packets which contain information on the destination. At any stage in the network this information is interpreted and the data stored until it can be forwarded to the next stage. This is called "self-routing" and practically used with a lot of switching networks commercially available.

3.2.4 Integrated Architectures

In an integrated architecture the task of event building is integrated with the task of subsequent trigger processing and/or data storage. This is achieved by connecting transferring and processing elements, replacing them by transputers and connecting them in a mesh structure (like in figure 3.10). The communication links between the transputers can be used to transfer the data from one stage to another. The transputers can do the data assembly and the trigger processing as well as the data flow managment. This is particularly promising if the trigger processing does not need the full event data all at once but works in several steps. It offers full configurability and the data flow can be dynamically controlled. Nevertheless, such an event building system is limited by the link speed on the input side and, for big systems, by the complexity of control needed.

FIGURE 3.10. Integrated Architecture using Transputers



3.2.5 Examples

Table 3.2 gives an overview of the event building systems used in experiments in high energy physics. It is not exhaustive and only gives a rough estimate of the total throughput achieved. This latter has of course to be seen in the context of the whole data acquisition system and depends on the trigger processing made up-stream of the event building system.

TABLE 3.2. Overview of Event Building Systems in Today’s Experiments

Experiment	Architecture	Technology	Total Throughput	Reference
Aleph	tree-like bus system	VMEbus, VICbus	< 1 MB/s	[Rue89] [Per94]
Delphi	tree-like bus system	Fastbus	< 1 MB/s	[Ada92]
L3	tree-like bus system	Fastbus	8.0 MB/s	[Ang94]
Opal	tree-like bus system	VICbus & VSBus	2.5 MB/s	[Bai93]
H1	tree-like bus system	VMEbus & custom fibre optic link	1.5 MB/s 12.0 MB/s (max)	[Hay92]
Zeus	crossbar switch	custom-made switch	24.0 MB/s	[Beh93]
D0	array of multi-port memories & trigger processors	custom-made bus (token ring)	40.0 MB/s (in) 0.8 MB/s (out)	[Cut92]
CDF	crossbar switch	ULTRANET Hub	50 MB/s	[Bar88] [Pat94]

The overview shows the wide variety of architectures and technologies used today. It also reveals a trend towards increasing total throughput. This development is a result of the increased requirements of experiments and the increased performance available in technology. Bus based systems, however, are not suited to provide the performance required for LHC experiments. They do not scale in number of connected sources and destinations and are limited in total throughput by the medium. Parallel event building system based on commercial standard for high speed interconnects and switching elements seems to offer the most promising solution and the LHC experiments [ATL94] have expressed a clear interest to investigate this further.

3.3 High Speed Data Links

Several interconnection standards for high performance computing environments or for networks in the field of telecommunication have evolved over the last ten years. They are based on point-to-point links which can be extended to networks with large numbers of end-points. They have different protocol characteristics, but allow sometimes to use one standard on top of the other. They differ in their approaches to error detection and recovery and they are usually used with different hardware.

3.3.1 High Performance Parallel Interface

The High Performance Parallel Interface (HiPPI) standard [HIP90] addresses the requirement to standardize and improve I/O bandwidth for high performance computing environments.

HiPPI is a 32-bit parallel, point-to-point link which transfers data in one direction (simplex) at 100 MB/s or 200 MB/s. It runs over 25 m of twisted pair cable, using one cable for the 100 MB/s version and two cables for the 200 MB/s version. Flow control across HiPPI is independent of the distance if there is enough buffer space and it is possible to build fibre optic “extenders” which run at the full bandwidth for up to a few km. Error detection is provided by byte-parity on the 32-bit data words and a checksum at the end of every (256-word) burst of data (LLRC word). The protocol on HiPPI is very simple using only *Request*, *Connect*, *Packet*, *Burst* and *Ready* signals. It allows variable length packets and supports connection-oriented as well as packet-oriented transfers.

The simplicity of the protocol allows to build switches which support very easy routing mechanisms. They can be staged to make large networks without introducing much overhead to the throughput. HiPPI is now a major standard in high performance computing and a variety of components is available. The RD13 project [RD13/95] chose HiPPI for its event building studies and this work will present the details of the prototype based on this.

3.3.2 Fibre Channel

The Fibre Channel standard [FCS94] is aiming at supporting a broad performance range of peripheral devices and networks in computing using multiple physical media. It addresses a similar field as HiPPI and is in some regard its successor. It overcomes the problem of uni-directional traffic and allows much more flexibility.

Fibre Channel is based on point-to-point links transferring data in both directions (duplex) over two different uni-directional cables or fibres running at rates defined between 16.5 MB/s up to 132.5 MB/s. It supports single mode and multi-mode optical fibres as well as coax cables and can cover distances depending on the medium, from 10 m up to 10 km. 8-bit data is encoded in 10-bit transmission characters, offering a disparity (excess 1's or 0's) to detect transmission errors.

The signalling protocol controls the data to be transferred between two ports based on frames which can be link control frames or data frames. The link control frames are used for transferring signals like *Acknowledge*, *Busy* and *Reject*. The data frames can contain up to 2112 Bytes of user data. The flow control of Fibre Channel distinguishes between three different classes: class 1 is a service providing dedicated connections with each frame being acknowledged by the receiving port. Class 2 is a frame switched connection-less service where multiple ports can share the bandwidth of a given physical channel by multiplexing the frames which are all individually acknowledged by the receiver. Class 3 is identical to class 2 without the acknowledgement of the frames. This is a datagram service.

Fibre Channel can be used in a wide range of architectures from single point-to-point connections up to ringlets and switched architectures (which in the Fibre Channel slang are called fabrics). This standard is evolving rapidly and workstation interfaces as well as switching elements are becoming available off-the-shelf. Its applicability in data acquisition systems is investigated by the RD11 project [RD11/95][Cha95].

3.3.3 Asynchronous Transfer Mode

The Asynchronous Transfer Mode [ATM92] is a switching and multiplexing standard for broadband integrated services digital networks (B-ISDN) used in telecommunications. It is considered as a specific packet-oriented transfer mode based on fixed length packets which can be transmitted over a variety of physical media used in wide area telecommunications as well as in local area networks.

Its most important characteristic is the flexibility in bandwidth allocation through statistical multiplexing of different data streams allowing every bit rate to be arranged between the user and the network, up to the link speed. The fixed-size packet, called cell, has 53 Bytes of which 5 are header information and 48 are user information. The header information contains identifiers on the channel to be used and some priority information and it determines the routing through a network. The user information carries the user's data. Protection against transmission errors is not included for performance reasons but instead is provided by higher level protocols available on top of ATM like the AAL5 (ATM Adaptation Layer 5) which groups cells into fragments of up to 64 kByte and contains some flow control mechanisms.

ATM is becoming more and more available and switching networks are being delivered. The applicability of ATM networks for event building systems is investigated in the RD31 project [RD31/95][Let94].

3.3.4 Scalable Coherent Interface

The Scalable Coherent Interface [SCI92] is a standard for large multiprocessor systems and shared memory architectures where it shall replace standard busses to overcome their limitations in bandwidth and scalability. This is achieved by connecting processors and memories with a set of simplex point-to-point links in ring structures.

Each single link allows packet-oriented transactions with an address, a command and 0, 16, 64 or 256 Bytes of data. The link can be 16-bit parallel cable with a rate of 1 GB/s and a maximum distance of 10 m or a 1-bit serial optical fibre for larger distances and up to 256 MB/s throughput. These links can be interconnected in networks or in ringlets. Rings can be interconnected by bridges which are made from SCI nodes arranged back-to-back.

The protocol splits transactions into a request and a response each one requiring a packet. *Read* and *Write* primitives are defined as well as more complex transactions, for instance lock and cache coherency primitives. They are very important in multiprocessor environments with shared memory configurations.

SCI is now available and interfaces to workstations and standard busses can be bought off-the-shelf. Complicated architectures like rings, bridges and switching elements and their applicability to event building systems are investigated in the RD24 project [RD24/95].

3.4 Data Flow Management

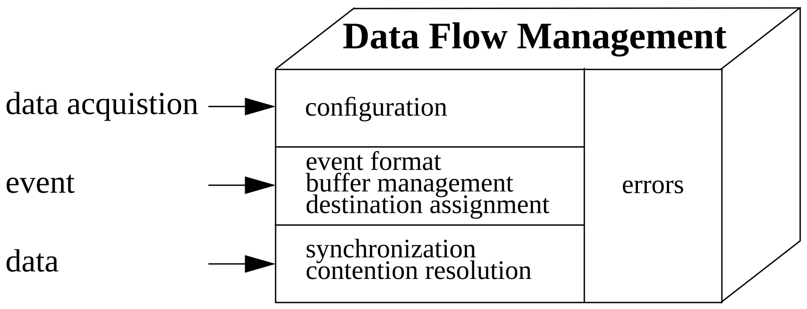
The data flow management (see section 3.1.2) is the control element of an event building system. It uses a set of rules which is also called the event building protocol [Bog95]. It describes the dynamic behaviour of the event building system where the sources and destinations are associated with data flow processes, the *Src* and *Dst* process respectively.

The data flow management controls transfer of data, event handling and the configuration of the event building system. These three aspects correspond to the following layers of the data flow management:

- **Data Layer:**
  On the lowest layer the data flow management synchronizes the sources, the interconnecting network and the destinations. It routes data from a known source to a known destination and has in particular to resolve contention in the interconnecting network.
- **Event Layer:**
  The middle layer of the data flow management is concerned about events and event fragments, their format and buffer management. Event fragments have to be assigned to a unique destination where they are assembled to full events.
- **Data Acquisition Layer:**
  The highest layer of the data flow management describes the event building system in terms of sources, interconnecting network, destinations and their various parameters. It provides the communication with the overall control system of the data acquisition.

In all three layers the data flow management has to detect and handle errors. It must ensure that the functionalities of an event building system are guaranteed: that the data are assembled in the correct way, that no data get lost or corrupted, that no deadtime is introduced to the data acquisition system and it should also minimize the latency.

FIGURE 3.11. Layers of Data Flow Management



The data flow management can be centralized or distributed over the data flow processes. And in most cases there is a supervisor process taking care of exceptions and the communication with the control system of the overall data acquisition system. The data flow management can be implemented in hardware or software and will usually create some control flow which either is transferred over the same network as the event data or a separate network.

3.4.1 Data Flow Processes

Data flow processes are an implementation of the data flow management. They receive, process and send events and event fragments. They have an input data connection and one or more output data connections. A data connection is defined as a logical path on which data will flow sequentially. Since data are described by a three-layered data model (see section 3.1.1) a connection will have to define means for synchronization, transfer of event descriptors and transfer of full data. The *Src* processes have an input data connection from the up-stream data acquisition and an outgoing data connection to the interconnecting network. The *Dst* processes have an incoming data connection from the interconnecting network and an output data connection to the down-stream data acquisition.

In addition, the *Src* and *Dst* processes have a buffer management to make sure that no data get lost. They may send a signal down-stream when the buffers fill up which can be used to throttle the data or they might overwrite and lose data. Furthermore, they have to know the event format which describes the event descriptors, their relation to the actual data and the coding of the actual data in bits and bytes. The *Dst* process furthermore has to do the actual merging of the events. It might re-order out-of-the-order event fragments, build several events in parallel and re-format the full event. It must watch over the number of event fragments still missing and ensure that missing fragments will be detected.

These functions need a high flexibility and will most likely be implemented in software. The data flow processes require some processing power, event fragment buffer, data connections to the input/output data streams, data connections to the interconnecting network and a connection to the data flow management or to a network for exchange of control messages. A standardized real-time operating system might be helpful to develop and maintain the data flow processes which will likely be multi-threaded in order to deal with the many signals from the data connections, the data flow management and the overall control system of the data acquisition system.

3.4.2 Contention Resolution

The contention resolution becomes important when several sources are requesting a connection to the same destination (output blocking) or are contending for the same internal resource (internal blocking). In such cases, arbitration is required defining which connection will actually be established or which source will be granted the requested resource. There are several algorithms for contention resolution:

- **Round-Robin:**
  The requests of the sources are handled in a cyclic manner. The next one is the one next on the cycle.
- **Random:**
  The requests are handled in random order. The next one is sampled from a uniform random distribution.
- **Fifo:**
  The requests are handled in the order they arrive. All requests are put in a fifo and the next one is the one in front of the fifo.
- **Priority:**
  The requests are weighted with a static priority associated with the source they come from. The next one is the one with the highest priority.
- **Programmable:**
  The requests are weighted with a dynamic priority. The next one is then chosen from a programmable algorithm taking into account the priorities.

3.4.3 Destination Assignment

The destination assignment has to define where full events are going to be assembled. It assigns a unique destination to all event fragments with a given event identifier. The destination assignment controls thus the flow of data through the interconnecting network and has to work in an efficient way to avoid contention. There are several ways to accomplish this task.

The destination assignment can be static or dynamic. In a static scheme the decision is not based on any information about the status of destinations or the status of the interconnecting network. In the dynamic scheme this information is taken into account and the event is built at a destination such that the interconnecting network and the destinations are optimally used. An example of a static scheme is round-robin or random. Dynamic algorithms can choose the next free destination, or take the load of the interconnecting network into account routing event fragments to unused parts, or calculate the next destination from the data sent so far.

Another important aspect is who controls the data flow. This is not to be confused with the control of the actual data transfer which can be source-driven (i.e. the data is written) or destination-driven (i.e. the data is read) [Bog95]. In a push scheme the sources initiate the flow logically: they assign the destination by either applying a static algorithm or by choosing the destination from a list. It does not matter if the data then are actually sent through the interconnecting network or if it is read from it by the destination. In a pull scheme the destinations initiate the transfer in the sense that they send requests to the sources.

The data flow control can be centralized by using a single server or a single list of free destinations or of event fragments available. Alternatively it can be distributed so that the sources and destinations communicate and decide on a destination.

3.4.4 Configuration

The configuration is part of the highest level of the data flow management. It consists of a set of rules to define how the data flow processes and the data flow management are to be configured and how they communicate with each other and the rest of the data acquisition system.

The configuration management provides the framework of data flow processes, the data flow management and the communication channels between them. The static description of this framework could be held in a database. The dynamic aspect of the configuration management is described in three phases:

1. The data flow processes and if necessary the data flow management process are generated and started. A connection for exchange of control, status and error messages is established.
2. The data connections between the processes are set up and it is ensured that the processes at both ends know about each other. In addition, the data flow processes create a connection to the data flow management or set up a distributed scheme.
3. The data flow processes are ready to receive and send event data. The configuration management can then finally change the state of the processes by stopping and/or deleting them.

The configuration management should be made in a way that it is scalable with the number of data flow processes and connections. The modular structure should support different types of connections and their synchronization schemes. A prototype of such a protocol, not explicitly designed for event building, has been developed in the RD13 project [Mor92]. The data flow protocol contains elements of the data layer, of the event layer and the data acquisition layer.

3.4.5 Error Handling

The data flow management also has to detect and recover from errors. Errors can occur at several points in the event building system:

- Control transmission:

There can be errors in the control connections which lead to missing synchronization signals or to duplicated signals if the hardware has a retry mechanism.

- Data transmission:

  The interconnecting network can produce errors due to hardware failures. These errors are usually detected by the hardware and the mechanisms are part of the standard used.

- Fragment transmission:

  An event fragment sent through the interconnecting network can get lost either because the hardware link has a failure, the internal resources have an overflow (e.g. cell loss in ATM) or the fragment gets misrouted.

- Data flow processes:

  The sources and destinations can be in an erroneous state or hung up due to malfunctions of the processes and/or processors. Their buffers can overflows, the synchronization with the outside data acquisition system can be lost etc.

Error detection is already partly covered by some high speed interconnect standards but some aspects have to be included in the software of the data flow management and have to use checkwords, acknowledgements and/or time-outs. Error recovery can be done by re-sending a control message or by re-routing a single event fragment or all event fragments belonging to the same event. An event can also be tagged as being corrupted or could be dropped after which the data flow processes continue their work. The different options have to be tested under realistic and provoked conditions to measure their efficiency in detecting errors and the time they need to recover from errors.

4.0 Event Building Prototype

As mentioned in chapter 3 event building systems for future experiments in high energy physics based on standard bus systems would not provide adequate performance and alternative solutions providing a high level of parallelism have to be envisaged. Interconnecting networks made of networks of small switches or a single large-scale switch will have to be used. Several high speed interconnecting standards are evolving in computer and telecommunication industries which will offer the aggregate bandwidth and level of connectivity required. High energy physics experiments will greatly profit from such huge industrial efforts if parallel event building systems can be built using these technologies.

In order to investigate the suitability of industrial standards for event building systems prototypes have to be built. These are to be regarded as small scaled-down versions of parallel event building systems and of other parts of modern data acquisition systems where data assembly with high bandwidth and a high level of connectivity is needed. The prototypes serve in particular as input to simulation models which simulate large-scale event building systems before these are actually built.

A prototype can provide knowledge about the important features of parallel event building systems, encourage analysis of the problem and help to define the requirements for event building systems. Implementing a prototype, furthermore, helps to understand the equipment available or needed, helps to evaluate the suitability of a given technology and requires software which adapts the hardware to its required functionality. The structure of software needed, the degree of independence from a given technology and the integration with an overall data acquisition system can be investigated. Finally, measurements with the prototype lead to typical values for the performance indices with which different technologies can be compared and a final choice can be made. The measurements will also be used to tune the simulation model which will be extended to a realistic event building system.

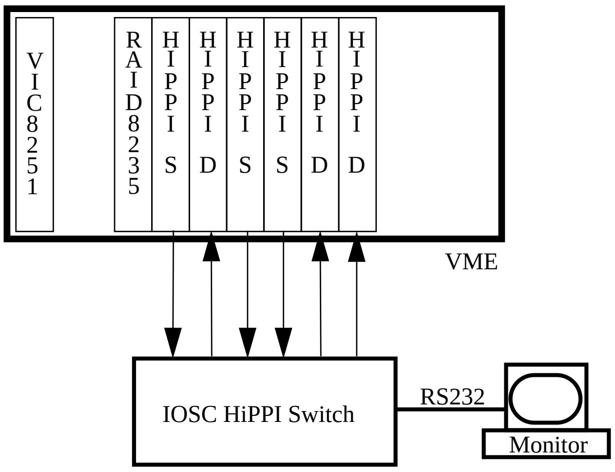
The HiPPI standard is one of the evolving high speed interconnect standards. Switches are commercially available and so are VME-HiPPI interfaces. The latter are important as VME is a de-facto standard for today’s data acquisition systems and many sources and destinations of experiments in high energy physics reside in VME crates. A prototype of an parallel event building system based on a HiPPI switch with a modern data acquisition system compatible with the RD13 system [RD13/95] has been built and tested.

4.1 Hardware

The whole prototype event building system as shown in figure 4.1 is housed in one VME crate except for the HiPPI switch itself which is a completely independent module. The VME crate contains the VME-HiPPI interfaces and a processor board for control. The VME-HiPPI interfaces are cabled up with the switch which routes and transfers the data from sources to destinations providing a bandwidth of 100 MB/s per link.

This system can be regarded as a model of a parallel event building system where the VME-HiPPI interfaces play the role of sources (HiPPI source) and destinations (HiPPI destination) respectively. They can be taken as the readout modules collecting data from the subdetectors or modules of subdetectors and full event trigger processors, interfaces to full event processing farms or interfaces to data storage media. With different requirements the prototype can also be regarded as a model for other parts in the data acquisition system where a high level of parallelism and connectivity is required (e.g. the ATLAS LVL2 system, see section 2.4.4).

FIGURE 4.1. Layout of the Prototype Event Building System

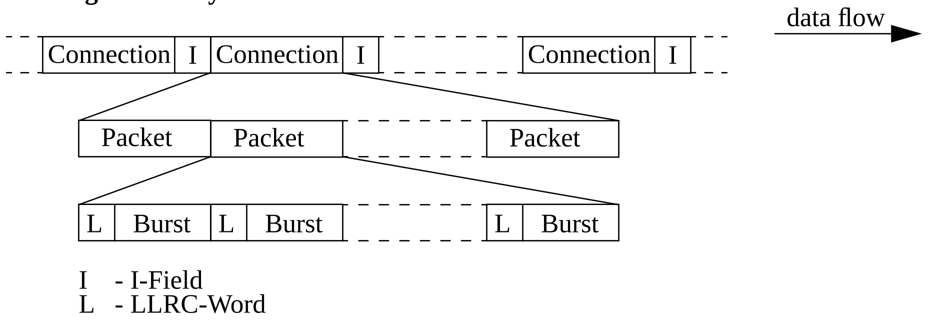


The HiPPI interfaces and the HiPPI switch have the role of the interconnecting network. The contention resolution is implemented in the hardware while the destination assignment has to be in software. The processor board runs a mini data acquisition system with simulated data input. It implements the data flow processes, the data flow management and is also used to monitor functionality and to measure performance. A VIC 8251 [VIC92] is used to drive the VME system's clock and to provide arbitration of the VME-bus. It is not used otherwise and will not be mentioned further.

4.1.1 HiPPI Standard

The HiPPI standard has already been mentioned in section 3.3.1. It is a clock-driven, uni-directional high speed interconnect with two different data rates of 100 MB/s or 200 MB/s using either a 32-bit or a 64-bit wide data bus. The following will describe the 32-bit wide solution which was used in the prototype. It should be expected that using the 64-bit wide data bus would roughly double the performance.

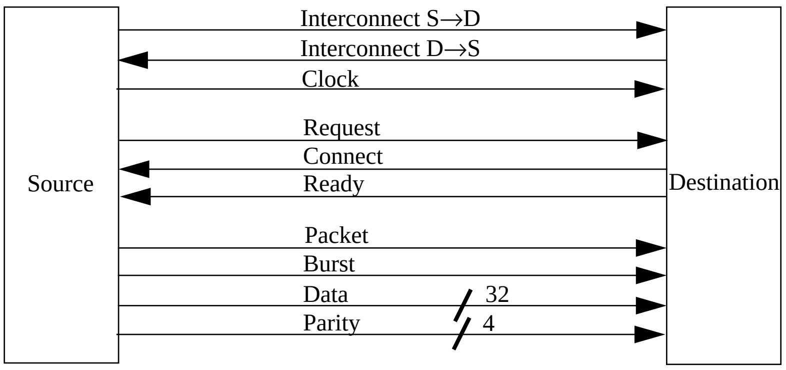
FIGURE 4.2. Framing Hierarchy of HiPPI



The HiPPI standard is based on connections between source and destination. Once a connection has been established the data flows in packets with each packet having none or more bursts. A burst consists of 256 words sent in a contiguous clock period. Shorter bursts with less than 256 words are only allowed as the first or last burst of a packet. This framing hierarchy of the HiPPI standard is shown in figure 4.2.

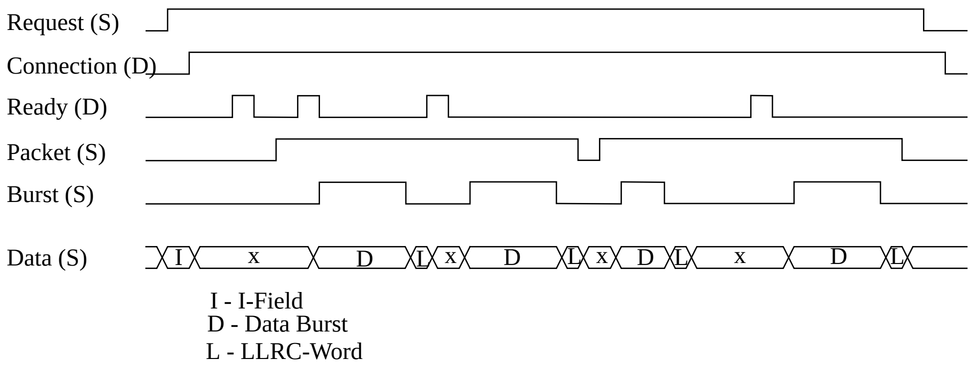
Establishing a HiPPI connection requires only a few signals between sources and destinations which are shown in figure 4.3. All signals except *Interconnect* use differential emitter-coupled logic (ECL) drivers and receivers. *Interconnect* signals use single-ended ECL drivers and receivers and are used to indicate connected cables and that the other side is powered up.

FIGURE 4.3. Summary of HiPPI Signals



A typical waveform is shown in figure 4.4. First a connection is requested by sending a *Request* and supplying the I-field (data word for routing information) on the data bus. The destination accepts a connection by asserting the *Connection* signal. In case the destination wants to reject a connection on purpose it will assert the *Connection* signal for only 4 to 16 clock cycles before releasing it again. The source will then interpret this as *Answer-Reject*. If the signal is asserted for more than 16 clock cycles the source will interpret this as an *AnswerAccept* and will start transmitting the data in packets and bursts which are delimited by a *Packet* and *Burst* signals respectively. Each burst is immediately followed by a LLRC (Length/Longitudinal Redundancy Check word) which as well as the parity sent on the parity lines in parallel with each burst is used for error detection.

FIGURE 4.4. Typical HiPPI Waveforms



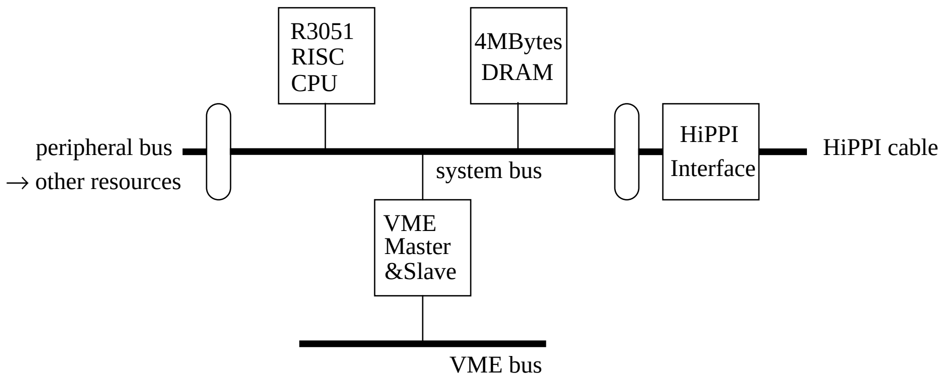
The destination controls the flow of data by using the *Ready* signal because a source can only send as many bursts as it has received *Ready* signals. If the *Ready* signal arrives before the source has sent a burst the signals will be counted and the next burst can be sent immediately. If the source has not received enough *Ready* signals, the sending of a burst will be delayed until the next *Ready* signal.

4.1.2 VME-HiPPI Interface

The VME-HiPPI interface is a VME based board composed of a RIO board [RIO91] and a HiPPI interface [HIP92]. Local intelligence and memory are provided by the RIO board and two different HiPPI interfaces are available to be attached on the RIO board: a HiPPI source (HiPPI/S) and a HiPPI destination (HiPPI/D) thus making two types of VME-HiPPI interfaces.

The VME-HiPPI interfaces consist of a R3051 RISC controller, 4 MByte DRAM, VME master and slave interfaces and a HiPPI interface. Other resources are an EPROM for bootstrapping, different timers as well as CSR registers and a fifo for command passing. All the elements are grouped around a custom-made internal bus with a maximum data transfer rate of 50 MB/s. This structure is shown in figure 4.5.

FIGURE 4.5. Block Diagram of the RIO 8262/HiPPI



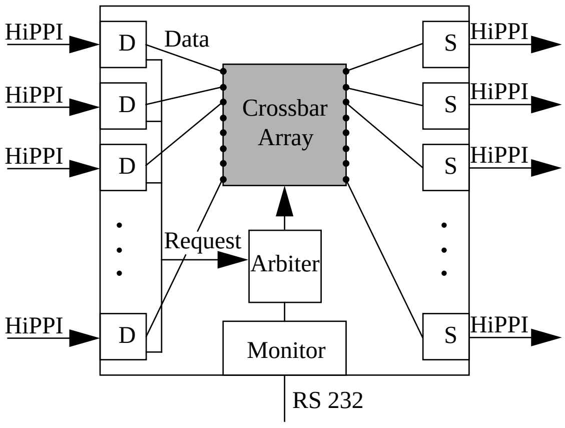
The VME master and slave interfaces connect the board to the VME backplane bus and allow access to all the resources on the board from the VME side. The HiPPI interface was developed jointly by CES and CERN [Pra92] and implements an equivalent of the HiPPI framing protocol in hardware. The RISC controller can access the interface by writing command words in registers, will read status information from these and receive interrupts from the interface. The packing into bursts is handled by the interface and hidden to the RISC controller. Data of 1024 words can be buffered by the interface and data transfer between the local memory and HiPPI interface can be made in a fly-by mode which does not require any further interaction from the RISC controller after setup (DMA).

4.1.3 HiPPI Switch

The HiPPI switch [SWI92] is a non-blocking, 8 input and 8 output crossbar switch. It is the heart of the prototype event building system and can connect any of its input ports with any of the output ports with each connection allowing a bandwidth of 100 MB/s. Several connections can be established at the same time without interfering with each other if they do not include the same sources or destinations. Thus the aggregate bandwidth of the switch is 800 MB/s in total.

The switch (figure 4.6) consists internally of four different kind of boards: destination ports on input, source ports on output, an arbiter with the slave array and a monitoring board. The destination ports implement a HiPPI destination, receive data from the input and send connection requests to the arbiter. The arbiter receives these requests and configures the slave array which is a passive crossbar switching array. The outputs of the slave array are connected to the source ports which implement a HiPPI source, receive data from the slave array and send it to the output. A monitoring board can be accessed via a RS232 link. It manages the logical addressing tables and monitors the performance of the slave array.

FIGURE 4.6. Block Diagram of the HiPPI Switch



Following the HiPPI standard the I-Field is used for routing information. The bits of the I-field have the following interpretation:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
L	VU		DW	D	PS		C	routing							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				routing											

L	locally administered (not used)				
VU	vendor unique (not used)				
DW	double wide (not used)				
D	direction:				
	0	(2,1,0)	source addressing,	(11..0)	logical addressing
	1	(23,22,21)	source addressing,	(23..12)	logical addressing
PS	path select:				
	00	source addressing			
	01, 11	logical addressing			
	10	reserved (not used)			
C	camp on				

The addressing of a destination can be done in two different ways: source and logical addressing. In the source addressing 3 bits of the routing field are used directly to determine the output port. The I-field is rotated by 3 bits in order to allow routing by subsequent switches. In logical addressing the routing field is used as a pointer to a table which contains the output port. This table is directly programmable via the monitor board.

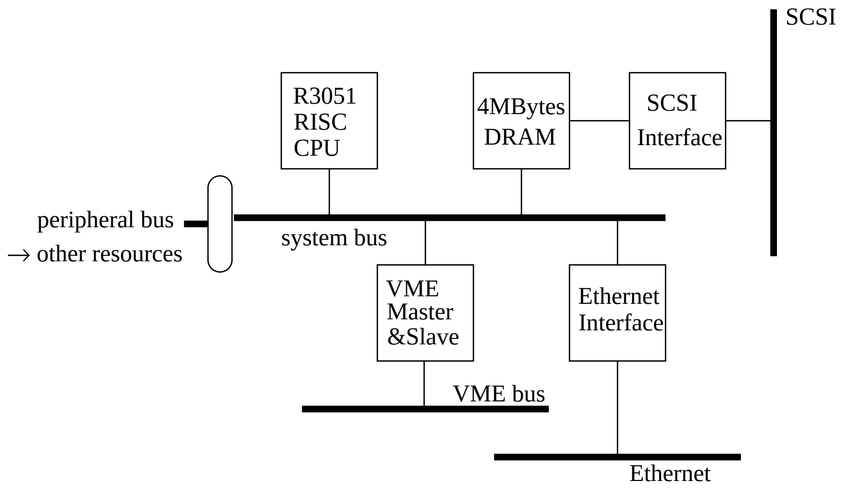
Another important feature is the “camp-on” facility. If the according bit is set in the I-field a request to an output port will not be rejected when this one is busy but it will “camp on” which means it will be kept pending until the output port becomes available. If several requests are camping on the same output port the arbiter will serve them in a round-robin manner.

4.1.4 RAID Processor

The RAID processor [RAID92] is a VME based board with a local processing unit, memory and other resources that make it useful as a general-purpose processor within the VME addressing space. In particular it runs an operating system.

The RAID board consists of a R3000 RISC processor with a floating point unit, 32 MByte DRAM, a DMA device and VME master and slave interfaces. These elements are grouped around a custom-made bus with a maximum data transfer rate of 80 MB/s. An interface to Ethernet supports full TCP/IP protocol and an SCSI interface connects the RAID board to storage media. Other resources like different timers, CSR registers and fifos exist on the peripheral bus. This structure is show in figure 4.7.

FIGURE 4.7. Block Diagram of the RAID 8239



4.2 Software

Software was developed to use the above presented hardware as a prototype event building system. This software consists of a minimal data acquisition system with simulated data input and methods for performance measurements. It is an implementation of a three-layered data flow management (see section 3.4).

The data layer consists of the HiPPI standard, the firmware for the HiPPI/S and HiPPI/D modules and libraries for the communication between these modules and the RAID processor. The firmware sends or receives event fragments using the HiPPI standard and exchanges VME interrupts and single word commands with the data flow processes thus making the data layer completely transparent from the higher layers.

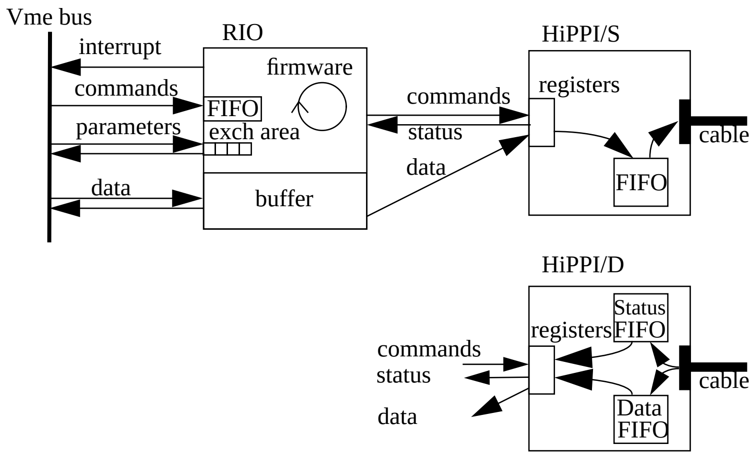
The event layer implements data flow processes which send, receive or process event fragments. It contains the destination assignment and collects performance statistics. The event format, buffer management and event assembly are common on data and event layer. The assembly can be accomplished in either.

The data acquisition layer implements a hard-coded configuration of the data flow processes, the data flow management and their connections.

4.2.1 HiPPI Firmware

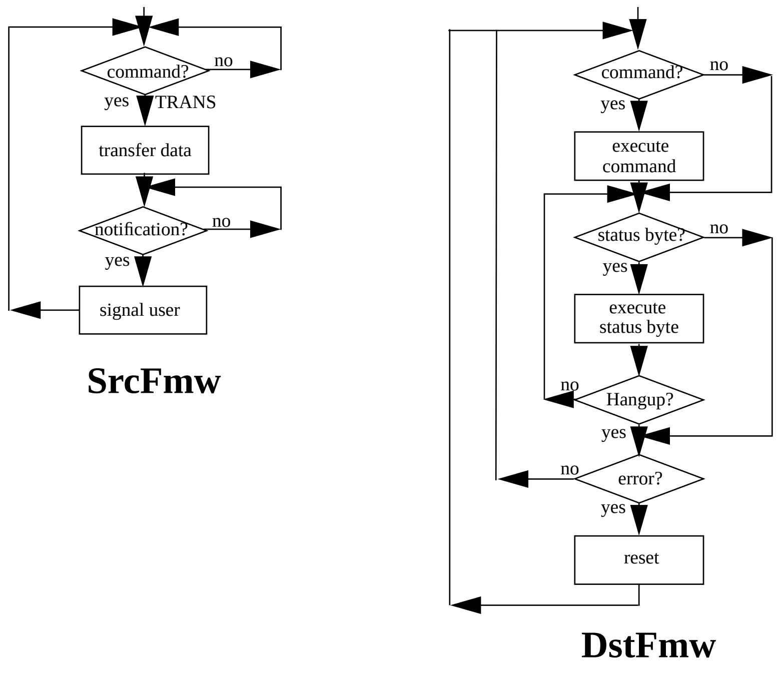
Firmware [Spi95a] was developed for the VME-HiPPI interfaces to allow them to be used as VME-accessible interfaces for the RAID processor and to give access to the HiPPI link. The VME-HiPPI interfaces receive commands via VME and are capable of generating VME interrupts at the end of transfers or in case of errors. The firmware was conceived to let the VME-HiPPI interface look as an extensional resource to the RAID processor (figure 4.8). There are two versions of the firmware: one for the HiPPI/S module, *SrcFmw*, and one for the HiPPI/D module, *DstFmw*.

FIGURE 4.8. Firmware for the RIO /HiPPI



The firmware was newly developed, based on former existing firmware [Bij93][Buo93a]. This was necessary to match the requirements of the event building system and to improve the performance. The firmware has to be cross-compiled, then down-loaded to the VME-HiPPI interfaces and started. As there is no operating system on the RIO board the firmware is a simple loop checking the command fifo and various registers of the HiPPI interface. If it finds a command or certain status on the HiPPI interface it can react accordingly (figure 4.9). The firmware accesses the RIO boards resources by using the BIOS library [BIOS92]. It also uses some monitoring program communicating with RAID processor which allows debugging. Several options in the firmware can be chosen at compilation time.

FIGURE 4.9. Block Diagram of Firmware



As the HiPPI interface on the RIO board has the HiPPI protocol implemented in hardware the firmware only has to be concerned about the framing protocol and by the transfer of data from local memory to the HiPPI interface. Each transfer command leads to a dedicated HiPPI connection which is expressed in the following command sequence:

RingRequest	to request a connection using the I-field
PacketSetup	to start a packet and define the number of words to be sent, this is used by the HiPPI interface to pack the data into bursts
ImmediateData	to transfer data from local memory to the HiPPI interface's fifo
PacketEnd	to end a packet
HangupRequest	to break the connection

The firmware on the HiPPI/D module reads status information from the HiPPI interface's status fifo and reacts. The status words are called "Indicates" and the sequence for a normal data transfer is the following:

RingIndicate	read the I-field, get a free event descriptor, establish the connection
TransferIndicate	read a data burst into memory
PacketEndIndicate	close a packet
HangupIndicate	break the connection, merge the event fragment and generate an interrupt (optionally)

The *Ready* and *Hangup* signals are automatically generated by the HiPPI interface. The *Connection* signal is under firmware control and depends on the availability of a free event in the buffer. If none is available the connection will be pending until a free event becomes available or the request is dropped. This acts as a back pressure signal if the destination cannot take the data as fast as the sources can send them.

Both firmware versions check the error conditions reported in a register of the HiPPI interface. These include parity and LLRC errors as well as events in the HiPPI protocol (e.g. HiPPI/S: unexpected HiPPI/D hang-up) and time-outs (e.g. time between two bursts is too long). Thus the interfaces have information if the data was correctly sent (HiPPI/S), or correctly received (HiPPI/D).

The *SrcFmw* and *DstFmw* communicate with a program on the RAID processor following a simple user-client paradigm. The program will send commands to the RIO fifo and additionally use the same fields in a VME accessible exchange area. The RIO will then report back by sending a VME interrupt and/or writing status information in the exchange area. A library [Spi95b] collects all the commands and accesses of the exchange area and provides simple commands for asynchronous transfer waiting for a VME interrupt or synchronous transfer polling the status field. Each event fragment is handled as one connection. The I-field is taken from the event descriptor and contains the following fields:

I-field:	0xHHEEEDSS
H	field for HiPPI parameters (see section 4.1.3)
E	field for event identifier
D	field for destination identifier
S	field for source identifier

It supports both, the source and logical routing on the switch.

4.2.2 Event Handling

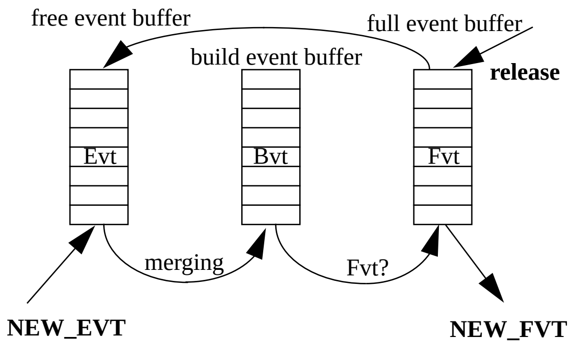
The firmware as well as the data flow processes use the same event format and buffer management which are closely related. The event merging is based on these and can be carried out either in the firmware or in the data flow process.

The event format defines an event descriptor and the actual data. The event descriptor contains a manyfold identifier for event identifier, source identifier and destination identifier, a status words to describe size and quality of data (errors), pointers to the actual data and pointers for internal use by the buffer management or event merging. The actual data of an event is an unformatted contiguous block of VME-accessible memory. This can easily be extended to use hierarchial block structures [Amb94a]. The synchronization aspect of event data is handled by the communication between firmware and data flow process.

The buffer management defines fifos and a data area. The fifos hold events descriptors with each one having a pointer to a fixed size slot in the data area. The data area is the place where the actual data is kept. This allows the event descriptor to be moved rather than the actual data.

The event merging (figure 4.10) is based on three states of an event: event fragment, intermediate event in stage to be built and full event (*evt*, *bvt*, *fvt*). The merging is a routine which uses a single fifo (*BvtBuf*) to hold the intermediate events. It is started every time a new event fragment becomes available. The new event fragment is taken and the *BvtBuf* searched if there is already an event with the same event identifier. If not, the event fragment is added to the *BvtBuf*. If there is already an event the new fragment will be merged by chaining the pointers. The number of chained event fragments is counted and compared to the number of fragments expected. If the event is complete it is taken from the *BvtBuf* and returned.

FIGURE 4.10. EventMerging



This scheme implements concurrent event building where several event fragments arriving out of order can be built at the same time.

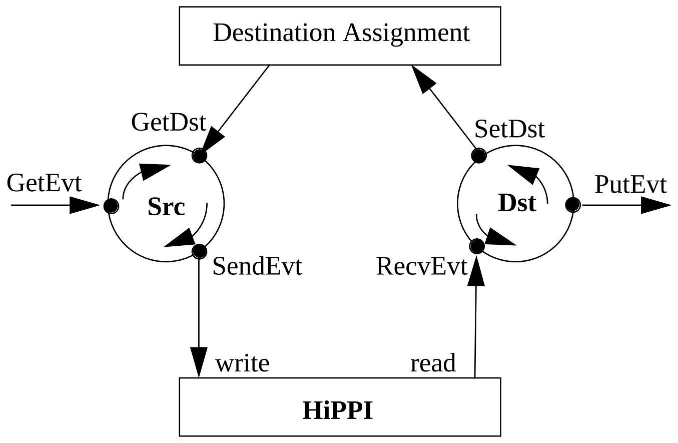
4.2.3 Data Flow Processes

Data flow processes (see section 3.4.1) have been developed. They are conceived as processing elements between an input and an output connection. These connections are static descriptions of the data flow and have three layers like the data they are transporting: the synchronization layer describes a way to synchronize the processes on both sides, the description layer describes where to find event descriptors and the data layer describes the actual data transport.

Two different types of connections have been defined: the shared-memory-type and the HiPPI-type. In the shared-memory-type the synchronization is done by means of interprocess communication, the descriptors are held in fifos and the data in the data area of the buffer management. The HiPPI-type connection uses VME interrupts for synchronization and accesses event descriptors by VME slave access. The actual data is usually not moved but some data are pre-loaded onto the HiPPI/S module and only the event descriptor is moved to change event size and/or destination. Some tests have been made where the actual data was moved using the RAID boards DMA device.

The *Src* process receives new events on its input connection, reads the event descriptor and finds a destination using the destination assignment scheme (see section 4.2.4). Then it sends the event fragment to the output connection which is of the HiPPI-type. Depending on the control scheme it might intermittently store the event either on the input side or on the output side as needed. The *Dst* process receives a new event from an input HiPPI-type connection and reads the descriptor. It can have an output connection to the downstream data acquisition system. The *Dst* process reports its status according to the actual destination assignment scheme.

FIGURE 4.11. Data Flow Processes



Both data flow processes are implemented as a collection of routines using the firmware library and some features of the operating system to implement the various kind of connections.

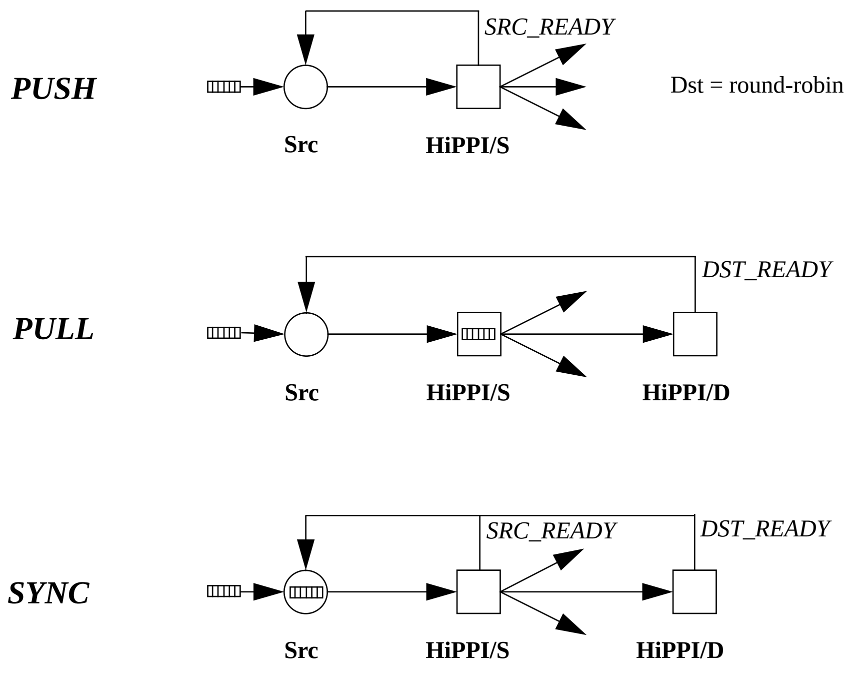
4.2.4 Destination Assignment

Three different destination assignment schemes have been implemented: *PUSH*, *PULL* and *SYNC* as shown in figure 4.12:

- In the ***PUSH*** scheme no information on the destination is used for routing the event fragments. The destination where the event is going to be built is assigned statically by means of round-robin or a random function.
- In the ***PULL*** scheme the destinations send a request to the sources for the next event. The sources will react by sending the next event to the specified destination. Nevertheless, the request will be sent to the HiPPI/S module without checking the status of the last transfer. If the last transfer is not finished yet the request will be queued in the HiPPI/S module and the *Src* process is ready to receive the next event.
- In the ***SYNC*** scheme the destinations send a request like in the *PULL* scheme but instead of queueing the request on the HiPPI/S module the *Src* process will wait until the last transfer is finished successfully. The event fragment with the destination already assigned will be queued on an internal output queue of the *Src* process until the previous transfer will be finished. Only after this the new request to the HiPPI/S module will be sent.

In the schemes above the *Dst* process receives the signal from the HiPPI/D module that a new full event has been built. This signal is directly used as a request for the *Src* process. Since this request will always arrive at the sources one after the other no additional assignment on which event fragment is handled by which destination request is necessary.

FIGURE 4.12. Destination Assigment Schemes



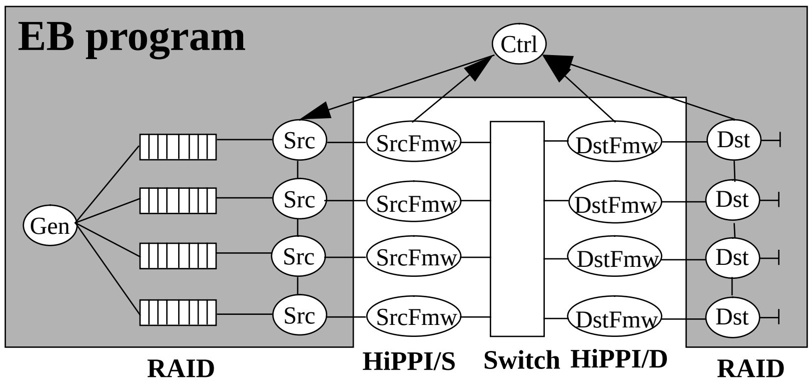
4.2.5 Configuration

The configuration of the *Src* and *Dst* processes and the destination assignment scheme is hard-coded in the EB program (as shown in figure 4.13). The number of sources and destinations as well as the destination assignment scheme are chosen at compilation time.

The EB program runs under EP/LX [EPLX91] on the RAID processor. EP/LX is a UNIX variance with real-time features compliant to POSIX 1003.4 [POS90]. The program creates one single UNIX process reacting to the event synchronization signals. It consists of only a single thread of control reading from the signal queue and reacting accordingly which means to call a data flow process. These latter are implemented as C routines in the EB program. The program can also wait for a particular data flow process to get an interrupt. No other run control functions [Jon93] are implemented. The program is the only user process on the RAID processor and runs at high priority.

When the program starts it creates static structures for the processes and establishes the connections between these. In the HiPPI-type connection this means to setup a connection between the EB program and the HiPPI module and to pre-load the event data. An additional data flow process is created for event generation (*Gen* process) which generates events as soon as one of the input fifos of a *Src* process becomes empty. This way, the rate measured always equals the maximum throughput T^{max} (section 3.1.3). After setting up the connections the *Gen* process starts generating a number of events and the *Src* and *Dst* processes will react according to the destination assignment scheme. The *Dst* process is dummy as the merging is done in firmware and the signal of a new full event is directly transferred to a destination request. If one of the data flow processes detects an error the EB program will report it and stop.

FIGURE 4.13. EB Program



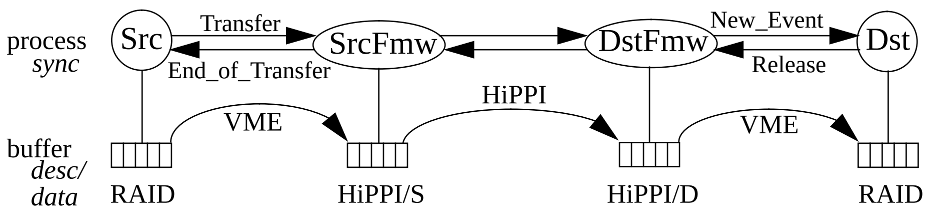
4.3 Simple Data Transfers

Before the system was actually used for parallel event building simple data transfers from one source to one destination were investigated to test and debug them working together. The results are especially useful to see what typical performance values there are.

4.3.1 Data Path

Simple data transfers from a single source to a single destination use the data flow management, which in this case is a simple transfer protocol. The data flows logically (figure 4.14) from the *Src* process to the HiPPI/S module running the *SrcFmw*, then via the HiPPI cables and through the HiPPI switch to the HiPPI/D module running the *DstFmw* from where they get to the *Dst* process. The synchronization between the HiPPI modules is completely handled by the HiPPI protocol as is the transfer of actual data. The event descriptors are transported between the HiPPI modules and the data flow processes by VME read access and between the HiPPI modules partly in the I-field and are partly reconstructed on the receiving side. The flow of actual data between the HiPPI modules and the data flow processes can be made by VME using different options.

FIGURE 4.14. Data Flow for Simple Data Transfers



Different functions have to be carried out at each step of the data flow. Each of these can be done in hardware or software and produce an overhead which contributes to the latency. Since some of these hardware and software processes run concurrently the latency is not a simple sum of the overheads. Although values for the individual overheads are known (see section 4.3.4) the total latency in a given configuration should always be measured.

Simple data transfers can be used to measure the minimum latency and individual overheads. In this case the EB program has only one *Src* and one *Dst* process and uses several special configurations of hardware and software to measure the contribution of each individual step in the data flow:

- *Src* process dummy:

  The *Src* process starts the first data transfer from a pre-loaded event on the HiPPI/S module. The *SrcFmw* then loops over a given number of events and reports at the end to the *Src* process. This way one does not include the overhead due to the *Src* process or the communication between the *Src* process and the *SrcFmw*.
- HiPPI/D dummy:

  A special HiPPI chip is available that responds to the HiPPI protocol without any (measurable) delay and without reading any data, thus running as an ideally fast HiPPI destination. This chip, called “never ending destination” (NED), can be connected directly to the output of the HiPPI/S module. It can be used to measure the latency without any contribution from a real HiPPI destination.
- Firmware options:

  SrcFmw and *DstFmw* can both be configured to use pre-allocated and pre-loaded data in which case the overhead due to internal buffer management is avoided.
- Interrupts:

  The *Src* and *Dst* process have several options to communicate with the HiPPI modules and EP/LX offers different possibilities to handle VME interrupts.
- Reading Data:

  The data transfer between *Src* process and HiPPI/S module and HiPPI/D module and *Dst* process can be achieved via VME using different options.

4.3.2 Method of Measurement

The idea is to factorize the contribution to the latency coming from the synchronization level, the event descriptors and the actual data moving. The model is that the average latency depends on a constant overhead due to synchronization and event descriptors and a data size dependent part for the data moving:

$$L = \mathrm{overhead}(\mathrm{sync}) + \mathrm{overhead}(\mathrm{desc}) + \mathrm{time}(\mathrm{data}) \qquad \text{(EQ 4.1)}$$

$$\text{with} \quad \mathrm{time}(\mathrm{data}) \to 0 \quad \text{as} \quad \mathrm{size}(\mathrm{data}) \to 0$$

Thus the latency will be measured as the time an ideally zero-sized event fragment needs to go from source to destination. Actually it turns out that zero-sized event fragments are not allowed in the firmware. A better way to measure latency is to measure it for different event fragment sizes and then to extrapolate to a zero-sized event. This also has the advantage that since the overhead times are statistically distributed one gets a more stable and accurate value from the different measurements.

All measurements are done with the EB program running in a large loop of simple data transfers. Only pre-loaded events of constant size are transferred. The times are measured using the RAID timers [Spi94d]. The loops are repeated for different sizes from 1 word to up to 3 MByte. From repetitions of the measurements it can be seen that all rates are reproducible within ~2.5% for data sizes less than or equal to 4 kByte and within ~1% for bigger data sizes. The time for a single data transfer can be calculated from the measured rate as

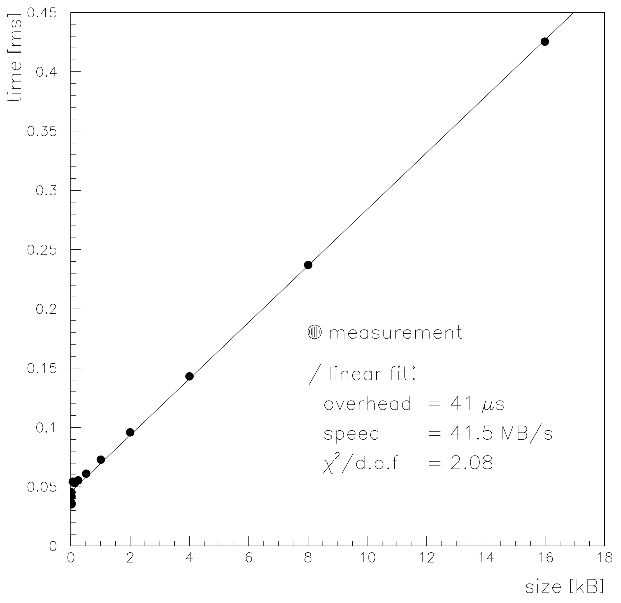
$$\mathrm{time}(\mathrm{data}) = \frac{\mathrm{size}(\mathrm{data})}{\mathrm{rate}} \qquad \text{(EQ 4.2)}$$

A simple linear model is used to describe the event size dependent part: the time to transfer one event is determined by an *overhead* and a link *speed*:

$$\mathrm{time} = \mathrm{overhead} + \frac{\mathrm{size}}{\mathrm{speed}} \qquad \text{(EQ 4.3)}$$

Figure 4.15 shows the time for single transfers from source to destination in dependence of the event fragment size. A good linear approximation can be seen.

FIGURE 4.15. Transfer Times of Simple Data Transfers

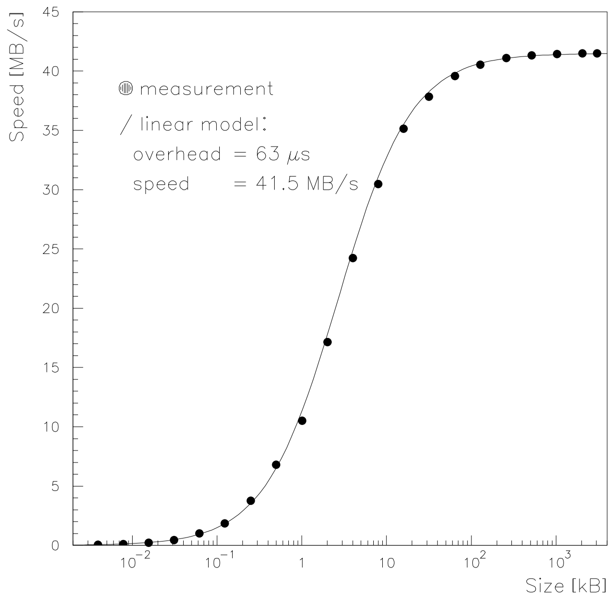


A χ^2-fit leads to a χ^2/d.o.f between 2 and 3. That this is not closer to 1 is due to the fact that some of the overhead times are not randomly distributed, e.g. the VME interrupt handling time depends on the situation in which the VME interrupt arrives. The transfer itself consists only of one HiPPI burst for data sizes less than or equal to 1 kByte, and only for large sizes are several bursts sent. Nevertheless, the linear approximation is good enough for an estimation of the latencies and individual overheads. Typical values of the overhead will be presented in section 4.3.4 and are usually between 10 and 100 μs. They have a reproducibility of about 1 μs. Applying the linear model, the data rate can be calculated as

$$\mathrm{rate} = \frac{\mathrm{size}}{\mathrm{overhead} + \frac{\mathrm{size}}{\mathrm{speed}}} \qquad \text{(EQ 4.4)}$$

For large packets the rate converges to *speed* while for very small packet sizes it goes proportional to *size/overhead*. Both parameters have an equal influence when the size is equal to *overhead*/*speed* which with the usual overhead values is the case for packet sizes of around 4 kByte. Figure 4.15 shows the data rate measured with the one fitted and the good agreement between these two.

FIGURE 4.16. Data Rate: Measurement and Linear Model



4.3.3 Minimum Latency

Different configurations and the method described above were used to measure the overhead times and resulting latencies.

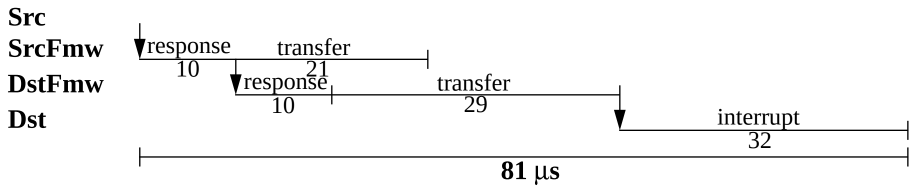
The influence of the switch was measured by comparing simple data transfers through the switch and through the cable only. No difference was found within a resolution of 1 µs, thus in accordance with the manufacturers specification. And there is no difference between source and logical addressing. Furthermore no degradation was observed when the switch was used for concurrent but exclusive connections.

The overhead from the *SrcFmw* was found to be 21 µs. This was measured using a dummy *Src* process and a dummy HiPPI/D module (NED). The overhead from the *DstFmw* for receiving an event is about 29 µs. This is slower than the *SrcFmw* can send the data because the HiPPI/D module has to check the parity and LLRC error flag and the size of data bursts which is not a-priori known, as for the *SrcFmw.*

Since the *SrcFmw* and *DstFmw* run in a loop and have to check various registers they have a limited response time of 10 µs in both cases, i.e the *SrcFmw* needs 10 µs to respond to a command and the *DstFmw* needs 10 µs to either respond to a command or a HiPPI transfer. An empty loop without any state changes takes about 6 µs. The cycle time is the shortest time between two consecutive HiPPI transfers and includes the loop of status polling and the HiPPI transfer. The cycle time on the *SrcFmw* is 33 µs, on the *DstFmw* 42 µs.

The minimum latency for simple data transfers between the HiPPI/S and HiPPI/D module is 49 µs. With a minimum interrupt handling time of 32 µs this means that the minimum latency for transfers between *Src* and *Dst* process is about 81 µs in total, which on a time graph looks as shown in figure 4.17.

FIGURE 4.17. Latency of Simple Data Transfers



Src and *Dst* process can run concurrently in which case the frequency of transfers is given by the frequency the *Src* process can send data. If the *Src* process waits for an interrupt signalling the end of a transfer the time between two transfers is 63 µs, corresponding to 15.9 kHz. The maximum frequency at which a HiPPI/S module can send event fragments is only given by the cycle time which corresponds to 30.3 kHz. The maximum frequency at which the HiPPI/D module can receive event fragments is 23.8 kHz.

4.3.4 Other Overheads

The latency measured so far was the minimum latency which comes from the HiPPI protocol, the switch and the necessary actions in the firmware to write or read data to or from the HiPPI interface. Other contributions to the latency in more realistic setups come from the different ways to handle the VME interrupts, the reading and exchange of event descriptors and other status variables, the event buffer management and event assembly.

In EP/LX there are different ways to attach a process to the arrival of an interrupt. The shortest one is the use of a semaphore which needs 32 µs between the VME interrupt is received and the program waiting on the interrupt carries out the next statement. Other possibilities of synchronization between the HiPPI modules and the EB program are polling a status variable on the HiPPI module or attaching an interrupt handler routine which will signal the process of the presence of an interrupt. Another method is using a unique signal queue [Fum95] in which all interrupts are put and read from sequentially by the EB program.

The exchange of the event descriptor between the firmware and the data flow processes takes around 2 µs. This is due to reading or writing single fields of the event descriptor in single word VME accesses and is dominated by the time to access the VME bus.

The buffer management contains the handling of event descriptors, taking them from a fifo or putting them into one. It takes somewhat different times in the firmware or in the data flow processes. The times for the event assembly in the *DstFmw* have been measured as above by single transfers merging *n* event fragments to a full event. The average times for different *n* are shown in figure 4.18a). After subtracting the cycle time for the *DstFmw* a linear dependence between the full time and *n* can be seen. This leads to 37 µs for setting up the event assembly and finally putting the full event back into the free event buffer, and a per-event time of 44 µs due to searching, chaining the event fragments and checking if the full event is complete. These times are roughly in agreement with the times for buffer management. All overheads contributing to the latency are summarized in table 4.1.

FIGURE 4.18. Overhead of Merging (see text)

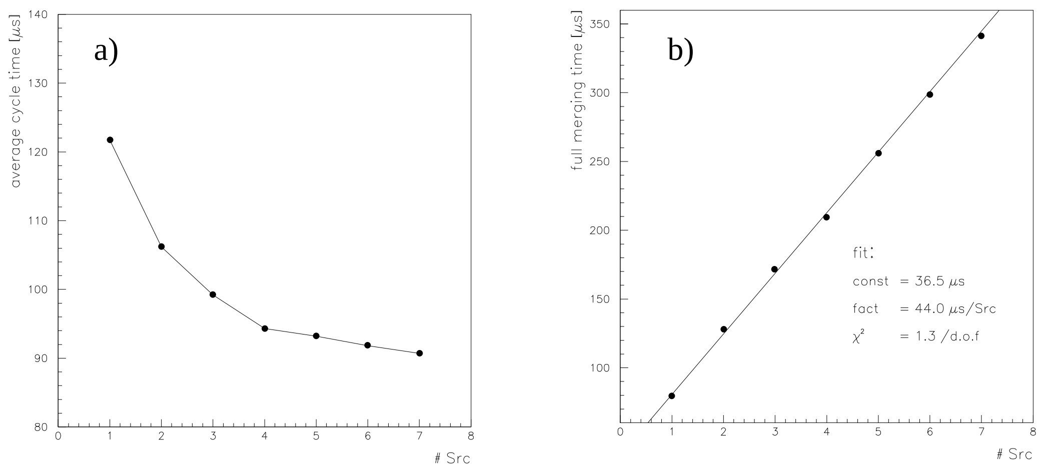


TABLE 4.1. Overhead Times (all values in µs)

Stack	Function	Overhead
Src	buffer management[a] event generation (Gaussian size) writing event descriptor	6 34 2
Src-SrcFmw	response time	10
SrcFmw	buffer management[b]	27
SrcFmw-HiPPI	sending data	21
HiPPI	HiPPI switch	<1
HiPPI-DstFmw	response time receiving data	10 29
DstFmw	buffer management[b] event merging	36 $\sim (37 + n \cdot 44) / n$ [c]
DstFmw-Dst	polling status semaphore attach routine signal queue	30 32 36 76
Dst	buffer management[a] reading event descriptor	6 2

a. either getting an event descriptor from a fifo or putting one back into a fifo
b. getting an event descriptor from a fifo and putting it back into a fifo in the end
c. average overhead per event, merging n events

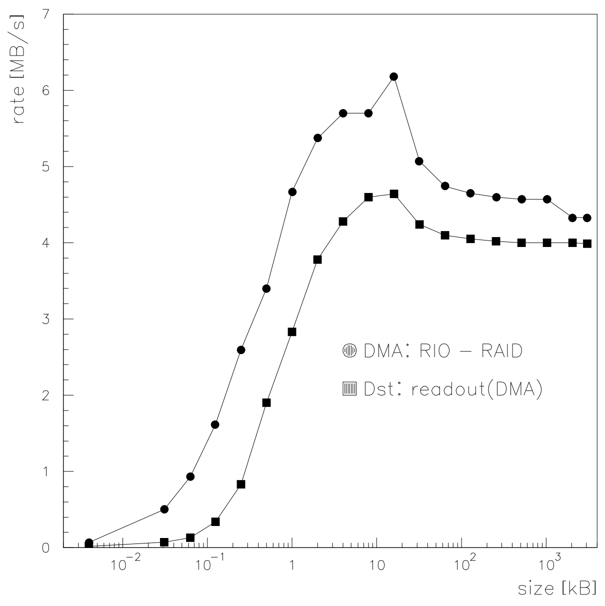
4.3.5 Data Moving

Data moving between the HiPPI modules via the HiPPI cable and through the switch was in all configurations compatible with a linear model and the transfer speed was measured to be 41.5 MB/s. This value is not limited by the HiPPI protocol which should allow 100 MB/s but rather limited by the RIO system bus which transfers data at this rate between the local memory and the HiPPI interface and represents 83% of the design maximum data rate of this bus.

Moving the actual data between the VME-HiPPI interfaces and the RAID processor was only tested for the *Dst* process. This was considered to be a test to check data errors in the HiPPI transfer and to test the principle rather than a realistic data flow chain. This is because the transfer between the HiPPI modules and the RAID processor is not optimized. Nevertheless, different possibilities were tested:

- VME D32:
  The simplest transfer is the single word access on VME bus where each single word of the actual data is transferred independently.
- VME DMA:
  Another method is using the DMA device of the RAID processor to move blocks of data over the VME bus.
- RIO D32:
  The HiPPI Module has a VME master interface which can be used to write words from local memory into the RAID processors memory.
- RIO fly-by:
  The RIO module VME master interface has a block move mode [RIO91].

FIGURE 4.19. Data Moving using DMA



The data rate for the VME DMA method is shown in figure 4.19 compared to a pure DMA test. One can clearly see that the data rate cannot be expressed any longer in a linear model. This is due to the fact that the DMA was optimized for packets of 16 kByte and uses chaining of blocks for bigger transfers [Buo93b]. The peak transfer rates of all the different methods are summarized in table 4.2.

TABLE 4.2. Peak Rates of Data Moving

Method	Maximum Data Rate (MB/s)
VME D32	~1.6
VME DMA	~4.6
RIO D32	~1.1
RIO fly-by	-[a]

a. fly-by could not be tested

4.4 Parallel Event Building

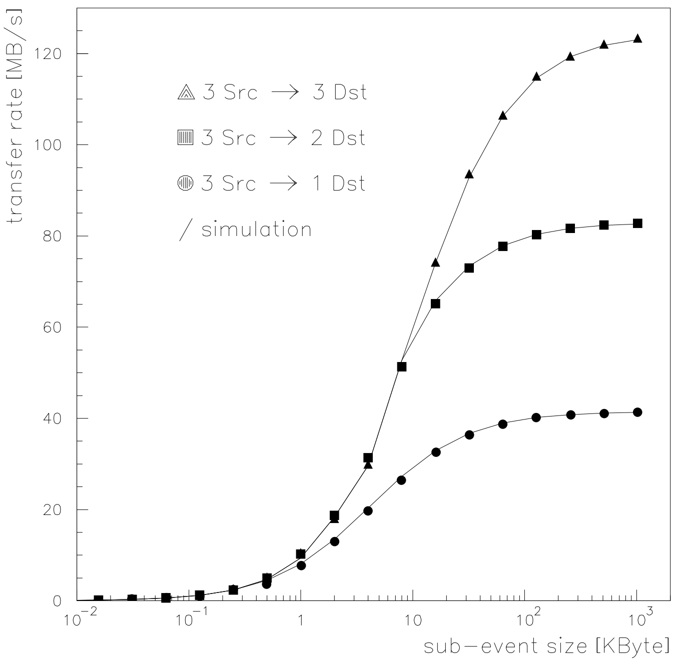
The parallel event building system was used to measure the performance of several destinations running in parallel. The effect of waiting for the interconnecting network to become available can in particular be seen in the scaling behaviour with the number of destinations. The simulation was used to find realistic parameters for the data flow processes which are combinations of the overheads discussed in the previous section. Different parameters like event fragment size distribution and the different data flow management schemes were investigated.

4.4.1 Fixed Event Sizes

Figure 4.20 shows the measured maximum throughput of a parallel event building system with three sources to a varying number of destinations with increasing event fragment size. This figure first reveals that the parallel event building system runs fine and produces good measurements. It furthermore shows the good scaling behaviour for event fragment sizes above 10 kByte. This means that a system with *n* destinations has *n*-times the performance of a system with one destination. In other words, event building is performed in parallel using several links of the interconnecting network concurrently if the event fragment sizes are fixed and equal in all sources. This scalability, however, is limited for small event fragment sizes only by the fact that all data flow processes run on a single processor board.

The figure also shows the good agreement between the measurements and the simulations which were carried out with the simulation program (see section 5.4). This model uses in total 4 parameters. It was discovered that a single parameter to describe the latency is not enough but that the complexity can be expressed in two parameters for the firmware and two parameters for the data flow processes. The firmware parameters were taken from the simple data transfers (see section 4.3.4) and describe the time which the *Src* process needs to send the event fragment and the time the *Dst* process needs to merge the event fragment. The two parameters for the data flow process describe the time needed to handle an interrupt and the time to generate new events. The values have been chosen to be in good agreement with the measurements and to be unique for the whole set of measurements in different configurations. They are shown in table 4.3.

FIGURE 4.20. Performance of the Event Building Prototype



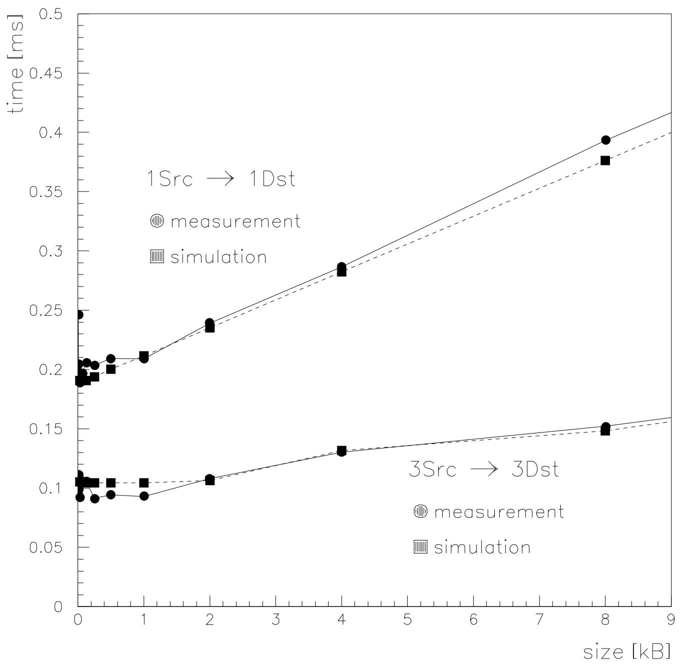
There is only one time for the interrupt handling and the time for event generation follows a linear law with 76 µs overhead and 34 µs per source. These times are combined from the values of the overhead times measured in the simple data transfers (see section 4.3.4) and reflect a realistic program. A more detailed simulation would require a complex parametrization of the operating system and of the behaviour of the data flow processes. But these two parameters are fully sufficient as can be seen in a comparison of the transfer times for small event fragment sizes shown in figure 4.21. The differences between measurement and simulation are less than 5% in most cases.

TABLE 4.3. Simulation Parameters for the *PUSH* scheme (all values in µs)

#Src	Interrupt	Generation	SrcFmw	DstFmw
1	45	110	33	90
2	45	144	33	72
3	45	178	33	65

A detailed tracing of the EB program and analysis of the simulation program reveal that the parallel event building system with fixed event fragment sizes turns itself into a barrel shifter mode (see section 3.2.3). Each transfer takes the same time after which the status of the switch is changed and the next fragments can be sent. This apart from the interrupt handling time keeps all lines busy and assures maximum parallelism. The different switch states are repeated in cyclic order and the buffer occupancy grows with the position of the source in this cycle. The last source must be capable of holding at least as many event fragments as there are sources in the event building system.

FIGURE 4.21. Transfer Times: Comparison between Measurement and Data



4.4.2 Event Size Variations

So far, fixed event fragment size has been used and all transfers take the same time. Of course this will change in more realistic conditions when the event fragment size varies from event to event and from source to source. Due to these fluctuations some sources have to wait for others and the parallelism is disturbed, the maximum throughput decreases.

Figure 4.22 shows this influence for different event models: independent Gaussian with different σ_{rel} or independent exponential event fragment size distributions. The exponential case can be regarded as some sort of worst case due to the long tails in the distribution and the higher probability of having big differences within one event. In a 2×2 setup the maximum throughput for exponential fragment sizes is reduced to 68% compared to fixed event fragment sizes.

The influence of correlated event fragment sizes has been studied with some pattern of Gaussian (σ_{rel} = 10%) distributed event fragment sizes and a fraction r of pathological events. A pathological event has one event fragment being 10 time bigger than the others but the average full event size is the same as before. If r equals 100% then the throughput is only slightly better than exponentially distributed event fragments sizes as can be seen in figure 4.23. The maximum throughput of a 2×2 setup is in this case 71% compared to fixed the event fragment sizes.

FIGURE 4.22. Influence of Event Fragment Size Variations

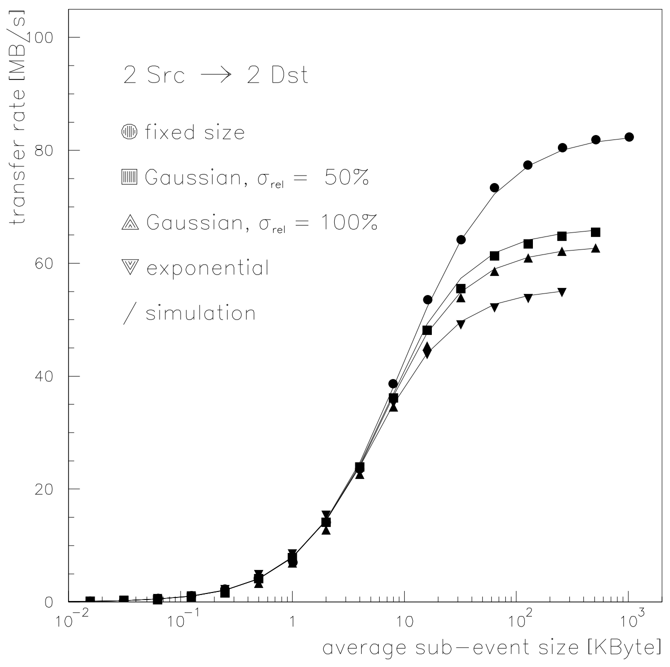
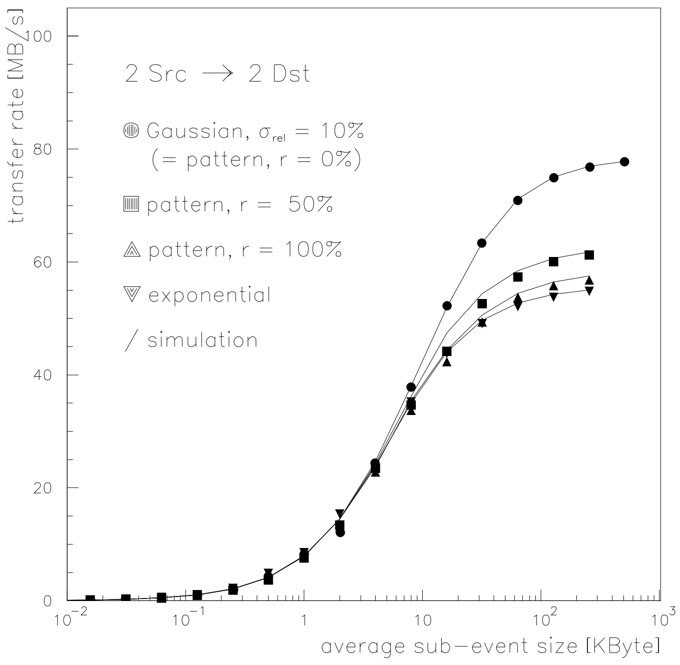


FIGURE 4.23. Event Fragment Size Correlations (see text)

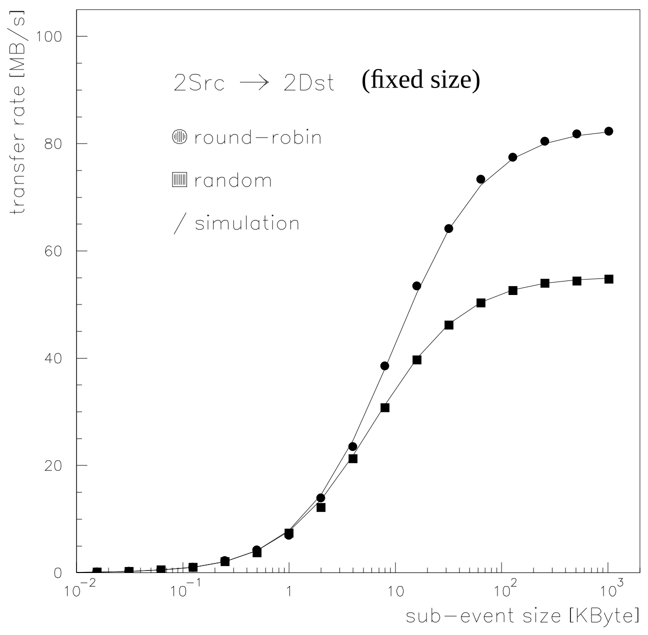


For both types of event fragment size variation the agreement between measurement and simulation is very good and differences in the two are less than 5%.

4.4.3 Destination Assignment

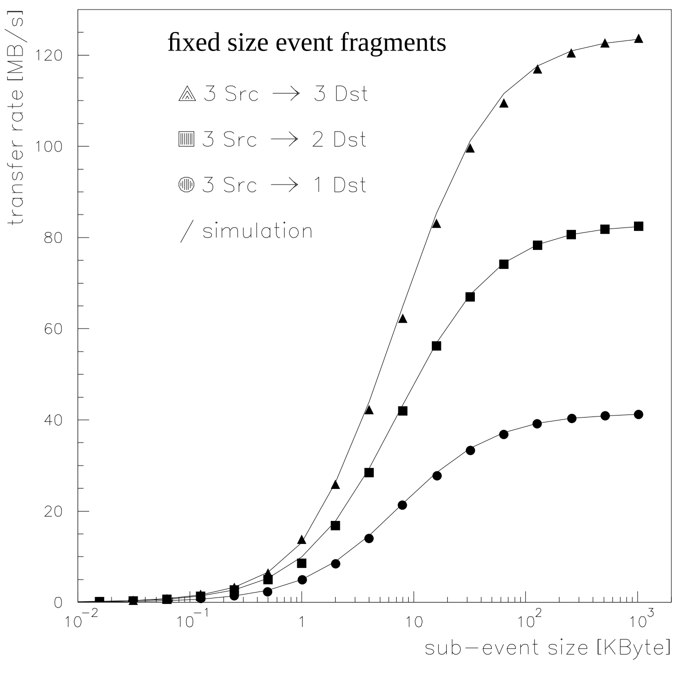
In the previous measurements only the *PUSH* scheme with a round-robin destination assignment has been used. Assigning the destination randomly in a 2×2 setup is more inefficient and the maximum throughput is only about 66% compared to round-robin, as can be seen in figure 4.24. This decrease in performance is due to the fact that one particular destination can be assigned to several full events in sequence which reduces the parallelism and thus the maximum throughput.

FIGURE 4.24. Random Destination Assignment



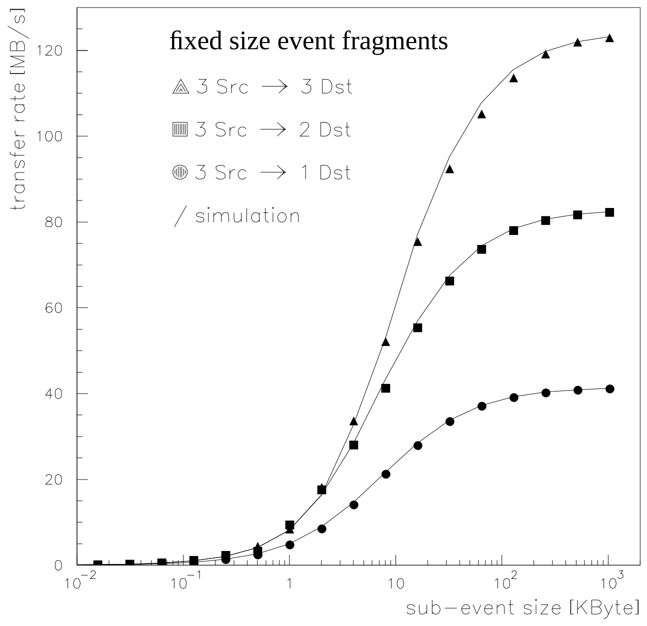
Figures 4.25 and 4.26 show the performance for the *PULL* and *SYNC* schemes. Good scalability can be seen which in the case of *PULL* is even valid for smaller event fragment sizes down to about 1 kByte.

FIGURE 4.25. Event Building Performance, *PULL* Scheme



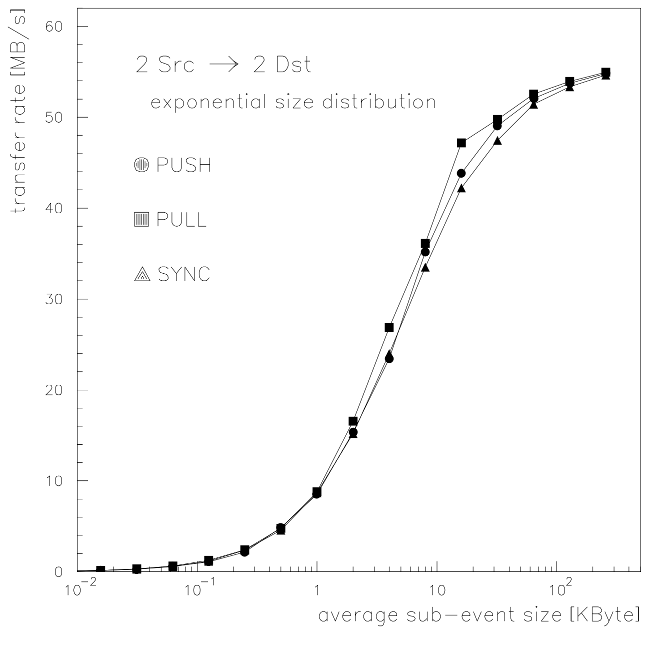
The simulations were carried out with the same parameters as before (see table 4.3). For the *PULL* scheme is has been taken into account that the event generation is always done after an interrupt and the two times have been added. In the *SYNC* scheme one interrupt signals the end of a transfer and another one signals the generation of a new event. Interrupt time and event generation time have to be added. The agreement between measurement and simulation is good in both cases.

FIGURE 4.26. Event Building Performance, *SYNC* Scheme



A comparison of all three schemes for a 2×2 setup and individual exponential event fragment size distributions is shown in figure 4.27.

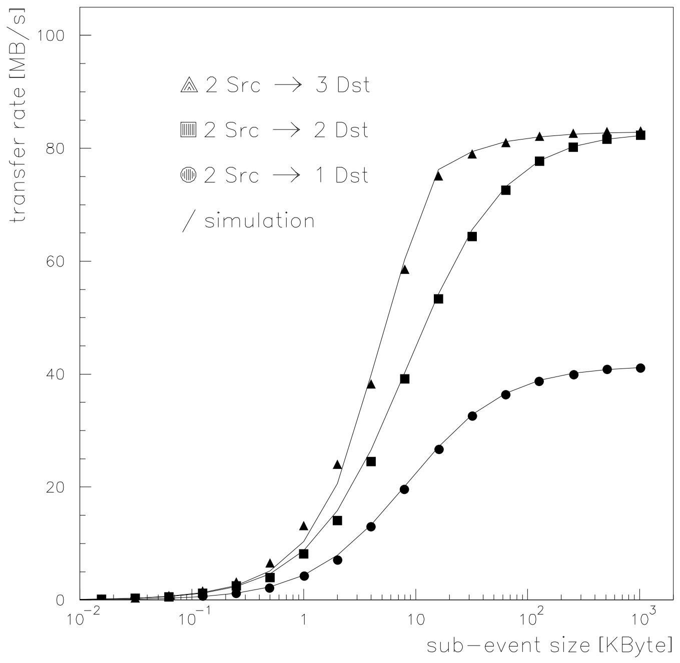
FIGURE 4.27. Comparison of Different Destination Assignment Schemes



No essential differences can be seen for the 2×2 setup and the little differences there are can be explained by the fact that the data flow processes have to deal with a different number of interrupts. For this setup each event creates only one interrupt in the *PULL* scheme, 2 in the *PUSH* and 3 in the *SYNC* scheme.

Figure 4.28 shows a setup of two sources with a varying number of destinations using the *PULL* scheme. Interesting, here, is that three destinations have a higher performance than two. This is due to the fact that a HiPPI/S module can send the next event fragment after 33 μs while the HiPPI/D module will still be busy for another 72 μs to merge the event. So, if there are more destinations than sources, the merging process of the previous event fragment and the sending of the next event fragment can be done concurrently. The maximum throughput is increased for event fragment sizes between 1 and 200 kByte. This covers also the event fragments sizes expected in ATLAS event building system (see section 2.4.5).

FIGURE 4.28. Event Building Performance, *PULL* Scheme



4.4.4 Fault Tolerance

Errors in the event building prototype can occur on the level of control and data signals, the level of data bursts and connections and on the level of the data flow management. They can have different sources and different detection mechanisms have been implemented as well as different measures to recover from an error condition.

Control signals were transported via VME interrupts. In normal operation there were no problems. But with other users in the same crate there could be clashes on the VME bus leading to bus errors. These stopped the whole crate and had to be recovered by re-setting the crate. Furthermore, the multitude of VME interrupts from the VME-HiPPI interfaces could only be handled by using different interrupt levels because otherwise some modules would not be handled at all due to the bus arbitration. The internal signals in the EB program were handled by the EP/LX and shared memory which were stable and safe.

The HiPPI protocol detects transmission errors by a byte-to-byte parity check and one LLRC word per burst. These checks are carried out in hardware and not a single error was detected in the estimated 10^{12} Bytes sent in total. The software in any case would have tagged the event fragment and have continued without any loss in performance. No violations of the HiPPI protocol on the burst level were detected.

The protection against errors on the connection level consists of time-outs which have to be longer than the longest full event transfer times (around 1 s for 1 MB events!). Such errors could be emulated by switching off a VME-HiPPI interface or by re-configuring the switch. In such cases a retry worked well but was quite slow due to the long time-outs. Anyway, errors of this kind can be detected easily as they lead to dedicated messages in the firmware or can even be read from the front panel of the VME-HiPPI interfaces.

On the data flow level the biggest problem was the fact that the data flow processes were single-threaded. No status information is available while they are waiting for a signal from the VME-HiPPI interfaces. Future data flow processes should be multi-threaded to allow status requests even while waiting for events. Stopped *Src* processes were in principle no problem as the others carry on sending event fragments. Some algorithm nevertheless has to make sure that the *Dst* processes will detect the missing *Src* process. Otherwise the build event buffer (*BvtBuf*) will overflow with half built events and no more full events will be sent out. Stopped *Dst* processes were no problem for *PULL* and *SYNC* schemes as they just stop sending requests and lose probably one event. The *PUSH* scheme, however, will block on the stopped *Dst* process and only after some time-out finally skip the dead *Dst* process and carry on, probably having lost some events due to overflowing buffers or having introduced some deadtime.

In general the whole setup can be deemed as rather robust. Error detection and recovery on several level was provided by the hardware (HiPPI and VME) and software (EP/LX, firmware, data flow processes). While some parts were already there, others had to be added to make the event building system safe. Very rarely a complete failure due to cabling problems could be detected rather quickly and re-set fast.

4.5 Conclusions

A parallel event building prototype system based on a commercially available technology has been operated successfully. High speed interconnects using the HiPPI standard and a switch have been integrated in a VME environment. The system which uses HiPPI for fast data transfers and VME for control flow was integrated with an extendable and modular software. This consists of the data flow processes and of a three-layered data flow management: one layer is technology dependent while the other two can be used with other high speed interconnect standards.

Though the prototype shows in principle successful operation of a parallel event building system using commercially available technology based on high speed interconnects and switching elements, more studies with bigger prototypes in realistic environments are needed to conclude on the feasibility of this technique for event building systems of experiments at the LHC. The main significance of this prototype, however, is that it provides a parametrized model of behaviour in terms of overheads and link speed. This model will be used for large-scale simulations of event building systems like needed in the ATLAS experiment. The model will likewise be valid for similar systems using connection-oriented networks and crossbar switches (e.g. Fibre Channel class 1).

A data flow management has been developed. It was successfully used to adapt a particular technology to the requirements of a parallel event building system. The generic levels of the data flow management contain event format, buffer management and event assembly which could also be used in future event building systems and which could partly migrate to hardware. The real-time UNIX operating system EP/LX has proven to be a very good tool for signal handling as well as a development environment.

Simple data transfers from one source to one destination were used to measure the minimum latency to be 49 µs. This is mainly due to the HiPPI protocol, the switch itself contributes with less than 1 µs. The interrupt handling could be measured to be minimally 32 µs, so that the latency between *Src* process and *Dst* process is 81 µs in total. The maximum frequency for sending events is 30.3 kHz, and for receiving events 23.8 kHz. The link speed is 41.5 MB/s and the transfer time can be described in a linear model using a link speed and overhead. For small numbers the performance of the parallel event building system scales with the number of destinations for event fragments size bigger than 1 to 10 kByte. The system has mechanisms for error detection and recovery and runs safely.

The measurements from the prototype could be compared to a simulation program which uses only a few parameters for link speed and overheads of about 100 µs. The agreement of prototype measurements and simulations with different event fragment sizes and different data flow management schemes is good and there is confidence in the model which will be used with the same parameters for large-scale simulations of ATLAS-like systems.

The prototype, however, had some limitations: the EB program is only running on one processor board which leads to performance restrictions especially for small event fragment sizes. This cannot be regarded as realistic because future parallel event building systems must use several boards. Unfortunately, the system was not big enough to test more realistic event fragments size distributions as expected from physics with their special correlations. Running under realistic data taking conditions in an environment with several sources and destinations, changing running conditions and unforeseen pathological situations will have to be tested to draw conclusions on the scalability of the system.

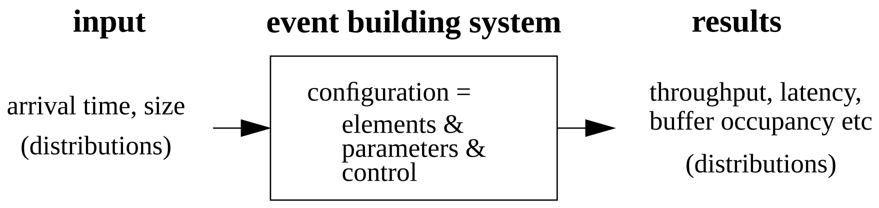
Nevertheless, the prototype is the first one of this kind working and it can be hoped that the next-generation prototypes will go beyond it in performance and complexity. This, at the moment, can only be achieved by means of simulations which are based on and verified with the measurements from this prototype.

5.0 Simulation Program

Simulations are used to study parallel event building systems on the scale needed for the experiments at the LHC. They are used to compare and evaluate different architectures, technologies and data flow management schemes before the systems are actually built. Simulations are complementary to building prototypes in the sense that the simulation models are validated with the help of prototype measurements before they are extrapolated to scaled-up setups.

The simulation model of a generic event building system (see section 3.1) is described in terms of a configuration, an input data stream and a set of performance indices. The configuration of an event building system describes its architecture, the data flow management and the timing parameters for data transfer, processing and buffering. The input data stream is described in terms of distributions for the arrival time and for the size of event fragments. These distributions can be based on simple parametrizations using random number generators or come directly from the off-line simulations of the detector. The performance indices are the result of the simulation.

FIGURE 5.1. Simulation of Event Building Systems



5.1 Little’s Law

Although an event building system can in principal be described as a network of queues the details of the interconnecting network and the interdependence of the queues are difficult to express in mathematical probability functions. The approach of queueing theory is therefore not followed here and only one result from queueing theory will be reported: Little’s law. This states [Kle75]:

The average number of customers in a queueing system N is equal to the average arrival rate of customers to that system λ, times the average time spent in that system T:

$$N = \lambda \cdot T$$

This law which can be understood intuitively is a formal result of queueing theory and does not depend on any specific assumptions regarding the arrival time distribution, the service time distribution, the number of servers or the queueing discipline. For an event building system it relates the number of full events N_{Fvt} to the latency L:

$$\langle N_{Fvt} \rangle = \langle f \rangle \cdot \langle L \rangle \qquad \text{(EQ 5.1)}$$

where f is the input frequency (see section 3.1.3). Little’s law can also be applied for the event fragments:

$$\langle B \rangle = \langle f \rangle \cdot \langle L_{Frg} \rangle \qquad \text{(EQ 5.2)}$$

Using the relation of latency and fragment latency (equation 3.3) and Little's law in the two forms one can derive a relation between buffer occupancy and latency:

$$\langle B \rangle \leq \langle f \rangle \cdot \langle L \rangle \qquad \text{(EQ 5.3)}$$

5.2 Discrete Event Simulation

A simpler description of event building systems can be made using discrete-event simulations which were utilized throughout this work.

5.2.1 Definition

Discrete-event simulations are based on a model that describes the system in terms of logical relationships and state changes of the elements at certain points in time. The whole simulation can be carried out as a sequence of events, with an event being a certain point in time when an element changes its state.

The events are ordered by simulation time which is not to be confused with the wall clock time. The events are scheduled in a time-ordered list. At any instant of simulation time there can be multiple concurrent events and any event can cause other events to be scheduled or de-scheduled. The simulation is carried out by effecting all necessary state changes at a given simulation time and after that, advancing the simulation time to the next event pending on the list.

From the point of view of formulating the model, there exist two different approaches. In the event-oriented approach the programmer must define the events in time which are not allowed to spend time themselves. The list of events is directly accessed. In the process-oriented approach the programmer has to define processes which contain at least two different events separated in simulation time. Thus the process becomes a time spending action. The user has no direct influence on what is put on the list of events but is concerned about how long an action takes and how it interacts with others. Both approaches are equivalent in terms of the simulations. The element of random is introduced in discrete-event simulations by use of random times for processes. The random times are sampled from random number generators.

Usually the simulations are carried out with the help of a simulation language which contains syntactical elements to describe the model and its actions, a compiler that compiles the description into computer executable code and some runtime facilities that allow the handling of events in a more or less comfortable way. Examples of such languages are Verilog [VER90], VHDL [VHD87], Simscript [SIM90] and MODSIM II [MOD91]. Of course, high level programming languages such as FORTRAN,C or C++ [Ver93] can be used to do discrete-event simulations.

5.2.2 MODSIM II

MODSIM II [MOD91] is a proprietary, general purpose, high level language for discrete-event simulations. It has a modular structure, is block oriented and is based on MODULA 2 with object-oriented features like encapsulation and inheritance.

In addition it has syntactic elements to describe time elapsing methods (WAIT DURATION). It is process-oriented with the processes containing one or more of these statements. The processes can be started synchronously (WAIT FOR) or asynchronously (TELL) in which case several processes can be active at the same time. Arbitrary synchronization between processes can be achieved with the help of a trigger object. Scheduled activities can be interrupted and the reaction of an interrupted action can be defined.

MODSIM II has a library of predefined objects like the already mentioned trigger object and a resource object which is used for simulation of the basic case of processes competing for a limited-capacity object. The library also contains various list objects and various random number generators. It also provides I/O facilities for the reading of parameters and the printing of results. MODSIM II has a script which only re-compiles the modules that actually need compilation and links them with the runtime scheduler which is the heart of the discrete-event simulation. It can also run a debugger and be interfaced with a graphics tool for graphical objects and animation.

It is used in the HEP community for simulations of data acquisition systems. In the ATLAS collaboration it was chosen to be used for simulations of the functional model of the ATLAS data acquisition system and is the basis for several detailed studies on different technologies [Hun95]. It is used throughout this work.

5.3 DAQ Simulation Library (DSL)

The DAQ Simulation Library [Spi93][Amb94b] is a high level tool based on MODSIM II. The DSL (with the graphical user interface and the tracing facility) is a high-level description language for simulations of DAQ systems. It can be used for simulations of any kind of DAQ system and has the possibility to include lower level hardware descriptions. The GUI allows an easy configuration, initialization and on-line monitoring of a simulation program. The tracing facility allows a highly flexible analysis of the output.

5.3.1 Elements

The DAQ Simulation Library consists of generic objects to describe any kind of DAQ system. The basic elements are:

- **Items** are information carrying data accumulations that are passed in a DAQ system, e.g. event data, trigger signals.
- **Processes** are the active objects in a DAQ system passing items and acting on them, e.g. readout or recording process.
- **Resources** are the limiting factors the processes have to compete for in order to fulfill their task, e.g. cpu, buffer, transfer media.
- **Control** elements are abstract objects controlling the processes and carrying information on the data flow, e.g. timers, allocation algorithms.

The main idea of the DSL is to use the smallest indivisible (**“atomic”**) processes that can then be used to build up any DAQ system. A dozen **“atomic”** processes have been defined and constitute the core of the DSL.

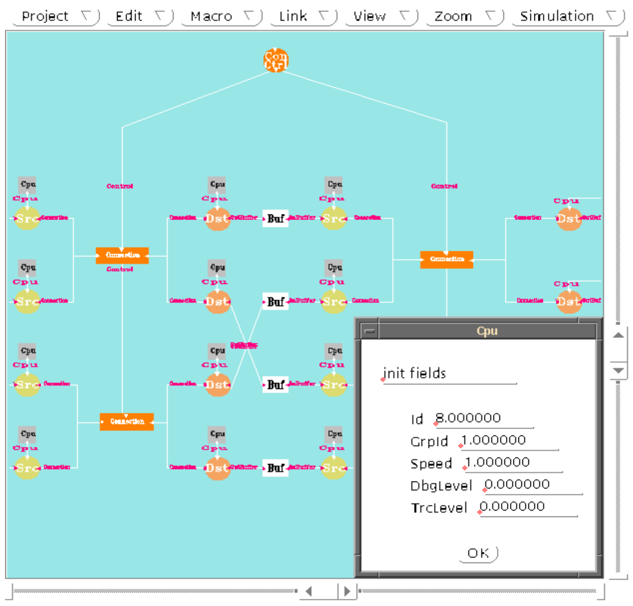
The DSL has a generic level consisting of objects for a generic description of DAQ systems, and a user level where inheritance is used to combine the generic objects with user dependent features. Thus the DSL allows the possibility to refine the objects and to include hardware dependent features.

5.3.2 Graphical User Interface

A graphical user interface [Dji94a] based on the graphical objects in MODSIM II is used to easily configure the simulation model, to initialize each object and to monitor parameters on-line. The GUI has three windows:

- the **library window** displays the objects of the DSL;
- the **configuration window** is a canvas on which the configuration to be simulated is built;
- the **display window** monitors parameters while running the program.

FIGURE 5.2. Configuration Window with an Example



Cpu

init fields

Id 8.000000
GrpId 1.000000
Speed 1.000000
DbgLevel 0.000000
TrcLevel 0.000000

Figure 5.2 shows an example of the configuration window with a part of data acquisition system and the input window for the parameters of one of the elements. Additional features are available

- for saving and reloading whole configurations and their initialization values;
- for grouping of objects (very useful for copying parts of the configuration);
- for organizing views in a hierarchial way (very useful for complex configurations).

While the GUI can be used to build a configuration and to debug it, there is also a fast version available which can be used to run the program without graphics, thus increasing the performance for time consuming simulations.

5.3.3 Tracing Facility

The tracing facility is a tool that allows each single “atomic” process to report on its activity by writing a trace record in a file. This facility can be switched on and off for each individual process. The format of the trace record can be extended by the user.

The trace file can have binary or ascii format and can be processed off-line (i.e. after running the simulation) by a tool which is implemented as a C program. This tool can:

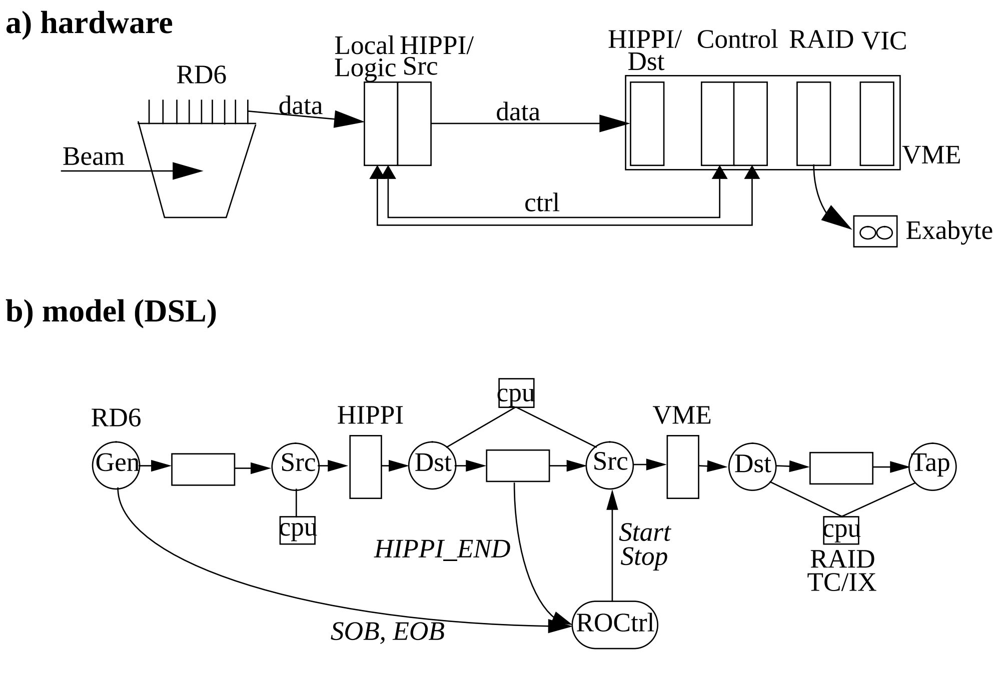
- reproduce each individual trace record;
- produce general statistics, e.g. number of events generated, size of the events, etc.;
- produce statistics on each type of trace record, e.g. event generation frequency, buffer usage over time, etc.;
- can order the records on an event-by-event basis, e.g. latencies, lifetime of event, etc.

The results of the various analyses are written in *ntuple* format and can be visualized with the help of *PAW* [Bru93].

5.3.4 Example

As an example of an application of the DSL the readout of the combined RD6/RD13 testbeam in November 1993 has been simulated [Spi94a]. This setup (figure 5.3) consisted of a single chain of data flow using a HIPPI link and had a total data rate of 1.5 MB/s. This example was used as a proof of principle: it showed the easy mapping between reality and simulation and the consistency between the values measured and the values simulated. The simulation could then be used for changes of parameters and extensions of the setup.

FIGURE 5.3. RD6/RD13 Testbeam Setup



The DSL has been successfully used for simple examples. It was used for simulations of functional models of the whole ATLAS readout, LVL-2 triggering, event building and LVL-3 triggering [Koz94]. Many of the ideas of DSL have been influencing the ATLAS simulation program which is also MODSIM II based [Hun95].

5.4 Simulation Program for Event Building Systems (SIMEB)

SIMEB is a program for simulations of event building systems[1]. It uses a subset of simplified DSL objects. The simplifications of the objects were applied in two ways: first, the high-level hierarchy of inheritance was moved into a single level and the objects are no longer inherited from abstract objects. Second, some objects were grouped together in order to facilitate the communication between them and have less TELL and WAIT statements.

The simulation program implements a model of a generic event building system and can simulate different input event fragment size distributions and different data flow management schemes. It is fully configurable in the number of sources and destinations as well as their parameters. The DSL graphical user interface is not used and the program therefore performs better. The DSL tracing facility is replaced by collecting dedicated statistics of buffer occupancies and latencies.

5.4.1 Elements

The simulation program implements the following objects:

- The ***EvtFragmentObj*** represents the event fragment. It contains the event descriptor with identifiers, information concerning the size and kind of data and concerning routing. This object also has some timing information for collection of statistics.

  The *EvtFragmentObjs* are actually not allocated and de-allocated at run-time. A pool of event fragments with pre-allocated objects is used instead to improve the performance.

- The ***EvtGeneratorObj*** is the motor of the simulation program. It generates event fragments and sends them to the *SrcProcObjs*. The time between two consecutive events can be fixed, exponentially distributed or at maximum rate in which case a new event is created whenever one is needed (see table 5.1). The event fragment size can be fixed, exponentially or Gaussian distributed. Dedicated patterns can be generated (see section 6.3.3) or the distributions can be read from a file coming from off-line simulations of the detector. In case of a static destination assignment scheme this is also done in the *EvtGeneratorObj*.

- The ***SrcProcObj*** implements the *Src* process (see section 3.4.1) of an event building system. It has a buffer to store incoming event fragments, has a connection to the *DataFlowMgmtObj* to receive destination requests and/or information about the destination status and an outgoing connection to the interconnecting network. This part, called the source port, can have a buffer for event fragments queued to the destinations and connects directly to the destination port of the *DstProcObj*.

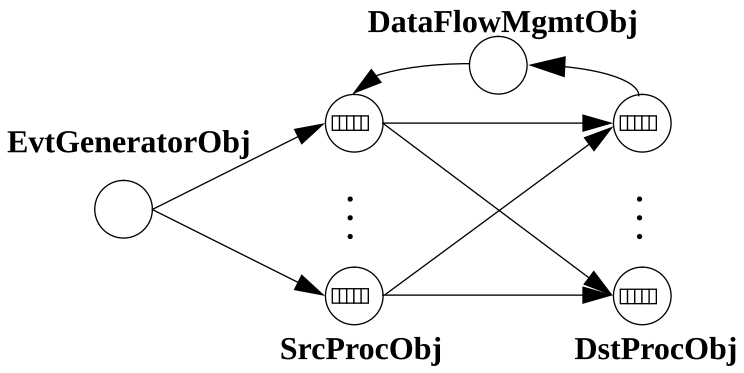
1. The first version of this program was written by W. Greiman [Gre94].

- The ***DstProcObj*** implements the *Dst* process (see section 3.4.1) of an event building system. It has an incoming connection from the interconnecting network which receives connection requests and data from the *SrcProcObjs* and implements the link model and the arbitration mechanism. There are no additional switching objects between *SrcProc-Obj* and *DstProcObj* in order to minimize the number of objects in the simulation. In addition to the destination port the *DstProcObj* has a buffer for the event assembly and a connection to the *DataFlowMgmtObj* to send requests or status change information. An outgoing connection is not modelled. The fully built events are put directly back into the pool of events.

  SrcProcObj and *DstProcObj* are composed objects. They have a cpu object to spend time for their various actions. They are a realistic model of the corresponding processes in the prototype (see section 4.4.1).

- The ***DataFlowMgmtObj*** implements the data flow management scheme (see section 3.4.3). It connects to the *SrcProcObjs* and *DstProcObjs* and manages the destination assignment.

FIGURE 5.4. Elements of the SIMEB Program



5.4.2 Configuration

The simulation program can be configured in two steps: at compilation time and at run-time.

TABLE 5.1. Options for the Pre-Compilation of the Simulation Program

Mode	Option	Meaning
destination assignment	PUSH	source-initiated
	PULL	destination-initiated, queue event fragments on source port
	SYNC	destination-initiated, synchronize with source port
	DFM	use data flow manager (see section 7.2.3)
cpu	CPU	*SrcProcObjs* and *DstProcObjs* have cpu and spend time
	PROTO	all *SrcProcObj* and *DstProcObj* share one single cpu
event generation	MAXRATE	generate event fragments whenever needed

A pre-compilation step is used to specify the destination assignment scheme, the cpu and event generation mode. Table 5.1 shows the possible options. A little *make* script (UNIX) will run the C pre-processor on the source code and generate the MODSIM II code which will then be compiled and linked to an executable program. The executable program can read parameters from the command line or from a file. It uses flags for the parameters some of which are shown in table 5.2.

TABLE 5.2. Some Input Parameters of the Simulation Program

Flag	Input Parameter
-i	number of *SrcProcObjs*
-o	number of *DstProcObjs*
-r	input frequency
-e	average event fragment size
-l	link speed
-d	link dead time (due to firmware)
-p	processing time for interrupt because of new event
-q	processing time for interrupt at end of transfer
-t	processing time for merging event fragments

5.4.3 Collection of Statistics

The collection of statistics is done in the *EvtGeneratorObj*, the *SrcProcObj* and the *Dst-ProcObj*. The values reported at the end of the program are the following:

- the total simulation time, the total amount of data transferred and the total throughput achieved;
- the latency of full events and the latencies of event fragments in the individual buffers of the *SrcProcObjs*;
- the time weighted average and maximum number of full events in the event building system;
- the time weighted average and maximum buffer occupancies in the *SrcProcObjs*, expressed in number of event fragments as well as in size.

A little *awk* script (UNIX) can be used to extract the numbers and to prepare them for input with *PAW* [Bru93] which can then be used to present the different results from simu-lation runs with different parameters.

6.0 Simulation of Event Building Systems

The SIMEB program presented in the previous section (5.4) is used to simulate large-scale systems based on a realistic model of the ATLAS event building system as presented in section 2.4.5.

In the data acquisition system of the ATLAS detector, the ROBs together with the LVL3-links of the readout crates act as sources. The switch-farm interfaces are the destinations. The model presented in this section is based on the assumption that the event building system is completely decoupled from the up-stream and down-stream data acquisition system. This implies that buffers are large enough or equivalently that trigger processing of events is fast enough not to require any feedback or throttling of the input. The model of the switch and the parameters from the prototype measurements are used for the simulation, notably the fact that the switching delay is negligible and that the transfer time can be described in terms of an overhead and a link speed.

All simulations presented in the following section were performed with the SIMEB program, using 100 sources and destinations and an average full event size of 1 MByte. The input frequency varied between 0.5 and 4.0 kHz. Realistic parameters for the interconnecting network were taken from the event building prototype (see section 4.3.4): the transfer speed was assumed to be 40 MB/s and the overhead of the transfers 100 µs. The default contention resolution in the switch is made in a round-robin manner and the default destination assignment scheme is a *PUSH* algorithm like in the prototype with the destinations being assigned in a round-robin manner.

6.1 Simple Model

First, a simple model is used to investigate the influence of the different parameters. This model consists of a 100×100 setup with an event fragment size of 10 kByte per source. The arrival time is on average 1 ms, corresponding to a frequency of 1 kHz. It can either be constant or exponentially distributed, leading to a Poisson distributed number of events in a given time interval.

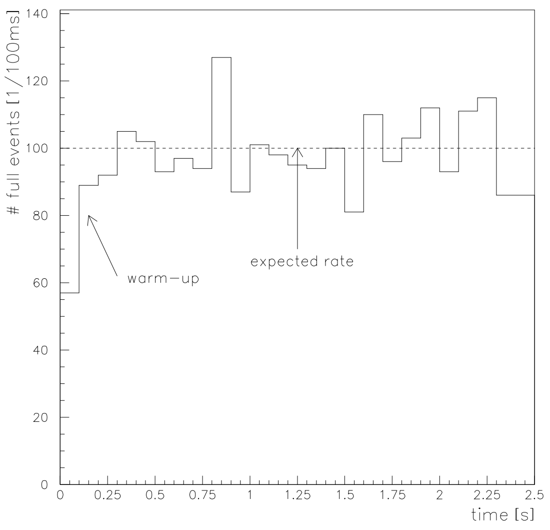
6.1.1 Warm-up and Convergence

Before doing any simulations, the simple model was used to investigate the convergence of the results: an important feature in discrete-event simulations of queueing systems is the fact that the queues are usually empty in the initial state of the simulations. When the simulations are started the queues fill up until they reach a certain occupancy depending on the system's throughput and the statistical fluctuations. This, of course, is only the case if the system has a stable state, otherwise the queues will grow indefinitely.

The phenomenon of filling queues at the beginning of the simulations is called "warm-up" and has to be taken into account when interpreting the results of a simulation. Figure 6.1 shows the evolution of the full event frequency shortly after the start of the simulation of the simple model with exponential size distributions and an exponential arrival time distribution. It can be seen that the frequency in the first ~ 0.3 s is much lower than expected.

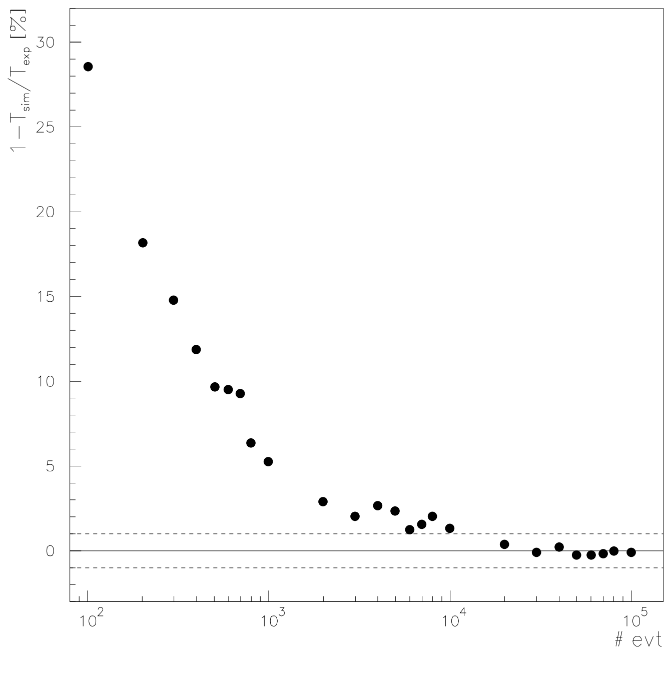
Only after that, the frequency reaches the expected value around which it fluctuates statistically.

FIGURE 6.1. Warm-up Effect



There are at least two ways to take the warm-up into account: one can either start the statistics collection only after some time when the system has already become stable, or one can make long runs in which the influence of warm-up becomes negligible. The influence of warm-up on the total throughput of the same setup as before is shown in figure 6.2. This shows the deviation of the simulated throughput at 1 kHz from the expected throughput versus the number of events simulated. It can be seen that this deviation is less than 1% for more than 20,000 events and less than 0.5% for more than 30,000 events.

FIGURE 6.2. Convergence of Throughput



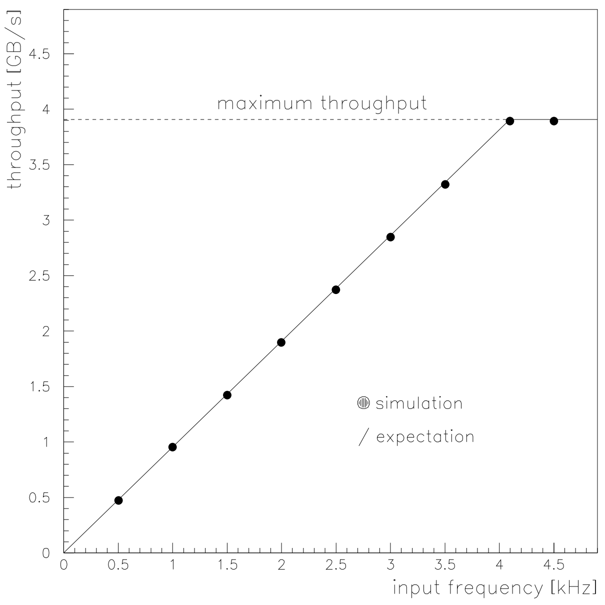
All simulations in this section were carried out with 30,000 events. The simulations of throughput are supposed to have a better resolution than 0.5%. The latencies have been checked with different random sequences to be statistically stable within 1%. The maxima of latency and buffer occupancy quoted are always obtained for 30,000 events.

6.2 Ideal Event Building System

In the ideal event building system the event fragment sizes are constant. This corresponds to a detector which does not do any sparsification but reads out all the channels all the time. Ideally the transfer of an event fragment does not have any overhead and the transfer time is only determined by the link speed.

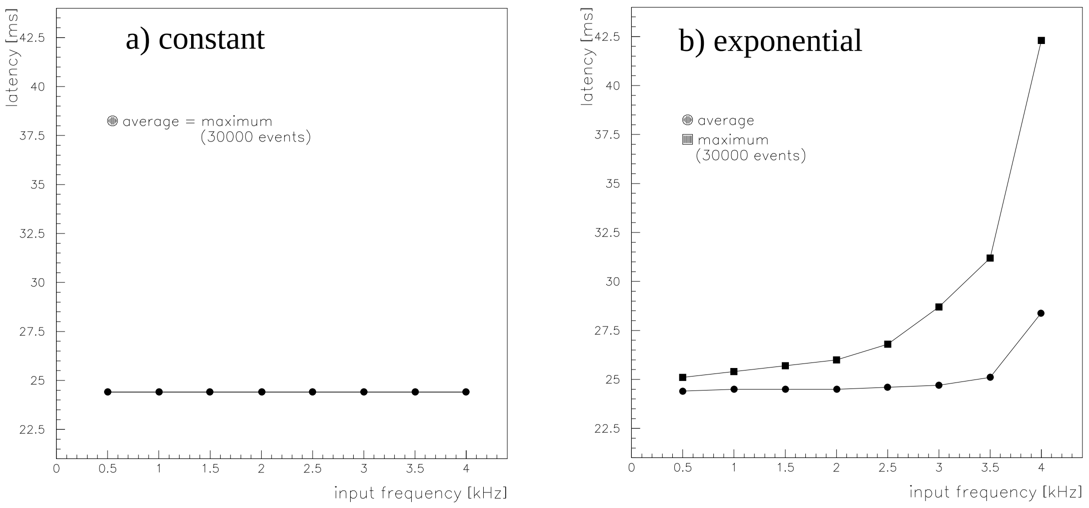
The simulated total throughput of the ideal event building system is shown in figure 6.3 in dependence of the input frequency. As expected, the throughput increases linearly with the input frequency until the switch becomes saturated and all links are fully used. The maximum throughput is then given by the link speed only and is $T^{max} = N_{Dst} \cdot \text{speed} = 3.91$ GB/s which corresponds to a maximum input frequency of $f^{max} = 4.10$ kHz. In the ideal event building system the switch runs in barrel shifter mode (see section 3.2.3) with the states of the switch being repeated in fixed time slots. The event fragments are sent to the destination always in the same order.

FIGURE 6.3. Throughput vs Input



The throughput does not show any difference if simulated with constant or exponential arrival time and the maximum throughput is the same as before, 3.91 GB/s. The latencies, on the other hand, differ: if the arrival time is constant the latency is constant. Each event takes the same time to be sent to the destination, the time being given by the link speed: $L = N_{Src} \cdot \text{size} / \text{speed} = 24.4$ ms. If the arrival time is exponentially distributed the number of events in a given time interval fluctuates and sometimes events arrive faster than the switch can handle them. In this case the input buffers derandomize the input traffic and store the events. The event fragments have to wait and the latency increases. The average latency deviates from the constant case in particular as the input frequency approaches the maximum because then frequently events arrive faster than they can be handled. The maximum latency for 1 kHz is about 4% larger than average and about 29% larger for 3.5 kHz. Due to the long tails in the exponential distribution the maximum latency is not finite when the input frequency approaches the maximum.

FIGURE 6.4. Latency vs Input for Different Arrival Times



The latency is larger than with constant arrival time due to the additional waiting time that can only be greater or equal to 0, but the throughput is in both cases the same. This is because events arriving faster than average will be stored and will be sent to the switch with maximum input frequency. In times when the events come less frequently than average, the additional events will be sent, so that in total, the throughput is the same as with constant arrival time.

FIGURE 6.5. Little's Law for Full Events

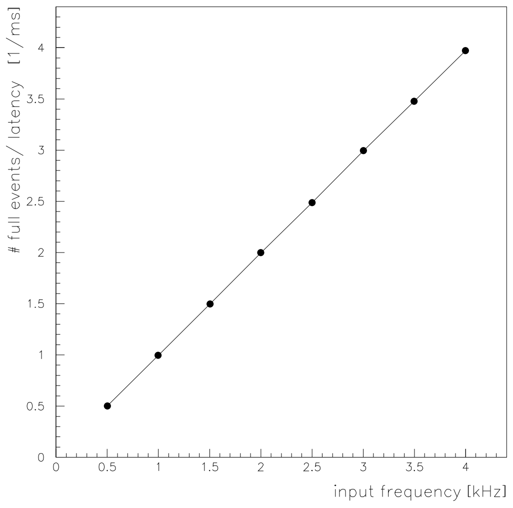
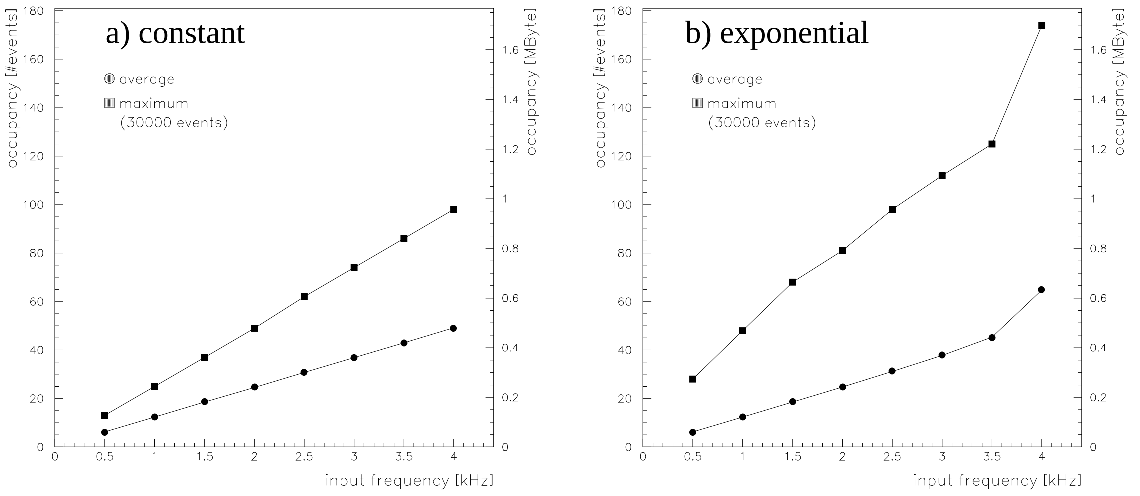


Figure 6.5 shows the validity of Little's law for the number of full events in the event building system (equation 5.1). The behaviour of the buffer occupancy in the ideal event

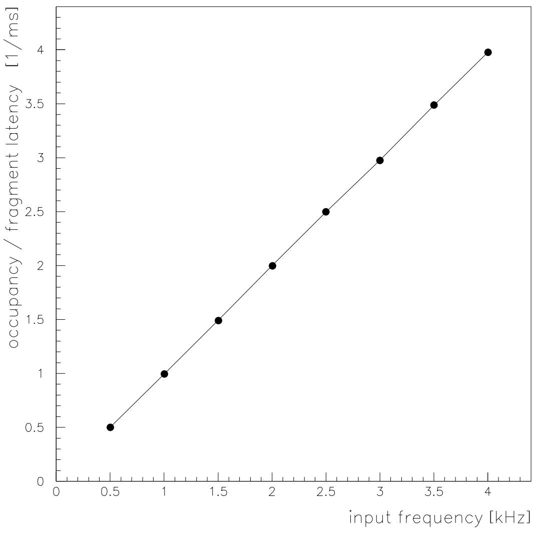
building system is similar to the behaviour of the latency. For constant arrival time any given source has to hold as many fragments as its position in the barrel shifter cycle has. The maximum occupancy is twice as large as the average and is equal to the number of sources for the maximum input frequency.

FIGURE 6.6. Buffer Occupancy vs Input for Different Arrival Times



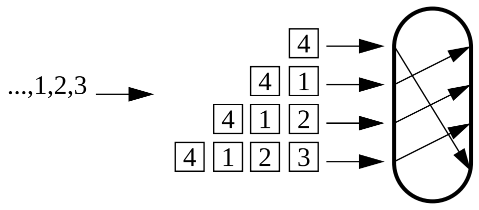
If the arrival time is exponentially distributed then the average occupancy remains similar to the constant arrival time but the maximum occupancy reflects the effect of derandomization and therefore can be more than 2 times the average occupancy and exceed 100 event fragments. For 1 kHz about 47 event fragments are stored at maximum, about 80% more than with constant arrival time, at 3.5 kHz 128 need to be stored, 50% more than with constant arrival time. As with the latency, the buffer occupancy is not finite when the input frequency approaches the maximum. There is no stable state.

FIGURE 6.7. Little’s Law for Event Fragments



As for the full events Little’s law is also valid for the event fragments (equation 5.2). Little’s law only allows to calculate an upper limit for the buffer occupancy using the latency and the input frequency (see equation 5.3). In the ideal event building system there is a simple relation between average buffer occupancy and average latency which will be deduced in the following based on the fact that the ideal event building system works in barrel shifter mode and the event fragments are strictly ordered.

FIGURE 6.8. Barrel Shifter Mode



The full event to a given destination is completely sent before the next one arrives for the same destination, so that there is no waiting time and the fragment latency can be calculated:

$$L_{Frg}(evt, src) = \sum_{i=1}^{src} (speed \cdot size(evt, src)) \qquad \text{(EQ 6.1)}$$

$$= (src \cdot time) \qquad \text{(EQ 6.2)}$$

where *src* is the position of the source in the barrel shifter cycle and *time* the constant transfer time per event fragment (= 244.1 μs). The average fragment latency can now be calculated with the number of sources N_{Src} to be

$$\langle\langle L_{Frg}(evt, src)\rangle_{evt}\rangle_{src} = \frac{N_{Src}+1}{2} \cdot time \qquad \text{(EQ 6.3)}$$

The maximum latency of an event fragment is:

$$\max_{src} L_{Frg}(evt, src) = N_{Src} \cdot time \qquad \text{(EQ 6.4)}$$

Using the definition of the latency (see equation 3.3 in section 3.1.3) it follows:

$$\langle L(evt)\rangle_{evt} = 2 \cdot \frac{N_{Src}}{N_{Src}+1} \cdot \langle\langle L_{Frg}(evt, src)\rangle_{evt}\rangle_{src} \qquad \text{(EQ 6.5)}$$

Using this relation and Little’s law for full event and event fragments, one gets the following relation, strictly valid in the ideal event building system with constant arrival time:

$$\langle B\rangle = \frac{1}{2} \cdot \frac{N_{Src}+1}{N_{Src}} \cdot \langle f\rangle \cdot \langle L\rangle \qquad \text{(EQ 6.6)}$$

The ratio of full event latency and the latency of an individual event fragment for maxima and average is shown in figure 6.9, for exponential arrival time distribution. The ratio of maxima has to be exactly equal to 1 by definition (equation 3.3). The ratio of averages is close to 2, though equation 6.6 cannot be derived for exponential arrival times because waiting times of the event fragments have to be considered. It seems, however, that their influence is negligible when the input frequency is smaller than the maximum input frequency (< 90%), leading to the empirical law:

$$\langle B \rangle \approx \frac{\langle f \rangle}{2} \cdot \langle L \rangle \qquad \text{(EQ 6.7)}$$

This is valid only with a certain accuracy for big N_{Src} and with input frequencies smaller than the maximum input frequency. With the exponential arrival times in the ideal event building system it has an accuracy of 1% for frequencies < 3.5 kHz. As a rule of thumb this formula can be useful.

FIGURE 6.9. Latency/ Fragment Latency vs Input (exponential arrival time distribution)

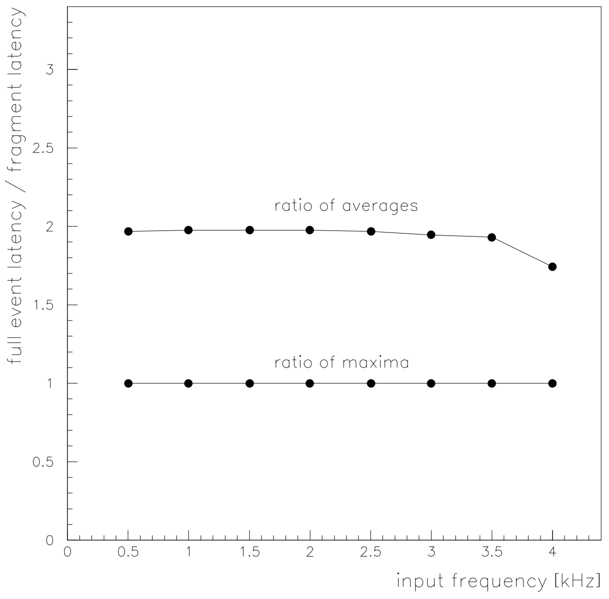
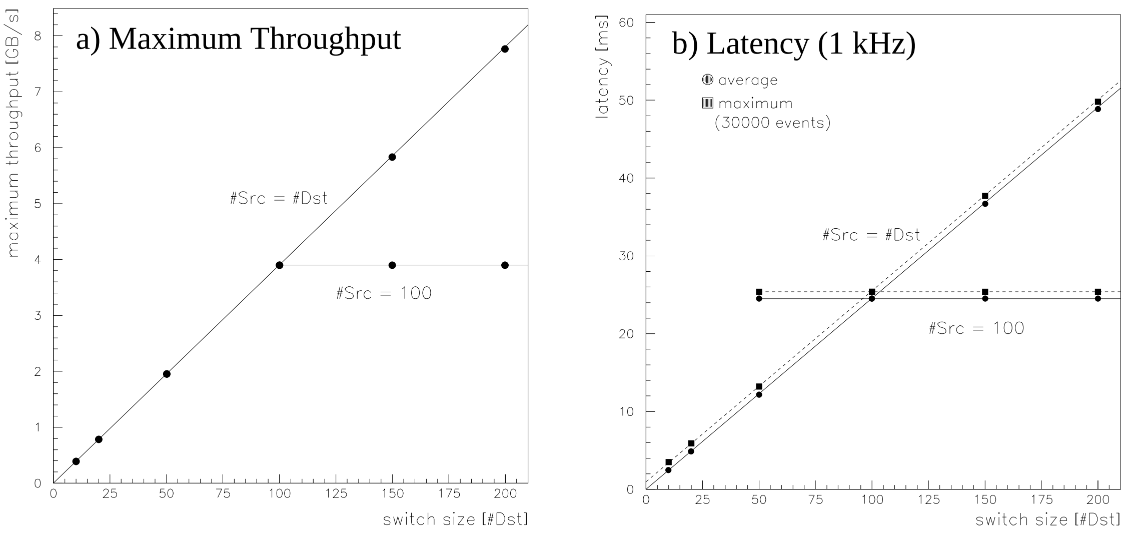


FIGURE 6.10. Scaling Behaviour (exponential arrival time distribution)



The scaling behaviour with the number of sources and destinations is shown in figure 6.10. The maximum throughput increases linearly with the number of sources and destinations in square systems. The increase in number of sources is also followed by an increase in the full event size. This scaling law shows that for the ideal event building system the maximum input frequency is only determined by the link speed and the event fragment size: f^{max} = speed/size = 4.10 kHz. If the number of sources is kept constant at 100 and the number of destinations is increased no change can be seen.

The latency, both average and maximum, scale as well with the number of sources, as is shown in figure 6.10b) for 1 kHz. In the case of 100 sources and a varying number of destinations the latency stays the same because in the ideal event building system runs in barrel shifter mode and if the input frequency is smaller than f^{max} the latency can be calculated as $L = N_{Src} \cdot$size/speed. The latency even stays the same if the number of destinations is decreased. This is due to the fact that at 1 kHz, the maximum rate for 50 destinations is not reached, which would be at 2.05 kHz.

6.2.1 Overhead and Processing Times

One assumption for the ideal event building system was that there are no overheads. This section will try to understand the influence of overheads on the performance of an event building system. Three different sets of overheads were simulated, the arrival time is always exponentially distributed:

- **No Overhead:**
  This is the ideal event building system presented before.
- **Single Overhead:**
  In this case each transfer time is increased about the constant time of 100 µs. Both, *Src* and *Dst* process are busy for that time.
- **Detailed Overheads:**
  In this case different overheads for different tasks are used: 45 µs for the *Src* process to handle a new event signal and 33 µs to send the data, and 54 µs for the *Dst* process to merge the event at the end of each transfer. These times were derived from the prototype (see section 4.3.4).

TABLE 6.1. Maximum Throughput with Different Overheads

Overhead	Maximum Throughput [GB/s]	Efficiency [%]
no overhead	3.906	100
overhead = 100 µs	2.762	71
detailed overhead	2.654	68

The maximum throughput and efficiency for the three sets are summarized in table 6.1. For a single overhead they can be calculated with a formula similar to equation 4.4:

$$T^{max}_{overhead} = \varepsilon_{overhead} \cdot T^{max}_{ideal} \qquad \text{(EQ 6.8)}$$

$$\text{where} \quad \varepsilon_{overhead} = \frac{size}{size + overhead \cdot speed} \qquad \text{(EQ 6.9)}$$

The influence of the overheads leads in the two cases of single and detailed overhead to an efficiency of about 70%. In the case of the detailed overhead this cannot be calculated easily because the processes run concurrently. The maximum input frequency for a given event size can be calculated from the maximum throughput divided by the size. However, for very small event fragment sizes (like in the ATLAS LVL2 system, see section 2.4.4) the overhead directly determines the maximum input frequency, e.g. for 100 µs f^{max} is 10 kHz.

FIGURE 6.11. Latency vs Input with Different Overheads

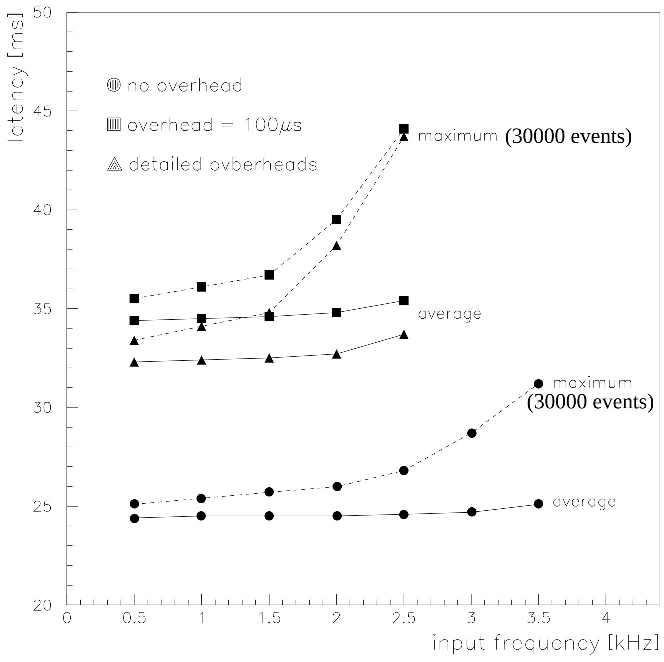


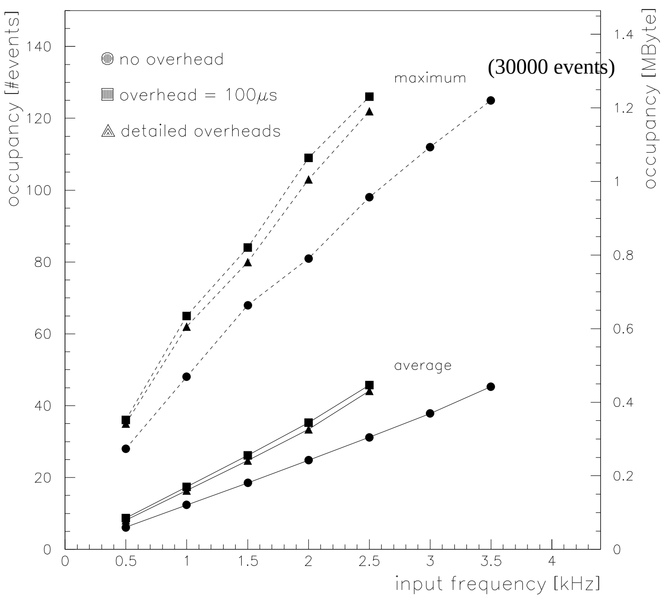
Figure 6.11 shows the influence of the overhead on the latency. For the single overhead the average latency can be estimated as:

$$L \approx N_{Src} \cdot \left(overhead + \frac{size}{speed} \right) \qquad \text{(EQ 6.10)}$$

Therefore the latency is 100×100 µs = 10 ms larger than in the ideal case. The average latency for the detailed overheads is about 8 ms more, which is less than for the single overhead. The maximum latency shows a similar behaviour as before, growing faster as the input frequency reaches the maximum at a lower value.

Little's law is true for full events and event fragments. The buffer occupancy is larger because the transfer times and therefore the latencies are longer. The maximum buffer occupancy for both overheads at 1 kHz input frequency is about 65 events which corresponds to 650 kByte which is 30% more than with no overhead. The ratio of full event latency to latency of an individual event fragment is still close to 2 (equation 6.7) with an accuracy of about 2%. The scaling behaviour with the number of sources and destinations is the same as shown in the previous section.

FIGURE 6.12. Buffer Occupancy vs Input with Different Overheads



The sets with a single overhead and detailed overheads show differences which are negligible compared to the difference they both have in comparison to the ideal event building system. For further simulations throughout this section the single overhead was used as an approximation for the detailed overhead.

6.3 Event Size Variations and Correlations

Another important parameter is the event fragment size. So far only constant event fragment sizes have been considered. But what happens if the event fragment sizes are not constant from event to event and not even from source to source? Constant fragment event sizes are good assumptions for detectors which read out all the channels regardless of wether they were hit or not. This is a good assumption for some of the detectors in ATLAS, but others, especially the ones with a low occupancy, will only read out channels with a signal above a certain threshold. This will reduce the total amount of data but it will also introduce a new phenomenon: varying event fragment sizes.

6.3.1 Full Event Size

Before actually varying the event fragment size on an event-to-event and source-to-source basis, the influence of the event size as such was investigated. Therefore, equal and constant event fragment sizes between 1 and 20 kByte were simulated. The maximum throughput varies with the event fragment size as shown in figure 6.13. It increases slowly and approaches the maximum throughput of the ideal event building system. This is because the efficiency can be described with equation 6.9. The efficiency converges to 1 when the event fragment size is so big that the transfer time is dominated by the speed and the overhead becomes negligible. Figure 6.13 also shows the maximum input frequency. This is only determined by the link speed as before and with equation 6.9 one gets:

$$f^{max} = \varepsilon_{overhead} \cdot \frac{speed}{size} = \frac{1}{overhead + size/speed} \qquad \text{(EQ 6.11)}$$

The maximum input frequency is 1 kHz for event fragment sizes of 36.9 kByte.

FIGURE 6.13. Maximum Throuhput and Maximum Input Frequency vs Event Size

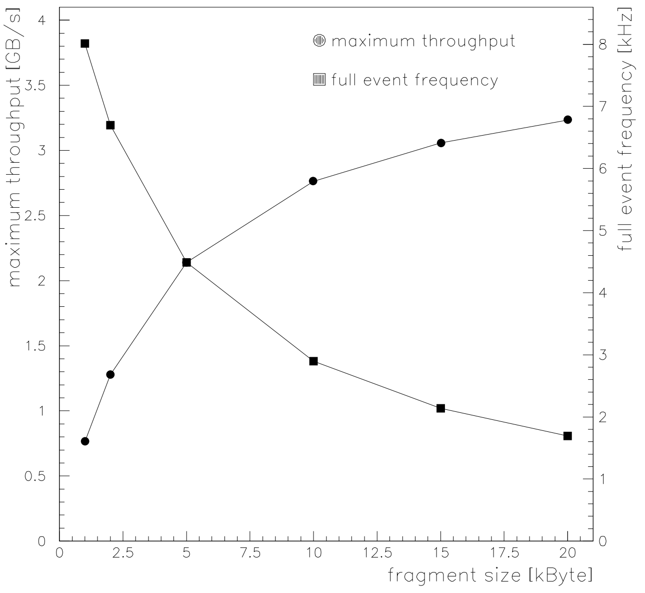
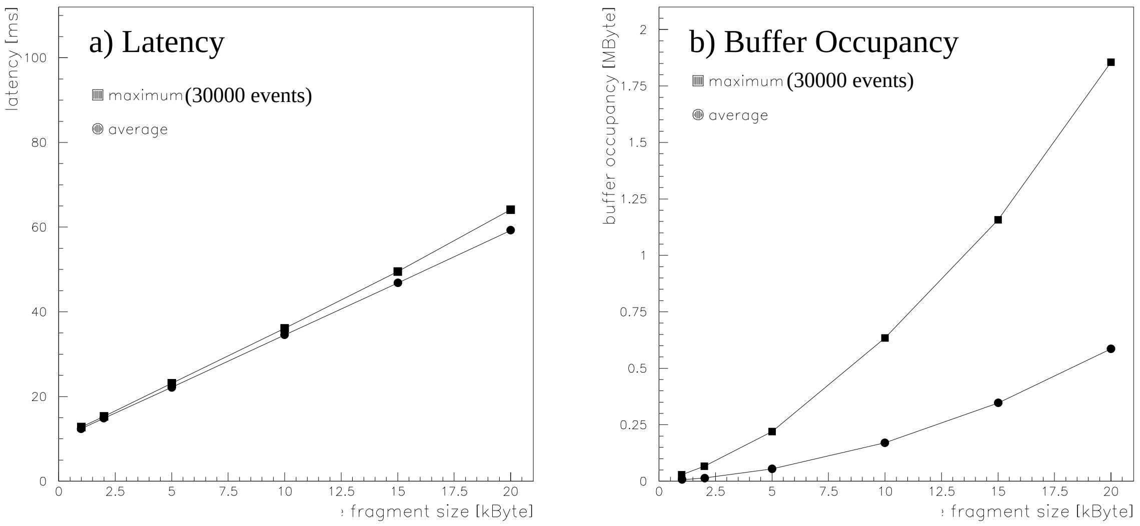


FIGURE 6.14. Variation of Event Fragment Size (fixed size, 1 kHz)



The latency (see equation 6.10) at 1 kHz increases linearly with the event fragment size until it approaches the limit of 36.9 kByte where it becomes infinite. Beyond this limit it will increase proportionally with time. The buffer occupancy, now expressed in size, increases consequentially $\propto size^2$ (Little’s Law) and needs a maximum buffer space of about 1.8 MByte for event fragment sizes of 20 kByte.

6.3.2 Event Size Distributions

The influence of event fragment size variations on a source-to-source basis was investigated with statistical distributions. The average event fragment size per source was kept constant at 10 kByte and the sources were treated independently. Two different distributions were tested:

- **Gaussian:**
  In this sample independent Gaussian distributions were used with σ_{rel} of 10%, 50% and 100%. The distributions were limited to a minimum of 4 Byte and a maximum of 20 kByte. The minimum limit can be regarded as an "empty buffer" message. The maximum comes from the maximum buffer size. Former experiments show that event sizes are Gaussian distributed [Map95b].
- **Exponential**:
  In this sample independent exponential distributions were used. These were only limited to a minimum of 4 Byte but not to any maximum. These somewhat artificial distributions turn out to be quite useful. Due to the long tails in the distribution they can be regarded as a worst case and are good to test the performance of the system.

The maximum throughput for the different event fragment size distributions is summarized in table 6.2., which also contains the relative efficiency compared to the constant event fragment sizes; i.e. the efficiencies are factorized as a contribution from the overhead and one from the size variations and ε_{total} is the product of ε_{size} and $\varepsilon_{overhead}$:

$$T_{real}^{max} = \varepsilon_{size} \cdot \varepsilon_{overhead} \cdot T_{ideal}^{max} \qquad \text{(EQ 6.12)}$$

It can clearly be seen that the fluctuations of the event fragment sizes decrease the throughput and thus the link efficiency. This comes from the fact that the switch still running in barrel shifter mode will have to adapt the time slot to the largest fragments at a given time. The smaller fragments will not use their time slot fully and the throughput decreases. The exponential size distribution with its long tail towards the big event fragments reduces the link efficiency to about 60% of the value with the single overhead and constant size, this is in total about 42% of the theoretical throughput.

TABLE 6.2. Throughput of Different Event Fragment Size Distributions

Event Size Variations	Maximum Throughput [GB/s]	ε_{total} [%]	ε_{size} [%]
fixed event size	2.762	71	100
Gaussian, σ_{rel} = 10%	2.603	67	94
Gaussian, σ_{rel} = 50%	2.176	56	79
Gaussian, σ_{rel} = 100%	2.080	53	75
exponential	1.623	42	59

Since the maximum throughput is decreased, the maximum input frequency is also decreased. An input frequency of 1 kHz is much closer to the maximum where latency and buffer occupancy become infinite. This maximum is 1.70 kHz for the exponential distributions, an input frequency of 1.5 kHz already sees a strong increase in the latency and buffer occupancy. For 1 kHz input frequency the latency increases about 32% on the average value, and about 86% on the maximum value. The average buffer occupancy increases between 10 and 20% and for an exponential size distribution and 1 kHz input frequency a maximum buffer of 900 kByte must be provided. For both sets, Little's law was checked to be true for full events and event fragments. The relation between latency and latency of an individual event fragment (equation 6.7) is also still valid.

FIGURE 6.15. Latency for Different Event Fragment Size Distributions

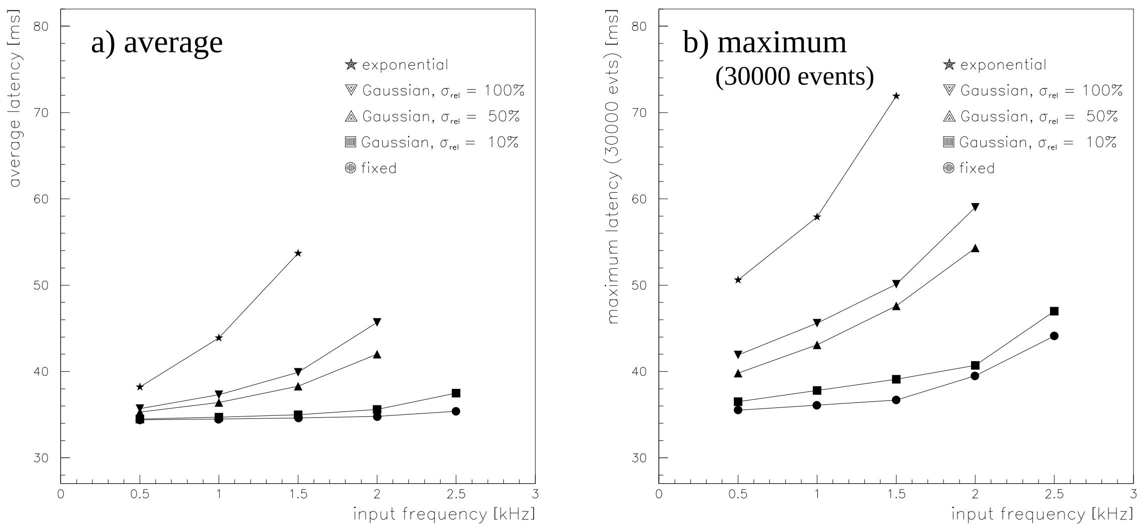
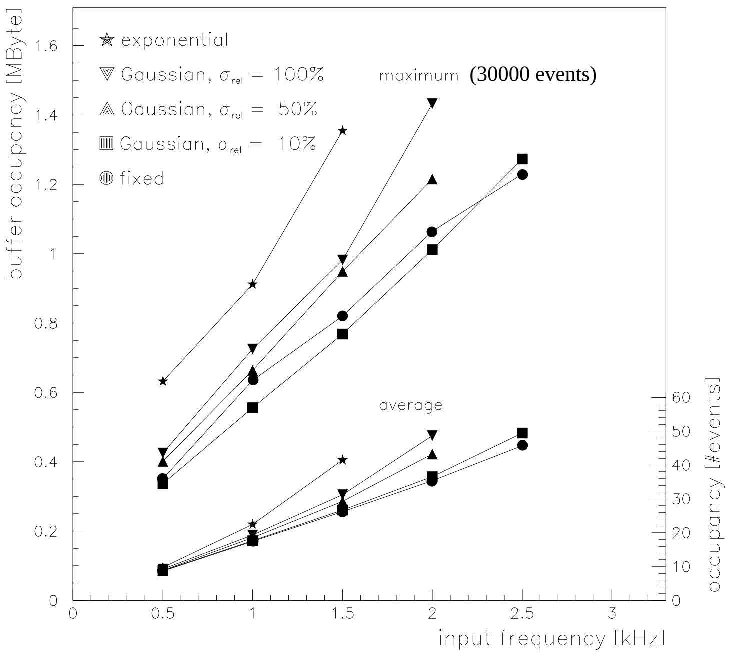
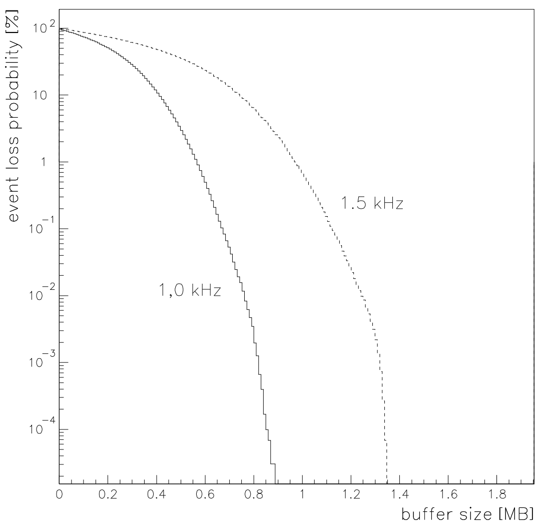


FIGURE 6.16. Buffer Occupancy for Different Event Fragment Size Distributions



The buffer occupancy can also be expressed as the probability that an event gets lost if the buffer has a fixed size. This is shown in figure 6.17 where the event loss probability versus the buffer size is shown. It can be seen that for example for an input frequency of 1 kHz a buffer of 600 kByte would lead to 1% of the events to be lost, at 1.5 kHz this would lead to every 5th event being lost. If the events are not to be lost, the event building system has to send a signal to the up-stream data acquisition system, in which case the event loss probability translates into a contribution to the deadtime.

FIGURE 6.17. Event Loss Probability (exponentially size distribution)



6.3.3 Event Size Correlations

The event size distributions investigated so far, were all based on independent distributions. Three other sets have been simulated to take into account correlations between the sources:

- **Pattern A:**
  One event fragment is 10 times bigger than the others. The source for this big fragment is distributed uniformly over the 100 sources per event. Fluctuations of a Gaussian distribution with a σ_{rel} of 10% were used to sample the event sizes and the average event fragment size 10 kByte.
- **Pattern B:**
  A subset of 10 sources was sampled with Gaussian distributed event fragment sizes of a σ_{rel} of 50%. The other 9×10 sources were then correlated to this subset with a correlation factor of 1 with a 5% Gaussian fluctuation.
- **Pattern C:**
  This is similar to pattern B, but the event sizes were scaled in groups of 10 with factors of 0.1, 0.5, 1.0, 1.5 and 1.9. The full event size on average was left unchanged. This pattern simulates correlated sources with different average event fragment sizes.

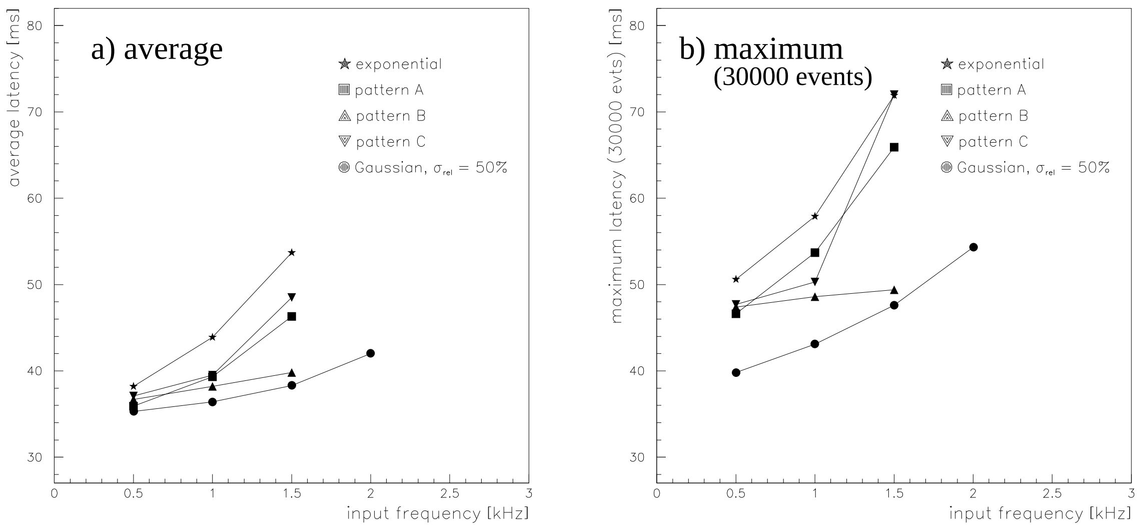
The maximum throughput achieved for these patterns is summarized in table 6.3 together with their efficiencies ε_{size}. These efficiencies are similar to the varying event sizes. The correlation of event fragment sizes seems not to create an additional overhead.

TABLE 6.3. Throughput for Correlated Event Fragment Sizes

Pattern	Maximum Throughput [GB/s]	ε_{size} [%]
pattern A	1.751	63
pattern B	2.186	79
pattern C	1.594	58

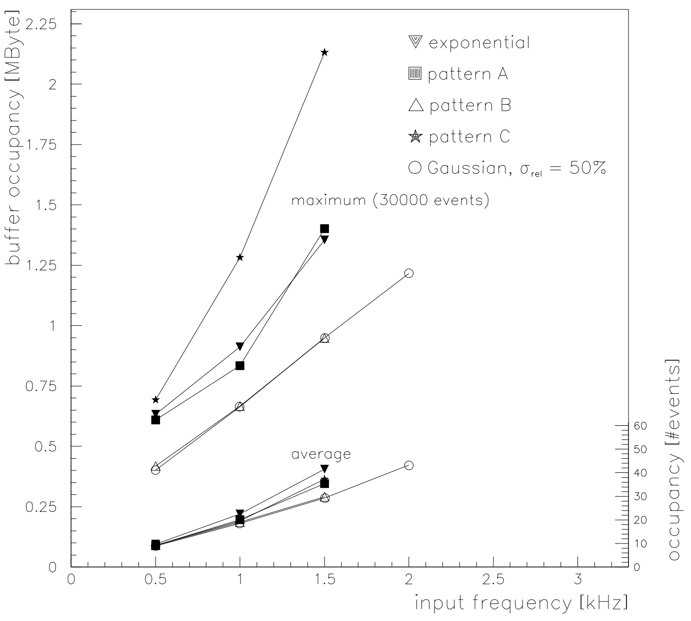
From the latencies one can see that pattern A is very similar to the independent exponential distributions while pattern B is somewhat worse than the independent Gaussian distributions. The influence due to the correlations is, however, not as large as the influence of the fluctuations in general, as can be seen especially pattern A. It is the differences in the event fragment sizes that change the timing of the barrel shifter mode and lead to inefficiencies. This can also be seen in pattern C which has big differences in the event fragment sizes and thus the worst efficiency.

FIGURE 6.18. Latency for Correlated Event Fragment Sizes



The latencies and buffer occupancies for the correlated event fragment sizes are shown in figure 6.18 and 6.19. As expected they show a very similar behaviour as before. Pattern A is actually very similar to the exponential case and the maximum occupancy for pattern B is almost identical to the Gaussian case with σ_{rel} = 50%. Pattern C also has some similarities to the exponential distributions. The buffer occupancies, expressed in memory size, are worse due to the bigger event fragments. For 1 kHz input frequency a buffer size of 1.25 MByte is required to lose less than one event out of 30,000 (probability $\leq 3\cdot10^{-5}$).

FIGURE 6.19. Buffer Occupancy for Correlated Event Fragment Sizes

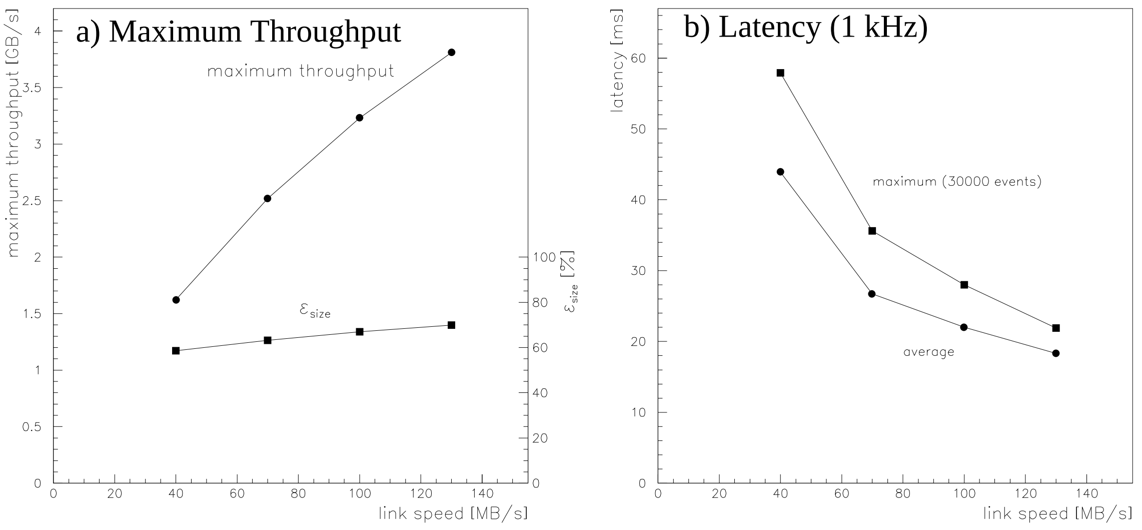


Little’s law was verified for both, full events and event fragments. The relation between latency and latency of an individual event fragment is again given as in equation 6.7 but due to the big fluctuations the probability of overlapping events increases and the rule is only valid within 5%.

6.3.4 Link Speed

The dependance of the maximum throughput on the link speed is shown in figure 6.20a) which also shows the efficiency contribution ε_{size}. Figure 6.20b) shows the latency in dependance on the link speed. It can also be estimated with equation 6.10 and it thus decreases when the link speed increases.

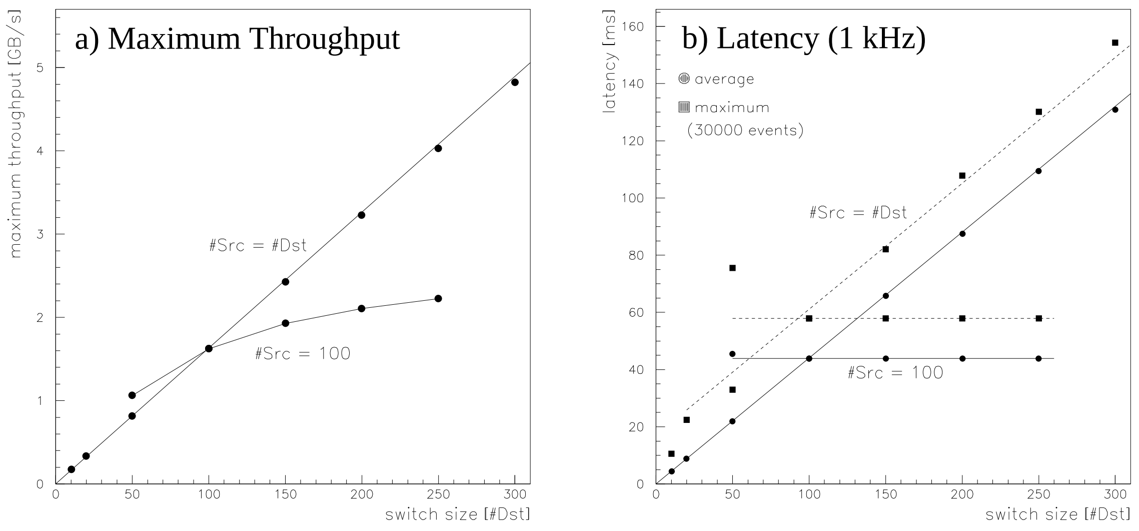
FIGURE 6.20. Variation of Link Speed



6.3.5 Scaling Behaviour

The scaling behaviour was checked with independent exponential event fragment size distributions. It can be seen that the efficiency for a square switch is about 42%, independent of the number of sources and destinations. For a constant number of sources, however, varying the number of destinations has some influence and increasing the number of destinations increases the maximum throughput. This is because the probability of two full events to wait for the same destination decreases when there are more destinations and sources can be free due to the fact that they did not use their time slot fully. But the maximum throughput increases slower than the aggregate bandwidth and thus the total efficiency degrades.

FIGURE 6.21. Scaling Behaviour (exponential size distribution)



The latency for exponential event fragment size distributions for square switches is scalable as before. It stays the same even when changing the number of destinations and leaving the number of sources constant.

As in section 6.3.1 the influence of the event fragment size was checked. A very similar behaviour as before was observed when changing the average event fragment size from 1 to 15 kByte. The maximum input frequency is scaled down with the 60% efficiency and reaches the 1 kHz line already at 20.1 kByte. Similar to equation 6.11 one has:

$$f^{max} = \frac{\varepsilon_{size}}{overhead + size/speed} \qquad \text{(EQ 6.13)}$$

The only visible influence the exponential distribution shows is that the maximum throughput is smaller by about 60% which is why latencies and buffer occupancies grow faster. For 15 kByte event fragments, however, a buffer space of 1.8 MByte would still lead to an event loss probability of less than $3{\cdot}10^{-5}$.

FIGURE 6.22. Event Size Dependence (exponential size distribution): Maximum Throughput and Maximum Input Frequency

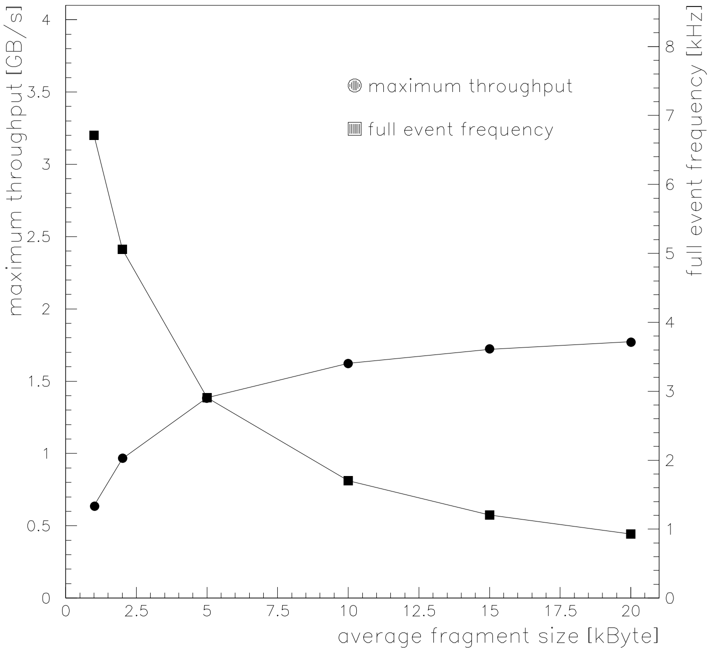
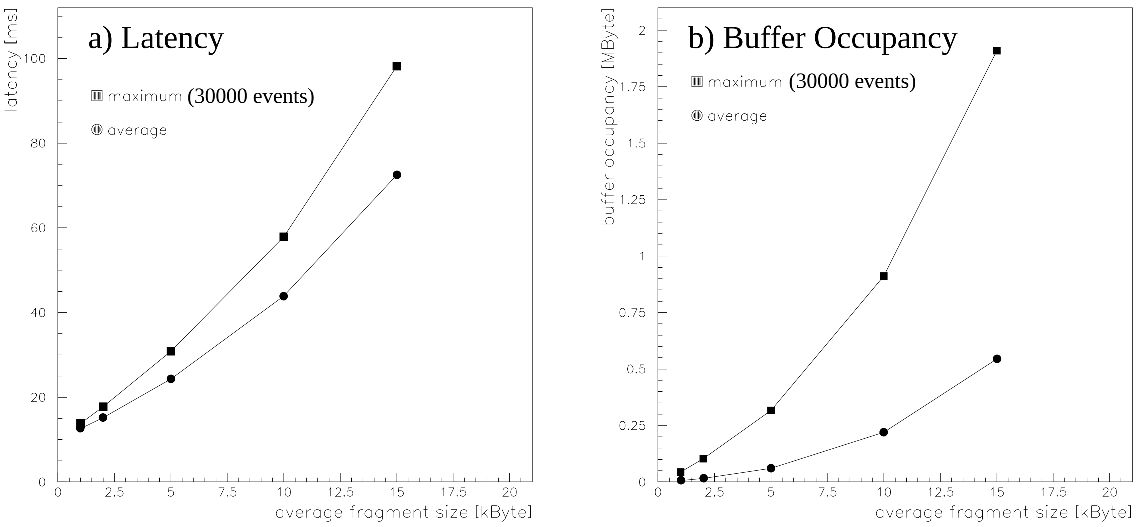


FIGURE 6.23. Event Size Dependence (exponential size distribution, 1 kHz)



6.4 ATLAS Model

In the previous section it has been shown that the event fragment size distributions play an important role in the performance of an event building system. ATLAS off-line simulations were used to get more realistic event fragment size distributions. These simulations were carried out with physics events and a detailed detector geometry. They deliver hit distributions which can be used to estimate the amount of data per ROB (see section 2.4.2), the variations from event to event and the correlations among the different sources.

6.4.1 Off-line Simulations

The ATLAS SCT was simulated which due to its low hit occupancy of 1% per event will only read out the channels actually hit and lead to varying event fragment sizes. The physics events taken for this simulation were events which had passed the LVL1 calorimeter trigger for isolated electrons. They represent about 55% of the expected LVL1-accepted trigger rate [ATL94]. An additional LVL2 trigger was not run because otherwise the statistics would have been too low. It can be hoped that the event fragment sizes do not change significantly between LVL1-accepted events and LVL2-accepted events. SIMEB was used to read in the event fragment sizes on an event-to-event basis and simulations were carried out as before. The off-line simulations were carried out in several steps:

1. Generation of physics events:

   The event generator used was PYTHIA [Ben87]. Two-jet events were produced with p_T(hard) > 35 GeV and $|\eta$(hard)$| < 2.5$. Minimum bias events have been generated with GENCL [Aln87] with $\sqrt{s} = 14$ TeV.

2. Detector Simulation:

   The detector simulations were performed with the SLUG/DICE/GEANT package [ATL95]. The detector layout used was the DICE “INNE” 3 layout - the baseline layout for the technical proposal [Bai94]. The size of the Silicon wafers is 6×12 cm^2 in $r\varphi\times z$. Each wafer contains 800 φ-strips. The wafers are organized in layers at radii of 30, 40, 50 and 60 cm. The number of channels, only reading out the strips aligned in φ, is $2.4\cdot10^6$, as can be seen from table 6.4.

TABLE 6.4. Parameters of the SCT (only φ-layer)

Layer	Nominal Radius [cm]	Number of Wafers [$r\varphi\times z$]	Number of channels
1	30	36×14	403,200
2	40	48×14	537,600
3	50	60×14	672,000
4	60	72×14	806,400
total		3072	2,419,200

3. Trigger Simulation:

   The trigger simulations were performed with the ATRIG program [Car94]. The hits from the jet sample and an average of 18 minimum bias events per event, to simulate high luminosity running and to give realistic occupancies in the detector, were superimposed and digitized. A hit in the SCT was defined if there was any energy deposit in the channel. This is an overestimation of the number of hits, especially as no charge division and readout noise were simulated which could reduce the number. On the other hand no cross-talk between channels and no additional noise hits were simulated either.

   A first level calorimeter trigger was applied to identify isolated electrons with $|p_T| >$ 35 GeV [Bor93]. Only events passing this trigger were considered for further analysis.

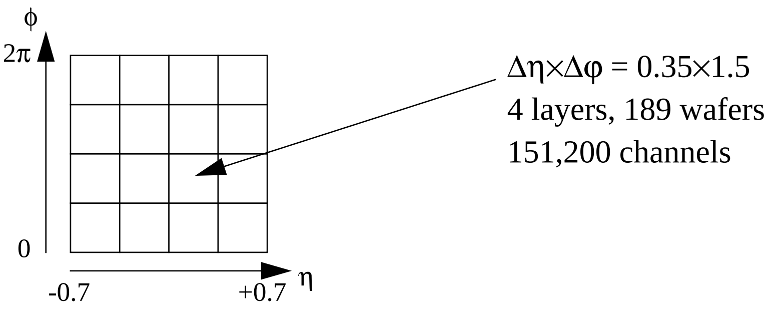
The standard ATRIG program was modified to perform the mapping of hits onto sources of the event building system and to write the event fragment sizes event-by-event to an ASCII file with a simple format based on keywords.

A total of ~ 1,000 already pre-filtered two-jet events were used with a total of ~ 3,000 minimum bias events in the way described above. A total set of 3075 full events were obtained with their event fragment size distributions.

6.4.2 Readout Buffer Mapping

The Silicon detectors will be read out with one optical fibre for every two wafers. This fibre could have a low bandwidth and any of them could be multiplexed onto high bandwidth fibres going to one ROB each [Haw95]. Several ROBs sitting in the same readout crate will have one link to the event building system. The mapping of fibres to ROBs has not yet been defined. So, the assumption was made that all fibres belonging to the same physical $\eta\times\varphi$ segment are actually read out by the same ROB. The collection of ROBs in readout crates was made so that every source covers a square array of an $\eta\times\varphi$ segment. In total 4×4 sources were used.

FIGURE 6.24. Source Mapping



The amount of data per hit transferred, is given by an overhead containing the fibre identifier and then per digitization, an identifier and the pulse heights. The number of bits transferred per wafer is $31 + 36\cdot N_{hits}$, where N_{hits} is the number of hits on this wafer [Haw95].

This value is calculated per fibre and since there are 189 fibres in the source and reading out the stereo-angled strips as well, this leads to

$$\mathrm{size}(\mathrm{evt}, \mathrm{src}) = 2 \cdot (732 + 4.5 \cdot N_{hits}) \ \ \mathrm{Byte} \qquad \text{(EQ 6.14)}$$

The presampler, electromagnetic calorimeter and hadronic calorimeter in the barrel region have 139,264 channels [ATL94]. They will be read out completely with 3 words per channel [Boc95]. After a LVL2-accept the data should be sparsified by a factor 2 [Map95a]. For a 4×4 array of sources one yields an average fragment size of 12.75 kByte. Due to fluctuations in the sparsification this will also vary. It is assumed that the fluctuation will be of Gaussian shape with a σ_{rel} of 10%.

6.4.3 Event Size Distributions

The simulations with 18 minimum bias events lead to an average occupancy of the detector of about 0.4% as can be calculated from the number of hits per event, shown in figure 6.25. The occupancy per fibre is shown in figure 6.26. The average flow of data per fibre can be estimated to be about 1.72 MB/s, assuming a LVL1-accepted trigger rate of 100 kHz. This value is in agreement with earlier simulations [Haw95].

FIGURE 6.25. Occupancy in the SCT

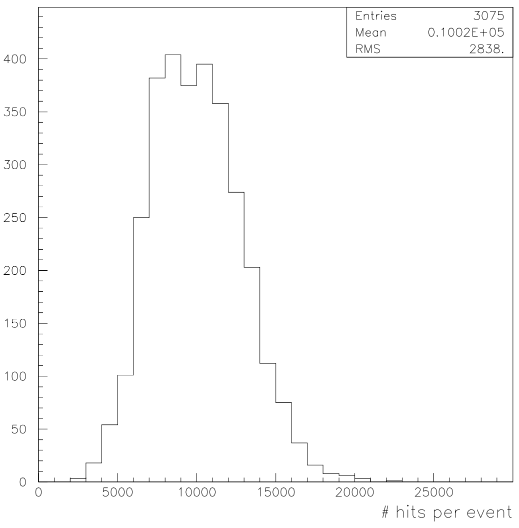
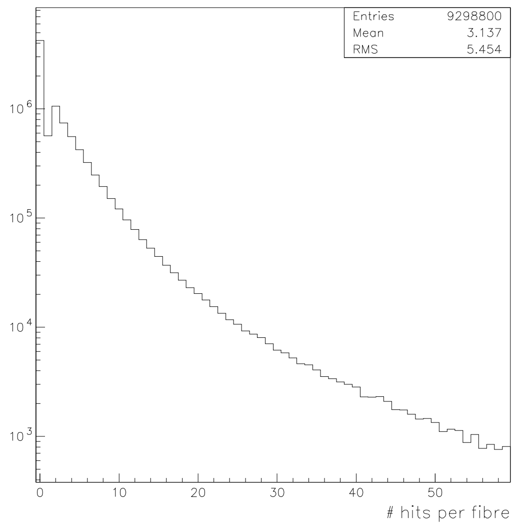
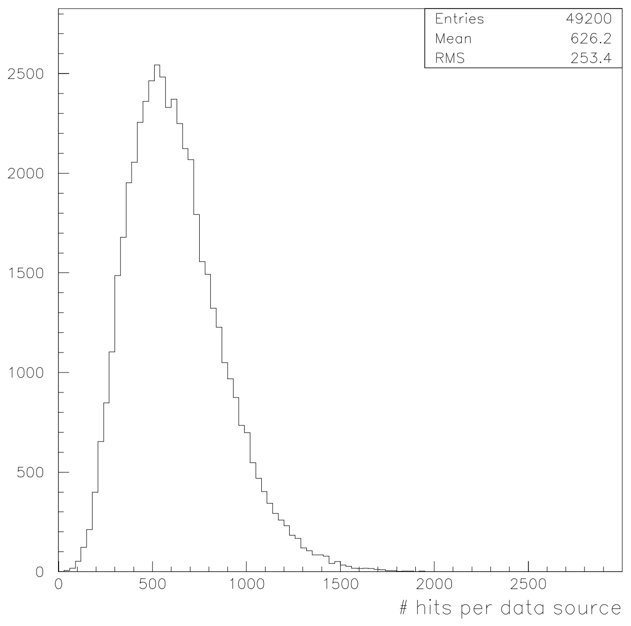


FIGURE 6.26. Hits per Fibre



Every source has on average 626 hits in each event. Using equation 6.14 leads to an average event fragment size of 6.93 kByte. Figure 6.27 shows the shape of the event fragment size distribution. It has an *rm*s of about 40%, the maximum event fragment size can be calculated to be 19 kByte (= 2,000 hits) which is about 3 times the average size. The shape is of the distribution is not exactly Gaussian but rather little extended towards bigger event fragments. Nevertheless, this distribution is quite “well behaved” and does not have very long tails. This comes from the fact that so many channels are being added up that the law of big numbers can be applied.

FIGURE 6.27. Event Fragment Size Distribution



6.4.4 Results

The ASCII files containing the occupancy in the sources were used for input with SIMEB. The set of 3075 events was randomized by skipping on average 5 events, thus allowing the simulations to run for the required 30,000 events without repeating the events in the same order. Three different sets of simulations were performed:

- **Set A (SCT):**
  The files were read as they are, i.e. the simulation had 16 sources with a realistic event fragment size distribution.
- **Set B (SCT+CALO):**
  The SCT sample was used and 16 additional sources to represent the calorimeters. They had independent Gaussian event fragment distributions around an average of 12.75 kByte with σ_{rel} = 10%. The minimum event fragment size was 4 Byte and the maximum 25.5 kByte. In total an average of 9.84 kByte per source resulted.
- **Set C (4SCT+4CALO):**
  This set was similar to set B, but three additional sets of each subdetector were introduced. These were sampled from the first set using a correlation factor of 1 with a Gaussian fluctuation of σ_{rel} = 5%. In total 128 sources were used and the full event size was 1.23 MByte. This set comes very close to the values defined in table 2.9.

The maximum throughput, input frequency and efficiency for the different sets of events are summarized in table 6.5. $\varepsilon_{overhead}$ has been calculated using equation 6.9 to be about 70% and ε_{size} was calculated using equation 6.12 to be about 80% which is thus comparable to a Gaussian distribution with σ_{rel} = 50%.

TABLE 6.5. Maximum Throughput of Realistic Event Fragment Sizes

Set	Number of Sources	Full Event Size [MByte]	Maximum Throughput [GB/s]	f^{max} [kHz]	ε_{size} [%]
A	16	0.10	0.32	3.001	81
B	32	0.31	0.71	2.365	81
C	128	1.23	2.82	2.350	80

The latencies are shown in figure 6.28. They show normal behaviour and can be estimated with equation 6.10. Only when approaching the maximum input frequency will they increase drastically. For 1 kHz in set C, the average is about 47 ms and the maximum about 65 ms which is about 40% more than average.

FIGURE 6.28. Latency for Realistic Event Fragment Sizes

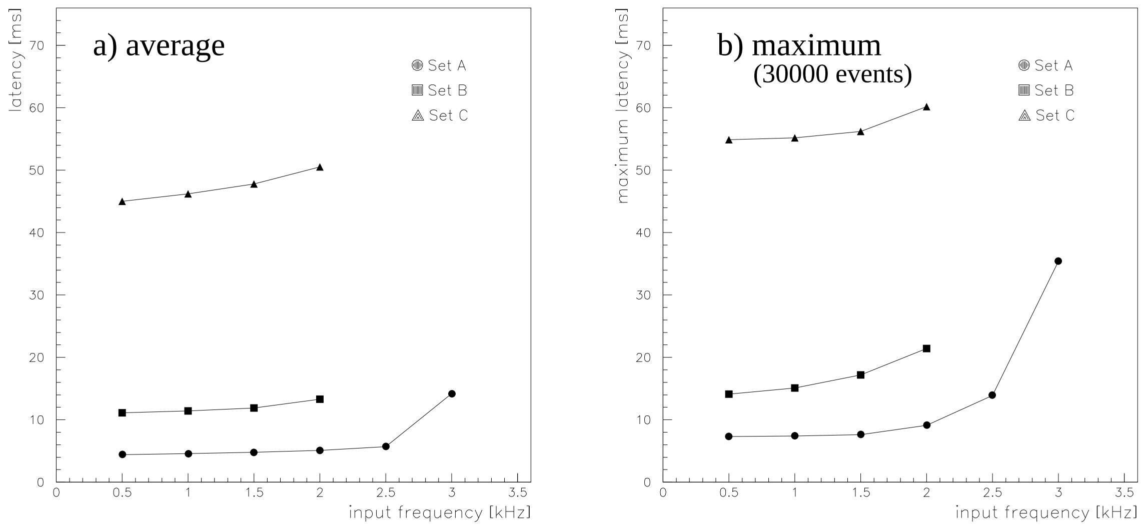
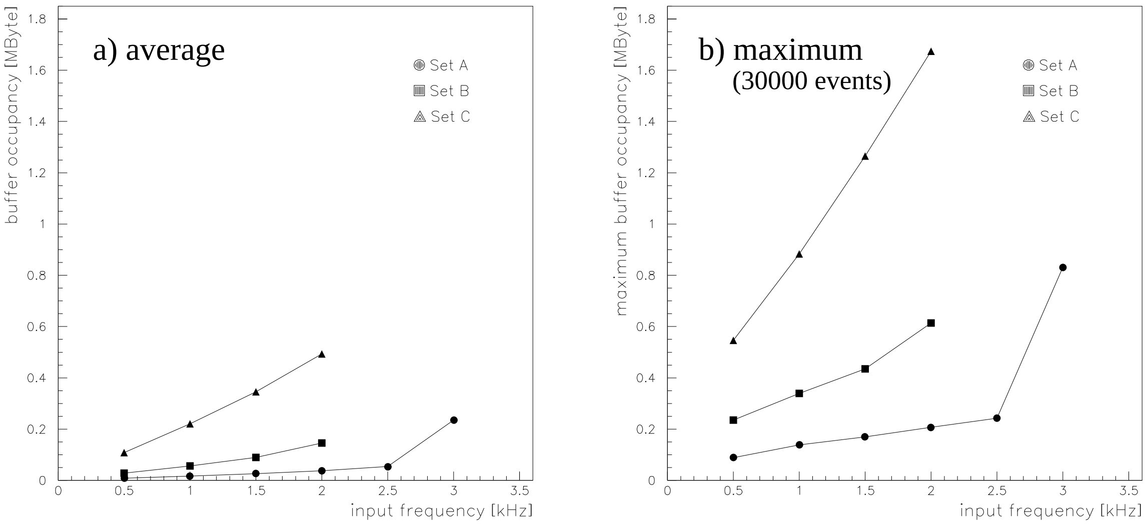
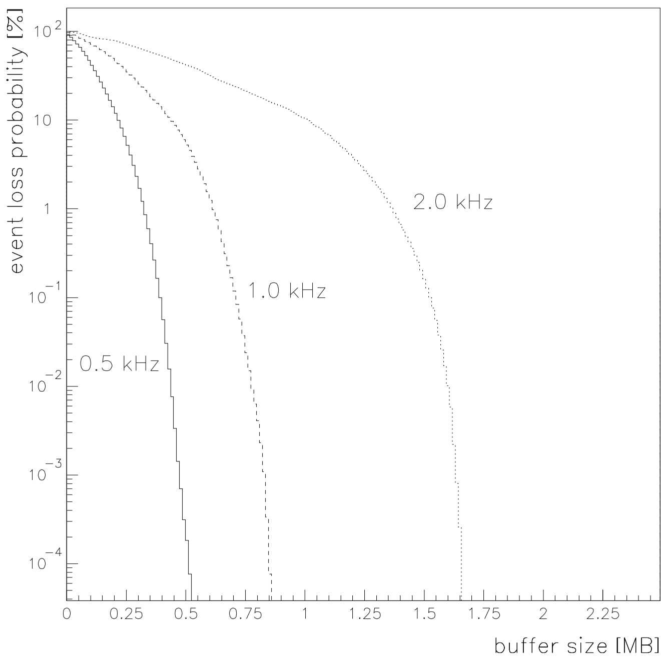


FIGURE 6.29. Buffer Occupancy for Realistic Event Fragment Sizes



The buffer occupancies are shown in figure 6.29. As expected they are quite similar to the buffer occupancies shown in section 6.3.2. For 1 kHz in set C a maximum buffer of 0.9 MByte is needed for 1 kHz and this increases to 1.7 MByte for 2 kHz which is still a reasonable value. The buffer occupancies expressed in event loss probability are shown for set C in figure 6.30. For an input frequency of 1 kHz a buffer of about 650 kByte is needed to lose less than 1% of the events. For 2 kHz this must be 1.4 MByte.

FIGURE 6.30. Event Loss Probability for Realisitic Event Fragment Sizes



6.5 Conclusions

Simulations of parallel event building systems based on the switch model from the prototype measurements were performed. The model is based on the assumptions that the switching delay is negligible and that the transfer times depend linearly on the link speed and the overhead. The destinations were assigned in a round-robin manner and it was assumed that the *Src* and *Dst* processes are completely independent from the up-stream and down-stream data acquisition system.

Values from the prototype measurements of 40 MB/s for the link speed and of about 100 µs for the overhead were used. To simulate the full-scale ATLAS-like event building systems, setups of 100 sources and 100 destinations have been simulated with a full event size of 1 MByte and an input frequency of 1 kHz. The influence of different parameters and the scaling behaviour of the event building system were measured starting from an ideal event building up to a realistic detector setup with realistic event fragment sizes. A model for the event building system is given which is expressed in relations between the performance indices.

With the assumptions made in the simulations, the event building system runs in barrel shifter mode. The efficiency which is defined as the usage of the ideal switch has two contributions: from the overhead and from the event size distributions or correlations. The first contribution can be calculated (equation 6.9) and comes from the time the link is not used because of some overheads in the software and hardware. The second contribution comes from the fact that varying event fragments lead to a varying time slot of the barrel shifter mode and the time slot which will not be used fully by all the event fragments. This cannot easily be calculated, but values for typical statistical distributions have been measured (table 6.2). This efficiency can get closer to 100% if the design of event building system takes into account a uniformity of the event fragment sizes.

The latency and buffer occupancy have a smooth behaviour as long as the input frequency is far from the maximum. The latency can be estimated with equation 6.10. Little's laws were verified for full events and event fragments and a simple approximate relation between average buffer occupancy and latency was deduced (equation 6.7). When the input frequency approaches the maximum all values grow infinitely and no stable state is reached at the maximum input frequency.

Based on the assumption that the switching delay is negligible, the event building systems were found to be scalable in number of sources and destinations. They also show a scalable behaviour with the full event size if the total throughput is far from the maximum. The behaviour of the system was checked in the range of 10 to 300 sources, respectively 0.1 to 20 MByte full event size for input frequencies between 0.5 and 4 kHz and no deviations were detected.

The influence of arrival time fluctuations is small: there is no effect on the efficiency if the buffers of the sources are capable of derandomizing the input traffic. Latency and buffer occupancy can be explained. The overhead on the contrary leads to an efficiency of about 70% with the parameters chosen. The efficiency contribution of the event size variations has typical values of 60 to 90% for some statistical distributions. The independent exponential distributions are the worst case which can be used as a benchmark test. Event size correlations do not have a further effect. They are fully described by the event fragment size fluctuations.

Realistic event size distributions were obtained from off-line simulations. In a system with 128 sources and a full event size of 1.23 MByte the total efficiency is about 56% which comes from a contribution to the overhead ($\varepsilon_{overhead} \approx 70\%$) and to the size variations ($\varepsilon_{size} \approx 80\%$). The latter is comparable to a Gaussian event size variation with $\sigma_{rel} = 50\%$. Typical values for the latency are between 45 and 55 ms. The buffer occupancy has an average of around 0.3 kByte at 1 kHz and is smaller than 1.8 MByte at maximum for 2 kHz.

These results show that under the assumption of the model obtained from the prototype measurements, full-scale ATLAS-like parallel event building systems become feasible. Expected increases in the link speed and improvements in the processing times for future hardware and software will increase further the performance. But next-generation prototypes with more sources and destinations and further studies including the up-stream and down-stream data acquisition system have to show that the model is still valid for big systems.

7.0 Simulation of Data Flow Management

As mentioned before (see section 3.4), the data flow management of an event building system has to implement some control functions to ensure that the event fragments will be correctly transmitted and assembled. The data flow management covers the way the interconnecting network resolves contention, the way destinations are assigned, the way the event fragments are buffered and sent to the interconnecting network and finally how the network is organized, if it has one big switch or a network of several smaller switching units.

While in the previous section the contention resolution was always made in a round-robin manner and the destination assignment was done by the *Src* process statically in a round-robin manner, this section will investigate alternative data flow management schemes and their influence on the performance of a parallel event building system.

The simulations performed were based on a 100×100 setup with the parameters from the prototype measurements (see section 4.4.1), i.e. a link speed of 40 MB/s, a transfer overhead of 33 μs, an event assembly of 54 μs and an event handling time of 45 μs in the data flow processes. The detailed overheads were chosen in this case because for the different data flow management schemes the individual contributions to the total overhead are important. If not mentioned otherwise, the event fragment size was sampled from independent exponential distributions with an average of 10 kByte. This was used because the exponential distributions showed the biggest differences in performance and the influence of the different data flow management schemes can be seen easily.

7.1 Contention Resolution

When within the interconnecting network two or more sources want to send an event fragment to the same destination, contention occurs and the network has to decide which source it will serve next. Several algorithms of contention resolution have been compared: Round-robin, Random, Fifo and Priority (see section 3.4.2 for the definition).

TABLE 7.1. Different Contention Resolution Modes

Mode	Maximum Throughput [GB/s]	Latency (1 kHz) [ms]		Buffer Occupancy (1 kHz) [kByte]	
		Average	Maximum	Average	Maximum
Round-Robin	1.607	43.0	57.1	220.9	1026.8
Random	1.608	43.1	57.6	221.4	931.1
Fifo	1.610	43.1	59.4	221.4	937.5
Priority	-	43.1	58.7	221.4	939.6

Table 7.1 shows the performance values for these contention resolution algorithms in a *PUSH* scheme. For the priority algorithm no maximum throughput could be simulated because the highest priority source will always send event fragments while the others can never send any. This algorithm is unsuited for an event building system.

In general, all algorithms have very similar performance values within the statistical fluctuations. For fixed event size they are even equal (not shown in table 7.1). This means that the influence of the contention resolution is negligible, assuming that the time for the arbitration is in all cases very small compared to the other overheads. Therefore in the following simulations only the round-robin algorithm will be considered which is also the algorithm that the switch in the prototype system used.

7.2 Destination Assignment Schemes

When a source has received an event fragment it needs to know which destination to sent it to. The destination assignment has to make sure that at any time there is a unique relation between any full event and a destination. This can be achieved in different ways. The previous section used a source-initiated algorithm with destinations being assigned in a round-robin manner.

7.2.1 Random Destination Assignment

As an alternative to assigning destinations in a round-robin manner, they could be sampled randomly. In order to assure the event building functionality, all event fragments belonging to the same full event have to be assigned to the same destination. Thus in random assignment the actual assigning can only be done in an earlier stage of the data flow, probably using the trigger broadcasting mechanism to send the destination identifier. The *Src* process will then only look up a field in the event descriptor and send the fragment to the assigned destination.

TABLE 7.2. Random Destination Assignment

Mode	Event Size (10 kByte)	Maximum Throughput [GB/s]	Latency (1 kHz) [ms]		Buffer Occupancy (1 kHz) [kByte]	
			Average	Maximum	Average	Maximum
Round-Robin	fix	2.590	33.3	35.0	168.1	620.0
	exp	1.607	43.0	57.1	220.9	1026.8
Random	fix	1.416	250.8	435.8	1241.3	4090.0
	exp	1.299	276.6	432.0	1374.7	4144.4

Table 7.2 summarizes the performance values for random destination assignment in comparison with round-robin assignment. It is obvious that random assignment is much less efficient, has much higher latencies and needs much larger buffer space. This is true in both cases, with fixed and exponential event fragments sizes and comes from the fact that with a certain probability the same destination will be assigned twice (or more often) in

sequence. The event building system cannot any longer work in barrel shifter mode and will not use the links in parallel. It is interesting to note, that fixed and exponential event size have a very similar inefficiency with random assignment (ε_{fix} = 36%, ε_{exp} = 33%). In this case the influence of the destination assignment is much bigger than the influence of the event size fluctuations.

7.2.2 Destination-Initiated Assignment

Instead of the *Src* process assigning the destinations, the *Dst* processes could “pull” the event fragments by sending a request when they are ready to receive and assemble the next event. This assignment has the advantage of taking the actual status of the destinations into account but on the other hand it requires a network to transfer the destination requests to the *Src* process.

Two schemes have been implemented, the *PULL* and the *SYNC* scheme which are identical to the schemes of the same names used in the prototype (see section 4.2.4). In both schemes the *Src* processes store the event fragments until they receive a request from a *Dst* process. The difference between the two schemes is in sending the event fragments asynchronously or waiting till the preceding event fragment has been sent successfully. In this sense the *SYNC* scheme is in the middle between the *PUSH* and the *PULL* scheme. In both destination-initiated assignment schemes, however, it is assumed that the broadcasting network between destinations and sources sends all request without any delay and preserves their order. This way no additional synchronization between the *Src* processes, or between the *Dst* processes is needed and the sources will assign the “next” event fragment to a given destination request.

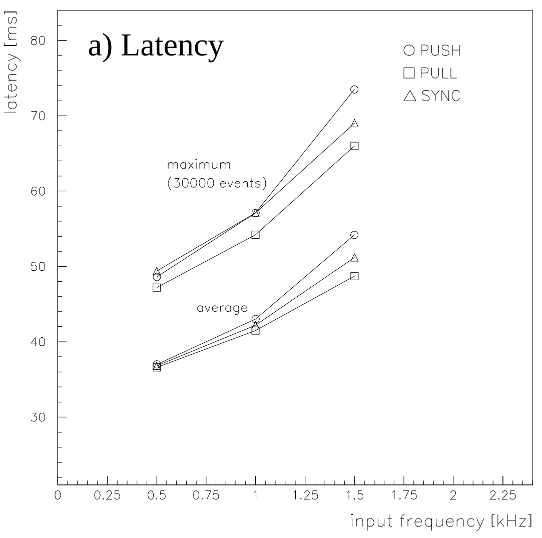
Table 7.3 summarizes the maximum throughput and efficiencies of the two schemes in comparison to the *PUSH* scheme. The influence of the overhead times on the efficiency differs in the three cases. This is due to the fact that in each scheme a different number of signals are received by the *Src* process for sending an event fragment: in the *PULL* scheme it receives one signal from the destination and in the *PUSH* scheme one for having sent the event fragment; in the *SYNC* scheme it receives both. Nevertheless the *SYNC* scheme is still more efficient than the *PUSH* scheme because it takes the status of the destinations into account.

TABLE 7.3. Maximum Throughput for Different Destination Assignments Schemes

Mode	Maximum Throughput [GB/s]		$\varepsilon_{fix} \equiv \varepsilon_{overhead}$ [%]	ε_{exp} [%]	$\varepsilon_{fix}/\varepsilon_{exp} \equiv \varepsilon_{size}$ [%]
	fix	exp			
PUSH	2.590	1.607	66.3	41.1	62.0
PULL	2.868	1.737	73.4	44.5	60.6
SYNC	2.632	1.670	67.4	42.8	63.4

The contribution of the event size variation to the efficiency was calculated as the ratio of efficiencies of the fixed size and the exponential size samples. The results are within 2% around $\varepsilon_{size} = 62\%$. The influence of the event size variations is in all schemes very similar.

FIGURE 7.1. Different Destination Assignment Schemes (exponential size distribution, 1 kHz)



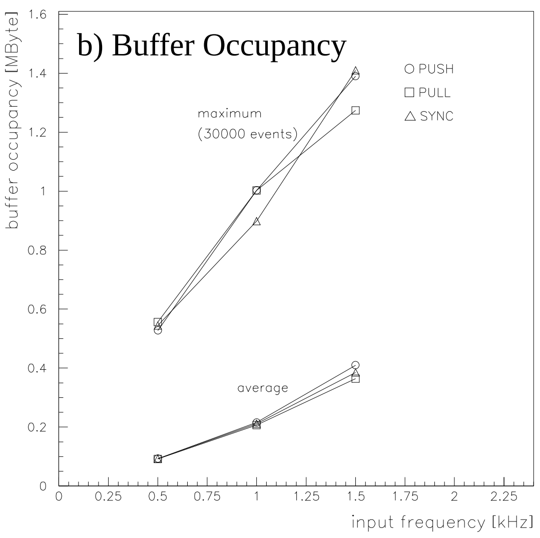
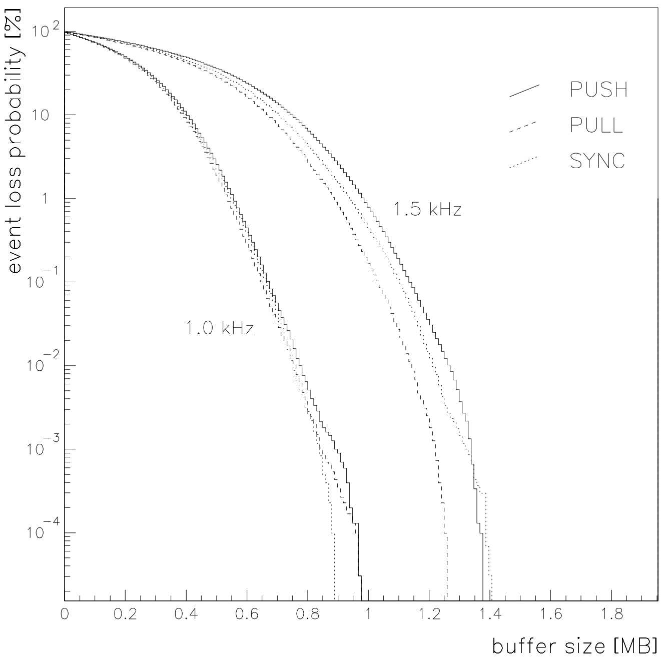


Figure 7.1 shows the latencies and buffer occupancies for exponential event size distributions at 1 kHz in the three different schemes. The *PULL* scheme has the lowest latency and buffer occupancy while the *PUSH* and *SYNC* scheme are slightly higher. In general, only differences of less than 10% exist between the different schemes. This is also reflected in figure 7.2 which shows the event loss probability. In all three schemes a buffer of about 600 kByte is needed to lose less than 1% of the events with an input frequency of 1 kHz. At this frequency a buffer of 1 MByte will lose events with a probability less than 10^{-5}.

FIGURE 7.2. Different Destination Assignment Schemes: Event Loss Probability



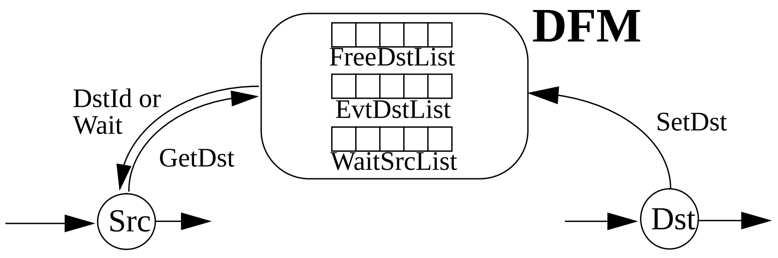
7.2.3 Data Flow Manager

A data flow manager (DFM) could also be used to do the destination assignment. This manager would be connected to the sources and destinations. It would act as a supervisor which keeps track of the status of the sources and destinations and keeps lists or tables of the events assigned. The *Src* processes would make requests (*GetDst*) by sending an event identifier and waiting for a destination identifier as a reply. The *Dst* processes would send messages (*SetDst*) to indicate that they are ready to accept another event.

An actual implementation of a data flow manager requires a network connecting the sources and destinations with the data flow manager and it also needs the data flow manager itself. The network could physically be the same as the interconnecting data network for data transfer if it has a low latency for message passing and allows messages in both directions. The data flow manager could be a centralized process or distributed among the sources and destinations and must be capable of handling messages and lists in a short time. For each full event a total number of N_{Src}+1 messages are exchanged at least. At a frequency of 1 kHz this means that the time to handle one such message is less than 10 µs. If the data flow manager task is distributed among the sources and destinations they will each keep independent lists of events and destinations. Synchronization methods using broadcast or multicast messages with latencies of less than 10 µs have to be used to keep the tables consistent.

A possible implementation of a protocol between the sources and destinations and a centralized data flow manager could look as follows: after the data flow manager has received the *GetDst* request it will check wether the event, with the given identifier, has already been assigned. If not, it will check if there is a free destination. If no free destination is available it will put the source on a waiting list and send back a message causing the source to wait until further notice. This is necessary because if the source was kept waiting on the data flow manager it would block other requests which could be served. This has been tested and it was found that the maximum throughput in this case is only 1.15 GB/s or ε_{exp} = 29%. If the data flow manager receives a *SetDst* message it will put the destination on the free list if there is no source waiting. If there is a source waiting then it will take the first from the list, assign it to the new destination, update the event table and scan the whole list and notify all sources waiting for the same event identifier.

FIGURE 7.3. Data Flow Manager



The maximum throughput of the ideal data flow manager spending no time to handle the messages is given in table 7.4. The performance is very close to the *PUSH* scheme.

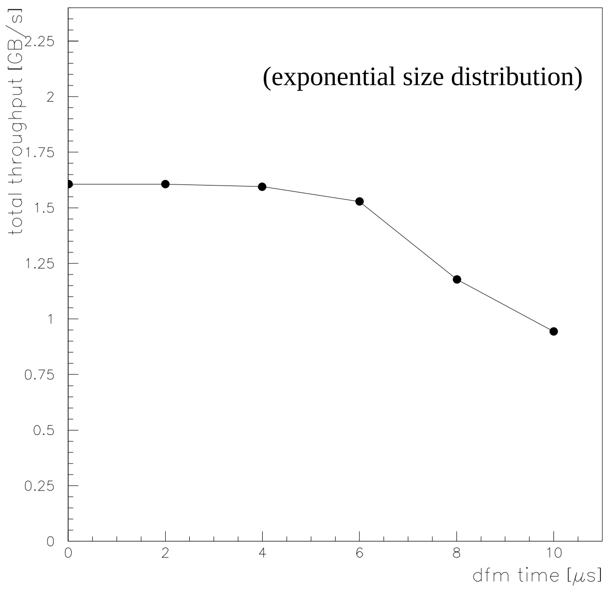
because the synchronization sending the event fragments is included and the source will only send a fragment at a time. The contribution of the size variations is calculated as before and has the same value as in the *PUSH* scheme.

TABLE 7.4. Maximum Throughput for an Ideal Data Flow Manager

Mode	Event Size (10 kByte)	Maximum Throughput [GB/s]	ε_{total} [%]	ε_{size} [%]
DFM	fix	2.590	66.3	100.0
	exp	1.606	41.1	62.0

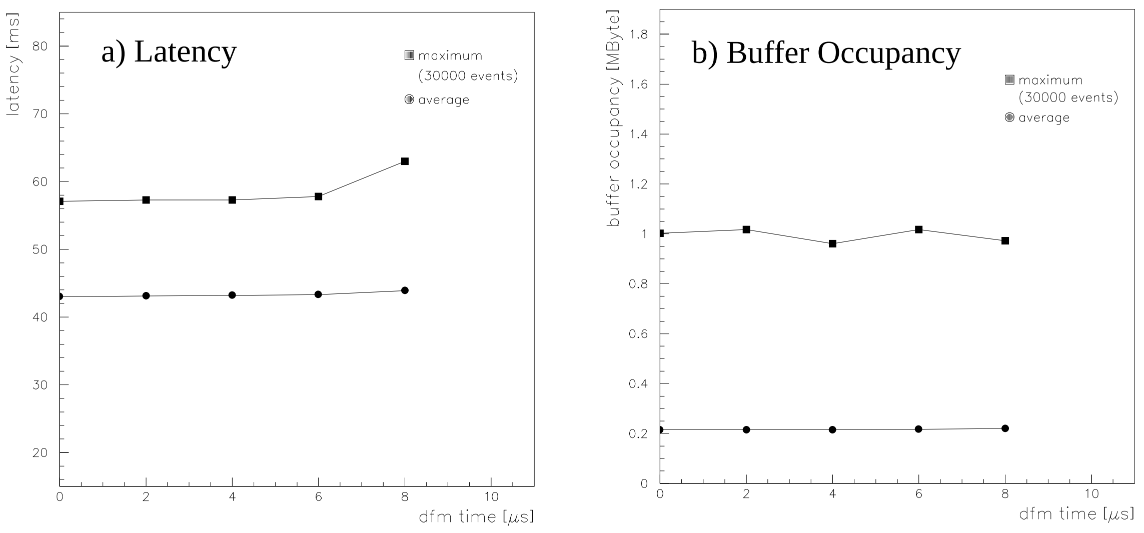
The performance, of course, changes if the data flow manager spends time. This is shown in figure 7.4 for the maximum throughput and in figure 7.5 for the latencies and buffer occupancies with exponential size distributions at 1 kHz input frequency. For small processing times the maximum throughput is unchanged and completely dominated by the overhead and size distributions. At around 6 μs this changes and the throughput decreases rapidly with increasing processing time. At a processing time of 10 μs the total throughput is below 1 GB/s which corresponds to an input frequency of less than 1 kHz.

FIGURE 7.4. Influence of Processing Time: Maximum Throughput



The latencies do not change much if the processing time increases. This is because each source makes its request independently and in barrel shifter mode they arrive at different times. The total latency is only increased by about 2% for 2 μs compared to the *PUSH* scheme. The buffer occupancy seems constant. Only when the processing time approaches 10 μs, latency and buffer occupancy become infinite and grow proportional to time because then the input frequency is larger than the maximum input frequency.

FIGURE 7.5. Influence of Processing Time (exponential size distribution, 1 kHz)



The model of the data flow manager can also be used to understand the influence of a non-negligible switching delay: if the interconnecting network has to process each source request on a single resource with a time of more than 6 µs this will have the same effect as discussed above because the source requests get serialized, the switch is not used in parallel and the performance degrades.

7.3 Traffic Shaping

It has been shown that the ideal event building system as presented in section 6.2 works in barrel shifter mode. The size variations lead to varying time slots of and some event fragments will not use it fully leading to wasted bandwidth and decreased efficiency.

Instead of wasting parts of the time slots by waiting on the next event fragment in the scheme, an attempt could be made to use the time slots more efficiently by breaking the order of the event fragments. There might be an event fragment which has already been assigned to a destination which is free. Sending this one instead will use bandwidth otherwise wasted. The efficiency could be increased and the latency and buffer occupancy reduced, because the event fragments do not have to wait for the others.

The scheme to decide which event fragment to send next is logically independent from the destination assignment scheme and works only on event fragments that have already been assigned. The basic idea behind such a scheme is to smooth out the fluctuations in the input traffic by breaking the time correlations between the event fragments. With a more uniform input traffic some of the bandwidth otherwise wasted might be recovered.

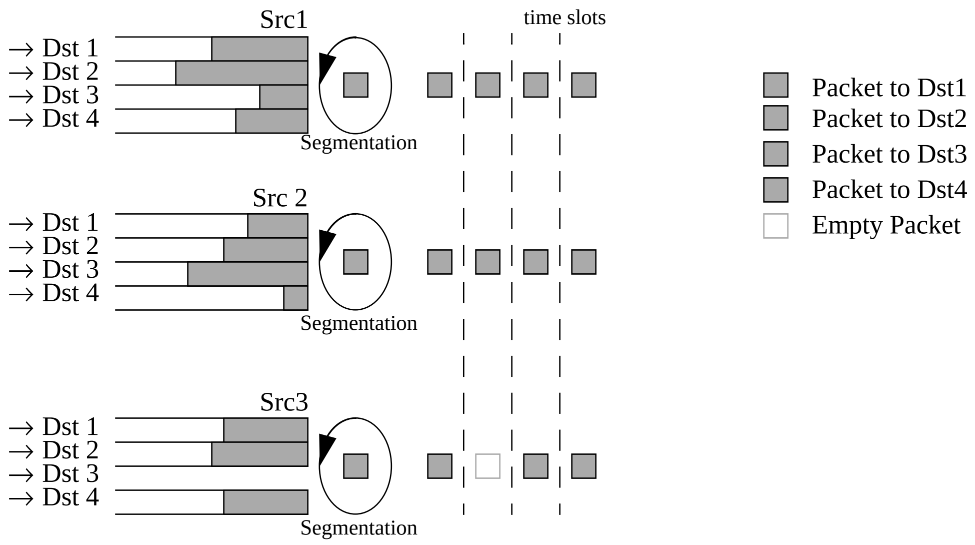
7.3.1 Definition

The term traffic shaping comes from studies on event building system based on packet-oriented networks where it is a suitable method to increase the performance of a switching network [RD31/93]. The event fragments will be segmented by the *Src* process into many packets (about 100 to 1000, depending on event fragment size and packet size in the network standard). The flow of these packets is strongly concentrated in an event building system. The concentration can be broken by shaping the input traffic according to two principles:

1. allocating an average bandwidth from any source to all destinations, such that the aggregate bandwidth at any destination does not exceed the available bandwidth at the destination;
2. breaking the instantaneous time correlation between the individual packets, thus levelling the traffic from the sources.

An example is shown in figure 7.6. Each source has a queue for event fragments going to one destination. The segmentation works on all event fragments in a round-robin manner. With the sources being synchronized, the switching network can work in barrel shifter mode. Other schemes are presented in [RD31/93]. Some of them do not even require synchronization of the sources and can easily be implemented in hardware and run efficiently. Simulations of these schemes show that they increase the total throughput, decrease the latency and buffer occupancy and decrease the packet loss, which in packet-oriented networks without flow control is an important performance parameter.

FIGURE 7.6. Packet-Based Barrel Shifter [RD31/93]



The studies carried out in this work are based on a connection-oriented network where each event fragment is sent in one connection. Schemes proposed for systems in which the event fragment is segmented into many packets cannot easily be adapted. Other schemes have to be envisaged.

7.3.2 A Possible Algorithm

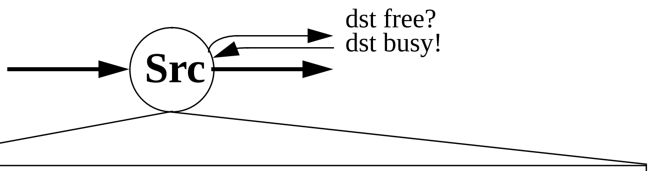
The HiPPI standard as well as the Fibre Channel (class 1) standard define flow control and allow a check wether a destination is busy or not. This could be used for a possible algorithm for traffic shaping: an event fragment whose destination is busy will be skipped temporarily and put back in the buffer behind the next waiting. This one will be tried instead.

Such a traffic shaping scheme has been implemented in the simulation by applying two modifications:

1. Sending of event fragments to the interconnecting network is no longer achieved in a simple fifo mode but rather under the control of an algorithm.
2. Every event fragment descriptor has a skip counter which is incremented each time the event fragment is skipped.

The skipping algorithm is called each time a new event fragment arrives at the *Src* process' input buffer or a previous transfer of an event fragment is terminated. It will take the next event fragment from the buffer and check if its destination is free. If it is, then it will send the event fragment immediately. If not and if there is no other event fragment waiting or the skip counter of the event fragment is at the maximum allowed value, the source will send the request and wait until the destination becomes idle. If there is another event fragment in the buffer then it will increment the skip counter of the event fragment and add it after the next one in the buffer. This will be tried instead and handled in the same way.

FIGURE 7.7. Skip Algorithm



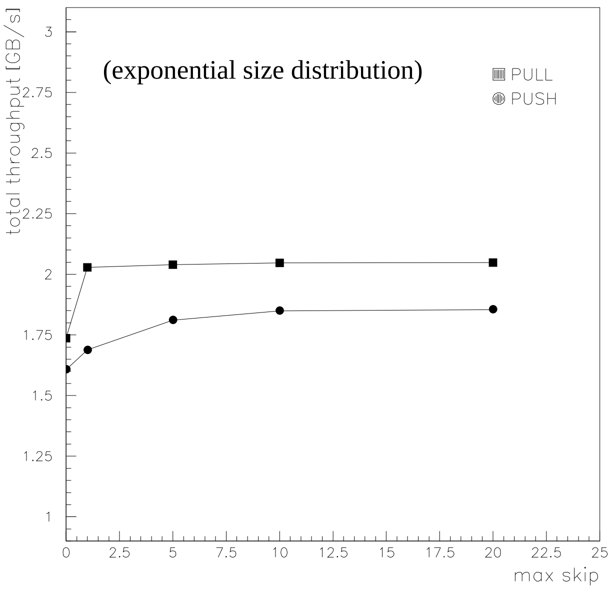
```
loop
    if Buffer not empty
        take evt from buffer
        if evt.Dst = idle or Buffer = empty or evt.Skip = MaxSkip
            wait for evt to be sent
        else
            evt.Skip++
            add evt to buffer (after first, if any)
        end if
    end if
end loop
```

The skip counter has to make sure that no event fragment can be skipped too often which would result in increasing latencies and buffer occupancies. At the same time this algorithm requires the destinations to be capable of building several full events concurrently because the order of the event fragments is changed. The time for checking a destinations status was taken into account by using the firmware overhead of 33 µs (see section 4.3.3).

7.3.3 Performance

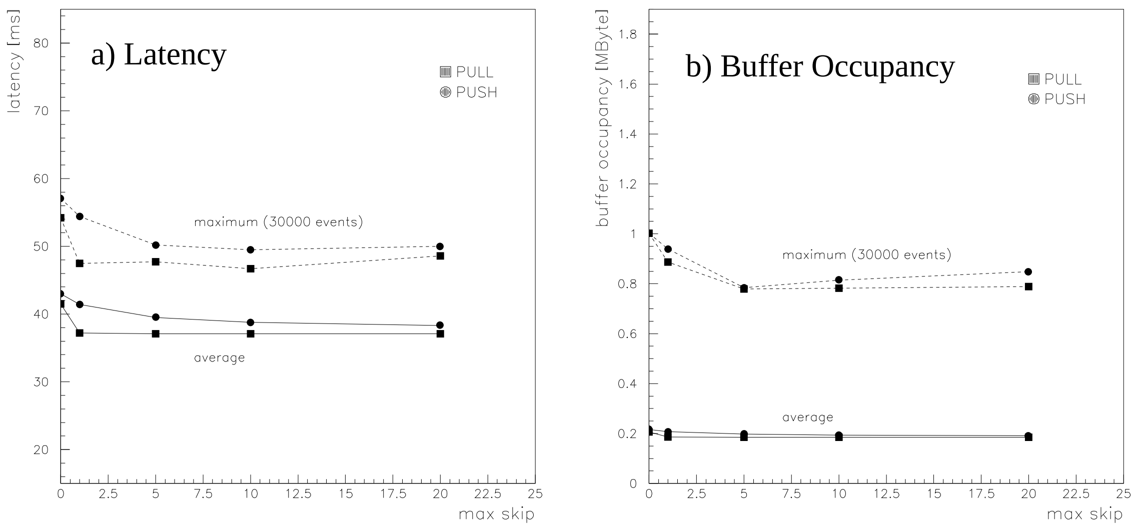
The *SKIP* algorithm was applied to two destination assignment schemes, the *PUSH* and the *PULL* scheme. The maximum allowed skip count has been varied. Figure 7.8 shows the maximum throughput obtained for different values for the maximum allowed skip count. For a value equal to 0 the original values are obtained. For values of less than 5 the maximum throughput increases and reaches a new value around a skip counter of 10. The improvement for *PUSH* is about +15%, and for *PULL* +18%.

FIGURE 7.8. Maximum Throughput with Skipping



The latencies and buffer occupancies are shown in figure 7.9. The latency decreases slightly for *PUSH*, about -8%, and for *PULL* about -11%. The buffer occupancy stays nearly constant. After having decreased to a minimum at a skip counter of 5 latency and buffer occupancy seem to increase again. For bigger values some event fragments can be skipped more often and are kept in the buffer for a longer time.

FIGURE 7.9. Skipping (exponential size distribution, 1 kHz)



Testing if the destination of an event fragment is busy or not costs some time and it will not always be useful to implement a scheme like this when the testing takes longer than the waiting would have been. This can be seen in figures 7.10 and 7.11 where the *SKIP* scheme is shown for fixed event size. Though latency and buffer occupancy at 1 kHz are not affected the maximum throughput is. When running at maximum speed, checking if a destination is busy or not costs time which could be used more efficiently just waiting.

FIGURE 7.10. Maximum Throughput with Skipping

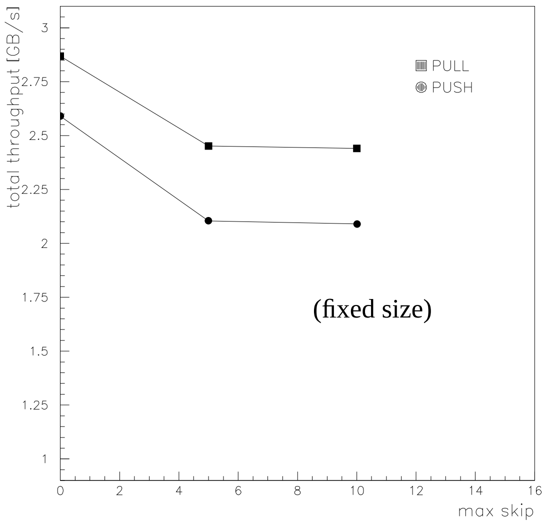
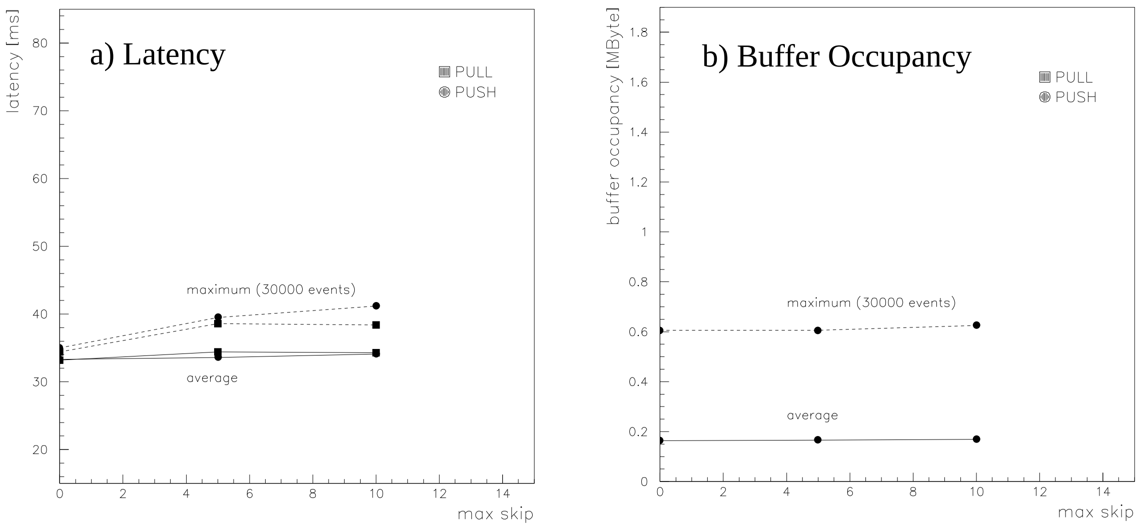


FIGURE 7.11. Skipping (fixed size, 1 kHz)



7.3.4 Summary and Outlook

The simple *SKIP* scheme has shown that it can improve the performance of the event when event sizes are varying. The time it takes to check a destination’s status, however, is important and can have a negative effect, as can be seen with fixed size samples. Wether an improvement is achieved depends on the variations in the event size.

Other more sophisticated schemes can be envisaged, where more than one event can be skipped, or the whole list of waiting and already assigned event fragments is scanned for one which can be sent immediately. These algorithms must not spend much time and must make sure that a particular event fragment is not skipped too often, or even always.

All algorithms of traffic shaping lead to disordering the event fragments. As a result the *Dst* process must be capable of building several full events concurrently which leads to more buffer space needed and more processing time for searching the *BvtBuf*. At maximum, the number of events to be built in parallel equals the number of times an event fragment can be skipped.

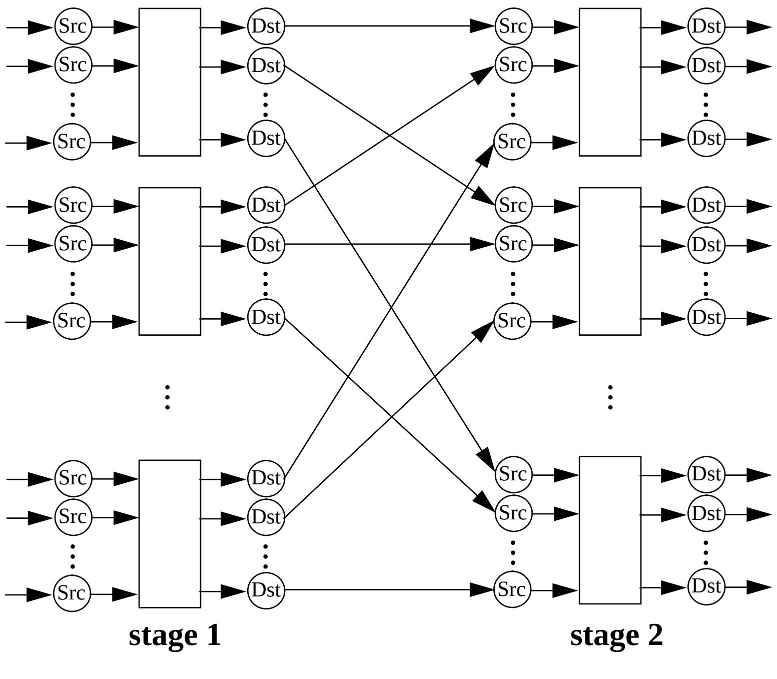
7.4 Two-Stage Event Builder

The event building systems presented so far are based on one single large switch. It is still an open question wether such large switches will be available and what their cost will be. It is not clear either if these big switches will have the same behaviour as small ones or if they will have a non-negligible switching delay or even a common internal resource which the sources send their requests to. The use of a single large switch presents also system development, integration and reliability problems. As an alternative to a single large switch a system could be build based on a network of smaller switching units.

7.4.1 Model

An architecture for a two stage event building system is shown in figure 7.12. It has 100 sources and 100 destinations and the interconnecting network consists of two stages each one being built from 10 switches with 10×10 ports. The destinations of the first stage are connected to the sources of the second stage in such a way that full interconnectivity is realized. In a realistic setup the first stage could build event fragments for a subdetector while the second would assemble full events.

FIGURE 7.12. Two-Stage Event Building System



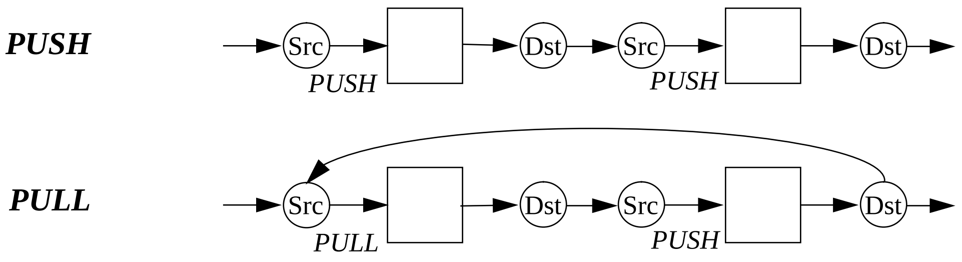
Two destination assignment schemes have been simulated with this setup:

- ***PUSH*:**

  In the *PUSH* scheme the destinations are assigned in a static way. The destination of the first stage is given in round-robin manner so that all switches are loaded in a uniform way, the second destination is assigned round-robin on each switch of the second stage, i.e. event fragments on the second stage go to the first destination of the first switch, then to the first destination of the second switch and so on. In a certain sense, the new scheme uses the old *PUSH* scheme on both stages.
- ***PULL***:

  In the *PULL* schemes the destination of the second stage make requests to the sources of the first stage. Since there is only one unique way from a source in the first stage to the destinations of the second stage, the second routing is thus defined completely and the second stage uses the old *PUSH* scheme. In order to have good load-balancing the destination requests at the start were sent in a round-robin manner as explained above.

FIGURE 7.13. Destination Assignment Schemes of the Two-Stage Event Building System



The synchronization of the *PUSH* scheme was synchronous, like before. For the *PULL* scheme the first stage is asynchronous but the second is synchronous. The SIMEB program was changed to accomplish the new setup and the new schemes. The modifications were applied in the hard-coded configuration, all data flow processes were left unchanged. The parameters were as before, i.e. 40 MB/s link speed, detailed overheads and input traffic of 10 kByte fragments on average.

7.4.2 Performance

The maximum throughput of the two-stage event building system is summarized in table 7.5 for the two destination assignment schemes and for fixed and exponentially distributed event sizes.

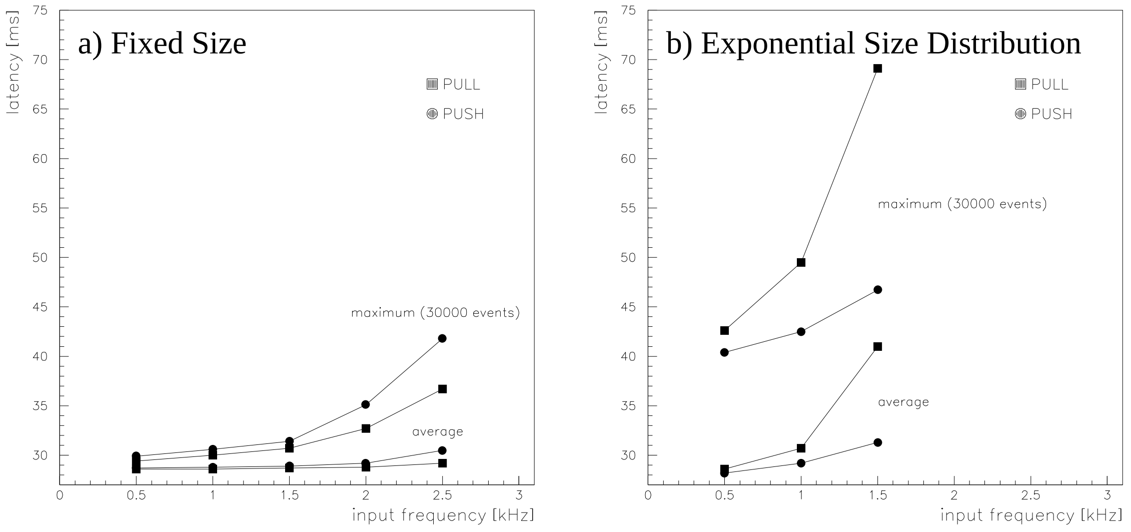
TABLE 7.5. Maximum Throughput of a Two-Stage Event Building System

Mode	Maximum Throughput [GB/s]		$\varepsilon_{fix} \equiv \varepsilon_{overhead}$ [%]	ε_{exp} [%]	$\varepsilon_{fix}/\varepsilon_{exp} \equiv \varepsilon_{size}$ [%]
	fix	exp			
PUSH	2.590	1.668	66.3	42.7	64.4
PULL	2.872	1.578	73.5	40.4	60.0

The efficiencies are in both cases similar to the single big switch. The only difference is in the *PULL* scheme: in this case the two-stage event building system is performing less well than a single big switch. This is due to the fact that the *Dst* processes make their request to the first stage and are idle while the subdetector events are built in the first stage. These results are not in contradiction to previous results presented in [Gre94] which used a random destination assignment scheme.

In contrast to the efficiency, the latency shows a difference compared to the big switch. This is shown in figure 7.14. At 1 kHz the latency for fixed and exponential size distributions is around 19 ms which is 12% and 33% less respectively than in the single switch. This reduction comes from the fact that on the first stage the transfer of 10 event fragments can be done in parallel and on the second stage the overhead time is only spent 10 instead of 100 times.

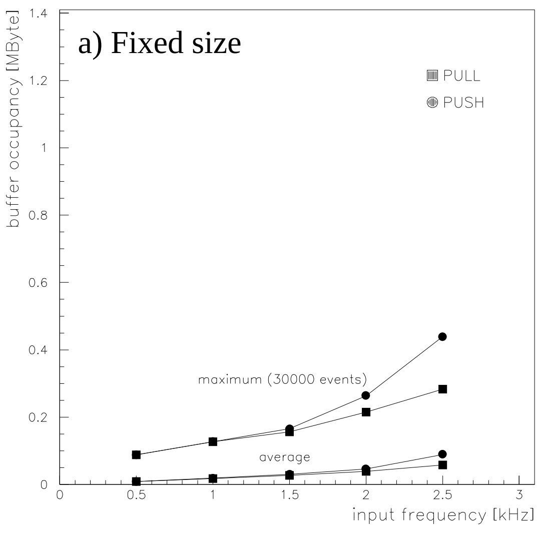
FIGURE 7.14. Latency of Two-Stage Event Building System



The buffer occupancies are shown in figures 7.15 and 7.16 for the first and second stage respectively. Since the subdetector events which are sent to the second stage are around 10 times bigger, the efficiency contribution from the overhead on the second stage is $\varepsilon_{overhead}$ = 96% and in total 45% bigger. The maximum throughput on the second stage is bigger and the buffer occupancy does not increase as fast with the input frequency as in the single switch system. It is the throughput of the first stage that actually limits the system and defines the maximum throughput.

For the exponential event fragment size distribution a buffer space of around 400 kByte on the first and around 800 kByte on the second stage must be provided to lose events with a probability of less than 10^{-5}. This buffer space has to be provided for each node of the two stages of the event building system. And though each buffer can be smaller than in the single switch (around 900 kByte, see section 6.3), in total the two-stage system needs about 30 MByte more buffer space. Regarding the development of prices for memory [Mor95] this might, however, not be an important issue. There are twice as many data flow processes as in the single stage system. The intermediate processes between the two stages require resources like buffer space and processing power. They have to run on interface boards which provide these resources and have links to the first and second stage of the interconnecting network.

FIGURE 7.15. Buffer Occupancy for First Stage



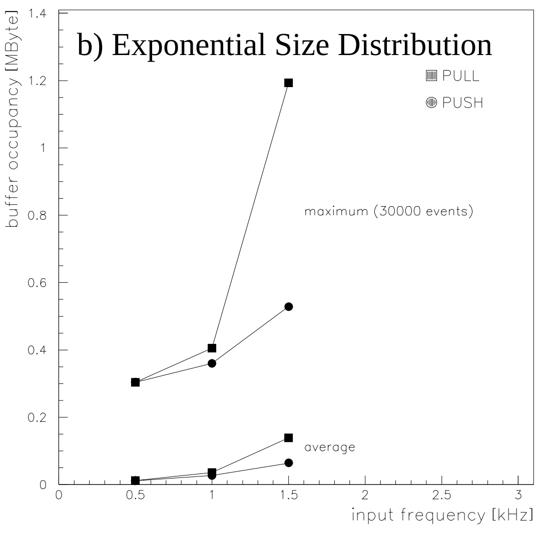
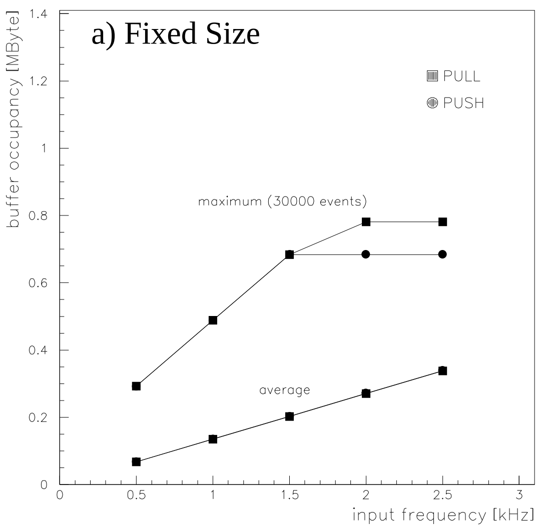
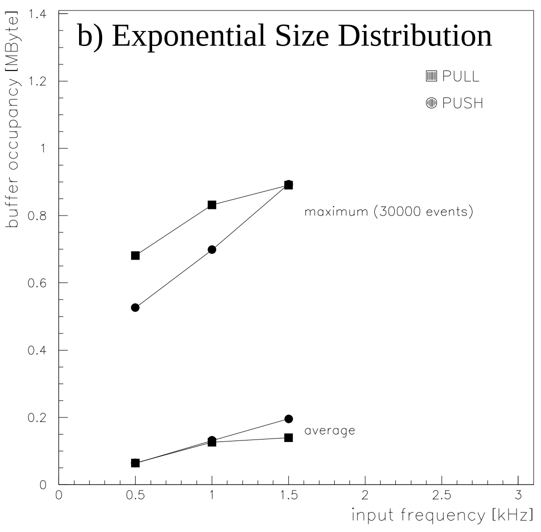


FIGURE 7.16. Buffer Occupancy for Second Stage





7.5 Conclusions

Different data flow management schemes have been simulated based on the model obtained from the prototype extrapolated to a 100×100 setup and an average of 10 kByte event fragment size per source. These data flow management schemes covered the contention resolution of the interconnecting network, the destination assignment and the synchronization between the data flow processes and the interconnecting network. The last includes in particular the order of event fragments and if the interconnecting network runs in barrel shifter mode or not. Finally a two-stage system was compared to a single big switch.

The contention resolution in the switch has no visible influence on the performance. The destination assignment schemes show only small differences depending on the different numbers of actions to be taken by the data flow processes. The exception is the random assignment which performs significantly less well than source-initiated or destination-initiated schemes. In a scheme using a data flow manager the same performance is obtained if the processing time of the data flow manager to deal with the various source and destination requests does not exceed a certain value. This is equally true if the data flow manager is implemented as a task distributed among the data flow processes. The tables would be distributed among the *Src* and *Dst* processes and synchronization messages would have to be exchanged to keep them consistent. This can equally be regarded as a requirement for an event building system which uses a single resource for the switching.

Traffic shaping schemes like those proposed for packet-oriented networks cannot directly be applied on the systems presented in this work which are connection-oriented. A possible scheme using the HiPPI protocol as an example was implemented in the simulation program. It breaks the time correlation between the event fragments by skipping fragments if their destination is busy. It was found that this can increase the performance if the event fragment sizes vary and the increase depends on the time to check the status of a destination in comparison with the variations of the event fragment sizes.

A two-stage event building system can be built from smaller switching units. The efficiency is similar to a single big switch with the destination assignment schemes used. But the latency is smaller and the second stage will perform better because it receives more uniform and bigger sized data. Nevertheless, such a system will require more data flow processes, resources and thus more interface boards. However, as the cost and availability of big switches are not known today this might be a valuable alternative to pursue.

8.0 Conclusions and Outlook

At the LHC with a centre-of-mass energy of $\sqrt{s} = 14$ TeV and a luminosity of $L = 10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$ a new field of research in high energy physics will be opened up. Amongst other interesting aspects in and beyond the standard model the discovery of the Higgs boson, if it exists, will become possible in the mass range of $90\ \mathrm{GeV} < m_H < 1\ \mathrm{TeV}$.

Indispensable for the discovery of the Higgs boson with a general-purpose experiment at the LHC is a sophisticated trigger and data acquisition system. Event building techniques required for a total bandwidth of 1 to 10 GB/s have been investigated in the present work. Parallel event building systems using commercially available high speed interconnects and switches were studied in two complementary ways: prototyping and modelling.

The prototype was based on the HiPPI standard and was built from off-the-shelf modules. Modular and scalable software was developed and the setup could be operated as a parallel event building system of limited size. It was shown that the behaviour of the switch can be described in a linear model: the transfer times are determined by an overhead and a link speed, the switching delay is negligible. The total throughput in small-scale setups was measured to be scalable with the number of destinations. The measurements were used to obtain realistic parameters and to verify the simulation program with different event size distributions and different data flow management schemes. A summary of the results is presented in section 4.5.

The simulation program is based on the linear model and the parameters obtained from prototype measurements and was used to simulate a realistic event building system of the ATLAS detector with realistic event size distributions from off-line simulations. It was shown that, under the assumptions of the model, a total throughput of 2.8 GB/s or equivalently an input frequency of 2.4 kHz can be achieved. Other simulations were used to study the influence of different parameters and of different data flow management schemes and summaries of the results are presented in sections 6.5 and 7.5.

These results show that small-scale parallel event building system can successfully be built using a commercial standard for high speed interconnects and switches. A particular technology was chosen, but the results should also be valid for similar techniques, in particular for Fibre Channel (class 1) which is a successor to the HiPPI standard. It can be expected that the technological development will improve on the parameters of link speed and overhead and provide more bandwidth. Though it is not clear if the linear model is still valid for full-scale systems the simulation results show that commercially available high speed interconnects and switches are a promising candidate for building the parallel event building system of a detector at the LHC.

There are still open questions, like how a switch-based event building system will work in a realistic multi-detector environment under changing running conditions and synchronization and error conditions unobservable with small laboratory setups. A next-generation prototype will have to show how a switch-based event building system can be integrated with different detectors and how it can work under realistic conditions of data taking, e.g. in a testbeam environment.

The model presented in this work assumed a complete statistical decoupling of the event building system from the up-stream and down-stream data acquisition system. Further studies need to show how fluctuations in the processing times at the LVL3 trigger and limited buffer space in the switch-farm-interfaces can influence the performance and what kind of data flow management will become necessary to throttle the data flow to an overloaded destination. Another assumption was the negligible switching delay which was observed in the HiPPI switch, which is not negligible with some of today’s available switches, e.g. the Ancor Fibre Channel switch has a switching delay of 67 µs [Cha95]. The manufacturers, however, promise to reduce this value in coming generations of switches to a few µs and the development of this important parameter has to be followed attentively.

The results of the studies carried out in this work are encouraging a use of commercial standards for high speed interconnects and switching elements for parallel event building systems. The final technological choice for the event building system of the LHC experiments will have to be made in a few years (~1999). Until then, further studies of the issues mentioned above will have to be addressed and the technological developments to be observed so that a decision based on the best performance over cost ratio can be made.

References

All RD13 technical notes, as well as this work itself, can be found on the WWW under URL http://rd13doc.cern.ch/welcome.html.

[Abe95] F. Abe et al. (CDF Collaboration), Phys. Rev. Lett. 74(1995) 2626;
S. Abachi et al. (D0 Collaboration), Phys. Rev. Lett. 74(1995) 2633.

[Ada92] W. Adam et al., Design and Performance of the DELPHI Data Acquisition System, IEEE Trans. Nucl. Sci. 39(1992) 166.

[Aln87] G.J. Alner et al., The UA5 High Energy $\bar{p}p$ Simulation Program, Nucl. Phys. B291 (1987) 445.

[ALI93] N. Antoniou et al. (ALICE Collaboration), A Large Ion Collider Experiment at the CERN Large Hadron Collider, CERN/LHCC 93-16, 1993.

[Amb94a] G. Ambrosini, G. Fumagalli, M. Huet, Event Format Library, RD13/TN109, Feb. 1994.

[Amb94b] G. Ambrosini et al., Modelling of Data Acquisition Systems, Proc. Computing in High Energy Physics 1994, San Francisco, California, ed. S.C. Loken, LBL 35822(1994);
G. Ambrosini et al., DAQ Simulation Library (DSL), presented at Conf. for Data Acquisition and Event Building, Fermilab, Batavia, Illinois, 1994.

[Ang94] T. Angelov et al., Performances of the central L3 Data Acquisition System, NIM A306(1991) 536.

[Arn83] G. Arnison et al. (UA1 Collaboration), Phys. Lett. B122(1983) 103;
M. Banner et al. (UA2 Collaboration), Phys. Lett. B122(1983) 476;
P. Bagnaia et al. (UA2 Collaboration), Phys. Lett. B129(1983) 130.

[ATL94] ATLAS collaboration, Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN, CERN/LHCC/94-43.

[ATL95] ATLAS Collaboration, Off-line Software Group; documentation can be found on the WWW under http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS/documents.html.

[ATM92] ITU-TSS Committee, recommendation I.150: Asynchronous Transfer Mode Functional Characteristics (available from Int. Telecom. Union, Geneva, Switzerland).

[Bai93] J.T.M. Baines et al., The Data Acquisition System of the OPAL Detector at LEP, NIM A325(1993) 271.

[Bai94] J.T.M. Baines et al., Inner Detector Layout for Technical Proposal, ATLAS/INDET-No-66, 1994.

[Bar88] E. Barsotti et al, Fastbus Data Acquisition for CDF, NIM A269 (1988), 82.

[Bar90] E. Barsotti et al., Effects of various Event Building Techniques on Data Acquisition System Architectures, FERMILAB-CONF 90/61, 1990.

[Beh93] U. Behrens et al., The Event Builder of the ZEUS Experiment, NIM A332 (1993) 253.

[Ben87] H.U. Bengtsson and T. Sjøstrand, Comp. Phys. Comm. 46 (1987) 43;
T. Sjøstrand, CERN-TH.6488/92.

[Bij93] E. v.d. Bij, HiPPI 8262/S VME to HiPPI Interface, Firmware Manual, CERN/ECP Jul. 1993.

[BIOS92] Creative Electronics Systems (CES), BIOS Manual, 1992.

[Blon94] A. Blondel, Precision Electroweak Physics at LEP, CERN-PPE/94-133, Aug. 1994.

[Boc95] R. Bock and P. LeDû, ATLAS Detector Data Read-Out Specification for Modelling a Level-2 Trigger, ATLAS/DAQ-No (in prep.), 1995.

[Bog95] A. Bogaerts et al, Event Building Protocols, ATLAS/DAQ-No-46, Aug. 1995.

[Bor93] K. Borer et al. (RD2 collaboration), A Study of a Second Level Track Trigger for ATLAS, NIM A336 (1993) 59.

[Bru93] R. Brun et al, PAW - Physics Analysis Workstation, The Complete Reference, CERN Program Library Long Write-up Q121, 1993.

[Bru94] R. Brun et al., GEANT 3.21, Detector Description and Simulation Tool, CERN Program Library Long Write-up W5013, 1994.

[Buo93a] S. Buono and J.R. Hansen, User’s Libraries for the RIO/HiPPI 8262/S Module, RD13/TN59, Apr. 1993;
S. Buono and E. Orichtchine, User’s Libraries for the RIO/HiPPI 8262/D Module, RD13/TN80, Sep. 1993.

[Buo93b] S. Buono and D. Prigent, Memory Transfer Tests using DMA Libraries, RD13/TN96, Dec. 1993.

[Car94] J. Carter et al., ATLAS Trigger Simulation User Guide, Revision 1.00, 1994.

[Cha95] F. Chantemargue et al., Fibre Channel Equipment under Test, EAST/95 (draft), 1995.

[CMS94] CMS Collaboration, The Compact Muon Solenoid, Technical Proposal, CERN/LHCC 94-38.

[Cut92] D. Cutts, Operation of the D0 Data Acquisition System, Proc. Computing in High Energy Physics 1992, Annecy, France, ed. C. Verkerk and W. Wojcik, CERN 92/07, 1992, p.262.

[Dji94a] K. Djidi, A general Graphical User Interface with MODSIM II, RD13/TN107, Feb. 1994;

K. Djidi, A DAQ User Interface for ATLAS simulations, RD13/TN118, Jun. 1994.

[EPLX91] Control Data, EP/LX reference Manual, 1991.

[Eva95] L.R. Evans, The Large Hadron Collider, subm. to Particle Accelerator Conference, Dallas, May 1995.

[Far95] Ph. Farthouat et al. (ATLAS T/DAQ Steering Group), Trigger and DAQ Interfaces with Front-End Systems: Requirements Document, ATLAS/DAQ-No (draft), Jul. 1995.

[FCS94] ANSI Working Group X3T11, Fibre Channel Standard.

[Fum95] G. Fumagalli, G. Mornacchi, Managing Multiple Interrupts in EP/LX, RD13/TN82, May 1995.

[Gre94] W. Greiman, Design and Simulation of Fibre Channel based Event Builders, RD13/TN132, Oct. 1994.

[Haw95] R. Hawkings, The Level-2 Silicon Track Trigger (T2SI) Implementation in ATRIG, ATLAS/INDET-No-46, Sep. 1995.

[Hay92] W. J. Haynes, Experiences at HERA with the H1 Data Acquisition System, Proc. Computing in High Energy Physics 1992, Annecy, France, ed. C. Verkerk and W. Wojcik, CERN 92/07, 1992, p. 151.

[Her90] J.J. Hernadez et al. (Particle Data Group), Review of Particle Properties, Phys. Lett. B239 (1990) Chapt. V.

[HIP90] ANSI X3T9.3/91-005, High Performance Parallel Interface Standard.

[HIP92] Creative Electronics Systems (CES), HiPPI/D Interface, 1992.
Creative Electronics Systems (CES), HiPPI/S interface, 1992.

[Hor95] C. Hortnagl, S. Hunt, SIMDAQ Users Guide, ATLAS/DAQ-No-41, Jan 1995.

[Hun95] S. Hunt et al, SIMDAQ - Simulation for LHC DAQ System, subm. to Real-Time Conference 1995, Michigan.

[Jon93] R. Jones et al., Run Control Requirements, RD13/TN60, Mar. 1993.

[Kle75] L. Kleinrock, Queueing Systems, Vols. I&II, J. Wiley and Sons, 1975.

[Koz94] V. Kozlov, Functional Simulation of Detector, Front-End and Read-Out Parts of a LHC-like DAQ Architecture, RD13/TN120, Jun. 1994.

[Let94] M. Letheren et al. (RD31 Collaboration), An Asynchronous Data-Driven Event Building Scheme based on ATM Switching Fabrics, IEEE Trans. Nucl. Sci. 41 (1994).

[Map95a] L. Mapelli, Global Architecture for the ATLAS Data Acquisition and Trigger, Part I - Functional Model, ATLAS/DAQ-No-22 and RD13/TN136, Jan. 1995;

L. Mapelli, ATLAS DAQ and Trigger, ATLAS Physics Workshop, Trest, Czech Republic, Jun 12-16, 1995.

[Map95b] L. Mapelli, private communications, 1995.

[MOD91] MODSIM II, The Language for Object-Oriented Programming, CACI Products Company, La Jolla, California, 1991.

[Mor92] G. Mornacchi, Data Flow Protocol Prototype, RD13/TN15, May 1992.

[Mor94] G. Mornacchi, Preliminary Performance Study of LVL3-Like MP Architectures, ATLAS/DAQ-No-23, 1994.

[Mor95] G, Mornacchi, private communications, 1995.

[Par90] C.F. Parkman, Summary of Backplane Bus Characteristics, CERN-CN 90-26, 1990.

[Pat94] J. Patrick et al., The CDF ULTRANET Data Acquisition System, Proc. Computing in High Energy Physics 1994, San Francisco, California, ed. S.C. Loken, LBL 35822(1994).

[Per94] J. A. Perlas, Design and Implementation of a VME based Event Building Protocol for the new ALEPH Data Acquisition System, Proc. Computing in High Energy Physics 1994 in San Francisco, California, ed. S.C. Loken 1994, LBL 35822(1994).

[POS90] IEEE (POSIX) 1003.4, Real-Time Extension for Portable Operating Systems.

[Pra92] A. van Praag et al., HiPPI Developments for CERN Experiments, IEEE Trans. Nucl. Sci. 39(1992) 880.

[RAID92] Creative Electronics System, RAID Processor Board, 1992.

[RD11/95]RD11 Collaboration (EAST), Status Report: Embedded Architectures for Second-Level Triggering (EAST), CERN/LHCC 95-45, 1995.

[RD13/95]RD13 Collaboration, Status Report of a Scalable Data Taking System at a Testbeam for LHC, CERN/LHCC 95-47, 1995.

[RD24/95]RD24 Collaboration, RD24 Status Report, Application of the Scalable Coherent Interface to Data Acquisition at LHC, CERN/LHCC 95-42, 1995.

[RD31/93]RD31 Collaboration, RD-31 Status Report ‘93, NEBULAS: High Performance Data-Driven Event Building Architectures based on an Asynchronous Self-routing Packet-Switching Network, CERN/DRDC/93-55, 1993;
I. Mandjavidze (RD31 Collaboration), Review of ATM, Fibre Channel and Conical Network Simulation Results, Proc. Int DAQ Conf., Fermilab, Batavia, Illinois, 1994.

[RD31/95]RD31 Collaboration, RD-31 Status Report '95, NEBULAS: High Performance Data-Driven Event Building Architectures based on Asynchronous Self-Routing Packet-Switched Networks, CERN/LHCC 95-14, 1995.

[RIO91] Creative Electronics System (CES), RIO Board, 1991.

[Rue89] W. v. Rueden, The ALEPH Data Acquisition System, IEEE Trans. Nucl. Sci. 36 (1989), no 5, 1444.

[SCI92] ISO/ANSI/IEEE Standard 1596-1992, Scalable Coherent Interface.

[SIM90] E.C. Russell, Building Simulation Models with Simscript II.5, CACI Products Company, La Jolla, 1990.

[Spi93] R. Spiwoks, Requirements of a Simulation Program for DAQ Modelling, RD13/TN89, Oct. 1993;
R. Spiwoks, Proposal of a Library and Skeleton Program for DAQ Simulations, RD13/TN90, Oct. 1993;
R. Spiwoks et al., DAQ Simulation Library (DSL), A Reference Manual, RD13/TN111, Jun. 1994.

[Spi94a] R. Spiwoks, Modelling of the RD6/RD13 Testbeam Setup, RD13/TN97, Jan. 1994.

[Spi94b] R. Spiwoks, Requirements of an Event Building System, RD13/TN111, Apr. 1994.

[Spi94d] R. Spiwoks, A Library for the RAID ZCIO Timers, RD13/TN45, Oct. 1994.

[Spi95a] R. Spiwoks, The RD13 HiPPI/D Firmware (DstFmw), RD13/TN129, Mar. 1995;
R. Spiwoks, The RD13 HiPPI/S Firmware (SrcFmw), RD13/TN143, Mar. 1995.

[Spi95b] R. Spiwoks, User Library for the HiPPI/D module, RD13/TN130, Apr. 1995;
R. Spiwoks, User Library for the HiPPI/S module, RD13/TN144, Apr. 1995.

[SWI92] Input Output Switching Corporation, HiPPI Switch 1992.

[Vel77] M. Veltmann, Ann. Phys. (NY) B8 (1977) 475;
B.W. Lee, C. Quigg and H. Thacker, Phys. Rev. D16 (1977) 1519.

[VER90] VERILOG is a product of Cadence Design Systems Inc., also IEEE Standard Committee 1364.

[Ver93] J.C. Vermeulen, Simulation of Data Acquisition and Trigger Systems in C++, EAST/93-22, 1993.

[VIC92] Creative Electronics System, VIC 8252, 1992.

[VHD87] IEEE VHDL Language Reference Manual, IEEE Standard 1076-1987.

[Wu87] S.L. Wu et al., Proc. ECFA Workshop on LEP200, eds. A. Böhm and W. Hoogland, Vol. II (1987).