ALSIM

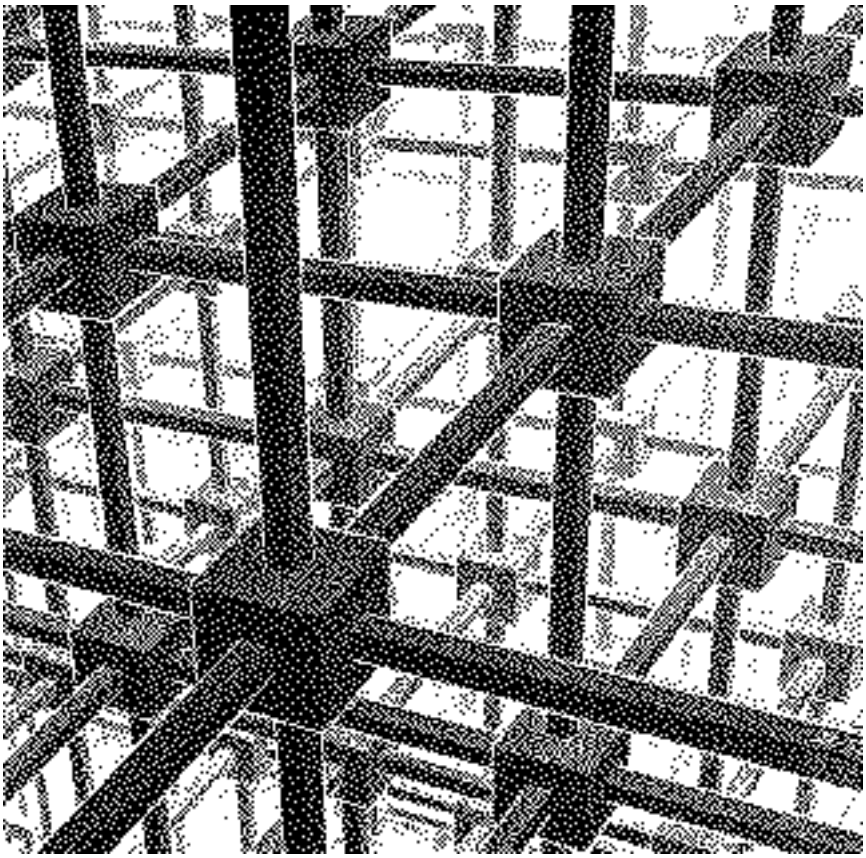
Reference Manual and User's Manual for ALICE DAQ Modelling and Simulation

B. Wu, B. Kvamme, B. Skaali

Department of Physics
University of Oslo
0316 Oslo, Norway

G. Harangozo, P. Vande Vyvre

ECP/DS, CERN
1211 Geneva 23
Switzerland



Version 0.21

Copyright Notice

ALSIM-Reference Manual and User's Manual

Requests for information could be addressed to:

Bin Wu
Department of Physics
University of Oslo
P.O. Box 1048 Blindern
0316 Oslo, Norway
Tel: +47 22 856188
Fax: +47 22 856422
Email: bin.wu@fys.uio.no

OBS!

This is a *DRAFT* of *ALSIM-Reference Manual and User's Manual*. It is subject to change without notice. THERE IS NO WARRANTY FOR THE PROGRAM. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. 7/11/97

Cover art: M.C.Escher's Gallery

Table of Content

Chapter 1. About This Manual

This manual describes the details of a simulation environment *ALSIM* for the ALICE DAQ system. It could be served as the reference manual as well as the user's manual.

Organization of this manual

The first chapter, i.e. this chapter introduce the structure of this manual and some conventions, etc.; Chapter 2 provides background and introduction of ALSIM; Chapter 3 explains the system requirement for each component and Chapter 4 guides the readers to run their first simulation and display the result step by step; Chapter 5 describes how the models for these components are built and how they are connected. Chapter 6 shows how to write configuration files that are used as input files to the simulation system and how the input file is parsed by the preprocessor; Chapter 7 explain in detail how the histogram package works. Chapter 8 analyses results by go through an example we used in Chapter 4. Chapter 9 illustrates the Graphics Presenter and how to use it. Chapter 10-12 are the chapters that dedicated to technology implementation of different switching networks, namely SCI, ATM and Fibre Channel. Chapter 13 is case studies, which gives readers a feeling on how we could use ALSIM to get interesting results out of simulation. Rest of this manual are Appendix A ALSIM source file and compilation, and Appendix B Configuration file syntax.

How to use this manual

This manual is not a long one, and we tried to organize the chapters in a more understandable sequence. For those who have a SUN-SPARC station and have copied our executable code, "Getting Started" is probably the right chapter to start with, otherwise you have to go through Appendix A first and install ALSIM and compile it. MODSIM II (release 1.9.8 or later) must be installed. "Getting Started" is also a chapter for someone who wants to get a feeling on how ALSIM works. Chapter 5-12 are the real "meat" in this manual, and are the guides for serious users. Appendix A provides a list of the source codes and a guide on how to compile them. Appendix B gives a full description of syntax of the configuration file.

Conventions used in this manual

The following conventions are used in this manual:

Bold	Bold text donates that the text is important
italic	donates the keyword in configuration file only.
helvetica	The characters and text that are sections of code, programming examples, and syntax examples.
%	prompt on screen before keyboard input
`courier`	The characters and text that are to be literally input from the keyboard.
<>	The text in between are a substitution of other text

Acronyms

ANSI: American National Standards Institute

IEEE: Institute of Electrical and Electronics Engineers, Inc.

Related document and references

[1] *CACI Products Company,* "MODSIM II, The Language for Object-Oriented Programming", Reference Manual, User's Manual, Tutorial, CACI Products Company, La Jolla, CA 92037

[2] *Application Software Group, Computing and Networks Division, CERN*, "PAW, the complete reference, version 1.14", CERN Geneva, Switzerland, July 1992

[3] *P. Vande Vyvre, et al.* "ALICE Data Acquisition System Program of R&D Work", ALICE Note 95-03, CERN, March 22, 1995

[4] *B. Wu, B. Kvamme, B. Skaali.* "ALSIM-Proposal for an ALICE DAQ Modelling and Simulation Project", ALICE note 1994-34 DAQ, CERN Geneva, Switzerland, Dec. 1994

[5] *D. Calvet,* "Using ATM Switches in Model SIMDAQ", ATLAS DAQ Note XXX, August 21 1994 (DRAFT)

[6] *A. Bogaerts, B. Wu,* "The SCILab Cook Book", RD24 Note, CERN, Geneva, Switzerland, July 1994

[7] *Christian Hörtnagl,* "MODSIM II Histogramming Package for use with ATLAS Trigger/DAQ-Simulations" Version 1.0, July 14, 1994

[8] *B. Wu,* "FCSIM - A Fibre Channel Simulator in MODSIM II", Technical Report UIO/PHYS/95-10, August 1995

ftp email service

A version for SUN SPARC workstations, including all examples mentioned in this manual, is available by ftp file transfer.

Currently, anonymous ftp and WWW services are available in Oslo. For anonymous ftp, the server is ftp.fys.uio.no, /pub/DAQ/ALICE.

The URL for WWW page is: http://www.fys.uio.no/project/alice/alice.html

Chapter 2. Introduction

Background

The Heavy Ion Experiment ALICE is one of the three experiments at the LHC at CERN. Modelling and simulation of such a large scale data acquisition system is necessary in order to study the behaviour of the system. An object-oriented approach will result in a flexible and open-ended simulation environment. Models of components can easily be added, and various configurations can be implemented dynamically by means of description files. A generic model of the system can be expanded into technology dependent models [4].

For this purpose, we started to write an ALICE Data Acquisition System Simulator -- ALSIM in the programming language MODSIM II [1] onwards October 1994. With the experience in MODSIM based simulations, especially from the participation in the ATLAS DAQ modelling programme, we have our frame work and the simplest model working in months.

Goal

The goals of *ALSIM* are,

- To investigate the feasibility of the proposed DAQ architecture. On a longer term, to lay a basis for evaluating the evolution of the DAQ system architecture.
- To provide a general understanding of the total system behaviour and the system components, and how the physics and read-out requirements influence the design of the system.
- To provide guidelines for further investigations and prototype developments for choosing the right technology for interconnects and switches, such as ATM, Fibre Channel, SCI, etc.
- To provide simulation results for the Technical Proposal.

Platforms

The current version of ALSIM has been tested on SUN SPARC workstations. Both the MODSIM II and GNU C-compilers are available for a large variety of platforms. The input pre-processor is a UNIX shell script using sed, awk and the C preprocessor. Filters to transform the ASCII output files from `ALSIM` into input suitable to many spreadsheet programs (e.g. Excel) or the CERN PAW [2] data presentation package are also simple UNIX scripts based on sed and awk. Therefore, for running ALSIM, one needs either a SUN SPARC station, or any machine with MODSIM II (release 1.9 or later) and GNU C-compiler installed.

On SPARC Stations with SunOS 4.1, graphics display requires the environment variable LD_LIBRARY_PATH to be set in such a way that OpenWindows-Libraries are searched for before X-Libraries. Otherwise an error message “undefined symbol: _XtQString” will occur.

MODSIM II

MODSIM is the language chosen for modelling, for several reasons. It is an object-oriented language that is specially designed for discrete-event simulation. It is also a simulation language widely used at CERN and other research institutes related to CERN, including the University of Oslo. Furthermore, it was chosen for modelling the ATLAS DAQ systems.

A multi-user licence is installed on *ptsun.cern.ch* for access at CERN. The MODSIM compiler can also be obtained at a rate of $5000 per licence. It is available on a large number of platforms, like SUN SPARC Stations, HP machines, and even PCs.

ALSIM

ALSIM accept a system description in ASCII format as the input file. The input pre-processor is a UNIX shell script using sed, awk and the C preprocessor.

The output of the simulation is in ASCII format. Filters can transform the ASCII output files into input that suits many spreadsheet programs (e.g. Microsoft Excel) or the CERN PAW data presentation package. PAW - Physics Analysis Workstation, is an interactive utility for visualizing experimental data on a computer graphics display. PAW can do a variety of analysis tasks on physics data, which are typically statistical distributions of measured particle interactions. We will see PAW plots later. To automate the processing of simulation results, we have a set of PAW files containing pre-defined macros. UNIX tool “xgraph” is also used for displaying results.

A graphic interface is also provided to dynamically display the traffic load and buffer status. The whole simulation procedure is illustrated in Figure 1.

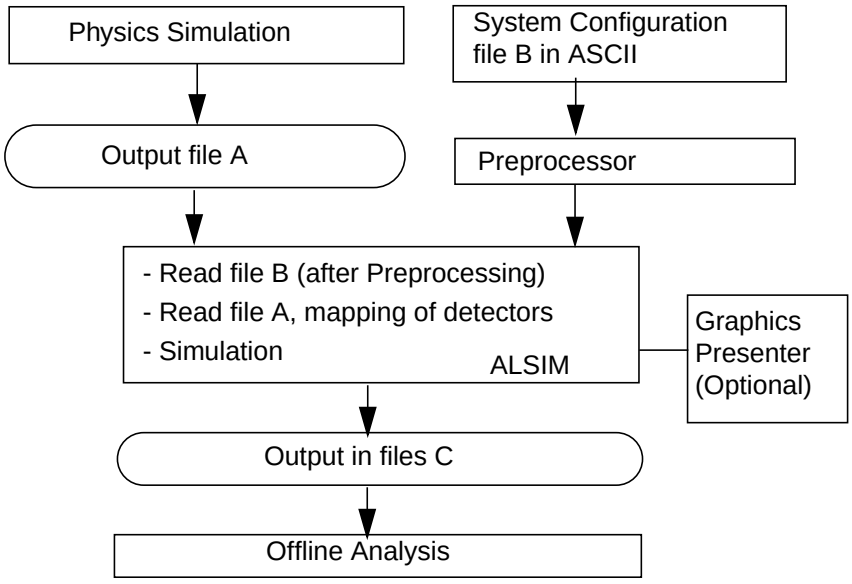
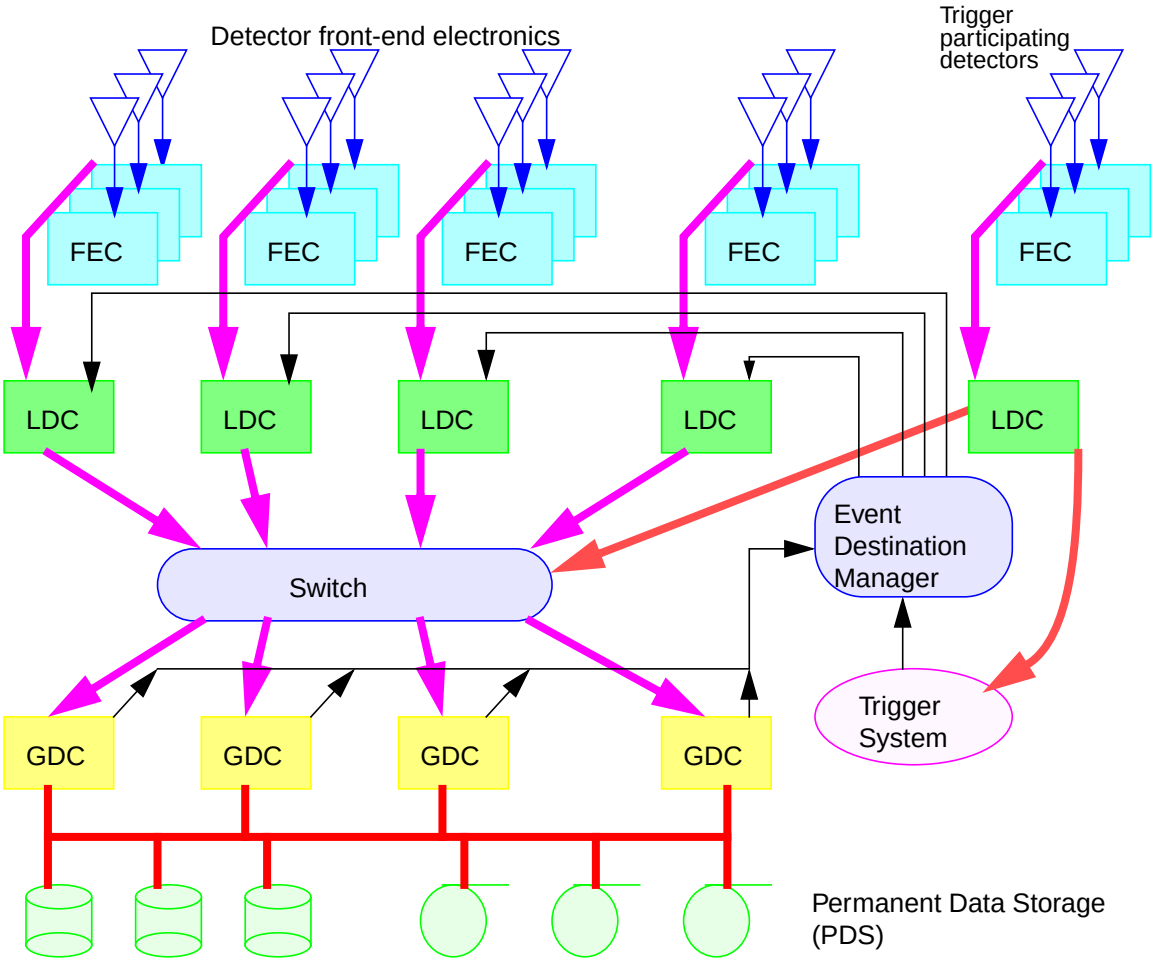


FIGURE 1. The block diagram of ALSIM simulation environment

Chapter 3. System Requirements

General architecture of an ALICE data acquisition system

The basis for the project is the proposed ALICE DAQ system as described by P. Vande Vyvre, et al. in “ALICE Data Acquisition System Program of R&D Work” [3]. The general architecture is shown in Figure 2. The system will perform front-end data read-out, event building, data reduction and data storage. An estimated trigger rate of up to 50 events/s of a size of between 30 and 50 MBytes results in a total throughput of up to 2.5 GBytes/s. The current proposal does not include a 2. level trigger. If such an intermediate trigger can be efficiently implemented the throughput could be significantly reduced.



FEC: Front-End Crate; LDC: Local Data Concentrator; GDC: Global Data Collector

FIGURE 2. Proposed general architecture of the ALICE data acquisition system

In the following paragraphs we will try to summarize the parameters which should be included in the simulation and that should be easily configurable. However, this does not imply that they are built-in in the current version of ALSIM.

Physics input to DAQ simulation

The basis for the DAQ simulation studies will be input from Physics Simulations in the form of data files which contain patterns of events for the required detector granulation.

The text format trigger data file in ASCII will have a free-format structure, consisting of keywords and data, and some special characters for easing the parse work by the DAQ simulator. However, due to the size of each file (can be as large as tens of MBytes for an event), the exchange format of column wise ntuples (CWN) is proposed and under investigation.

The Front-End Crate

The FEC is modelled after the front-end electronics required for the read-out of each subdetector. Parameters like output channel latency, output channel bandwidth could be defined for each individual FEC in the configuration files. A BasicDAQPort (called FECToLDCPort) is used to pass sub-event in each FEC to the corresponding LDC.

The parameters of the FEC would be the following:

- Output channel latency
- Output channel bandwidth

The Local Data Concentrator

The tasks of a LDC is to collect event data from the corresponding FEC, and to find the correct destination, i.e. GDC. It then sends the data to the switch matrix. Logically, it also contains a port connected to the EDM.

In order to perform the task of assigning the right GDC to the incoming sub-events, it has a table containing the reserved GDC ids (identifier) for each sub-event. The content of the table is filled by the broadcast messages from EDM which contains the identity of a GDC. The LDC also includes a processor model (e.g. VME processor) for the future extension of data processing/compressing. A Dual-Port-Memory buffer, typically of the size of NumberOfGDCs * SubEventSize, is required to buffer the sub-event for possible processing operation on the sub-event and to wait for the assignment of a free GDC by the EDM.

The parameters of the LDC that can be specified in the configuration file, are as follows:

- Input/output channel latency
- Input/output channel bandwidth
- Memory size
- Latency of the data inside the memory. This will represent the time of processing of the data inside the processor

An operating system model is used to control and coordinate the different parts that involve the processor.

The high speed links and the Switch - Event builder part I

The parameters are the following:

- The high speed link latency
- The high speed link bandwidth
- The switch latency

- The switch aggregated bandwidth

In addition, the protocol which will handle the distribution of events to the different output of the switch should be taken into account. Several strategies are possible: first free, round-robin etc.

The possible implementations are well known: ATM, Fibre Channel (FC) and SCI.

Event Destination Manager - Event builder part II

The task of Event Destination Manager (EDM) is to send all the messages of the same event to the same and free GDC. Two schemes could be implemented, the “polling”-scheme and the “broadcasting”-scheme. They will be discussed in detail later.

The Global Data Collector

A GDC receives data (after event building) from LDCs through the switch, formats and records them onto a PDS. The most probable implementation is one or several commercial processors (workstation or PC) sharing the same memory and the same input and output channels. The internal structure of GDC is made in a way that we can put in the processes that are necessary to simulate such a commercial processor. The important parameters are,

- Memory size
- Input/Output channel latency
- Input/Output channel bandwidth
- Latency of the data inside the memory for processing

The GDC model has the functionality of collecting all the sub-events that belong to the same event and sends a free signal to EDM when it has finished in processing this event. It is therefore necessary to have a large buffer space, with a minimal size of the largest single event.

The Permanent Data Storage

The Permanent Data Storage can be reduced to a simple data sink of a given bandwidth with following parameters,

- Cartridge setup time
- Cartridge capacity
- Data recorder bandwidth

In addition, the simulation could also include the time taken for the change of volume, etc.

Chapter 4. Getting Started

If you do not have a SUN SPARC compatible workstation you will not get going so quickly and you will have to read the appendices first. Or you have to install MODSIM on your local platform. MODSIM II (release 1.9.8 or later required) is available on different platforms. In Appendix A you will find all the source files of ALSIM that should be copied and the instructions to compile the `ALSIM` in case you use other platforms.

A simple starting model

The simplest model one can simulate in ALSIM is a system with two FECs, two LDCs, a switch, two GDCs, two PDSs and an EDM model as in Figure 3. A Event Generator (also called Data Generator, DG) will generate events and the events are split into messages (sub-events) before they are pipelined in the FECs. The DG also sends the Level 1 Trigger information to the EDM. The DG will be replaced by a physics simulation model in the future.put them in FECs. The FECs push messages to corresponding LDC and then through the switch. The messages are finally received by GDC and stored in Permanent Data Storage devices (PDS). The switch is a generic model which is implemented by point-to-point links. It could be built from various switching networks in the future.

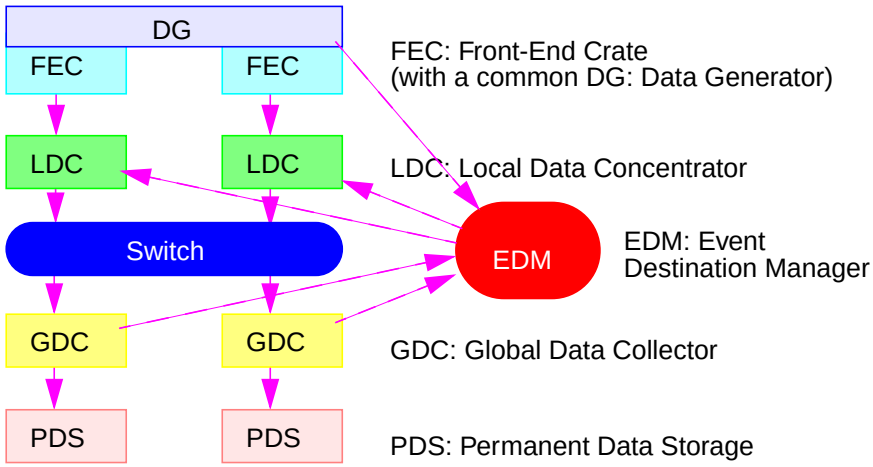


FIGURE 3. The simplest physical model to start with

The corresponding simulation model simulation objects are in Figure 4. Eight port objects are used to link the

whole system together. They are child objects of the objects FEC, LDC, Switch, GDC and DS, respectively.

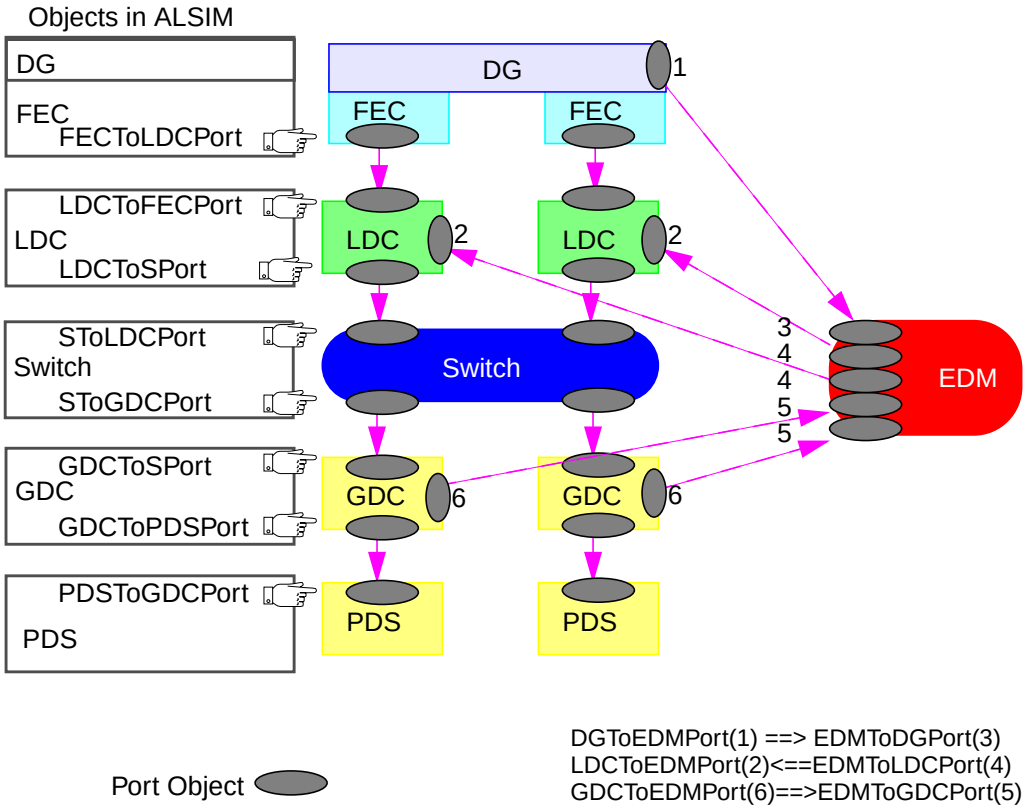


FIGURE 4. Modelling Objects for Figure 3

Simulation setup and objects

The configuration of a whole system is expected to be specified in an ASCII input file to the simulator. We name the simulator ALSIM. The configuration file format is shown in Figure 5. This is an incomplete file.

```
/* ALSIM CONFIGURATION PROGRAM */
/* USE alsim alice.conf TO START SIMULATION */
graphics 1000 /* graphics updateRateInNano */
debuglevel    2    /* debug level 2 */
simtime    5000000 /* simtime */
#define DAQSYSTEM 10003   /* DAQ System */
#define DAQSwitch 10004   /* Generic Switch */
#define DG 11000   /* Data Geneator */
#define EDM 11001   /* Event Destination Manager */
#define FEC 11002   /* Front End Crate */
#define LDC 11003   /* Local Data Concentrator */
#define GDC 11004   /* Global Data Collector */
#define PDS 11005 /* Permanent Data Storage */
#define Switch DAQSwitch
#define OSObject 12001   /* Operative System */
#define LDCStoreProcess 12005 /* LDCStoreProcess */
#define GDCStoreProcess 12006 /* GDCStoreProcess*/
#define BasicDAQPort    13001   /* Generic Port */
#define FECToLDCPort    BasicDAQPort
#define LDCToFECPort    BasicDAQPort
#define LDCToSPort      BasicDAQPort
#define SToLDCPort      BasicDAQPort
#define SToGDCPort      BasicDAQPort
#define GDCToSPort      BasicDAQPort
#define GDCToPDSPort    BasicDAQPort
#define PDSToGDCPort    BasicDAQPort
#define LDCToEDMPort    BasicDAQPort
#define EDMToLDCPort    BasicDAQPort
#define GDCToEDMPort    BasicDAQPort
#define EDMToGDCPort    BasicDAQPort
#define EDMToDGPort     BasicDAQPort
#define DGToEDMPort     BasicDAQPort
/* define object classes */
#define GenOUT 1 /* define port types */
#define GenIN 2 /* define port types */
#define GenBI 3 /* define port types */

/*  ------- Message Generator ----------- */
object DG * /*Define one Data Generator*/
  message 1000 10000 /* message size in byte and message
interval in ns */
/* define port connecting DG to EDM */
  object DGToEDMPort DGToEDMPort1 EDMToDGPort1
            EDMToDGPort GenOUT 1
  end object
end object  /* It is the message-supplier of all FECs */

/*  ------- Event Manager ----------- */
object EDM *   /* define one but only one Event Manager */
  object EDMToDGPort EDMToDGPort1 DGToEDMPort1
            DGToEDMPort GenIN 1
  end object /* define port connecting EDM to DG */
  object EDMToLDCPort EDMToLDCPort1 LDCToEDMPort1
            LDCToEDMPort GenBI 1
  end object /* define port connecting EDM to LDC */
  object EDMToLDCPort EDMToLDCPort2 LDCToEDMPort2
            LDCToEDMPort GenBI 2
  end object /* define port connecting EDM to LDC */
 object EDMToGDCPort EDMToGDCPort1 GDCToEDMPort1
            GDCToEDMPort GenIN 1
  end object /* define port connecting EDM to GDC */
 object EDMToGDCPort EDMToGDCPort2 GDCToEDMPort2
            GDCToEDMPort GenIN 2
  end object /* define port connecting EDM to GDC */
end object  /* Event Manager */

/*  ------- FECrate ----------- */
object FEC *   /* define 1 FE Crate */
  object FECToLDCPort FECToLDCPort1 LDCToFECPort1
            LDCToFECPort GenOUT 1
  end object /* define port connecting FEC to LDC  */
end object  /* Front End Buffer */

/* ---------- Local Data Concentrator -------- */
object LDC  *
  /* define port connecting LDC to FEC */
  object LDCToFECPort LDCToFECPort1 FECToLDCPort1
            FECToLDCPort GenIN 1
  end object
/* define port connecting LDC to Switch  */
  object LDCToSPort LDCToSPort1 SToLDCPort1 SToLD-
            CPort GenOUT 1
  end object
/* define port connecting LDC to EDM */
  object LDCToEDMPort LDCToEDMPort1 EDMToLDCPort1
            EDMToLDCPort GenBI 1
  end object
  logicalPort dataPortIn LDCToFECPort1 LDCToFECPort
  logicalPort dataPortOut LDCToSPort1 LDCToSPort
  object OSObject *
    clock 1        /* 1 ns clockspeed */
    object LDCStoreProcess *
      processTime 100       /* 100 ns processTime */
      bufferSize  10000
    end object
  end object /* OS Object */
end object  /* LDC */
/* ---------- Switch ---------- */
object Switch *  2 2  /* define  Switch 2 X 2 */
  /* define ports connecting Switch to LDC */
  object SToLDCPort SToLDCPort1 LDCToSPort1
            LDCToSPort GenIN 1 end object
  object SToLDCPort SToLDCPort2 LDCToSPort2
            LDCToSPort GenIN 2 end object
  /* define ports connecting Switch to GDC */
  object SToGDCPort SToGDCPort1 GDCToSPort1
            GDCToSPort GenOUT 33 end object
 object SToGDCPort SToGDCPort2 GDCToSPort2
            GDCToSPort GenOUT 34 end object
end object  /* Switch */
/* ---------- Grobal Data Collector ------- */
object GDC *  /* define 1 GDC */
  /* define port connecting GDC to Switch */
  object GDCToSPort GDCToSPort1 SToGDCPort1 SToGD-
            CPort GenIN 1 end object
/* define port connecting GDC to  PDS*/
  object GDCToPDSPort GDCToPDSPort2 PDSToGDCPort2
            PDSToGDCPort GenOUT 2 end object
  /* define port connecting GDC to EDM */
 object GDCToEDMPort GDCToEDMPort2 EDMToGDCPort2
            EDMToGDCPort GenOUT 2 end object
  logicalPort dataPortIn GDCToSPort1 GDCToSPort
  logicalPort dataPortOut GDCToPDSPort1 GDCToPDSPort
  object OSObject *
    delay 100
    clock 1        /* 1 ns clockspeed */
    object GDCStoreProcess *
      processTime 100       /* 10000 ns processTime used for for-
matting, etc. */
      bufferSize  20000       /* messageSize*systemSize*10 Mbyte
as default */
    end object
  end object /* OS Object */
end object  /* GDC */
/*------------Data Storage Device -------------*/
object  PDS * /* define 1 PDS */
  delay 1000
  /* define port connecting GDC to PDS */
  object PDSToGDCPort PDSToGDCPort1 GDCToPDSPort1
            GDCToPDSPort GenIN 1
  end object
end object  /* PDS 1 */
end object /* root */
```

FIGURE 5. alice.conf (Incomplete!!!)

Despite being relatively simple as configurations go, there are a number of features worthy of discussion. First, and import, if not foremost, is that much of the configuration file is comment. Definitions are also of important since they can improve readability of the file. The syntax of comment and definition are the same as those in the C programming language. Rest of the statements will be explained later. This example is not a complete one as some of the functions are not listed. The purpose is to give a taste on how the files could be look like.

Run

To run the simulation, simply type

```
% alsim alice.conf > & alice.out &
```

`alsim` is a shell script which transforms `example0.conf` into a file that is suitable for `ALSIM` which reads from standard input and produces ASCII on standard output which is redirected to `alice.out`. Since simulation programs may run for a long time, so it is often a good idea to run these in the background.

Result analysis with PAW

You may inspect the output file directly (more detail see the Chapter: Analysis of result) or transform it into an input file suitable for either PAW or Excel.

Histogram can also be produced. For more detail, please refer to Chapter 7.“Histogram Package” on page 36.

Chapter 5. Modelling the ALICE DAQ system

This chapter is not complete. It will be updated regularly.

ALSIM -- A Hierarchical System

Benefiting from the modular feature of MODSIM, we can easily reuse some of the objects from the ATLAS simulation. As an objective-oriented language, MODSIM also allows us to build our system in a hierarchical way and the concept of inheritance is widely used.

Hierarchy of the objects in ALSIM

The inheritance of the objects in the MODSIM code bases on the MOD called BasicDAQObject. All the components in the system is derived from the BasicDAQObject. The EDM for example, is inherited directly from BasicDAQObject, while the LDCs and GDCs for example also have HardwareObjects, OSObjects built in.

Object “Message” that is used to pass information and data belongs to a class other than the system components and will be described later.

ALSIM hierarchy

The hierarchy of the whole system is shown in Figure 6.

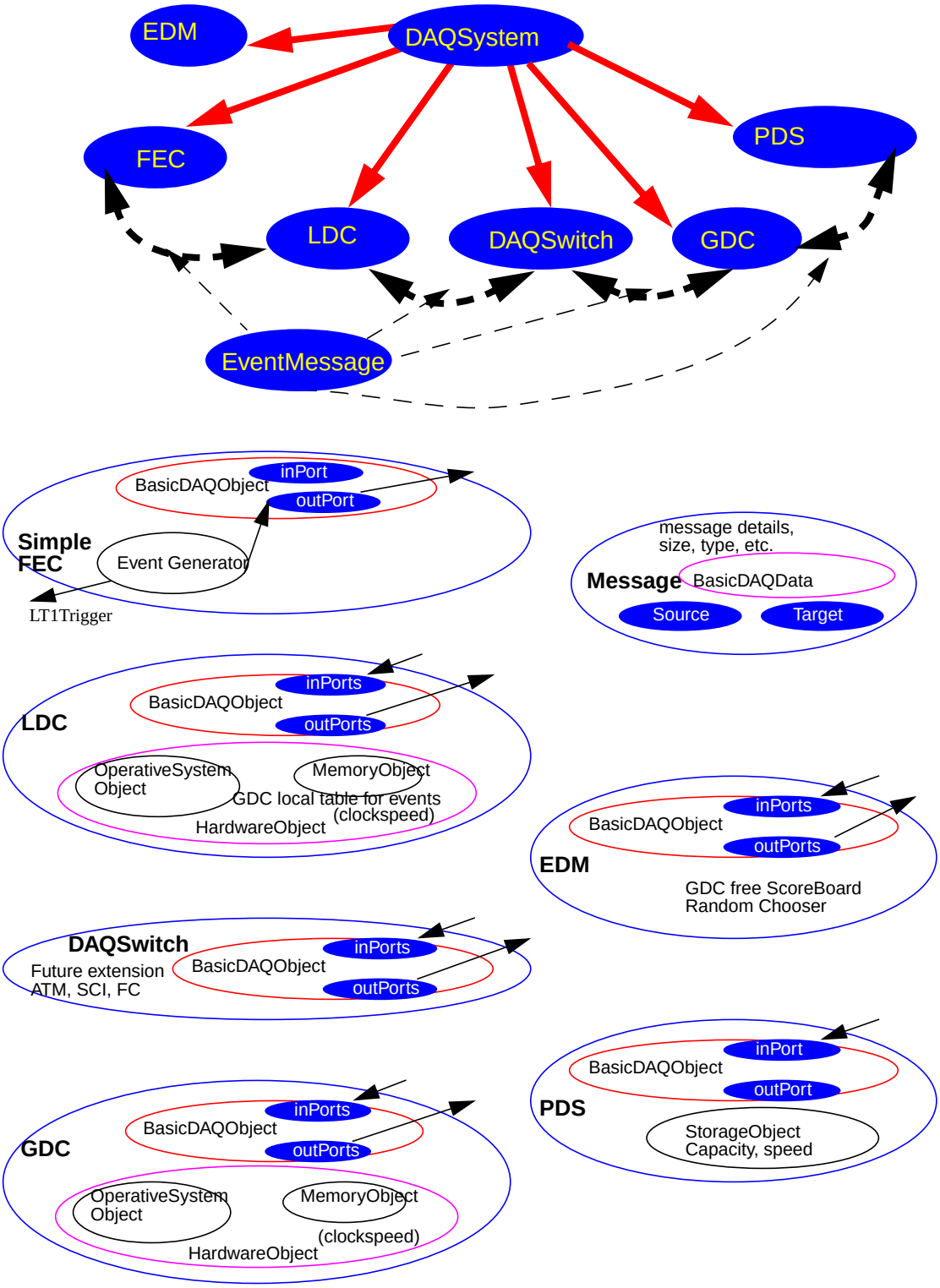


FIGURE 6. Hierarchy of ALSIM (not exhaustive)

Physics input to DAQ simulation - Event Generator (DG)

In the long run, the input to ALSIM should come from the physics simulation in a data file. However, due to the time constrains and shortage of man-power, and most of all, the study showing that the raw events do not differ very much from one to the other, we invented a pseudo Event Generator (also called Data Generator, DG). The messages (sub-events) size can either be fixed or normal distributed, while the interval time between the messages are fixed or exponential distributed.

The structure of DG

The structure of DG shown in Figure 7 has one I/O port that sends the L1Trigger information to the EDM, and the events are split into sub-events (messages) before they are pipelined in the Front-End Crates.

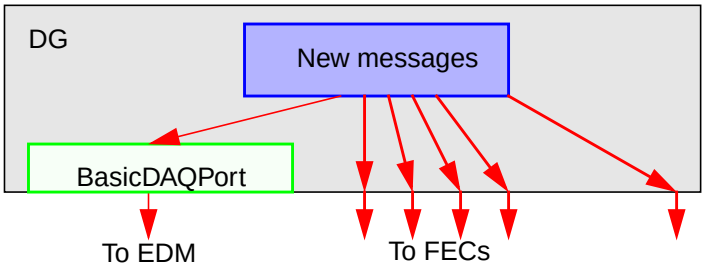


FIGURE 7. The internal structure of DG

Signals

Following is a list of all the signals that DG sends.

OUT:.

- Sends TPC sub event to the FEC (or extended FEC via MUX).
- Sends dimuon sub event to the FEC (or extended FEC via MUX).
- Send TPC trigger signal to EDM.
- Send dimuon trigger signal to EDM.

Parameters

All the parameters can be set in the configuration file. The way to define the DG is described in Chapter 6.“System Configuration As Input File and Preprocessor” on page 31 and Appendix B.“Configuration File Syntax” on page 58. Each of the ports connected to the object has its own parameters, clock and hardware delay.

- Hardware delay: delay for message when entering the object.
- TPC Message: message size, message interval, message size distribution, message interval distribution, shape of message size distribution curve and the interval the message size must be between.
- Dimuon Message: message size, message interval, message size distribution, message interval distribution, shape of message size distribution curve and the interval the message size must be between.

The Simple Front-End Crate

The FEC get sub-events from the Event Generator. It stores the events until it is read out by the LDC. The size of the buffer can be defined for each individual FEC in the configuration files. The buffer follows the rule of First-Come-First-Serve. When the buffer is full the FEC sends a busy signal to the Event Generator, which stops sending events and trigger signals.

The structure of the FEC

The block diagram of FEC is shown in Figure 8..

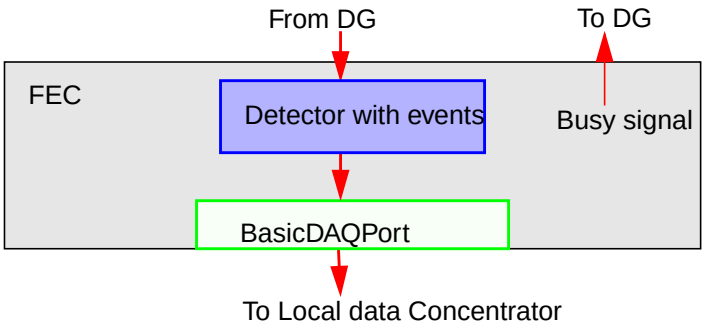


FIGURE 8. The internal structure of FEC

Signals

Below is a list of all the signals that FEC receives and sends.

IN:

- Receive sub-events from the DG.

OUT:.

- Send sub-events to the corresponding LDC.
- Send busy signal to the DG.

Parameters

All the parameters can be set in the configuration file. The way to define FEC is described in Chapter 6.“System Configuration As Input File and Preprocessor” on page 31 and Appendix B.“Configuration File Syntax” on page 58. Each of the ports connected to the object has its own parameters, clock and hardware delay.

- Hardware delay: delay of a message when it enters the object.
- Buffer size: the size of the buffer.

Multiplex Card - The Extended Front-End Crate, part I

After using ALSIM for a while, we found a need to extend our FEC functions to model a more complete and

complex real system. Therefore we introduce the extended FEC model that consists of two parts, the Multiplex Card (MUX) and the Zero Suppression Card (ZSC). The task of MUX is to receive sub-events from a number of sources and pass them on to a Zero Suppression Card(ZSC) over a fibre link. In this version of ALSIM the MUX receives one sub-event which it sends to the corresponding ZSC. It has only space for one sub-event in the buffer. Whenever an event arrives, it sends a busy signal to the DG, which will stop sending more events and trigger signals. When the sub-event is sent from the MUX, the busy signal is disabled.

The structure of MUX

The block diagram of a MUX is shown in Figure 8.

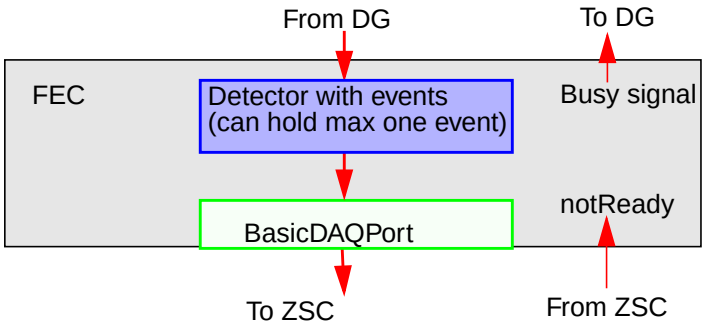


FIGURE 9. The internal structure of MUX

Signals

Below is a list of all the signals that MUX receives and sends.

IN:

- Receive sub-events from DG.

OUT:.

- Send sub-events to the corresponding ZSC.
- Send busy signals to DG.

Parameters

All the parameters can be set in the configuration file. The way to define MUX is described in Chapter 6.“System Configuration As Input File and Preprocessor” on page 31 and Appendix B.“Configuration File Syntax” on page 58. The port connected to the object has its own parameter, hardware delay.

- Hardware delay: delay for message when entering the object.

Zero Suppression Card - The Extended Front-End Crate, part II

The ZSC card receives events from the MUX. It zero suppresses the event, and stores it in the buffer. The size of the buffer can be defined for each individual ZSC in the configuration files. The buffer follows the rule of First-Come-First-Serve. When the buffer is full the ZSC sends a “notReady” signal to the MUX, which stops

sending events.

The structure of ZSC

A ZSC has the internal structure block diagram as Figure 8.

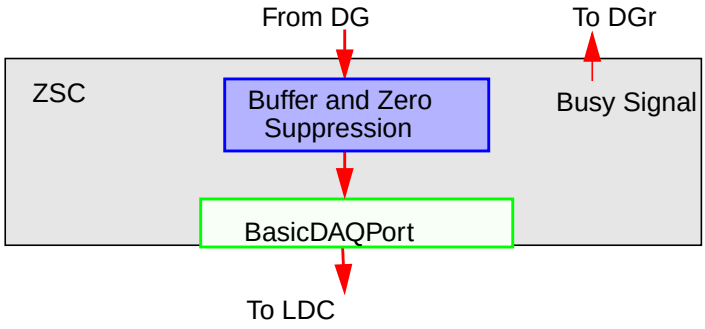


FIGURE 10. The internal structure of ZSC

Signals

Below is a list of all the signals that ZSC receives and sends.

IN:

- Receive sub-events from MUX.

OUT:.

- Send the sub-events to the LDC.
- Send “notReady” signal to MUX.

Parameters

All the parameters can be set in the configuration file. The way to define ZSC is described in Chapter 6.“System Configuration As Input File and Preprocessor” on page 31 and Appendix B.“Configuration File Syntax” on page 58. Each of the ports connected to the object has its own parameters, clock and hardware delay.

- Hardware delay: delay for a sub-event when it enters the object.
- Buffer size: the size of the buffer.
- Zero Suppression Rate: the factor that the event size is reduced by.

The Local Data Concentrator

The tasks of a LDC is to read sub-events from the corresponding FEC[1], and send them through the switch to the GDC chosen by the EDM. The information of chosen GDC is embedded in a special message from the EDM to the LDC for each event. Each LDC has a buffer in which it can store sub-events. The LDC stops reading the sub-events from the FEC if its buffer is full.

1. Here refers to both simple FEC and extended FEC.

The structure of LDC

In order to perform these operations, a LDC has a structure like the one shown in Figure 11. It consists of the main object LDC, which includes the objects HardwareObject, OSObject and LDCStoreProcess. It uses port to communicate with other objects. The figure show a system with Basic Topology (Definition and more details in “Advanced Switch System Setup - Extension I” on page 25.).

Apart from the ports that used to communicated between components in the system, the HardwareObject contains the clock speed, hardware delay, etc. facts before the message is forwarded to the operative system (OSObject).

The LDCStoreProcess has the task of assigning the right GDC to the incoming messages. So the LDCStoreProcess has a table containing the reserved GDC ids for each pending message. The content of the table is filled by the broadcasting message from EDM which contains an identity of a GDC. A CPU is also built-in, to perform these tasks though, it may not be necessary for these hardware-based tasks. However, baring in mind the possible future extensions, we keep the CPU in. A buffer (typically NumberOfGDCs * MessageSize) is required for several reasons, namely buffer the message for possible processing operation on the message, wait until the response message from EDM to come back, and in case of fluctuation due to burst-traffic, no message will be lost. If the buffer is full, the LDCStoreProcess stops reading messages from the FEC.

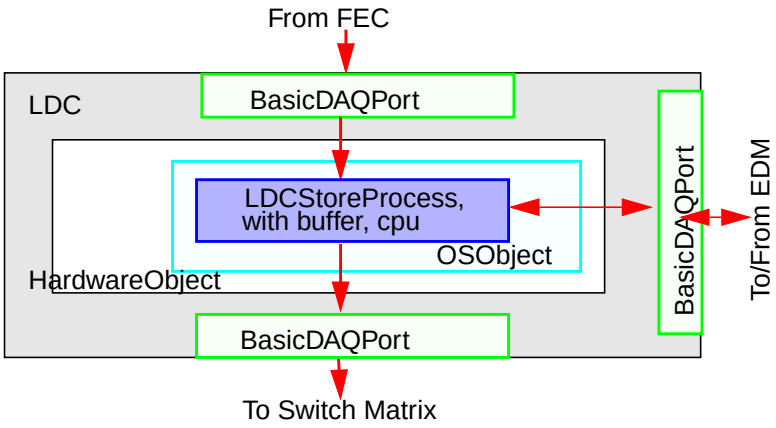


FIGURE 11. The internal structure of LDC

Signals

Following is a list of all the signals that LDC receives and sends.

IN:

- Receive trigger signal from the EDM.
- Receive information from the EDM on which GDC is chosen for the event.
- Receive sub-events from the corresponding FEC or ZSC.

OUT:

- Request for sub-events from the corresponding FEC or the ZSC.
- Send sub-events to the SWITCH.

Parameters

All the parameters can be set in the configuration file. The way to define the LDC is described in Chapter 6.“System Configuration As Input File and Preprocessor” on page 31 and Appendix B.“Configuration File Syntax” on page 58. Each of the ports connected to the object has its own parameters, clock and hardware delay.

- Hardware delay: delay for a message when it enters the object
- clock: speed of input links
- Process time (OSObject): time used by the operating system
- Buffer size (LDCStoreProcess): the size of the buffer.

The Switch - Event builder part I

The generic switch model shown in Figure 12 is a crossbar architecture. The number of inputs depends on numbers of LDCs while the number of outputs is related to the numbers of GDCs. Though often the case, it is not necessary that the number of the inputs is same as the number of the outputs.

The switch is completely passive. Whenever a message comes into any of the input ports, its target address (the target is a GDC) is checked and it is forwarded to the output port that corresponds the target GDC..

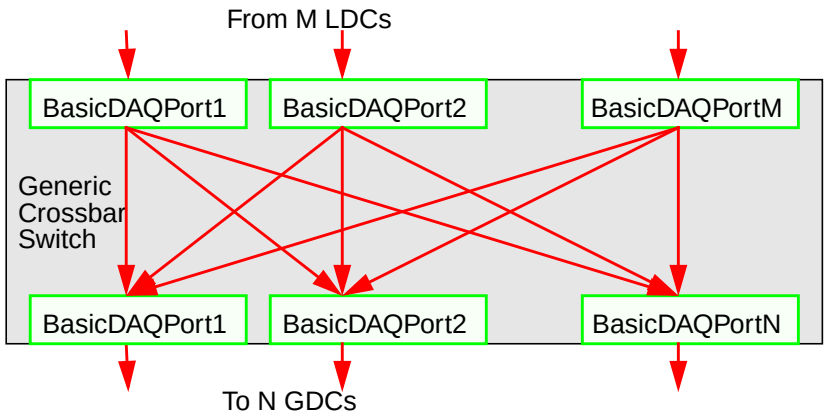


FIGURE 12. The internal structure of a generic switch

Signals

Following is a list of all the signals that the switch receives and sends.

IN:

- Receive sub-events from the corresponding LDC.

OUT:

- Send sub-events to the corresponding GDC.

Parameters

All the parameters can be set in the configuration file. Input rate (clock) and switch delay imply all the complexities inside the switch.

- delay: delay for a message to go through the switch.
- clock: speed of input link.

The Event Destination Manager - Event builder part II

The task of Event Destination Manager (EDM) is to send all the messages of the same event to the same and free GDC. The block diagram of the EDM is shown in Figure 13.

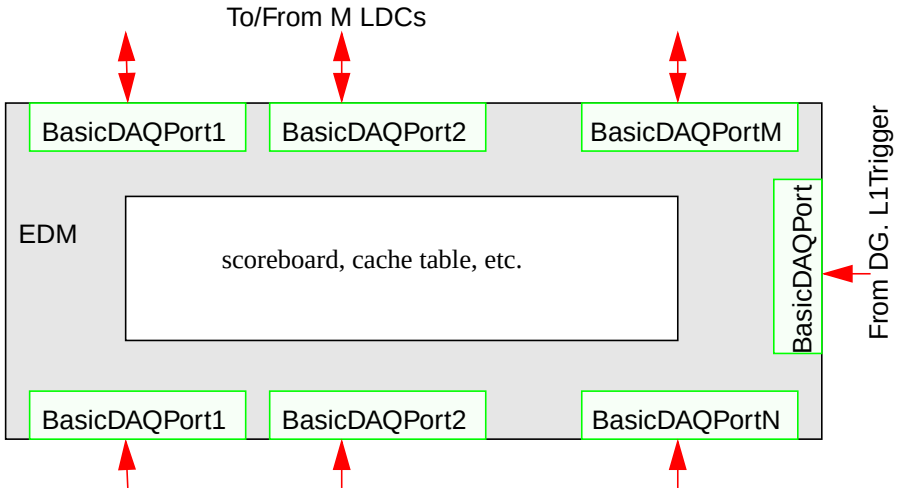


FIGURE 13. The internal structure of EDM

A scoreboard is used to record free GDCs. It is implemented as shown in Figure 14.a. An “AND” logic is used for fast checking whether there is any GDC free.

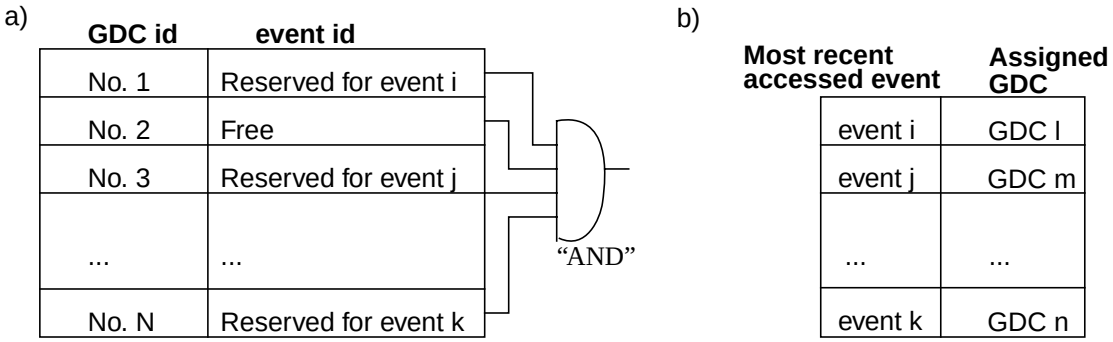


FIGURE 14. a) The scoreboard of EDM maintain the GDC status;
b) A small cache table used for fast searching the GDC id.

The algorithm of EDM is explained in the following items.

1. Once an event (number i) is generated, a small message (trigger) will be sent from the DG to the EDM to indicate this information;
2. The EDM checks its score-board (see Figure 15);

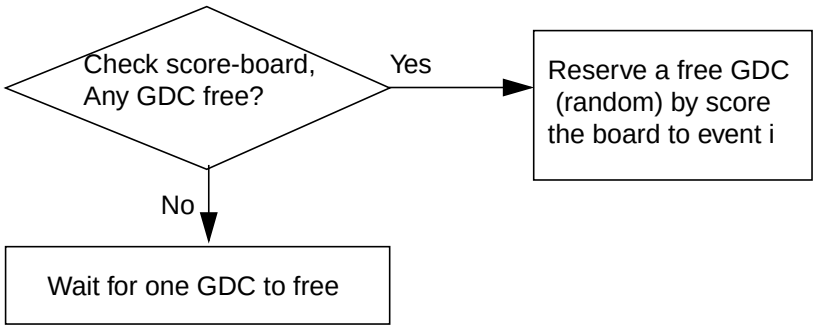


FIGURE 15. Flow diagram of the EDM when checking its scoreboard

3. Whenever the time for event i to be sent from LDC to GDC, LDC consults the EDM about the (supposed to be) reserved GDC id (This is a polling scheme rather than broadcasting);
4. If no GDC is reserved, the message will be piped in the buffer of LDC. The pipe has a threshold that alarms when the buffer becomes insufficient and it will back-press the DG from sending more triggers, until there is space again.

A small cache table is also implemented, because when a LDC ask the scoreboard about the GDC id, do an exhaustive searching of all the eventIds in the table that corresponding to the event in LDC is a time-consuming task. So a small cache will be used to hold the most recently returned eventIds from the EDM.

The above scheme use pulling instead of broadcasting (pushing). Why we say so? Because once the LDCs try to send anything, they ask EDM about the GDC address, thus we call it a pulling scheme. Its counterpart, on the other hand, is done by the EDM broadcasts a small message to all the LDCs, say that “if you have an event with number i to send, please follow the information in this broadcasting-message”. So the first two steps are exactly the same as pushing, and the rests steps are,

3. EDM broadcasts a message to all the LDCs, telling them the reserved GDC id for certain event (event i);
4. IF there is a sub-event (event i) already stored in the LDC buffer, sends it to the assigned GDC, ELSE saves the GDC id together with the event number i in a local memory;
5. Every time a EDM get a message from DG it triggers LDCs to read data from FECs, and IF there is a reserved GDC for this sub-event, sends the sub-event to the assigned GDC, ELSE saves the sub-event.

The advantage of broadcasting scheme is that the connection (traffic) between EDM and LDC is unidirectional, while the pulling scheme requires bidirectional links. The drawbacks are that all LDCs need to keep a table, and broadcasting implies all the LDCs have data from FECs which may not necessarily be the case.

The later scheme (broadcasting) is implemented in the current version of ALSIM.

The Global Data Collector

The GDCs receive messages form the switch, format them and record them onto a storage media.

The structure of the GDC

The internal structure of GDC is made in a way that we can put in the processes that a necessary to simulate a commercial processor. Figure 16 shows a basic version.

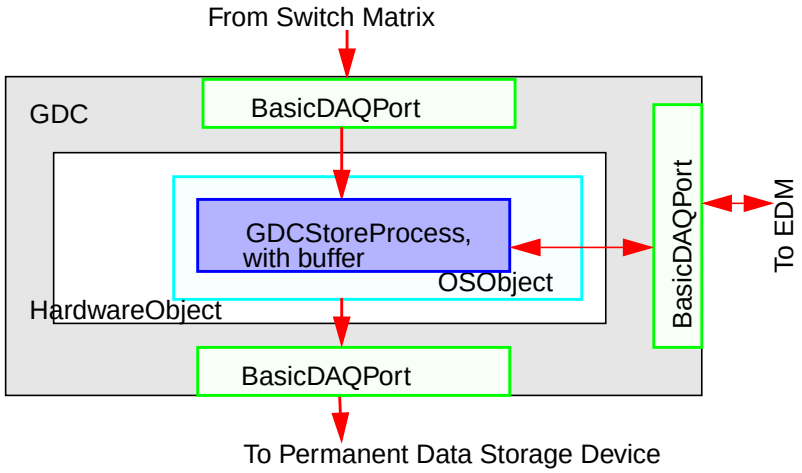


FIGURE 16. The internal structure of GDC

The main task of GDCStoreProcess is to collect all the messages that belong to the same event it is processing. It is therefore necessary to have a large buffer space, with a minimal size of the largest single event. In our simulator, the default is set to 50 Mbytes. Another important job of the GDCStoreProcess is to send a free signal to EDM to tell that it has finished with this event and the data is stored somewhere in PDS.

It is should be emphasized that after an event was generated and split into messages in DG, the messages (sub-events) with data are passed through FECs, LDCs and the switch. The content and size of each message are untouched except the target and source address. The messages are disposed in GDCStoreProcess while the whole event is reconstructed. The reconstructed events are then forwarded to PDS devices.

Signals

Below is a list of all the signals that GDC receive and sends.

IN:

- Receive sub-events from the SWITCH.

OUT:

- Send reconstructed events to the corresponding PDS.
- Send free signal to the EDM.

Parameters

All the parameters are set in the configuration file. Each of the ports connected to the object has its own param-

eters, clock and hardware delay.

- Hardware delay: delay for a message when it enters the object.
- clock: speed of input links
- Process time (OSObject): time used by the operating system
- Buffer size (GDCStoreProcess): the size of the buffer in each GDC.

The Permanent Data Storage

The Permanent Data Storage (PDS) will, as the name says, be used to store the event data. It receives the data from the corresponding GDC and stores it.

The structure of the PDS

The internal structure is shown in Figure 17.

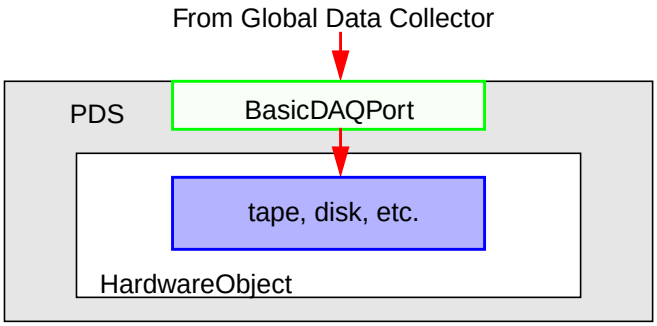


FIGURE 17. The internal structure of PDS

Signals

Below is a list of all the signals that PDS receives.

IN:

- Receive sub-events from the corresponding GDC.

Parameters

All the parameters are set in the configuration file. Each of the ports connected to the object has its own parameters, clock and hardware delay.

- Hardware delay: delay for a message when it enters the object.
- clock: input link clock.

The possible implementations of PDS are described in a later chapter.

Message Object

Message Objects are used to pass information, both in event transmission and event-building process. It could have various size, depends on its type.

Message format

All the messages simulated in ALSIM have following format as shown in Figure 18.

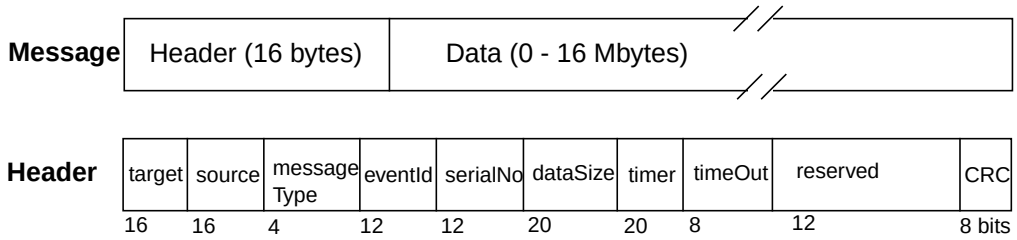


FIGURE 18. message format

target - the target address of the node in the system where the message is going to be sent to. We assume that each unit (such as FEC, LDC, GDC, EDM, PDS) in the system has an identify number, which is uniquely defined. There could be up to 64K nodes (16 bits) in a system, which we believe is adequate for the ALICE DAQ system being modelled.

source - the source address of the node in the system where the message is sent from (16 bits).

messageType - the type of message depends on where it sends from and where it sends to.The defined message type is illustrated in Figure 19.

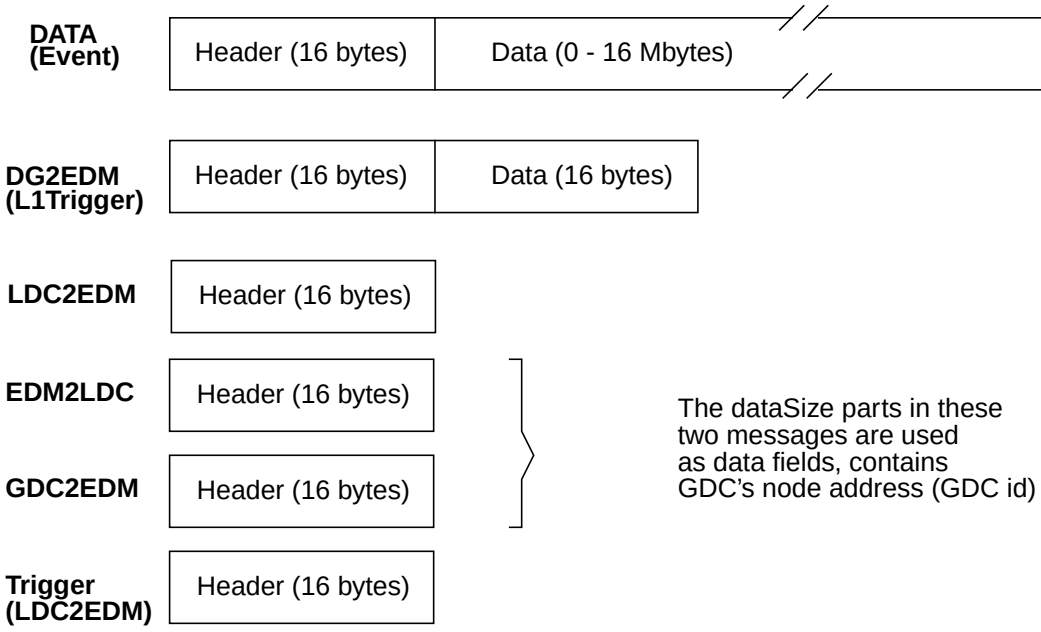


FIGURE 19. message types

eventId - the eventId of which this message belongs to. 4096 (12 bits) eventIds should be enough as it is not possible (except in a fault system) that the 1st event is still in the system while the 4097th event is generated.

serialNo - the serial number of this message. It is used for amalgamate process in the GDCs to be sure that all the messages belong to the event have arrived. An event should not be split into more than 4 K sub-events.

dataSize - the size of data part, from 0 - 1 M, representing 0 - 16 Mbytes where each unit representing 16 bytes.

timer - the 20-bit timer is used to note down the generate-time of the message. It is used for preventing a message is deadlocked somewhere in the system. We assume there are some units, for example FECs, switch elements, etc. have a clock runs at a same pace and they also have the logics of check whether the message is timeout (see next paragraph) or not, if so, the message is discarded.

timeOut - this 8-bit field is specified when the message is generated and is used to check whether a message stayed in the system too long.

IF (timeOut+timer > realClock) THEN discardThisMessage;

reserved - reserved for future use.

CRC - Cyclic Redundancy Check bits for error protection and correction (TBD).

Advanced Switch System Setup - Extension I

To cope with the different switch technologies and to simplify event building switching network, the ALSIM simulator support another advanced switch system setup. It is radically different from the one we have described before (typical example in Figure 4). The difference lies in the way the EDM communicates with the LDC and the GDC, compare Figure 4 and Figure 20.

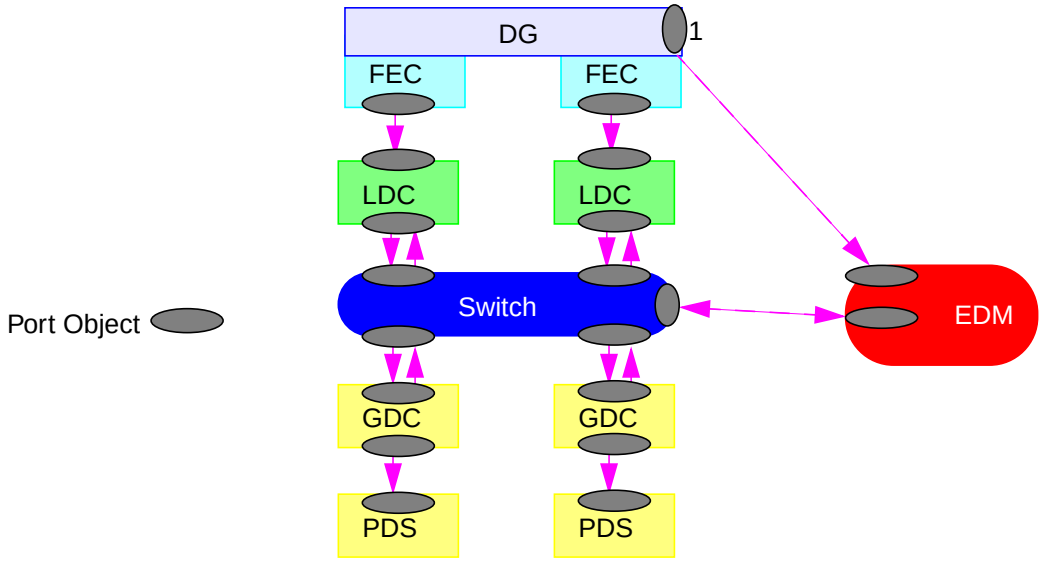


FIGURE 20. Advanced system setup for event building (Tech Topology)

The setup with physical connections between EDM/LDC, and between EDM/GDC, is called Basic Topology and the one we introduce here (via the switch) is called Tech Topology, which through the switch there will be

a combination of data and control information.

The topology type is defined in the configuration file. The presence of keyword *edm2switch* tells the simulator that you want to use Tech Topology, otherwise the default topology is Basic.

Extended FEC Setup - Extension II

We have already mentioned in the previous sections that we have two FEC setup modes, the simple FEC and the extended FEC. In this section we will explain and illustrate them in detail. A simple FEC shown in Figure 21.a has a direct connection to the DG and contains only one buffer for a sub-event. It's task is only to store the (only) sub-event, set busy signal, wait till the data is read by LDC and reset busy signal. The number of FEC and LDC are equal.

The extended FEC, is more complicated. It includes a number of Multiplex Cards (MUX) and Zero Suppression Cards (ZSC), as shown in Figure 21.b. The connection between MUX and ZSC is one-to-one while the connection between ZSC and LDC is many-to-one. The typical number of ZSC per LDC is eighteen.

The presence of keyword *extendedFEC* in the configuration file will allow you to use the extended FEC, and if it is omitted the default is simple mode.

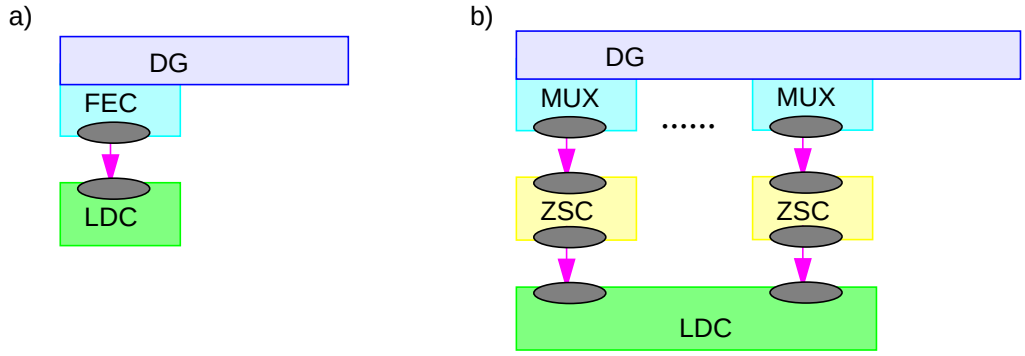


FIGURE 21. a) Simple Front End Crate and b) Extended Front End Create

The Complete Event Building Flow Diagrams

In this section, we are going to summarize the complete procedure of event building that is modelled in ALSIM. The flow diagrams of both the simple and the extended FEC are included.

Simple FEC (Figure 22 and Figure 24)

- The DG generates an event and trigger signal. The event is split into sub-events (equal to the number of FECs) and sent to the FECs. The trigger signal is sent to the EDM;
- The FECs receive the sub-events and store it in the buffer until the corresponding LDC reads it out. If the buffer is full it sends a busy signal to the DG, which stops generating more events and trigger signals until the buffer is free again;
- The EDM receives the trigger signal and distributes it to all the LDCs together with the free GDC for that event;
- Each LDC receives trigger signal and reads out the FEC if it has space in the buffer. If not it will wait until there is space available;
- Each LDC reads the sub-event from its corresponding FEC and stores it in the buffer;

- The sub-events are then pushed through the switch according to the network protocol;
- The GDC receives sub-events (that belong to the same event) from all the LDCs and builds a complete event, which is sent to the PDS;
- The PDS receives the complete event and stores it.

Extended FEC (Figure 23 and Figure 24)

- The DG generates an event and trigger signal. The event is split into sub-events (equal to the number of MUXs) and sent to the MUXs. The trigger signal is sent to the EDM;
- The MUXs receive the sub-events and send a busy signal to DG, which stops generating events and trigger signals. The event is sent to the ZSC if it is ready, and the busy signal is reset;
- The ZSC receives the sub-event from the corresponding MUX. It zero suppresses the sub-event and stores it in the buffer. If the buffer is full, it sends a “notReady” signal to the corresponding MUX, which will stop sending data to the ZSC. The ZSC send the sub-event to the LDC on the LDC’s request.
- The EDM receives the trigger signal from DG and distributes it to all the LDCs;
- The EDM finds the free GDC for that event and broadcasts to all the LDCs;
- Each LDC receives trigger signal and reads out the FEC if it has space in the buffer. If not it will wait until there is space available;
- Each LDC reads the sub-event from its corresponding FEC and stores it in the buffer;
- The sub-events are then pushed through the switch according to the network protocol;
- The GDC receives sub-events (that belong to the same event) from all the LDCs and builds a complete event, which is sent to the PDS;
- The PDS receives the complete event and stores it.

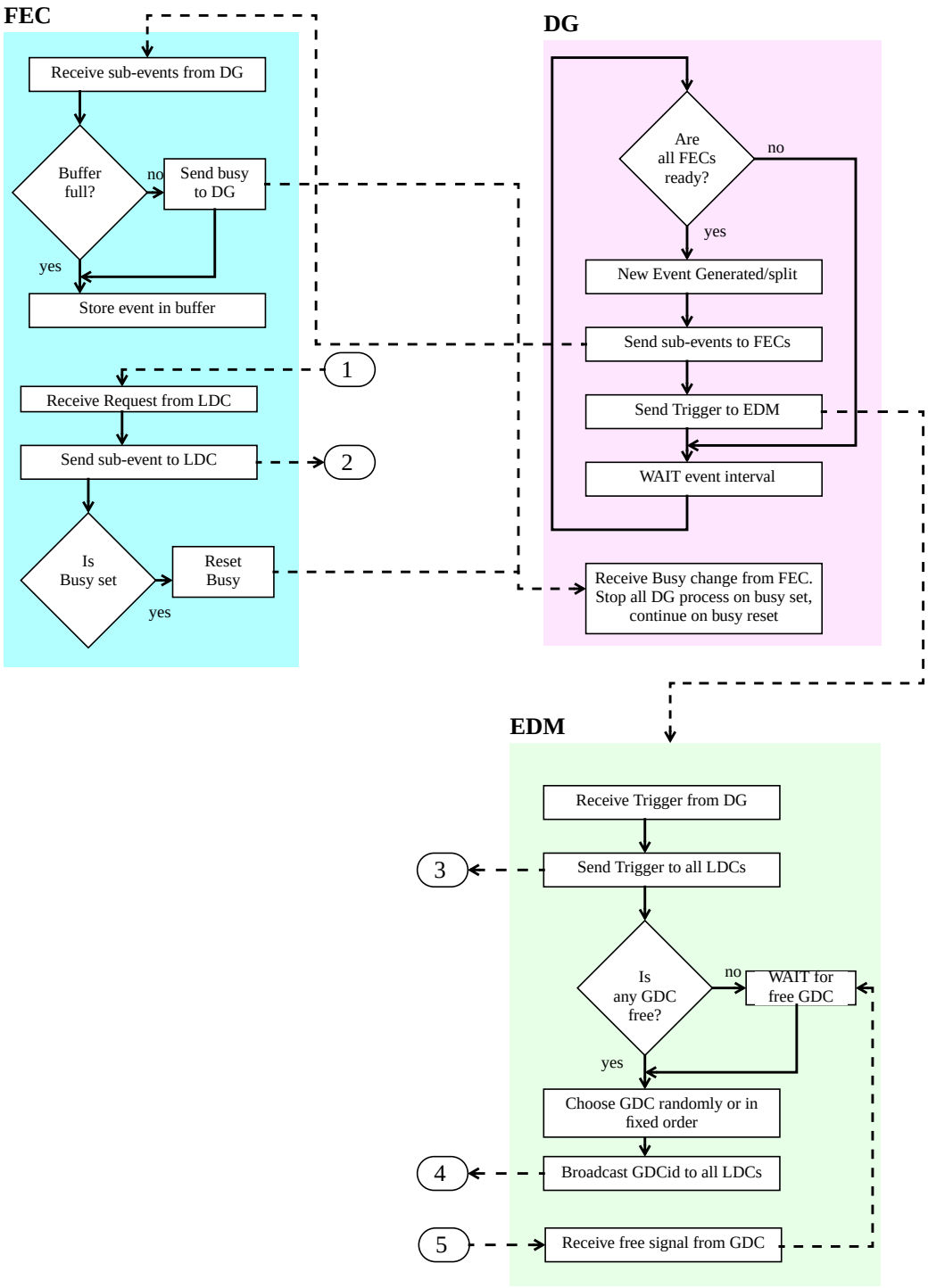


FIGURE 22. Event building procedure (Simple FEC)

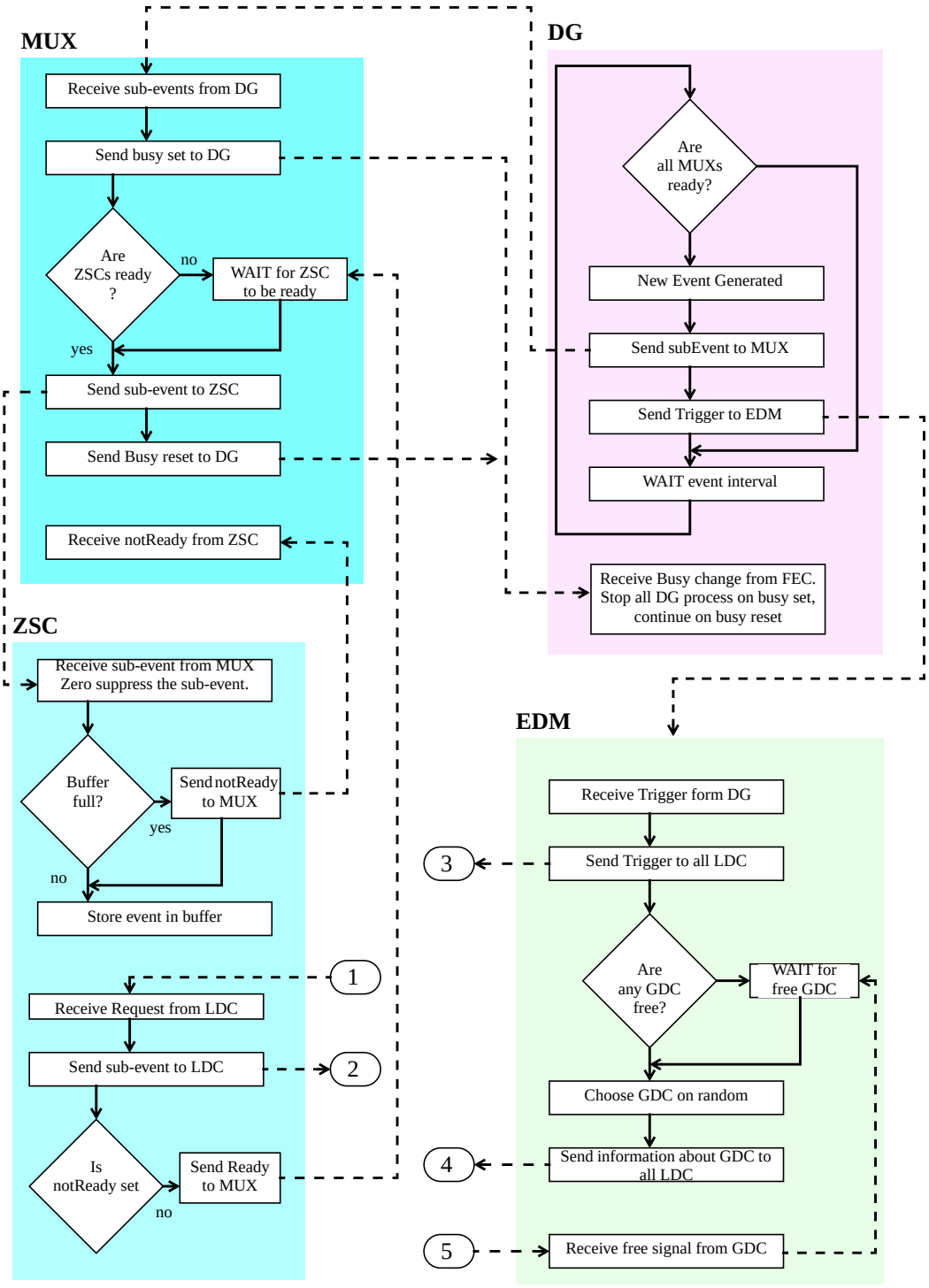


FIGURE 23. Event building procedure (Extended FEC)

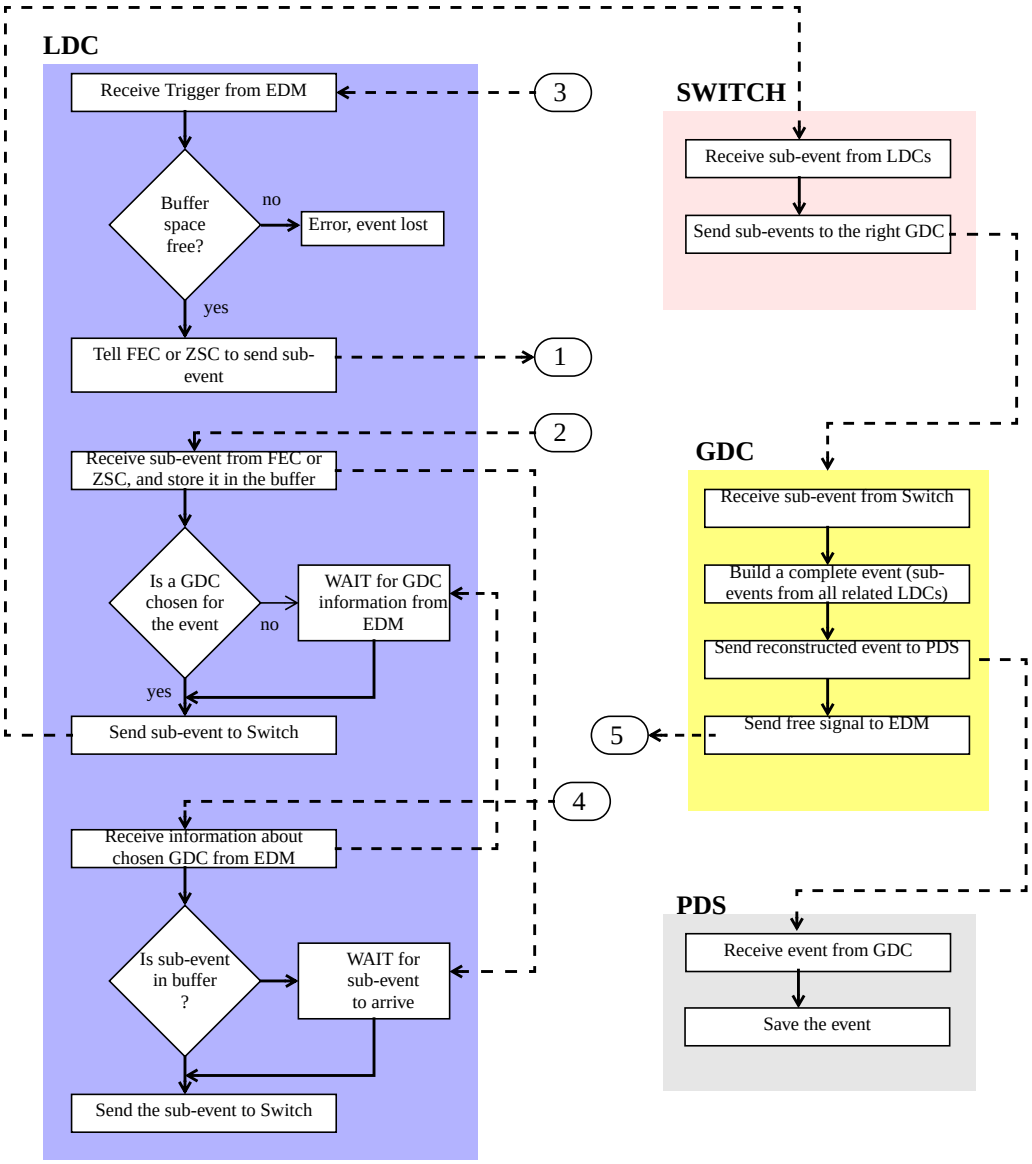


FIGURE 24. Event building procedure, cont. (Simple and Extended FEC)

Chapter 6. System Configuration As Input File and Preprocessor

Configuration of a simulation model

From the set of MODSIM objects the configuration of a particular model to be simulated can be built dynamically by means of an ASCII configuration file, like the "alice.conf" in Chapter "Getting Started". This chapter will go into very detail on how to write such a configuration file since this is the main work the users need to do and in the meanwhile explain how the ALSIM works.

The configuration file represents the configuration of the system you want to model and is served as the input file to `ALSIM` after preprocessing. We normally use ".conf" as the suffix of the names of the configuration files.

A configuration generator is being developed.

The general format of the configuration files

ALICE DAQ systems are defined as a collection of Parent Objects (FEC Objects, LDC Objects, etc.) with several ports (Child Objects), see Figure 25.

```
< Macro Definition > /* Comment has the same syntax as in C*/
<Simulation time, debuglevel, HistogrammingYesNo, etc. >
begin
    object ParentObjectType1 ParentObjectid /* f.ex. FEC */
      <... define parameters, such as clock, buffer size, etc.>
      object PortType1 PortId1 PortId2 PortType1 INOUT /*PortId1 is associated
          with PortId2 of the same type, INOUT says if the PortId1 is an
          input or output port */
      end object

      object PortType2 PortId3 PortId4 PortType2 INOUT end object

      <...other ports in the same ParentObject, other port could be of SCI port,
       ATM port, FC port type>
    end object /* ParentObjectType1 */

    object ParentObjectType2 <...>

    end object
    <...>
end
```

FIGURE 25. The configuration file format

The macro definition is for improving readability of the file. Simulation parameters such as simulation time, debuglevel, etc. can be specified in the file. Parent Object declaration always starts with keyword *object* and followed by the objectname, which is also a numeric value. Each Parent Object consists of several Child Objects, usually the input port and the output port. The technology dependent port could also be specified in addition to the general port object (and even coexist with each other) so that several technologies could be integrated into the same system. The syntax of the configuration files is presented in detail in Appendix

B.“Configuration File Syntax” on page 58.

Objects identities

Any of the objects has its unique identity for being recognized by the simulator when the configuration files are read in.

BASICDAQOBJECT	10001 /* ClassId of BasicDAQObject */
DAQSYSTEM	10003 /* ClassId of DAQSystem */
DAQSWITCH	10004 /* Generic Switch */
DAQBUFFER	10005 /* DAQ Buffer */
DGID	11000 /* Data Generator */
EDMID	11001 /* Event Destination Manager */
FECID	11002 /* Front End Crate */
LDCID	11003 /* Local Data Concentrator */
GDCID	11004 /* Global Data Collector */
PDSID	11005 /* Permanent Data Storage */
MUXID	11006 /* Multiplex Card */
ZSCID	11007 /* Zero Suppression Card */
TRIGGERBUFFER	12000 /* Trigger Buffer for OS */
OSOBJECT	12001 /* Operative System */
LDCSTOREPROCESS	12005 /* LDCStoreProcess */
GDCSTOREPROCESS	12006 /* GDCStoreProcess */
BASICDAQPORT	13001 /* Generic Port */
LOWESTPORTCLASS	13000 /* ALL the port.ClassId should be 13xxx */
HIGHESTPORTCLASS	13999 /* ALL the port.ClassId should be 13xxx */

Other pre-defined constants, words, etc.

The other constants that are pre-defined in ALSIM is listed below.

GenOUT	1
GenIN	2
GenBI	3

Keyword description

The purpose of keywords is to let the ALSIM easily read in the numbers that follow the keyword so that avoid unnecessary errors. This also makes the configuration file more readable. We list all the keywords here in alphabet sequence. Any unrecognized keywords will be simply omitted, without giving a warning by the ALSIM.

begin, beginning of a configuration file.

bufferSize, followed by one parameter, the size of the buffer in the object in which the definition is embraced in. The unit is byte. This definition is limited to frames of objects LDCStoreProcess and GDCStoreProcess. The default values are 10 Mbytes and 50 Mbytes for LDC, GDC respectively.

clock, followed by one parameter, the clock period of the hardware when the data path is 8-bit-wide. The unit ns. The default is 1 ns period for 8-bit-wide, implies 1000 Mbyte/s bandwidth. (Not fully supported in code!)

contentionMode, followed by a parameter that decides which method the free GDC is chosen by. Default 0 (when keyword is omitted) means random. If the number is 1, the free GDC with lowest id will be chosen. This keyword can only be used in EMC Object definition.

dataPortIn, is a keyword that follows the keyword “logicalPort”, to specify with port is used as the data (messages, events) input.

dataPortOut, is a keyword that follows the keyword “logicalPort”, to specify which port is used as the data (messages, events) output.

debuglevel, followed by a number specifying the level of debug (see Table 14).

Table 14: Debug

Debug Level	Functional description
0	Print statistics 1. at the print time, 2. in the end. Default.
1	Print simulation timer, ex, “simulation time is 500000 ns”, etc.
2	Print configuration in the beginning
4	Debugging objects FEC, LDC, GDC, PDS, Switch
6	Debugging Port objects
8	Debugging OSObject, LDCStoreProcess, GDCStoreProcess
10	Debugging CPU

delay, followed by one parameter, the hardware delay caused by the object in which the delay definition is embraced in. The unit is ns. The default values are, BasicDAQPort = 100 ns, DG = 100 ns, FEC = 100 ns, LDC = 100 ns, GDC = 100 ns, PDS = 1000 ns, DAQSwitch = 1000 ns.

dimuon, followed by two parameters, the first specifies the number of LDC which receives dimuon events. The second parameter specifies the number of GDC which receives dimuon events.

DIMUONmessage, followed by six parameters, parameter 1 is the message size generated in DG in bytes and parameter 2 is the interval between two successive events in ns. The default value is 1 Mbytes in average and 0.02 second in average. For message size, it is of exponential distribution, with boundaries of 1 Mbytes/10 and 1 Mbytes*10, if the parameter 3 equals to 0 (default). Otherwise it is of fixed size. The interarrival time is fixed, if the parameter 4 equals to 0 (default). Otherwise it is of exponential distribution. Parameter 5 specifies the standard deviation of the message size distribution. The default value is 5. Parameter 6 specifies the range for the message size generated. The range is given in percentage. For example, if range is 20, and message size is 1000 bytes, the generated message size can vary between 800 and 1200 bytes. The default value is 20.

edm2switch, this keyword indicates that communication between EDM and LDC and GDC goes through the switch, in Tech Topology.

end, the end of a configuration file.

end object, the end of the declaration of an object.

extended FEC, this keyword indicates that the extended FEC (with MUX and ZSC) is simulated.

histogramInterval, controls how often the histograms below write results to file. It is followed by a parameter that specifies the interval time (in nanoseconds) for writing intermediate output files. One should be careful not to put the value too short, otherwise one may end up with hundreds of files. If the keyword is omitted, only the final files will be written.

histogramOccupancy, enables writing output files for the graph display on how long time the buffer was occupied with a particular number of messages.

histogramRecord, enables writing output files for the graph display of buffer occupancy versus simulation time.

histogramLatency, enables writing output files for the graph display of event latency.

logicalPort, defines the paths for messages. It is followed by another pre-defined keyword, for example dataPortOut.

object, start of the declaration of an object.

pipeline, defines whether the message should be treated in virtual cut-through technique or store-and-forward technique. (TBD: supported in configuration file.)

processTime, followed by one parameter, the process time that is needed in the CPU. The unit is ns. This definition is limited to frames of objects LDCStoreProcess and GDCStoreProcess. The defaults values are 1000 ns 1000000 ns respectively for LDCStoreProcess and GDCStoreProcess.

simtime, the simulation time to run. It is followed by three parameters, parameter 1 is the simulation time, parameter 2 is the reset time which reset all the queues and statistics (in order to eliminate the fluctuation caused by simulation start) (TBD) and parameter 3 is the print statistics time (In addition to printing in the end)(TBD). The unit is ns.

token, this keyword indicates that a token sent between the LDCs. The LDC can only send data to the switch if it has a token.

TPCmessage, followed by six parameters, parameter 1 is the message size generated in DG in bytes and parameter 2 is the interval between two successive events in ns. The default value is 1 Mbytes in average and 0.02 second in average. For message size, it is of exponential distribution, with boundaries of 1 Mbytes/10 and 1 Mbytes*10, if the parameter 3 equals to 0 (default). Otherwise it is of fixed size. The interarrival time is fixed, if the parameter 4 equals to 0 (default). Otherwise it is of exponential distribution. Parameter 5 specifies the standard deviation of the message size distribution. The default value is 5. Parameter 6 specifies the range for the message size generated. The range is given in percentage. For example, if range is 20, and message size is 1000 bytes, the generated message size can vary between 800 and 1200 bytes. The default value is 20.

zeroSuppressionRate, followed by a parameter that gives the rate which the event size is reduced by. Example: If the event size is 10K, and zeroSuppressionRate is 10, then the event size will be 1K. This keyword can only be used in ZSC Object definition.

TBD

Preprocessor

With the help of a preprocessor, all the configuration files used as input files can be written in a very detailed and understandable way like the `alice.conf`. The program can be passed to a sh program called alsim we prepared for you to translate the program into the format that the ALSIM can accept. alsim uses the cpp, sed

and awk.

If you use this preprocessor to process the `alice.conf`, you will get `alice0.conf` as the output of the preprocessor. Type the following line,

```
% alsimp alice.conf > alice0.inp
```

The `alice0.inp` is an ASCII file that consists of only numeric values and keywords.

Comments are extensively used in programming. In configuration files the form of comments is consistent with the C programming language because the files will pass a C preprocessor (cpp, done in `alsim`) before it is send as input to `ALSIM`. The C macros such as `#define` and `#include` are also used.

The `alice0.inp` could be fed into ALSIM directly as following,

```
% ALSIM < alice0.inp > & alice.out &
```

Debug/Trace

Debug/Trace is possible in ALSIM. It helps both during the modelling work and when it is used. The debug is enabled by setting the debug level to a certain value, see “keywords” in this chapter. The high the debug level, the more information will be print when you run the simulation. The user should be aware that the size of the output file could well be over 10 Mbyte range within a few minutes (real clock) if the level is higher than, for instance, 2.

Chapter 7. Histogram Package

Introduction

The histogram package is a collection of routines which are used to provide histogramming of the ALSIM program. The package provides the opportunity to get a graphical view of some of the important parameters. The graphs can be presented either on-line or off-line (only off-line works in this version). The histogram package program was developed by Christian Hörtnagl for the ATLAS Trigger/DAQ Simulation program, and it has been transformed to work with ALSIM. There is a sub-directory called HISTOGRAM with all the source codes for the package. One needs to compile them and link the objective codes into ALSIM. In addition to that, one also needs a few lines in the configuration file "alice.conf" for enabling the package, and a setup file called "historc" for specifying the behaviour of the output produced.

A more detailed documentation of the original package can be found in [7].

How to control what is being histogramed

The histogram package can display four different histograms (only three are used in this version). The detail of each histogram is explained in "What is being histogramed" on page 37. User can turn on and off each of the histograms depending on the information one needs. Three keywords are used for controlling,

histogramOccupancy

histogramRecord

histogramLatency

If any of the keywords is found in the ".conf" file, that specific histogram is to be displayed. Users can also control how often the histogram files are produced by the keyword:

histogramInterval parameter(real)

The parameter after the keyword is the interval time (in nanoseconds) for writing intermediate output files. One should be careful not to put the values too short, otherwise one may end with hundreds of files. If the keyword is omitted, then only the final output files will be generated.

How to control the histogramming

The package uses a setup-file `historc` where the user can control the behaviour of the output produced. Its syntax is similar to what is used by conventions throughout the Unix-world.

"historc" consists of two types of statements. One is comments, which is, of course, only for readability and they always start with #. The other is commands that use a form

```
tag: value
```

Table 15 lists all currently supported tags and the semantics of values that are allowed to go with them. The following paragraph quotes form a sample setup-file:

```
# resource file to be used with Histogram Package for ALSIM simulations
backend: xgraph
wildcard: .*
```

```
mode: offline
path: results
timer_interval 10.0
record_interval 100.0
```

Table 15: Commands in file “historc”

tag	value	description
backend	asciiart (default)	Pseudo-graphical representations of histograms are printed to stdout when in on-line-mode, or to files when in offline-mode.
	gnuplot	gnuplot is selected at the preferred tool for online- or offline-mode.
	psgnuplot	Graphical-representation of histograms are stored in files using Post-script. Note that this value may only be used when in offline-mode.
	xgraph	xgraph is selected at the preferred tool for online- or offline-mode.
mode	on-line (default)	Switches to on-line-modes: Histograms are shown immediately, i.e. while the simulation program is still running. Note that on-line-mode and psgnuplot are incompatible options.
	offline	Switches to offline-modes: Histograms are stored in date-files in formats suitable for later offline-analysis and display, using a tool as indicated by the backend-option.
path		All data-files will be created in the specified directory.
wildcard		Calls to print will only have an effect for histograms whose labels match regular expression given at this tag’s value. See manual-pages for the Unix-command “ed” for a description of regular expressions.
timer_interval	timer_interval	See description of eventLatency graph below. The default value is 1.0.
record_interval	record_interval	See description of BufferRecord graph below. The default value is 1.0.

What is being histogramed

The output files produced from a simulation run is saved in the sub-directory `results`.

```
silicon% ls ./results/

GDC_0001_BufferOccupancy.final.xgraph    GDC_0001_BufferRecordBytes.final.xgraph
GDC_0002_BufferOccupancy.final.xgraph    GDC_0002_BufferRecordBytes.final.xgraph
LDC_0001_BufferOccupancy.final.xgraph    LDC_0001_BufferRecordBytes.final.xgraph
LDC_0002_BufferOccupancy.final.xgraph    LDC_0002_BufferRecordBytes.final.xgraph
GDC_0001_BufferRecordEvents.final.xgraph
GDC_0002_BufferRecordEvents.final.xgraph
LDC_0001_BufferRecordEvents.final.xgraph
LDC_0002_BufferRecordEvents.final.xgraph eventLatency.final.xgraph
```

The name convention of the output files is described as follows,

objectName_objectSerialNo_label.timeOfProduction.extention

As an example:

GDC_0002_BufferOccupancy.final.xgraph

GDC: is the location of the buffer.

0002: is the object's serial number.

BufferOccupancy: is a label that describes the type of graph.

final: tells when the graph was produced.

xgraph: tells which graphical program it is produced for.

One can use `xgraph` to view the files, as the examples shown in Figure 26, Figure 27 and Figure 28. The first two graphs deal with buffers and the last one is used to monitor the event latency.

Time spent with occupancy

The histogram "histogramOccupancy" measure for how long time a buffer is filled with a certain amount of data. Figure 26 shows how long time the buffer was occupied with a particular number of bytes. The x-axis shows how much of the buffer that is occupied (in KBytes), and the y-axis shows the time.

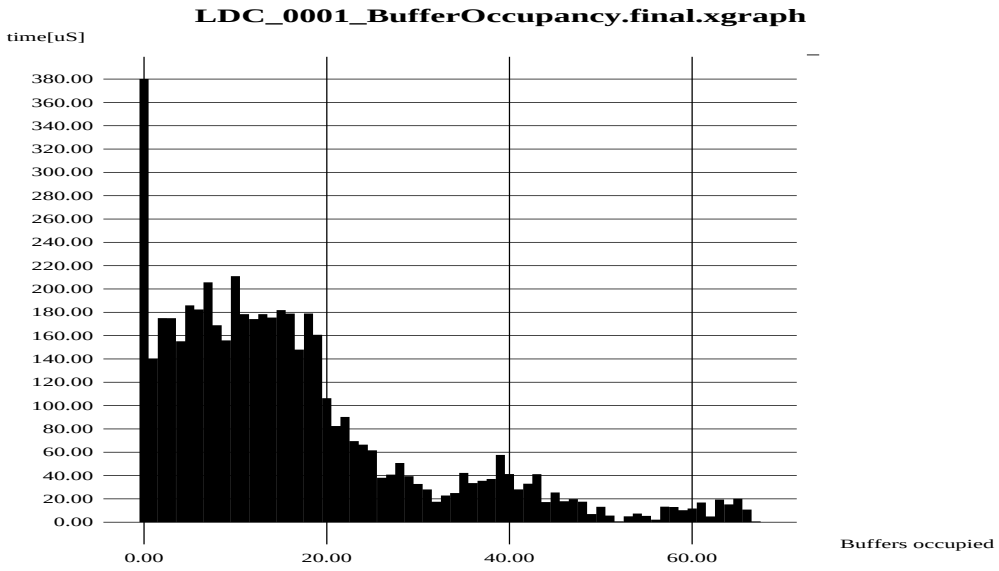


FIGURE 26. Time spent with occupancy

The graph was produced by UNIX command:

```
% xgraph -bar -brw 1 -nl -t "LDC_0001_BufferOccupancy.final.xgraph" -x "Buffers
occupied" -y "time[uS]" LDC_0001_BufferOccupancy.final.xgraph
```

Record buffer occupancy by time

The histogram “histogramRecord” show how much off the buffers that are occupied as a function of time. Two graphs are produced, one show occupancy in bytes and the other in events. Figure 27 shows how the buffers are occupied with bytes as a function of time. The resolution of the x-axis is determined by the variable “record_interval” in the “historc” file. If the value is 100.0, only one value in an interval of 100 us will be registered, and that will be the largest value in the interval. The x-axis shows the simulation time, and the y-axis shows the buffers occupation in Kbytes.

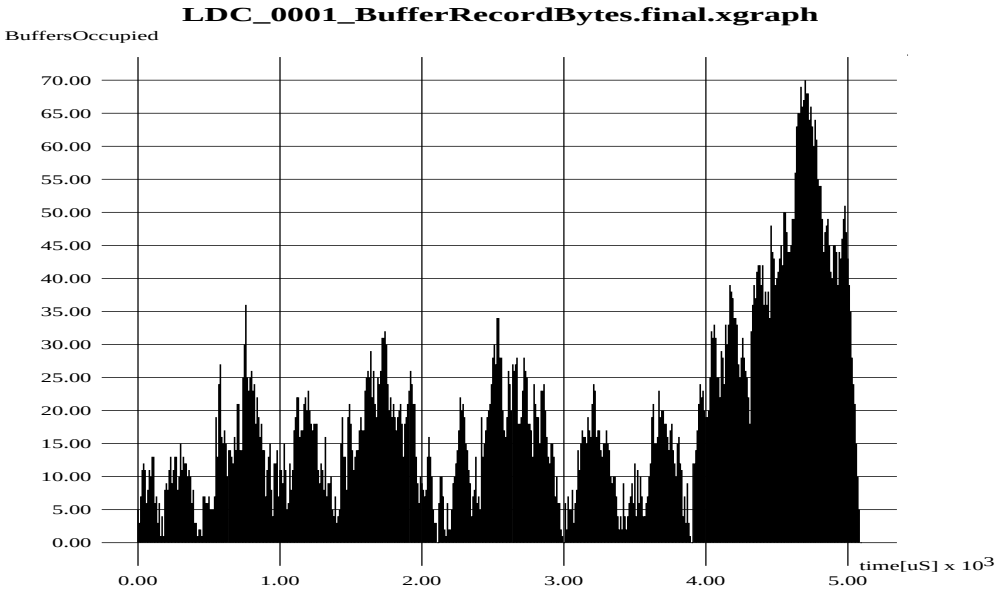


FIGURE 27. Record buffer occupancy by time

The graph was produced by UNIX command:

```
% xgraph -bar -brw 1 -nl -t “LDC_0001_BufferRecordBytes.final.xgraph” -x
“time[uS]” -y “BuffersOccupied” LDC_0001_BufferRecordBytes.final.xgraph
```

Event Latency

The graph produced by “histogramLatency” is shown in Figure 28. It shows the time it takes for events to go through the whole system. The x-axis shows latency time, and the y-axis shows the number of messages that had that long latency. The accuracy of the x-axis is determined by the variable timer_interval in the “historc” file. If the number is 10.0 then all the messages in between an interval of 10 us will be counted as same time, i.e. all messages with latency between 15 us to 25 us will give a raise to the x-value 20 us, as shown in the

graph below.

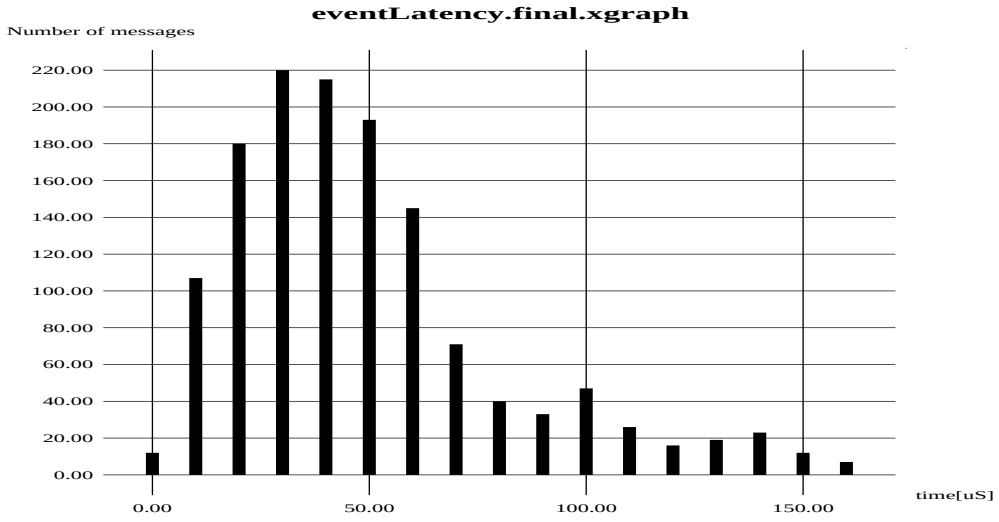


FIGURE 28. Event latency versus time

The graph was produced by UNIX command:

```
% xgraph -bar -brw 3 -nl -t “eventLatency.final.xgraph” -x “time[uS]” -y “Number
of messages” eventLatency.final.xgraph
```

Buffer Hit

This is not a real graph, but only a point. It tells how many messages that was received by a buffer. Not used yet.

Chapter 8. Result Analysis

The results of simulation are in ASCII format. They could be inspected directly or saved in a file and processed later by PAW or any other tools. An example of output ASCII file is listed in Figure 29 with a run of the system in Figure 5.

```
ALSIM version 0.08 starting.....
ALSIM input file read.....
--- ALSIM started at:  Thu Jun  8 11:18:23 1995
**************************************************************************
*            Statistics of the Whole ALSIM System
**************************************************************************
  Parent:     Id:   Port:          Id:   RemotePort:    Id:    InBW(MB/s):    OutBW(MB/s):
+EventGen     1     BasicDAQPort   1     BasicDAQPort   2      00000000000    1.449
+EDM          1     BasicDAQPort   2     BasicDAQPort   1      1.449          00000000000
+EDM          1     BasicDAQPort   3     BasicDAQPort   11     00000000000    1.449
+EDM          1     BasicDAQPort   4     BasicDAQPort   14     00000000000    1.449
+EDM          1     BasicDAQPort   5     BasicDAQPort   21     0.346          00000000000
+EDM          1     BasicDAQPort   6     BasicDAQPort   24     0.378          00000000000
+FEC          1     BasicDAQPort   7     BasicDAQPort   9      00000000000    53.758
+FEC          2     BasicDAQPort   8     BasicDAQPort   12     00000000000    46.549
+LDC          1     BasicDAQPort   9     BasicDAQPort   7      53.758         00000000000
+LDC          1     BasicDAQPort   10    BasicDAQPort   15     00000000000    53.758
+LDC          1     BasicDAQPort   11    BasicDAQPort   3      1.449          00000000000
+LDC          2     BasicDAQPort   12    BasicDAQPort   8      46.549         00000000000
+LDC          2     BasicDAQPort   13    BasicDAQPort   16     00000000000    46.549
+LDC          2     BasicDAQPort   14    BasicDAQPort   4      1.449          00000000000
+DAQSwitch    1     BasicDAQPort   15    BasicDAQPort   10     53.758         00000000000
+DAQSwitch    1     BasicDAQPort   16    BasicDAQPort   13     46.549         00000000000
+DAQSwitch    1     BasicDAQPort   17    BasicDAQPort   19     00000000000    50.590
+DAQSwitch    1     BasicDAQPort   18    BasicDAQPort   22     00000000000    49.718
+GDC          1     BasicDAQPort   19    BasicDAQPort   17     50.590         00000000000
+GDC          1     BasicDAQPort   20    BasicDAQPort   25     00000000000    50.243
+GDC          1     BasicDAQPort   21    BasicDAQPort   5      00000000000    0.346
+GDC          2     BasicDAQPort   22    BasicDAQPort   18     49.718         00000000000
+GDC          2     BasicDAQPort   23    BasicDAQPort   26     00000000000    49.339
+GDC          2     BasicDAQPort   24    BasicDAQPort   6      00000000000    0.378
+PDS          1     BasicDAQPort   25    BasicDAQPort   20     50.243         00000000000
+PDS          2     BasicDAQPort   26    BasicDAQPort   23     49.339         00000000000

The average latency of 249 messages from DG to EDM is 174 ns.
The average latency of 498 messages from EDM to LDC is 242 ns.
The average latency of 249 messages from GDC to EDM is 52 ns.
The average latency of 498 messages from DG to GDC is 9638 ns.
The average latency of 249 messages from GDC to DS is 8699 ns.
The average latency of 249 events is 18337 ns.

Exiting: SimTime   = 5500.000000
Exiting: ResetTime = 0.000000 simulation time units
Exiting: StopTime  = 5000000.000000 nanoseconds
--- ALSIM Normal termination at: Thu Jun  8 11:18:35 1995
Run took 12 seconds.
Adios amigo !
```

FIGURE 29. alice.out

Chapter 9. ALSIM Graphics Presenter

In order to give an overview of what happens during a simulation, we implemented a window-based graphics presenter. We are not aiming at seeing packet/messages flowing around, but see the traffic flow on the links, and the distribution of packet/messages in the whole system. The displaying system should be dynamic. However, the update of the display should not take too much CPU resource.

System setup for running MODSIM graphics

In order to run MODSIM graphics, following path must be set, (we mentioned this before)

```
% setenv LD_LIBRARY_PATH /usr/openwin/lib
```

Current graphics presenter

Figure 30 is a window captured during the simulation of alice.conf with a graphics presenter.

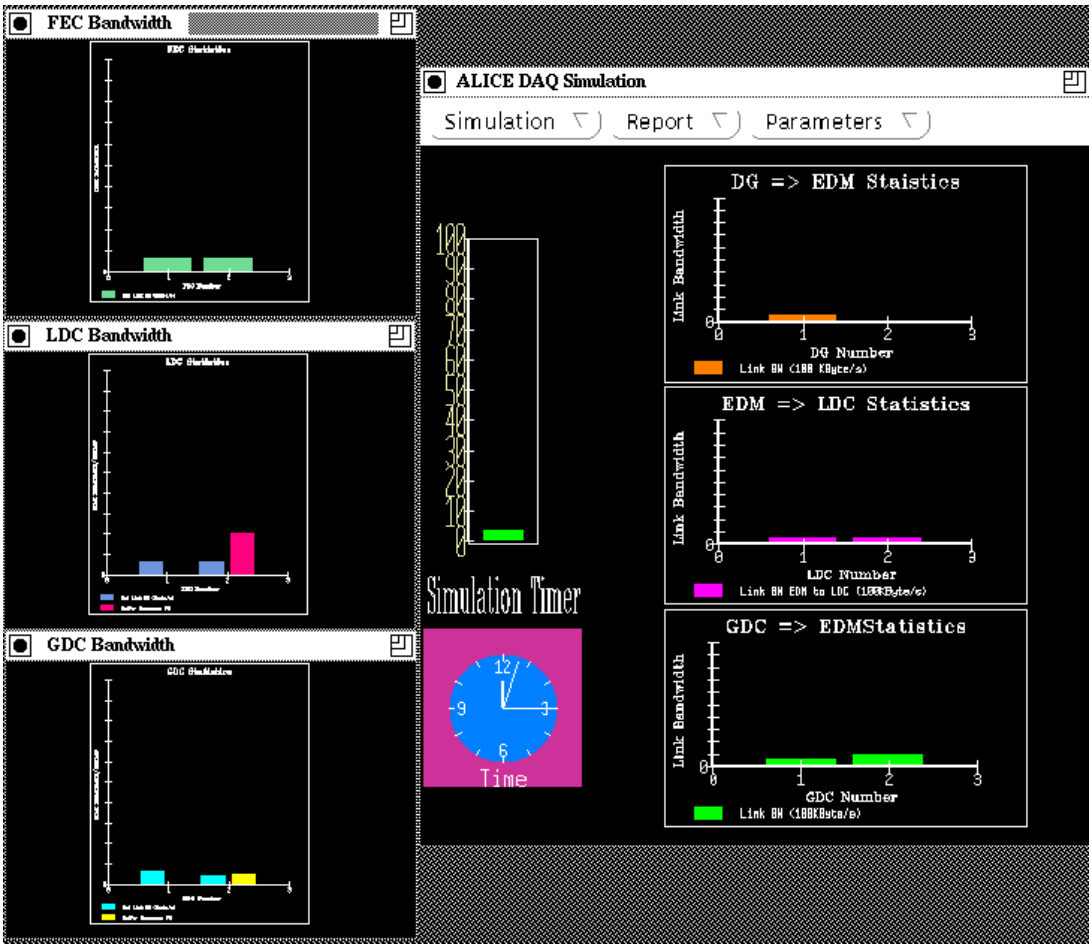


FIGURE 30. A window captured during the simulation of alice.conf

Graphics presenter - future extension

A future graphics presenter would display each object in detail and the messages would also be shown so that we could see messages flowing in the system. The purposes are to present our work better, to understand better on how the system works, specially the hot spots could be easily detected. It could be like in Figure 31.

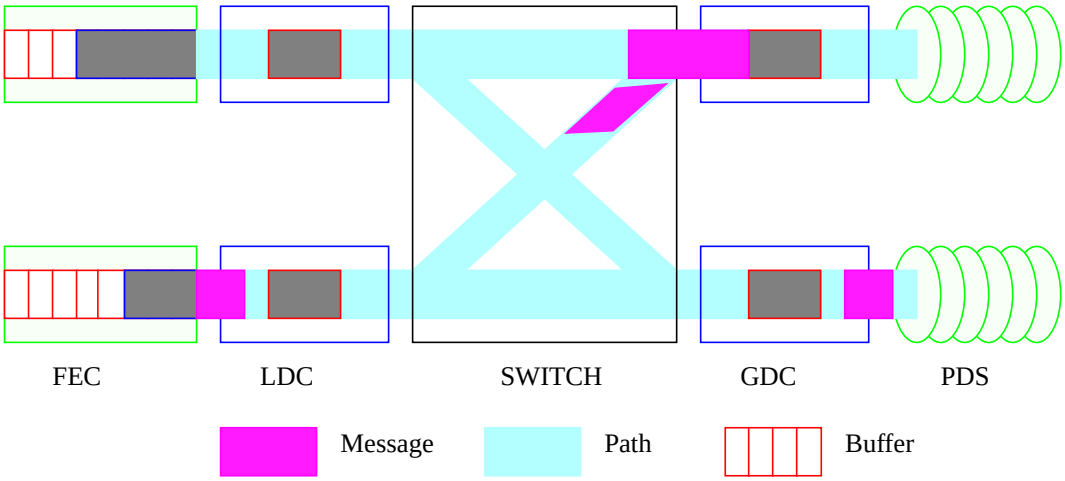


FIGURE 31. Graphics presenter - future extension

Chapter 10. Using SCI Technology in Switching Network

SCILab introduction

SCILab is a simulation environment for the Scalable Coherent Interface and large SCI based systems [6]. SCILab simulates the behaviour of SCI nodes (interfaces, processors, memories), networks (connections, switches), and applications (models of processors, cache coherency). It handles contention (by simulating SCI protocols for packet transmission, bandwidth allocation and queue acceptance) and calculate latencies which are incurred when packets are transmitted over physical media, queued in interfaces, transferred to memories and CPUs and finally processed by application hardware and software. To make it run fast, the simulation is driven by a chain of “discrete events” which typically occur when packets arrive or leave an SCI node. It allows simulating almost any topology. Many parameters can be modified. SCILab has been used for more complex systems of a few thousand nodes.

SCILab consists of TopoEngine and SCIMP. The TopoEngine can generate and route many popular, regular SCI topologies of arbitrary large size. The SCIMP is the SCI simulator. The object-oriented technique has resulted in very modular and flexible code. SCI modules could easily be integrated in the simulation of large and complex data acquisition systems for experiments at CERN’s Large Hadron Collider.

Popular networks such as meshes and multistage Banyan networks have been carefully and extensively simulated as well as some irregular topologies for data acquisition systems. As SCI is gaining more acceptance in the computer industry, more systems will be based on it. Simulation is a valuable tool for the design and it has helped the first vendor of SCI in optimizing their hardware.

Integrating SCILAB to ALSIM simulation environment

The general ALSIM event builder is performed by a generic switch between LDCs and GDCs. The objective of ALICE DAQ simulations with SCI is to replace this generic switches with SCI switching systems, as shown

in Figure 32.a. The SCI-based switching networks could be a single SCI ring (Figure 32.b) or complex

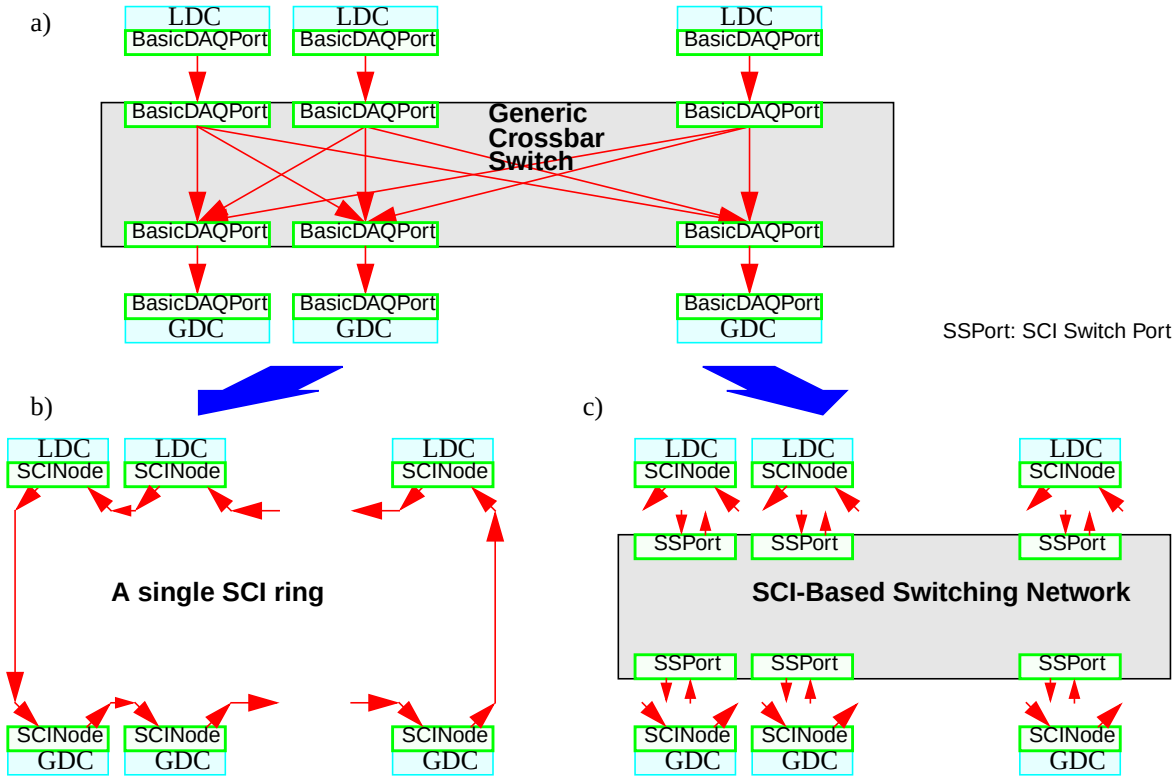


FIGURE 32. From generic switch to SCI-based switching systems

network topologies (Figure 32.c) such as meshes and multistage Banyan networks, etc.

The ring structure is simple. In order to form an event builder, one simply needs to connect the output link of one SCI node (which is part of either a LDC or a GDC) to another node's input link. After all the nodes are connected, the system setup is finished. The way of connection can influence the system performance, due to the flow of data is from LDCs to GDCs. If we do as in Figure 32.b, then the rightest LDC node will be the bottleneck, due to all the packets from LDCs will pass that node before reaching their destination GDCs. An alternative way of connection with load balancing is shown in Figure 33.

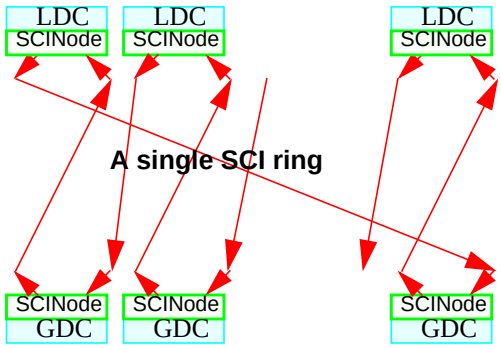


FIGURE 33. A single SCI ring with LDCs and GDCs interleaved

How does it work, an overview

The task is to integrate the two domains' software together (Figure 34). The generic and SCI domains are linked through interfaces called SCI *Ports*. Generic messages are disassembled into SCI packets, transmitted through the network according to SCI protocols, and reassembled upon arrival.

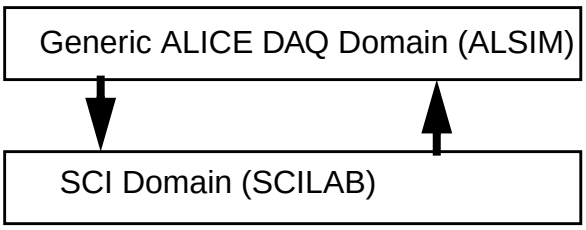


FIGURE 34. Integrating two simulation environments

Description of SCI specific objects

Object SCIPort

The SCIPort is a composite object with a pointer field to a DAQSCINode object. An SCIPort object can either be an IN port or an OUT port, which is unidirectional. SCIPorts have to send or receive messages through its DAQSCINode. For example, if a LDC will send a message to a GDC, and it has an SCIPort as the OUT port, the LDC will invoke the WAITFOR METHOD SendMessage of the SCIPort. In turn, the METHOD SendMessage of SCIPort will invoke the WAITFOR METHOD SendMessage of its DAQSCINode.

SCI supports at most 256-byte long packets. The size of a message is normally much larger. So SCIPort must calculate the number of packets it delivers to a DAQSCINode. In our implementation, we use 64 bytes as the default data size for each packet passed to DAQSCINode. The incoming message is cut into SCI packets in SCIPort and send over SCI networks.

The output queues may be more than one-packet deep in a DAQSCINode, SCIPort tries to saturate them. It is fully described in the MODSIM II code (in the WAITFOR METHOD SendAllPacket of SCIPort).

Object DAQSCINode

SCINodes are bidirectional, therefore they can be linked to one IN port or one OUT port, or both, depending on the application. DAQSCINode can send and receive packets at the same time. It is derived from Object SCInode in Module SCItrans. ASK METHOD SetPorts is new. It takes care of the threads of its IN and OUT

ports. In Figure 35, the links between DAQSCINode and SCIPort are shown.

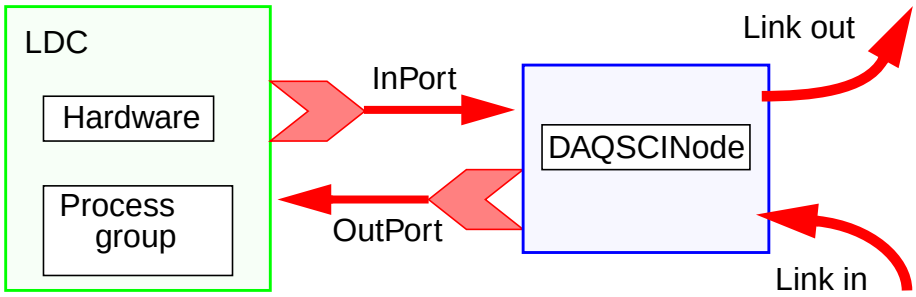


FIGURE 35. A LDC hooked to an SCI node

How to use/configure the system

Changes in configuration file

In alice.conf, following changes must be made to adapt with SCI, the left column is the old file while the right should be in the new one.

Original alice.conf	**New Alice.conf**
	#define SCIDAQPort 13002
	#define SCIDAQSystem 20004
	#define SCIDAQSwitch 20004
#define Switch DAQSwitch	#define Switch SCIDAQSwitch
#define LDCToSPort BasicDAQPort	#define LDCToSPort SCIDAQPort
#define SToLDCPort BasicDAQPort	##define SToLDCPort SCIDAQPort
#define SToGDCPort BasicDAQPort	#define SToGDCPort SCIDAQPort
#define GDCToSPort BasicDAQPort	#define GDCToSPort SCIDAQPort
	#include “SCI.conf”

Example SCI topology

SCI switching networks have an internal structure of smaller switches. The topology of these networks can be altered, but must be specified in a configuration file. An example of an SCI network consisting of a hybrid of SCI rings and a $8_R x 8_R$ SCI switching network interconnecting 32 LDCs and 31 GDCs, 1 EDM, is shown in Figure 36.Placing the EDM at the GDC side of the switch is just one of the many alternatives. With possible implementation of fast switch in industry, the system could be optimized to a cheaper and simpler switching

network as shown in Figure 37.

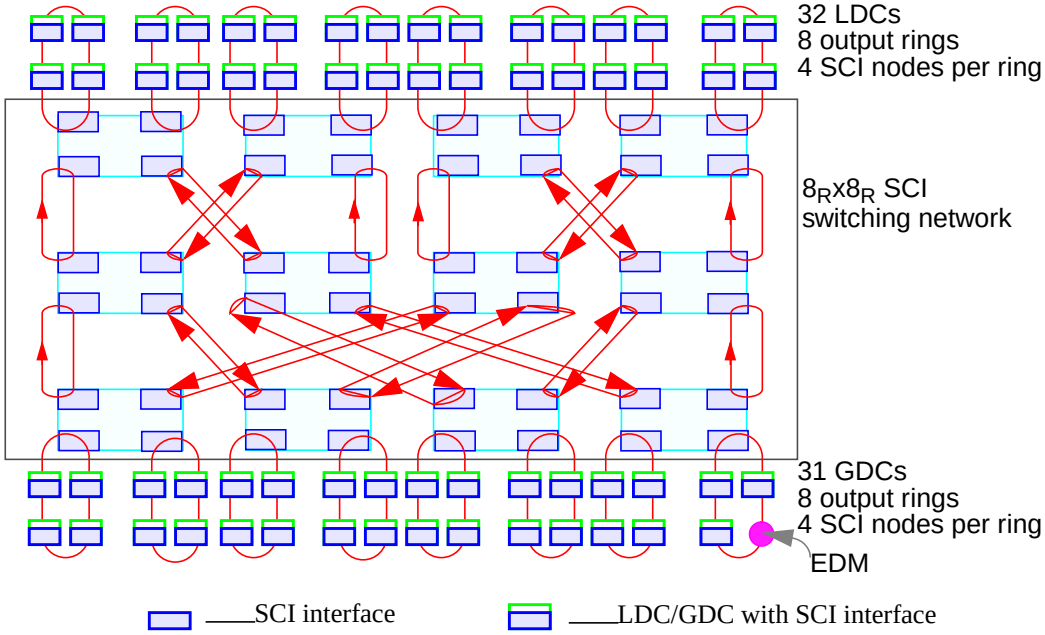


FIGURE 36. SCI $8_R x 8_R$ switching network

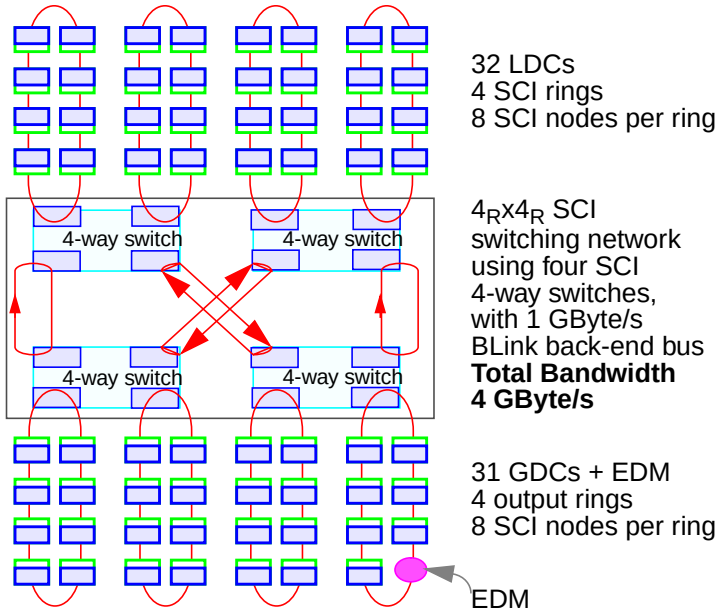


FIGURE 37. SCI $4_R x 4_R$ switching network

Simulation results and analysis

TBD

Chapter 11. Using ATM Technology in Switching Network

The aim of ATM simulations in ALSIM is to replace the generic switches with ATM switches and to study the performance of the ATM switches.

ATM model introduction

The ATM code we use in our simulation is developed by Denis Calvet in CEN Saclay, France. Different models of ATM switches are available. These are:

- ATM Alcatel 155 Mbit/s
- ATM Alcatel 622 Mbit/s
- ATM Fore 155 Mbit/s
- AT&T ATM switch fabric, 622 Mbit/s (Phoenix)

These four models can be grouped into two flavours: simplified models, like ATM Alcatel 155 and 622 switches, ATM Fore switch and detailed models, like AT&T ATM Fabric.

For all the models, modulation of traffic is done at the ATM cell level. Each ATM is of 53 bytes long, with 5 bytes header and 48 bytes payload. Cells are assembled together to form a packet. The ATM adaptation layer is AAL5, hence simulation deals with AAL5 packets. An AAL5 packet consists of up to 64 k ATM cells. If the data to be transferred exceed this size, it must be split into several packets. This is not implemented in the models yet, so an ALL5 packet size is of unlimited. Models do not consider the segmentation and reassemble phase. Nor is processor I/O modelled.

When sending information to particular destination, it is assumed that the connection is already opened and it will remain open until the transmission is done. It is also assumed that the fabric can handle all possible connections.

For simplified models, ideal cases were assumed. Indefinite size input and output buffers were used and no congestion were modelled. If two packets compete for the same output, available bandwidth for each will be halved.

The detailed model, i.e. the AT&T ATM switch fabric is based on Phoenix switching element. Each switching element is 2x2. They are assembled in an Omega network with self routing algorithm. Back press scheme is used when the traffic is too fast. Each switching element can store up to 9 ATM cells. The whole fabric is assumed to have infinite input buffer and 4000 cells output buffer.

The speed and delay of the four models are listed in Table 16.

Table 16: Parameters for ATM switch models

Switch type	Link bandwidth	Setup time	Transfer delay
Alcatel155	155Mbit/s with 80% max utilization	1 us	80.0 us
Alcatel622	622 Mbit/s with 80% max utilization	1 us	80.0 us

Table 16: Parameters for ATM switch models

Switch type	Link bandwidth	Setup time	Transfer delay
Fore	155 Mbit/s	1 us	10.0 us
AT&T	622 Mbit/s	650 ns input setup 650 ns output setup	650 ns per switch element

For more detail, please refer to [5].

Integrating ATM to ALSIM simulation environment

The general ALSIM event builder is performed by a generic switch between LDCs and GDCs. The objective of ALICE DAQ simulations with ATM is to replace this generic switches with ATM switching systems, as shown in Figure 38. The possible ATM switches have already been discussed in the last section.

a)

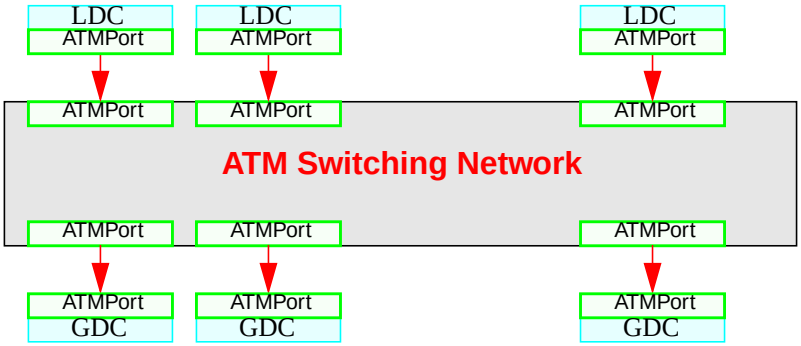


FIGURE 38. ATM-based switching systems

How to use/configure the system

Changes in configuration file

In the atlas.conf, following changes must be made to adapt with ATM, the left column is the old file while the right should be in the new one.

Original alice.conf

New Alice.conf

```
#define ATMAlcatel155 1
#define ATMAlcatel622 2
#define ATMFore 3
#define ATMATT 4

#define ATMPort 13003
#define BasicATMSwitch 20010
#define AttATMSwitch    20011
```

#define Switch DAQSwitch	#define Switch BasicATMSwitch /* IF Simple switch */ #define Switch AttATMSwitch /* IF AT&T Switch */
#define LDCToSPort BasicDAQPort #define SToLDCPort BasicDAQPort #define SToGDCPort BasicDAQPort #define GDCToSPort BasicDAQPort	#define LDCToSPort ATMPort #define SToLDCPort ATMPort #define SToGDCPort ATMPort #define GDCToSPort ATMPort
object Switch * 2 2	object Switch * 2 2 ATMAlcatel155 /* IF Simple switch */ /* object Switch * 2 2 ATMAlcatel622 IF Simple switch */ /* object Switch * 2 2 ATMFore IF Simple switch */ object Switch * 2 2 ATMATT /* IF AT&T Switch */

Simulation results and analysis

Results come in a comprehensive text format. For simplified models, the following is calculated:

- mean load for each input;
- number of packets transmitted by each input;
- number of cells transmitted by each input;
- mean load on each output;
- number of cells received by each output;
- number of cells received by each output;
- normalized histogram of output bandwidth utilization for each output.

For detail model, results show,

- mean load for each input;
- number of packets transmitted by each input;
- number of cells transmitted by each input;
- normalized histogram of output bandwidth utilization for each output.
- histogram of output buffer occupancy (optional).

For example, the simulation result is,

```
Input Traffic:
 INPort # 1 load: 0.286071 Cell Sent: 1135 Packet sent: 47
 INPort # 2 load: 0.277455 Cell Sent: 1074 Packet sent: 47

Output Traffic:
 OUTPort # 1 load: 0.323495 Cell received: 1229 Packet Received: 46
 OUTPort # 2 load: 0.240031 Cell received: 980 Packet Received: 48
```

The first line after “Input Traffic” tells you that link #1 was used at 28.6071 % of the available bandwidth (average load from statistics reset time to dump time). It sent 47 AAL5 packets, which amount to a total of 1135 ATM cells.
It is the same for rest of the lines in “Input Traffic” and “Output Traffic”.

```
Bandwidth utilization histograms:
 Output # 1 : [0%,5%]= 0.000000
 Output # 1 : [5%,10%]= 0.000000
 Output # 1 : [10%,15%]= 0.000000
 Output # 1 : [15%,20%]= 0.000000
 Output # 1 : [20%,25%]= 0.000000
 Output # 1 : [25%,30%]= 0.000000
 Output # 1 : [30%,35%]= 0.000000
 Output # 1 : [35%,40%]= 0.000000
 Output # 1 : [40%,45%]= 2.173913
 Output # 1 : [45%,50%]= 34.782609
 Output # 1 : [50%,55%]= 15.217391
 Output # 1 : [55%,60%]= 4.347826
 Output # 1 : [60%,65%]= 8.695652
 Output # 1 : [65%,70%]= 13.043478
 Output # 1 : [70%,75%]= 4.347826
 Output # 1 : [75%,80%]= 4.347826
 Output # 1 : [80%,85%]= 0.000000
 Output # 1 : [85%,90%]= 4.347826
 Output # 1 : [90%,95%]= 8.695652
 Output # 1 : [95%,100%]= 0.000000
```

Bandwidth utilization histograms:
When an input send a number of cells, the minimum time to transfer this packet can be computed: number_of_cells * cell_transfer_delay. The actual time required to transfer this packet is longer due to queuing in various places. The ratio of the two previous numbers tells you the fraction of the BW that each packet used to reach its output. This is plotted on the histogram. If all bins are empty except the last one, it means that all packets went through the fabric at the maximum possible speed: they used between 95% to 100% of the link BW. If all bins are empty except the first ones, this means that packets could not use much of the BW due to contentions. A system that work nicely should have the last bin much more populated than the first ones.

Histogram TBD.

Chapter 12. Using FC Technology in Switching Network

Background

Fibre Channel (FC) provides an interconnect mechanism to network heterogeneous systems such as peripheral devices and computing devices based on physical media such as copper or fibre, at speeds of 133 Mbit/s, 266 Mbit/s, 531 Mbit/s or 1062 Mbit/s. The interconnection can be based on bus, ring or switch topologies, with a possibility of connecting up to 16 million nodes.

FC supports different classes of services. For class1 service, which is connection-oriented, the existing generic ALSIM model is good enough to be adapted, with some justification of parameters. This is the work that is being done right now.

On the long run, class 2 service maybe needed, due to the long setup time of Class 1 service is not suitable for short packets, such as the trigger signal. FCSIM could be used for this purpose. FCSIM is a simulator developed at the University of Oslo. It is a stand alone FC simulator that supports both Class 1 and Class 2 service. It can be easily configurable with parameters, and is object-oriented for future expansion.

Simulation of ALICE DAQ system with Class 1 FC network

Some work is being done in this area.

TBD

FCSIM

For more detail of FCSIM, please refer to “FCSIM - A Fibre Channel Simulator in MODSIM II” [8].

Chapter 13. Implementation Examples

Technology dependent DAQ implementations, some thoughts

A possible implementation of the proposed general architecture is to use the switch network also for distribution of synchronization information between GDC and LDC.

This implementation is based on that most of switching networks are bidirectional for fault-tolerant purpose. If we call the path from LDC to GDC forward-path, which is used for passing events, the backward-path from GDC to LDC is then used for confirmation of arrival of events, which ensures the system to be error-free. Malfunction of switches in the switch network could also be detected and the faulty switch could be wound around.

An example of such a switching network is the SCI based network, of which the return path is a must in any topology. The traffic in the backward-path normally does not consume all the bandwidth of the paths. Therefore, there is capacity left for synchronization traffic, which is distributed by EDM as before.

The bus architecture between processors and data recording devices may be a potential bottleneck. As an alternative, the bus could be replaced by point-to-point links. If the data is supposed to be distributed onto different disks, tape, the EDM should take care of the situation. Another alternative would be using a switch between GDCs and disks.

Data recording implementation

If we consider that we will use a data recording device with the following parameters, bandwidth is 20 Mbyte/s, capacity is 50 Gbytes, rewinding and load/unload time is 50 seconds. [1]

We also assume almost no data reduction is done in any sense, thus we will have to store in one month

30 days x 24 hours x 60 minutes x 60 seconds x 1 Gbyte/second data rate.

This could mean 50 000 (30x24x60x60x1/50) tapes in total!

Let’s look in the other way, say how many such tape drivers we need.

With 1 Gbyte/s data rate, we need fifty such 20 Mbyte/s drivers. For each tape, it takes 50 Gbyte/20Mbyte/s= 2500 second. We must also provide one extra tape drivers for shifting when any tapes are used up.

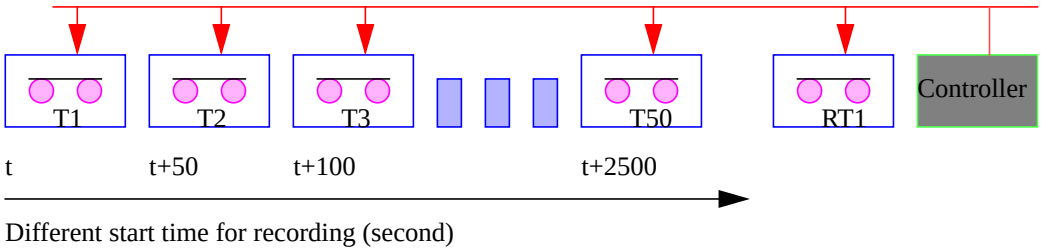


FIGURE 39. An implementation of data recording system

How it works? We can see from Figure 39 that each of the tape driver start at well defined time. The interval

1. These parameters are chosen based on the data sheet of current available Middle-Range devices, but better in a factor of two.

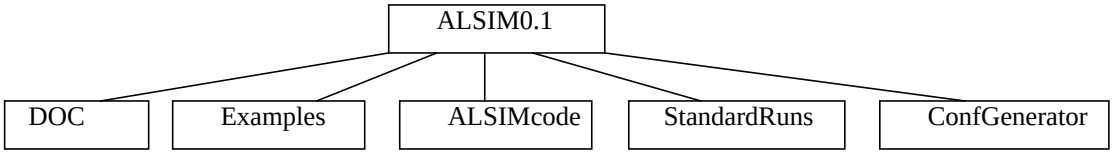
between the first and the next is about 50 seconds. Once the T1 tape is full, it is swapped with the reserved tape (RT1). And RT1 starts to fill up again, but that will be after 2500 seconds. In the meanwhile, T1 is doing rewinding and load/unload work. After 50 seconds, it will be free again. At this time point, T2 should also be full and require swapping. This process can go until we run out of tapes. In real world, a few more reserved tape drivers should be available in case the hardware of a driver fails. A controller is needed to maintain (mostly keep track of) the drivers.

Appendix A. Source File and Compilation

If your platform is not a SUN SPARC station, you will have to use “anonymous” transfer (ftp) a UNIX tar file from our server fidibus.uio.no (see ftp service), and uncompress/untar it. Please follow the steps below

```
% ftp fidibus.uio.no
Connected to fidibus.uio.no.
220 fidibus FTP server (ULTRIX Version 4.1 Mon Jun 12
00:38:17 EST 1991) ready.
Name (fidibus.uio.no:binw): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> cd pub
250 CWD command successful.
ftp> cd ^AALICE/ALSIM
250 CWD command successful.
ftp> bi
200 Type set to I.
ftp> get ALSIM0.1.tar.gz
200 PORT command successful.
150 Opening data connection for ALSIM0.1.tar.gz
(129.240.22.239,2906) (2229705 bytes).
226 Transfer complete.
local: ALSIM0.1.tar.gz remote: ALSIM0.1.tar.gz
2229705 bytes received in 8 seconds (2.7e+02 Kbytes/s)
ftp> quit
221 Goodbye.
% gunzip ALSIM0.1.tar.gz
% tar xf ALSIM0.1.tar
% ls -l ALSIM0.1
total 6
drwxr-x---  2 binw         1536 Aug 31 17:30 ALSIMcode
drwxr-x---  2 binw          512 Aug 31 14:19 ConfGenerator
drwxr-x---  2 binw          512 Aug 31 14:19 DOC
drwxr-x---  2 binw          512 Aug 31 17:34 Examples
drwxr-x---  2 binw          512 Aug 31 17:34 StandardRuns
```

The correct setup of ALSIM directory will be as the tree structure below,



The correct setup of ALSIM should consist of all the files listed below where ALSIM is the executable code

for SUN 4.

```
% ls ALSIM0.1/ALSIMcode
```

ALSIM	ALSIM.lib	DAnimation.mod
DBasicDAQObject.mod	DBasicDAQPort.mod	DDAQSwitch.mod
DDAQSystem.mod	DEDM.mod	DFEC.mod
DGDC.mod	DGDCStoreProcess.mod	DHardwareObject.mod
DLDC.mod	DLDCStoreProcess.mod	DMessage.mod
DEventGen.mod	DOSObject.mod	DPDS.mod
DProcessObject.mod	History.txt	IAnimation.mod
IBasicDAQObject.mod	IBasicDAQPort.mod	IDAQSwitch.mod
IDAQSystem.mod	IEDM.mod	IFEC.mod
IGDC.mod	IGDCStoreProcess.mod	IHardwareObject.mod
ILDC.mod	ILDCStoreProcess.mod	IMessage.mod
IEventGen.mod	IOSObject.mod	DPDS.mod
IProcessObject.mod	MALSIM.mod	README
historc	SCILAB	ATM
HISTOGRAM	results	compileALSIM
DMUX.mod	IMUX.mod	DZSC.mod
IZSC.mod		

```
% ls ALSIM0.1/ALSIMcode/ATM
```

DATMPort.mod	DAttATMFabric.mod	DAttElement.mod
DBasicATMSwitch.mod	IATMPort.mod	IAttATMFabric.mod
IAttElement.mod	IBasicATMSwitch.mod	MTest.mod

```
% ls ALSIM0.1/ALSIMcode/SCILAB
```

DSCIDAQSystem.mod	DSCIPort.mod	DSCInodes.mod
DSCItrans.mod	ISCIDAQSystem.mod	ISCIPort.mod
ISCInodes.mod	ISCItrans.mod	SCItrans.c
MTest.mod		

```
% ls ALSIM0.1/ALSIMcode/HISTOGRAM
```

DAQPipe.c	DAQPipe.h	DDAQAssert.mod
DDAQHisto.mod	DDAQPipe.mod	DDAQSetupFile.mod
IDAQAssert.mod	IDAQHisto.mod	IDAQPipe.mod
DDAQSetupFile.mod	MTest.mod	

The way to compile them is:

```
% compileALSIM
```

If you did all above procedures correct, there should be no compile errors. Otherwise, please contact the authors. After compiling, the executable file ALSIM should be there.

Appendix B. Configuration File Syntax

This appendix presents the complete syntax of the ALSIM configuration file in BNF format.

Conventions

The following conventions are used in describing this syntax.

- Reserved words are written in boldface.
- The vertical bar symbol (|) separates alternative items.
- Square brace ([...]) denote optional items.
- Curly braces ({...}) identify items that belong together.
- Curly braces with plus ({...}+) identify an item that is repeated zero or more times.
- Curly braces with multiply ({...}*) identify an item that is repeated one or more times.
- The starting nonterminal name is < configuration file >.

< configuration file >	::=	< MacroDefinition> < SystemParameters > **begin** < ParentObjects > [< includeTechFile >] **end**

PARENT OBJECT

< ParentObjects >	::=	< BasicParentObjects > | < TechnologyParentsObjects >
< BasicParentObjects >	::=	< DGObject > < EDMObject > { < FECObject > }+ { < LDCObject > }+ < SWITCHObject > { < GDCObject > }+ { < PDSObject > }+
< TechnologyParentsObjects >	::=	< DGObject > < EDMTechObject > { < ExtendedFECObjects > }+ { < LDCTechObject > }+ < SWITCHTechObject > { < GDCTechObject > }+ { < PDSObject > }+

< DGObject > ::= **object 11000** < ObjectNumber >
[**delay** < Integer >]
[**TPCmessage** < MessageSize > < MessageInterval > [< Size-Distribution >] [< IntervallDistribution >] [< sigma >]
[< range >]]
[**DIMUONmessage** < MessageSize > < MessageInterval >
[< SizeDistribution >] [< IntervallDistribution >] [< sigma >]
[< range >]]
< OutPort >
end object

< EDMObject > ::= **object 11001** < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }
{ < InPort > } +
{< OutPort > }+
< EDMLogicalPort >
end object

< EDMTechObject > ::= **object 11001** < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }
{< BiPort > }
< EDMLogicalPort >
end object

< ExtendedFECObjects > { FECObject }+ | { MUXObject ZSCObject }+

< FECObject > ::= **object 11002** < ObjectNumber >
[**delay** < Integer >]
[**bufferSize** < Integer >]
{ < InPort > }
{ < OutPort > }
end object

< MUXObject > ::= **object 11002** < ObjectNumber >
[**delay** < Integer >]
{ < OutPort > }
end object

< ZSCObject > ::= **object 11002** < ObjectNumber >
[**delay** < Integer >]
[**bufferSize** < Integer >]
[**zeroSuppressionRate** < Integer >]
{ < OutPort > }
end object

< LDCObject > ::= **object 11003** < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }
{ < OutPort > }
{< InPort > }
< LDCLogicalPort >
< LDCOSObject >
end object

< LDCTechObject > ::= **object 11003** < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }
{ < BiPort > }
< LDCLogicalPort >
< LDCOSObject >
end object

< SWITCHObject > ::= **object** < SwitchId > < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }+
{ < OutPort > }+
end object

< SWITCHTechObject > ::= **object** < SwitchId > < ObjectNumber >
[**delay** < Integer >]
{ < BiPort > }+
end object

< GDCObject > ::= **object 11004** < ObjectNumber >
[**delay** < Integer >]
{ < InPort > }
{ < OutPort > }
{ < OutPort > }
< GDCLogicalPort >
< GDCOSObject >
end object

< GDCTechObject > ::= **object 11004** < ObjectNumber >
[**delay** < Integer >]
{ < BiPort > }
{ OutPort }
< GDCLogicalPort >
< GDCOSObject >
end object

< PDSObject > ::= **object 11005** < ObjectNumber >
[**delay** < Integer >]
< InPort >
end object

CHILD OBJECT

< OutPort > ::= **object** < PortObjectId > < PortObjectNumber > < Connecting-PortObjectNumber > < ConnectingPortObjectId > 1 < Integer >
[**delay** < Integer >]
[**clock** < real >]
end object

< InPort > ::= **object** < PortObjectId > < PortObjectNumber > < Connecting-PortObjectNumber > < ConnectingPortObjectId > 2 < Integer >
[**delay** < Integer >]
[**clock** < real >]
end object

< BiPort > ::= **object** < PortObjectId > < PortObjectNumber > < Connecting-PortObjectNumber > < ConnectingPortObjectId > 3 < Integer >
[**delay** < Integer >]
[**clock** < real >]
end object

< LDCOSObject > ::= **object 12001** < ObjectNumber >
[**delay** < Integer >]
[**clock** < real >]
< LDCStoreProcessObject >
end object

< GDCOSObject > ::= **object 12001** < ObjectNumber >
[**delay** < Integer >]
[**clock** < real >]
< GDCStoreProcessObject >
end object

< LDCStoreProcessObject > ::= **object 12005** < ObjectNumber >
[**processTime** < interger >]
[**bufferSize** < Integer >]
end object

< GDCStoreProcessObject > ::= **object 12006** < ObjectNumber >
[**processTime** < interger >]
[**bufferSize** < Integer >]
end object

LOGICAL DEFINITIONS

< EDMLogicalPort > ::= **logicalport decisionPortIn** < PortObjectNumber > < PortObject-Id >
logicalport decisionPortOut < PortObjecNumber > < PortOb-jectId>

< LDCLogicalPort >	::=	**logicalport dataPortIn** < PortObjectNumber > < PortObjectId > **logicalport dataPortOut** < PortObjectNumber > <PortObjectId> **logicalport decisionPortIn** < PortObjectNumber > < PortObject-Id >
< GDCLogicalPort >	::=	**logicalport dataPortIn** < PortObjectNumber > < PortObjectId > **logicalport dataPortOut** < PortObjectNumber > <PortObjectId> **logicalport decisionPortOut** < PortObjectNumber > < PortObjectId >

PARAMETERS

< SystemParameters >	::=	[**graphics** < Integer >] [< Histogram >] [**debuglevel** < Integer >] [**edm2switch**] [**extendedFEC**] [**dimuon** < Integer > < Integer >] [**token**] { **simtime** < Integer > }
< Histogram >	::=	[**histogramLatency**] [**histogramRecord**] [**histogramOccupancy**] [**histogramInterval** < Integer >]
< SwitchId >	::=	**10004** | **20004** | **20010** | **20011**
< PortObjectId >	::=	**13001** | **13002** | **13003**
< ConnectingPortObjectId >	::=	**13001** | **13002** | **13003**
< ObjectNumber >	::=	< Integer > | ***** (This is the token '*', not for repeatation)
< PortObjectNumber >	::=	< Integer >
< ConnectingPortObjectNumber >	::=	< Integer >
< MessageSize >	::=	< Integer >
< MessageInterval >	::=	< Integer >
< SizeDistribution >	::=	< Integer >
< IntervallDistribution >	::=	< Integer >
< sigma >	::=	< real >
< range >	::=	< real >
< Integer >	::=	< digit > { < digit > }*
< real >	::=	real number

< digit >	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< Text >	::=	{ all characters}

MACROS

< includeTechFile >	::=	**#include** "< Text > "
< MacroDefinition >	::=	MacroDefinition is the same the defined in the C Programming Language. Same for the comments. The #include is listed specifically dues to the fact that different technology configuration files are usually a separate file than the generic one.